This manual describes the operators of the image analysis system HALCON, version 6.1.4, in COM syntax. It was generated on February 1, 2005.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without prior written permission of the publisher.

Copyright © 1997-2005 by MVTec Software GmbH, München, Germany

More information about HALCON can be found at: http://www.mvtec.com
Contents

1 Classification ................................. 1
   ClearSampset ........................................ 1
   CloseAllClassBox .................................. 1
   CloseClassBox ..................................... 2
   CreateClassBox .................................... 2
   DescriptClassBox ................................ 2
   EnquireClassBox ................................... 3
   EnquireRejectClassBox ............................ 4
   GetClassBoxParam .................................. 4
   LearnClassBox ..................................... 5
   LearnSampsetBox .................................. 6
   ReadClassBox ...................................... 7
   ReadSampset ...................................... 7
   SetClassBoxParam .................................. 8
   TestSampsetBox .................................... 9
   WriteClassBox .................................... 10

2 File ............................................. 11
   2.1 Images .......................................... 11
      ReadImage ........................................ 11
      ReadSequence .................................... 12
      WriteImage ....................................... 14
   2.2 Misc ............................................ 15
      FileExists ........................................ 15
      ReadWorldFile .................................... 16
   2.3 Region .......................................... 17
      ReadRegion ....................................... 17
      WriteRegion ...................................... 17
   2.4 Text ............................................ 18
      CloseAllFiles .................................... 18
      CloseFile ......................................... 18
      FnewLine .......................................... 19
      FreadChar ......................................... 20
      FreadString ....................................... 20
      FwriteString ...................................... 21
      OpenFile .......................................... 22
   2.5 Tuple ............................................ 23
      ReadTuple ......................................... 23
      WriteTuple ....................................... 23
   2.6 XLD ............................................. 24
      ReadContourXldArcInfo ............................ 24
      ReadPolygonXldArcInfo ............................ 24
      WriteContourXldArcInfo ........................... 25
      WritePolygonXldArcInfo ........................... 26

3 Filter ........................................... 29
## 3.1 Affine-Transformations
- AffineTransImage ........................................... 29
- AffineTransImageSize .................................... 31
- MirrorImage .................................................. 32
- PolarTransImage .......................................... 33
- RotateImage .................................................. 34
- ZoomImageFactor .......................................... 35
- ZoomImageSize .............................................. 36

## 3.2 Arithmetic
- AbsImage ..................................................... 37
- AddImage ....................................................... 38
- DivImage ....................................................... 39
- InvertImage .................................................. 40
- MaxImage ...................................................... 41
- MinImage ....................................................... 42
- MultiImage .................................................... 43
- ScaleImage .................................................... 44
- SubImage ....................................................... 45

## 3.3 Bit
- BitAnd .......................................................... 46
- BitLshift ....................................................... 46
- BitMask ......................................................... 47
- BitNot .......................................................... 48
- BitOr ........................................................... 49
- BitRshift ....................................................... 49
- BitSlice ........................................................ 50
- BitXor .......................................................... 51
- Bit .............................................................. 52

## 3.4 Color
- CfaToRgb ...................................................... 53
- Rgb1ToGray .................................................... 53
- Rgb3ToGray .................................................... 54
- TransFromRgb ................................................. 55
- TransToRgb .................................................... 58

## 3.5 Edges
- CloseEdges .................................................... 61
- CloseEdgesLength ........................................... 62
- DerivateGauss ............................................... 63
- DiffOfGauss ................................................... 66
- EdgesImage .................................................... 67
- EdgesSubPix ................................................... 70
- FreiAmp ........................................................ 73
- FreiDir ........................................................ 73
- HighpassImage ............................................... 75
- InfoEdges ...................................................... 76
- KirschAmp ...................................................... 77
- KirschDir ....................................................... 78
- Laplace ........................................................ 79
- LaplaceOfGauss ............................................. 81
- PrewittAmp .................................................... 81
- PrewittDir ...................................................... 82
- Roberts ........................................................ 83
- RobinsonAmp .................................................. 84
- RobinsonDir .................................................... 85
- SobelAmp ....................................................... 87
- SobelDir ......................................................... 88

## 3.6 Enhancement
- Emphasize ..................................................... 90
- EquHistogram ............................................... 91
### 3.7 FFT

- ConvolFit
- ConvolGabor
- EnergyGabor
- FitGeneric
- FitImage
- FitImageInv
- GenBandfilter
- GenBandpass
- GenFilterMask
- GenGabor
- GenHighpass
- GenLowpass
- GenSinBandpass
- GenStdBandpass
- PhaseDeg
- PhaseRad
- PowerByte
- PowerLn
- PowerReal

### 3.8 Lines

- BandpassImage
- LinesFacet
- LinesGauss

### 3.9 Match

- CornerResponse
- ExhaustiveMatch
- ExhaustiveMatchMg
- FillDvf
- GenGaussPyramid
- Monotony
- OpticalFlowMatch

### 3.10 Misc

- ConvolImage
- ExpandDomainGray
- GrayInside
- GraySkeleton
- LutTrans
- MapImage
- PrincipalComp
- Symmetry
- TopographicSketch

### 3.11 Noise

- AddNoiseDistribution
- AddNoiseWhite
- GaussDistribution
- NoiseDistributionMean
- SpDistribution

### 3.12 Smoothing

- AnisotropeDiff
- EliminateMinMax
- EliminateSp
- FillInterlace
- GaussImage
- InfoSmooth
- MeanImage
### 3.1 Wiener-Filter
- GenPsfDefocus
- GenPsfMotion
- SimulateDefocus
- SimulateMotion
- WienerFilter
- WienerFilterNi

### 3.14 Texture
- DeviationImage
- EntropyImage
- TextureLaws

### 4 Graphics

#### 4.1 Drawing
- DragRegion1
- DragRegion2
- DragRegion3
- DrawCircle
- DrawCircleMod
- DrawEllipse
- DrawEllipseMod
- DrawLine
- DrawLineMod
- DrawPoint
- DrawPointMod
- DrawPolygon
- DrawRectangle1
- DrawRectangle1Mod
- DrawRectangle2
- DrawRectangle2Mod
- DrawRegion

#### 4.2 Gnuplot
- GnuplotClose
- GnuplotOpenFile
- GnuplotOpenPipe
- GnuplotPlotCtrl
- GnuplotPlotFunct1D
- GnuplotPlotImage

#### 4.3 LUT
- DispLut
- DrawLut
- GetFixedLut
- GetLut
- GetLutStyle
- QueryLut
- SetFixedLut
- SetLut
- SetLutStyle
<table>
<thead>
<tr>
<th>4.6 Parameters</th>
<th>218</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetComprise</td>
<td>218</td>
</tr>
<tr>
<td>GetDraw</td>
<td>219</td>
</tr>
<tr>
<td>GetFix</td>
<td>219</td>
</tr>
<tr>
<td>GetHsi</td>
<td>220</td>
</tr>
<tr>
<td>GetIcon</td>
<td>221</td>
</tr>
<tr>
<td>GetInsert</td>
<td>222</td>
</tr>
<tr>
<td>GetLineApprox</td>
<td>222</td>
</tr>
<tr>
<td>GetLineStyle</td>
<td>222</td>
</tr>
<tr>
<td>GetLineWidth</td>
<td>223</td>
</tr>
<tr>
<td>GetPaint</td>
<td>223</td>
</tr>
<tr>
<td>GetPart</td>
<td>224</td>
</tr>
<tr>
<td>GetPartStyle</td>
<td>224</td>
</tr>
<tr>
<td>GetPixel</td>
<td>225</td>
</tr>
<tr>
<td>GetRgb</td>
<td>226</td>
</tr>
<tr>
<td>GetShape</td>
<td>226</td>
</tr>
<tr>
<td>QueryAllColors</td>
<td>227</td>
</tr>
<tr>
<td>QueryColor</td>
<td>228</td>
</tr>
<tr>
<td>QueryColored</td>
<td>228</td>
</tr>
<tr>
<td>QueryGray</td>
<td>229</td>
</tr>
<tr>
<td>QueryInsert</td>
<td>230</td>
</tr>
<tr>
<td>QueryLineWidth</td>
<td>230</td>
</tr>
<tr>
<td>QueryPaint</td>
<td>230</td>
</tr>
<tr>
<td>QueryShape</td>
<td>231</td>
</tr>
<tr>
<td>SetColor</td>
<td>232</td>
</tr>
<tr>
<td>SetColored</td>
<td>233</td>
</tr>
<tr>
<td>SetComprise</td>
<td>233</td>
</tr>
<tr>
<td>SetDraw</td>
<td>234</td>
</tr>
<tr>
<td>SetFix</td>
<td>235</td>
</tr>
<tr>
<td>SetGray</td>
<td>236</td>
</tr>
<tr>
<td>SetHsi</td>
<td>237</td>
</tr>
<tr>
<td>SetIcon</td>
<td>238</td>
</tr>
<tr>
<td>SetInsert</td>
<td>238</td>
</tr>
<tr>
<td>SetLineApprox</td>
<td>239</td>
</tr>
<tr>
<td>SetLineWidth</td>
<td>240</td>
</tr>
</tbody>
</table>

4.5 Output

<table>
<thead>
<tr>
<th>DispArc</th>
<th>203</th>
</tr>
</thead>
<tbody>
<tr>
<td>DispArrow</td>
<td>204</td>
</tr>
<tr>
<td>DispChannel</td>
<td>205</td>
</tr>
<tr>
<td>DispCircle</td>
<td>206</td>
</tr>
<tr>
<td>DispColor</td>
<td>208</td>
</tr>
<tr>
<td>DispDistribution</td>
<td>208</td>
</tr>
<tr>
<td>DispEllipse</td>
<td>209</td>
</tr>
<tr>
<td>DispImage</td>
<td>211</td>
</tr>
<tr>
<td>DispLine</td>
<td>212</td>
</tr>
<tr>
<td>DispObj</td>
<td>213</td>
</tr>
<tr>
<td>DispPolygon</td>
<td>214</td>
</tr>
<tr>
<td>DispRectangle1</td>
<td>214</td>
</tr>
<tr>
<td>DispRectangle2</td>
<td>216</td>
</tr>
<tr>
<td>DispRegion</td>
<td>217</td>
</tr>
<tr>
<td>DispXId</td>
<td>218</td>
</tr>
</tbody>
</table>

4.4 Mouse

<table>
<thead>
<tr>
<th>GetMbutton</th>
<th>199</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetMposition</td>
<td>200</td>
</tr>
<tr>
<td>GetMshape</td>
<td>201</td>
</tr>
<tr>
<td>QueryMshape</td>
<td>202</td>
</tr>
<tr>
<td>SetMshape</td>
<td>202</td>
</tr>
</tbody>
</table>

4.3 Parameters

<table>
<thead>
<tr>
<th>WriteLut</th>
<th>199</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------</td>
</tr>
<tr>
<td>SetLineWidth</td>
<td>241</td>
</tr>
<tr>
<td>SetPaint</td>
<td>241</td>
</tr>
<tr>
<td>SetPart</td>
<td>245</td>
</tr>
<tr>
<td>SetPartStyle</td>
<td>246</td>
</tr>
<tr>
<td>SetPixel</td>
<td>247</td>
</tr>
<tr>
<td>SetRgb</td>
<td>248</td>
</tr>
<tr>
<td>SetShape</td>
<td>249</td>
</tr>
<tr>
<td>4.7 Text</td>
<td></td>
</tr>
<tr>
<td>GetFont</td>
<td>250</td>
</tr>
<tr>
<td>GetStringExtents</td>
<td>251</td>
</tr>
<tr>
<td>GetTposition</td>
<td>251</td>
</tr>
<tr>
<td>GetTshape</td>
<td>252</td>
</tr>
<tr>
<td>.NewLine</td>
<td>253</td>
</tr>
<tr>
<td>QueryFont</td>
<td>253</td>
</tr>
<tr>
<td>QueryTshape</td>
<td>254</td>
</tr>
<tr>
<td>ReadChar</td>
<td>254</td>
</tr>
<tr>
<td>ReadString</td>
<td>255</td>
</tr>
<tr>
<td>SetFont</td>
<td>256</td>
</tr>
<tr>
<td>SetTposition</td>
<td>257</td>
</tr>
<tr>
<td>SetTshape</td>
<td>258</td>
</tr>
<tr>
<td>WriteString</td>
<td>258</td>
</tr>
<tr>
<td>4.8 Window</td>
<td></td>
</tr>
<tr>
<td>ClearRectangle</td>
<td>259</td>
</tr>
<tr>
<td>ClearWindow</td>
<td>260</td>
</tr>
<tr>
<td>CloseWindow</td>
<td>261</td>
</tr>
<tr>
<td>CopyRectangle</td>
<td>261</td>
</tr>
<tr>
<td>DumpWindow</td>
<td>263</td>
</tr>
<tr>
<td>GetOsWindowHandle</td>
<td>265</td>
</tr>
<tr>
<td>GetWindowAttr</td>
<td>265</td>
</tr>
<tr>
<td>GetWindowExtents</td>
<td>266</td>
</tr>
<tr>
<td>GetWindowPointer3</td>
<td>267</td>
</tr>
<tr>
<td>GetWindowType</td>
<td>267</td>
</tr>
<tr>
<td>MoveRectangle</td>
<td>268</td>
</tr>
<tr>
<td>NewExternWindow</td>
<td>268</td>
</tr>
<tr>
<td>OpenTextWindow</td>
<td>269</td>
</tr>
<tr>
<td>OpenWindow</td>
<td>272</td>
</tr>
<tr>
<td>QueryWindowType</td>
<td>275</td>
</tr>
<tr>
<td>SetWindowAttr</td>
<td>275</td>
</tr>
<tr>
<td>SetWindowDc</td>
<td>276</td>
</tr>
<tr>
<td>SetWindowExtents</td>
<td>279</td>
</tr>
<tr>
<td>SetWindowType</td>
<td>280</td>
</tr>
<tr>
<td>SlideImage</td>
<td>281</td>
</tr>
<tr>
<td>5 Image</td>
<td></td>
</tr>
<tr>
<td>5.1 Access</td>
<td>285</td>
</tr>
<tr>
<td>GetGrayval</td>
<td>285</td>
</tr>
<tr>
<td>GetImagePointer1</td>
<td>286</td>
</tr>
<tr>
<td>GetImagePointer1Rect</td>
<td>287</td>
</tr>
<tr>
<td>GetImagePointer3</td>
<td>288</td>
</tr>
<tr>
<td>GetImageTime</td>
<td>289</td>
</tr>
<tr>
<td>5.2 Channel</td>
<td></td>
</tr>
<tr>
<td>AccessChannel</td>
<td>290</td>
</tr>
<tr>
<td>AppendChannel</td>
<td>290</td>
</tr>
<tr>
<td>ChannelsToImage</td>
<td>291</td>
</tr>
<tr>
<td>Compose2</td>
<td>291</td>
</tr>
<tr>
<td>Compose3</td>
<td>292</td>
</tr>
<tr>
<td>Compose4</td>
<td>293</td>
</tr>
<tr>
<td>Compose5</td>
<td>293</td>
</tr>
<tr>
<td>5.3 Creation</td>
<td>302</td>
</tr>
<tr>
<td>--------------</td>
<td>-----</td>
</tr>
<tr>
<td>CopyImage</td>
<td>302</td>
</tr>
<tr>
<td>GenImage1</td>
<td>302</td>
</tr>
<tr>
<td>GenImage1Extern</td>
<td>303</td>
</tr>
<tr>
<td>GenImage1Rect</td>
<td>305</td>
</tr>
<tr>
<td>GenImage3</td>
<td>306</td>
</tr>
<tr>
<td>GenImageConst</td>
<td>307</td>
</tr>
<tr>
<td>GenImageGrayRamp</td>
<td>308</td>
</tr>
<tr>
<td>GenImageProto</td>
<td>310</td>
</tr>
<tr>
<td>GenImageSurfaceFirstOrder</td>
<td>311</td>
</tr>
<tr>
<td>GenImageSurfaceSecondOrder</td>
<td>312</td>
</tr>
<tr>
<td>RegionToBin</td>
<td>314</td>
</tr>
<tr>
<td>RegionToLabel</td>
<td>315</td>
</tr>
<tr>
<td>RegionToMean</td>
<td>316</td>
</tr>
<tr>
<td>5.4 Domain</td>
<td>317</td>
</tr>
<tr>
<td>AddChannels</td>
<td>317</td>
</tr>
<tr>
<td>ChangeDomain</td>
<td>318</td>
</tr>
<tr>
<td>FullDomain</td>
<td>318</td>
</tr>
<tr>
<td>GetDomain</td>
<td>319</td>
</tr>
<tr>
<td>Rectangle1Domain</td>
<td>319</td>
</tr>
<tr>
<td>ReduceDomain</td>
<td>320</td>
</tr>
<tr>
<td>5.5 Features</td>
<td>321</td>
</tr>
<tr>
<td>AreaCenterGray</td>
<td>321</td>
</tr>
<tr>
<td>CoocFeatureImage</td>
<td>322</td>
</tr>
<tr>
<td>CoocFeatureMatrix</td>
<td>323</td>
</tr>
<tr>
<td>EllipticAxisGray</td>
<td>324</td>
</tr>
<tr>
<td>EntropyGray</td>
<td>325</td>
</tr>
<tr>
<td>FitSurfaceFirstOrder</td>
<td>326</td>
</tr>
<tr>
<td>FitSurfaceSecondOrder</td>
<td>327</td>
</tr>
<tr>
<td>FuzzyEntropy</td>
<td>328</td>
</tr>
<tr>
<td>FuzzyPerimeter</td>
<td>329</td>
</tr>
<tr>
<td>GenCoocMatrix</td>
<td>330</td>
</tr>
<tr>
<td>GrayHisto</td>
<td>332</td>
</tr>
<tr>
<td>GrayHistoAbs</td>
<td>333</td>
</tr>
<tr>
<td>GrayProjections</td>
<td>334</td>
</tr>
<tr>
<td>Histo2Dim</td>
<td>334</td>
</tr>
<tr>
<td>Intensity</td>
<td>335</td>
</tr>
<tr>
<td>MinMaxGray</td>
<td>336</td>
</tr>
<tr>
<td>MomentsGrayPlane</td>
<td>338</td>
</tr>
<tr>
<td>PlaneDeviation</td>
<td>339</td>
</tr>
<tr>
<td>SelectGray</td>
<td>340</td>
</tr>
<tr>
<td>ShapeHistoAll</td>
<td>341</td>
</tr>
<tr>
<td>ShapeHistoPoint</td>
<td>343</td>
</tr>
<tr>
<td>5.6 Format</td>
<td>344</td>
</tr>
<tr>
<td>ChangeFormat</td>
<td>344</td>
</tr>
<tr>
<td>CropDomain</td>
<td>344</td>
</tr>
<tr>
<td>CropDomainRel</td>
<td>345</td>
</tr>
<tr>
<td>CropPart</td>
<td>346</td>
</tr>
</tbody>
</table>
### 7.2 Shape-Based

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClearAllShapeModels</td>
<td>401</td>
</tr>
<tr>
<td>ClearShapeModel</td>
<td>401</td>
</tr>
<tr>
<td>CreateScaledShapeModel</td>
<td>402</td>
</tr>
<tr>
<td>CreateShapeModel</td>
<td>405</td>
</tr>
<tr>
<td>FindScaledShapeModel</td>
<td>408</td>
</tr>
<tr>
<td>FindScaledShapeModels</td>
<td>411</td>
</tr>
<tr>
<td>FindShapeModel</td>
<td>415</td>
</tr>
<tr>
<td>FindShapeModels</td>
<td>418</td>
</tr>
<tr>
<td>GetShapeModelParams</td>
<td>421</td>
</tr>
<tr>
<td>InspectShapeModel</td>
<td>422</td>
</tr>
<tr>
<td>ReadShapeModel</td>
<td>423</td>
</tr>
<tr>
<td>SetShapeModelOrigin</td>
<td>424</td>
</tr>
<tr>
<td>WriteShapeModel</td>
<td>425</td>
</tr>
</tbody>
</table>

### 8 Morphology

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1 Gray-Values</td>
<td>427</td>
</tr>
<tr>
<td>DualRank</td>
<td>427</td>
</tr>
<tr>
<td>GenDiscSe</td>
<td>427</td>
</tr>
<tr>
<td>GrayBothat</td>
<td>429</td>
</tr>
<tr>
<td>GrayClosing</td>
<td>430</td>
</tr>
<tr>
<td>GrayClosingShape</td>
<td>431</td>
</tr>
<tr>
<td>GrayDilation</td>
<td>432</td>
</tr>
<tr>
<td>GrayDilationRect</td>
<td>433</td>
</tr>
<tr>
<td>GrayDilationShape</td>
<td>433</td>
</tr>
<tr>
<td>GrayErosion</td>
<td>434</td>
</tr>
<tr>
<td>GrayErosionRect</td>
<td>435</td>
</tr>
<tr>
<td>GrayErosionShape</td>
<td>436</td>
</tr>
<tr>
<td>GrayOpening</td>
<td>437</td>
</tr>
<tr>
<td>GrayOpeningShape</td>
<td>438</td>
</tr>
<tr>
<td>GrayRangeRect</td>
<td>439</td>
</tr>
<tr>
<td>GrayTophat</td>
<td>439</td>
</tr>
<tr>
<td>ReadGraySe</td>
<td>440</td>
</tr>
<tr>
<td>8.2 Region</td>
<td>441</td>
</tr>
<tr>
<td>BottomHat</td>
<td>441</td>
</tr>
<tr>
<td>Boundary</td>
<td>442</td>
</tr>
<tr>
<td>Closing</td>
<td>443</td>
</tr>
<tr>
<td>ClosingCircle</td>
<td>444</td>
</tr>
<tr>
<td>ClosingGolay</td>
<td>445</td>
</tr>
<tr>
<td>ClosingRectangle1</td>
<td>446</td>
</tr>
<tr>
<td>Dilation1</td>
<td>447</td>
</tr>
<tr>
<td>Dilation2</td>
<td>448</td>
</tr>
<tr>
<td>DilationCircle</td>
<td>450</td>
</tr>
<tr>
<td>DilationGolay</td>
<td>451</td>
</tr>
<tr>
<td>DilationRectangle1</td>
<td>452</td>
</tr>
<tr>
<td>DilationSeq</td>
<td>452</td>
</tr>
<tr>
<td>Erosion1</td>
<td>453</td>
</tr>
<tr>
<td>Erosion2</td>
<td>454</td>
</tr>
<tr>
<td>ErosionCircle</td>
<td>456</td>
</tr>
<tr>
<td>ErosionGolay</td>
<td>457</td>
</tr>
<tr>
<td>ErosionRectangle1</td>
<td>458</td>
</tr>
<tr>
<td>ErosionSeq</td>
<td>459</td>
</tr>
<tr>
<td>Fitting</td>
<td>460</td>
</tr>
<tr>
<td>GenStructElements</td>
<td>461</td>
</tr>
<tr>
<td>GolayElements</td>
<td>462</td>
</tr>
<tr>
<td>HitOrMiss</td>
<td>463</td>
</tr>
<tr>
<td>HitOrMissGolay</td>
<td>464</td>
</tr>
</tbody>
</table>
10 Regions

10.1 Access
GetRegionChain
GetRegionContour
GetRegionConvex
GetRegionPoints
GetRegionPolygon
GetRegionRuns

10.2 Affine-Transformations
AffineTransRegion
MirrorRegion
MoveRegion
TransposeRegion
ZoomRegion

10.3 Creation
GenCheckerRegion
GenCircle
GenEllipse
GenEmptyRegion
GenGridRegion .......................................................... 517
GenRandomRegion ..................................................... 518
GenRandomRegions ..................................................... 519
GenRectangle1 .......................................................... 521
GenRectangle2 .......................................................... 522
GenRegionContourXId .................................................. 524
GenRegionHisto .......................................................... 524
GenRegionHline .......................................................... 525
GenRegionLine .......................................................... 526
GenRegionPoints .......................................................... 527
GenRegionPolygon ...................................................... 528
GenRegionPolygonFilled .............................................. 529
GenRegionPolygonXId ................................................... 530
GenRegionRuns .......................................................... 530
LabelToRegion .......................................................... 531

10.4 Features .......................................................... 532
  AreaCenter .......................................................... 532
  Circularity .......................................................... 533
  Compactness .......................................................... 534
  ConnectAndHoles .................................................... 535
  Contlength .......................................................... 536
  Convexity .......................................................... 536
  DiameterRegion ...................................................... 537
  Eccentricity .......................................................... 538
  EllipticAxis .......................................................... 539
  EulerNumber .......................................................... 540
  FindNeighbors ....................................................... 541
  GetRegionIndex ..................................................... 542
  GetRegionThickness ............................................... 543
  HammingDistance .................................................... 543
  HammingDistanceNorm .............................................. 544
  InnerCircle .......................................................... 545
  MomentsRegion2Nd ................................................... 547
  MomentsRegion2NdInvar ............................................ 548
  MomentsRegion2NdRelInvar ....................................... 549
  MomentsRegion3Rd .................................................. 549
  MomentsRegion3RdInvar ............................................ 550
  MomentsRegionCentral ............................................. 551
  MomentsRegionCentralInvar ..................................... 552
  OrientationRegion .................................................. 553
  Roundness .......................................................... 554
  RunlengthDistribution ............................................ 555
  RunlengthFeatures ................................................ 556
  SelectRegionPoint .................................................. 557
  SelectRegionSpatial ............................................... 558
  SelectShape .......................................................... 560
  SelectShapeProto .................................................... 562
  SelectShapeStd ....................................................... 564
  SmallestCircle ....................................................... 565
  SmallestRectangle1 .................................................. 566
  SmallestRectangle2 .................................................. 567
  SpatialRelation ..................................................... 568
  TestRegionPoint ..................................................... 569

10.5 Sets .......................................................... 570
  Complement .......................................................... 570
  Difference .......................................................... 571
  Intersection .......................................................... 572
  Union1 .............................................................. 572
  Union2 .............................................................. 573
10.6 Transformation

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackgroundSeg</td>
<td>574</td>
</tr>
<tr>
<td>ClipRegion</td>
<td>574</td>
</tr>
<tr>
<td>ClipRegionRel</td>
<td>575</td>
</tr>
<tr>
<td>Connection</td>
<td>576</td>
</tr>
<tr>
<td>DistanceTransform</td>
<td>577</td>
</tr>
<tr>
<td>EliminateRuns</td>
<td>578</td>
</tr>
<tr>
<td>ExpandRegion</td>
<td>578</td>
</tr>
<tr>
<td>FillUp</td>
<td>580</td>
</tr>
<tr>
<td>FillUpShape</td>
<td>580</td>
</tr>
<tr>
<td>HammingChangeRegion</td>
<td>582</td>
</tr>
<tr>
<td>Interlaced</td>
<td>583</td>
</tr>
<tr>
<td>JunctionsSkeleton</td>
<td>584</td>
</tr>
<tr>
<td>MergeRegionsLineScan</td>
<td>586</td>
</tr>
<tr>
<td>PartitionDynamic</td>
<td>587</td>
</tr>
<tr>
<td>PartitionRectangle</td>
<td>588</td>
</tr>
<tr>
<td>RankRegion</td>
<td>589</td>
</tr>
<tr>
<td>RemoveNoiseRegion</td>
<td>591</td>
</tr>
<tr>
<td>ShapeTrans</td>
<td>591</td>
</tr>
<tr>
<td>Skeleton</td>
<td>592</td>
</tr>
<tr>
<td>SortRegion</td>
<td>593</td>
</tr>
<tr>
<td>SplitSkeletonLines</td>
<td>594</td>
</tr>
<tr>
<td>SplitSkeletonRegion</td>
<td>595</td>
</tr>
</tbody>
</table>

11 Segmentation

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoThreshold</td>
<td>597</td>
</tr>
<tr>
<td>BinThreshold</td>
<td>597</td>
</tr>
<tr>
<td>CharThreshold</td>
<td>598</td>
</tr>
<tr>
<td>CheckDifference</td>
<td>598</td>
</tr>
<tr>
<td>Class2DimSup</td>
<td>599</td>
</tr>
<tr>
<td>Class2DimUnsup</td>
<td>601</td>
</tr>
<tr>
<td>ClassNdimBox</td>
<td>602</td>
</tr>
<tr>
<td>ClassNdimNorm</td>
<td>603</td>
</tr>
<tr>
<td>DetectEdgeSegments</td>
<td>604</td>
</tr>
<tr>
<td>DynThreshold</td>
<td>605</td>
</tr>
<tr>
<td>ExpandGray</td>
<td>606</td>
</tr>
<tr>
<td>ExpandGrayRef</td>
<td>608</td>
</tr>
<tr>
<td>ExpandLine</td>
<td>608</td>
</tr>
<tr>
<td>FastThreshold</td>
<td>610</td>
</tr>
<tr>
<td>HistoToThresh</td>
<td>611</td>
</tr>
<tr>
<td>HysteresisThreshold</td>
<td>612</td>
</tr>
<tr>
<td>LearnNdimBox</td>
<td>613</td>
</tr>
<tr>
<td>LearnNdimNorm</td>
<td>614</td>
</tr>
<tr>
<td>LocalMax</td>
<td>614</td>
</tr>
<tr>
<td>NonmaxSuppressionAmp</td>
<td>615</td>
</tr>
<tr>
<td>NonmaxSuppressionDir</td>
<td>616</td>
</tr>
<tr>
<td>Plateaus</td>
<td>619</td>
</tr>
<tr>
<td>PlateausCenter</td>
<td>620</td>
</tr>
<tr>
<td>Pouring</td>
<td>621</td>
</tr>
<tr>
<td>Region Growing</td>
<td>622</td>
</tr>
<tr>
<td>Region Growing Mean</td>
<td>623</td>
</tr>
<tr>
<td>Region Growing N</td>
<td>625</td>
</tr>
<tr>
<td>Threshold</td>
<td>626</td>
</tr>
<tr>
<td>ThresholdSubPix</td>
<td>628</td>
</tr>
<tr>
<td>Watersheds</td>
<td>632</td>
</tr>
<tr>
<td>ZeroCrossing</td>
<td>633</td>
</tr>
<tr>
<td>ZeroCrossingSubPix</td>
<td>634</td>
</tr>
</tbody>
</table>

597
## 12 System

<table>
<thead>
<tr>
<th>Section</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>12.1 Database</strong></td>
<td>SendTuple, SendRegion, SendImage, ReceiveXld, ReceiveTuple, ReceiveRegion, ReceiveImage, OpenSocketConnect, OpenSocketAccept, GetSocketTimeout, GetNextSocketDataType, CloseSocket</td>
</tr>
<tr>
<td><strong>12.2 Error-Handling</strong></td>
<td>GetCheck, GetErrorText, GetSpy, QuerySpy, SetCheck, SetSpy</td>
</tr>
<tr>
<td><strong>12.3 Information</strong></td>
<td>DispInfo, GetChapterInfo, GetKeywords, GetOperatorInfo, GetOperatorName, GetParamInfo, GetParamNames, GetParamNum, GetParamTypes, QueryOperatorInfo, QueryParamInfo, SearchOperator</td>
</tr>
<tr>
<td><strong>12.4 Operating-System</strong></td>
<td>CountSeconds, SystemCall, WaitSeconds</td>
</tr>
<tr>
<td><strong>12.5 Parallelization</strong></td>
<td>CheckParHwPotential, LoadParKnowledge, StoreParKnowledge</td>
</tr>
<tr>
<td><strong>12.6 Parameters</strong></td>
<td>GetSystem, SetSystem</td>
</tr>
<tr>
<td><strong>12.7 Serial</strong></td>
<td>ClearSerial, CloseAllSerials, CloseSerial, GetSerialParam, OpenSerial, ReadSerial, SetSerialParam, WriteSerial</td>
</tr>
<tr>
<td><strong>12.8 Sockets</strong></td>
<td>CloseSocket, GetNextSocketDataType, GetSocketTimeout, OpenSocketAccept, OpenSocketConnect, ReceiveImage, ReceiveRegion, ReceiveTuple, ReceiveXld, SendImage, SendRegion, SendTuple</td>
</tr>
</tbody>
</table>
SendXId .......................................................... 680
SetSocketTimeout .......................................... 681
SocketAcceptConnect ....................................... 681

13 Tools ......................................................... 683

13.1 Affine-Transformations .................................. 683
AffineTransPoint2D ........................................... 683
AffineTransPoint3D ........................................... 684
DvToHomMat2d .................................................. 685
HomMat2dCompose ............................................. 686
HomMat2dIdentity ............................................ 686
HomMat2dInvert .............................................. 687
HomMat2dRotate .............................................. 687
HomMat2dScale .............................................. 689
HomMat2dSlant ............................................... 690
HomMat2dToAffinePar ........................................ 691
HomMat2dTranslate .......................................... 691
HomMat3dCompose ............................................ 693
HomMat3dIdentity .......................................... 694
HomMat3dInvert .............................................. 695
HomMat3dRotate .............................................. 695
HomMat3dScale .............................................. 697
HomMat3dTranslate .......................................... 698
VectorAngleToRigid .......................................... 700
VectorToHomMat2d ........................................... 701
VectorToRigid ................................................ 702
VectorToSimilarity ......................................... 703

13.2 Background-Estimator ................................... 704
CloseAllBgEsti ............................................... 704
CloseBgEsti .................................................. 704
CreateBgEsti .................................................. 705
GetBgEstiParams ............................................. 708
GiveBgEsti .................................................... 709
RunBgEsti ..................................................... 710
SetBgEstiParams ............................................. 712
UpdateBgEsti .................................................. 714

13.3 Barcode .................................................... 715
Decode1DBarCode ............................................. 715
Decode2DBarCode ............................................. 716
Discrete1DBarCode .......................................... 717
Find1DBarCode ............................................... 718
Find1DBarCodeRegion ........................................ 721
Find2DBarCode ............................................... 722
Gen1DBarCodeDescr ......................................... 725
Gen1DBarCodeDescrGen ..................................... 726
Gen2DBarCodeDescr ......................................... 727
Get1DBarCode ................................................ 727
Get2DBarCode ................................................ 730
Get2DBarCodePos ............................................. 734

13.4 Calibration ................................................. 735
CalltabPoints .................................................. 735
CameraCalibration .......................................... 737
ChangeRadialDistortionCamPar ............................. 742
ChangeRadialDistortionContoursXld ...................... 743
ChangeRadialDistortionImage .............................. 743
ContourToWorldPlaneXld .................................... 744
ConvertPoseType ............................................. 745
CreateCalltab ............................................... 746
CreatePose 749
DispCaltab 753
FindCaltab 754
FindMarksAndPose 755
GenImageToWorldPlaneMap 757
GenRadialDistortionMap 760
GetLineOfSight 761
GetPoseType 762
HandEyeCalibration 763
HomMat3dToPose 770
ImagePointsToWorldPlane 771
ImageToWorldPlane 773
PoseToHomMat3d 775
Project3DPoint 776
ReadCamPar 777
ReadPose 778
SetOriginPose 779
SimCaltab 780
WriteCamPar 782
WritePose 784

13.5 Fourier-Descriptor 785
AbsInvarFourierCoeff 785
Fourier1Dim 786
Fourier1DimInv 787
InvarFourierCoeff 787
MatchFourierCoeff 788
MoveContourOrig 789
PrepContourFourier 790

13.6 Function 790
AbsFunct1D 790
CreateFunct1DArray 791
CreateFunct1DPairs 791
DistanceFunct1D 792
Funct1DToPairs 793
GetPairFunct1D 793
IntegrateFunct1D 794
MatchFunct1DTrans 794
NegateFunct1D 795
NumPointsFunct1D 796
ReadFunct1D 796
SampleFunct1D 797
ScaleYFunct1D 797
SmoothFunct1DGauss 798
SmoothFunct1DMean 798
TransformFunct1D 799
WriteFunct1D 800
XRangeFunct1D 800
YRangeFunct1D 801

13.7 Geometry 801
AngleLI 801
AngleLx 802
DistanceLr 803
DistancePI 805
DistancePp 806
DistancePr 807
DistancePs 808
DistanceRrMin 809
DistanceRrMinDil 810
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadOcrTrainfNames</td>
<td>875</td>
</tr>
<tr>
<td>ReadOcrTrainfSelect</td>
<td>876</td>
</tr>
<tr>
<td>TestOcrClassBox</td>
<td>876</td>
</tr>
<tr>
<td>TrainOcrClassBox</td>
<td>877</td>
</tr>
<tr>
<td>WriteOcr</td>
<td>878</td>
</tr>
<tr>
<td>WriteOcrTrainf</td>
<td>879</td>
</tr>
<tr>
<td>WriteOcrTrainImage</td>
<td>880</td>
</tr>
<tr>
<td>13.13 OCV</td>
<td>881</td>
</tr>
<tr>
<td>CloseAllOcvs</td>
<td>881</td>
</tr>
<tr>
<td>CloseOcv</td>
<td>881</td>
</tr>
<tr>
<td>CreateOcvProj</td>
<td>882</td>
</tr>
<tr>
<td>DoOcvSimple</td>
<td>883</td>
</tr>
<tr>
<td>ReadOcv</td>
<td>884</td>
</tr>
<tr>
<td>TrainOcvProj</td>
<td>885</td>
</tr>
<tr>
<td>WriteOcv</td>
<td>885</td>
</tr>
<tr>
<td>13.14 Shape-from</td>
<td>886</td>
</tr>
<tr>
<td>DepthFromFocus</td>
<td>886</td>
</tr>
<tr>
<td>EstimateAlAm</td>
<td>887</td>
</tr>
<tr>
<td>EstimateSIAI1Lr</td>
<td>887</td>
</tr>
<tr>
<td>EstimateSIAI1Zc</td>
<td>888</td>
</tr>
<tr>
<td>EstimateTiltLr</td>
<td>889</td>
</tr>
<tr>
<td>EstimateTiltZc</td>
<td>889</td>
</tr>
<tr>
<td>PhotStereo</td>
<td>890</td>
</tr>
<tr>
<td>SelectGrayvaluesFromChannels</td>
<td>891</td>
</tr>
<tr>
<td>SfsModLr</td>
<td>892</td>
</tr>
<tr>
<td>SfsOrigLr</td>
<td>893</td>
</tr>
<tr>
<td>SfsPentland</td>
<td>893</td>
</tr>
<tr>
<td>ShadeHeightField</td>
<td>895</td>
</tr>
<tr>
<td><strong>14 Tuple</strong></td>
<td>897</td>
</tr>
<tr>
<td>14.1 Arithmetic</td>
<td>897</td>
</tr>
<tr>
<td>TupleAbs</td>
<td>897</td>
</tr>
<tr>
<td>TupleAcos</td>
<td>897</td>
</tr>
<tr>
<td>TupleAdd</td>
<td>898</td>
</tr>
<tr>
<td>TupleAsin</td>
<td>898</td>
</tr>
<tr>
<td>TupleAtan</td>
<td>899</td>
</tr>
<tr>
<td>TupleAtan2</td>
<td>899</td>
</tr>
<tr>
<td>TupleCeil</td>
<td>900</td>
</tr>
<tr>
<td>TupleCos</td>
<td>900</td>
</tr>
<tr>
<td>TupleCosh</td>
<td>900</td>
</tr>
<tr>
<td>TupleDiv</td>
<td>901</td>
</tr>
<tr>
<td>TupleExp</td>
<td>901</td>
</tr>
<tr>
<td>TupleFabs</td>
<td>902</td>
</tr>
<tr>
<td>TupleFloor</td>
<td>902</td>
</tr>
<tr>
<td>TupleFmod</td>
<td>903</td>
</tr>
<tr>
<td>TupleLdexp</td>
<td>903</td>
</tr>
<tr>
<td>TupleLog</td>
<td>904</td>
</tr>
<tr>
<td>TupleLog10</td>
<td>904</td>
</tr>
<tr>
<td>TupleMult</td>
<td>904</td>
</tr>
<tr>
<td>TupleNeg</td>
<td>905</td>
</tr>
<tr>
<td>TuplePow</td>
<td>905</td>
</tr>
<tr>
<td>TupleRad</td>
<td>906</td>
</tr>
<tr>
<td>TupleSin</td>
<td>906</td>
</tr>
<tr>
<td>TupleSinh</td>
<td>907</td>
</tr>
<tr>
<td>TupleSqrt</td>
<td>907</td>
</tr>
<tr>
<td>TupleSub</td>
<td>907</td>
</tr>
<tr>
<td>TupleTan</td>
<td>908</td>
</tr>
<tr>
<td>TupleTanh</td>
<td>908</td>
</tr>
</tbody>
</table>
14.2 Bit-Operations

 TupleBand .................................................. 909
 TupleBnot .................................................. 909
 TupleBor .................................................. 909
 TupleBxor .................................................. 910
 TupleLsh .................................................. 911
 TupleRsh .................................................. 911

14.3 Comparison

 TupleEqual .................................................. 912
 TupleGreater .............................................. 912
 TupleGreaterEqual ....................................... 913
 TupleLess .................................................. 913
 TupleLessEqual .......................................... 914
 TupleNotEqual .......................................... 914

14.4 Conversion

 TupleChr .................................................. 915
 TupleChrt .................................................. 915
 TupleDeg .................................................. 916
 TupleIsNumber ........................................... 916
 TupleNumber ............................................. 917
 TupleOrd .................................................. 917
 TupleOrds ................................................. 917
 TupleReal .................................................. 918
 TupleRound ................................................. 918
 TupleString ............................................... 919

14.5 Creation

 TupleConcat ............................................... 920
 TupleGenConst ........................................... 921

14.6 Element-Order

 TupleInverse .............................................. 921
 TupleSort .................................................. 921
 TupleSortIndex .......................................... 922

14.7 Features

 TupleDeviation ............................................. 922
 TupleLength ............................................... 923
 TupleMax .................................................. 923
 TupleMean .................................................. 924
 TupleMin .................................................. 924
 TupleSum ................................................... 924

14.8 Logical-Operations

 TupleAnd .................................................. 925
 TupleNot .................................................. 925
 TupleOr ..................................................... 926
 TupleXor ................................................... 926

14.9 Selection

 TupleFirstN ............................................... 927
 TupleLastN ............................................... 927
 TupleSelect .............................................. 928
 TupleSelectRange ......................................... 928
 TupleStrBitSelect .......................................... 929

14.10 String-Operators

 TupleEnvironment .......................................... 929
 TupleSplit ............................................... 930
 TupleStrFirstN ............................................ 931
 TupleStrLastN ............................................. 931
 TupleStrchr ............................................... 932
 TupleStrlen ............................................... 932
 TupleStrrchr .............................................. 933
## 15 XLD

### 15.1 Access
- GetContourXld
- GetLinesXld
- GetParallelsXld
- GetPolygonXld

### 15.2 Creation
- GenContourPolygonRoundedXld
- GenContourPolygonXld
- GenContourRegionXld
- GenContoursSkeletonXld
- GenEllipseContourXld
- GenParallelsXld
- GenPolygonsXld
- ModParallelsXld

### 15.3 Features
- AreaCenterXld
- CircularityXld
- CompactnessXld
- ContourPointNumXld
- ConvexityXld
- DiameterXld
- DistEllipseContourXld
- EccentricityXld
- EllipticAxisXld
- FitCircleContourXld
- FitEllipseContourXld
- FitLineContourXld
- GetContourAngleXld
- GetContourAttribXld
- GetContourGlobalAttribXld
- GetRegressParamsXld
- InfoParallelsXld
- LengthXld
- LocalMaxContoursXld
- MaxParallelsXld
- MomentsAnyXld
- MomentsXld
- OrientationXld
- QueryContourAttribsXld
- QueryContourGlobalAttribsXld
- SelectContoursXld
- SelectShapeXld
- SmallestCircleXld
- SmallestRectangle1Xld
- SmallestRectangle2Xld

### 15.4 Transformation
- AddNoiseWhiteContourXld
- AffineTransContourXld
- AffineTransPolygonXld
- ClipContoursXld
- CombineRoadsXld
- GenParallelContourXld
- MergeContLineScanXld
- RegressContoursXld
16 Class Hierarchy

16.1 HBarcodeX .......................... 993
16.2 HBgEstiX .......................... 994
16.3 HClassBoxX ......................... 994
16.4 HFeatureSetX ....................... 995
16.5 HFileX .......................... 995
16.6 HFramegrabberX ................. 996
16.7 HFuncti0ndX ....................... 998
16.8 HGnuplotX ......................... 999
16.9 HHistogramX ....................... 999
16.10 HHomMat2dX ..................... 1000
16.11 HHomMat3dX ..................... 1001
16.12 HImageX ........................ 1001
16.13 HHinfoX ........................ 1013
16.14 HMeasureX ....................... 1014
16.15 HMiscX ........................ 1015
16.16 HOBJECTX ....................... 1017
16.17 HOCRX ........................ 1018
16.18 HOcvX ........................ 1019
16.19 HOperatorSetX ................. 1019
16.20 HPoseX ........................ 1058
16.21 HRegionX ....................... 1059
16.22 HSerialX ......................... 1067
16.23 HShapeModelX ............... 1067
16.24 HSocketX ....................... 1068
16.25 HSystemX ...................... 1069
16.26 HTemplateX .................... 1070
16.27 HTupleX ........................ 1071
16.28 HUntypedObjectX ............ 1074
16.29 HVariationModelX .......... 1075
16.30 HWindowX ...................... 1075
16.31 HXLDX ........................ 1082
16.32 HXLDContX ..................... 1083
16.33 HXLDExtParaX ................. 1085
16.34 HXLDModParaX ............... 1085
16.35 HXLDParaX .................... 1085
16.36 HXLDPolyX .................... 1086
Chapter 1

Classification

```c
void HOperatorSetX.ClearSampset ([in] VARIANT SampKey )
```

Free memory of a data set.

*ClearSampset* frees the memory which was used for training data set having read by *ReadSampset*. This memory is only reusable in combination with *ReadSampset*.

**Parameter**

- `SampKey` (input control) .................. `feature_set ← HFeatureSetX / VARIANT` Number of the data set.

**Result**

*ClearSampset* returns TRUE. An exception handling is raised if the key `SampKey` does not exist.

**Parallelization Information**

*ClearSampset* is local and processed completely exclusively without parallelization.

**Possible Predecessors**

CreateClassBox, EnquireClassBox, LearnClassBox, WriteClassBox

**See Also**

TestSampsetBox, LearnSampsetBox, ReadSampset

**Tools**

```c
void HMiscX.CloseAllClassBox ( )
void HOperatorSetX.CloseAllClassBox ( )
```

Destroy all classifiers.

*CloseAllClassBox* deletes all classifiers and frees the used memory space. All the trained data will be lost.

**Attention**

Since all classifiers are closed by *CloseAllClassBox* all handles become invalid.

**Result**

If it is possible to close the classifiers the operator *CloseAllClassBox* returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

*CloseAllClassBox* is local and processed completely exclusively without parallelization.

**Alternatives**

*CloseClassBox*

**Tools**
void HOperatorSetX.CloseClassBox ([in] VARIANT ClassifHandle )

Destroy the classifier.

CloseClassBox deletes the classifier and frees the used memory space. All the trained data will be lost. For saving this trained data you should use WriteClassBox before.

Parameter

▷ ClassifHandle (input control) ...class_box ~ HClassBoxX / VARIANT
   Classifier’s handle number.

Result

CloseClassBox returns TRUE.

Parallelization Information

CloseClassBox is local and processed completely exclusively without parallelization.

Possible Predecessors

CreateClassBox, EnquireClassBox, LearnClassBox, WriteClassBox

See also

CreateClassBox, EnquireClassBox, LearnClassBox

Module

Tools

void HClassBoxX.CreateClassBox ( )

void HOperatorSetX.CreateClassBox ([out] VARIANT ClassifHandle )

Create a new classifier.

CreateClassBox creates a new adaptive classifier. All procedures which are explained in chapter classification refer to such a initialized classifier (of type 2). See LearnClassBox for more details about the functionality of the classifier.

Parameter

▷ ClassifHandle (output control) ...class_box ~ HClassBoxX / VARIANT
   Classifier’s handle number.

Result

CreateClassBox returns TRUE if the parameter is correct. An exception handling is raised if a classifier with this name already exists or there is not enough memory.

Parallelization Information

CreateClassBox is local and processed completely exclusively without parallelization.

Possible Successors

LearnClassBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

See also

LearnClassBox, EnquireClassBox, CloseClassBox

Module

Tools

void HClassBoxX.DescriptClassBox ([in] long Dimensions )

void HOperatorSetX.DescriptClassBox ([in] VARIANT ClassifHandle, [in] VARIANT Dimensions )

Description of the classifier.

A classifier uses a set of hyper cuboids for every class. With these hyper cuboids it is attempted to get the array attributes inside the class. DescriptClassBox returns for every class the expansion of every appropriate cuboid from dimension 1 up to Dimensions (to ‘standard_output’).
Parameter

- ClassifHandle (input control) .............................................. class_box  \( \sim HClassBoxX / \text{VARIANT} \)
  Classifier's handle number.

- Dimensions (input control) .............................................. integer  \( \sim \text{long} / \text{VARIANT} \)
  Highest dimension for output.

  Default Value: 3

Result

DescriptClassBox returns TRUE.

Parallelization Information

DescriptClassBox is local and processed completely exclusively without parallelization.

Possible Predecessors

CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successors

EnquireClassBox, LearnClassBox, WriteClassBox, CloseClassBox

See also

CreateClassBox, EnquireClassBox, LearnClassBox, ReadClassBox, WriteClassBox

Tools

```c
[out] long Class HClassBoxX.EnquireClassBox ([in] VARIANT FeatureList )

void HOperatorSetX.EnquireClassBox ([in] VARIANT ClassifHandle, [in] VARIANT FeatureList, [out] VARIANT Class )
```

Classify a tuple of attributes.

FeatureList is a tuple of any floating point- or integer numbers (attributes) which has to be assigned to a class with assistance of a previous trained (LearnClassBox) classifier. It is possible to specify attributes as unknown by indicating the symbol `*` instead of a number. If you specify n values, then all following values, i.e. the attributes n+1 until `max`, are automatically supposed to be undefined.

See LearnClassBox for more details about the functionality of the classifier.

You may call the procedures LearnClassBox and EnquireClassBox alternately, so that it is possible to classify already in the phase of learning. This means you could see when a satisfying behavior had been reached.

Parameter

- ClassifHandle (input control) .............................................. class_box  \( \sim HClassBoxX / \text{VARIANT} \)
  Classifier's handle number.

- FeatureList (input control) .............................................. real  \( \sim \text{VARIANT} ( \text{integer, real, string} ) \)
  Array of attributes which has to be classified.

  Default Value: 1.0

- Class (output control) .............................................. integer  \( \sim \text{long} / \text{VARIANT} \)
  Number of the class to which the array of attributes had been assigned.

Result

EnquireClassBox returns TRUE.

Parallelization Information

EnquireClassBox is local and processed completely exclusively without parallelization.

Possible Predecessors

CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successors

LearnClassBox, WriteClassBox, CloseClassBox

See also

TestSampsetBox, LearnClassBox, LearnSampsetBox

Alternatives

EnquireRejectClassBox
CHAPTER 1. CLASSIFICATION

Classify a tuple of attributes with rejection class.

FeatureList is a tuple of any floating point- or integer numbers (attributes) which has to be assigned to a class with assistance of a previous trained (LearnClassBox) classificator. It is possible to specify attributes as unknown by indicating the symbol ‘∗’ instead of a number. If you specify n values, then all following values, i.e. the attributes n+1 until ‘max’, are automatically supposed to be undefined.

If the array of attributes cannot be assigned to a class, i.e. the array does not reside inside of one of the hyper boxes, then in contrary to EnquireClassBox not the next class is going to be returned, but the rejection class -1 as a result is going to be passed.

See LearnClassBox for more details about the functionality of the classificator.

You may call the procedures LearnClassBox and EnquireClassBox alternately, so that it is possible to classify already in the phase of learning. By this means you could see when a satisfying behavior had been reached.

Parameter

▷ ClassifHandle (input control) . . . . . . . . . . . . . . class
  Classifier’s handle number.

▷ FeatureList (input control) . . . . . . . . . . . . . . . real
  Array of attributes which has to be classified.
  Default Value : 1.0

▷ Class (output control) . . . . . . . . . . . . . . . . . integer
  Number of the class, to which the array of attributes had been assigned or -1 for the rejection class.

Result

EnquireRejectClassBox returns TRUE.

Parallelization Information

EnquireRejectClassBox is local and processed completely exclusively without parallelization.

Possible Predecessors

CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successors

LearnClassBox, WriteClassBox, CloseClassBox

See also

TestSampsetBox, LearnClassBox, LearnSampsetBox

Alternatives

EnquireClassBox

Module

Tools

[out] long Class HClassBoxX. EnquireRejectClassBox
([[in] VARIANT FeatureList ])

void HOperatorSetX. EnquireRejectClassBox
([[in] VARIANT ClassifHandle, [in] VARIANT FeatureList, [out] VARIANT Class ])

Get information about the current parameter.

GetClassBoxParam gets the parameter of the classificator. The meaning of the parameter is explained in SetClassBoxParam.
Default values:
‘min_samples_for_split’ = 80,
’split_error’ = 0.1,
‘prop_constant’ = 0.25

Parameter

▶ **ClassifHandle** (input control) ..........................class_box  ~ HClassBoxX / VARIANT
Classifier’s handle number.

▶ **Flag** (input control) .................................string  ~ String / VARIANT
Name of the system parameter.
**Default Value**: ‘split_error’
**List of values**: Flag ∈ {‘split_error’, ‘prop_constant’, ‘used_memory’, ‘min_samples_for_split’}

▶ **Value** (output control) ..............................number  ~ VARIANT (integer, real)
Value of the system parameter.

Result

GetClassBoxParam returns TRUE. An exception handling is raised if **Flag** has been set with wrong values.

Parallelization Information

GetClassBoxParam is local and processed completely exclusively without parallelization.

Possible Predecessors

CreateClassBox, EnquireClassBox, LearnClassBox, WriteClassBox

Possible Successors

SetClassBoxParam, LearnClassBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

See also

CreateClassBox, SetClassBoxParam

Module

Tools

```
void HClassBoxX.LearnClassBox ([in] VARIANT Features, [in] long Class )
void HOperatorSetX.LearnClassBox ([in] VARIANT ClassifHandle,  
  [in] VARIANT Features, [in] VARIANT Class )
```

Train the classifier.

**Features** is a tuple of any floating point numbers or integers (attributes) which has to be assigned to the class **Class**. This class is specified by an integer. You may use procedure **EnquireClassBox** later to find the most probable class for any array (=tupel). The algorithm tries to describe the set of arrays of one class by hyper cuboids in the feature space. On demand you may even create several cuboids per class. Hence it is possible to learn disjunct concepts, too. I.e. such concepts which split in several ”cluster” of points in the feature space. The data structure is hidden to the user and only accessible with such procedures which are described in this chapter.

It is possible to specify attributes as unknown by indicating the symbol ‘∗’ instead of a number. If you specify n values, then all following values, i.e. the attributes n+1 until ‘max’, are automatically supposed to be undefined.

You may call the procedures **LearnClassBox** and **EnquireClassBox** alternately, so that it is possible to classify already in the phase of learning. By this means you could see when a satisfying behavior had been reached. The classifier is going to be bigger using further training. This means, that it is not advisable to continue training after reaching a satisfactory behavior.

Parameter

▶ **ClassifHandle** (input control) ..........................class_box  ~ HClassBoxX / VARIANT
Classifier’s handle number.

▶ **Features** (input control) ..............................number  ~ VARIANT (integer, real, string)
Array of attributes to learn.
**Default Value**: [1.0,1.5,2.0]

▶ **Class** (input control) .................................integer  ~ long / VARIANT
Class to which the array has to be assigned.
**Default Value**: 1

HALCON 6.1.4
LearnClassBox returns TRUE for a normal case. An exception handling is raised if there are memory allocation problems. The number of classes is constrained. If this limit is passed, an exception handling is raised, too.

Parallelization Information

LearnClassBox is local and processed completely exclusively without parallelization.

Possible Predecessors

CreateClassBox, EnquireClassBox

Possible Successors

TestSampsetBox, LearnClassBox, EnquireClassBox, WriteClassBox, CloseClassBox

ClearSampset

See also

TestSampsetBox, CloseClassBox, CreateClassBox, EnquireClassBox, LearnSampsetBox

Module

Tools


Train the classifier with one data set.

LearnSampsetBox trains the classifier with data for the key SampKey (see ReadSampset). The training sequence is terminated at least after NSamples examples. If NSamples is bigger than the number of examples in SampKey, then a cyclic start at the beginning occurs. If the error underpasses the value StopError, then the training sequence is prematurely terminated. StopError is calculated with N / ErrorN. Whereby N signifies the number of examples which were wrong classified during the last ErrorN training examples. Typically ErrorN is the number of examples in Sampkey and NSamples is a multiple of it. If you want a data set with 100 examples to run 5 times at most and if you want it to terminate with an error lower than 5%, then the corresponding values are NSamples = 500, ErrorN = 100 and StopError = 0.05. A protocol of the training activity is going to be written in file Outfile.

Parameter

- **ClassifHandle** (input control) ..........................class_box ～ HClassBoxX / VARIANT
  Classifier’s handle number.
- **SampKey** (input control) ...............................feature_set ～ HFeatureSetX / VARIANT
  Number of the data set to train.
- **Outfile** (input control) .................................filename ～ String / VARIANT
  Name of the protocol file.
  Default Value : ‘training prot’
- **NSamples** (input control) ..............................integer ～ long / VARIANT
  Number of arrays of attributes to learn.
  Default Value : 500
- **StopError** (input control) ............................real ～ double / VARIANT
  Classification error for termination.
  Default Value : 0.05
- **ErrorN** (input control) ...............................integer ～ long / VARIANT
  Error during the assignment.
  Default Value : 100
LearnSampsetBox returns TRUE. An exception handling is raised if key SampKey does not exist or there are problems while opening the file.

---

**Parallelization Information**

LearnSampsetBox is local and processed completely exclusively without parallelization.

---

**Possible Predecessors**

CreateClassBox

---

**Possible Successors**

TestSampsetBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

---

See also

TestSampsetBox, EnquireClassBox, LearnClassBox, ReadSampset

---

**Module**

---

**Tools**

---

```c
void HClassBoxX.ReadClassBox ([in] String FileName )
void HOperatorSetX.ReadClassBox ([in] VARIANT ClassifHandle, [in] VARIANT FileName )
```

Read the classifier from a file.

ReadClassBox reads the saved classifier from the file FileName (see WriteClassBox). The values of the current classifier are overwritten.

---

**Attention**

All values of the classifier are going to be overwritten.

---

**Parameter**

- **ClassifHandle** (input control) \(\sim\) HClassBoxX / VARIANT
  Classifier’s handle number.
- **FileName** (input control) \(\sim\) String / VARIANT
  Filename of the classifiers.

**Default Value**: 'klaasifikatorl'

---

**Result**

ReadClassBox returns TRUE. An exception handling is raised if it was not possible to open file FileName or the file has the wrong format.

---

**Parallelization Information**

ReadClassBox is local and processed completely exclusively without parallelization.

---

**Possible Predecessors**

CreateClassBox

---

**Possible Successors**

TestSampsetBox, EnquireClassBox, WriteClassBox, CloseClassBox, ClearSampset

---

See also

CreateClassBox, WriteClassBox

---

**Module**

---

---

```c
void HFeatureSetX.ReadSampset ([in] String FileName )
void HOperatorSetX.ReadSampset ([in] VARIANT FileName, [out] VARIANT SampKey )
```

Read a training data set from a file.
The training examples are accessible with the key `SampKey` by calling procedures `ClearSampset` and `LearnSampsetBox`. You may edit the file using an editor. Every row contains an array of attributes with corresponding class. An example for a format might be:

```
(1.0, 25.3, *, 17 | 3)
```

This row specifies an array of attributes which belong to class 3. In this array the third attribute is unknown. Attributes upwards 5 are supposed to be unknown, too. You may insert comments like `/* .. */` in any place.

---

**Parameter**

- **FileName** (input control) ................................................................. filename  ~ String / VARIANT
  
  Filename of the data set to train.

  **Default Value:** ‘sampset1’

- **SampKey** (output control) ................................................................. feature_set  ~ HFeatureSetX / VARIANT
  
  Identification of the data set to train.

---

**Result**

`ReadSampset` returns TRUE. An exception handling is raised if it is not possible to open the file or it contains syntax errors or there is not enough memory.

---

**Parallelization Information**

`ReadSampset` is *local* and processed *completely exclusively* without parallelization.

---

**Possible Predecessors**

- `CreateClassBox`

---

**Possible Successors**

- `TestSampsetBox`, `EnquireClassBox`, `WriteClassBox`, `CloseClassBox`, `ClearSampset`

---

**See also**

- `TestSampsetBox`, `ClearSampset`, `LearnSampsetBox`

---

**Module**

---

```c
void HClassBoxX.SetClassBoxParam ([in] String Flag, [in] VARIANT Value )

void HOperatorSetX.SetClassBoxParam ([in] VARIANT ClassifHandle, [in] VARIANT Flag, [in] VARIANT Value )
```

---

Set system parameters for classification.

`SetClassBoxParam` modifies parameter which manipulate the training sequence while calling `LearnClassBox`. Only parameters of the classifieric are modified, all other classifierics remain un-modified. ‘min_samples_for_split’ is the number of examples at least which have to train in one cuboid of this classifieric, before the cuboid is allowed to divide itself. ‘split_error’ indicates the critical error. By its exceeding the cuboid divides itself, if there are more than ‘min_samples_for_split’ examples to train. ‘prop_constant’ manipulates the extension of the cuboids. It is proportional to the average distance of the training examples in this cuboid to the center of the cuboid. More detailed:

```
extension × prop = average distance of the expectation value.
```

This relation is valid in every dimension. Hence inside a cuboid the dimensions of the feature space are supposed to be independent.

The parameters are set with problem independent default values, which must not modified without any reason. Parameters are only important during a learning sequence. They do not influence on the behavior of `EnquireClassBox`.

Default setting:

```
'min_samples_for_split’ = 80,
'split_error’ = 0.1,
'prop_constant’ = 0.25
```
Parameter

- **ClassifHandle** (input control)  
  Classifier’s handle number.
- **Flag** (input control)  
  Name of the wanted parameter.
  - Default Value: 'split_error'
  - Suggested values: Flag ∈ {‘min_samples_for_split’, ‘split_error’, ‘prop_constant’}
- **Value** (input control)  
  Value of the parameter.
  - Default Value: 0.1

Result

ReadSampset returns TRUE.

Parallelization Information

SetClassBoxParam is local and processed completely exclusively without parallelization.

Possible Predecessors
CreateClassBox, EnquireClassBox

Possible Successors
LearnClassBox, TestSampsetBox, WriteClassBox, CloseClassBox, ClearSampset

See also
EnquireClassBox, GetClassBoxParam, LearnClassBox

Module

Tools

```
[out] double Error HClassBoxX.TestSampsetBox  
([in] HFeatureSetX SampKey )

[out] double Error HFeatureSetX.TestSampsetBox  
([in] HClassBoxX ClassifHandle )

void HOperatorSetX.TestSampsetBox ([in] VARIANT ClassifHandle,  
[in] VARIANT SampKey, [out] VARIANT Error )
```

Classify a set of arrays.

In contrast to LearnSampsetBox there is not a learning here. Typically you use TestSampsetBox to classify independent test data. Error gives you information about the applicability of the learned training set on new examples.

Parameter

- **ClassifHandle** (input control)  
  Classifier’s handle number.
- **SampKey** (input control)  
  Key of the test data.
- **Error** (output control)  
  Error during the assignment.

Result

TestSampsetBox returns TRUE. An exception handling is raised, if if key SampKey does not exist or problems occur while opening the file.

Parallelization Information

TestSampsetBox is local and processed completely exclusively without parallelization.

Possible Predecessors
CreateClassBox, LearnClassBox, SetClassBoxParam

Possible Successors
EnquireClassBox, LearnClassBox, WriteClassBox, CloseClassBox, ClearSampset

HALCON 6.1.4
CHAPTER 1. CLASSIFICATION

See also
EnquireClassBox, LearnClassBox, LearnSampsetBox, ReadSampset

Tools

void HClassBoxX.WriteClassBox ( [in] String FileName )
void HOperatorSetX.WriteClassBox ( [in] VARIANT ClassifHandle,
[ in] VARIANT FileName )

Save the classifier in a file.

WriteClassBox saves the classifier in a file. You may read the data by calling ReadClassBox.

Attention
If a file with this name exists, it is overwritten without a warning. The file can not be edited.

Parameter

▷ ClassifHandle (input control) ........................................ class_box ~ HClassBoxX / VARIANT
Classifier’s handle number.

▷ FileName (input control) ............................. filename ~ String / VARIANT
Name of the file which contains the written data.

Default Value : 'klassifikator1'

Result
WriteClassBox returns TRUE. An exception handling is raised if it was not possible to open file FileName.

Parallelization Information
WriteClassBox is local and processed completely exclusively without parallelization.

Possible Predecessors
CreateClassBox, EnquireClassBox, LearnClassBox, TestSampsetBox, WriteClassBox

Possible Successors
CloseClassBox, ClearSampset

See also
CreateClassBox, ReadClassBox

Module
 Tools
2.1 Images

void HImageX.ReadImage ([in] VARIANT FileName )

void HOperatorSetX.ReadImage ([out] HUntypedObjectX Image, [in] VARIANT FileName )

Read an image with different file formats.

The operator ReadImage reads the indicated image files from the background storage and generates the image. One or more files can be indicated. The region of the generated image object (= all pixels of the matrix) is maximal chosen.

All images files written by the operator WriteImage (format 'ima') have the extension '.ima'. A description file can be available for every image in HALCON format (same file name with extension '.exp'). The type of the pixel data (byte, int4, real) can also be taken from the description file. If this information is not available the type byte is used as presetting.

Besides the HALCON format TIFF, GIF, BMP, JPEG, PNG, PCX, SUN-Raster, PGM, PPM, PBM and XWD files can also be read. The gray values of PBM images are set at the values 0 and 255. The file formats are either recognized by the extension (if indicated) or because of the internal structure of the files. If the extension is indicated the image can be found faster. In case of PGM, PPM and PBM the corresponding extension (e.g. 'pgm') or the general value 'pnm' can be used. In case of TIFF 'tiff' and 'tif' are accepted. In case of colored images an image with three color channels (matrices) is created, the red channel being stored in the first, the blue channel in the second and the green channel in the third component (channel number).

Image files are searched in the current directory (determined by the environment variable) and in the image directory of HALCON. The image directory of HALCON is preset at '. ' and '/usr/local/halcon/images' in a UNIX environment and can be set via the operator SetSystem. More than one image directory can be indicated. This is done by separating the individual directories by a colon.

Furthermore the search path can be set via the environment variable HALCONIMAGES (same structure as 'image\dir'). Example:

    setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"

HALCON also searches images in the subdirectory "'images'" (Images for the program examples). The environment variable HALCONROOT is used for the HALCON directory.

Attention

If CMYK or YCCK JPEG files are read, HALCON assumes that these files follow the Adobe Photoshop convention that the CMYK channels are stored inverted, i.e., 0 represents 100% ink coverage, rather than 0% ink as one would expect. The images are converted to RGB images using this convention. If the JPEG file does not follow this convention, but stores the CMYK channels in the usual fashion, InvertImage must be called after reading the image.

If PNG images that contain an alpha channel are read, the alpha channel is returned as the second or fourth channel of the output image, unless the alpha channel contains exactly two different gray values, in which case a one or
three channel image with a reduced domain is returned, in which the points in the domain correspond to the points with the higher gray value in the alpha channel.

Parameter

- **Image** (output iconic) ….. image ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Read image.

- **FileName** (input control) ……. filename(-array) ~ VARIANT (string)
  
  Name of the image to be read.

  **Default Value:** 'fabrik'

  **Suggested values:** FileName ∈ {'monkey', 'fabrik', 'mreut'}

Example

/* Reading an image: */
read_image(Image,'monkey').

/* Reading 3 images into an image object: */
read_image(Bildobjekt,['house_red','house_green','house_blue']).

/* Setting of search path for images on '/mnt/images' and '/home/images': */
set_system('image_dir','/mnt/images:/home/images').

Result

If the parameters are correct the operator ReadImage returns the value TRUE. If the indicated files cannot be found ReadImage returns the value FAIL. Otherwise an exception handling is raised.

Parallelization Information

ReadImage is processed under mutual exclusion against itself and without parallelization.

Possible Successors

DispImage, Threshold, RegionGrowing, CountChannels, Decompose3, ClassNdimNorm, GaussImage, FillInterlace, ZoomImageSize, ZoomImageFactor, CropPart, WriteImage, Rgb1ToGray

See also

SetSystem, WriteImage

Alternatives

ReadSequence

Module

Image / region / XLD management


Read images.

The operator ReadSequence reads unformatted image data, from a file and returns a "suitable" image. The image data must be filled consecutively pixel by pixel and line by line.
Any file headers (with the length HeaderSize bytes) are skipped. The parameters SourceWidth and SourceHeight indicate the size of the filled image. DestWidth and DestHeight indicate the size of the image. In the simplest case these parameters are the same. However, areas can also be read. The upper left corner of the required image area can be determined via StartRow and StartColumn.

The pixel types 'bit', 'byte', 'short' (16 bits, unsigned), 'signed_short' (16 bits, signed), 'swapped_long' (32 bits, with swapped segments), and 'real' (32 bit floating point numbers) are supported. Furthermore, the operator ReadSequence enables the extraction of components of a RBG image, if a trip le of three bytes (in the sequence “red”, “green”, “blue”) was filed in the image file. For the red component the pixel type ‘r_byte’ must be chosen, and correspondingly for the green and blue components ‘g_byte’ or ‘b_byte’, respectively. 'MSBFirst' (most significant bit first) or the inversion thereof ('LSBFirst') can be chosen for the bit order (BitOrder). The byte orders (ByteOrder) 'MSBFirst' (most significant byte first) or 'LSBFirst', respectively, are processed analogously. Finally an alignment (Pad) can be set at the end of the line: 'byte', 'short' or 'long'. If a whole image sequence is stored in the file a single image (beginning at Index 1) can be chosen via the parameter Index.

Image files are searched in the current directory (determined by the environment variable) and in the image directory of HALCON. The image directory of HALCON is preset at ‘.’ and '/usr/local/halcon/images' in a UNIX environment and can be set via the operator SetSystem. More than one image directory can be indicated. This is done by separating the individual directories by a colon.

Furthermore the search path can be set via the environment variable HALCONIMAGES (same structure as 'image_dir'). Example:

```bash
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"
```

HALCON also searches images in the subdirectory ”’images’” (Images for the program examples). The environment variable HALCONROOT is used for the HALCON directory.

---

**Attention**

If files of pixel type 'real' are read and the byte order is chosen incorrectly (i.e., differently from the byte order in which the data is stored in the file) program error and even crashes because of floating point exceptions may result.

---

**Parameter**

- **Image** (output iconic) .................. image \(\rightarrow\) HImageX / HUntypedObjectX ( byte, int2, uint2, int4 ) Image read.
- **HeaderSize** (input control) .................... integer \(\rightarrow\) long / VARIANT Number of bytes for file header.
  Default Value : 0
- **SourceWidth** (input control) .................. extent.x \(\rightarrow\) long / VARIANT Number of image columns of the filed image.
  Default Value : 512
- **SourceHeight** (input control) .................. extent.y \(\rightarrow\) long / VARIANT Number of image lines of the filed image.
  Default Value : 512
- **StartRow** (input control) ........................ point.y \(\rightarrow\) long / VARIANT Starting point of image area (line).
  Default Value : 0
  Restriction : (StartRow < SourceHeight)
- **StartColumn** (input control) .................. point.x \(\rightarrow\) long / VARIANT Starting point of image area (column).
  Default Value : 0
  Restriction : (StartColumn < SourceWidth)
- **DestWidth** (input control) .................. extent.x \(\rightarrow\) long / VARIANT Number of image columns of output image.
  Default Value : 512
  Restriction : (DestWidth \(\leq\) SourceWidth)
- **DestHeight** (input control) .................. extent.y \(\rightarrow\) long / VARIANT Number of image lines of output image.
  Default Value : 512
  Restriction : (DestHeight \(\leq\) SourceHeight)
CHAPTER 2. FILE

PixelType (input control) ............................................. string  \( \sim \) String / VARIANT
Type of pixel values.
Default Value: ‘byte’
List of values: PixelType \( \in \{ \text{`bit’}, \text{`byte’}, \text{`r_byte’}, \text{`g_byte’}, \text{`b_byte’}, \text{`short’}, \text{`signed_short’}, \text{`long’}, \text{`swapped_long’} \} \)

BitOrder (input control) ............................................. string  \( \sim \) String / VARIANT
Sequence of bits within one byte.
Default Value: ‘MSBFirst’
List of values: BitOrder \( \in \{ \text{`MSBFirst’}, \text{`LSBFirst’} \} \)

ByteOrder (input control) ............................................. string  \( \sim \) String / VARIANT
Sequence of bytes within one ‘short’ unit.
Default Value: ‘MSBFirst’
List of values: ByteOrder \( \in \{ \text{`MSBFirst’}, \text{`LSBFirst’} \} \)

Pad (input control) ............................................. string  \( \sim \) String / VARIANT
Data units within one image line (alignment).
Default Value: ‘byte’
List of values: Pad \( \in \{ \text{`byte’}, \text{`short’}, \text{`long’} \} \)

Index (input control) ............................................. integer  \( \sim \) long / VARIANT
Number of images in the file.
Default Value: 1
(result)

FileName (input control) ............................................. filename  \( \sim \) String / VARIANT
Name of input file.

If the parameter values are correct the operator \texttt{ReadSequence} returns the value TRUE. If the file cannot be
opened FAIL is returned. Otherwise an exception handling is raised.

\textbf{Parallelization Information}
\texttt{ReadSequence} is processed under mutual exclusion against itself and without parallelization.

\textbf{Possible Successors}
DispImage, CountChannels, Decompose3, WriteImage, Rgb1ToGray

\textbf{See also}

ReadImage

\textbf{Alternatives}

ReadImage

\textbf{Module}
Image / region / XLD management

\begin{verbatim}
void \texttt{HImageX.WriteImage} ([in] String Format, [in] long FillColor, 
[in] VARIANT FileName ) 

void \texttt{HOperatorSetX.WriteImage} ([in] IObjectX Image, 
\end{verbatim}

Write images in graphic formats.

The operator \texttt{WriteImage} returns the indicated image (\texttt{Image}) in different image formats in files. Pixels
outside the region receive the color defined by \texttt{FillColor}. For gray value images a value between 0 (black) and 255 (white) must be passed, with RGB color images the RGB values can be passed directly as a hexadecimal
value: e.g., 0xffff00 for a yellow background (red=255, green=255, blue=0).

The following formats are currently supported:

\texttt{‘tiff’}  TIFF format, 3-channel-images (RGB): 3 samples per pixel; other images (grayvalue images): 1 sample per
pixel, 8 bits per sample, uncompressed,72 dpi; file extension: *.tif

\texttt{‘bmp’}  Windows-BMP format, 3-channel-images (RGB): 3 bytes per pixel; other images (gray value image): 1
byte per pixel; file extension: *.bmp

HALCON/COM Reference Manual, 2005-2-1
'jpeg' JPEG format, with lost of information; together with the format string the quality value determining the compression rate can be provided: e.g., 'jpeg 30'. Attention: images stored for being processed later should not be compressed with the jpeg format according to the lost of information.

'png' PNG format (lossless compression); together with the format string, a compression level between 0 and 9 can be specified, where 0 corresponds to no compression and 9 to the best possible compression. Alternatively, the compression can be selected with the following strings: 'best', 'fastest', and 'none'. Hence, examples for correct parameters are 'png', 'png 7', and 'png none'. Images of type byte and uint2 can be stored in PNG files. If an image with a reduced domain is written, the region is stored as the alpha channel, where the points within the domain are stored as the maximum gray value of the image type and the points outside the domain are stored as the gray value 0. If an image with a full domain is written, no alpha channel is stored.

'ima' The data is written binary line by line (without header or carriage return). The size of the image and the pixel type are stored in the description file "FileName.exp". byte, int2, uint2, int4 and real images can be written. The file extension is: '.ima'

Attention

Parameter

▷ **Image** (input iconic) ...... image(-array) \(\sim\) HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real )

Output image(s).

▷ **Format** (input control) .................................................. string \(\sim\) String / VARIANT

Graphic format.

Default Value: 'tiff'

List of values: Format \(\in\) { 'tiff', 'bmp', 'jpeg', 'ima', 'jpeg 100', 'jpeg 80', 'jpeg 60', 'jpeg 40', 'jpeg 20', 'png', 'png best', 'png fastest', 'png none' }

▷ **FillColor** (input control) .............................................. integer \(\sim\) long / VARIANT

Fill gray value for pixels not belonging to the image region.

Default Value: 0

Suggested values: FillColor \(\in\) {-1, 0, 255, 0xff0000, 0xff00}

▷ **FileName** (input control) ............................................ filename(-array) \(\sim\) VARIANT( string )

Name of graphic file.

Result

If the parameter values are correct the operator **WriteImage** returns the value TRUE. Otherwise an exception handling is raised. If the file cannot be opened **WriteImage** returns FAIL.

Parallelization Information

**WriteImage** is processed under *mutual exclusion* against itself and without parallelization.

Possible Predecessors

OpenWindow, ReadImage

Module

Image / region / XLD management

2.2 Misc

```cpp
void HMiscX.FileExists ([in] String FileName )

void HOperatorSetX.FileExists ([in] VARIANT FileName )
```

Check whether file exists.

The operator **FileExists** checks whether the indicated file already exists. If this is the case, the operator **FileExists** returns TRUE, otherwise FALSE.
_Parameter_

- **FileName** (input control) .......................................................... filename ~ String / VARIANT
  
  Name of file to be checked.

  **Default Value:** /bin/cc

_Result_

The operator **FileExists** returns the value TRUE (file exists) or FALSE (file does not exist).

_Parallelization Information_

**FileExists** is reentrant and processed without parallelization.

_OpenFile_

_Possible Successors_

**OpenFile**

**Alternatives**

**OpenFile**

**Module**

_Basic operators_

---

```c
void HHomMat2dX.ReadWorldFile (in String FileName )
void HOperatorSetX.ReadWorldFile (in VARIANT FileName,
                                 [out] VARIANT WorldTransformation )
```

Read the geo coding from an ARC/INFO world file.

**ReadWorldFile** reads a geocoding from an ARC/INFO world file with the file name **FileName** and returns it as a homogeneous 2D transformation matrix in **WorldTransformation**. To find the file **FileName**, all directories contained in the HALCON system variable ’image_dir’ (usually this is the content of the environment variable HALCONIMAGES) are searched (see **ReadImage**). This transformation matrix can be used to transform XLD contours to the world coordinate system before writing them with **WriteContourXldArcInfo**. If the matrix **WorldTransformation** is inverted by calling **HomMat2dInvert**, the resulting matrix can be used to transform contours that have been read with **ReadContourXldArcInfo** to the image coordinate system.

---

_Parameter_

- **FileName** (input control) .......................................................... filename ~ String / VARIANT
  
  Name of the ARC/INFO world file.

- **WorldTransformation** (output control) .......................... affine2d ~ HHomMat2dX / VARIANT (real)
  
  Transformation matrix from image to world coordinates.

  **Number of elements:** 6

_Result_

If the parameters are correct and the world file could be read, the operator **ReadWorldFile** returns the value TRUE. Otherwise an exception is raised.

_Parallelization Information_

**ReadWorldFile** is reentrant and processed without parallelization.

_Possible Successors_

**HomMat2dInvert**, **AffineTransContourXld**, **AffineTransPolygonXld**

_See also_

**WriteContourXldArcInfo**, **ReadContourXldArcInfo**, **WritePolygonXldArcInfo**, **ReadPolygonXldArcInfo**

---

_Sub-pixel operators_
2.3 Region

```plaintext
void HRegionX.ReadRegion ([in] String FileName )
void HOperatorSetX.ReadRegion ([out] HUntypedObjectX Region, [in] VARIANT FileName )
```

Read binary images or HALCON regions.

The operator `ReadRegion` reads regions from a binary file. The data is stored in packed form.

**Tiff:** Binary Tiff images with extension `‘tiff’` or `‘tif’`. The result is always one region. The color black is used as foreground.

**BMP:** Binary Windows bitmap images with extension `‘bmp’`. The result is always one region. The color black is used as foreground.

**HALCON regions:** File format of HALCON for regions. Several images can be stored (in one file) or read simultaneously via the operators `WriteRegion` and `ReadRegion`. All region files have the extension `.reg`, which is not indicated when reading or writing the file.

A search path (`‘image_dir’`) can be defined analogous to the operator `ReadImage`.

---

**Attention**

---

**Parameter**

- `Region` (output iconic) .................................... region(-array) ~ HRegionX / HUntypedObjectX Read region.
- `FileName` (input control) ....................................... filename ~ String / VARIANT Name of the region to be read.

---

/** Reading of regions and giving them gray values. */
read_image(Img,’bild_test5’)
read_region(Regs,’reg_test5’)
reduce_domain(Img,Regs,Res)

---

**Result**

If the parameter values are correct the operator `ReadRegion` returns the value TRUE. If the file cannot be opened FAIL is returned. Otherwise an exception handling is raised.

---

**Parallelization Information**

`ReadRegion` is reentrant and processed without parallelization.

---

**Possible Predecessors**

ReadImage
ReduceDomain, DispRegion

---

**Possible Successors**

WriteRegion, ReadImage

---

See also

**Module**

Image / region / XLD management

---

```plaintext
void HRegionX.WriteRegion ([in] String FileName )
void HOperatorSetX.WriteRegion ([in] IObjectX Region, [in] VARIANT FileName )
```

Write regions on file.
The operator **WriteRegion** writes the regions of the input images (in runlength coding) to a binary file. The data is stored in packed form. The output data can be read via the operator **ReadRegion**. If no extension has been specified in **FileName**, the extension '.reg' is appended to **FileName**.

---

**Attention**

---

- **Region** (input iconic) .......................... region(-array)  \(\sim H\text{RegionX} / \text{IHObjectX} \)
  Region of the images which are returned.
- **FileName** (input control) .......................... filename  \(\sim \text{String} / \text{VARIANT} \)
  Name of region file.
  
  **Default Value**: 'region.reg'

---

**Example**

```
regiongrowing(Img, Segmente, 3, 3, 5, 10)
write_region(Segmente, 'result1').
```

---

**Result**

If the parameter values are correct the operator **WriteRegion** returns the value TRUE. If the file cannot be opened FAIL is returned. Otherwise an exception handling is raised.

---

**Parallelization Information**

**WriteRegion** is reentrant and processed without parallelization.

---

**Possible Predecessors**

OpenWindow, ReadImage, ReadRegion, Threshold, Regiongrowing

---

**See also**

**ReadRegion**

---

**Module**

Image / region / XLD management

---

### 2.4 Text

```c
void HMiscX.CloseAllFiles ( )

void HOperatorSetX.CloseAllFiles ( )
```

Close all open files.

**CloseAllFiles** closes all open files.

---

**Attention**

Since all files are closed by **CloseAllFiles** all handles become invalid.

---

**Result**

If it is possible to close the files the operator **CloseAllFiles** returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

**CloseAllFiles** is local and processed completely exclusively without parallelization.

---

**Alternatives**

Closing a text file.

---

**Module**

Basic operators

```c
void HOperatorSetX.CloseFile ([in] VARIANT FileHandle )
```

Closing a text file.

---

**HALCON/COM Reference Manual, 2005-2-1**
The operator **CloseFile** closes a file which was opened via the operator **OpenFile**.

**Parameter**

- **FilHandle** (input control) → **HFileX / VARIANT**
  
  **Example**

```c
open_file(’/tmp/data.txt’,’input’,FileHandle)
// ....
close_file(FileHandle).
```

**Result**

If the file handle is correct **CloseFile** returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

**CloseFile** is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessors**

- **OpenFile**

**See also**

- **OpenFile**

**Module**

**Basic operators**

```c
void HFileX.FnewLine ( )
```

```c
void HOperatorSetX.FnewLine ([in] VARIANT FileHandle )
```

Create a line feed.

The operator **FnewLine** puts out a line feed into the output file. At the same time the output buffer is cleaned.

**Attention**

**Parameter**

- **FilHandle** (input control) → **HFileX / VARIANT**

**Example**

```c
fwrite_string(FileHandle,’Good Morning’)
fnew_line(FileHandle)
```

**Result**

If an output file is open and it can be written to the file the operator **FnewLine** returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

**FnewLine** is **reentrant** and processed **without** parallelization.

**Possible Predecessors**

- **FwriteString**

**See also**

- **FwriteString**

**Module**

**Basic operators**
Read a character from a text file.

The operator \texttt{FreadChar} reads a character from the current input file. \texttt{FreadChar} returns the character sequence ‘\texttt{eof}’. At the end of a line the value ‘\texttt{nl}’ is returned.

\begin{verbatim}
repeat >
  fread_char(FileHandle:Char)
  (if(Char = ‘\texttt{nl}’) > fnew_line(FileHandle)) |
  (if(Char != ‘\texttt{nl}’) > fwrite_string(FileHandle,Char))
until(Char = ‘\texttt{eof}’).
\end{verbatim}

\textbf{Result}  
If an input file is open the operator fread_char returns TRUE. Otherwise an exception handling is raised.

\textbf{Parallelization Information}
\texttt{FreadChar} is reentrant and processed without parallelization.

\textbf{Possible Predecessors}
\texttt{OpenFile}

\textbf{Possible Successors}
\texttt{CloseFile}

\textbf{See also}
\texttt{OpenFile, CloseFile, FreadString}

\textbf{Alternatives}
\texttt{FreadString, ReadString}

\textbf{Module}
Basic operators

---

Read strings from a text file.

The operator \texttt{FreadString} reads a string from the current input file. A string begins with the first representable character: letters, numbers, additional characters (except blanks). A string ends when a blank or a line skip is reached. Several successive line skips are ignored. If the end of the file is reached \texttt{IsEOF} return the value 1, otherwise 0.

\begin{verbatim}
repeat >
  fread_string(FileHandle:OutString)
  (if(Char = ‘\texttt{nl}’) > fnew_line(FileHandle)) |
  (if(Char != ‘\texttt{nl}’) > fwrite_string(FileHandle,Char))
until(Char = ‘\texttt{eof}’).
\end{verbatim}

\textbf{Result}  
If an input file is open the operator fread_string returns TRUE. Otherwise an exception handling is raised.

\textbf{Parallelization Information}
\texttt{FreadString} is reentrant and processed without parallelization.

\textbf{Possible Predecessors}
\texttt{OpenFile}

\textbf{Possible Successors}
\texttt{CloseFile}

\textbf{See also}
\texttt{OpenFile, CloseFile, FreadString}

\textbf{Alternatives}
\texttt{FreadString, ReadString}

\textbf{Module}
Basic operators

---
Parameter

- **FileHandle** (input control) .......................... file  \(\sim HFileX / \text{VARIANT} \)
  File handle.
- **OutString** (output control) .......................... string  \(\sim \text{String} / \text{VARIANT} \)
  Read character sequence.
- **IsEOF** (output control) .............................. integer  \(\sim \text{long} / \text{VARIANT} \)
  Reached end of file.

Example

```c
fwrite_string(FileHandle,'Please enter text and return: ..')
fread_string(FileHandle,OutString,IsEOF) >
fwrite_string(FileHandle,['here it is again: ',OutString])
fnew_line(FileHandle).
```

Result

If a file is open and a suitable string is read **FreadString** returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

**FreadString** is reentrant and processed without parallelization.

Possible Predecessors

- **OpenFile**

Possible Successors

- **CloseFile**

See also

- **OpenFile**, **CloseFile**, **FreadChar**

Alternatives

- **FreadChar**, **ReadString**

Module

**Basic operators**

```c
void HFileX.FwriteString ([in] \text{VARIANT} String)
void HOperatorSetX.FwriteString ([in] \text{VARIANT} FileHandle,
[in] \text{VARIANT} String)
```

Write values in a text file.

The operator **FwriteString** puts out a string or numbers on the output file. The operator **OpenFile** opens a file. The call **SetSystem(::{‘flushFile’, <boolean-value>:})** determines whether the output characters are put out directly on the output medium. If the value ‘flushFile’ is set to ‘false’ the characters (especially in case of screen output) show up only after the operator **FnewLine** is called.

Strings as well as whole numbers and floating point numbers can be used as arguments. If more than one value serves as input the values are put out consecutively without blanks.

Attention

Parameter

- **FileHandle** (input control) .......................... file  \(\sim HFileX / \text{VARIANT} \)
  File handle.
- **String** (input control) .............................. string(-array)  \(\sim \text{VARIANT} (\text{integer, real, string}) \)
  Values to be put out on the text file.

Default Value: ‘hallo’
Example

fwrite_string(FileHandle, ['text with numbers:', 5, ' and ', 1.0]).
/* results in the following output: */
/* 'text with numbers:5 and 1.00000' */

Result
If the writing procedure was carried out successfully the operator fwriteString returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
fwriteString is reentrant and processed without parallelization.

Possible Predecessors
OpenFile

Possible Successors
CloseFile

See also
OpenFile, CloseFile, SetSystem

Alternatives
WriteString

Module
Basic operators

void HFileX.OpenFile ([in] String FileName, [in] String FileType )
void HOperatorSetX.OpenFile ([in] VARIANT FileName, [in] VARIANT FileType, [out] VARIANT FileHandle )

Open text file.
The operator OpenFile opens a file. FileType determines whether this file is an input ('input') or output file ('output' or 'append'). OpenFile creates files which can be accessed either by reading ('input') or by writing ('output' or 'append') are created. For terminal input and output the file names 'standard' ('input' and 'output') and 'error' (only 'output') are reserved.

Attention

Parameter

> FileName (input control) ........................................... filename ~ String / VARIANT
Name of file to be opened.
Default Value: 'standard'
Suggested values: FileName ∈ {'standard', 'error', '/tmp/dat.dat'}

> FileType (input control) ........................................... string ~ String / VARIANT
Type of file.
Default Value: 'output'
List of values: FileType ∈ {'input', 'output', 'append'}

> FileHandle (output control) ................................. file ~ HFileX / VARIANT
File handle.

Example

/* Creating of an output file with the name '/tmp/log.txt' and writing */
/* of one string: */
open_file('/tmp/log.txt', 'output', FileHandle)
fwrite_string(FileHandle, 'these are the first and last lines')
fnew_line(FileHandle)
close_file(FileHandle).
If the parameters are correct the operator `OpenFile` returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

`OpenFile` is local and processed completely exclusively without parallelization.

**Possible Successors**

`FwriteString`, `FreadChar`, `FreadString`, `CloseFile`

**See also**

`CloseFile`, `FwriteString`, `FreadChar`, `FreadString`

---

### 2.5 Tuple

**Read a tuple from a file.**

The operator `ReadTuple` reads the contents of `FileName` and converts it into `Tuple`. The file has to be generated by `WriteTuple`.

**Parameter**

- **FileName** (input control) 
  Name of the file to be read.
- **Tuple** (output control) 
  Tuple with any kind of data.

**Result**

If the parameters are correct the operator `ReadTuple` returns the value TRUE. If the file could not be opened `ReadTuple` returns FAIL. Otherwise an exception handling is raised.

**Parallelization Information**

`ReadTuple` is reentrant and processed without parallelization.

**See also**

`WriteTuple`, `GnuplotPlotCtrl`, `WriteImage`, `WriteRegion`, `OpenFile`

---

### 2.5 Tuple

**Write a tuple to a file.**

The operator `WriteTuple` writes the contents of `Tuple` to a file. The data is written in an ASCII format. Therefore, the file can be exchanged between different architectures. There is no specific extension for this kind of file.
### 2.6 XLD

Read XLD contours to a file in ARC/INFO generate format.

**ReadContourXldArcInfo** reads the lines stored in ARC/INFO generate format in the file **FileName**, and returns them as XLD contours in **Contours**. To find the file **FileName**, all directories contained in the HALCON system variable 'image\dir' (usually this is the content of the environment variable HALCONIMAGES) are searched (see **ReadImage**). The returned contours are in world coordinates. They can be transformed to the image coordinate system with the operator **AffineTransContourXld**. The necessary transformation matrix can be generated by using **ReadWorldFile** to read the transformation matrix from image to world coordinates, and inverting this matrix by calling **HomMat2dInvert**.

#### Parameter

- **Contours** (output iconic) ...............................xld_cont(-array) \(\sim\) HXLDContX / HUntypedObjectX
  
  Read XLD contours.

- **FileName** (input control) .................................filename \(\sim\) String / VARIANT
  
  Name of the ARC/INFO file.

#### Example

```c
/* Read the transformation and invert it */
read_world_file ('image.tfw', WorldTransformation)
hom_mat2d_invert (WorldTransformation, ImageTransformation)
/* Read the image */
read_image (Image, 'image.tif')
/* Read the line data */
read_contour_xld_arc_info (LinesWorld, 'lines.gen')
/* Transform the line data to image coordinates */
affine_trans_contour_xld (LinesWorld, Lines, ImageTransformation)
```

#### Result

If the parameters are correct and the file could be read, the operator **ReadContourXldArcInfo** returns the value TRUE. Otherwise an exception is raised.
Parallelization Information

ReadContourXldArcInfo is reentrant and processed without parallelization.

Possible Successors

HomMat2dInvert, AffineTransContourXld

See also

ReadWorldFile, WriteContourXldArcInfo, ReadPolygonXldArcInfo

Module

Sub-pixel operators

void HXLDPolyX.ReadPolygonXldArcInfo ([in] String FileName )

void HOperatorSetX.ReadPolygonXldArcInfo ([out] HUntypedObjectX Polygons, [in] VARIANT FileName )

Read XLD polygons from a file in ARC/INFO generate format.

ReadPolygonXldArcInfo reads the lines stored in ARC/INFO generate format in the file FileName, and returns them as XLD polygons in Polygons. To find the file FileName, all directories contained in the HALCON system variable 'image_dir' (usually this is the content of the environment variable HALCONIMAGES) are searched (see ReadImage). The returned polygons are in world coordinates. They can be transformed to the image coordinate system with the operator AffineTransPolygonXld. The necessary transformation matrix can be generated by using ReadWorldFile to read the transformation matrix from image to world coordinates, and inverting this matrix by calling HomMat2dInvert.

Parameter

◶ Polygons (output iconic) .......................... xld_poly(-array) 〜 HXLDPolyX / HUntypedObjectX

Read XLD polygons.

◶ FileName (input control) .............................. filename 〜 String / VARIANT

Name of the ARC/INFO file.

Example

/* Read the transformation and invert it */
read_world_file (‘image.tfw’, WorldTransformation)
hom_mat2d_invert (WorldTransformation, ImageTransformation)
/* Read the image */
read_image (Image, ‘image.tif’)
/* Read the line data */
read_polygon_xld_arc_info (LinesWorld, ‘lines.gen’)
/* Transform the line data to image coordinates */
affine_trans_polygon_xld (LinesWorld, Lines, ImageTransformation)

Result

If the parameters are correct and the file could be read, the operator ReadPolygonXldArcInfo returns the value TRUE. Otherwise an exception is raised.
Write XLD contours to a file in ARC/INFO generate format.

WriteContourXldArcInfo writes the XLD contours Contours to an ARC/INFO generate format file with name FileName. If no absolute path is given in FileName, the output file is created in the current directory of the HALCON process. The contours must have been transformed to the world coordinate system with AffineTransContourXld beforehand. The necessary transformation can be read from an ARC/INFO world file with ReadWorldFile.

Parameter

- **Contours** (input iconic) .............................. xld_cont(-array) \(\sim\) HXLDContX / IHObjectX  
  XLD contours to be written.

- **FileName** (input control) ............................ filename \(\sim\) String / VARIANT  
  Name of the ARC/INFO file.

Example

/* Read transformation and image */  
read_world_file ('image.tfw', WorldTransformation)  
read_image (Image, 'image.tif')  
/* Segment image */  
...  
/* Write result */  
affine_trans_contour_xld (Contours, ContoursWorld, WorldTransformation)  
write_contour_xld_arc_info (ContoursWorld, 'result.gen')

Result

If the parameters are correct and the file could be written, the operator WriteContourXldArcInfo returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

WriteContourXldArcInfo is reentrant and processed without parallelization.

Possible Predecessors

AffineTransContourXld

See also

ReadWorldFile, ReadContourXldArcInfo, WritePolygonXldArcInfo

Module

Sub-pixel operators

Write XLD polygons to a file in ARC/INFO generate format.

WritePolygonXldArcInfo writes the XLD polygons Polygons to an ARC/INFO generate format file with name FileName. If no absolute path is given in FileName, the output file is created in the current directory of the HALCON process. The polygons must have been transformed to the world coordinate system with AffineTransPolygonXld beforehand. The necessary transformation can be read from an ARC/INFO world file with ReadWorldFile.

Attention

The XLD contours that are possibly referenced by Polygons are not stored in the ARC/INFO file, since this is not possible with the ARC/INFO generate file format. Therefore, when the polygons are read again using
ReadPolygonXldArcInfo, this information is lost, and no references to contours are generated for the polygons. Hence, operators that access the contours associated with a polygon, e.g., SplitContoursXld will not work correctly.

Parameter

- **Polygons** (input iconic) ......................... xld_poly(-array) \(\rightarrow\) HXLDPolyX / IHObjectX
  XLD polygons to be written.

- **FileName** (input control) ......................... filename \(\rightarrow\) String / VARIANT
  Name of the ARC/INFO file.

Example

```c
/* Read transformation and image */
read_world_file ('image.tfw', WorldTransformation)
read_image (Image, 'image.tif')
/* Segment image */
...
/* Write result */
affine_trans_polygon_xld (Polygons, PolygonsWorld, WorldTransformation)
write_polygon_xld_arc_info (PolygonsWorld, 'result.gen')
```

Result

If the parameters are correct and the file could be written, the operator WritePolygonXldArcInfo returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

WritePolygonXldArcInfo is reentrant and processed without parallelization.

Possible Predecessors

AffineTransPolygonXld

See also

ReadWorldFile, ReadPolygonXldArcInfo, WriteContourXldArcInfo

Module

Sub-pixel operators
Chapter 3

Filter

3.1 Affine-Transformations

Apply an arbitrary affine 2D transformation to images. AffineTransImage applies an arbitrary affine 2D transformation, i.e., scaling, rotation, translation, and slant (skewing), to the images given in Image and returns the transformed images in ImageAffinTrans. The affine transformation is described by the homogeneous transformation matrix given in HomMat2D, which can be created using the operators HomMat2dIdentity, HomMat2dScale, HomMat2dRotate, HomMat2dTranslate, etc., or be the result of operators like VectorAngleToRigid.

The components of the homogeneous transformation matrix are interpreted as follows: The row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system for the image. In particular, this assures that rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

The region of the input image is ignored, i.e., assumed to be the full rectangle of the image. The region of the resulting image is set to the transformed rectangle of the input image. If necessary, the resulting image is filled with zero (black) outside of the region of the original image.

Generally, transformed points will lie between pixel coordinates. Therefore, an appropriate interpolation scheme has to be used. The interpolation can also be used to avoid aliasing effects for scaled images. The quality and speed of the interpolation can be set by the parameter Interpolation:

- **none**  No interpolation: The gray value is determined from the nearest pixel’s gray value (possibly low quality, very fast).
- **constant**  Use equally weighted interpolation between adjacent pixels (medium quality and run time).
- **weighted**  Use Gaussian interpolation between adjacent pixels (best quality, slow).

In addition, the system parameter 'int_zooming' (see SetSystem) affects the accuracy of the transformation. If 'int_zooming' is set to 'true', the transformation for byte, int2 and uint2 images is carried out internally using fixed point arithmetic, leading to much shorter execution times. However, the accuracy of the transformed gray values is smaller in this case. For byte images, the differences to the more accurate calculation (using 'int_zooming' = 'false') is typically less than two gray levels. Correspondingly, for int2 and uint2 images, the gray value differences are less than 1/128 times the dynamic gray value range of the image, i.e., they can be as large as 512 gray levels.
if the entire dynamic range of 16 bit is used. For real images, the parameter ‘intZooming’ does not affect the accuracy, since the internal calculations are always done using floating point arithmetic.

The size of the target image can be controlled by the parameter AdaptImageSize: With value ‘true’ the size will be adapted so that no clipping occurs at the right or lower edge. With value ‘false’ the target image has the same size as the input image. Note that, independent of AdaptImageSize, the image is always clipped at the left and upper edge, i.e., all image parts that have negative coordinates after the transformation are clipped.

Attention

The region of the input image is ignored.

Parameter

- **Image** (input iconic) …… (multichannel-)image(-array) \( \sim \) HImageX / IHObjectX ( byte, int2, uint2, real )
  - Input image.
- **ImageAffinTrans** (output iconic) …… (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX ( byte, int2, uint2, real )
  - Transformed image.
- **HomMat2D** (input control) ……………… affine2d \( \sim \) VARIANT / HHomMat2dX ( real )
  - Input transformation matrix.
  - Number of elements : 6
- **Interpolation** (input control) ………………… string \( \sim \) String / VARIANT
  - Type of interpolation.
  - Default Value : ‘constant’
  - List of values : Interpolation \( \in \) \{‘none’, ‘constant’, ‘weighted’\}
- **AdaptImageSize** (input control) ………………… string \( \sim \) String / VARIANT
  - Adaptation of size of result image.
  - Default Value : ‘false’
  - List of values : AdaptImageSize \( \in \) \{‘true’, ‘false’\}

Example

/* Reduction of an image (512 x 512 Pixels) by 50%, rotation */
/* by 180 degrees and translation to the upper-left corner: */

hom_mat2d_identity(Matrix1)
hom_mat2d_scale(Matrix1,0.5,0.5,256.0,256.0,Matrix2)
hom_mat2d_rotate(Matrix2,3.14,256.0,256.0,Matrix3)
hom_mat2d_translate(Matrix3,-128.0,-128.0,Matrix4,)
affine_trans_image(Image,TransImage,Matrix4,1).

/* Enlarging the part of an image in the interactively */
/* chosen rectangular window sector: */

draw_rectangle2(WindowHandle,L,C,Phi,L1,L2)
hom_mat2d_identity(Matrix1)
get_system(width,Width)
get_system(height,Height)
hom_mat2d_translate(Matrix1,Height/2.0-L,Width/2.0-C,Matrix2)
hom_mat2d_rotate(Matrix2,3.14-Phi,Height/2.0,Width/2.0,Matrix3)
hom_mat2d_scale(Matrix3,Height/(2.0*L2),Width/(2.0*L1),
    Height/2.0,Width/2.0,Matrix4)
    affine_trans_image(Image,Matrix4,TransImage,1).

Result

AffineTransImage returns TRUE if all parameter values are correct. If the input is empty the behaviour can be set via SetSystem(::‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

AffineTransImage is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors

HomMat2dIdentity, HomMat2dTranslate, HomMat2dRotate, HomMat2dScale
3.1. AFFINE-TRANSFORMATIONS

See also

SetPartStyle

Alternatives

AffineTransImageSize, ZoomImageSize, ZoomImageFactor, MirrorImage, RotateImage, AffineTransRegion

Module

Image filters

```
[out] HImageX ImageAffinTrans HImageX.AffineTransImageSize
([in] VARIANT HomMat2D, [in] String Interpolation, [in] long Width,
[in] long Height )
```

```
[out] HImageX ImageAffinTrans HHomMat2dX.AffineTransImageSize
([in] HImageX Image, [in] String Interpolation, [in] long Width,
[in] long Height )
```

```
void HOperatorSetX.AffineTransImageSize ([in] IHObjectX Image,
[out] HUntypedObjectX ImageAffinTrans, [in] VARIANT HomMat2D,
```

Apply an arbitrary affine 2D transformation to an image and specify the output image size. **AffineTransImageSize** applies an arbitrary affine 2D transformation, i.e., scaling, rotation, translation, and slant (skewing), to the images given in **Image** and returns the transformed images in **ImageAffinTrans**.

The affine transformation is described by the homogeneous transformation matrix given in **HomMat2D**, which can be created using the operators **HomMat2dIdentity**, **HomMat2dScale**, **HomMat2dRotate**, **HomMat2dTranslate**, etc., or be the result of operators like **VectorAngleToRigid**.

The components of the homogeneous transformation matrix are interpreted as follows: The row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system for the image. In particular, this assures that rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

The region of the input image is ignored, i.e., assumed to be the full rectangle of the image. The region of the resulting image is set to the transformed rectangle of the input image. If necessary, the resulting image is filled with zero (black) outside of the region of the original image.

Generally, transformed points will lie between pixel coordinates. Therefore, an appropriate interpolation scheme has to be used. The interpolation can also be used to avoid aliasing effects for scaled images. The quality and speed of the interpolation can be set by the parameter **Interpolation**:

- **none**  No interpolation: The gray value is determined from the nearest pixel’s gray value (possibly low quality, very fast).
- **constant**  Use equally weighted interpolation between adjacent pixels (medium quality and run time).
- **weighted**  Use Gaussian interpolation between adjacent pixels (best quality, slow).

In addition, the system parameter **‘int_zooming’** (see **SetSystem**) affects the accuracy of the transformation. If **‘int_zooming’** is set to **‘true’**, the transformation for byte, int2 and uint2 images is carried out internally using fixed point arithmetic, leading to much shorter execution times. However, the accuracy of the transformed gray values is smaller in this case. For byte images, the differences to the more accurate calculation (using **‘int_zooming’ = ‘false’**) is typically less than two gray levels. Correspondingly, for int2 and uint2 images, the gray value differences are less than 1/128 times the dynamic gray value range of the image, i.e., they can be as large as 512 gray levels if the entire dynamic range of 16 bit is used. For real images, the parameter **‘int_zooming’** does not affect the accuracy, since the internal calculations are always done using floating point arithmetic.

The size of the target image is specified by the parameters **Width** and **Height**. Note that the image is always clipped at the left and upper edge, i.e., all image parts that have negative coordinates after the transformation are clipped. If the affine transformation (in particular, the translation) is chosen appropriately, a part of the image can be transformed as well as cropped in one call. This is useful, for example, when using the variation model (see **CompareVariationModel**), because with this mechanism only the parts of the image that should be examined, are transformed.
CHAPTER 3. FILTER

Attention

The region of the input image is ignored.

Parameter

▷ **Image** (input iconic) .... (multichannel-)image(-array) ↦ **HImageX** / **IObjectX** ( byte, int2, uint2, real )
Input image.

▷ **ImageAffinTrans** (output iconic) .... (multichannel-)image(-array) ↦ **HImageX** / **HUntypedObjectX** ( byte, int2, uint2, real )
Transformed image.

▷ **HomMat2D** (input control) ................. affine2d ↦ VARIANT / **HHomMat2dX** ( real )
Input transformation matrix.

Number of elements : 6

▷ **Interpolation** (input control) ................. string ↦ String / VARIANT
Type of interpolation.

Default Value : ‘constant’

List of values: Interpolation ∈ {‘none’, ‘constant’, ‘weighted’}

▷ **Width** (input control) ......... extent.x ↦ long / VARIANT
Width of the output image.

Default Value : 640

Suggested values : Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}

▷ **Height** (input control) ......... extent.y ↦ long / VARIANT
Height of the output image.

Default Value : 480

Suggested values : Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}

Result

**AffineTransImageSize** returns TRUE if all parameter values are correct. If the input is empty the behaviour can be set via **SetSystem(::{'noObjectResult',<Result>:})**. If necessary, an exception handling is raised.

Parallelization Information

**AffineTransImageSize** is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors

HomMat2dIdentity, HomMat2dTranslate, HomMat2dRotate, HomMat2dScale

See also

SetPartStyle

Alternatives

**AffineTransImage**, **ZoomImageSize**, **ZoomImageFactor**, **MirrorImage**, **RotateImage**, **AffineTransRegion**

Module

Image filters

[out] HImageX **ImageMirror** **HImageX.MirrorImage** ([in] String **Mode** )


**Mirror an image.**

**MirrorImage** reflects an image **Image** about one of three possible axes. If **Mode** is set to ‘row’, it is reflected about the horizontal axis, if **Mode** is set to ‘column’, about the vertical axis, and if **Mode** is set to ‘main’, about the main diagonal \( x = y \).
3.1. AFFINE-TRANSFORMATIONS

- **ImageMirror** (output iconic) ……… (multichannel-)image-array) \( \sim \) HImageX / HUntypedObjectX (byte, int2, uint2, int4, real)

Reflected image.

- **Mode** (input control) …………………………………………………………… string \( \sim \) String / VARIANT

Axis of reflection.

**Default Value** : ‘row’

**List of values** : Mode \( \in \) {‘row’, ‘column’, ‘main’}

--- Example ---

```
read_image(Image,’affe’)
disp_image(Image,WindowHandle)
mirror_image(Image,MirImage,’row’).
disp_image(MirImage,WindowHandle)
```

--- Parallelization Information ---

**MirrorImage** is reentrant and automatically parallelized (on tuple level, channel level).

--- See also ---

**RotateImage**, HomMat2dRotate

--- Alternatives ---

HomMat2dRotate, AffineTransImage, RotateImage

--- Module ---

Image filters

--- Transform an image to polar coordinates ---

**PolarTransImage** transforms an image in cartesian coordinates to an image in polar coordinates. The size of the resulting image is selected with **Width** and **Height**. **Width** determines the angular resolution, while **Height** determines the resolution of the radius. **Row** and **Column** determine the center of the polar coordinate system in the original image **ImageXY**. This point is mapped to the upper row of **ImagePolar**.

A point \((x’,y’)\) in the result image corresponds to the point \((x,y)\) in the original image in the following manner:

\[
x = y' \cos(2\pi (x' / \text{resultwidth})) + \text{Column} \quad y = y' \sin(2\pi (x' / \text{resultwidth})) + \text{Row}
\]

--- Parameter ---

- **ImageXY** (input iconic) ……… (multichannel-)image-array) \( \sim \) HImageX / IHObjectX (byte, int2, uint2)

Input image in cartesian coordinates.

- **ImagePolar** (output iconic) ……… (multichannel-)image-array) \( \sim \) HImageX / HUntypedObjectX (byte, int2, uint2)

Result image in polar coordinates.

- **Row** (input control) …………………………………………………………… point.y \( \sim \) long / VARIANT

Row coordinate of the center of the coordinate system.

**Default Value** : 100

**Suggested values** : Row \( \in \) \{0, 10, 100, 200\}

**Typical range of values** : 0 \( \leq \) Row \( \leq \) 0

**Minimum Increment** : 1

**Recommended Increment** : 1

---

HALCON 6.1.4
\textbf{Column} (input control) \hspace{1cm} \text{point.x} \rightarrow \text{long / VARIANT}

Column coordinate of the center of the coordinate system.

\textbf{Default Value} : 100
\textbf{Suggested values} : Column \in \{0, 10, 100, 200\}
\textbf{Typical range of values} : 0 \leq \text{Column} \leq 0
\textbf{Minimum Increment} : 1
\textbf{Recommended Increment} : 1

\textbf{Width} (input control) \hspace{1cm} \text{extent.x} \rightarrow \text{long / VARIANT}

Width of the result image.

\textbf{Default Value} : 314
\textbf{Suggested values} : Width \in \{100, 200, 157, 314, 512\}
\textbf{Typical range of values} : 2 \leq \text{Width} \leq 2
\textbf{Minimum Increment} : 1
\textbf{Recommended Increment} : 10

\textbf{Height} (input control) \hspace{1cm} \text{extent.y} \rightarrow \text{long / VARIANT}

Height of the result image.

\textbf{Default Value} : 200
\textbf{Suggested values} : Height \in \{100, 128, 256, 512\}
\textbf{Typical range of values} : 2 \leq \text{Height} \leq 2
\textbf{Minimum Increment} : 1
\textbf{Recommended Increment} : 10

\begin{verbatim}
read_image(Image,'affe')
disp_image(Image,WindowHandle)
polar_trans_image(Image,PolarImage,100,100,314,200).
disp_image(PolarImage,WindowHandle)
\end{verbatim}

\textbf{Parallelization Information}

\textbf{PolarTransImage} is reentrant and automatically parallelized (on tuple level, channel level).

\textbf{Alternatives}

\textbf{AffineTransImage}

\textbf{Module}

\textbf{Image filters}

\begin{verbatim}
\end{verbatim}

\textit{Rotate an image about its center.}

\textit{RotateImage} rotates the image \textit{Image} counterclockwise by \textit{Phi} degrees about its center. This operator is much faster if \textit{Phi} is a multiple of 90 degrees than the general operator \textit{AffineTransImage}. For rotations by 90, 180, and 270 degrees, the region is rotated accordingly. For all other rotations the region is set to the maximum region, i.e., to the extent of the resulting image. The effect of the parameter \textit{Interpolation} is the same as in \textit{AffineTransImage}. It is ignored for rotations by 90, 180, and 270 degrees. The size of the resulting image is the same as that of the input image, with the exception of rotations by 90 and 270 degrees, where the width and height will be exchanged.

\textbf{Attention}

The angle \textit{Phi} is given in degrees, not in radians.
3.1. AFFINE-TRANSFORMATIONS

Parameter

- **Image** (input iconic) ............ (multichannel-)image(-array) $\sim$ **HImageX / HIObjectX** (byte, int2, uint2)
  Input image.
- **ImageRotate** (output iconic) ........ (multichannel-)image(-array) $\sim$ **HImageX / HUntypedObjectX** (byte, int2, uint2)
  Rotated image.
- **Phi** (input control) ............................. angle.deg $\sim$ **VARIANT** (integer, real)
  Rotation angle.
  Default Value: 90
  Suggested values: $\Phi \in \{90, 180, 270\}$
  Typical range of values: $0 \leq \Phi \leq 0$
  Minimum Increment: 0.001
  Recommended Increment: 0.2
- **Interpolation** (input control) ....................... string $\sim$ **String / VARIANT**
  Type of interpolation.
  Default Value: 'constant'
  List of values: Interpolation $\in \{\text{none}, \text{constant}, \text{weighted}\}$

Example

```plaintext
read_image(Image,'affe')
disp_image(Image,WindowHandle)
rotate_image(ImageRotImage,270).
disp_image(RotImage,WindowHandle)
```

Parallelization Information

**RotateImage** is reentrant and automatically parallelized (on tuple level, channel level).

See also

MirrorImage

Alternatives

HomMat2dRotate,AffineTransImage

Module

Image filters

```
[out] HImageX ImageZoomed **HImageX.ZoomImageFactor**
([in] double ScaleWidth, [in] double ScaleHeight, [in] String Interpolation )

void **HOperatorSetX.ZoomImageFactor** ([in] HIObjectX Image,
[out] HUntypedObjectX ImageZoomed, [in] VARIANT ScaleWidth,
[in] VARIANT ScaleHeight, [in] VARIANT Interpolation )
```

Zoom an image by a given factor.

**ZoomImageFactor** scales the image **Image** by a factor of **ScaleWidth** in width and a factor **ScaleHeight** in height. The parameter **Interpolation** determines the type of interpolation used (see **AffineTransImage**).

Parameter

- **Image** (input iconic) ............ (multichannel-)image(-array) $\sim$ **HImageX / HIObjectX** (byte, int2, uint2, real)
  Input image.
- **ImageZoomed** (output iconic) ........ (multichannel-)image(-array) $\sim$ **HImageX / HUntypedObjectX** (byte, int2, uint2, real)
  Scaled image.
CHAPTER 3. FILTER

Scale Width (input control) .............................................. extent.x \sim double / VARIANT
Scale factor for the width of the image.
Default Value: 0.5
Suggested values: ScaleWidth \in \{0.25, 0.5, 1.5, 2.0\}
Typical range of values: 0.001 \leq \text{ScaleWidth} \leq 0.001
Minimum Increment: 0.001
Recommended Increment: 0.1

Scale Height (input control) .......................................... extent.y \sim double / VARIANT
Scale factor for the height of the image.
Default Value: 0.5
Suggested values: ScaleHeight \in \{0.25, 0.5, 1.5, 2.0\}
Typical range of values: 0.001 \leq \text{ScaleHeight} \leq 0.001
Minimum Increment: 0.001
Recommended Increment: 0.1

Interpolation (input control) ...................................... string \sim String / VARIANT
Type of interpolation.
Default Value: 'constant'
List of values: Interpolation \in \{'none', 'constant', 'weighted'\}

Example

read_image(Image,'affe')
disp_image(Image,WindowHandle)
zoom_image_factor(Image,ZooImage,0,0.5,0.5).
disp_image(ZooImage,WindowHandle)

Parallelization Information
ZoomImageFactor is reentrant and automatically parallelized (on tuple level, channel level).

See also
HomMat2dScale, AffineTransImage

Alternatives
ZoomImageFactor, AffineTransImage, HomMat2dScale

Module

Image filters

\[
\text{HImageX.ZoomImageSize} ([\text{in}] \text{long Width, [\text{in}] long Height, [\text{in}] String Interpolation })
\]
\[
\text{HOperatorSetX.ZoomImageSize} ([\text{in}] \text{IOBJECTX Image, [\text{out}] HUntypedObjectX ImageZoom, [\text{in}] VARIANT Width, [\text{in}] VARIANT Height, [\text{in}] VARIANT Interpolation })
\]

Zoom an image to a given size.
ZoomImageSize scales the image Image to the size given by Width and Height. The parameter Interpolation determines the type of interpolation used (see AffineTransImage).

Parameter

\begin{itemize}
\item Image (input iconic) ...... (multichannel-)image(-array) \sim \text{HImageX / IObjectX ( byte, int2, uint2, real )}
Input image.
\item ImageZoom (output iconic) ...... (multichannel-)image(-array) \sim \text{HImageX / HUntypedObjectX ( byte, int2, uint2, real )}
Scaled image.
\end{itemize}
3.2. ARITHMETIC

- **Width** (input control) ........................................... $\text{extent.x} \sim \text{long / VARIANT}$
  - Width of the resulting image.
  - Default Value: 512
  - Suggested values: Width $\in \{128, 256, 512\}$
  - Typical range of values: $2 \leq \text{Width} \leq 2$
  - Minimum Increment: 1
  - Recommended Increment: 10

- **Height** (input control) ........................................... $\text{extent.y} \sim \text{long / VARIANT}$
  - Height of the resulting image.
  - Default Value: 512
  - Suggested values: Height $\in \{128, 256, 512\}$
  - Typical range of values: $2 \leq \text{Height} \leq 2$
  - Minimum Increment: 1
  - Recommended Increment: 10

- **Interpolation** (input control) ................................. $\text{string} \sim \text{String / VARIANT}$
  - Type of interpolation.
  - Default Value: 'constant'
  - List of values: Interpolation $\in \{\text{’none’, ’constant’, ’weighted’}\}$

Example:

```plaintext
read_image(Image,’affe’)
disp_image(Image,WindowHandle)
zoom_image_size(Image,ZooImage,0,200,200).
disp_image(ZooImage,WindowHandle)
```

---

**Parallelization Information**

`ZoomImageSize` is reentrant and automatically parallelized (on tuple level, channel level).

See also

- `HomMat2dScale`, `AffineTransImage`

**Alternatives**

- `ZoomImageFactor`, `AffineTransImage`, `HomMat2dScale`

**Module**

Image filters

### 3.2 Arithmetic

```plaintext
[out] HImageX ImageAbs HImageX.AbsImage ( )
void HOperatorSetX.AbsImage ([in] IObjectX Image,
[out] HUntypedObjectX ImageAbs )
```

Calculate the absolute value (modulus) of an image.

The operator `AbsImage` calculates the absolute gray values of images of any type and stores the result in `ImageAbs`. The power spectrum of complex images is calculated as a ‘real’ image. The operator `AbsImage` generates a logical copy of unsigned images.

**Parameter**

- **Image** (input iconic) ……… (multichannel-)image(-array) $\sim \text{HImageX / IObjectX}$ ( int1, int2, int4, real, complex )
  - Image(s) for which the absolute gray values are to be calculated.

- **ImageAbs** (output iconic) ……… (multichannel-)image(-array) $\sim \text{HImageX / HUntypedObjectX}$ ( int1, int2, int4, real )
  - Result image(s).
The operator `AbsImage` returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(\::'noObjectResult',<Result>::)`.

Parallelization Information

`AbsImage` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

ConvertImageType, PowerByte

Image filters

Add two images.

The operator `AddImage` adds two images. The gray values \((g_1, g_2)\) of the input images (\(Image1\) and \(Image2\)) are transformed as follows:

\[
g' := (g_1 + g_2) \times Mult + Add
\]

If an overflow or an underflow occurs the values are clipped. This is not the case with int2 images if \(Mult\) is equal to 1 and \(Add\) is equal to 0. To reduce the runtime the underflow and overflow check is skipped. The resulting image is stored in \(ImageResult\).

It is possible to add byte images with int2, uint2 or int4 images and to add int4 to int2 or uint2 images. In this case the result will be of type int2 or int4 respectively.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Please note that the runtime of the operator varies with different control parameters. For frequently used combinations special optimizations were used.

Parameter

- **Image1** (input iconic) …… (multichannel-)image(-array) \(\sim HImageX / IHObjectX\) (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Image(s) 1.

- **Image2** (input iconic) …… (multichannel-)image(-array) \(\sim HImageX / IHObjectX\) (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Image(s) 2.

- **ImageResult** (output iconic) …… (multichannel-)image(-array) \(\sim HImageX / HUntypedObjectX\) (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Result image(s) by the addition.

- **Mult** (input control) …… …… number \(\sim\) VARIANT (integer, real)
  Factor for gray value adaption.

  **Default Value** : 0.5
  **Suggested values** : \(Mult \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 5.0\}\)
  **Typical range of values** : \(-255.0 \leq Mult \leq -255.0\)
  **Minimum Increment** : 0.001
  **Recommended Increment** : 0.1
3.2. ARITHMETIC

Add (input control) ................................................................. number \(\sim\) VARIANT (integer, real)

Value for gray value range adaption.

Default Value: 0
Suggested values: Add \(\in\) \{0, 64, 128, 255, 512\}
Typical range of values: \(-512.0 \leq \text{Add} \leq 512.0\)
Minimum Increment: 0.01
Recommended Increment: 1.0

Example

\[
\begin{align*}
\text{read\_image}(\text{Image0}, \text{"fabrik"}) \\
\text{disp\_image}(\text{Image0}, \text{WindowHandle}) \\
\text{read\_image}(\text{Image1}, \text{"Affe"}) \\
\text{disp\_image}(\text{Image1}, \text{WindowHandle}) \\
\text{add\_image}(\text{Image0}, \text{Image1}, \text{Result}, 2.0, 10.0) \\
\text{disp\_image}(\text{Result}, \text{WindowHandle}) \\
\end{align*}
\]

Result

The operator AddImage returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘noObjectResult’, <Result>:).

If necessary an exception handling is raised.

Parallelization Information

AddImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

SubImage, MultImage

Alternatives

SubImage, MultImage

Module

Image filters

Divide two images.

The operator DivImage divides two images. The gray values \((g_1, g_2)\) of the input images (Image1) are transformed as follows:

\[
g' := g_1/g_2 \ast \text{Mult} + \text{Add}
\]

If an overflow or an underflow occurs the values are clipped.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Parameter

\[
\begin{align*}
\text{Image1} (\text{input iconic}) & \ldots \ldots \text{(multichannel-image-array)} \sim \text{HImageX} / \text{HObjectX} \ (\text{byte, int1, int2, uint2, int4, real, complex}) \\
\text{Image2} (\text{input iconic}) & \ldots \ldots \text{(multichannel-image-array)} \sim \text{HImageX} / \text{HObjectX} \ (\text{byte, int1, int2, uint2, int4, real, complex}) \\
\end{align*}
\]
CHAPTER 3. FILTER

Result image(s) by the division.

**Mult** (input control) 
Factor for gray range adaption.

*Default Value*: 255

*Suggested values*: Mult ∈ \{0.1, 0.2, 0.5, 1.0, 2.0, 3.0, 10, 100, 500, 1000\}

*Typical range of values*: -1000 ≤ Mult ≤ -1000

*Minimum Increment*: 0.001

*Recommended Increment*: 1

**Add** (input control) 
Value for gray range adaption.

*Default Value*: 0

*Suggested values*: Add ∈ \{0.0, 128.0, 256.0, 1025\}

*Typical range of values*: -1000 ≤ Add ≤ -1000

*Minimum Increment*: 0.01

*Recommended Increment*: 1

Example

```c
read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
div_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)
```

The operator **DivImage** returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator **SetSystem**(__:‘noObjectResult’,<Result>__):

If necessary an exception handling is raised.

**Parallelization Information**

**DivImage** is *reentrant* and automatically *parallelized* (on *tuple level*, *channel level*, *domain level*).

See also **AddImage**, **SubImage**, **MultImage**

Alternatives

**AddImage**, **SubImage**, **MultImage**

Module

Image filters

[out] HImageX ImageInvert HImageX.InvertImage ( )

void HOperatorSetX.InvertImage ([in] IObjectX Image, [out] HUntypedObjectX ImageInvert )

Invert an image.

The operator **InvertImage** inverts the gray values of an image. For images of the ’byte’ and ’cyclic’ type the result is calculated as:

\[ g' = 255 - g \]

Images of the ’direction’ type are transformed by

\[ g' = (g + 90) \mod 180 \]
In the case of signed types the values are negated. The resulting image has the same pixel type as the input image. Several images can be processed in one call. An output image is generated for every input image.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic) ...... (multichannel-)image(-array) (\sim) <strong>HImageX / HObjectX</strong> (byte, direction, cyclic, int1, int2, uint2, int4, real)</td>
</tr>
<tr>
<td>Input image(s).</td>
</tr>
<tr>
<td><strong>ImageInvert</strong> (output iconic) ...... (multichannel-)image(-array) (\sim) <strong>HImageX / HUntypedObjectX</strong> (byte, direction, cyclic, int1, int2, uint2, int4, real)</td>
</tr>
<tr>
<td>Image(s) with inverted gray values.</td>
</tr>
</tbody>
</table>

**Example**

```
read_image(Orig,"fabrik")
invert_image(Orig,Invert)
disp_image(Invert,WindowHandle).
```

**Parallelization Information**

**InvertImage** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Successors**

Watersheds

ScaleImage, AddImage, SubImage

**Alternatives**

ScaleImage

**Module**

Image filters

```
[out] HImageX ImageMax
HImageX.MaxImage ([in] HImageX Image2)

void HOperatorSetX.MaxImage ([in] HObjectX Image1,
[in] HObjectX Image2, [out] HUntypedObjectX ImageMax)
```

Calculate the maximum of two images pixel by pixel.

**MaxImage** calculates the maximum of the images **Image1** and **Image2** (pixel by pixel). The result is stored in the image **ImageMax**. The resulting image has the same pixel type as the input image. If several (pairs of) images are processed in one call, every i-th image from **Image1** is compared to the i-th image from **Image2**. Thus the number of images in both input parameters must be the same. An output image is generated for every input pair.

**Attention**

The two input images must be of the same type and size.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image1</strong> (input iconic) ...... (multichannel-)image(-array) (\sim) <strong>HImageX / HObjectX</strong> (byte, int1, int2, uint2, int4, real, direction, cyclic)</td>
</tr>
<tr>
<td>Image(s) 1.</td>
</tr>
<tr>
<td><strong>Image2</strong> (input iconic) ...... (multichannel-)image(-array) (\sim) <strong>HImageX / HObjectX</strong> (byte, int1, int2, uint2, int4, real, direction, cyclic)</td>
</tr>
<tr>
<td>Image(s) 2.</td>
</tr>
<tr>
<td><strong>ImageMax</strong> (output iconic) ...... (multichannel-)image(-array) (\sim) <strong>HImageX / HUntypedObjectX</strong> (byte, int1, int2, uint2, int4, real, direction, cyclic)</td>
</tr>
<tr>
<td>Result image(s) by the maximization.</td>
</tr>
</tbody>
</table>
read_image(Bild1,"affe")
read_image(Bild2,"fabrik")
max_image(Bild1,Bild2,Max)
disp_image(Max,WindowHandle)

If the parameter values are correct the operator MaxImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘‘noObjectResult’,<Result>):
If necessary an exception handling is raised.

MaxImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

MinImage

Alternatives

MaxImage

Module

Image filters

[out] HImageX ImageMin
HImageX.MinImage ([in] HImageX Image2 )

void HOperatorSetX.MinImage ([in] IHObjectX Image1,
[in] IHObjectX Image2, [out] HUntypedObjectX ImageMin )

Calculate the minimum of two images pixel by pixel.

The operator MinImage determines the minimum (pixel by pixel) of the images Image1 and Image2. The result is stored in the image ImageMin. The resulting image has the same pixel type as the input image. If several (pairs of) images are processed in one call, every i-th image from Image1 is compared to the i-th image from Image2. Thus the number of images in both input parameters must be the same. An output image is generated for every input pair.

Parameter

▷ Image1 (input iconic) …… (multichannel-)image(-array) ~ HImageX / IHObjectX ( byte, int1, int2, uint2, int4, real, direction, cyclic )

   Image(s) 1.

▷ Image2 (input iconic) …… (multichannel-)image(-array) ~ HImageX / IHObjectX ( byte, int1, int2, uint2, int4, real, direction, cyclic )

   Image(s) 2.

▷ ImageMin (output iconic) …… (multichannel-)image(-array) ~ HImageX / HUntypedObjectX ( byte, int1, int2, uint2, int4, real, direction, cyclic )

   Result image(s) by the minimization.

Result

If the parameter values are correct the operator MinImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(‘‘noObjectResult’,<Result>):
If necessary an exception handling is raised.

MinImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

MaxImage, MinImage

Alternatives

GrayErosion

HALCON/COM Reference Manual, 2005-2-1
Multiply two images.

\[g' := g_1 \times g_2 \times \text{Mult} + \text{Add}\]

If an overflow or an underflow occurs the values are clipped.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

**Parameter**

- **Image1** (input iconic) ...... (multichannel-image-array) \(\sim\) HImageX / IHOBJECTX (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Image(s) 1.

- **Image2** (input iconic) ...... (multichannel-image-array) \(\sim\) HImageX / IHOBJECTX (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Image(s) 2.

- **ImageResult** (output iconic) ...... (multichannel-image-array) \(\sim\) HImageX / HUntypedOBJECTX (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)

  Result image(s) by the product.

- **Mult** (input control) ........................................ number \(\sim\) VARIANT (integer, real)
  
  Factor for gray range adaption.

  **Default Value**: 0.005

  **Suggested values**: Mult \(\in\) \{0.001, 0.01, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0\}

  **Typical range of values**: -255.0 \(\leq\) Mult \(\leq\) -255.0

  **Minimum Increment**: 0.001

  **Recommended Increment**: 0.1

- **Add** (input control) ........................................ number \(\sim\) VARIANT (integer, real)

  Value for gray range adaption.

  **Default Value**: 0

  **Suggested values**: Add \(\in\) \{0.0, 128.0, 256.0\}

  **Typical range of values**: -512.0 \(\leq\) Add \(\leq\) -512.0

  **Minimum Increment**: 0.01

  **Recommended Increment**: 1.0

**Example**

```cpp
read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
mult_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)
```

**Result**

The operator **MultImage** returns the value TRUE if the parameters are correct. The behavior in case of empty
input (no input images available) is set via the operator `SetSystem(::’noObjectResult’,<Result>:)`.

If necessary an exception handling is raised.

-- Parallelization Information --
**MultImage** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also **AddImage, SubImage, DivImage**

-- Alternatives --

**AddImage, SubImage, DivImage**

**Module**

Image filters

```plaintext

```

Scale the gray values of an image.

The operator **ScaleImage** scales the input images (**Image**) by the following transformation:

\[
g' := g \times \text{Mult} + \text{Add}
\]

If an overflow or an underflow occurs the values are clipped.

This operator can be applied, e.g., to map the gray values of an image, i.e., the interval \([GMin,GMax]\), to the maximum range \([0:255]\). For this, the parameters are chosen as follows:

\[
\text{Mult} = \frac{255}{GMax - GMin} \quad \text{Add} = -\text{Mult} \times GMin
\]

The values for \(GMin\) and \(GMax\) can be determined, e.g., with the operator **MinMaxGray**.

**Parameter**

- **Image** (input iconic) …… (multichannel-)image(-array) \(\sim\) **HImageX / IHObjectX** (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)

  Image(s) whose gray values are to be scaled.

- **ImageScaled** (output iconic) …… (multichannel-)image(-array) \(\sim\) **HImageX / HUntypedObjectX** (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)

  Result image(s) by the scale.

- **Mult** (input control) ………………………………………………………………………………………………………. number \(\sim\) **VARIANT** (integer, real)

  Scale factor.

  Default Value : 0.01

  Suggested values : \(\text{Mult} \in \{0.001, 0.003, 0.005, 0.008, 0.01, 0.02, 0.03, 0.05, 0.08, 0.1, 0.5, 1.0\}\)

  Typical range of values : \(-255.0 \leq \text{Mult} \leq -255.0\)

  Minimum Increment : 0.001

  Recommended Increment : 0.1

- **Add** (input control) ………………………………………………………………………………………………………… number \(\sim\) **VARIANT** (integer, real)

  Offset.

  Default Value : 0

  Suggested values : \(\text{Add} \in \{0, 10, 50, 100, 200, 500\}\)

  Typical range of values : \(-512.0 \leq \text{Add} \leq -512.0\)

  Minimum Increment : 0.01

  Recommended Increment : 1.0
Example

/* Complement of the gray values: */
scale_image(Bild, Invert, -1.0, 255.0).

Result

The operator \texttt{ScaleImage} returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem::’noObjectResult’,<Result>:}) Otherwise an exception treatment is carried out.

Parallelization Information

\texttt{ScaleImage} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

\texttt{MinMaxGray}

See also

\texttt{MinMaxGray}

Alternatives

\texttt{MultImage, AddImage, SubImage}

Module

\texttt{Image filters}


Subtract two images.

The operator \texttt{SubImage} subtracts two images. The gray values \((g_1, g_2)\) of the input images \texttt{(ImageMinuend} and \texttt{ImageSubtrahend)} are transformed as follows:

\[ g' := (g_1 - g_2) \ast \text{Mult} + \text{Add} \]

If an overflow or an underflow occurs the values are clipped.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

Parameter

- \texttt{ImageMinuend} (input iconic) ...... (multichannel-)image(-array) \sim \texttt{HImageX / IObjectX} (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Minuend(s).

- \texttt{ImageSubtrahend} (input iconic) ...... (multichannel-)image(-array) \sim \texttt{HImageX / IObjectX} (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Subtrahend(s).

- \texttt{ImageSub} (output iconic) ...... (multichannel-)image(-array) \sim \texttt{HImageX / HUntypedObjectX} (byte, int1, int2, uint2, int4, real, direction, cyclic, complex)
  
  Result image(s) by the subtraction.

HALCON 6.1.4
Mult (input control) .................................................. number  ~ VARIANT ( integer, real )
Correction factor.
Default Value : 1.0
Suggested values : Mult ∈ {0.0, 1.0, 2.0, 3.0, 4.0}
Typical range of values : -255.0 ≤ Mult ≤ -255.0
Minimum Increment : 0.001
Recommended Increment : 0.1

Add (input control) .................................................. number  ~ VARIANT ( integer, real )
Correction value.
Default Value : 128.0
Suggested values : Add ∈ {0.0, 128.0, 256.0}
Typical range of values : -512.0 ≤ Add ≤ -512.0
Minimum Increment : 0.01
Recommended Increment : 1.0

Example

read_image(Image0,"fabrik")
disp_image(Image0,WindowHandle)
read_image(Image1,"Affe")
disp_image(Image1,WindowHandle)
sub_image(Image0,Image1,Result,2.0,10.0)
disp_image(Result,WindowHandle)

Result

The operator SubImage returns the value TRUE if the parameters are correct. The behavior in case of empty
input (no input images available) is set via the operator SetSystem(::'noObjectResult',<Result>):
If necessary an exception handling is raised.

Parallelization Information
SubImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors
DualThreshold

See also
AddImage, MultImage, DynThreshold, CheckDifference

Alternatives
MultImage, AddImage, SubImage

Module

Image filters

3.3 Bit

[out]  HImageX ImageAnd HImageX.BitAnd ([in]  HImageX Image2 )
void HOperatorSetX.BitAnd ([in]  IObjectX Image1, [in]  IObjectX Image2,
[out]  HUntypedObjectX ImageAnd )

Bit-by-bit AND of all pixels of the input images.

The operator BitAnd calculates the “and” of all pixels of the input images bit by bit. The semantics of the
“and” operation corresponds to that of C for the respective types (signed char, unsigned char, short, unsigned short,
int/long). The images must have the same size and pixel type. The pixels within the definition range of the image
in the first parameter are processed.
Several images can be processed in one call. In this case both input parameters contain the same number of images
which are then processed in pairs. An output image is generated for every pair.
3.3. BIT

Parameter

- **Image1** (input iconic) .... (multichannel-)image(-array) \( \sim \) HImageX / IObjectX (byte, direction, cyclic, int1, uint2, int4)
  - Input image(s) 1.

- **Image2** (input iconic) .... (multichannel-)image(-array) \( \sim \) HImageX / IObjectX (byte, direction, cyclic, int1, uint2, int4)
  - Input image(s) 2.

- **ImageAnd** (output iconic) .... (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4)
  - Result image(s) by AND-operation.

Example

read_image(Image0,’affe’)
disp_image(Image0,WindowHandle)
read_image(Image1,’fabrik’)
disp_image(Image1,WindowHandle)
bit_and(Image0,Image1,ImageBitA)
disp_image(ImageBitA,WindowHandle).

Result

If the images are correct (type and number) the operator **BitAnd** returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** \( (::’noObjectResult’,<Result>::) \) If necessary an exception handling is raised.

Parallelization Information

**BitAnd** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

- BitMask, AddImage, MaxImage

Alternatives

- BitMask, AddImage, MaxImage

Module

Image filters

Left shift of all pixels of the image.

The operator **BitLshift** calculates a “left shift” of all pixels of the input image bit by bit. The semantics of the “left shift” operation corresponds to that of C ("<<") for the respective types (signed char, unsigned char, short, unsigned short, int/long). If an overflow occurs the result is limited to the maximum value of the respective pixel type. Only the pixels within the definition range of the image are processed.

Several images can be processed in one call. An output image is generated for every input image.

Parameter

- **Image** (input iconic) .... (multichannel-)image(-array) \( \sim \) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4)
  - Input image(s).

- **ImageLShift** (output iconic) .... (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4)
  - Result image(s) by shift operation.
CHAPTER 3. FILTER

- **Shift** (input control) .......................................................... integer $\sim$ long / VARIANT
  - Default Value : 3
  - Suggested values : $\text{Shift} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31\}$
  - Typical range of values : $0 \leq \text{Shift} \leq 0$
  - Minimum Increment : 1
  - Recommended Increment : 1
  - Restriction : $((\text{Shift} \geq 1) \land (\text{Shift} \leq 31))$
  
  If the images are correct (type) and if \text{Shift} has a valid value the operator \text{BitLshift} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \text{SetSystem} (\text{::'noObjectResult'},<\text{Result}>:) If necessary an exception handling is raised.

  **Parallelization Information**
  \text{BitLshift} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

  **Alternatives**
  \text{BitRshift}

  **See also**
  \text{ScaleImage}

  **Module**
  Image filters

---

- **BitMask** (input control) ...................................................... integer $\sim$ long / VARIANT
  - Default Value : 128
  - Suggested values : $\text{BitMask} \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$
  - List of values : $\text{BitMask} \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$

  Logical “AND” of each pixel using a bit mask.

  The operator \text{BitMask} carries out an “and” operation of each pixel with a fixed mask. The semantics of the “and” operation corresponds to that of C for the respective types (signed char, unsigned char, unsigned short, short, int/long). Only the pixels within the definition range of the image are processed.

  Several images can be processed in one call. An output image is generated for every input image.

  **Parameter**
  - **Image** (input iconic) .... (multichannel-)image(-array) $\sim$ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4 )
    - Input image(s).
  - **ImageMask** (output iconic) .... (multichannel-)image(-array) $\sim$ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4 )
    - Result image(s) by combination with mask.
  - **BitMask** (input control) .................................................. integer $\sim$ long / VARIANT
    - Bit field
    - Default Value : 128
    - Suggested values : $\text{BitMask} \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$
    - List of values : $\text{BitMask} \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096\}$

  If the images are correct (type) the operator \text{BitMask} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \text{SetSystem} (\text{::'noObjectResult'},<\text{Result}>:) If necessary an exception handling is raised.

  **Parallelization Information**
  \text{BitMask} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

  **Possible Successors**
  Threshold, BitOr

  **See also**
  BitAnd, BitLshift

HALCON/COM Reference Manual, 2005-2-1
Alternatives

BitSlice

Module

Image filters

[out] HImageX ImageNot HImageX.BitNot ( )

void HOperatorSetX.BitNot ([in] IHObjectX Image,
[out] HUntypedObjectX ImageNot )

Complement all bits of the pixels.

The operator BitNot calculates the “complement” of all pixels of the input image bit by bit. The semantics of the “complement” operation corresponds to that of C (“∼”) for the respective types (signed char, unsigned char, short, unsigned short, int/long). Only the pixels within the definition range of the image are processed. Several images can be processed in one call. An output image is generated for every input image.

Parameter

▷ Image (input iconic) ...... (multichannel-)image(-array) ~ HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4 )

Input image(s).

▷ ImageNot (output iconic) ...... (multichannel-)image(-array) ~ HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4 )

Result image(s) by complement operation.

Example

read_image(Image0,’affe’)
disp_image(Image0,WindowHandle)
bitor(Image0,ImageBitN)
disp_image(ImageBitN,WindowHandle).

Result

If the images are correct (type) the operator BitNot returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem(::’noObjectResult’,<Result>:)

If necessary an exception handling is raised.

Parallelization Information

BitNot is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

BitSlice, BitMask

Alternatives

BitOr, BitAnd, AddImage

Module

Image filters

[out] HImageX ImageOr HImageX.BitOr ([in] HImageX Image2 )

void HOperatorSetX.BitOr ([in] IHObjectX Image1, [in] IHObjectX Image2,
[out] HUntypedObjectX ImageOr )

Bit-by-bit OR of all pixels of the input images.

The operator BitOr calculates the “or” of all pixels of the input images bit by bit. The semantics of the “or” operation corresponds to that of C for the respective types (signed char, unsigned char, short, unsigned short, int/long). The images must have the same size and pixel type. The pixels within the definition range of the image in the first parameter are processed.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.
CHAPTER 3. FILTER

Parameter

▷ **Image1** (input iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Input image(s) 1.

▷ **Image2** (input iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Input image(s) 2.

▷ **ImageOr** (output iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Result image(s) by OR-operation.

Example

read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
read_image(Image1,'fabrik')
disp_image(Image1,WindowHandle)
bit_or(Image0,Image1,ImageBitO)
disp_image(ImageBitO,WindowHandle).

Result

If the images are correct (type and number) the operator **BitOr** returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (\(::\)’noObjectResult’,<Result>::) If necessary an exception handling is raised.

Parallelization Information

**BitOr** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

BitXor, BitAnd

Alternatives

BitAnd, AddImage

Module

Image filters

```
[out] HImageX ImageRShift HImageX.BitRshift ([in] long Shift )
```

Right shift of all pixels of the image.

The operator **BitRshift** calculates a “right shift” of all pixels of the input image bit by bit. The semantics of the “right shift” operation corresponds to that of C (“\(\gg\)”) for the respective types (signed char, unsigned char, short, unsigned short, int/long). Only the pixels within the definition range of the image are processed.

Several images can be processed in one call. An output image is generated for every input image.

Parameter

▷ **Image** (input iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Input image(s).

▷ **ImageRShift** (output iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Result image(s) by shift operation.
3.3. BIT

- **Shift** (input control) ............................................................. integer  \(\rightarrow\) long / VARIANT
  
  Default Value: 3
  
  Suggested values: \(\text{Shift} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31\}\)

  Typical range of values: \(0 \leq \text{Shift} \leq 0\)

  Minimum Increment: 1

  Recommended Increment: 1

  Restriction: \(((\text{Shift} \geq 1) \land (\text{Shift} \leq 31))\)

If the images are correct (type) and **Shift** has a valid value the operator **BitRshift** returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (\(\text{::'noObjectResult'},\langle\text{Result}\rangle\)) If necessary an exception handling is raised.

Parallelization Information

**BitRshift** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

Possible Successors

Threshold, BitOr

---

**Image** (input iconic) .... (multichannel-)image(-array)  \(\rightarrow\) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Input image(s).

**ImageSlice** (output iconic) .... (multichannel-)image(-array)  \(\rightarrow\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

Result image(s) by extraction.

**Bit** (input control) ............................................................. integer  \(\rightarrow\) long / VARIANT

Bit to be selected.

Default Value: 8

Suggested values: \(\text{Bit} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 32\}\)

Typical range of values: \(1 \leq \text{Bit} \leq 1\)

Minimum Increment: 1

Recommended Increment: 1

Restriction: \(((\text{Bit} \geq 1) \land (\text{Bit} \leq 32))\)

If the images are correct (type) and **Bit** has a valid value, the operator **BitSlice** returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (\(\text{::'noObjectResult'},\langle\text{Result}\rangle\)) If necessary an exception handling is raised.

Parallelization Information

**BitSlice** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

Possible Successors

Threshold, BitOr

---

HALCON 6.1.4
Bit-by-bit XOR of all pixels of the input images.

The operator **BitXor** calculates the “xor” of all pixels of the input images bit by bit. The semantics of the “xor” operation corresponds to that of C for the respective types (signed char, unsigned char, short, unsigned short, int/long). The images must have the same size and pixel type. The pixels within the definition range of the image in the first parameter are processed.

Several images can be processed in one call. In this case both input parameters contain the same number of images which are then processed in pairs. An output image is generated for every pair.

### Parameter

- **Image1** (input iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

  Input image(s) 1.

- **Image2** (input iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

  Input image(s) 2.

- **ImageXor** (output iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4)

  Result image(s) by XOR-operation.

### Example

```c
read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
read_image(Image1,'fabrik')
disp_image(Image1,WindowHandle)
bit_xor(Image0,Image1,ImageBitX)
disp_image(ImageBitX,WindowHandle).
```

### Result

If the parameter values are correct the operator **BitXor** returns the value TRUE. The behavior in case of empty input (no input images available) can be determined by the operator **SetSystem** (::'noObjectResult',<Result>;) If necessary an exception handling is raised.

### Parallelization Information

**BitXor** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

**See also**

**BitAnd, BitLshift**

---

**Alternatives**

**BitOr, BitAnd**

---

**Module**

**Image filters**

---
3.4 Color

**HImageX** and **HImageX.CfaToRgb** ([in] String **CFAType**, [in] String **Interpolation**)  

Convert a single-channel color filter array image into an RGB image.

**CfaToRgb** converts a single-channel color filter array image **CFAImage** into an RGB image **RGBImage**. Color filter array images are typically generated by single-chip CCD cameras. The conversion from color filter array image to RGB image is typically done on the camera itself or is performed by the device driver of the frame grabber that is used to grab the image. In some cases, however, the device driver simply passes the color filter array image through unchanged. In this case, the corresponding HALCON frame grabber interface typically converts the image into an RGB image. Hence, the operator **CfaToRgb** is normally used if the images are not being grabbed using the HALCON frame grabber interface (GrabImage or GrabImageAsync), but are grabbed using function calls from the frame grabber SDK, and are passed to HALCON using **GenImage1** or **GenImage1Extern**.

In single-chip CCD cameras, a color filter array in front of the sensor provides (subsampled) color information. The most frequently used filter is the so-called Bayer filter. The color filter array has the following layout in this case:

```
  G B G B G B · · ·
  R G R G R G · · ·
  G B G B G B · · ·
  R G R G R G · · ·
  · · · · · · · · · ·
```

Each gray value of the input image **CFAImage** corresponds to the brightness of the pixel behind the corresponding color filter. Hence, in the above layout, the pixel (0,0) corresponds to a green color value, while the pixel (0,1) corresponds to a blue color value. The layout of the Bayer filter is completely determined by the first two elements of the first row of the image, and can be chosen with the parameter **CFAType**. In particular, this enables the correct conversion of color filter array images that have been cropped out of a larger image (e.g., using **CropPart** or **CropRectangle1**). The algorithm that is used to interpolate the RGB values is determined by the parameter **Interpolation**. Currently, the only possible choice is 'bilinear'.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CFAImage</strong></td>
<td>(input iconic) ......................... image ~ HImageX / IHOBJECTX (byte, uint2)</td>
</tr>
<tr>
<td><strong>RGBImage</strong></td>
<td>(output iconic) ....................... image ~ HImageX / HUNTYPEdOBJECTX (byte, uint2)</td>
</tr>
<tr>
<td><strong>CFAType</strong></td>
<td>(input control) ....................... string ~ String / VARIANT</td>
</tr>
<tr>
<td>Default Value:</td>
<td>'bayer_gb'</td>
</tr>
<tr>
<td>List of values:</td>
<td>{&quot;bayer_gb&quot;, &quot;bayer_gr&quot;, &quot;bayer_bg&quot;, &quot;bayer_rg&quot;}</td>
</tr>
<tr>
<td><strong>Interpolation</strong></td>
<td>(input control) ....................... string ~ String / VARIANT</td>
</tr>
<tr>
<td>Default Value:</td>
<td>'bilinear'</td>
</tr>
<tr>
<td>List of values:</td>
<td>{&quot;bilinear&quot;}</td>
</tr>
</tbody>
</table>

**Result**  

**CfaToRgb** returns TRUE if all parameters are correct. If the input is empty the behavior can be set via **SetSystem**(::'noObjectResult',<Result>). If necessary, an exception handling is raised.

**Parallelization Information**  

**CfaToRgb** is reentrant and automatically parallelized (on tuple level, domain level).
Transform an RGB image into a gray scale image.

Rgb1ToGray transforms an RGB image into a gray scale image. The three channels of the RGB image are passed as the first three channels of the input image. The image is transformed according to the following formula:

\[
k = 0.299r + 0.587g + 0.144b\]

**Parameter**

- **RGBImage** (input iconic) ..................... image(-array) \(\sim\) HImageX / IObjectX (byte, int2)
  Three-channel RBG image.
- **GrayImage** (output iconic) ..................... image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, int2)
  Gray scale image.

**Example**

```c
/* Transformation from rgb to gray */
read_image(Image,’patras’)
disp_color(Image,WindowHandle)
rgb1_to_gray(Image,GrayImage)
disp_image(GrayImage,WindowHandle).
```

**Parallelization Information**

Rgb1ToGray is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessors**

GenImage1Extern, GenImage1, GrabImage

**Possible Successors**

Decompose3

**See also**

TransFromRgb

**Module**

Image filters
\[ k = 0.299r + 0.587g + 0.144b. \]

**Parameter**

- **ImageRed** (input iconic) \( \mapsto \) image(-array) \( \sim HImageX / IObjectX \) (byte, int2)
  - Input image (red channel).
- **ImageGreen** (input iconic) \( \mapsto \) image(-array) \( \sim HImageX / IObjectX \) (byte, int2)
  - Input image (green channel).
- **ImageBlue** (input iconic) \( \mapsto \) image(-array) \( \sim HImageX / IObjectX \) (byte, int2)
  - Input image (blue channel).
- **ImageGray** (output iconic) \( \mapsto \) image(-array) \( \sim HImageX / HUntypedObjectX \) (byte, int2)
  - Gray scale image.

**Example**

```c
/* Transformation from rgb to gray */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image,Rimage,Gimage,Bimage)
rgb3_to_gray(Rimage,Gimage,Bimage,GrayImage)
disp_image(GrayImage,WindowHandle).
```

**Parallelization Information**

- **Rgb3ToGray** is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessors**

- **Decompose3**

**Alternatives**

- **Rgb1ToGray, TransFromRgb**

**Module**

**Image filters**

```
[out] HImageX ImageResult1 HImageX.TransFromRgb
```


Transform an image from the RGB color space to an arbitrary color space. **TransFromRgb** transforms an image from the RGB color space to an arbitrary color space (**ColorSpace**). The three channels of the image are passed as three separate images on input and output.

The following transformations are supported:

- **'yiq'**
  \[
  \begin{pmatrix}
  Y \\
  I \\
  Q
  \end{pmatrix} =
  \begin{pmatrix}
  0.299 & 0.587 & 0.144 \\
  0.595 & -0.276 & -0.333 \\
  0.209 & -0.522 & 0.287
  \end{pmatrix}
  \begin{pmatrix}
  R \\
  G \\
  B
  \end{pmatrix}
  \]

- **'yuv'**
  \[
  \begin{pmatrix}
  Y \\
  U \\
  V
  \end{pmatrix} =
  \begin{pmatrix}
  0.299 & 0.587 & 0.114 \\
  -0.147 & -0.289 & 0.436 \\
  0.615 & -0.515 & 0.100
  \end{pmatrix}
  \begin{pmatrix}
  R \\
  G \\
  B
  \end{pmatrix}
  \]

HALCON 6.1.4
'argyb' \[
\begin{pmatrix}
A \\
Rg \\
Yb
\end{pmatrix}
= 
\begin{pmatrix}
0.30 & 0.59 & 0.11 \\
0.50 & -0.50 & 0.00 \\
0.25 & 0.25 & -0.50
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

'ciexyz' \[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
= 
\begin{pmatrix}
0.476 & 0.299 & 0.175 \\
0.262 & 0.656 & 0.082 \\
0.020 & 0.161 & 0.909
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

'hls' \[
\begin{align*}
\text{min} &= \min(R,G,B) \\
\text{max} &= \max(R,G,B) \\
L &= (\text{min} + \text{max}) / 2 \\
\text{if} (\text{max} == \text{min}) \\
\quad H &= 0 \\
\quad S &= 0 \\
\text{else} \\
\quad \text{if} (L > 0.5) \\
\quad \quad S &= (\text{max} - \text{min}) / (2 - \text{max} - \text{min}) \\
\quad \text{else} \\
\quad \quad S &= (\text{max} - \text{min}) / (\text{max} + \text{min}) \\
\quad \text{fi} \\
\quad \text{if} (R == \text{max}) \\
\quad \quad H &= ((G - B) / (\text{max} - \text{min})) * 60 \\
\quad \text{elif} (G == \text{max}) \\
\quad \quad H &= (2 + (B - R) / (\text{max} - \text{min})) * 60 \\
\quad \text{elif} (B == \text{max}) \\
\quad \quad H &= (4 + (R - G) / (\text{max} - \text{min})) * 60 \\
\quad \text{fi} \\
\text{fi}
\end{align*}
\]

'hsi' \[
\begin{pmatrix}
M1 \\
M2 \\
I1
\end{pmatrix}
= 
\begin{pmatrix}
\begin{bmatrix}
\frac{2}{\sqrt{3}} \\
0 \\
\frac{-1}{\sqrt{3}}
\end{bmatrix} & \begin{bmatrix}
\frac{-1}{\sqrt{3}} \\
\frac{1}{\sqrt{3}} \\
\frac{-1}{\sqrt{3}}
\end{bmatrix}
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

\[
\begin{pmatrix}
H \\
S \\
I
\end{pmatrix}
= 
\begin{pmatrix}
\arctan\frac{M2}{\sqrt{M1^2 + M2^2}}
\end{pmatrix}
\]

'hsv' \[
\begin{align*}
\text{min} &= \min(R,G,B) \\
\text{max} &= \max(R,G,B) \\
V &= \text{max} \\
\text{if} (\text{max} == \text{min}) \\
\quad S &= 0 \\
\quad H &= 0 \\
\text{else} \\
\quad S &= (\text{max} - \text{min}) / \text{max} \\
\quad \text{if} (R == \text{max}) \\
\quad \quad H &= ((G - B) / (\text{max} - \text{min})) * 60 \\
\quad \text{elif} (G == \text{max}) \\
\quad \quad H &= (2 + (B - R) / (\text{max} - \text{min})) * 60 \\
\quad \text{elif} (B == \text{max}) \\
\quad \quad H &= (4 + (R - G) / (\text{max} - \text{min})) * 60 \\
\quad \text{fi} \\
\text{fi}
\end{align*}
\]

'ihs' \[
\begin{align*}
\text{min} &= \min(R,G,B) \\
\text{max} &= \max(R,G,B)
\end{align*}
\]
3.4. COLOR

\[
I = \frac{(R + G + B)}{3}
\]

if \((I == 0)\)

\[
H = 0
\]

\[
S = 1
\]

else

\[
S = 1 - \min / I
\]

if \((S == 0)\)

\[
H = 0
\]

else

\[
A = \frac{(R + R - G - B)}{2}
\]

\[
B = (R - G) \ast (R - G) + (R - B) \ast (G - B)
\]

\[
C = \sqrt{B}
\]

if \((C == 0)\)

\[
H = 0
\]

else

\[
H = \text{acos}(A / C)
\]

fi

if \((B > G)\)

\[
H = 2 \ast \pi - H
\]

fi

fi

'cielab'

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
= \begin{pmatrix}
0.476 & 0.299 & 0.175 \\
0.262 & 0.656 & 0.082 \\
0.020 & 0.161 & 0.909
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

\[
L = 116 \ast Y^{\frac{1}{3}} - 16
\]

\[
a = 500 \ast (X^{\frac{1}{3}} - Y^{\frac{1}{3}})
\]

\[
b = 200 \ast (Y^{\frac{1}{3}} - Z^{\frac{1}{3}})
\]

'ili2i3'

\[
\begin{pmatrix}
I1 \\
I2 \\
I3
\end{pmatrix}
= \begin{pmatrix}
0.333 & 0.333 & 0.333 \\
1.0 & 0.0 & -1.0 \\
-0.5 & 1.0 & -0.5
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

'ciexyz2'

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
= \begin{pmatrix}
0.620 & 0.170 & 0.180 \\
0.310 & 0.590 & 0.110 \\
0.000 & 0.066 & 1.020
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

'ciexyz3'

\[
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix}
= \begin{pmatrix}
0.618 & 0.177 & 0.205 \\
0.299 & 0.587 & 0.114 \\
0.000 & 0.056 & 0.944
\end{pmatrix}
\begin{pmatrix}
R \\
G \\
B
\end{pmatrix}
\]

If one of the following conditions is fulfilled, certain scalings are performed accordingly to the image type:

- Considering byte images, the domain of color space values is generally mapped to the full domain of [0..255]. The origin of signed values is shifted to 128.

- Hue values are represented by angles of [0..2\pi] and are coded for the particular image types differently:
  - byte-images map the angle domain on [0..255].
  - int4-images are coded in angle minutes [0..21600].
  - real-images are coded in radians [0..2\pi].

- int4-images map a domain of [0..1] of transformed values on [0..10000].
 CHAPTER 3. FILTER

Parameter

- **ImageRed** (input iconic) .................. image(-array)  ~ HImageX / IObjectX (byte, int4, real)
  Input image (red channel).
- **ImageGreen** (input iconic) ............... image(-array)  ~ HImageX / IObjectX (byte, int4, real)
  Input image (green channel).
- **ImageBlue** (input iconic) ................. image(-array)  ~ HImageX / IObjectX (byte, int4, real)
  Input image (blue channel).
- **ImageResult1** (output iconic) ............ image(-array)  ~ HImageX / HUntypedObjectX (byte, int4, real)
  Color-transformed output image (channel 1).
- **ImageResult2** (output iconic) ............ image(-array)  ~ HImageX / HUntypedObjectX (byte, int4, real)
  Color-transformed output image (channel 1).
- **ImageResult3** (output iconic) ............ image(-array)  ~ HImageX / HUntypedObjectX (byte, int4, real)
  Color-transformed output image (channel 1).
- **ColorSpace** (input control) .............. string  ~ String / VARIANT
  Color space of the output image.
  Default Value: 'hsv'
  List of values: ColorSpace ∈ {'cielab', 'hsv', 'hsi', 'yiq', 'yuv', 'argyb', 'ciexyz', 'ciexyz2', 'ciexyz3', 'hls', 'ihs', 'i1i2i3'}

Example
/* Transformation from rgb to hsv and conversely */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image,ImageR,ImageG,ImageB)
trans_from_rgb(ImageR,ImageG,ImageB,Image1,Image2,Image3,'hsv')
trans_to_rgb(Image1,Image2,Image3,ImageRed,ImageGreen,ImageBlue,'hsv')
compose3(ImageRed,ImageGreen,ImageBlue,Image)
disp_color(Image,WindowHandle).

Result
TransFromRgb returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via
SetSystem(‘noObjectResult’,<Result>:).
If necessary, an exception handling is raised.

Parallelization Information
TransFromRgb is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessors
Decompose3

Possible Successors
Compose3

See also
TransToRgb

Alternatives
RgbToGray, Rgb3ToGray

Module
Image filters

[out] HImageX ImageRed HImageX.TransToRgb ([in] HImageX ImageInput2,
in] HImageX ImageInput3, [out] HImageX ImageGreen, [out] HImageX ImageBlue,
in] String ColorSpace )

void HOperatorSetX.TransToRgb ([in] IObjectX ImageInput1,
in] IObjectX ImageInput2, [in] IObjectX ImageInput3,
out] HUntypedObjectX ImageRed, [out] HUntypedObjectX ImageGreen,

Transform an image from an arbitrary color space to the RGB color space.

HALCON/COM Reference Manual, 2005-2-1
TransToRgb transforms an image from an arbitrary color space (ColorSpace) to the RGB color space. The three channels of the image are passed as three separate images on input and output.

The following transformations are supported:

'yiq'

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
0.999 & 0.962 & 0.615 \\
0.949 & -0.220 & -0.732 \\
0.999 & -1.101 & 1.706
\end{bmatrix}\begin{bmatrix}
Y \\
I \\
Q
\end{bmatrix}
\]

'yuv'

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
1.0 & 0.0 & 1.140 \\
1.0 & -0.394 & -0.581 \\
1.0 & 2.032 & 0.0
\end{bmatrix}\begin{bmatrix}
Y \\
U \\
V
\end{bmatrix}
\]

'argyb'

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
1.00 & 1.29 & 0.22 \\
1.00 & -0.71 & 0.22 \\
1.00 & 0.29 & -1.78
\end{bmatrix}\begin{bmatrix}
A \\
Rg \\
Yb
\end{bmatrix}
\]

'ciexyz'

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
2.750 & -1.149 & -0.426 \\
-1.118 & 2.026 & 0.033 \\
0.138 & -0.333 & 1.104
\end{bmatrix}\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]

'hls'

\[
\begin{align*}
Hi &= \text{integer}(H \times 6) \\
Hf &= \text{fraction}(H \times 6) \\
\text{if } (L \leq 0.5) &\text{ max } = L \times (S + 1) \\
\text{else } &\text{ max } = L + S - (L \times S) \\
\text{fi } &\text{ min } = 2 \times L - \text{max} \\
\text{if } (S == 0) &R = L \\
&G = L \\
&B = L \\
\text{else } &\text{if } (Hi == 0) \\
&R = \text{max} \\
&G = \text{min} + Hf \times (\text{max} - \text{min}) \\
&B = \text{min} \\
\text{elif } (Hi == 1) &R = \text{min} + (1 - Hf) \times (\text{max} - \text{min}) \\
&G = \text{max} \\
&B = \text{min} \\
\text{elif } (Hi == 2) &R = \text{min} \\
&G = \text{max} \\
&B = \text{min} + Hf \times (\text{max} - \text{min}) \\
\text{elif } (Hi == 3) &R = \text{min} \\
&G = \text{min} + (1 - Hf) \times (\text{max} - \text{min}) \\
&B = \text{max} \\
\text{elif } (Hi == 4) &R = \text{min} + Hf \times (\text{max} - \text{min}) \\
&G = \text{min} \\
&B = \text{max}
\end{align*}
\]
elif (Hi == 5)
  R = max
  G = min
  B = min + (1 - Hf) * (max - min)
fi

'hsi'

M1 = S * \sin H
M2 = S * \cos H
I1 = \frac{I}{\sqrt{3}}

\begin{pmatrix}
  R \\
  G \\
  B
\end{pmatrix} = \begin{pmatrix}
  \frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}} \\
  \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\
  \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}}
\end{pmatrix} \begin{pmatrix}
  M1 \\
  M2 \\
  I1
\end{pmatrix}

'hsv'

if (S == 0)
  R = V
  G = V
  B = V
else
  Hi = integer(H)
  Hf = fraction(H)
  if (Hi == 0)
    R = V
    G = V * (1 - (S * (1 - Hf)))
    B = V * (1 - S)
  elif (Hi == 1)
    R = V * (1 - (S * Hf))
    G = V
    B = V * (1 - S)
  elif (Hi == 2)
    R = V * (1 - S)
    G = V
    B = V * (1 - (S * (1 - Hf)))
  elif (Hi == 3)
    R = V * (1 - S)
    G = V * (1 - (S * Hf))
    B = V
  elif (Hi == 4)
    R = V * (1 - (S * (1 - Hf)))
    G = V * (1 - S)
    B = V
  elif (Hi == 5)
    R = V
    G = V * (1 - S)
    B = V * (1 - (S * Hf))
fi
fi

If one of the following conditions is fulfilled, certain scalings are expected accordingly to the image type:

- Considering byte images, the domain of color space values is generally expected to be spread to the full domain of $[0..255]$. The origin of signed values must be shifted to 128.
- Hue values are represented by angles of $[0..2\pi]$ and are coded for the particular image types differently:
  - byte-images map the angle domain on $[0..255]$.
  - int4-images are coded in angle minutes $[0..21600]$. 

HALCON/COM Reference Manual, 2005-2-1
real-images are coded in radians $[0..2\pi]$.

- For int4-images a domain of $[0..1]$ of transformed values must be spread to $[0..10000]$.

---

### Parameter

- **ImageInput1** (input iconic) ... image(-array) $\sim$ HImageX / HObjectX (byte, int4, real) Input image (channel 1).
- **ImageInput2** (input iconic) ... image(-array) $\sim$ HImageX / HObjectX (byte, int4, real) Input image (channel 2).
- **ImageInput3** (input iconic) ... image(-array) $\sim$ HImageX / HObjectX (byte, int4, real) Input image (channel 3).
- **ImageRed** (output iconic) ... image(-array) $\sim$ HImageX / HUntypedObjectX (byte, int4, real) Red channel.
- **ImageGreen** (output iconic) ... image(-array) $\sim$ HImageX / HUntypedObjectX (byte, int4, real) Green channel.
- **ImageBlue** (output iconic) ... image(-array) $\sim$ HImageX / HUntypedObjectX (byte, int4, real) Blue channel.
- **ColorSpace** (input control) ... string $\sim$ String / VARIANT Color space of the input image.
- **Default Value:** 'hsv'
- **List of values:** ColorSpace $\in \{ \text{'hsi', 'yiq', 'yuv', 'argyb', 'ciexyz', 'hls', 'hsv'} \}$

### Example

```c
/* Tranformation from rgb to hsv and conversely */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image,Image1,Image2,Image3,'hsv')
trans_from_rgb(Image1,Image2,Image3,Image4,Image5,'hsv')
trans_to_rgb(Image4,Image5,Image6,Image7,Image8,'hsv')
compose3(ImageRed,ImageGreen,ImageBlue,Multichannel)
disp_color(Multichannel,WindowHandle).
```

### Result

TransToRgb returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

---

**Parallelization Information**

TransToRgb is reentrant and automatically parallelized (on tuple level, domain level).

---

**Possible Predecessors**

Decompose3

---

**Possible Successors**

Compose3, DispColor

---

**See also**

Decompose3

---

**Module**

Image filters

### 3.5 Edges

```c
[out] HRegionX RegionResult HRegionX.CloseEdges ([in] HImageX EdgeImage,
[in] long MinAmplitude )

void HOperatorSetX.CloseEdges ([in] HObjectX Edges,
[in] HObjectX EdgeImage, [out] HUntypedObjectX RegionResult,
[in] VARIANT MinAmplitude )
```

Close edge gaps using the edge amplitude image.
CloseEdges closes gaps in the output of an edge detector, and thus tries to produce complete object contours. This is done by examining the neighbors of each edge point to determine the point with maximum amplitude (i.e., maximum gradient), and adding the point to the edge if its amplitude is larger than the minimum amplitude passed in \texttt{MinAmplitude}. This operator expects as input the edges (\texttt{Edges}) and amplitude image (\texttt{EdgeImage}) returned by typical edge operators, such as \texttt{EdgesImage} or \texttt{SobelAmp}. \texttt{CloseEdges} does not take into account the edge directions that may be returned by an edge operator. Thus, in areas where the gradient is almost constant the edges may become rather “wiggly.”

\begin{itemize}
  \item \textbf{Edges} (input iconic) \rightarrow \texttt{HRegionX / IHObjectX} \\
  Region containing one pixel thick edges.
  \item \textbf{EdgeImage} (input iconic) \rightarrow \texttt{HImageX / IHObjectX (byte, int4)} \\
  Edge amplitude (gradient) image.
  \item \textbf{RegionResult} (output iconic) \rightarrow \texttt{HRegionX / HUntypedObjectX} \\
  Region containing closed edges.
  \item \textbf{MinAmplitude} (input control) \rightarrow \texttt{integer} \rightarrow \texttt{long / VARIANT} \\
  Minimum edge amplitude.
\end{itemize}

Default Value : 16

Suggested values : \texttt{MinAmplitude} \in \{5, 8, 10, 12, 16, 20, 25, 30, 40, 50\}

Typical range of values : \(1 \leq \text{MinAmplitude} \leq 1\)

Minimum Increment : 1

Recommended Increment : 1

Restriction : \((\text{MinAmplitude} \geq 0)\)

\begin{itemize}
  \item \textbf{Result} \rightarrow \texttt{HRegionX / IHObjectX} \\
  \text{CloseEdges} \text{ returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.}
\end{itemize}

\begin{itemize}
  \item \textbf{Parallelization Information} \\
  \text{CloseEdges} \text{ is reentrant and automatically parallelized (on tuple level).}
\end{itemize}

\begin{itemize}
  \item \textbf{Possible Predecessors} \\
  EdgesImage, SobelAmp, Threshold, Skeleton
  \item \textbf{Possible Successors} \\
  Skeleton
  \item \texttt{GraySkeleton}
\end{itemize}

\begin{itemize}
  \item \textbf{Alternatives} \\
  \text{CloseEdgesLength, Dilation1, Closing}
\end{itemize}

\begin{itemize}
  \item \textbf{Module} \\
  \text{Image filters}
\end{itemize}

\begin{verbatim}
[out] HRegionX ClosedEdges HRegionX.CloseEdgesLength

void HOperatorSetX.CloseEdgesLength ([in] IHObjectX Edges,
[in] IHObjectX Gradient, [out] HUntypedObjectX ClosedEdges,
\end{verbatim}

Close edge gaps using the edge amplitude image.

\texttt{CloseEdgesLength} closes gaps in the output of an edge detector, and thus tries to produce complete object contours. This operator expects as input the edges (\texttt{Edges}) and amplitude image (\texttt{Gradient}) returned by typical edge operators, such as \texttt{EdgesImage} or \texttt{SobelAmp}.

Contours are closed in two steps: First, one pixel wide gaps in the input contours are closed, and isolated points are eliminated. After this, open contours are extended by up to \texttt{MaxGapLength} points by adding edge points until either the contour is closed or no more significant edge points can be found. A gradient is regarded as significant if it is larger than \texttt{MinAmplitude}. The neighboring points examined as possible new edge points are the point in the direction of the contour and its two adjacent points in an 8-neighborhood. For each of these points, the sum of
its gradient and the maximum gradient of that point’s three possible neighbors is calculated (look ahead of length 1). The point with the maximum sum is then chosen as the new edge point.

Parameter

- **Edges** (input iconic) .......................... region(-array) \( \sim \) HRegionX / HObjectX
  Region containing one pixel thick edges.

- **Gradient** (input iconic) ........................ image \( \sim \) HImageX / HObjectX (byte)
  Edge amplitude (gradient) image.

- **ClosedEdges** (output iconic) ...................... region(-array) \( \sim \) HRegionX / HUntypedObjectX
  Region containing closed edges.

- **MinAmplitude** (input control) ................... integer \( \sim \) long / VARIANT
  Minimum edge amplitude.
  Default Value: 16
  Suggested values: \( \text{MinAmplitude} \in \{5, 8, 10, 12, 16, 20, 25, 30, 40, 50\}\)
  Typical range of values: \(1 \leq \text{MinAmplitude} \leq 1\)
  Minimum Increment: 1
  Recommended Increment: 1
  Restriction: \((\text{MinAmplitude} \geq 0)\)

- **MaxGapLength** (input control) ................... integer \( \sim \) long / VARIANT
  Maximal number of points by which edges are extended.
  Default Value: 3
  Suggested values: \( \text{MaxGapLength} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50, 70, 100\}\)
  Typical range of values: \(1 \leq \text{MaxGapLength} \leq 1\)
  Minimum Increment: 1
  Recommended Increment: 1
  Restriction: \(((\text{MaxGapLength} > 0) \land (\text{MaxGapLength} \leq 127))\)

**Result**

CloseEdgesLength returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

CloseEdgesLength is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- EdgesImage, SobelAmp, Threshold, Skeleton
- CloseEdges, Dilation1, Closing

Alternatives

References


Module

**Image filters**

```plaintext
[out] HImageX DerivGauss HImageX.DerivateGauss ([in] double Sigma, [in] String Component )

```

Convolve an image with derivatives of the Gaussian.

DerivateGauss convolves an image with the derivatives of a Gaussian and calculates various features derived thereof. Possible values for Component are:

- ‘none’ Smoothing only.
- ‘x’ First derivative along x.

\[
g'(x, y) = \frac{\partial g(x, y)}{\partial x}
\]
'y’ First derivative along y.
\[ g'(x, y) = \frac{\partial g(x, y)}{\partial y} \]

'gradient’ Absolute value of the gradient.
\[ g'(x, y) = \sqrt{\frac{\partial g(x, y)^2}{\partial x} \frac{\partial g(x, y)^2}{\partial y}} \]

'gradient_dir’ Gradient direction in radians
\[ \phi = \text{atan2}(\frac{\partial g(x, y)}{\partial y}, \frac{\partial g(x, y)}{\partial x}) \]

'xx’ Second derivative along x.
\[ g'(x, y) = \frac{\partial^2 g(x, y)}{\partial x^2} \]

'yy’ Second derivative along y.
\[ g'(x, y) = \frac{\partial^2 g(x, y)}{\partial y^2} \]

'xy’ Second derivative along x and y.
\[ g'(x, y) = \frac{\partial^2 g(x, y)}{\partial x \partial y} \]

'xxx’ Third derivative along x.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial x^3} \]

'yyy’ Third derivative along y.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial y^3} \]

'xxy’ Third derivative along x, x and y.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial x^2 \partial y} \]

'xyy’ Third derivative along x, y and y.
\[ g'(x, y) = \frac{\partial^3 g(x, y)}{\partial x \partial y^2} \]

'det’ Determinant of the Hessian matrix:
\[ \text{DET} = \frac{\partial^2 g(x, y)}{\partial x^2} \frac{\partial^2 g(x, y)}{\partial y^2} - \left(\frac{\partial^2 g(x, y)}{\partial y \partial x}\right)^2 \]

'laplace’ Laplace operator (trace of the Hessian matrix):
\[ TR = \frac{\partial^2 g(x, y)}{\partial x^2} + \frac{\partial^2 g(x, y)}{\partial y^2} \]

'mean_curvature’ Mean curvature \( H \)
\[ a = (1 + \frac{\partial g(x, y)^2}{\partial x}) \frac{\partial^2 g(x, y)}{\partial y^2} \]
\[ b = 2 \frac{\partial g(x, y)}{\partial x} \frac{\partial g(x, y)}{\partial y} \frac{\partial^2 g(x, y)}{\partial y \partial x} \]
\[ c = (1 + \frac{\partial g(x, y)^2}{\partial y}) \frac{\partial^2 g(x, y)}{\partial x^2} \]
\[ d = (1 + \frac{\partial g(x, y)^2}{\partial x} + \frac{\partial g(x, y)^2}{\partial y})^2 \]
\[ H = \frac{a - b + c}{d} \]
\[ H = \frac{1}{2}(\kappa_{\text{min}} + \kappa_{\text{max}}) \]
'gauss curvature' Gaussian curvature $K$

$$K = \frac{\text{DET}}{(1 + \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2})^2}$$

'area' Differential Area $A$

$$A = EG - F^2$$

$$E = 1 + \frac{\partial g(x,y)}{\partial x}$$

$$F = \frac{\partial g(x,y) \partial g(x,y)}{\partial x \partial y}$$

$$G = 1 + \frac{\partial g(x,y)}{\partial y}$$

'eigenvalue1' First eigenvalue

$$\lambda_1 = a + \sqrt{a^2 - \left(\frac{\partial^2 g(x,y) \partial^2 g(x,y)}{\partial x^2 \partial y^2} - \frac{\partial^2 g(x,y)^2}{\partial y \partial x}\right)}$$

'eigenvalue2' Second eigenvalue

$$\lambda_2 = a - \sqrt{a^2 - \left(\frac{\partial^2 g(x,y) \partial^2 g(x,y)}{\partial x^2 \partial y^2} - \frac{\partial^2 g(x,y)^2}{\partial y \partial x}\right)}$$

'eigenvector direction' Direction of the eigenvector corresponding to the first eigenvalue in radians

'main1 curvature' First principal curvature

$$\kappa_{max} = H + \sqrt{H^2 - K}$$

'main2 curvature' Second principal curvature

$$\kappa_{min} = H - \sqrt{H^2 - K}$$

'kitchen_rosenfeld' Second derivative perpendicular to the gradient

$$k = \frac{\partial^2 g(x,y) \partial g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y) \partial g(x,y)}{\partial y^2} - 2 \frac{\partial^2 g(x,y) \partial g(x,y)}{\partial x \partial y}$$

'zuniga_haralick' Normalized second derivative perpendicular to the gradient

$$k = \frac{\partial^2 g(x,y) \partial g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y) \partial g(x,y)}{\partial y^2} - 2 \frac{\partial^2 g(x,y) \partial g(x,y)}{\partial x \partial y}$$

'de_saint_venant' Second derivative along and perpendicular to the gradient

$$k = \frac{\partial g(x,y) \partial g(x,y)}{\partial x} \left(\frac{\partial^2 g(x,y)}{\partial x^2} - \frac{\partial^2 g(x,y)}{\partial y^2}\right) - \left(\frac{\partial g(x,y)^2}{\partial x} - \frac{\partial g(x,y)^2}{\partial y}\right) \frac{\partial^2 g(x,y)}{\partial x \partial y}$$
CHAPTER 3. FILTER

\[ \text{Parameter} \]

- **Image**: (input iconic) \( \sim \) \( HImageX / HObjectX \) (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Input image.

- **DerivGauss**: (output iconic) \( \sim \) \( HImageX / HUntypedObjectX \) (real)
  
  Filtered result image.

- **Sigma**: (input control) \( \sim \) \( double / VARIANT \)
  
  Sigma of the Gaussian.
  
  **Default Value**: 1.0
  
  **Suggested values**: \( \Sigma \in \{0.7, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0\} \)
  
  **Typical range of values**: \( 0.2 \leq \Sigma \leq 0.2 \)
  
  **Minimum Increment**: 0.01
  
  **Recommended Increment**: 0.1
  
  **Restriction**: \((\Sigma > 0.0)\)

- **Component**: (input control) \( \sim \) \( String / VARIANT \)
  
  Derivative or feature to be calculated.
  
  **Default Value**: ‘x’
  

\[ \text{Parallelization Information} \]

**DerivateGauss** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Successors**

ZeroCrossing, DualThreshold

**See also**

ZeroCrossing, DualThreshold

**Alternatives**

Laplace, LaplaceOfGauss, GaussImage, SmoothImage

**Module**

Image filters

---

**Approximate the LoG operator (Laplace of Gaussian).**

**DiffOfGauss** approximates the Laplace-of-Gaussian operator by a difference of Gaussians. The standard deviations of these Gaussians can be calculated, according to Marr, from the Parameter **Sigma** of the LoG and the ratio of the two standard deviations (**SigFactor**) as:

\[
\begin{align*}
\sigma_1 &= \frac{\Sigma}{\sqrt{-2\log(\text{SigFactor})}} \\
\sigma_2 &= \frac{\sigma_1}{\text{SigFactor}} \\
\text{DiffOfGauss} &= (\text{Image} \ast \text{gauss}(\sigma_1)) - (\text{Image} \ast \text{gauss}(\sigma_2))
\end{align*}
\]

For a **SigFactor** = 1.6, according to Marr, an approximation to the Mexican-Hat-Operator results. The resulting image is stored in **DiffOfGauss**.

---

HALCON/COM Reference Manual, 2005-2-1
3.5. EDGES

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>(input iconic) (multichannel-)image(-array) (\sim) HImageX / IObjectX (byte) Input image</td>
</tr>
<tr>
<td>DiffOfGauss</td>
<td>(output iconic) (multichannel-)image(-array) (\sim) HImageX / HUntypedObjectX (int2) LoG image.</td>
</tr>
<tr>
<td>Sigma</td>
<td>(input control) (real) Smoothing parameter of the Laplace operator to approximate. Default Value: 3.0 Suggested values: (\Sigma \in {2.0, 3.0, 4.0, 5.0}) Typical range of values: (0.2 \leq \Sigma \leq 0.2) Minimum Increment: 0.01 Recommended Increment: 0.1 Restriction: ((\Sigma &gt; 0.0))</td>
</tr>
<tr>
<td>SigFactor</td>
<td>(input control) (real) Ratio of the standard deviations used (Marr recommends 1.6). Default Value: 1.6 Typical range of values: (0.1 \leq \text{SigFactor} \leq 0.1) Minimum Increment: 0.01 Recommended Increment: 0.1 Restriction: ((\text{SigFactor} &gt; 0.0))</td>
</tr>
</tbody>
</table>

Example

```
read_image(Image,'fabrik')
diff_of_gauss(Image,Laplace,2.0,1.6)
zero_crossing(Laplace,ZeroCrossings).
```

Complexity

The execution time depends linearly on the number of pixels and the size of sigma.

Result

DiffOfGauss returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

DiffOfGauss is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

ZeroCrossing, DualThreshold

Alternatives

Laplace, DerivateGauss

References


Module

Image filters

```
[out] HImageX ImaAmp HImageX.EdgesImage ([out] HImageX ImaDir,
in] long High)

void HOperatorSetX.EdgesImage ([in] IObjectX Image,
[out] HUntypedObjectX ImaAmp, [out] HUntypedObjectX ImaDir,
in] VARIANT High)
```

Extract edges using Deriche, Lanser, Shen, or Canny filters.
EdgesImage detects step edges using recursively implemented filters (according to Deriche, Lanser and Shen) or the conventionally implemented “derivative of Gaussian” filter (using filter masks) proposed by Canny. Thus, the following edge operators are available: 'deriche1', 'lanser1', 'deriche1.int4', 'deriche2', 'lanser2', 'deriche2.int4', 'shen', 'mshen', and 'canny' (parameter Filter).

The edge amplitudes (gradient magnitude) and directions are returned in ImaAmp and ImaDir, respectively. The edge operators 'deriche1' bzw. 'deriche2' are also available for int4-images, and return the signed filter response instead of its absolute value. This behavior can be obtained for byte-images as well by selecting 'deriche1.int4' bzw. 'deriche2.int4' as filter. This can be used to calculate the second derivative of an image by applying EdgesImage (with parameter 'lanser2') to the signed first derivative. Edge directions are stored in 2-degree steps, i.e., an edge direction of \( \theta \) degrees with respect to the horizontal axis is stored as \( \theta / 2 \) in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let \([E_x, E_y]\) denote the image gradient. Then the following edge directions are returned as \( r/2 \):

<table>
<thead>
<tr>
<th>direction</th>
<th>( E_x / E_y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>from bottom to top</td>
<td>0/+ 0</td>
</tr>
<tr>
<td>from lower right to upper left</td>
<td>+/−</td>
</tr>
<tr>
<td>from right to left</td>
<td>+/0 90</td>
</tr>
<tr>
<td>from upper right to lower left</td>
<td>+/+</td>
</tr>
<tr>
<td>from top to bottom</td>
<td>0/+ 180</td>
</tr>
<tr>
<td>from upper left to lower right</td>
<td>−/+</td>
</tr>
<tr>
<td>from left to right</td>
<td>+/0 270</td>
</tr>
<tr>
<td>from lower left to upper right</td>
<td>−/−</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

The “filter width” (i.e., the amount of smoothing) can be chosen arbitrarily, and can be estimated by calling InfoEdges for concrete values of the parameter Alpha. It decreases for increasing Alpha for the Deriche, Lanser and Shen filters and increases for the Canny filter, where it is the standard deviation of the Gaussian on which the Canny operator is based. “Wide” filters exhibit a larger invariance to noise, but also a decreased ability to detect small details. Non-recursive filters, such as the Canny filter, are realized using filter masks, and thus the execution time increases for increasing filter width. In contrast, the execution time for recursive filters does not depend on the filter width. Thus, arbitrary filter widths are possible using the Deriche, Lanser and Shen filters without increasing the run time of the operator. The resulting advantage in speed compared to the Canny operator naturally increases for larger filter widths. As border treatment, the recursive operators assume that the images to be zero outside of the image, while the Canny operator repeats the gray value at the image’s border. Comparable filter widths can be obtained by the following choices of Alpha:

- \( \text{Alpha('lanser1')} = \text{Alpha('deriche1')}, \)
- \( \text{Alpha('deriche2')} = \text{Alpha('deriche1')}/2 \),
- \( \text{Alpha('lanser2')} = \text{Alpha('deriche2')}, \)
- \( \text{Alpha('shen')} = \text{Alpha('deriche1')}/2, \)
- \( \text{Alpha('mshen')} = \text{Alpha('shen')}, \)
- \( \text{Alpha('canny')} = 1.77/\text{Alpha('deriche1')}. \)

The originally proposed recursive filters ('deriche1', 'deriche2', 'shen') return a biased estimate of the amplitude of diagonal edges. This bias is removed in the corresponding modified version of the operators ('lanser1', 'lanser2' and 'mshen'), while maintaining the same execution speed.

For relatively small filter widths \((11 \times 11), i.e., \) for Alpha ('lanser2' = 0.5), all filters yield similar results. Only for “wider” filters differences begin to appear: the Shen filters begin to yield qualitatively inferior results. However, they are the fastest of the implemented operators — closely followed by the Deriche operators.

EdgesImage optionally offers to apply a non-maximum-suppression (NMS = 'nms'/'inms'/'hvnms'; 'none' if not desired) and hysteresis threshold operation \((\text{Low, High}; \) at least one negative if not desired) to the resulting edge image. Conceptually, this corresponds to the following calls:

- \( \text{nonmax_suppression.dir(..., NMS,...)} \)
- \( \text{hysteresis_threshold(..., Low, High, 999,...)} \)
However, this calling sequence would return appropriately modified regions, whereas \texttt{EdgesImage} marks suppressed edge points by 0 in the amplitude and 255 in the direction image, while leaving the region of the image unchanged.

\begin{verbatim}
read_image(Image,'fabrik')
edges_image(Image,Amp,Dir,'lanser2',0.5,'none',-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30).
\end{verbatim}

\textbf{Result}

\texttt{EdgesImage} returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via \texttt{SetSystem('noObjectResult',<Result>}). If necessary, an exception handling is raised.

\textbf{Parallelization Information}

\texttt{EdgesImage} is reentrant and automatically parallelized (on tuple level, channel level).
CHAPTER 3. FILTER

Possible Predecessors
InfoEdges

Possible Successors
Threshold, HysteresisThreshold, CloseEdgesLength

See also
InfoEdges, NonmaxSuppressionAmp, HysteresisThreshold, BandpassImage

Alternatives
SobelDir, FreiDir, KirschDir, PrewittDir, RobinsonDir

References


S. Castan, J. Zhao und J. Shen: “Optimal Filter for Edge Detection Methods and Results”; Proc. of the First European Conference on Computer Vision, Antibes; Lecture Notes on computer Science; no. 427; S. 12-17; Springer-Verlag; 1990.

Image filters

```c

```

Extract sub-pixel precise edges using Deriche, Lanser, Shen, or Canny filters.

`EdgesSubPix` detects step edges using recursively implemented filters (according to Deriche, Lanser and Shen) or the conventionally implemented “derivative of Gaussian” filter (using filter masks) proposed by Canny. Thus, the following edge operators are available:

- `deriche1`, `lanser1`, `deriche2`, `lanser2`, `shen`, `mshen`, `canny` and `sobel` (parameter `Filter`).

The extracted edges are returned as sub-pixel precise XLD contours in `Edges`. For each edge point the following attributes are defined (see `GetContourAttribXld`):

- `edge_direction`: Edge direction
- `angle`: Direction of the normal vectors to the contour (oriented such that the normal vectors point to the right side of the contour as the contour is traversed from start to end point; the angles are given with respect to the row axis of the image.)
- `response`: Edge amplitude (gradient magnitude)

The “filter width” (i.e., the amount of smoothing) can be chosen arbitrarily, and can be estimated by calling `InfoEdges` for concrete values of the parameter `Alpha`. It decreases for increasing `Alpha` for the Deriche, Lanser and Shen filters and increases for the Canny filter, where it is the standard deviation of the Gaussian on
which the Canny operator is based. “Wide” filters exhibit a larger invariance to noise, but also a decreased ability to detect small details. Non-recursive filters, such as the Canny filter, are realized using filter masks, and thus the execution time increases for increasing filter width. In contrast, the execution time for recursive filters does not depend on the filter width. Thus, arbitrary filter widths are possible using the Deriche, Lanser and Shen filters without increasing the run time of the operator. The resulting advantage in speed compared to the Canny operator naturally increases for larger filter widths. As border treatment, the recursive operators assume that the images to be zero outside of the image, while the Canny operator repeats the gray value at the image’s border. Comparable filter widths can be obtained by the following choices of $\alpha$:

$$\begin{align*}
\alpha(lanser1') &= \alpha(deriche1') \\
\alpha(deriche2') &= \alpha(deriche1')/2 \\
\alpha(lanser2') &= \alpha(deriche2') \\
\alpha(shen') &= \alpha(deriche1')/2 \\
\alpha(mshen') &= \alpha(shen') \\
\alpha(canny') &= 1.77/\alpha(deriche1')
\end{align*}$$

The originally proposed recursive filters (‘deriche1’, ‘deriche2’, ‘shen’) return a biased estimate of the amplitude of diagonal edges. This bias is removed in the corresponding modified version of the operators (‘lanser1’, ‘lanser2’ und ‘mshen’), while maintaining the same execution speed. For relatively small filter widths ($11 \times 11$), i.e., for $\alpha(\text{lanser2'}) = 0.5$, all filters yield similar results. Only for “wider” filters differences begin to appear: the Shen filters begin to yield qualitatively inferior results. However, they are the fastest of the implemented operators — closely followed by the Deriche operators. 

**EdgesSubPix** links the edge points into edges by using an algorithm similar to a hysteresis threshold operation, which is also used in **LinesGauss**. Points with an amplitude larger than high are immediately accepted as belonging to an edge, while points with an amplitude smaller than low are rejected. All other points are accepted as edges if they are connected to accepted edge points (see also **LinesGauss** and **HysteresisThreshold**).

Because edge extractors are often unable to extract certain junctions, a mode that tries to extract these missing junctions by different means can be selected by appending ‘junctions’ to the values of **Filter** that are described above. This mode is analogous to the mode for completing junctions that is available in **LinesGauss**.

---

**Parameter**

- **Image** (input iconic) ................................. image $\sim HImageX / HObjectX$ (byte, uint2)  
  Input image.

- **Edges** (output iconic) ................................. xld_cont $\sim HXLDContX / HUntypedObjectX$  
  Extracted edges.

- **Filter** (input control) ................................. string $\sim \text{String} / \text{VARIANT}$  
  Edge operator to be applied.
  
  **Default Value** : ‘lanser2’
  
  **List of values** : $\{ \text{‘deriche1’}, \text{‘lanser1’}, \text{‘deriche2’}, \text{‘lanser2’}, \text{‘shen’}, \text{‘mshen’}, \text{‘canny’}, \text{‘sobel’}, \text{‘deriche1’} \text{‘junctions’}, \text{‘lanser1’} \text{‘junctions’}, \text{‘deriche2’} \text{‘junctions’}, \text{‘lanser2’} \text{‘junctions’}, \text{‘shen’} \text{‘junctions’}, \text{‘mshen’} \text{‘junctions’}, \text{‘canny’} \text{‘junctions’}, \text{‘sobel’} \text{‘junctions’} \}$

- **Alpha** (input control) ................................. real $\sim \text{double} / \text{VARIANT}$  
  Filter parameter: small values result in strong smoothing, and thus less detail (opposite for ‘canny’).
  
  **Default Value** : 0.5
  
  **Suggested values** : $0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1$
  
  **Typical range of values** : $0.2 \leq \alpha \leq 0.2$
  
  **Minimum Increment** : 0.01
  
  **Recommended Increment** : 0.1
  
  **Restriction** : $(\alpha > 0.0)$

- **Low** (input control) ................................. integer $\sim \text{long} / \text{VARIANT}$  
  Lower threshold for the hysteresis threshold operation.
  
  **Default Value** : 20
  
  **Suggested values** : $5, 10, 15, 20, 25, 30, 40$
  
  **Typical range of values** : $1 \leq Low \leq 1$
  
  **Minimum Increment** : 1
  
  **Recommended Increment** : 5
  
  **Restriction** : $(\text{Low} > 0)$
\textbf{High} (input control) ................................................................. integer $\rightarrow$ long / VARIANT  
Upper threshold for the hysteresis threshold operation.  
\textbf{Default Value} : 40  
\textbf{Suggested values} : \textbf{High} $\in \{10, 15, 20, 25, 30, 40, 50, 60, 70\}$  
\textbf{Typical range of values} : $1 \leq \text{High} \leq 1$  
\textbf{Minimum Increment} : 1  
\textbf{Recommended Increment} : 5  
\textbf{Restriction} : $((\text{High} > 0) \land (\text{High} \geq \text{Low}))$

--- Example ---

\texttt{read\_image(Image,'fabrik')}  
\texttt{edges\_sub\_pix(Image,Edges,'lanser2',0.5,20,40)}.

--- Complexity ---

Let $A$ be the number of pixels in the domain of \texttt{Image}. Then the runtime complexity is $O(A \times \Sigma)$ for the Canny filter and $O(A)$ for the recursive Lanser, Deriche, and Shen filters. 

Let $S = \text{Width} \times \text{Height}$ be the number of pixels of \texttt{Image}. Then \texttt{EdgesSubPix} requires at least $60 \times S$ bytes of temporary memory during execution.

--- Result ---

\texttt{EdgesSubPix} returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via \texttt{SetSystem('noObjectResult',<Result>)}. If necessary, an exception handling is raised.

--- Parallelization Information ---

\texttt{EdgesSubPix} is reentrant and automatically parallelized (on tuple level).

--- See also ---

\texttt{InfoEdges, HysteresisThreshold, BandpassImage, LinesGauss, LinesFacet}

--- Alternatives ---

\texttt{SobelDir, FreiDir, KirschDir, PrewittDir, RobinsonDir, EdgesImage}

--- References ---


S.Castan, J.Zhao und J.Shen: “Optimal Filter for Edge Detection Methods and Results”; Proc. of the First European Conference on Computer Vision, Antibes; Lecture Notes on computer Science; no. 427; S. 12-17; Springer-Verlag; 1990.
Detect edges (amplitude) using the Frei-Chen operator.

**FreiAmp** calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 & 0 & -1 \\
1 & 0 & -1 \\
1 & 0 & -1
\end{bmatrix}
\]

The result image contains the maximum response of the masks \(A\) and \(B\).

**Example**

```c
read_image(Image,'fabrik')
frei_amp(Image,Frei_amp)
threshold(Frei_amp,Edges,128,255).
```

**Result**

**FreiAmp** always returns TRUE. If the input is empty the behaviour can be set via `SetSystem` ("noObjectResult",<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

**FreiAmp** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessors**

GaussImage, SigmaImage, MedianImage, SmoothImage

**See also**

BandpassImage, LaplaceOfGauss

**Alternatives**

SobelAmp, KirschAmp, PrewittAmp, RobinsonAmp, Roberts

**Module**

Image filters
CHAPTER 3. FILTER

\[ A = \begin{bmatrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{bmatrix} \]

\[ B = \begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix} \]

The result image contains the maximum response of the masks \( A \) and \( B \). The edge directions are returned in \texttt{ImageEdgeDir}, and are stored in 2-degree steps, i.e., an edge direction of \( \theta \) degrees with respect to the horizontal axis is stored as \( \theta/2 \) in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let \([E_x, E_y]\) denote the image gradient. Then the following edge directions are returned as \( \theta/2 \):

<table>
<thead>
<tr>
<th>Intensity Increase</th>
<th>( E_x/E_y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge direction ( \theta )</td>
<td></td>
</tr>
<tr>
<td>From bottom to top</td>
<td>( 0/+ 0 )</td>
</tr>
<tr>
<td>From lower right to upper left</td>
<td>( +/- [0, 90] )</td>
</tr>
<tr>
<td>From right to left</td>
<td>( +/- 90 )</td>
</tr>
<tr>
<td>From upper right to lower left</td>
<td>( +/- [90, 180] )</td>
</tr>
<tr>
<td>From top to bottom</td>
<td>( 0/+ 180 )</td>
</tr>
<tr>
<td>From upper left to lower right</td>
<td>( +/- [180, 270] )</td>
</tr>
<tr>
<td>From left to right</td>
<td>( +/- 270 )</td>
</tr>
<tr>
<td>From lower left to upper right</td>
<td>( +/- [270, 360] )</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

**Example**

```plaintext```
read_image(Image, 'fabrik')
frei_dir(Image, Frei_dirA, Frei_dirD)
threshold(Frei_dirA, Res, 128, 255).
```

**Result**

\texttt{FreiDir} always returns TRUE. If the input is empty the behaviour can be set via \texttt{SetSystem} ("noObjectResult", <Result>). If necessary, an exception handling is raised.

**Parallelization Information**

\texttt{FreiDir} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, channel level, domain level).

## Possible Predecessors

GaussImage, SigmaImage, MedianImage, SmoothImage

## Possible Successors

HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength

See also

BandpassImage, LaplaceOfGauss

HALCON/COM Reference Manual, 2005-2-1
3.5. EDGES

Alternatives
EdgesImage, SobelDir, RobinsonDir, PrewittDir, KirschDir

Image filters

```
[out] HImageX Highpass HImageX.HighpassImage ([in] long Width, 
[in] long Height )
void HOperatorSetX.HighpassImage ([in] IHObjectX Image, 
```

Extract high frequency components from an image.

HighpassImage extracts high frequency components in an image by applying a linear filter with the following matrix (in case of a 7 × 5 matrix):

```
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 -35 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
```

This corresponds to applying a mean operator (MeanImage), and then subtracting the original gray value. A value of 128 is added to the result, i.e., zero crossings occur for 128.

This filter emphasizes high frequency components (edges and corners). The cutoff frequency is determined by the size (Height × Width) of the filter matrix: the larger the matrix, the smaller the cutoff frequency is.

At the image borders the pixels’ gray values are mirrored. In case of over- or underflow the gray values are clipped (255 and 0, resp.).

Attention
If even values are passed for Height or Width, the operator uses the next larger odd value instead. Thus, the center of the filter mask is always uniquely determined.

Parameter

- **Image** (input iconic) ....................... (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte )
  Input image.

- **Highpass** (output iconic) ............ (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte )
  High-pass-filtered result image.

- **Width** (input control) ......................... extent.x \(\sim\) long / VARIANT
  Width of the filter mask.
  Default Value : 9
  Suggested values : Width \(\in\) \{3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101\}
  Typical range of values : 3 \(\leq\) Width \(\leq\) 3
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : (Width ∧ odd)

- **Height** (input control) ......................... extent.y \(\sim\) long / VARIANT
  Height of the filter mask.
  Default Value : 9
  Suggested values : Height \(\in\) \{3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101\}
  Typical range of values : 3 \(\leq\) Height \(\leq\) 3
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : (Height ∧ odd)

Result

HighpassImage returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.
HighpassImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

Threshold, Skeleton

See also

DynThreshold

Alternatives

MeanImage, SubImage, ConvolImage, BandpassImage

Module

Image filters

```
```

Estimate the width of a filter in EdgesImage. InfoEdges returns an estimate of the width of any of the filters used in EdgesImage. To do so, the corresponding continuous impulse responses of the filters are sampled until the first filter coefficient is smaller than five percent of the largest coefficient. Alpha is the filter parameter (see EdgesImage). Seven edge operators are supported (parameter Filter):

- 'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen' und 'canny'.

The parameter Mode ('edge'/'smooth') is used to determine whether the corresponding edge or smoothing operator is to be sampled. The Canny operator (which uses the Gaussian for smoothing) is implemented using conventional filter masks, while all other filters are implemented recursively. Therefore, for the Canny filter the coefficients of the one-dimensional impulse responses \( f(n) \) with \( n \geq 0 \) are returned in Coeffs in addition to the filter width.

Parameter

- Filter (input control) ................................................. string  \( \rightarrow \) String / VARIANT
  Name of the edge operator.
  Default Value : 'lanser2'
  List of values : Filter \( \in \{ 'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen', 'canny' \} \)
- Mode (input control) ................................................... string  \( \rightarrow \) String / VARIANT
  1D edge filter ('edge') or 1D smoothing filter ('smooth').
  Default Value : 'edge'
  List of values : Mode \( \in \{ 'edge', 'smooth' \} \)
- Alpha (input control) .................................................. real  \( \rightarrow \) double / VARIANT
  Filter parameter: small values result in strong smoothing, and thus less detail (opposite for 'canny').
  Default Value : 0.5
  Typical range of values : \( 0.2 \leq Alpha \leq 0.2 \)
  Minimum Increment : 0.01
  Recommended Increment : 0.1
  Restriction : \((Alpha > 0.0)\)
- Size (output control) ................................................ integer  \( \rightarrow \) long / VARIANT
  Filter width in pixels.
- Coeffs (output control) ............................................. integer  \( \rightarrow \) VARIANT(integer)
  For Canny filters: Coefficients of the “positive” half of the 1D impulse response.

Example

```
read_image(Image,'fabrik')
info_edges('lanser2','edge',0.5,Size,Coeffs)
edges_image(Image,Amp,Dir,'lanser2',0.5,'none',-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30).
```
InfoEdges returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

InfoEdges is reentrant and processed without parallelization.

Possible Successors

EdgesImage, Threshold, Skeleton

See also

EdgesImage

Module

Image filters

[out] HImageX ImageEdgeAmp HImageX.KirschAmp ( )

void HOperatorSetX.KirschAmp ([in] IHObjectX Image,
[out] HUntypedObjectX ImageEdgeAmp )

Detect edges (amplitude) using the Kirsch operator.

KirschAmp calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

\[
\begin{bmatrix}
-3 & -3 & 5 \\
-3 & 0 & 5 \\
-3 & -3 & 5 \\
-3 & 5 & 5 \\
-3 & 0 & 5 \\
-3 & -3 & -3 \\
5 & 5 & 5 \\
-3 & 0 & -3 \\
-3 & -3 & -3 \\
5 & 5 & -3 \\
5 & 0 & -3 \\
-3 & -3 & -3 \\
5 & -3 & -3 \\
5 & 0 & -3 \\
5 & -3 & -3 \\
-3 & -3 & -3 \\
5 & 0 & -3 \\
5 & 5 & -3 \\
-3 & -3 & -3 \\
-3 & 0 & -3 \\
5 & 5 & 5 \\
-3 & -3 & -3 \\
-3 & 0 & 5 \\
-3 & 5 & 5
\end{bmatrix}
\]

The result image contains the maximum response of all masks.

Parameter

- Image (input iconic) ............(multichannel-)image(-array)  \rightarrow  HImageX / IHObjectX ( byte, int2, uint2 )
  Input image.

- ImageEdgeAmp (output iconic) ....... image(-array)  \rightarrow  HImageX / HUntypedObjectX ( byte, int2, uint2 )
  Edge amplitude (gradient magnitude) image.
read_image(Image,'fabrik')
kirsch_amp(Image,Kirsch_amp)
threshold(Kirsch_amp,Edges,128,255).

**Result**

*KirschAmp* always returns TRUE. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>).` If necessary, an exception handling is raised.

**Parallelization Information**

*KirschAmp* is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessors**

GaussImage, SigmaImage, MedianImage, SmoothImage

**See also**

BandpassImage, LaplaceOfGauss

**Alternatives**

SobelAmp, FreiAmp, PrewittAmp, RobinsonAmp, Roberts

**Module**

Image filters

```c
[out] HImageX ImageEdgeAmp HImageX.KirschDir
{[out] HImageX ImageEdgeDir }

void HOperatorSetX.KirschDir ([in] IHOBJECTX Image,
[out] HUngroundedObjectX ImageEdgeAmp, [out] HUngroundedObjectX ImageEdgeDir )
```

Detect edges (amplitude and direction) using the Kirsch operator.

*KirschDir* calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

```
-3  -3  5  
-3  0  5  
-3  -3  5  

-3  5  5  
-3  0  5  
-3  -3  -3

5  5  5  
-3  0  -3  
-3  -3  -3

5  5  -3  
5  0  -3  
-3  -3  -3

5  -3  -3  
5  0  -3  
5  -3  -3

-3  -3  -3  
-3  0  -3  
5  5  5
```

HALCON/COM Reference Manual, 2005-2-1
The result image contains the maximum response of all masks. The edge directions are returned in ImageEdgeDir, and are stored as \( x/2 \). They correspond to the direction of the mask yielding the maximum response.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image (input iconic) ( \ldots \ldots ) (multichannel-)image(-array) ( \sim ) HImageX / IObjectX (byte, int2, uint2) Input image.</td>
</tr>
<tr>
<td>ImageEdgeAmp (output iconic) ( \ldots \ldots ) (multichannel-)image(-array) ( \sim ) HImageX / HObjectX (byte, int2, uint2) Edge amplitude (gradient magnitude) image.</td>
</tr>
<tr>
<td>ImageEdgeDir (output iconic) ( \ldots \ldots ) (multichannel-)image(-array) ( \sim ) HImageX / HObjectX (direction) Edge direction image.</td>
</tr>
</tbody>
</table>

Example

```
read_image(Image,'fabrik')
kirsch_dir(Image,Kirsch_dirA,Kirsch_dirD)
threshold(Kirsch_dirA,Res,128,255).
```

Result

KirschDir always returns TRUE. If the input is empty the behaviour can be set via SetSystem ('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

KirschDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

GaussImage, SigmaImage, MedianImage, SmoothImage

Possible Successors

HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength

See also

BandpassImage, LaplaceOfGauss

Alternatives

EdgesImage, SobelDir, RobinsonDir, PrewittDir, FreiDir

Module

Image filters

```
[-3  -3  -3  
-3  0   5  
-3  5   5  ]
```

\( \begin{bmatrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{bmatrix} \)

Calculate the Laplace operator by using finite differences.

Laplace filters the input images Image using a Laplace operator. Depending on the parameter NeighbourhoodType the following simple approximations of the Laplace operator are used:

'\texttt{n.4}'

\( \begin{bmatrix} 1 \\ -4 & 1 \\ 1 \end{bmatrix} \)
The filter 'n8' corresponds to the filter mask used in \texttt{HighpassImage(O:R:3,3:)}. For a Laplace operator with size $3 \times 3$, the corresponding filter is applied directly, while for larger filter sizes ($\text{Size} = 5, 7, 9$ and 11) the input image is first smoothed using a Gaussian filter of the selected size. Therefore,

\begin{verbatim}
laplace(O:R:int4,S,N:)
\end{verbatim}

for $\text{Size} >$ is equivalent to

\begin{verbatim}
gauss\_image(O:G:S:) \triangleright
laplace(G:R:int4,3,N:).
\end{verbatim}

\texttt{Laplace} either returns the absolute value of the Laplace filtered image ($\text{FilterType}\ 'abs'$) in a byte/uint2-image or the signed result ($\text{FilterType}\ 'int4'$) in an int4-image.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Image}</td>
<td>input iconic \hspace{1.5cm} (multichannel)-image(-array) $\triangleright$ \texttt{HImageX / HObjectX} (byte, uint2) Input image.</td>
</tr>
<tr>
<td>\textit{ImageLaplace}</td>
<td>output iconic \hspace{1.5cm} (multichannel)-image(-array) $\triangleright$ \texttt{HImageX / HUntypedObjectX} (byte, uint2, int4) Laplace-filtered result images.</td>
</tr>
<tr>
<td>\textit{FilterType}</td>
<td>input control \hspace{1.5cm} string $\triangleright$ \texttt{String / VARIANT} Calculate the absolute value of the filter to a byte/uint2-image or the signed result to an int4-image. Default Value : 'int4' List of values : $\text{FilterType} \in {\text{'abs'}, \text{'int4'}}$</td>
</tr>
<tr>
<td>\textit{Size}</td>
<td>input control \hspace{1.5cm} integer $\triangleright$ \texttt{long / VARIANT} Size of filter mask. Default Value : 3 List of values : $\text{Size} \in {3, 5, 7, 9, 11}$</td>
</tr>
<tr>
<td>\textit{NeighbourhoodType}</td>
<td>input control \hspace{1.5cm} string $\triangleright$ \texttt{String / VARIANT} Neighborhood used in the Laplace operator Default Value : 'n8_isotrop' List of values : $\text{NeighbourhoodType} \in {\text{'n4'}, \text{'n8'}, \text{'n8_isotrop'}}$</td>
</tr>
</tbody>
</table>

\texttt{Laplace} returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via \texttt{SetSystem('noObjectResult',<Result>).} If necessary, an exception handling is raised.

---

| Parallelization Information | \texttt{Laplace} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, channel level, domain level). |

---

| Possible Successors | \texttt{ZeroCrossing, DualThreshold} \hspace{1.5cm} See also \texttt{HighpassImage, EdgesImage} |
| Alternatives       | \texttt{DiffOfGauss, LaplaceOfGauss} Module |

---

| Image filters | 

---

HALCON/COM Reference Manual, 2005-2-1
3.5. EDGES

LoG-Operator (Laplace of Gaussian).

\texttt{LaplaceOfGauss} calculates the Laplace-of-Gaussian operator, i.e., the Laplace operator on a Gaussian smoothed image, for arbitrary smoothing parameters \( \Sigma \). The Laplace operator is given by:

\[
\Delta g(x, y) = \frac{\partial^2 g(x, y)}{\partial x^2} + \frac{\partial^2 g(x, y)}{\partial y^2}
\]

The derivatives in \texttt{LaplaceOfGauss} are calculated by appropriate derivatives of the Gaussian, resulting in the following formula for the convolution mask:

\[
\Delta G_\sigma(x, y) = \frac{1}{2\pi\sigma^4} \left( \frac{x^2 + y^2}{2\sigma^2} - 1 \right) \left[ \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) \right]
\]

\begin{itemize}
  \item \texttt{Image} (input iconic) \ldots (multichannel-)image(-array) \leadsto \texttt{HImageX} / \texttt{HObjectX} (byte, int1, int2, uint2, int4, real)
  \hspace{1cm} \text{Input image.}
  \item \texttt{ImageLaplace} (output iconic) \ldots (multichannel-)image(-array) \leadsto \texttt{HImageX} / \texttt{HUntypedObjectX} (int2)
  \hspace{1cm} \text{Laplace filtered image.}
  \item \texttt{Sigma} (input control) \ldots \text{number} \leadsto \texttt{VARIANT} (real, integer)
  \hspace{1cm} \text{Smoothing parameter of the Gaussian.}
  \hspace{1cm} \textbf{Default Value:} 2.0
  \hspace{1cm} \textbf{Suggested values:} \( \Sigma \in \{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0\} \)
  \hspace{1cm} \textbf{Typical range of values:} \( 0.7 \leq \Sigma \leq 0.7 \)
  \hspace{1cm} \textbf{Minimum Increment:} 0.01
  \hspace{1cm} \textbf{Recommended Increment:} 0.1
  \hspace{1cm} \textbf{Restriction:} \(((\Sigma > 0.7) \land (\Sigma \leq 25.0))\)
\end{itemize}

\texttt{LaplaceOfGauss} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

\texttt{ZeroCrossing}, \texttt{DualThreshold}

See also

\texttt{DerivateGauss}

Alternatives

\texttt{Laplace}, \texttt{DiffOfGauss}, \texttt{DerivateGauss}

Module

Image filters

\texttt{[out]} \texttt{HImageX ImageLaplace} \texttt{HImageX.LaplaceOfGauss ([in] VARIANT Sigma )}


Detect edges (amplitude) using the Prewitt operator.

\texttt{PrewittAmp} calculates an approximation of the first derivative of the image data and is used as an edge detector.

The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 1 & 1 \\
0 & 0 & 0 \\
-1 & -1 & -1
\end{bmatrix}
\]

HALCON 6.1.4
The result image contains the maximum response of the masks $A$ and $B$.

$A = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

$B = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

Detect edges (amplitude and direction) using the Prewitt operator.

PrewittDir calculates an approximation of the first derivative of the image data and is used as an edge detector. The filter is based on the following filter masks:

$A = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

$B = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

The result image contains the maximum response of the masks $A$ and $B$. The edge directions are returned in ImageEdgeDir, and are stored in 2-degree steps, i.e., an edge direction of $x$ degrees with respect to the horizontal.
axis is stored as \( x/2 \) in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let \([E_x, E_y]\) denote the image gradient. Then the following edge directions are returned as \( r/2 \):

<table>
<thead>
<tr>
<th>Intensity Change</th>
<th>Edge Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E_x/E_y )</td>
<td>( 0/0 )</td>
</tr>
<tr>
<td>( +/0 )</td>
<td>( 90 )</td>
</tr>
<tr>
<td>( +/+ )</td>
<td>( 90, 180 )</td>
</tr>
<tr>
<td>( 0/+ )</td>
<td>( 180 )</td>
</tr>
<tr>
<td>( +/+ )</td>
<td>( 270 )</td>
</tr>
<tr>
<td>( 0/0 )</td>
<td>( 360 )</td>
</tr>
<tr>
<td>( +/0 )</td>
<td>( 180, 270 )</td>
</tr>
<tr>
<td>( +/0 )</td>
<td>( 0, 90 )</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

Parameter

- **Image** (input iconic) 
  - *(multichannel-)*image-array \( \sim HImageX / IObjectX \) (byte, int2, uint2)
  - Input image.

- **ImageEdgeAmp** (output iconic) 
  - *(multichannel-)*image-array \( \sim HImageX / HUntypedObjectX \) (byte, int2, uint2)
  - Edge amplitude (gradient magnitude) image.

- **ImageEdgeDir** (output iconic) 
  - *(multichannel-)*image-array \( \sim HImageX / HUntypedObjectX \) (direction)
  - Edge direction image.

Example

```plaintext
read_image(Image, 'fabrik')
prewitt_dir(Image, PrewittA, PrewittD)
```

Result

PrewittDir always returns TRUE. If the input is empty the behaviour can be set via SetSystem (‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

PrewittDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

- GaussImage
- SigmaImage
- MedianImage
- SmoothImage

Possible Successors

- HysteresisThreshold
- Threshold
- GraySkeleton
- NonmaxSuppressionDir
- CloseEdges
- CloseEdgesLength

See also

- BandpassImage
- LaplaceOfGauss

Alternatives

- EdgesImage
- SobelDir
- RobinsonDir
- FreiDir
- KirschDir

Module

Image filters

```plaintext
<table>
<thead>
<tr>
<th>out</th>
<th>HImageX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageRoberts</td>
<td></td>
</tr>
<tr>
<td>HImageX.Roberts</td>
<td>([in] String FilterType )</td>
</tr>
</tbody>
</table>
```

```plaintext
| void |
| HOperatorSetX.Roberts | ([in] IObjectX Image, |
| out | HUntypedObjectX ImageRoberts, [in] VARIANT FilterType ) |
```

Detect edges using the Roberts filter.
Roberts calculates the first derivative of an image and is used as an edge operator. If the following mask describes a part of the image,

\[
\begin{array}{cc}
A & B \\
C & D
\end{array}
\]

the different filter types are defined as follows:

- 'roberts_max' \(\max(|A - D|, |B - C|)\)
- 'gradient_max' \(\max(|A + B - (C + D)|, |A + C - (B + D)|)\)
- 'gradient_sum' \(|A + B - (C + D)| + |A + C - (B + D)|\)

If an overflow occurs the result is clipped. The result of the operator is stored at the pixel with the coordinates of "D".

If an overflow occurs the result is clipped. The result of the operator is stored at the pixel with the coordinates of "D".

Parameter

- **Image** (input iconic) \(\cdots\) \((\text{multichannel-})\text{image(-array)}\) \(\sim\) \(\text{HImageX} / \text{HIObjectX} \) \((\text{byte, int2, uint2})\)
  - Input image.
- **ImageRoberts** (output iconic) \(\cdots\) \((\text{multichannel-})\text{image(-array)}\) \(\sim\) \(\text{HImageX} / \text{HUntypedObjectX} \) \((\text{byte, int2, uint2})\)
  - Roberts-filtered result images.
- **FilterType** (input control) \(\cdots\) \(\text{string}\) \(\sim\) \(\text{String} / \text{VARIANT}\)
  - Filter type.
  - Default Value: 'gradient_sum'
  - List of values: FilterType \(\in\) \{'roberts_max', 'gradient_max', 'gradient_sum'\}

Example

```
read_image(Image,'fabrik')
roberts(Image,Roberts,'roberts_max')
threshold(Roberts,Margin,128,255).
```

Result

Roberts returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Roberts is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

- GaussImage
- Threshold, Skeleton

Possible Successors

- Laplace, HighpassImage, BandpassImage
- See also
- Alternatives
- EdgesImage, SobelAmp, FreiAmp, KirschAmp, PrewittAmp

Module

Image filters

```c
[out] HImageX ImageEdgeAmp HImageX.RobinsonAmp ( )

void HOperatorSetX.RobinsonAmp ([in] HIObjectX Image,
[out] HUntypedObjectX ImageEdgeAmp )
```

Detect edges (amplitude) using the Robinson operator.
RobinsonAmp calculates an approximation of the first derivative of the image data and is used as an edge detector. In RobinsonAmp the following four of the originally proposed eight \(3 \times 3\) filter masks are convolved with the image. The other four masks are obtained by a multiplication by -1. All masks contain only the values 0,1,-1,2,-2.

\[
\begin{array}{ccc}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 \\
2 & 1 & 0 \\
1 & 0 & -1 \\
0 & -1 & -2 \\
0 & 1 & 2 \\
-1 & 0 & 1 \\
-2 & -1 & 0 \\
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{array}
\]

The result image contains the maximum response of all masks.

The result image contains the maximum response of all masks.

---

**Parameter**

- **Image** (input iconic) .......... (multichannel-)image(-array) ~ HImageX / IObjectX (byte, int2, uint2)
  - Input image.

- **ImageEdgeAmp** (output iconic) ..... (multichannel-)image(-array) ~ HImageX / HUntypedObjectX (byte, int2, uint2)
  - Edge amplitude (gradient magnitude) image.

---

**Example**

```c
read_image(Image,'fabrik')
robinson_amp(Image,Robinson_amp)
threshold(Robinson_amp,Edges,128,255).
```

---

**Result**

RobinsonAmp always returns TRUE. If the input is empty the behaviour can be set via SetSystem (‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

RobinsonAmp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessors**

GaussImage, SigmaImage, MedianImage, SmoothImage

**See also**

BandpassImage, LaplaceOfGauss

**Alternatives**

SobelAmp, FreiAmp, PrewittAmp, RobinsonAmp, Roberts

**Module**

Image filters

---

```c
[out] HImageX ImageEdgeAmp HImageX.RobinsonDir
([out] HImageX ImageEdgeDir )
```

void HOperatorSetX.RobinsonDir ([in] IObjectX Image,
[out] HUntypedObjectX ImageEdgeAmp, [out] HUntypedObjectX ImageEdgeDir )

Detect edges (amplitude and direction) using the Robinson operator.
RobinsonDir calculates an approximation of the first derivative of the image data and is used as an edge detector. In RobinsonAmp the following four of the originally proposed eight \(3 \times 3\) filter masks are convolved with the image. The other four masks are obtained by a multiplication by -1. All masks contain only the values 0, 1, -1, 2, -2.

\[
\begin{pmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{pmatrix}
\]

\[
\begin{pmatrix}
2 & 1 & 0 \\
1 & 0 & -1 \\
0 & -1 & -2
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & 1 & 2 \\
-1 & 0 & 1 \\
-2 & -1 & 0
\end{pmatrix}
\]

\[
\begin{pmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{pmatrix}
\]

The result image contains the maximum response of all masks. The edge directions are returned in ImageEdgeDir, and are stored as \(x/2\). They correspond to the direction of the mask yielding the maximum response.

**Parameter**

- **Image** (input iconic) .......... (multichannel-)image(-array) \(\rightarrow\) HImageX / HObjectX (byte, int2, uint2)
  
  Input image.

- **ImageEdgeAmp** (output iconic) ....... (multichannel-)image(-array) \(\rightarrow\) HImageX / HUntypedObjectX (byte, int2, uint2)
  
  Edge amplitude (gradient magnitude) image.

- **ImageEdgeDir** (output iconic) ...... (multichannel-)image(-array) \(\rightarrow\) HImageX/HUntypedObjectX (direction)
  
  Edge direction image.

**Example**

```
read_image(Image,‘fabrik’)
robinson_dir(Image,Robinson_dirA,Robinson_dirD)
```

**Result**

RobinsonDir always returns TRUE. If the input is empty the behaviour can be set via SetSystem (‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

RobinsonDir is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessors**

GaussImage, SigmaImage, MedianImage, SmoothImage

**Possible Successors**

HysteresisThreshold, Threshold, GraySkeleton, NonmaxSuppressionDir, CloseEdges, CloseEdgesLength

See also

BandpassImage, LaplaceOfGauss

**Alternatives**

EdgesImage, SobelDir, KirschDir, PrewittDir, FreiDir

Image filters
Detect edges (amplitude) using the Sobel operator.

SobelAmp calculates first derivative of an image and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1 \\
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
\]

These masks are used differently, according to the selected filter type. (In the following, \(a\) and \(b\) denote the results of convolving an image with \(A\) and \(B\) for one particular pixel.)

- \(\text{'sum\_sqrt'}\): \(\sqrt{a^2 + b^2}\)
- \(\text{'sum\_abs'}\): \((|a| + |b|)/2\)
- \(\text{'thin\_sum\_abs'}\): \((\text{thin}(|a|) + \text{thin}(|b|))/2\)
- \(\text{'thin\_max\_abs'}\): \(\max(\text{thin}(|a|), \text{thin}(|b|))\)
- \(\text{'x'}\): \(b\)
- \(\text{'y'}\): \(a\)

Here, \(\text{thin}(x)\) is equal to \(x\) for a vertical maximum (mask \(A\)) and a horizontal maximum (mask \(B\)), respectively, and 0 otherwise. Thus, for \(\text{'thin\_sum\_abs'}\) and \(\text{'thin\_max\_abs'}\) the gradient image is thinned. For the filter types \(x\) and \(y\) if the input image is of type byte the output image is of type int1, of type int2 otherwise. For a Sobel operator with size \(3 \times 3\), the corresponding filters \(A\) and \(B\) are applied directly, while for larger filter sizes (\(\text{Size} = 5, 7, 9\) and 11) the input image is first smoothed using a Gaussian filter of size \(\text{Size} - 2\). Therefore,

\[
sobel\_amp(I:E, \text{Dir: FilterTyp}, S:)
\]

for \(\text{Size} > 3\) is equivalent to

\[
\text{gauss\_image}(I:G:S-2) \triangleright \text{sobel\_amp}(G:E: \text{FilterType}, 3:).
\]
read_image(Image,'fabrik')
sobel_amp(Image,Amp,'sum_abs',3)
threshold(Amp,Edg,128,255).

Result
SobelAmp returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information
SobelAmp is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors
GaussImage, MeanImage, AnisotropeDiff, SigmaImage

Possible Successors
Threshold, NonmaxSuppressionAmp, GraySkeleton

See also
Laplace, HighpassImage, BandpassImage

Alternatives
FreiAmp, Roberts, KirschAmp, PrewittAmp, RobinsonAmp

Module
Image filters

[out] HImageX EdgeAmplitude HImageX.SobelDir

Detect edges (amplitude and direction) using the Sobel operator.
SobelDir calculates first derivative of an image and is used as an edge detector. The filter is based on the following filter masks:

\[
A = \begin{bmatrix}
1 & 2 & 1 \\
0 & 0 & 0 \\
-1 & -2 & -1
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1
\end{bmatrix}
\]

These masks are used differently, according to the selected filter type. (In the following, \(a\) und \(b\) denote the results of convolving an image with \(A\) und \(B\) for one particular pixel.)

- \(\text{sum} \_ \text{sqrt} = \sqrt{a^2 + b^2}\)
- \(\text{sum} \_ \text{abs} = (|a| + |b|)/2\)

For a Sobel operator with size \(3 \times 3\), the corresponding filters \(A\) and \(B\) are applied directly, while for larger filter sizes (\(\text{Size} = 5, 7, 9\) and 11) the input image is first smoothed using a Gaussian filter of size \(\text{Size} - 2\). Therefore,

\[
sobel \_ dir(I:Amp,Dir:FilterTyp,S:)
\]

for \(\text{Size} > 3\) is equivalent to
The edge directions are returned in `EdgeDirection`, and are stored in 2-degree steps, i.e., an edge direction of $x$ degrees with respect to the horizontal axis is stored as $x/2$ in the edge direction image. Furthermore, the direction of the change of intensity is taken into account. Let $[E_x, E_y]$ denote the image gradient. Then the following edge directions are returned as $r/2$:

<table>
<thead>
<tr>
<th>Edge Direction</th>
<th>$E_x / E_y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>From bottom to top</td>
<td>$0 / +0$</td>
</tr>
<tr>
<td>From lower right to upper left</td>
<td>$+ / -0$</td>
</tr>
<tr>
<td>From right to left</td>
<td>$0 / +90$</td>
</tr>
<tr>
<td>From upper right to lower left</td>
<td>$+ / -90$</td>
</tr>
<tr>
<td>From top to bottom</td>
<td>$- / +180$</td>
</tr>
<tr>
<td>From upper left to lower right</td>
<td>$- / +270$</td>
</tr>
<tr>
<td>From left to right</td>
<td>$0 / -270$</td>
</tr>
</tbody>
</table>

Points with edge amplitude 0 are assigned the edge direction 255 (undefined direction).

---

**Parameter**

- **Image** (input iconic) ................. (multichannel-)image(-array)  \( \sim \) HImageX / IHObjectX (byte, int2, uint2)
  Input image.

- **EdgeAmplitude** (output iconic) ....... (multichannel-)image(-array)  \( \sim \) HImageX / HUntypedObjectX (byte, int2, uint2)
  Edge amplitude (gradient magnitude) image.

- **EdgeDirection** (output iconic) ...... (multichannel-)image(-array)  \( \sim \) HImageX / HUntypedObjectX (direction)
  Edge direction image.

- **FilterType** (input control) ......................... string  \( \sim \) String / VARIANT
  Filter type.
  **Default Value**: ‘sum_abs’
  **List of values**: FilterType \( \in \) {'sum_abs', 'sum_sqrt'}

- **Size** (input control) ................................. integer  \( \sim \) long / VARIANT
  Size of filter mask.
  **Default Value**: 3
  **List of values**: Size \( \in \) {3, 5, 7, 9, 11, 13}

---

**Example**

```plaintext
read_image(Image,'fabrik')

sobel_dir(Image,Amp,Dir,’sum_abs’,3)

threshold(Amp,Edg,128,255).
```

---

**Result**

`SobelDir` returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem(’noObjectResult’,<Result>)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

`SobelDir` is reentrant and automatically parallelized (on tuple level, channel level).

---

**Possible Predecessors**

GaussImage, MeanImage, AnisotropeDiff, SigmaImage

---

**Possible Successors**

NonmaxSuppressionDir, HysteresisThreshold, Threshold

---

**See also**

Roberts, Laplace, HighpassImage, BandpassImage

---
3.6 Enhancement

Enhance contrast of the image.

The operator *Emphasize* emphasizes high frequency areas of the image (edges and corners). The resulting images appears sharper.

First the procedure carries out a filtering with the low pass (*MeanImage*). The resulting gray values (*res*) are calculated from the obtained gray values (*mean*) and the original gray values (*orig*) as follows:

\[
res := \text{round}((\text{orig} - \text{mean}) \times \text{Factor}) + \text{orig}
\]

*Factor* serves as measurement of the increase in contrast. The division frequency is determined via the size of the filter matrix: The larger the matrix, the lower the division frequency.

As an edge treatment the gray values are mirrored at the edges of the image. Overflow and/or underflow of gray values is clipped.

---

**Attention**

---

**Parameter**

- **Image** (input iconic) ..................(multichannel-)image(-array) \(\sim\) *HImageX / IHObjectX* ( byte, int2, uint2 )
  Image to be enhanced.

- **ImageEmphasize** (output iconic) .... (multichannel-)image(-array) \(\sim\) *HImageX / HUntypedObjectX* ( byte, int2, uint2 )
  contrast enhanced image.

- **MaskWidth** (input control) ..................................................extent.x \(\sim\) long / VARIANT
  Width of low pass mask.
  
  Default Value: 7
  Suggested values: MaskWidth \(\in\) \{3, 5, 7, 9, 11, 15, 21, 25, 31, 39\}
  Typical range of values: \(3 \leq \text{MaskWidth} \leq 3\)
  Minimum Increment: 2
  Recommended Increment: 2

- **MaskHeight** (input control) ..................................................extent.y \(\sim\) long / VARIANT
  Height of the low pass mask.
  
  Default Value: 7
  Suggested values: MaskHeight \(\in\) \{3, 5, 7, 9, 11, 15, 21, 25, 31, 39\}
  Typical range of values: \(3 \leq \text{MaskHeight} \leq 3\)
  Minimum Increment: 2
  Recommended Increment: 2

- **Factor** (input control) ..................................................real \(\sim\) double / VARIANT
  Intensity of contrast emphasis.
  
  Default Value: 1.0
  Suggested values: Factor \(\in\) \{0.3, 0.5, 0.7, 1.0, 1.4, 1.8, 2.0\}
  Typical range of values: \(0.0 \leq \text{Factor} \leq 0.0\text{(sqrt)}\)
  Minimum Increment: 0.01
  Recommended Increment: 0.2
  Restriction: \((0 < \text{Factor}) \land (\text{Factor} < 20)\)
Example

read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
draw_region(Region,WindowHandle)
reduce_domain(Image,Region,Mask)
emphasize(Mask,Sharp,7,7,2.0)
disp_image(Sharp,WindowHandle).

Result

If the parameter values are correct the operator Emphasize returns the value TRUE The be-

havior in case of empty input (no input images available) is set via the operator SetSystem (

::'noObjectResult',<Result>::). If necessary an exception handling is raised.

Parallelization Information

Emphasize is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

DispImage

See also

MeanImage,HighpassImage

Alternatives

MeanImage,SubImage,Laplace,AddImage

Module

Image filters

---

Histogram linearisation of images

The operator EquHistoImage enhances the contrast. The starting point is the histogram of the input images.

The following simple gray value transformation \( f(g) \) is carried out for byte images:

\[
  f(g) = 255 \sum_{x=0}^{g} h(x)
\]

\( h(x) \) describes the relative frequency of the occurrence of the gray value \( x \). For uint2 images, the only difference is that the value 255 is replaced with a different maximum value. The maximum value is computed from the number of significant bits stored with the input image, provided that this value is set. If not, the value of the system parameter 'int2_bits' is used (see SetSystem), if this value is set (i.e., different from -1). If none of the two values is set, the number of significant bits is set to 16.

This transformation linearises the cumulative histogram. Maxima in the original histogram are "spreaded" and thus the contrast in image regions with these frequently occurring gray values is increased. Supposedly homogenous regions receive more easily visible structures. On the other hand, of course, the noise in the image increases correspondingly. Minima in the original histogram are dually "compressed". The transformed histogram contains gaps, but the remaining gray values used occur approximately at the same frequency ("histogram equalization").

Attention

The operator EquHistoImage primarily serves for optical processing of images for a human viewer. For example, the (local) contrast spreading can lead to a detection of fictitious edges.

Parameter

- **Image** (input iconic) (multichannel-)image(-array) \( \sim \) HImageX / IObjectX (byte, uint2)
  Image to be enhanced.

- **ImageEquHisto** (output iconic) (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX (byte, uint2)
  Image with linearized gray values.
**Parallelization Information**

`EquHistImage` is reentrant and automatically parallelized (on tuple level, channel level).

**Possible Successors**

DispImage

**See also** ScaleImage, ScaleImageMax, Illuminate

**Alternatives**

ScaleImage

**References**


**Module** Image filters

```plaintext

```

**Illuminate image.**

The operator `Illuminate` enhances contrast. Very dark parts of the image are "illuminated" more strongly, very light ones are "darkened". If `orig` is the original gray value and `mean` is the corresponding gray value of the low pass filtered image detected via the operators `MeanImage` and filter size `MaskHeight x MaskWidth`. For byte-images `val` equals 127, for int2-images and uint2-images `val` equals the median value. The resulting gray value is `new`:

\[ new := \text{round}((val - mean) \times \text{Factor} + orig) \]

The low pass should have rather large dimensions (30 x 30 to 200 x 200). Reasonable parameter combinations might be:

<table>
<thead>
<tr>
<th>MaskHeight</th>
<th>MaskWidth</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>40</td>
<td>0.55</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>0.7</td>
</tr>
<tr>
<td>150</td>
<td>150</td>
<td>0.8</td>
</tr>
</tbody>
</table>

i.e. the larger the low pass mask is chosen, the larger `Factor` should be as well.

The following "spotlight effect" should be noted: If, for example, a dark object is in front of a light wall the object as well as the wall, which is already light in the immediate proximity of the object contours, are lightened by the operator `Illuminate`. This corresponds roughly to the effect that is produced when the object is illuminated by a strong spotlight. The same applies to light objects in front of a darker background. In this case, however, the fictitious "spotlight" darkens objects.

**Attention**

- **Parameter**
  - `Image` (input iconic) .............(multichannel-)image(-array) \(\sim\) HIMageX/IHObjectX (byte, int2, uint2)
    Image to be enhanced.
  - `ImageIlluminate` (output iconic) ....... (multichannel-)image(-array) \(\sim\) HIMageX/HUntypedObjectX (byte, int2, uint2)
    "Illuminate" image.
3.6. ENHANCEMENT

- **MaskWidth** (input control) .......................... extent.x  \(\sim\) long / VARIANT
  - Width of low pass mask.
  - Default Value: 101
  - Suggested values: \(\text{MaskWidth} \in \{31, 41, 51, 71, 101, 121, 151, 201\}\)
  - Typical range of values: \(3 \leq \text{MaskWidth} \leq 3\)
  - Minimum Increment: 2
  - Recommended Increment: 10

- **MaskHeight** (input control) .......................... extent.y  \(\sim\) long / VARIANT
  - Height of low pass mask.
  - Default Value: 101
  - Suggested values: \(\text{MaskHeight} \in \{31, 41, 51, 71, 101, 121, 151, 201\}\)
  - Typical range of values: \(3 \leq \text{MaskHeight} \leq 3\)
  - Minimum Increment: 2
  - Recommended Increment: 10

- **Factor** (input control) .......................... real  \(\sim\) double / VARIANT
  - Scales the "correction gray value" added to the original gray values.
  - Default Value: 0.7
  - Suggested values: \(\text{Factor} \in \{0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0\}\)
  - Typical range of values: \(0.0 \leq \text{Factor} \leq 0.0\)
  - Minimum Increment: 0.01
  - Recommended Increment: 0.2
  - Restriction: \((0 < \text{Factor}) \land (\text{Factor} < 5))\)

---

Example

```cpp
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
illuminate(Image,Better,40,40,0.55)
disp_image(Better,WindowHandle).
```

---

Result

If the parameter values are correct the operator *Illuminate* returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator *SetSystem* (::'noObjectResult',<Result>). If necessary an exception handling is raised.

---

Parallelization Information

*Illuminate* is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

Possible Successors

- DispImage

---

See also

- Emphasize,GrayHisto

---

Alternatives

- ScaleImageMax,EquHistoImage,MeanImage,SubImage

---

Module

Image filters

```cpp
[out] HImageX ImageScaleMax HImageX.ScaleImageMax ( )
void HOperatorSetX.ScaleImageMax ([in] IObjectX Image, [out] HUntypedObjectX ImageScaleMax )
```

Maximum gray value spreading in the value range 0 bis 255.

The operator *ScaleImageMax* calculates the minimum and maximum and scales the image to the maximum value range of a byte image. This way the dynamics (value range) is fully exploited. The number of different gray scales does not change, but in general the visual impression is enhanced. The gray values of images of the \(\text{real}, \text{int2}, \text{uint2}\) and \(\text{int4}\) type are scaled to the range 0 to 255 and returned as byte images.
CHAPTER 3. FILTER

The output always is an image of the type byte.

Parameter

▷ Image (input iconic) ...... (multichannel-)image(-array) \(\sim\) HImageX / IObjectX (byte, int2, uint2, int4, real)

Image to be scaled.

▷ ImageScaleMax (output iconic) .................. image(-array) \(\sim\) HImageX / HUntypedObjectX (byte)

contrast enhanced image.

Parallelization Information

ScaleImageMax is reentrant and automatically parallelized (on tuple level, channel level).

Possible Successors

DispImage

See also

MinMaxGray, GrayHisto

Alternatives

EquiHistoImage, ScaleImage, Illuminate, ConvertImageType

Module

Image filters

3.7 FFT

Convolve an image with a byte-mask in the frequency domain.

Convolfft convolves two (Fourier-transformed) images in the frequency domain, i.e., the pixels of the complex image ImageFFT are multiplied by the corresponding pixels of the filter ImageFilter. This image is a byte-image in which 0 corresponds to complete suppression of a frequency, and 255 corresponds to no suppression. The result image is of type 'complex'.

Attention

The filtering is always done on the entire image, i.e., the region of the image is ignored.

Parameter

▷ ImageFFT (input iconic) ......................... image(-array) \(\sim\) HImageX / IObjectX (complex)

Complex input image.

▷ ImageFilter (input iconic) ....................... image \(\sim\) HImageX / IObjectX (byte)

Filter in frequency domain.

▷ ImageConvol (output iconic) .................... image(-array) \(\sim\) HImageX / HUntypedObjectX (complex)

Result of applying the filter.

Result

Convolfft returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(::'noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Convolfft is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

FftImage, FftGeneric, GenHighpass, GenLowpass, GenBandpass, GenBandfilter

Possible Successors

PowerByte, PowerReal, PowerLn, FftImageInv, FftGeneric

See also

GenGabor, GenHighpass, GenLowpass, GenBandpass, ConvGabor, FftImageInv
### ConvolGabor

**Module**

Image filters

```
```

**ConvolGabor** convolves a Fourier-transformed image with a Gabor filter `GaborFilter` (see `GenGabor`) and its Hilbert transform in the frequency domain. The result image is of type 'complex'.

**Attention**

The filtering is always done on the entire image, i.e., the region of the image is ignored.

**Parameter**

- **ImageFFT** (input iconic) 
  
  Input image.
- **GaborFilter** (input iconic) 
  
  Gabor/Hilbert-Filter.
- **ImageResultGabor** (output iconic) 
  
  Result of the Gabor filter.
- **ImageResultHilbert** (output iconic) 
  
  Result of the Hilbert filter.

**Result**

**ConvolGabor** returns TRUE if all images are of correct type. If the input is empty the behaviour can be set via `SetSystem(::’noObjectResult’,<Result>;)`. If necessary, an exception handling is raised.

**Parallelization Information**

**ConvolGabor** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

FftImage, FftGeneric, GenGabor

**Possible Successors**

PowerByte, PowerReal, PowerLn, FftImageInv, FftGeneric

**See also**

ConvolImage

**Alternatives**

ConvolFft

**Module**

Image filters

```
```

**EnergyGabor** calculates the local contrast (Energy) of the two input images. The energy of the resulting image is then defined as

\[
\text{Energy} = \text{channel1}^2 + \text{channel2}^2 .
\]
Often the calculation of the energy is preceded by the convolution of an image with a Gabor filter and the Hilbert transform of the Gabor filter (see ConvolGabor). In this case, the first channel of the image passed to EnergyGabor is the Gabor-filtered image, transformed back into the spatial domain (see FftImageInv), and the second channel the result of the convolution with the Hilbert transform, also transformed back into the spatial domain. The local energy is a measure for the local contrast of structures (e.g., edges and lines) in the image.

Parameter

- **ImageGabor** (input iconic) ……………………. image(-array) \(\rightarrow\) HImageX / IObjectX (byte, real)
  
  1st channel of input image (usually: Gabor image).

- **ImageHilbert** (input iconic) ……………………. image(-array) \(\rightarrow\) HImageX / IObjectX (byte, real)
  
  2nd channel of input image (usually: Hilbert image).

- **Energy** (output iconic) ……………………. image(-array) \(\rightarrow\) HImageX / HObjectX (real)
  
  Image containing the local energy.

Result

EnergyGabor returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>):). If necessary, an exception handling is raised.

Parallelization Information

EnergyGabor is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessors

GenGabor, ConvolGabor, FftImageInv

Module

Image filters

Compute the fast Fourier transform of an image.

FftGeneric computes the fast Fourier transform of the input image Image. Because several definitions of the forward and reverse transforms exist in the literature, this operator allows the user to select the most convenient definition.

The general definition of a Fourier transform is as follows:

\[
F(m, n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} e^{s2\pi i (km/M + ln/N)} f(k, l)
\]

Opinions vary on whether the sign \(s\) in the exponent should be set to 1 or \(-1\) for the forward transform, i.e. the for going to the frequency domain. There is also disagreement on the magnitude of the normalizing factor \(c\). This is sometimes set to 1 for the forward transform, sometimes to \(MN\), and sometimes (in case of the unitary FFT) to \(\sqrt{MN}\). Especially in image processing applications the DC term is shifted to the center of the image.

FftGeneric allows to select these choices individually. The parameter Direction allows to select the logical direction of the FFT. (This parameter is not unnecessary; it is needed to discern how to shift the image if the DC term should rest in the center of the image.) Possible values are ‘to_freq’ and ‘from_freq’. The parameter Exponent is used to determine the sign of the exponent. It can be set to 1 or \(-1\). The normalizing factor can be set with Norm, and can take on the values ‘none’, ‘sqrt’ and ‘n’. The parameter Mode determines the location of the DC term of the FFT. It can be set to ‘dc_center’ or ‘dc_edge’.

In any case, the user must ensure the consistent use of the parameters. This means that the normalizing factors used for the forward and backward transform must yield \(MN\) when multiplied, the exponents must be of opposite sign, and Mode must be equal for both transforms.

A consistent combination is, for example ('to_freq', 1, 'sqrt', 'dc_center') for the forward transform and ('from_freq', 1, 'none', 'dc_edge') for the reverse transform. In this case, the FFT can be interpreted as interpolation.
with trigonometric basis functions. Another useful combination is \((\text{\textit{to\_freq}},1,\text{\textit{sqrt}},\text{\textit{dc\_center}})\) and \((\text{\textit{from\_freq}},-1,\text{\textit{sqrt}},\text{\textit{dc\_center}})\).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Image(-array)</td>
</tr>
<tr>
<td>ImageFFT</td>
<td>Image(-array)</td>
</tr>
<tr>
<td>Direction</td>
<td>string</td>
</tr>
<tr>
<td>Exponent</td>
<td>integer</td>
</tr>
<tr>
<td>Norm</td>
<td>string</td>
</tr>
<tr>
<td>Mode</td>
<td>string</td>
</tr>
</tbody>
</table>

**FftGeneric** returns TRUE if the input image is of correct type, its width and height are a power of 2, and all parameters are correct. If the input is empty the behaviour can be set via **SetSystem**. If necessary, an exception handling is raised.

**Parallelization Information**

**FftGeneric** is reentrant and automatically parallelized (on tuple level).

**Possible Successors**

ConvolFft, ConvolGabor, ConvertImageType, PowerByte, PowerReal, PowerLn, PhaseDeg, PhaseRad, EnergyGabor

**Alternatives**

FftImage, FftImageInv

**Module**

Image filters

```halcon
[out] HImageX ImageFFT HImageX.FftImage ( )

void HOperatorSetX.FftImage ([in] IHObjectX Image, [out] HUntypedObjectX ImageFFT )
```

**Compute the fast Fourier transform of an image.**

**FftImage** calculates the Fourier transform of the input image (Image), i.e., it transforms the image into the frequency domain. The algorithm used is the fast Fourier transform. This corresponds to the call

```halcon
FftGeneric(Image,ImageFFT,\"toFreq\",1,\"n\",\"dcCenter\")
```

The result image is of type ‘complex’.

**Attention**

The filtering is always done on the entire image, i.e., the region of the image is ignored. The images must be quadratic and the width and height must be a power of 2.
CHAPTER 3. FILTER

Parameter

▷ Image (input iconic) ................. image(-array) ⊲ HImageX / HObjectX (byte, real)
Input image.
▷ ImageFFT (output iconic) .......... image(-array) ⊲ HImageX / HUntypedObjectX (complex)
Fourier-transformed image.

Result

FftImage returns TRUE if the input image is of correct type and its width and height are a power of 2. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>:). If necessary, an exception handling is raised.

Parallelization Information

FftImage is reentrant and automatically parallelized (on tuple level).

Possible Successors

ConvolFft, ConvolGabor, ConvertedImageType, PowerByte, PowerReal, PowerLn, PhaseDeg, PhaseRad

See also

FftImageInv

Alternatives

FftGeneric

Module

Image filters

[out] HImageX ImageFFTInv HImageX.FftImageInv ( )

void HOperatorSetX.FftImageInv ([in] HObjectX Image, [out] HUntypedObjectX ImageFFTInv )

Compute the inverse fast Fourier transform of an image.

FftImageInv calculates the inverse Fourier transform of the input image (Image), i.e., it transforms the image back into the spatial domain. This corresponds to the call

\[
\text{FftGeneric(Image,ImageFFT,’fromFreq’,-1,’sqrt’,’dcCenter’)}
\]
\[
\text{ConvertImageType(ImageFFT,ImageFFTInv,’byte’)}
\]

The result image is of type ‘byte’.

Attention

The filtering is always done on the entire image, i.e., the region of the image is ignored. The images must be quadratic and the width and height must be a power of 2.

Parameter

▷ Image (input iconic) ................. image(-array) ⊲ HImageX / HObjectX (complex)
Input image.
▷ ImageFFTInv (output iconic) ........ image(-array) ⊲ HImageX / HUntypedObjectX (byte)
Inverse-Fourier-transformed image.

Result

FftImageInv returns TRUE if the input image is of correct type and its width and height are a power of 2. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>:). If necessary, an exception handling is raised.

Parallelization Information

FftImageInv is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

ConvolFft, ConvolGabor, FftImage

Possible Successors

ConvertImageType, EnergyGabor
3.7. FFT

See also FftImage, FftGeneric, EnergyGabor

Alternatives FftGeneric

Module

Image filters

```c
void HImageX.GenBandfilter ([in] double MinFrequency,

void HOperatorSetX.GenBandfilter ([out] HUntypedObjectX ImageFilter,
```

Generate an ideal band filter.

GenBandfilter generates an ideal band filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameters MinFrequency and MaxFrequency determine the cutoff frequencies of the filter (in pixels). The resulting image contains an annulus with the value 0, and the value 255 outside of this annulus.

Parameter

- **ImageFilter** (output iconic) ....................... image ~ HImageX / HUntypedObjectX (byte) Band filter in the frequency domain.
- **MinFrequency** (input control) ...................... real ~ double / VARIANT Minimum frequency.
  - Default Value : 20
  - Suggested values : MinFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  - Minimum Increment : 1
  - Recommended Increment : 1
  - Restriction : (MinFrequency > 0)
- **MaxFrequency** (input control) ..................... real ~ double / VARIANT Maximum frequency.
  - Default Value : 40
  - Suggested values : MaxFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  - Minimum Increment : 1
  - Recommended Increment : 1
  - Restriction : (MaxFrequency > 0)
- **Size** (input control) ................................. integer ~ long / VARIANT Size (dimension) of the image (filter).
  - Default Value : 512
  - List of values : Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Result

GenBandfilter returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallelization Information

GenBandfilter is reentrant and processed without parallelization.

Possible Successors

ConvolFft

See also GenHighpass, GenLowpass, GenBandpass

Alternatives GenCircle, PaintRegion

Module

Image filters


Generate an ideal bandpass filter.

GenBandpass generates an ideal bandpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameters MinFrequency and MaxFrequency determine the cutoff frequencies of the filter (in pixels). The resulting image contains an annulus with the value 255, and the value 0 outside of this annulus.

Parameter

- **ImageBandpass** (output iconic) ......................... image $\sim$ HImageX / HUntypedObjectX (byte )
  Bandpass filter in the frequency domain.
- **MinFrequency** (input control) ............................ real $\sim$ double / VARIANT
  Minimum frequency.
  Default Value : 20
  Suggested values : MinFrequency $\in \{ 10, 20, 30, 40, 50, 60, 70, 100 \}$
  Typical range of values : $1 \leq \text{MinFrequency} \leq 1$
  Minimum Increment : 1
  Recommended Increment : 1
  Restriction : (MinFrequency > 0)
- **MaxFrequency** (input control) ............................ real $\sim$ double / VARIANT
  Maximum frequency.
  Default Value : 40
  Suggested values : MaxFrequency $\in \{ 10, 20, 30, 40, 50, 60, 70, 100 \}$
  Typical range of values : $1 \leq \text{MaxFrequency} \leq 1$
  Minimum Increment : 1
  Recommended Increment : 1
  Restriction : (MaxFrequency > 0)
- **Size** (input control) ........................................ integer $\sim$ long / VARIANT
  Size (dimension) of the image (filter).
  Default Value : 512
  List of values : Size $\in \{ 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 \}$

Result

GenBandpass returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallelization Information

GenBandpass is reentrant and processed without parallelization.

Possible Successors

ConvolFft

See also

GenHighpass, GenLowpass, GenBandfilter

Alternatives

GenCircle, PaintRegion

Module

Image filters


Store a filter mask in the spatial domain as a real-image.
GenFilterMask stores a filter mask in the spatial domain as a real-image. The center of the filter mask lies in the center of the resulting image. The parameter Scale determines by which amount the values of the filter mask are multiplied (this results in larger values of the Fourier transform of the filter). The corresponding filter matrix, which is given in FilterMask can be generated either from a file or a tuple. The format of the filter matrix is described with the operator ConvolImage. Example filter masks can be found in the directory “filter” in the HALCON home directory. This operator is useful for visualizing the frequency response of filter masks (by applying a Fourier transform to the result image of this operator).

Parameter

- **ImageFilter** (output iconic) .......................... image(-array)  ～  HImageX / HUntypedObjectX (real)  
  Filter in the spatial domain.
- **FilterMask** (input control)  .......................... string(-array)  ～  VARIANT (integer, string)  
  Filter mask as file name or tuple.
  Default Value: ‘sobel’
  Suggested values: FilterMask ∈ {’laplace4’, ’laplace8’, ’lowpass_3_3’}
- **Scale** (input control)  .......................... real  ～  double / VARIANT  
  Scaling factor.
  Default Value: 1.0
  Suggested values: Scale ∈ [0.3, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0]
  Typical range of values: 0.001 ≤ Scale ≤ 0.001
  Minimum Increment: 0.001
  Recommended Increment: 0.1
  Restriction: (Scale > 0.0)
- **Size** (input control)  .......................... integer  ～  long / VARIANT  
  Size (dimension) of the image (filter).
  Default Value: 512
  List of values: Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Example

* If the filter should be read from a file:
  gen_filter_mask (Filter, ’lowpas_3_3’, 1.0, 512)
* If the filter should be directly passed as a tuple:
  gen_filter_mask (Filter, [3,3,9,1,1,1,1,1,1,1,1,1], 1.0, 512)
  fft_image (Filter, FilterFFT)
  set_paint (WindowHandle, ’3D-plot_hidden’)  
  disp_image (FilterFFT, WindowHandle)

Parallelization Information

GenFilterMask is reentrant and processed without parallelization.

Possible Successors

FftImage, FftGeneric

See also

ConvollImage

Module

Image filters

```c
void HImageX.GenGabor ([in] double Angle, [in] double Frequency,

void HOperatorSetX.GenGabor ([out] HUntypedObjectX ImageFilter, 
```

Generate a Gabor filter.

GenGabor generates a Gabor filter with a user-definable bandpass frequency range and sign for the Hilbert transformation. This is done by calculating a symmetrical filter in the frequency domain, which can be adapted by the
parameters *Angle*, *Frequency*, *Bandwidth* and *Orientation* such that a certain frequency band and a certain direction range in the spatial domain is filtered out in the frequency domain.

The parameters *Frequency* (central frequency = distance from the DC term) and *Orientation* (direction) determine the center of the filter. Larger values of *Frequency* result in higher frequencies being passed. A value of 0 for *Orientation* generates a horizontally oriented “crescent” (the bulge of the crescent points upward). Higher values of *Orientation* result in the counterclockwise rotation of the crescent.

The parameters *Angle* and *Bandwidth* are used to determine the range of frequencies and angles being passed by the filter. The larger *Angle* is, the smaller the range of angles passed by the filter gets (because the “crescent” gets narrower). The larger *Bandwidth* is, the smaller the frequency band being passed gets (because the “crescent” gets thinner).

The resulting image is a two-channel real-image, containing the Gabor filter in the first channel and the corresponding Hilbert filter in the second channel.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ImageFilter</strong></td>
<td>(output iconic) multichannel-image-array  ( \sim ) HImageX / HUntypedObjectX (real) Gabor and Hilbert filter.</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>(input control) real  ( \sim ) double / VARIANT Angle range, inversely proportional to the range of orientations.</td>
</tr>
<tr>
<td>Default Value</td>
<td>1.4</td>
</tr>
<tr>
<td>Suggested values</td>
<td><em>Angle</em> ( \in {1.0, 1.2, 1.4, 1.6, 2.0, 2.5, 3.0, 5.0, 6.0, 10.0, 20.0, 30.0, 50.0, 70.0, 100.0} <strong>Typical range of values</strong>: 1.0 ( \leq ) <em>Angle</em> ( \leq ) 1.0 <strong>Minimum Increment</strong>: 0.001 <strong>Recommended Increment</strong>: 0.1</td>
</tr>
<tr>
<td><strong>Frequency</strong></td>
<td>(input control) real  ( \sim ) double / VARIANT Distance of the center of the filter to the DC term.</td>
</tr>
<tr>
<td>Default Value</td>
<td>0.4</td>
</tr>
<tr>
<td>Suggested values</td>
<td><em>Frequency</em> ( \in {0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.699} <strong>Typical range of values</strong>: 0.0 ( \leq ) <em>Frequency</em> ( \leq ) 0.0 <strong>Minimum Increment</strong>: 0.00001 <strong>Recommended Increment</strong>: 0.005</td>
</tr>
<tr>
<td><strong>Bandwidth</strong></td>
<td>(input control) real  ( \sim ) double / VARIANT Bandwith range, inversely proportional to the range of frequencies being passed.</td>
</tr>
<tr>
<td>Default Value</td>
<td>1.0</td>
</tr>
<tr>
<td>Suggested values</td>
<td><em>Bandwidth</em> ( \in {0.1, 0.3, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0, 50.0} <strong>Typical range of values</strong>: 0.05 ( \leq ) <em>Bandwidth</em> ( \leq ) 0.05 <strong>Minimum Increment</strong>: 0.001 <strong>Recommended Increment</strong>: 0.1</td>
</tr>
<tr>
<td><strong>Orientation</strong></td>
<td>(input control) real  ( \sim ) double / VARIANT Angle of the principal orientation being passed.</td>
</tr>
<tr>
<td>Default Value</td>
<td>1.5</td>
</tr>
<tr>
<td>Suggested values</td>
<td><em>Orientation</em> ( \in {0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0, 2.2, 2.4, 2.6, 2.8, 3.0, 3.14} <strong>Typical range of values</strong>: 0.0 ( \leq ) <em>Orientation</em> ( \leq ) 0.0 <strong>Minimum Increment</strong>: 0.0001 <strong>Recommended Increment</strong>: 0.05</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>(input control) integer  ( \sim ) long / VARIANT Size (dimension) of the image (filter).</td>
</tr>
<tr>
<td>Default Value</td>
<td>512</td>
</tr>
<tr>
<td>List of values</td>
<td><em>Size</em> ( \in {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192} <strong>Result</strong></td>
</tr>
</tbody>
</table>

*GenGabor* returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

---

**Parallelization Information**

*GenGabor* is *reentrant* and processed *without* parallelization.

---

**Possible Predecessors**

FftImage, FftGeneric

HALCON/COM Reference Manual, 2005-2-1
Generate an ideal highpass filter.

**GenHighpass** generates an ideal highpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameter `Frequency` determines the cutoff frequency of the filter (in pixels). The resulting image contains a circle of radius `Frequency` with the value 0, and the value 255 outside of this circle.

- **ImageHighpass** (output iconic) - image \( \sim HImageX / HUntypedObjectX (\text{byte}) \)
  - Highpass filter in the frequency domain.

- **Frequency** (input control) - real \( \sim double / VARIANT\)
  - Cutoff frequency.
    - **Default Value**: 20
    - **Suggested values**: `Frequency` \(\in\) \{10, 20, 30, 40, 50, 60, 70, 100\}
    - **Typical range of values**: \(1 \leq Frequency \leq 1\)
    - **Minimum Increment**: 1
    - **Recommended Increment**: 1
    - **Restriction**: \((Frequency > 0)\)

- **Size** (input control) - integer \( \sim long / VARIANT\)
  - Size (dimension) of the image (filter).
    - **Default Value**: 512
    - **List of values**: `Size` \(\in\) \{16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}

**Result**

`GenHighpass` returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**

`GenHighpass` is reentrant and processed without parallelization.

---

Generate an ideal lowpass filter.

**GenLowpass** generates an ideal lowpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameter `Frequency` determines the cutoff frequency of the filter (in pixels). The resulting image contains a circle of radius `Frequency` with the value 0, and the value 255 outside of this circle.

- **ImageLowpass** (output iconic) - image \( \sim HImageX / HUntypedObjectX (\text{byte}) \)
  - Lowpass filter in the frequency domain.

- **Frequency** (input control) - real \( \sim double / VARIANT\)
  - Cutoff frequency.
    - **Default Value**: 20
    - **Suggested values**: `Frequency` \(\in\) \{10, 20, 30, 40, 50, 60, 70, 100\}
    - **Typical range of values**: \(1 \leq Frequency \leq 1\)
    - **Minimum Increment**: 1
    - **Recommended Increment**: 1
    - **Restriction**: \((Frequency > 0)\)

- **Size** (input control) - integer \( \sim long / VARIANT\)
  - Size (dimension) of the image (filter).
    - **Default Value**: 512
    - **List of values**: `Size` \(\in\) \{16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192\}

**Result**

`GenLowpass` returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**

`GenLowpass` is reentrant and processed without parallelization.
**GenLowpass** generates an ideal lowpass filter in the frequency domain. The DC term is assumed to lie in the center of the image. The parameter **Frequency** determines the cutoff frequency of the filter (in pixels). The resulting image contains a circle of radius **Frequency** with the value 255, and the value 0 outside of this circle.

**ImageLowpass** (output iconic) ................. image  ~ HImageX / HUntypedObjectX (byte)  
Highpass filter in the frequency domain.

**Frequency** (input control) .............................................. real  ~ double / VARIANT  
Cutoff frequency.  
Default Value : 20  
Suggested values : Frequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}  
Typical range of values : 1 ≤ Frequency ≤ 1  
Minimum Increment : 1  
Recommended Increment : 1

**Size** (input control) .............................................. integer  ~ long / VARIANT  
Size (dimension) of the image (filter).  
Default Value : 512  
List of values : Size ∈ {16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

**Result**

**GenLowpass** returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**  
**GenLowpass** is reentrant and processed without parallelization.

**Possible Successors**  
ConvolFft  
See also  
GenHighpass, GenBandpass, GenBandfilter  
Alternatives  
GenCircle, PaintRegion  
Module  
Image filters

```c
void HImageX.GenSinBandpass ([in] double Frequency, [in] long Size )

```

Generate a bandpass filter with sinusoidal shape.  
**GenSinBandpass** generates a rotationally invariant bandpass filter with the response being a sinusoidal function in the frequency domain. The maximum of the sine (255) is determined by **Frequency** (distance from the DC term in pixels). The filter is always zero for the DC term, rises with the sine function up to **Frequency**, and drops for higher frequencies accordingly. The range of the sine used is from 0 to π. All other points are set to zero.

**ImageFilter** (output iconic) ................. image(-array)  ~ HImageX / HUntypedObjectX (byte)  
Bandpass filter as image in the frequency domain.

**Frequency** (input control) .............................................. real  ~ double / VARIANT  
Distance of the filter’s maximum from the DC term.  
Default Value : 20  
Suggested values : Frequency ∈ {0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200}  
Typical range of values : 0.0 ≤ Frequency ≤ 0.0  
Minimum Increment : 1  
Recommended Increment : 2

**Size** (input control) .............................................. integer  ~ long / VARIANT  
Size (dimension) of the image (filter).  
Default Value : 512  
List of values : Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}
3.7. FFT

--- Result ---

**GenSinBandpass** returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

--- Parallelization Information ---

**GenSinBandpass** is reentrant and processed without parallelization.

--- Possible Predecessors ---

FftImage, FftGeneric

--- Possible Successors ---

ConvolFft

--- See also ---

FftImageInv

--- Alternatives ---

GenStdBandpass, GenBandpass, GenBandfilter, GenHighpass, GenLowpass

--- Module ---

Image filters

--- Parameters ---

```c

```

Generate a bandpass filter with Gaussian or sinusoidal shape.

**GenStdBandpass** generates a rotationally invariant bandpass filter with the response being determined by the parameters **Frequency** and **Sigma**: **Frequency** determines the location of the maximum response with respect to the DC term, while **Sigma** determines the width of the frequency band that passes the filter. For **Mode** = ’gauss’, a Gaussian response is generated with **Sigma** being the standard deviation. For **Mode** = ’sin’, a sine function is generated with the maximum at **Frequency** and the extent **Sigma**.

### Parameters

- **ImageFilter** (output iconic) — image(-array) ~ HImageX / HUntypedObjectX (byte)  
  Bandpass filter as image in the frequency domain.

- **Frequency** (input control) — real ~ double / VARIANT  
  Distance of the filter’s maximum from the DC term.
  **Default Value**: 20  
  **Suggested values**: Frequency ∈ {0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200}  
  **Typical range of values**: 0.0 ≤ Frequency ≤ 0.0  
  **Minimum Increment**: 1  
  **Recommended Increment**: 2

- **Sigma** (input control) — real ~ double / VARIANT  
  Bandwidth of the filter (standard deviation).
  **Default Value**: 1.0  
  **Suggested values**: Sigma ∈ {0.1, 0.3, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0, 50.0}  
  **Typical range of values**: 0.05 ≤ Sigma ≤ 0.05  
  **Minimum Increment**: 0.001  
  **Recommended Increment**: 0.1

- **Size** (input control) — integer ~ long / VARIANT  
  Size (dimension) of the image (filter).
  **Default Value**: 512  
  **List of values**: Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

- **Mode** (input control) — string ~ String / VARIANT  
  Filter type.
  **Default Value**: ’sin’  
  **List of values**: Mode ∈ {’sin’, ’gauss’}
**GenStdBandpass** returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

**Parallelization Information**

**GenStdBandpass** is reentrant and processed without parallelization.

Possible Predecessors

FftImage, FftGeneric

Possible Successors

ConvolFft, See also

FftImageInv

Alternatives

GenSinBandpass, GenBandpass, GenBandfilter, GenHighpass, GenLowpass

Image filters

```
[out] HImageX ImagePhase HImageX.PhaseDeg ( )
void HOperatorSetX.PhaseDeg ([in] IHObjectX ImageComplex,
[out] HUntypedObjectX ImagePhase )
```

Return the phase of a complex image in degrees.

**PhaseDeg** computes the phase of a complex image in degrees. The following formula is used:

\[
\text{phase} = \frac{180 \tan^{-1} \left( \frac{\text{imaginarypart}}{\text{realpart}} \right)}{\pi}
\]

Parameter

▷ **ImageComplex** (input iconic) ....................... image(-array) $\rightarrow$ HImageX / IHObjectX (complex)
   Input image in frequency domain.

▷ **ImagePhase** (output iconic) ...................... image(-array) $\rightarrow$ HImageX / HUntypedObjectX (direction)
   Phase of the image in degrees.

Result

**PhaseDeg** returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::_’noObjectResult’,<Result>::). If necessary, an exception handling is raised.

**Parallelization Information**

**PhaseDeg** is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessors

FftImage, FftGeneric

Possible Successors

DispImage

See also

FftImageInv

Alternatives

PhaseRad

Module

Image filters

```
[out] HImageX ImagePhase HImageX.PhaseRad ( )
void HOperatorSetX.PhaseRad ([in] IHObjectX ImageComplex,
[out] HUntypedObjectX ImagePhase )
```

Return the phase of a complex image in radians.

HALCON/COM Reference Manual, 2005-2-1
**PhaseRad** computes the phase of a complex image in radians. The following formula is used:

\[
\text{phase} = \tan^{-1}(\frac{\text{imaginarypart}}{\text{realpart}}) .
\]

- **Parameter**
  - **ImageComplex** (input iconic) ...................... image(-array) ➔ HImageX / IObjectX (complex)
    Input image in frequency domain.
  - **ImagePhase** (output iconic) ...................... image(-array) ➔ HImageX / HueTypedObjectX (real)
    Phase of the image in radians.

**PhaseRad** returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception handling is raised.

- **Parallelization Information**
  - **PhaseRad** is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessors**
- FftImage, FftGeneric
**Possible Successors**
- DispImage

**Alternatives**
- PhaseDeg

**Module**
- Image filters

---

**PowerByte** computes the power spectrum from the real and imaginary parts of a Fourier-transformed image (see FftImage), i.e., the modulus of the frequencies. The result image is of type 'byte'. The following formula is used:

\[
\sqrt{\text{realpart}^2 + \text{imaginarypart}^2} .
\]

- **Parameter**
  - **Image** (input iconic) .............................. image(-array) ➔ HImageX / IObjectX (complex)
    Input image in frequency domain.
  - **PowerByte** (output iconic) ...................... image(-array) ➔ HImageX / HueTypedObjectX (byte)
    Power spectrum of the input image.

**PowerByte** returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception handling is raised.

- **Parallelization Information**
  - **PowerByte** is reentrant and automatically parallelized (on tuple level, domain level).

**Possible Predecessors**
- FftImage, FftGeneric, ConvolFft, ConvolGabor
**Possible Successors**
- DispImage
Return the power spectrum of a complex image.

\texttt{PowerLn} computes the power spectrum from the real and imaginary parts of a Fourier-transformed image (see \texttt{FftImage}), i.e., the modulus of the frequencies. Additionally, the natural logarithm is applied to the result. The result image is of type \texttt{'real'}. The following formula is used:

$$\ln\left(\sqrt{\text{realpart}^2 + \text{imaginarypart}^2}\right).$$

\textbf{Parameter}

\textbf{Result}

\texttt{PowerLn} returns \texttt{TRUE} if the image is of correct type. If the input is empty the behaviour can be set via \texttt{SetSystem(::'noObjectResult',<Result>:)}. If necessary, an exception handling is raised.

\textbf{Parallelization Information}

\texttt{PowerLn} is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level, domain level).

\textbf{Possible Predecessors}

\texttt{FftImage, FftGeneric, ConvolFft, ConvolGabor}

\textbf{Possible Successors}

\texttt{DispImage, ConvertImageType, ScaleImage}

\textbf{See also}

\texttt{FftImage, FftGeneric}

\textbf{Module}

\texttt{Image filters}

\texttt{PowerReal} computes the power spectrum from the real and imaginary parts of a Fourier-transformed image (see \texttt{FftImage}), i.e., the modulus of the frequencies. The result image is of type \texttt{'real'}. The following formula is used:

$$\sqrt{\text{realpart}^2 + \text{imaginarypart}^2}.$$
3.8 Lines

### Parameter

- **Image** (input iconic) ................. image(-array) \(\rightarrow\) \(HImageX / IHObjectX\) (complex)
  - Input image in frequency domain.
- **ImageResult** (output iconic) ................. image(-array) \(\rightarrow\) \(HImageX / HUntypedObjectX\) (real)
  - Power spectrum of the input image.

---

### Result

PowerReal returns TRUE if the image is of correct type. If the input is empty the behaviour can be set via `SetSystem::{'noObjectResult',<Result>}`). If necessary, an exception handling is raised.

---

### Parallelization Information

PowerReal is reentrant and automatically parallelized (on tuple level, domain level).

---

### Possible Predecessors

FftImage, FftGeneric, ConvolFft, ConvolGabor

---

### Possible Successors

DispImage, ConvertImageType, ScaleImage

---

### See also

See also FftImage

---

### Alternatives

AbsImage, ConvertImageType, PowerByte, PowerLn

---

### Module

Image filters

---

### 3.8 Lines

**[out] HImageX ImageBandpass**

**HImageX.BandpassImage**

```c
(out) String FilterType )

void **HOperatorSetX.BandpassImage** ([in] IHObjectX Image,

[out] HUntypedObjectX ImageBandpass, [in] VARIANT FilterType)
```

**BandpassImage** serves as an edge filter. It applies a linear filter with the following convolution mask to **Image**:

**FilterType**: 'lines'

In contrast to the edge operator **SobelAmp** this filter detects lines instead of edges, i.e., two closely adjacent edges.

\[
\begin{bmatrix}
0 & -2 & -2 & -2 & 0 \\
-2 & 0 & 3 & 0 & -2 \\
-2 & 3 & 12 & 3 & -2 \\
-2 & 0 & 3 & 0 & -2 \\
0 & -2 & -2 & -2 & 0 \\
\end{bmatrix}
\]

At the border of the image the gray values are mirrored. Over- and underflows of gray values are clipped. The resulting images are returned in **ImageBandpass**.

---

### Attention

---

### Parameter

- **Image** (input iconic) .......... (multichannel-)image(-array) \(\rightarrow\) \(HImageX / IHObjectX\) (byte, uint2)
  - Input images.
- **ImageBandpass** (output iconic) .... (multichannel-)image(-array) \(\rightarrow\) \(HImageX / HUntypedObjectX\) (byte, uint2)
  - Bandpass-filtered images.
Filter type: currently only 'lines' is supported.

Default Value : 'lines'
List of values : FilterType ∈ {'lines'}

BandpassImage returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(::'noObjectResult',<Result>:). If necessary, an exception handling is raised.

Parallelization Information

BandpassImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

Threshold, Skeleton
HighpassImage, GraySkeleton
Alternatives
ConvolImage, TopographicSketch, TextureLaws

Detection of lines using the facet model.

The operator LinesFacet can be used to extract lines (curvilinear structures) from the image Image. The extracted lines are returned in Lines as sub-pixel precise XLD-contours. The parameter LightDark determines, whether bright or dark lines are extracted.

The extraction is done by using the facet model, i.e., a least squares fit, to determine the parameters of a quadratic polynomial in x and y for each point of the image. The parameter MaskSize determines the size of the window used for the least squares fit. Larger values of MaskSize lead to a larger smoothing of the image, but can lead to worse localization of the line. The parameters of the polynomial are used to calculate the line direction for each pixel. Pixels which exhibit a local maximum in the second directional derivative perpendicular to the line direction are marked as line points. The line points found in this manner are then linked to contours. This is done by immediately accepting line points that have a second derivative larger than High. Points that have a second derivative smaller than Low are rejected. All other line points are accepted if they are connected to accepted points by a connected path. This is similar to a hysteresis threshold operation with infinite path length (see HysteresisThreshold). However, this function is not used internally since it does not allow the extraction of sub-pixel precise contours.

The gist of how to select the thresholds in the description of LinesGauss also holds for this operator. A value of Sigma = 1.5 there roughly corresponds to a MaskSize of 5 here.

The extracted lines are returned in a topologically sound data structure in Lines. This means that lines are correctly split at junction points.

LinesFacet defines the following attributes for each line point:

'angle' The angle of the direction perpendicular to the line
'response' The magnitude of the second derivative

These attributes can be queried via the operator GetContourAttribXld.

Attention

The smaller the filter size MaskSize is chosen, the more short, fragmented lines will be extracted. This can lead to considerably longer execution times.
3.8. LINES

Parameter

- **Image** (input iconic) \( \leadsto HImageX / IOobjectX \) (byte, int1, int2, uint2, int4, real)
  Input image.

- **Lines** (output iconic) \( \leadsto HXLDContX / HUntypedObjectX \)
  Extracted lines.

- **MaskSize** (input control) \( \leadsto \) integer \( \leadsto long / VARIANT \)
  Size of the facet model mask.
  **Default Value**: 5
  **List of values**: MaskSize \( \in \{3, 5, 7, 9, 11\} \)

- **Low** (input control) \( \leadsto \) number \( \leadsto VARIANT \) (real, integer)
  Lower threshold for the hysteresis threshold operation.
  **Default Value**: 3
  **Suggested values**: Low \( \in \{0, 0.5, 1, 2, 3, 4, 5, 8, 10\} \)
  **Typical range of values**: \( 0 \leq \text{Low} \leq 0 \)
  **Recommended Increment**: 0.5
  **Restriction**: \((\text{Low} \geq 0)\)

- **High** (input control) \( \leadsto \) number \( \leadsto VARIANT \) (real, integer)
  Upper threshold for the hysteresis threshold operation.
  **Default Value**: 8
  **Suggested values**: High \( \in \{0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25\} \)
  **Typical range of values**: \( 0 \leq \text{High} \leq 0 \)
  **Recommended Increment**: 0.5
  **Restriction**: \((\text{High} \geq 0) \land (\text{High} \geq \text{Low})\)

- **LightDark** (input control) \( \leadsto \) string \( \leadsto String / VARIANT \)
  Extract bright or dark lines.
  **Default Value**: 'light'
  **List of values**: LightDark \( \in \{\text{dark}', \text{light}'\} \)

Example

```c
/* Detection of lines in an aerial image */
read_image(Image,'mreut4_3')
lines_facet(Image,Lines,5,3,8,'light')
disp_xld(Lines,WindowHandle).
```

Complexity

Let \( A \) be the number of pixels in the domain of Image. Then the runtime complexity is \( O(A \times \text{MaskSize}) \).

Let \( S = \text{Width} \times \text{Height} \) be the number of pixels of Image. Then LinesFacet requires at least \( 55 \times S \) bytes of temporary memory during execution.

Result

LinesFacet returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via `SetSystem(:'noObjectResult',<Result>)`. If necessary, an exception handling is raised.

Parallelization Information

LinesFacet is reentrant and processed without parallelization.

Possible Successors

GenPolygonsXld

See also

BandpassImage, DynThreshold, TopographicSketch

Alternatives

LinesGauss

References


Module

Sub-pixel operators
Detect lines and their width.

The operator \texttt{LinesGauss} can be used to extract lines (curvilinear structures) from the image \texttt{Image}. The extracted lines are returned in \texttt{Lines} as sub-pixel precise XLD-contours. The parameter \texttt{LightDark} determines, whether bright or dark lines are extracted. If \texttt{ExtractWidth} is set to \textquote{true} the line width is extracted for each line point. If \texttt{CorrectPositions} is set to \textquote{true}, \texttt{LinesGauss} compensates the effect of asymmetrical lines (lines having different contrast on each side of the line), and corrects the position and width of the line. This parameter is only meaningful if \texttt{ExtractWidth} \textquote{true}. Because the line extractor is unable to extract certain junctions because of differential geometric reasons, it tries to extract these by different means if \texttt{CompleteJunctions} is set to \textquote{true}.

The extraction is done by using partial derivatives of a Gaussian smoothing kernel to determine the parameters of a quadratic polynomial in $x$ and $y$ for each point of the image. The parameter \texttt{Sigma} determines the amount of smoothing to be performed. Larger values of \texttt{Sigma} lead to a larger smoothing of the image, but can lead to worse localization of the line. Generally, the localization will be much better than that of lines returned by \texttt{LinesFacet} with comparable parameters. The parameters of the polynomial are used to calculate the line direction for each pixel. Pixels which exhibit a local maximum in the second directional derivative perpendicular to the line direction are marked as line points. The line points found in this manner are then linked to contours. This is done by immediately accepting line points that have a second derivative larger than \texttt{High}. Points that have a second derivative smaller than \texttt{Low} are rejected. All other line points are accepted if they are connected to accepted points by a connected path. This is similar to a hysteresis threshold operation with infinite path length (see \texttt{HysteresisThreshold}). However, this function is not used internally since it does not allow the extraction of sub-pixel precise contours.

For the choice of the thresholds \texttt{High} and \texttt{Low} one has to keep in mind that the second directional derivative depends on the amplitude and width of the line as well as the choice of \texttt{Sigma}. The value of the second derivative depends linearly on the amplitude, i.e., the larger the amplitude, the larger the response. For the width of the line there is an approximately inverse exponential dependence: The wider the line is, the smaller the response gets. This holds analogously for the dependence on \texttt{Sigma}: The larger \texttt{Sigma} is chosen, the smaller the second derivative will be. This means that for larger smoothing correspondingly smaller values for \texttt{High} and \texttt{Low} have to be chosen. Two examples help to illustrate this: If 5 pixel wide lines with an amplitude larger than 100 are to be extracted from an image with a smoothing of \texttt{Sigma} = 1.5, \texttt{High} should be chosen larger than 14. If, on the other hand, 10 pixel wide lines with an amplitude larger than 100 and a \texttt{Sigma} = 3 are to be detected, \texttt{High} should be chosen larger than 3.5. For the choice of \texttt{Low} values between 0.25 \texttt{High} and 0.5 \texttt{High} are appropriate.

The extracted lines are returned in a topologically sound data structure in \texttt{Lines}. This means that lines are correctly split at junction points.

\texttt{LinesGauss} defines the following attributes for each line point if \texttt{ExtractWidth} was set to \textquote{false}:

- \textquote{angle} The angle of the direction perpendicular to the line
- \textquote{response} The magnitude of the second derivative

If \texttt{ExtractWidth} was set to \textquote{true} and \texttt{CorrectPositions} to \textquote{false}, the following attributes are defined in addition to the above ones:

- \textquote{width_left} The line width to the left of the line
- \textquote{width_right} The line width to the right of the line

Finally, if \texttt{CorrectPositions} was set to \textquote{true}, additionally the following attributes are defined:

- \textquote{asymmetry} The asymmetry of the line point
- \textquote{contrast} The contrast of the line point

Here, the asymmetry is positive if the asymmetric part, i.e., the part with the weaker gradient, is on the right side of the line, while it is negative if the asymmetric part is on the left side of the line. All these attributes can be queried via the operator \texttt{GetContourAttribXld}.

### HALCON/COM Reference Manual, 2005-2-1
Attention

In general, but in particular if the line width is to be extracted, \( \Sigma \geq \frac{w}{\sqrt{3}} \) should be selected, where \( w \) is the width (half the diameter) of the lines in the image. As the lowest allowable value \( \Sigma \geq \frac{w}{2.5} \) must be selected. If, for example, lines with a width of 4 pixels (diameter 8 pixels) are to be extracted, \( \Sigma \geq 2.3 \) should be selected.

Parameter

\( \text{Image} \) (input iconic) ................................. image  \( \sim \) HImageX / IHObjectX (byte, int1, int2, uint2, int4, real)
Input image.

\( \text{Lines} \) (output iconic) .............................. xld_cont  \( \sim \) HXLDContX / HUntypedObjectX
Extracted lines.

\( \text{Sigma} \) (input control) ............................ number  \( \sim \) VARIANT (real, integer)
Amount of Gaussian smoothing to be applied.
Default Value : 1.5
Suggested values : \( \Sigma \in \{1, 1.2, 1.5, 1.8, 2, 2.5, 3, 4, 5\} \)
Typical range of values : \( 0.7 \leq \Sigma \leq 0.7 \)
Recommended Increment : 0.1

\( \text{Low} \) (input control) .............................. number  \( \sim \) VARIANT (real, integer)
Lower threshold for the hysteresis threshold operation.
Default Value : 3
Suggested values : \( \text{Low} \in \{0, 0.5, 1, 2, 3, 4, 5, 8\} \)
Typical range of values : \( 0 \leq \text{Low} \leq 0 \)
Recommended Increment : 0.5
Restriction : \( (\text{Low} \geq 0) \)

\( \text{High} \) (input control) .............................. number  \( \sim \) VARIANT (real, integer)
Upper threshold for the hysteresis threshold operation.
Default Value : 8
Suggested values : \( \text{High} \in \{0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25\} \)
Typical range of values : \( 0 \leq \text{High} \leq 0 \)
Recommended Increment : 0.5
Restriction : \( ((\text{High} \geq 0) \wedge (\text{High} \geq \text{Low})) \)

\( \text{LightDark} \) (input control) ........................ string  \( \sim \) String / VARIANT
Extract bright or dark lines.
Default Value : 'light'
List of values : LightDark \( \in \{\text{dark}', 'light'\} \)

\( \text{ExtractWidth} \) (input control) ........................ string  \( \sim \) String / VARIANT
Should the line width be extracted?
Default Value : 'true'
List of values : ExtractWidth \( \in \{\text{true}', 'false'\} \)

\( \text{CorrectPositions} \) (input control) ........................ string  \( \sim \) String / VARIANT
Should the line position and width be corrected?
Default Value : 'true'
List of values : CorrectPositions \( \in \{\text{true}', 'false'\} \)

\( \text{CompleteJunctions} \) (input control) ........................ string  \( \sim \) String / VARIANT
Should junctions be added where they cannot be extracted?
Default Value : 'true'
List of values : CompleteJunctions \( \in \{\text{true}', 'false'\} \)

Example

/* Detection of lines in an aerial image */
read_image(Image,'mreut4_3')
lines_gauss(Image,Lines,1.5,3,8,'light','true','true','true')
disp_xld(Lines,WindowHandle).

Complexity

Let \( A \) be the number of pixels in the domain of \( \text{Image} \). Then the runtime complexity is \( O(A \times \Sigma) \).
Let \( S = \text{Width} \times \text{Height} \) be the number of pixels of \( \text{Image} \). Then \( \text{LinesGauss} \) requires at least \( 55 \times S \) bytes of temporary memory during execution.
**Result**

*LinesGauss* returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behaviour can be set via `SetSystem(:'noObjectResult',<Result>):`. If necessary, an exception handling is raised.

**Parallelization Information**

*LinesGauss* is reentrant and processed without parallelization.

**Possible Successors**

*GenPolygonsXld*

**See also**

*BandpassImage, DynThreshold, TopographicSketch*

**Alternatives**

*LinesFacet*

**References**


**Module**

Sub-pixel operators

### 3.9 Match

|---------------------------|----------------------------------------------------------|

Searching corners in images.

The operator *CornerResponse* extracts gray value corners in an image. The formula for the calculation of the response is:

\[
R(x, y) = A(x, y) \cdot B(x, y) - C^2(x, y) - \text{Weight} \cdot (A(x, y) + B(x, y))^2 \\
A(x, y) = W(u, v) \ast (\nabla_x I(x, y))^2 \\
B(x, y) = W(u, v) \ast (\nabla_y I(x, y))^2 \\
C(c, y) = W(u, v) \ast (\nabla_x I(x, y) \nabla_y I(x, y))
\]

where \( I \) is the input image and \( R \) the output image of the filter. The operator *GaussImage* is used for smoothing \((W)\), the operator *SobelAmp* is used for calculating the derivative \((\nabla)\).

The corner response function is invariant with regard to rotation. In order to achieve a suitable dependency of the function \( R(x, y) \) on the local gradient, the parameter Weight must be set to 0.04. With this, only gray value corners will return positive values for \( R(x, y) \), while straight edges will receive negative values. The output image type is identical to the input image type. Therefore, the negative output values are set to 0 if byte images are used as input images. If this is not desired, the input image should be converted into a real or int2 image with *ConvertImageType*. 

HALCON/COM Reference Manual, 2005-2-1
3.9. MATCH

Parameter

- **Image** (input iconic) .......... (multichannel-)image(-array)  \sim  HImageX / HObjectX (byte, int2, real) 
  Input image.
- **ImageCorner** (output iconic) .... multichannel-image-array  \sim  HImageX / HTypedObjectX (byte, int2, real) 
  Result of the filtering.

**Number of elements** : \((\text{ImageCorner} = \text{Image})\)

- **Size** (input control) ............................................. integer  \sim  long / VARIANT 
  Desired filtersize of the graymask.
  **Default Value** : 3
  **List of values** : \(\text{Size} \in \{3, 5, 7, 9, 11\}\)

- **Weight** (input control) ............................................. real  \sim  double / VARIANT 
  Weighting.
  **Default Value** : 0.04
  **Typical range of values** : \(0.0 \leq \text{Weight} \leq 0.0\)
  **Minimum Increment** : 0.001
  **Recommended Increment** : 0.01

Parallelization Information

**CornerResponse** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

LocalMax, Threshold

See also

GaussImage, SobelAmp, ConvertImageType

References


Module

Image filters

Matching of a template and an image.

The operator **ExhaustiveMatch** matches **ImageTemplate** and **Image** within the region of interest **RegionOfInterest**. Hereby the **ImageTemplate** will be moved over all points of **Image** which lie within the **RegionOfInterest**. With regard to the parameter **Mode**, a matching criterion will be calculated. The result values will be stored in **ImageMatch**.

The following matching criteria (**Mode**) are available:

- **norm_correlation**

  \[
  \text{ImageMatch}[i][j] = 255 \cdot \frac{\sum_{u,v} (\text{Image}[i-u][j-v] \cdot \text{ImageTemplate}[l-u][c-v])}{\sqrt{\sum_{u,v} (\text{Image}[i-u][j-v]^2) \cdot \sum_{s,t} (\text{ImageTemplate}[l-u][c-v]^2)}}
  \]

  whereby \(X[i][j]\) indicates the grayvalue in the \(i\)th column and \(j\)th row of the image \(X\). \((l,c)\) is the centre of the region of **ImageTemplate**. \(u\) and \(v\) are chosen so that all points of the template will be reached, \(i, j\) run across the **RegionOfInterest**. At the image frame only those parts of **ImageTemplate** will be considered which lie inside the image (i.e. \(u\) and \(v\) will be restricted correspondingly). Range of values: 0 - 255 (best fit).
'dfd' Calculating the average “displaced frame difference”:

\[
\text{ImageMatch}[i][j] = \frac{\sum_{u,v} |\text{Image}[i-u][j-v] - \text{ImageTemplate}[l-u][c-v]|}{\text{AREA(ImageTemplate)}}
\]

The terms are the same as in 'norm correlation'. AREA (X) means the area of the region X. Range of value 0 (best fit) - 255.

To calculate the normalized correlation as well as the “displaced frame difference” is (with regard to the area of ImageTemplate) very time consuming. Therefore it is important to restrict the input region (RegionOfInterest if possible, i.e. to apply the filter only in a very confined “region of interest”.

As far as quality is concerned, both modes return comparable results, whereby the mode 'dfd' is faster by a factor of about 3.5.

Parameter

- **Image** (input iconic) ............................... image \( \sim \text{HImageX} / \text{IHObjectX} \) (byte) Input image.
- **RegionOfInterest** (input iconic) ...................... region \( \sim \text{HRegionX} / \text{IHObjectX} \) Area to be searched in the input image.
- **ImageTemplate** (input iconic) ........................ image \( \sim \text{HImageX} / \text{IHObjectX} \) (byte) This area will be “matched” by Image within the RegionOfInterest.
- **ImageMatch** (output iconic) ............................ image \( \sim \text{HImageX} / \text{HUntypedObjectX} \) (byte) Result image: values of the matching criterion.
- **Mode** (input control) ................................. string \( \sim \text{String} / \text{VARIANT} \) Desired matching criterion.
  Default Value: 'dfd'
  List of values: Mode \( \in \{ \text{norm correlation}, \text{dfd} \} \)

Example

```plaintext
read_image(Image,‘monkey’)
disp_image(Image,WindowHandle)
draw_rectangle2(WindowHandle,Row,Column,Phi,Length1,Length2)
gen_rectangle2(Rectangle,Row,Column,Phi,Length1,Length2)
reduce_domain(Image,Rectangle,Template)
exhaustive_match(Image,Image,Template,ImageMatch,’dfd’)
invert_image(ImageMatch,ImageInvert)
local_max(Image,Maxima)
union1(Maxima,AllMaxima)
add_channels(AllMaxima,ImageInvert,FitMaxima)
threshold(FitMaxima,BestFit,230.0,255.0)
disp_region(BestFit,WindowHandle).
```

Result

If the parameter values are correct, the operator ExhaustiveMatch returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem (‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

ExhaustiveMatch is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

DrawRegion, DrawRectangle1

Possible Successors

LocalMax, Threshold

Alternatives

ExhaustiveMatchMg

Module

Image filters

HALCON/COM Reference Manual, 2005-2-1
Matching a template and an image in a resolution pyramid.

The operator \texttt{ExhaustiveMatchMg} is an additional option for the operator \texttt{ExhaustiveMatch} performing a matching of the image \texttt{Image} and the template \texttt{ImageTemplate}. Hereby \texttt{ImageTemplate} will be moved over all points of the region of \texttt{Image}, a matching criterion will be calculated with regard to the parameter \texttt{Mode} and the result values will be stored in \texttt{ImageMatch}.

Of images having been filtered this way, normally only those areas with good matching results are of interest. The size of the area to be searched, i.e. the region of the input image \texttt{Image}, determines decisively the runtime of the matching filter. Therefore it is recommendable to use at first \texttt{ExhaustiveMatchMg} with reduced image resolution in order to determine a “region of interest” in which good matching results can be expected; then in this restricted area only the real matching (see also \texttt{ExhaustiveMatch}) will be executed with normal resolution.

Hereby the Gauss-pyramids of \texttt{Image} and \texttt{ImageTemplate} will be composed (in particular the corresponding regions will be transformed as well). Then on each level of the resolution pyramids - starting with the startlevel \texttt{Level} - the matching inside the current “region of interest” will be executed. Whereby the “region of interest” on the startlevel is equivalent to the region of the input image \texttt{Image}. After the filtering, a new “region of interest” is determined with the help of a threshold operation and will be transformed on the next resolution level:

\begin{itemize}
  \item \texttt{Threshold}(.0,\texttt{Threshold}..), if \texttt{Mode} = ‘dfd’
  \item \texttt{Threshold}(.\texttt{Threshold},255..), if \texttt{Mode} = ‘crosscorrelation’
\end{itemize}

The final matching in the determined “region of interest” will then be calculated with the highest resolution (\texttt{Level 0}). The output image \texttt{ImageMatch} includes the corresponding filter result and the final “region of interest”, which is determined on the result image with the help of a threshold operation.

The operator \texttt{ExhaustiveMatchMg} therefore is not simply a filter, but can also be considered as a member of the class of region transformations.

\begin{center}
\begin{tabular}{|l|}
\hline
Parameter \\
\hline
\end{tabular}
\end{center}

- \texttt{Image} (input iconic) \hspace{1cm} \texttt{image(-array)} \rightarrow \texttt{HImageX / IHOBJECTX (byte)}
  Input image.

- \texttt{ImageTemplate} (input iconic) \hspace{1cm} \texttt{image} \rightarrow \texttt{HImageX / IHOBJECTX (byte)}
  The domain of this image will be matched with \texttt{Image}.

- \texttt{ImageMatch} (output iconic) \hspace{1cm} \texttt{image(-array)} \rightarrow \texttt{HImageX / HUntypedObjectX (byte)}
  Result image and result region: values of the matching criterion within the determined “region of interest”.

  \begin{itemize}
  \item \texttt{Number of elements} : \texttt{(ImageMatch = Image)}
  \end{itemize}

- \texttt{Mode} (input control) \hspace{1cm} \texttt{string} \rightarrow \texttt{String / VARIANT}
  Desired matching criterion.

  \begin{itemize}
  \item \texttt{Default Value} : ‘dfd’
  \item \texttt{List of values} : \texttt{Mode} \in \{‘crosscorrelation’, ‘dfd’\}
  \end{itemize}

- \texttt{Level} (input control) \hspace{1cm} \texttt{integer} \rightarrow \texttt{long / VARIANT}
  Startlevel in the resolution pyramid (highest resolution: \texttt{Level 0}).

  \begin{itemize}
  \item \texttt{Default Value} : 1
  \item \texttt{List of values} : \texttt{Level} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8\}
  \item \texttt{Restriction} : \texttt{(Image \& (height < Id))}
  \end{itemize}

- \texttt{Threshold} (input control) \hspace{1cm} \texttt{integer} \rightarrow \texttt{long / VARIANT}
  Threshold to determine the “region of interest”.

  \begin{itemize}
  \item \texttt{Default Value} : 30
  \item \texttt{Suggested values} : \texttt{Threshold} \in \{5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125, 130, 135, 140, 145, 150, 155, 160, 165, 170, 175, 180, 185, 190, 195, 200, 205, 210, 215, 220, 225, 230, 235, 240, 245, 250\}
  \item \texttt{Typical range of values} : \texttt{0 \leq Threshold \leq 0}
  \item \texttt{Minimum Increment} : 1
  \item \texttt{Recommended Increment} : 5
  \end{itemize}
If the parameter values are correct, the operator `ExhaustiveMatchMg` returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via `SetSystem` (`'noObjectResult',<Result>`). If necessary, an exception handling is raised.

**Parallelization Information**

`ExhaustiveMatchMg` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

`DrawRegion`, `DrawRectangle1`

**Possible Successors**

`Threshold`, `LocalMax`

See also `GenGaussPyramid`

**Alternatives**

`ExhaustiveMatch` Module

---

**Image filters**

```plaintext

```

Expansion of a displacement vector field onto undefined areas.

The operator `FillDvf` calculates a value for each undefined point in a displacement vector field by averaging its neighbors. The dimension of the neighborhod will be determined by the parameters `Width` and `Height`. As the gaps may be larger than the masks, the filter may be used iterat ively (`Iterations`). Correctly defined values in the input image will not be modified.

**Parameter**

- **Image** (input iconic) ................................. image(-array)  ـ  `HImageX / IObjectX ( dvf )`
  Input image.

- **ImageExpanded** (output iconic) ................. image(-array)  ـ  `HImageX / HUntypedObjectX ( dvf )`
  Result of adding the points which are missing.

- **Width** (input control) ................................. extent.x  ـ  long / VARIANT
  Width of the filtermask.
  **Default Value :** 5
  **Suggested values :** `Width ∈ {3, 5, 7, 9, 11, 15}`
  **Typical range of values :** `3 ≤ Width ≤ 3`
  **Minimum Increment :** 2
  **Recommended Increment :** 2

- **Height** (input control) ................................. extent.y  ـ  long / VARIANT
  Height of the filtermask.
  **Default Value :** 5
  **Suggested values :** `Height ∈ {3, 5, 7, 9, 11, 13, 15}`
  **Typical range of values :** `3 ≤ Height ≤ 3`
  **Minimum Increment :** 2
  **Recommended Increment :** 2

- **Iterations** (input control) ............................ integer  ـ  long / VARIANT
  Number of iterations.
  **Default Value :** 3
  **Suggested values :** `Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}`
  **Typical range of values :** `1 ≤ Iterations ≤ 1`
  **Minimum Increment :** 1
  **Recommended Increment :** 1

HALCON/COM Reference Manual, 2005-2-1
Parallelization Information

*FillDvf* is *reentrant* and automatically *parallelized* (on tuple level, domain level).

Possible Predecessors

OpticalFlowMatch

Module

Image filters

```plaintext

```

Calculating a Gauss pyramid.

The operator *GenGaussPyramid* calculates a pyramid of scaled down images. The scale by which the next image will be reduced is determined by the parameter *Scale*. For instance, a value of 0.5 for *Scale* will shorten the edge length of *Image* by 50%. This is exactly equivalent to the "normal" pyramid.

The parameter *Mode* determines the way of averaging. For a more detailed description concerning this parameter see also *AffineTransImage*. In the case that *Scale* is equal 0.5 there are the additional modes ‘min’ and ‘max’ available. In this case the minimum or the maximum of the four neighboring pixels is selected.

Please note that each level will be returned as an individual image, i.e. as one iconic object, with one matrix and its own domain. If a multichannel image is needed as a result, the operator *ImageToChannels* has to be used. A single level or more than one level can be selected by using *SelectObj* respectively *CopyObj*.

Parameter

- **Image** (input iconic) ......................... image ～ HImageX / IHObjectX ( byte, uint2, real )
  Input image.
- **ImagePyramid** (output iconic) .............. image ～ HImageX / HUntypedObjectX ( byte, uint2, real )
  Output images.
- **Mode** (input control) ......................... string ～ String / VARIANT
  Kind of filtermask.
  Default Value : ‘weighted’
  List of values : Mode ∈ { ‘none’, ‘constant’, ‘weighted’, ‘min’, ‘max’ }
- **Scale** (input control) ......................... real ～ double / VARIANT
  Factor for scaling down.
  Default Value : 0.5
  Suggested values : Scale ∈ { 0.2, 0.3, 0.4, 0.5, 0.6 }
  Typical range of values : 0.1 ≤ Scale ≤ 0.1
  Minimum Increment : 0.01
  Recommended Increment : 0.1
  Restriction : ((0.1 < Scale) ∧ (Scale < 0.9))

Parallelization Information

*GenGaussPyramid* is *reentrant* and automatically *parallelized* (on tuple level).

Possible Successors

ImageToChannels, CountObj, SelectObj, CopyObj

See also

*AffineTransImage*

Alternatives

ZoomImageSize, ZoomImageFactor

Module

Image filters
Calculating the monotony operation.

The operator `Monotony` calculates the monotony operator. Thereby the points which are strictly smaller than the current grayvalue will be counted in the 8 neighborhood. This number will be entered into the output image.

If there is a strict maximum, the value 8 is returned; in case of a minimum or a plateau, the value 0 will be returned. A ridge or a slope will return the corresponding intermediate values.

The monotony operator is often used to prepare matching operations as it is invariant with regard to lightness modifications.

**Parameter**

- **Image** (input iconic) ...............(multichannel-image-array) \(\sim\) HImageX / IHObjectX (byte, int2, uint2)
  
  Input image.

- **ImageMonotony** (output iconic) ...... (multichannel-image-array) \(\sim\) HImageX / HUntypedObjectX (byte, int2, uint2)
  
  Result of the monotony operator.

**Number of elements:** \(\text{ImageMonotony} = \text{Image}\)

**Parallelization Information**

Monotony is reentrant and automatically parallelized (on tuple level, channel level, domain level).

**Possible Predecessors**

GaussImage, MedianImage, MeanImage, SmoothImage, InvertImage

**Possible Successors**

Threshold, ExhaustiveMatch, DispImage

**Alternatives**

LocalMax, TopographicSketch, CornerResponse

**Module**

Image filters

Calculating the Displacement vector field by correlation methods.

The operator `OpticalFlowMatch` determines a displacement vector field (DVF) with the help of correlation methods. Simple- and multichannel Images may be used. A window of the size `SizeWindow` will be moved inside a mask of the size `SizeSearch` over a second image and there the grayvalues will be compared with the ones from the first image. The correlation showing the slightest error will determine the vector. In case of two vectors having the same weighting, the shorter vector will be prefered. After having processed one image point, the next image point at a distance of `Step` will be selected. It is therefore not guaranteed to get a dense displacement vector field.

The parameter `Threshold` specifies the maximal divergence of previously estimated correlations and the best estimation of a displacement vector.

The method used by the operator is determined by the parameter `Mode`. The following values for `Mode` can be selected:

- `dden` Calculation of displaced frame difference.
‘dfd_norm’ Calculation of displaced frame difference normalized by the mean value of the matching windows.

If the image has more than one channel the parameter Weights indicates the weighting of the individual channels. For each channel one weight factor must be passed.

For the parameters SizeWindow, SizeSearch and Step optionally two values can be passed: The first value indicates hereby the width (respectively the row), the second value the height (respectively the column).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image1</td>
<td>image(-array)</td>
<td>First input image (optionally with more than one channel).</td>
</tr>
<tr>
<td>Image2</td>
<td>image(-array)</td>
<td>Second input image (optionally with more than one channel).</td>
</tr>
<tr>
<td>VectorField</td>
<td>image(-array)</td>
<td>Displacement vector field to be calculated.</td>
</tr>
<tr>
<td>Mode</td>
<td>string</td>
<td>Kind of correlation. Default Value: ‘dfd’</td>
</tr>
<tr>
<td>Threshold</td>
<td>long</td>
<td>Maximal divergence of the previously estimated correlations and the best estimate of a displacement vector. Default Value: 10</td>
</tr>
<tr>
<td>Step</td>
<td>integer</td>
<td>Step width. Default Value: 10</td>
</tr>
<tr>
<td>SizeWindow</td>
<td>integer</td>
<td>Size of the correlation window. Default Value: 11</td>
</tr>
<tr>
<td>SizeSearch</td>
<td>integer</td>
<td>Size of the area to be searched. Default Value: 20</td>
</tr>
<tr>
<td>Weights</td>
<td>integer</td>
<td>Weighting of the channels. Default Value: 1</td>
</tr>
</tbody>
</table>

**Example**

```plaintext```
read_image(B1,'image1')
```plaintext```
mean_image(B1,L1,16,16)
read_image(B2,'image2')
mean(B2,L2,16,16)
optical_flow_match(L1,L2,VVF,'dfd',10,10,11,20,1)
disp_image(B1,WindowHandle)
set_color(WindowHandle,'red')
disp_image(VVF,WindowHandle).

Parallelization Information
OpticalFlowMatch is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
MeanImage, GaussImage, SmoothImage

Possible Successors
FillDvf
See also
DvfToInt1, DvfToHomMat2d

Module

3.10 Misc

Convolve an image with an arbitrary filter mask.

ConvolveImage convolves the input image Image with an arbitrary linear filter. The corresponding filter mask, which is given in FilterMask can be generated either from a file or a tuple. Several options for the treatment at the image’s borders can be chosen (Margin):

- gray value
- ‘continued’
- ‘cyclic’
- ‘mirrored’

Pixels outside of the image edges are assumed to be constant (with the indicated gray value).
Continuation of edge pixels.
Cyclic continuation of image edges.
Reflection of pixels at the image edges.

All image points are convolved with the filter mask. If an overflow or underflow occurs, the resulting gray value is clipped. Hence, if filters that result in negative output values are used (e.g., derivative filters) the input image should be of type int2. If a filename is given in FilterMask the filter mask is read from a text file with the following structure:

- ⟨Mask size⟩
- ⟨Inverse weight of the mask⟩
- ⟨Matrix⟩

The first line contains the size of the filter mask, given as two numbers separated by white space (e.g., 3 3 for \(3 \times 3\)). Here, the first number defines the height of the filter mask, while the second number defines its width. The next line contains the inverse weight of the mask, i.e., the number by which the convolution of a particular image point is divided. The remaining lines contain the filter mask as integer numbers (separated by white space), one line of the mask per line in the file. The file must have the extension “.fil”. This extension must not be passed to the operator. If the filter mask is to be computed from a tuple, the tuple given in FilterMask must also satisfy the structure described above. However, in this case the line feed is omitted.
Image (input iconic) .......... (multichannel-)image(-array) ~ HImageX / IHObjectX ( byte, int2, uint2 ) Image to be convolved.

ImageResult (output iconic) .......... multichannel-image-array ~ HImageX / HUntypedObjectX ( byte, int2, uint2 )

Convolved result image.

FilterMask (input control) ......................... string(-array) ~ VARIANT ( integer, string ) Filter mask as file name or tuple.

Default Value : 'sobel'

Suggested values : FilterMask ∈ {'sobel', 'laplace4', 'lowpas_3_3'}

Margin (input control) ......................... string ~ VARIANT ( integer, real, string ) Border treatment.

Default Value : 'mirrored'

Suggested values : Margin ∈ {'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255}

Parallelization Information

ConvolImage is reentrant and automatically parallelized (on tuple level, channel level).

Module

Image filters

```
[out] HImageX ExpandedImage HImageX.ExpandDomainGray
([in] long ExpansionRange )

void HOperatorSetX.ExpandDomainGray ([in] IHObjectX InputImage,
```

Expand the domain of an image and set the gray values in the expanded domain.

ExpandDomainGray expands the border gray values of the domain outwards. The width of the expansion is set by the parameter ExpansionRange. All filters in HALCON use gray values of the pixels outside the domain depending on the filter width. This may lead to undesirable side effects especially in the border region of the domain. For example, if the foreground (domain) and the background of the image differ strongly in brightness, the result of a filter operation may lead to undesired darkening or brightening at the border of the domain. In order to avoid this drawback, the domain is expanded by ExpandDomainGray in a preliminary stage, copying the gray values of the border pixels to the outside of the domain. In addition, the domain itself is also expanded to reflect the newly set pixels. Therefore, in many cases it is reasonable to reduce the domain again (ReduceDomain or ChangeDomain) after using ExpandDomainGray and call the filter operation afterwards. ExpansionRange should be set to the half of the filter width.

```
InputImage (input iconic) .......... image(-array) ~ HImageX / IHObjectX ( byte, int1, int2, uint2, int4, real ) Input image with domain to be expanded.

ExpandedImage (output iconic) .......... image(-array) ~ HImageX / HUntypedObjectX ( byte, int1, int2, uint2, int4, real ) Output image with new gray values in the expanded domain.

ExpansionRange (input control) ......................... integer ~ long / VARIANT Radius of the gray value expansion, measured in pixels.

Default Value : 2

Suggested values : ExpansionRange ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16}

Restriction : (ExpansionRange ≥ 1)
```

Example

read_image(Fabrik, 'fabrik.tif');
gen_rectangle2(Rectangle_Label,243,320,-1.55,62,28);
reduce_domain(Fabrik, Rectangle_Label, Fabrik_Label);
/* Character extraction without gray value expansion: */
mean_image(Fabrik_Label,Label_Mean_normal,31,31);
CHAPTER 3. FILTER

dyn_threshold(Fabrik_Label, Label_Mean_normal, Characters_normal, 10, 'dark');
devel_display(Fabrik);
devel_display(Characters_normal);
/* The characters in the border region are not extracted */
stop();
/* Character extraction with gray value expansion: */
expand_domain_gray(Fabrik_Label, Label_expanded, 15);
reduce_domain(Label_expanded, Rectangle_Label, Label_expanded_reduced);
mean_image(Label_expanded_reduced, Label_Mean_expanded, 31, 31);
dyn_threshold(Fabrik_Label, Label_Mean_expanded, Characters_expanded, 10, 'dark');
devel_display(Fabrik);
devel_display(Characters_expanded);
/* Now, even in the border region the characters are recognized */

Complexity

Let $L$ the perimeter of the domain. Then the runtime complexity is approximately $O(L) \times \text{ExpansionRange}$.

Result

ExpandDomainGray returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

Parallelization Information

ExpandDomainGray is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

ReduceDomain

Possible Successors

ReduceDomain, MeanImage, DynThreshold

See also

ReduceDomain, MeanImage

Module

Image filters

[out] HImageX ImageDist HImageX.GrayInside ( )

void HOperatorSetX.GrayInside ([in] IObjectX Image,
[out] HUntypedObjectX ImageDist )

Calculate the lowest possible gray value on an arbitrary path to the image border for each point in the image.

GrayInside determines the “cheapest” path to the image border for each point in the image, i.e., the path on which the lowest gray values have to be overcome. The resulting image contains the difference of the gray value of the particular point and the maximum gray value on the path. Bright areas in the result image therefore signify that these areas (which are typically dark in the original image) are surrounded by bright areas. Dark areas in the result image signify that there are only small gray value differences between them and the image border (which doesn’t mean that they are surrounded by dark areas; a small “gap” of dark values suffices). The value 0 (black) in the result image signifies that only darker or equally bright pixels exist on the path to the image border.

The operator is implemented by first segmenting into basins and watersheds the image using the Watersheds operator. If the image is regarded as a gray value mountain range, basins are the places where water accumulates and the mountain ridges are the watersheds. Then, the watersheds are distributed to adjacent basins, thus leaving only basins. The border of the domain (region) of the original image is now searched for the lowest gray value, and the region in which it resides is given its result values. If the lowest gray value resides on the image border, all result values can be calculated immediately using the gray value differences to the darkest point. If the smallest found gray value lies in the interior of a basin, the lowest possible gray value has to be determined from the already processed adjacent basins in order to compute the new values. An 8-neighborhood is used to determine adjacency. The found region is subtracted from the regions yet to process, and the whole process is repeated. Thus, the image is “stripped” form the outside.

Analogously to Watersheds, it is advisable to apply a smoothing operation before calling Watersheds, e.g., GaussImage, in order to reduce the amount of regions that result from the watershed algorithm, and thus to speed up the processing time.
### Image filters

#### GraySkeleton

**Thinning of gray value images.**

GraySkeleton applies a gray value thinning operation to the input image `Image`. Figuratively, the gray value “mountain range” is reduced to its ridge lines by setting the gray value of “hillsides” to the gray value at the corresponding valley bottom. The resulting ridge lines are at most two pixels wide. This operator is especially useful for thinning edge images, and is thus an alternative to NonmaxSuppressionAmp. In contrast to NonmaxSuppressionAmp, GraySkeleton preserves contours, but is much slower. In contrast to Skeleton, this operator changes the gray values of an image while leaving its region unchanged.

**Parameter**

- **Image** (input iconic) ................. (multichannel-)image(-array)  \(\sim\) HImageX / IHObjectX (byte)
  Image to be thinned.
- **GraySkeleton** (output iconic) ... (multichannel-)image(-array)  \(\sim\) HImageX / HUntypedObjectX (int2)
  Thinned image.

**Example**

```c
/* Seeking leaves of a beech tree in an aerial picture: */
read_image(Image,'wald1')
grey_skeleton(Image,Skelett)
mean_image(Skelett,MeanSkelett,7,7)
dyn_threshold(Skelett,MeanSkelett,Leafs,3,'light').
```

---

**HALCON 6.1.4**
CHAPTER 3. FILTER

GraySkeleton returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception is raised.

Parallelization Information

GraySkeleton is reentrant and automatically parallelized (on tuple level, channel level).

Possible Successors

MeanImage

See also

Skeleton,GrayDilationRect

Alternatives

NonmaxSuppressionAmp,NonmaxSuppressionDir,LocalMax

Module

Image filters

Transform an image with a gray-value look-up-table

LutTrans transforms an image Image by using a gray value look-up-table Lut. This table acts as a transformation function. In the case of byte-images, Lut has to be a tuple of length 256. In the case of int2-images, Lut has to be a tuple of length 256 := length := 65536. If the length of the Lut is := 32768, the transformation is applied to the positive gray values only, i.e., the first element of the Lut specifies the new gray value for the gray value 0. If the Lut is longer than 32768, exactly 65536 must be passed. In this case, the positive and negative gray values are transformed. In this case, the first element indicates the new gray value for the gray value -32768 of the input image, while the last element of the tuple indicates the new gray value for the gray value 32767. In all cases, the gray values of values outside the range of Lut are set to 0. In the case of uint2-images, Lut has to be a tuple of length 256 := length := 65536. Gray values outside the range of Lut are set to 0.

Parameter

- **Image** (input iconic) ..........(multichannel)-image(-array)  ~ HImageX / IHOBJECTX ( byte, int2, uint2 )
  
  Image whose gray values are to be transformed.

- **ImageResult** (output iconic) ...... (multichannel)-image(-array)  ~ HImageX / HUntypedObjectX ( byte, int2, uint2 )
  
  Transformed image.

- **Lut** (input control) .................................integer  ~ VARIANT ( integer )
  
  Table containing the transformation.

Example

/* To get the inverse of an image: */
read_image(Image,’wald1’)
def_tab(Tab,0)
lut_trans(Image,Invers,Tab,1,1)

def_tab(Tab,I) :- I=255
  Tab = 0
def_tab([Tk|Ts],I) :-
  Tk is 255 - I
  Iw is I -1
  def_tab(Ts,Iw)

Result

The operator LutTrans returns the value TRUE if the parameters are correct. Otherwise an exception is raised.
Parallelization Information
LutTrans is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Module

Image filters

[out] HImageX ImageMapped
HImageX.MapImage ([in] HImageX Map)

void HOperatorSetX.MapImage ([in] IHObjectX Image, [in] IHObjectX Map,
[out] HUntypedObjectX ImageMapped)

Apply a general transformation to an image.
MapImage transforms an image Image using an arbitrary transformation Map, which, for example, was previously generated using GenImageToWorldPlaneMap or GenRadialDistortionMap. The multi-channel image Map must be organized as follows:
The height and the width of Map define the size of the output image ImageMapped. The number of channels in Map defines whether no interpolation or bilinear interpolation should be used. If Map only consists of one channel, no interpolation is applied during the transformation. This channel contains ‘int4’ values that describe the geometric transformation: For each pixel in the output image ImageMapped the linearized coordinate of the pixel in the input image Image from which the gray value should be taken is stored.
If bilinear interpolation between the pixels in the input image should be applied, Map must consist of 5 channels. The first channel again consists of an ‘int4’ image and describes the geometric transformation. The channels 2-5 consist of an ‘uint2’ image each and contain the weights \([0...1]\) of the four neighboring pixels that are used during bilinear interpolation. If the overall brightness of the output image ImageMapped should not differ from the overall brightness of the input image Image, the sum of the four unscaled weights must be 1 for each pixel. The weights \([0...1]\) are scaled to the range of values of the ‘uint2’ image and therefore hold integer values from 0 bis 65535.

Furthermore, the weights must be chosen in a way that the range of values of the output image ImageMapped is not exceeded. The geometric relation between the four channels 2-5 is illustrated in the following sketch:

```
  2
  3
  4
  5
```
The reference point of the four pixels is the upper left pixel. The linearized coordinate of the reference point is stored in the first channel.

Attention
The weights must be chosen in a way that the range of values of the output image ImageMapped is not exceeded. For runtime reasons during the mapping process, it is not checked whether the linearized coordinates, which are stored in the first channel of Map, lie inside the input image. Thus, it must be ensured by the user that this constraint is fulfilled. Otherwise, the program may crash!

Parameter

▷ Image (input iconic) ……….. (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, uint2)
Image to be mapped.
▷ Map (input iconic) ………….. (multichannel-)image \(\sim\) HImageX / IHObjectX (int4, uint2)
Image containing the mapping data.
▷ ImageMapped (output iconic) … (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, uint2)
Mapped image.

Result
MapImage returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information
MapImage is reentrant and processed without parallelization.

Possible Predecessors
GenImageToWorldPlaneMap, GenRadialDistortionMap

See also
AffineTransImage, RotateImage

HALCON 6.1.4
CHAPTER 3. FILTER

Image filters

Module

Image filters

[out] HImageX PCAImage HImageX.PrincipalComp

(void) HOperatorSetX.PrincipalComp ([in] IObjectX MultichannelImage,
[out] HUntypedObjectX PCAImage, [out] VARIANT InfoPerComp)

Compute the principal components of multi-channel images.
PrincipalComp does a principal components analysis of multi-channel images. This is useful for images obtained, e.g., with the thematic mapper of the Landsat satellite. Because the spectral bands are highly correlated it is desirable to transform them to uncorrelated images. This can be used to save storage, since the bands containing little information can be discarded, and with respect to a later classification step.

The operator PrincipalComp takes a (multi-channel) image MultichannelImage and transforms it to the output image PCAImage, which contains the same number of channels, using the principal components analysis. The parameter InfoPerComp contains the relative information content of each output channel.

Parameter

- MultichannelImage (input iconic) ........................................... multichannel-image-array -> HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  Multi-channel input image.
- PCAImage (output iconic) .................................................. multichannel-image-array -> HImageX / HUntypedObjectX (real)
  Multi-channel output image.
- InfoPerComp (output control) ............................................. real(-array) -> VARIANT (real)
  Information content of each output channel.

Result

The operator PrincipalComp returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

Parallelization Information

PrincipalComp is reentrant and processed without parallelization.

Module

Image filters

[out] HImageX ImageSymmetry HImageX.Symmetry


void HOperatorSetX.Symmetry ([in] IObjectX Image,
[out] HUntypedObjectX ImageSymmetry, [in] VARIANT MaskSize,

Symmetry of gray values along a row.
Symmetry calculates the symmetry along a line. For each pixel the gray values of both sides of the line are compared: The absolute value of the differences of gray values with same distance to the pixel is computed. Each of these differences is weighted by the exponent (after division by 255) and the summed up.

\[
\text{sym} := 255 - \frac{255}{\text{MaskSize}} \sum_{i=1}^{\text{MaskSize}} \left( \frac{|g(i) - g(-i)|}{255} \right)^{\text{Exponent}}
\]

Pixels with a high symmetry have large gray values.

Attention

Currently only horizontal search lines are implemented.
### Parameter

- **Image** (input iconic) .................................. (multichannel-)image(-array) \( \sim \) HImageX / HObjectX (byte)  
  Input image.
- **ImageSymmetry** (output iconic) ............ (multichannel-)image(-array) \( \sim \) HImageX / HUntypedObjectX (byte)  
  Symmetry image.
- **MaskSize** (input control) ................................ number \( \sim \) long / VARIANT  
  Extension of search area.  
  Default Value: 40  
  Suggested values: \( \text{MaskSize} \in \{3, 5, 7, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 100, 120, 140, 180\} \)  
  Typical range of values: \( 3 \leq \text{MaskSize} \leq 3 \)  
  Minimum Increment: 1  
  Recommended Increment: 2
- **Direction** (input control) ................................ number \( \sim \) double / VARIANT  
  Angle of test direction.  
  Default Value: 0.0  
  Suggested values: \( \text{Direction} \in \{0.0\} \)  
  Typical range of values: \( 0.0 \leq \text{Direction} \leq 0.0 \)
- **Exponent** (input control) ................................ number \( \sim \) double / VARIANT  
  Exponent for weighting.  
  Default Value: 0.5  
  Suggested values: \( \text{Exponent} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1.0\} \)  
  Typical range of values: \( 0.05 \leq \text{Exponent} \leq 0.05 \)  
  Minimum Increment: 0.01  
  Recommended Increment: 0.1  
  Restriction: \( ((0 < \text{Exponent}) \land (\text{Exponent} \leq 1)) \)

### Example

```plaintext
read_image(Image,'monkey')
symmetry(Image,ImageSymmetry,70,0.0,0.5)
threshold(ImageSymmetry,SymmPoints,170,255)
```

### Result

If the parameter values are correct the operator **Symmetry** returns the value TRUE The behavior in case of empty input (no input images available) is set via the operator **SetSystem**(:'noObjectResult',<Result>):. If necessary an exception handling is raised.

### Parallelization Information

**Symmetry** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

### Possible Successors

**Threshold**

### Module

```plaintext
[out] HImageX Sketch HImageX.TopographicSketch ( )

void HOperatorSetX.TopographicSketch ([in] HObjectX Image,  
[out] HUntypedObjectX Sketch )
```

Compute the topographic primal sketch of an image.

**TopographicSketch** computes the topographic primal sketch of the input image **Image**. This is done by approximating the image locally by a bicubic polynomial (“facet model”). It serves to calculate the first and second partial derivatives of the image, and thus to classify the image into 11 classes. These classes are coded in the output image **Sketch** as numbers from 1 to 11. The classes are as follows:
In order to obtain the separate classes as regions, a threshold operation has to be applied to the result image with the appropriate thresholds.

Parameter

- **Image** (input iconic) ..................... (multichannel-)image(-array) \(\sim\) HImageX / HObjectX (byte) Image for which the topographic primal sketch is to be computed.
- **Sketch** (output iconic) .............. (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte) Label image containing the 11 classes.

Example

```c
/* To extract the Ridges from a Image */
read_image(Image,'sinus')
topographic_sketch(Image,Sketch)
threshold(Sketch,Ridges,3,3).
```

Complexity

Let \(n\) be the number of pixels in the image. Then \(O(n)\) operations are performed.

Result

TopographicSketch returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception is raised.

Parallelization Information

TopographicSketch is reentrant and automatically parallelized (on tuple level, channel level).

Possible Successors

Threshold

References


Module

Image filters

### 3.11 Noise

Add noise to an image.

AddNoiseDistribution adds noise distributed according to Distribution to the image Image. Resulting gray values are clipped to the range [0,255].
3.11. NOISE

Parameter

- **Image** (input iconic) ................. (multichannel-)image(-array) \(\sim\) \(\text{HImageX} / \text{IObjectX}\) (byte, int2)
  Input image.

- **ImageNoise** (output iconic) ........ (multichannel-)image(-array) \(\sim\) \(\text{HImageX} / \text{HUntypedObjectX}\) (byte, int2)
  Noisy image.

  **Number of elements:** \(\text{ImageNoise} = \text{Image}\)

- **Distribution** (input control) .................. distribution.values \(\sim\) \(\text{VARIANT}\) (real)
  Noise distribution.

  **Number of elements:** 513

Example

```c
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
isp_image(ImageNoise,WindowHandle).
```

Result

AddNoiseDistribution returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

AddNoiseDistribution is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

GaussDistribution, SpDistribution, NoiseDistributionMean

See also

SpDistribution, GaussDistribution, NoiseDistributionMean, AddNoiseWhite

Alternatives

AddNoiseWhite

Module

Image filters

```c
[out] \text{HImageX} \text{ImageNoise} \text{HImageX.AddNoiseWhite} ([in] \text{double Amp})

void \text{HOperatorSetX.AddNoiseWhite} ([in] \text{IObjectX Image},
[out] \text{HUntypedObjectX ImageNoise}, [in] \text{VARIANT Amp})
```

Add noise to an image.

AddNoiseWhite adds noise to the image Image. The noise is white noise, equally distributed in the interval \([-Amp,Amp]\), and is generated by using the C function “drand48” with an initial time dependent seed. Resulting gray values are clipped to the range \([0,255]\).

Parameter

- **Image** (input iconic) ................. (multichannel-)image(-array) \(\sim\) \(\text{HImageX} / \text{IObjectX}\) (byte, int2, uint2)
  Input image.

- **ImageNoise** (output iconic) ........ (multichannel-)image(-array) \(\sim\) \(\text{HImageX} / \text{HUntypedObjectX}\) (byte, int2, uint2)
  Noisy image.

  **Number of elements:** \(\text{ImageNoise} = \text{Image}\)
**Amp** (input control)  
Maximum noise amplitude.  

**Default Value:** 60.0  

**Suggested values:** Amp ∈ {1.0, 2.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}  

**Typical range of values:** 1.0 ≤ Amp ≤ 1.0  

**Minimum Increment:** 1.0  

**Recommended Increment:** 10.0  

**Restriction:** (Amp > 0)

---

Example

```plaintext
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
add_noise_white(Image,ImageNoise,90)
disp_image(ImageNoise,WindowHandle).
```

---

**Result**

`AddNoiseWhite` returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

`AddNoiseWhite` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

**See also**

`AddNoiseDistribution`, `NoiseDistributionMean`, `GaussDistribution`, `SpDistribution`

---

**Alternatives**

`AddNoiseDistribution`

---

**Module**

Image filters

---

```plaintext
[out] VARIANT Distribution HMiscX.GaussDistribution ([in] double Sigma )
```

```plaintext
void HOperatorSetX.GaussDistribution ([in] VARIANT Sigma, [out] VARIANT Distribution )
```

Generate a Gaussian noise distribution.

`GaussDistribution` generates a Gaussian noise distribution. The parameter `Sigma` determines the noise’s standard deviation. Usually, the result `Distribution` is used as input for the operator `AddNoiseDistribution`.

---

**Parameter**

- **Sigma** (input control)  
Standard deviation of the Gaussian noise distribution.  

**Default Value:** 2.0  

**Suggested values:** Sigma ∈ {1.5, 2.0, 3.0, 5.0, 10.0}  

**Typical range of values:** 0.0 ≤ Sigma ≤ 0.0  

**Minimum Increment:** 0.1  

**Recommended Increment:** 1.0  

**Distribution** (output control)  
Resulting Gaussian noise distribution.  

**Number of elements:** 513  

---

Example

```plaintext
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
 gauss_distribution(30,Dist)
 add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle).
```
3.11. NOISE

Parallelization Information

\texttt{GaussDistribution} is reentrant and processed without parallelization.

Possible Successors

\texttt{AddNoiseDistribution}

\texttt{SpDistribution, AddNoiseWhite, NoiseDistributionMean}

Alternatives

\texttt{SpDistribution, NoiseDistributionMean}

Module

Image filters

\begin{verbatim}
HRegionX.NoiseDistributionMean(const HImageX Image, const VARIANT FilterSize)

 void HOperatorSetX.NoiseDistributionMean(const IHObjectX ConstRegion, const IHObjectX Image, const VARIANT FilterSize, const VARIANT Distribution)
\end{verbatim}

Determine the noise distribution of an image.

\texttt{NoiseDistributionMean} calculates the noise distribution in a region of the image \texttt{Image}. The parameter \texttt{ConstRegion} determines a region of the image with approximately constant gray values. Ideally, the changes in gray values should only be caused by noise in this region. From this region the noise distribution is determined by using the \texttt{MeanImage} operator to smooth the image, and to use the gray value differences in this area as an estimate for the noise distribution, which is returned in \texttt{Distribution}.

Attention

It is important to ensure that the region \texttt{ConstRegion} is not too close to a large gradient in the image, because the gradient values are then used for calculating the mean. This means the distance of \texttt{ConstRegion} must be at least as large as the filter size \texttt{FilterSize} used for calculating the mean.

Parameter

\begin{itemize}
  \item \texttt{ConstRegion} (input iconic) \texttt{HRegionX / IHObjectX} region(-array)
  \item \texttt{Image} (input iconic) \texttt{HImageX / IHObjectX (byte)} image
  \item \texttt{FilterSize} (input control) \texttt{long / VARIANT} integer
  \item \texttt{Distribution} (output control) \texttt{VARIANT} distribution.values
\end{itemize}

\texttt{NoiseDistributionMean} is reentrant and processed without parallelization.

Possible Predecessors

\texttt{DrawRegion, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, Threshold, ErosionCircle, GaussImage, SmoothImage, SubImage}

Possible Successors

\texttt{AddNoiseDistribution, DispDistribution}

See also

\texttt{MeanImage, GaussDistribution}

Image filters

HALCON 6.1.4
Generate a salt-and-pepper noise distribution.

**SpDistribution** generates a noise distribution with the values 0 and 255. The parameters **PercentSalt** and **PercentPepper** determine the percentage of white and black noise pixels, respectively. The sum of these parameters must be smaller than 100. Usually, the result **Distribution** is used as input for the operator **AddNoiseDistribution**.

---

**Parameter**

- **PercentSalt** (input control) ........................................... number \(\sim\) **VARIANT** ( real, integer )
  - Percentage of salt (white noise pixels).
  - **Default Value**: 5.0
  - **Suggested values**: **PercentSalt** \(\in\) \{1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0\}
  - **Typical range of values**: \(0.0 \leq\) **PercentSalt** \(\leq 0.0\)
  - **Minimum Increment**: 0.1
  - **Recommended Increment**: 1.0
  - **Restriction**: \((0.0 \leq\) **PercentSalt**) \(\wedge (\) **PercentSalt** \(\leq 100.0)\)

- **PercentPepper** (input control) ........................................... number \(\sim\) **VARIANT** ( real, integer )
  - Percentage of pepper (black noise pixels).
  - **Default Value**: 5.0
  - **Suggested values**: **PercentPepper** \(\in\) \{1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0\}
  - **Typical range of values**: \(0.0 \leq\) **PercentPepper** \(\leq 0.0\)
  - **Minimum Increment**: 0.1
  - **Recommended Increment**: 1.0
  - **Restriction**: \((0.0 \leq\) **PercentPepper**) \(\wedge (\) **PercentPepper** \(\leq 100.0)\)

- **Distribution** (output control) ........................................... **distribution.values** \(\sim\) **VARIANT** ( real )
  - Resulting noise distribution.
  - **Number of elements**: 513

---

**Example**

```plaintext
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle)
```

---

**Parallelization Information**

**SpDistribution** is **reentrant** and processed **without** parallelization.

**Possible Successors**

- **AddNoiseDistribution**

**See also**

- GaussDistribution, NoiseDistributionMean, AddNoiseWhite

**Alternatives**

- GaussDistribution, NoiseDistributionMean

**Module**

Image filters
3.12 Smoothing

Smooth an image by edge-preserving anisotropic diffusion.

The operator `AnisotropeDiff` carries out an iterative, anisotropic smoothing process on the mathematical basis of physical diffusion. In analogy to the physical diffusion process describing the concentration balance between molecules dependent on the density gradient, the diffusion filter carries out a smoothing of the gray values dependent on the local gray value gradients.

For iterative calculation of the gray value of a pixel the gray value differences in relation to the four or eight neighbors, respectively, are used. These gray value differences, however, are evaluated differently, i.e., a non-linear diffusion process is carried out.

The evaluation is carried out by using a diffusion function (two different functions were implemented, namely `Mode` = 1 and/or 2), which — depending on the gradient — ensures that within homogenous regions the smoothing is stronger than over the margins of regions so that the edges remain sharp. The diffusion function is adjusted to the noise ratio of the image by a histogram analysis in the gradient image (according to Canny). A high value for `Percent` increases the smoothing effect but blurs the edges a little more (values from 80 - 90 percent are typical).

The parameter `Iteration` determines the number of iterations (typically 3–7).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image</code></td>
<td>(input iconic) (multichannel-)image-array (~ HimageX / IobjectX (byte))</td>
</tr>
<tr>
<td><code>ImageAniso</code></td>
<td>(output iconic) (multichannel-)image-array (~ HimageX / HUntypedObjectX (byte))</td>
</tr>
<tr>
<td><code>Percent</code></td>
<td>(input control) integer (~ long / VARIANT)</td>
</tr>
<tr>
<td>Default Value</td>
<td>80</td>
</tr>
<tr>
<td>Suggested values</td>
<td>Percent ∈ {65, 70, 75, 80, 85, 90}</td>
</tr>
<tr>
<td>Typical range of values</td>
<td>50 ≤ Percent ≤ 50</td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>1</td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>5</td>
</tr>
<tr>
<td><code>Mode</code></td>
<td>(input control) integer (~ long / VARIANT)</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>List of values</td>
<td>Mode ∈ {1, 2}</td>
</tr>
<tr>
<td><code>Iteration</code></td>
<td>(input control) integer (~ long / VARIANT)</td>
</tr>
<tr>
<td>Default Value</td>
<td>5</td>
</tr>
<tr>
<td>Suggested values</td>
<td>Iteration ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}</td>
</tr>
<tr>
<td>Typical range of values</td>
<td>1 ≤ Iteration ≤ 1</td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>1</td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>1</td>
</tr>
<tr>
<td><code>neighborhoodType</code></td>
<td>(input control) integer (~ long / VARIANT)</td>
</tr>
<tr>
<td>Default Value</td>
<td>8</td>
</tr>
<tr>
<td>List of values</td>
<td>neighborhoodType ∈ {4, 8}</td>
</tr>
</tbody>
</table>

**Example**

```halcon
read_image(Image, ‘fabrik’)
anisotrope_diff(Image, Aniso, 80, 1, 5, 8)
sub_image(Image, Aniso, Sub, 2.0, 127)
disp_image(Sub, WindowHandle).
```
CHAPTER 3. FILTER

Complexity

For each pixel: \( O(\text{Iterations} \times 18) \).

Result

If the parameter values are correct the operator \texttt{AnisotropeDiff} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem ('noObjectResult',<Result>)}. If necessary an exception handling is raised.

Parallelization Information

\texttt{AnisotropeDiff} is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors

\texttt{ReadImage}, \texttt{GrabImage}

Possible Successors

\texttt{Regiongrowing}, \texttt{Threshold}, \texttt{SubImage}, \texttt{DynThreshold}, \texttt{AutoThreshold}

See also

\texttt{SmoothImage}, \texttt{GaussImage}, \texttt{SigmaImage}, \texttt{RankImage}, \texttt{EliminateMinMax}

Alternatives

\texttt{SigmaImage}, \texttt{RankImage}

References


Module

Image filters

\begin{verbatim}
[out] HImageX filteredImage HImageX.EliminateMinMax

\end{verbatim}

Smooth an image in the spatial domain to suppress noise.

\texttt{EliminateMinMax} smooths an image by replacing gray values with neighboring mean values, or local minima/maxima. In order to prevent edges and lines from being smoothed, only those gray values that represent local minima or maxima are replaced (if there is a line or edge within an image there will be at least one neighboring pixel with a comparable gray value). \texttt{Gap} controls the strictness of replacement: Only gray values that exceed all other values within their local neighborhood more than \texttt{Gap} and all values that fall below their neighboring more than \texttt{Gap} are replaced: \( E(x,y) \) represents a \( N \times M \) sized rectangular neighborhood of an pixel at position \( (x,y) \), containing all pixels within the neighborhood except the pixel itself;

- if \( \text{grayvalue}(x,y) \geq \text{Gap} + \text{maximum}(E(x,y)) \) then replacement;
- else if \( \text{grayvalue}(x,y) + \text{Gap} \leq \text{minimum}(E(x,y)) \) then replacement;
- else adopt \( \text{grayvalue}(x,y) \) without change;

\texttt{Mode} specifies how to perform the new value in case of a replacement.

\texttt{Mode} = 1 \rightarrow replace a local maximum with next minor local maximum and replace a local minimum with next bigger local minimum

\texttt{Mode} = 2 \rightarrow replace with mean value of all pixels within the local neighborhood (including the replaced pixel)

\texttt{Mode} = 3 \rightarrow replace with median value of all pixels within the local neighborhood (including the replaced pixel (this is default and used if \texttt{Mode} has got any other value than 1 or 2)

\texttt{MaskWidth} and \texttt{MaskHeight} specify the width and height of the rectangular neighborhood. Border treatment: Pixels outside the image border are not considered (e.g.: With a local \( 3 \times 3 \)-mask the neighborhood of a pixel at \( (0,0) \) reduces to the pixels at \( (1,0) \), \( (0,1) \) and \( (1,1) \)).

Attention

\texttt{EliminateMinMax} only can work on byte images (HALCON image type BYTE_IMAGE). If \texttt{MaskWidth} or
**MaskHeight** is an even number, it is replaced by the next higher odd number (this allows the unique extraction of the center of the filter mask). Width/height of the mask may not exceed the image width/height.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>(input iconic)</td>
<td>(multichannel-)image (\Rightarrow) HI mageX / HObjectX (byte) Image to smooth.</td>
</tr>
<tr>
<td>filteredImage</td>
<td>(output iconic)</td>
<td>(multichannel-)image (\Rightarrow)HI mageX / HUntypedObjectX (byte) Smoothed image.</td>
</tr>
<tr>
<td>MaskWidth</td>
<td>(input control)</td>
<td>extent.x (\Rightarrow) long / VARIANT Width of filter mask.</td>
</tr>
<tr>
<td>Default Value</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Suggested values</td>
<td>MaskWidth (\in) {3, 5, 7, 9}</td>
<td></td>
</tr>
<tr>
<td>Typical range of values</td>
<td>(3 \leq \text{MaskWidth} \leq 3)</td>
<td></td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>odd</td>
<td></td>
</tr>
<tr>
<td>MaskHeight</td>
<td>(input control)</td>
<td>extent.y (\Rightarrow) long / VARIANT Height of filter mask.</td>
</tr>
<tr>
<td>Default Value</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Suggested values</td>
<td>MaskHeight (\in) {3, 5, 7, 9}</td>
<td></td>
</tr>
<tr>
<td>Typical range of values</td>
<td>(3 \leq \text{MaskHeight} \leq 3)</td>
<td></td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>odd</td>
<td></td>
</tr>
<tr>
<td>Gap</td>
<td>(input control)</td>
<td>number (\Rightarrow) double / VARIANT Gap between local maximum/minimum and all other gray values of the neighborhood.</td>
</tr>
<tr>
<td>Default Value</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Suggested values</td>
<td>Gap (\in) {1.0, 2.0, 5.0, 10.0}</td>
<td></td>
</tr>
<tr>
<td>Mode</td>
<td>(input control)</td>
<td>integer (\Rightarrow) long / VARIANT Replacement rule (1 = \text{next minimum/maximum}, 2 = \text{average}, 3 = \text{median}).</td>
</tr>
<tr>
<td>Default Value</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>List of values</td>
<td>Mode (\in) {1, 2, 3}</td>
<td></td>
</tr>
</tbody>
</table>

EliminateMinMax returns TRUE if all parameters are correct. If the input is empty EliminateMinMax returns with an error message.

Parallelization Information

EliminateMinMax is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

WienerFilter, WienerFilterNi

See also

MeanSp, MeanImage, MedianImage, MedianWeighted, GaussImage, SmoothImage

References


M. Lükenhaus: “Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse”; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Module

Image filters
CHAPTER 3. FILTER

Replace values outside of thresholds with average value.
The operator `EliminateSp` replaces all gray values outside the indicated gray value intervals (MinThresh to MaxThresh) with the neighboring mean values. Only those neighboring pixels which also fall within the gray value interval are used for averaging. If no such pixel is present in the vicinity the original gray value is used. The gray values in the input image falling within the gray value interval are also adopted without change.

Attention
If even values instead of odd values are given for MaskHeight or MaskWidth, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

Parameter

- **Image** (input iconic) .................................. (multichannel-)image(-array) ~ HImageX / IHObjectX (byte)
  
  Input image.

- **ImageFillSP** (output iconic) .... (multichannel-)image(-array) ~ HImageX / HUntypedObjectX (byte)
  
  Smoothed image.

- **MaskWidth** (input control) .......................................................... extent.x ~ long / VARIANT
  
  Width of filter mask.
  
  Default Value : 3
  
  Suggested values : MaskWidth ∈ {3, 5, 7, 9, 11}
  
  Typical range of values : 3 ≤ MaskWidth ≤ 3(lin)
  
  Minimum Increment : 2
  
  Recommended Increment : 2
  
  Restriction : odd

- **MaskHeight** (input control) .......................................................... extent.y ~ long / VARIANT
  
  Height of filter mask.
  
  Default Value : 3
  
  Suggested values : MaskHeight ∈ {3, 5, 7, 9, 11}
  
  Typical range of values : 3 ≤ MaskHeight ≤ 3(lin)
  
  Minimum Increment : 2
  
  Recommended Increment : 2
  
  Restriction : odd

- **MinThresh** (input control) .......................................................... integer ~ long / VARIANT
  
  Minimum gray value.
  
  Default Value : 1
  
  Suggested values : MinThresh ∈ {1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
  
  Typical range of values : 0 ≤ MinThresh ≤ 0(lin)
  
  Minimum Increment : 1
  
  Recommended Increment : 1

- **MaxThresh** (input control) .......................................................... integer ~ long / VARIANT
  
  Maximum gray value.
  
  Default Value : 254
  
  Suggested values : MaxThresh ∈ {5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254}
  
  Typical range of values : 1 ≤ MaxThresh ≤ 1(lin)
  
  Minimum Increment : 1
  
  Recommended Increment : 1
  
  Restriction : (MinThresh ≤ MaxThresh)

Example

```plaintext
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
eliminate_sp(Image,ImageMeansp,3,3,101,201)
disp_image(ImageMeansp,WindowHandle).
```

HALCON/COM Reference Manual, 2005-2-1
**3.12. SMOOTHING**

---

**Parallelization Information**

*EliminateSp* is *reentrant* and automatically *parallelized* (on *tuple level*, *channel level*, *domain level*).

---

**Possible Successors**

---

**DispImage**

---

See also: *GaussImage*, *SmoothImage*, *AnisotropeDiff*, *SigmaImage*, *EliminateMinMax*  
---

**Alternatives**

*MeanSp*, *MeanImage*, *MedianImage*, *EliminateMinMax*  
---

**Module**

---

**Image filters**

---

```plaintext
[out] HImageX ImageFilled HImageX.FillInterlace ([in] String Mode )

```

---

**Interpolate 2 video half images.**

The operator *FillInterlace* calculates an interpolated full image or removes odd/even lines from a video image composed of two half images. If an image is recorded with a video camera it consists of two half images recorded at different times but stored in one image in the digital form. This can lead to several errors in further processing. In order to reduce these errors the video image is modified. Every second line is re-calculated or removed. The parameter *Mode* determines whether this must be done for even (‘even’, ‘rmeven’) or odd (‘odd’, ‘rmodd’) line numbers. If you choose ‘even’ or ‘odd’ the gray values in the generated lines are calculated as mean values from the direct neighbors above or below the current pixel, respectively. If you choose ‘rmeven’ or ‘rmodd’ the even or odd lines numbers are removed (in that case the resulting image is only half as high as the input image). The value ‘switch’ for *Mode* cause the odd and even lines to be exchanged.

---

**Parameter**

- *ImageCamera* (input iconic) ..........(multichannel-)image(-array) ⊲ HImageX / IHObjectX (byte)  
  Gray image consisting of two half images.

- *ImageFilled* (output iconic) .......(multichannel-)image(-array) ⊲ HImageX / HUntypedObjectX (byte)  
  Full image with interpolated/removed lines.

- *Mode* (input control) .....................string ⊲ String / VARIANT  
  Instruction whether even or odd lines should be replaced/removed.  
  **Default Value**: ‘odd’  
  **List of values**: Mode ∈ {'odd', 'even', 'rmodd', 'rmeven', 'switch'}

---

**Example**

```plaintext
read_image(Image,'video_bild')
fill_interlace(Image,New,'odd')
sobel_amp(New,Sobel,'sum_abs',3).
```

---

**Complexity**

For each pixel: $O(2)$.

---

**Result**

If the parameter values are correct the operator *FillInterlace* returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator *SetSystem* (‘noObjectResult’,<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

*FillInterlace* is *reentrant* and automatically *parallelized* (on *tuple level*, *channel level*, *domain level*).

---

**Possible Predecessors**

---

---
Possible Successors
SobelAmp, EdgesImage, Regiongrow, DiffOfGauss, Threshold, DynThreshold,
AutoThreshold, MeanImage, GaussImage, AnisotropicDiff, SigmaImage, MedianImage
See also
MedianImage, GaussImage, CropPart
Module

Image filters

```
[out] HImageX ImageGauss HImageX.GaussImage ([in] long Size )
void HOperatorSetX.GaussImage ([in] IObjectX Image,
```

Smooth using discrete gauss functions.
The operator GaussImage smoothes images using the discrete Gaussian. The smoothing effect increases with
increasing filter size. The following filter sizes (Size) are supported (the sigma value of the gauss function is
indicated in brackets):

<table>
<thead>
<tr>
<th>Size</th>
<th>Sigma</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.81</td>
</tr>
<tr>
<td>5</td>
<td>0.93</td>
</tr>
<tr>
<td>7</td>
<td>1.50</td>
</tr>
<tr>
<td>9</td>
<td>2.00</td>
</tr>
<tr>
<td>11</td>
<td>2.45</td>
</tr>
</tbody>
</table>

For border treatment the gray values of the images are reflected at the image borders.

Parameter

- **Image** (input iconic) .... (multichannel-)image(-array) \(\sim\) HImageX / IObjectX (byte, int2, uint2, int4)
  Image to be smoothed.
- **ImageGauss** (output iconic) .... (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte,
  int2, uint2, int4)
  Filtered image.
- **Size** (input control) ....................... integer \(\sim\) long / VARIANT
  Required filter size.
  **Default Value:** 5
  **List of values:** Size \(\in\) \{3, 5, 7, 9, 11\}

Example

gauss_image(Input,Gauss,7)
regiongrow(Gauss,Segments,7,7,5,100).

Complexity

For each pixel: \(O(\text{Size} \times 2)\).

Result

If the parameter values are correct the operator GaussImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem
(‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

GaussImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

ReadImage, GrabImage

Possible Successors

Regiongrow, Threshold, SubImage, DynThreshold, AutoThreshold

HALCON/COM Reference Manual, 2005-2-1
3.12. SMOOTHING

See also
MeanImage, AnisotropeDiff, SigmaImage, GenLowpass

Alternatives
SmoothImage, DerivateGauss

Module
Image filters


Information on smoothing filter SmoothImage.
The operator InfoSmooth returns an estimation of the width of the smoothing filters used in routine SmoothImage. For this purpose the underlying continuous impulse answers of Filter are scanned until a filter coefficient is smaller than five percent of the maximum coefficient (at the origin). Alpha is the filter parameter (see SmoothImage). Currently four filters are supported (parameter Filter):

- 'deriche1', 'deriche2', 'shen' und 'gauss'.

The gauss filter was conventionally implemented with filter masks (the other three are recursive filters). In the case of the gauss filter the filter coefficients (of the one-dimensional impulse answer \( f(n) \) with \( n \geq 0 \)) are returned in Coeffs in addition to the filter size.

Parameter

- **Filter** (input control) .......................................................... string – String / VARIANT
  Name of required filter.
  Default Value : 'deriche2'
  List of values : Filter \( \in \{ \text{deriche1}, \text{deriche2}, \text{shen}, \text{gauss} \} \)

- **Alpha** (input control) ............................................................. real – double / VARIANT
  Filter parameter: small values effect strong smoothing (reversed in case of 'gauss').
  Default Value : 0.5
  Suggested values : Alpha \( \in \{ 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0 \} \)
  Typical range of values : 0.01 \( \leq \) Alpha \( \leq 0.01 \)
  Minimum Increment : 0.01
  Recommended Increment : 0.1
  Restriction : (Alpha > 0.0)

- **Size** (output control) ......................................................... integer – long / VARIANT
  Width of filter is approx. size x size pixels.

- **Coeffs** (output control) .......................................................integer – VARIANT ( integer )
  In case of gauss filter: coefficients of the “positive” half of the 1D impulse answer.

Example

```plaintext
info_smooth('deriche2', 0.5, Size, Coeffs)
smooth_image(Input, Smooth, 'deriche2', 7).
```

Result

If the parameter values are correct the operator InfoSmooth returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

InfoSmooth is reentrant and processed without parallelization.

Possible Predecessors

ReadImage
Smooth by averaging.

The operator **MeanImage** carries out a linear smoothing with the gray values of all input images (**Image**). The filter matrix consists of ones (evaluated equally) and has the size **MaskHeight** × **MaskWidth**. The result of the convolution is divided by **MaskHeight** × **MaskWidth**. For border treatment the gray values are reflected at the image edges.

**Attention**

If even values instead of odd values are given for **MaskHeight** or **MaskWidth**, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

**Parameter**

- **Image** (input iconic) …… (multichannel-)image(-array) ⇒ **HImageX / IHObjectX** ( byte, int2, uint2, int4, real, dvf )
  Image to be smoothed.
- **ImageMean** (output iconic) …… (multichannel-)image(-array) ⇒ **HImageX / HUntypedObjectX** ( byte, int2, uint2, int4, real, dvf )
  Smoothed image.
- **MaskWidth** (input control) ………………………………………… extent.x ⇒ long / VARIANT
  Width of filter mask.
  Default Value : 9
  Suggested values : MaskWidth ∈ {3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
  Typical range of values : 1 ≤ MaskWidth ≤ 1
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : odd
- **MaskHeight** (input control) ………………………………………… extent.y ⇒ long / VARIANT
  Height of filter mask.
  Default Value : 9
  Suggested values : MaskHeight ∈ {3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
  Typical range of values : 1 ≤ MaskHeight ≤ 1
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : odd

**Example**

```plaintext
read_image(Image,'fabrik')
mean_image(Image,Mean,3,3)
disp_image(Mean,WindowHandle).
```

**Complexity**

For each pixel: O(15).
If the parameter values are correct the operator \texttt{MeanImage} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem ('noObjectResult',<Result>)}. If necessary an exception handling is raised.

\textbf{Parallelization Information}

\texttt{MeanImage} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, channel level, domain level).

\textbf{Possible Successors}

DynThreshold, RegionGrowing

\textbf{See also}

AnisotropeDiff, SigmaImage, ConvolImage, GenLowpass

\textbf{Alternatives}

GaussImage, SmoothImage

\textbf{Module}

Image filters

\begin{verbatim}
[out] HImageX ImageMean HImageX.MeanN ( )
void HOperatorSetX.MeanN ([in] IObjectX Image, [out] HUntypedObjectX ImageMean )
\end{verbatim}

Average gray values over several channels.

The operator \texttt{MeanN} generates the pixel-by-pixel mean value of all channels. For each coordinate point the sum of all gray values at this coordinate is calculated. The result is the mean of the gray values (sum divided by the number of channels). The output image has one channel.

\textbf{Parameter}

\begin{itemize}
    \item \texttt{Image} (input iconic) \ldots multichannel-image-array \rightsquigarrow HImageX / IObjectX ( byte, int4, uint2, int4, real )
        Multichannel gray image.
    \item \texttt{ImageMean} (output iconic) \ldots image(-array) \rightsquigarrow HImageX / HUntypedObjectX ( byte, int4, uint2, int4, real )
        Result of averaging.
\end{itemize}

\textbf{Parallelization Information}

\texttt{MeanN} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level, domain level).

\textbf{Possible Predecessors}

Compose2, Compose3, Compose4, AddChannels

\textbf{Possible Successors}

DispImage

\textbf{See also}

CountChannels

\textbf{Module}

Image filters

\begin{verbatim}
\end{verbatim}

Suppress salt and pepper noise.

The operator \texttt{MeanSp} carries out a smoothing by averaging the values. Only the gray values within the interval from \texttt{MinThresh} to \texttt{MaxThresh} are averaged. Gray values which are too light or too dark are ignored during summation. If no gray value lies within the default interval during summation the original gray value is adopted.
If the thresholds are set at 0 or 255, respectively, the operator \texttt{MeanSp} behaves like \texttt{MeanImage} except for the running time.

The operator \texttt{MeanSp} is used to suppress extreme gray values (salt and pepper noise = white and black dots).

\begin{itemize}
  \item \texttt{Image} (input iconic) \ldots (multichannel-)image-array \leadsto \texttt{HImageX / IObjectX (byte)}
  Input image.
  \item \texttt{ImageSPMean} (output iconic) \ldots (multichannel-)image-array \leadsto \texttt{HImageX / HUntypedObjectX (byte)}
  Smoothed image.
  \item \texttt{MaskWidth} (input control) \ldots \texttt{extent.x} \leadsto \texttt{long / VARIANT}
  Width of filter mask.
  \hspace{1cm} Default Value : 3
  \hspace{1cm} Suggested values : \texttt{MaskWidth} \in \{3, 5, 7, 9, 11\}
  \hspace{1cm} Typical range of values : 3 \leq \texttt{MaskWidth} \leq 3(lin)
  \hspace{1cm} Minimum Increment : 2
  \hspace{1cm} Recommended Increment : 2
  \hspace{1cm} Restriction : odd
  \item \texttt{MaskHeight} (input control) \ldots \texttt{extent.y} \leadsto \texttt{long / VARIANT}
  Height of filter mask.
  \hspace{1cm} Default Value : 3
  \hspace{1cm} Suggested values : \texttt{MaskHeight} \in \{3, 5, 7, 9, 11\}
  \hspace{1cm} Typical range of values : 3 \leq \texttt{MaskHeight} \leq 3(lin)
  \hspace{1cm} Minimum Increment : 2
  \hspace{1cm} Recommended Increment : 2
  \hspace{1cm} Restriction : odd
  \item \texttt{MinThresh} (input control) \ldots \texttt{integer} \leadsto \texttt{long / VARIANT}
  Minimum gray value.
  \hspace{1cm} Default Value : 1
  \hspace{1cm} Suggested values : \texttt{MinThresh} \in \{1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101\}
  \hspace{1cm} Typical range of values : 0 \leq \texttt{MinThresh} \leq 0(lin)
  \hspace{1cm} Minimum Increment : 1
  \hspace{1cm} Recommended Increment : 1
  \item \texttt{MaxThresh} (input control) \ldots \texttt{integer} \leadsto \texttt{long / VARIANT}
  Maximum gray value.
  \hspace{1cm} Default Value : 254
  \hspace{1cm} Suggested values : \texttt{MaxThresh} \in \{5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254\}
  \hspace{1cm} Typical range of values : 0 \leq \texttt{MaxThresh} \leq 0(lin)
  \hspace{1cm} Minimum Increment : 1
  \hspace{1cm} Recommended Increment : 1
  \hspace{1cm} Restriction : (\texttt{MinThresh} \leq \texttt{MaxThresh})
\end{itemize}

\begin{itemize}
  \item \texttt{read_image(Image,’meer_rot’)}
  \item \texttt{disp_image(Image,WindowHandle)}
  \item \texttt{mean_sp(Image,ImageMeansp,3,3,101,201)}
  \item \texttt{disp_image(ImageMeansp,WindowHandle)}.
\end{itemize}

\begin{itemize}
  \item Parallelization Information
  \item MeanSp is reentrant and automatically parallelized (on tuple level, channel level, domain level).
  \item Possible Successors
  \item DispImage
  \item See also
  \item AnisotropeDiff, SigmaImage, GaussImage, SmoothImage, EliminateMinMax
\end{itemize}

HALCON/COM Reference Manual, 2005-2-1
3.12. SMOOTHING

Alternatives

MeanImage, MedianImage, MedianSeparate, EliminateMinMax

Module

Image filters

Median filtering with different rank masks.

The operator **MedianImage** carries out a non-linear smoothing of the gray values of all input images (**Image**). The shift mask (**MaskType**) is transmitted in the form of an object (more precisely: its region). Several border treatments can be chosen for filtering (**Margin**):

- **gray value**: Pixels outside of the image edges are assumed to be constant (with the indicated gray value).
- **'continued'**: Continuation of edge pixels.
- **'cyclic'**: Cyclic continuation of image edges.
- **'mirrored'**: Reflection of pixels at the image edges.

The indicated mask (= region of the mask image) is put over the image to be filtered in such a way that the center of the mask touches all pixels of the objects once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence according to their gray values. Thus, each of these sorted gray value sequences contains exactly as many gray values as the mask has pixels. From these sequences the median is selected and entered as resulting gray value at the corresponding output image.

### Parameter

- **Image** (input iconic) ........ (multichannel-)image(-array) \(\sim\) **HImageX / IObjectX** (byte, int2, uint2, int4, real)
  - Image to be filtered.
- **ImageMedian** (output iconic) ........ (multichannel-)image(-array) \(\sim\) **HImageX / HUntypedObjectX** (byte, int2, uint2, int4, real)
  - Median filtered image.
- **MaskType** (input control) .......................................................... string \(\sim\) **String / VARIANT**
  - Type of median mask.
  - **Default Value**: 'circle'
  - **List of values**: MaskType \(\in\) { 'circle', 'rectangle' }
- **Radius** (input control) .............................................................. integer \(\sim\) **long / VARIANT**
  - Radius of median mask.
  - **Default Value**: 1
  - **Suggested values**: Radius \(\in\) { 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59 }
  - **Typical range of values**: 1 \(\leq\) Radius \(\leq\) 1
  - **Minimum Increment**: 1
  - **Recommended Increment**: 2
- **Margin** (input control) .............................................................. string \(\sim\) **VARIANT** (integer, real, string)
  - Border treatment.
  - **Default Value**: 'mirrored'
  - **Suggested values**: Margin \(\in\) { 'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255 }

### Example

```plaintext
read_image(Image,'fabrik')
median_image(Image,Median,'circle',3,'continued')
disp_image(MedianWeighted,WindowHandle).
```

HALCON 6.1.4
CHAPTER 3. FILTER

For each pixel: $O(\sqrt{F^2} + 5)$ with $F = \text{area of MaskType}$.

If the parameter values are correct the operator MedianImage returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem ('noObjectResult', <Result>). If necessary an exception handling is raised.

Parallelization Information

MedianImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

ReadImage

Possible Successors

Threshold, DynThreshold, Regiongrowing

See also

GenCircle, GenRectangle, GrayErosionRect, GrayDilationRect

Alternatives

RankImage

References


Module

Image filters

Separated median filtering with rectangle masks.

The operator MedianSeparate carries out a variation of the median filtering: First two auxiliary images are created. The first one originates from a median filtering with a horizontal mask with a height of one pixel and the width MaskWidth followed by filtering with a mask with the height MaskHeight. The second auxiliary image is created by filtering with the same masks, but with a reversed sequence of the operation: first the vertical, then the horizontal mask. The output image results from averaging the two auxiliary images pixel by pixel.

The operator MedianSeparate is clearly faster than the normal operator MedianImage because both masks are one pixel wide, facilitating a very efficient processing. The runtime is practically independent of the size of the mask. For example, the operator MedianSeparate can be well used after texture filters, where large masks are needed.

The filter can also be used several times in a row in order to enhance the smoothing.

Parameter

- **Image** (input iconic) ………… (multichannel-)image(-array) $\rightarrow$ HImageX / HObjectX (byte, int2, uint2, int4, real)

  Image to be filtered.

- **ImageSMedian** (output iconic) ………… (multichannel-)image(-array) $\rightarrow$ HImageX / HUntypedObjectX (byte, int2, uint2, int4, real)

  Median filtered image.

- **MaskWidth** (input control) ……………………………………………………… extent.x $\rightarrow$ long / VARIANT

  Width of rank mask.

  Default Value : 25

  Suggested values : MaskWidth $\in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151\}$

  Typical range of values : $1 \leq \text{MaskWidth} \leq 1$

  Minimum Increment : 2

  Recommended Increment : 2

HALCON/COM Reference Manual, 2005-2-1
3.12. SMOOTHING

- **MaskHeight** (input control) ........................................... extent.y $\sim$ long / VARIANT
  Height of rank mask.

  **Default Value**: 25
  **Suggested values**: $\text{MaskHeight} \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151\}$
  **Typical range of values**: $1 \leq \text{MaskHeight} \leq 1$
  **Minimum Increment**: 2
  **Recommended Increment**: 2

- **Margin** (input control) ........................................... string $\sim$ VARIANT (integer, real, string)
  Border treatment.

  **Default Value**: ‘mirrored’
  **Suggested values**: $\text{Margin} \in \{\text{‘mirrored’}, \text{‘cyclic’}, \text{‘continued’}, 0, 30, 60, 90, 120, 150, 180, 210, 240, 255\}$

read_image(Image,’fabrik’)
median_separate(Image,MedianSeparate,5,5,3)
disp_image(MedianSeparate,WindowHandle).

---

**Example**

For each pixel: $O(\text{40})$.

---

**Parallelization Information**

- **MedianSeparate** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

---

**Possible Predecessors**

TextureLaws, SobelAmp, DeviationImage

---

**Possible Successors**

LearnNdimNorm, LearnNdimBox, MedianSeparate, Regiongrowing, AutoThreshold

---

**See also**

RankImage

---

**Alternatives**

MedianImage

---

**References**


---

**Module**

Image filters

```c
[out] HImageX ImageWMedian HImageX.MedianWeighted ([in] String MaskType, [in] long MaskSize )

```

Weighted median filtering with different rank masks.

The operator **MedianWeighted** calculates the median of the gray values within a local environment. In contrast to **MedianImage**, which uses all gray values within the environment exactly once, the operator **MedianWeighted** weights all gray values several times depending on their position. A gray value is received into the field to be sorted several times according to its weighting. The following masks are available:

- **'gauss'** (MaskSize = 3)
  
<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- **'inner'** (MaskSize = 3)
  
<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The operator \texttt{MedianWeighted} means that, contrary to \texttt{MedianImage}, gray value corners remain.

\begin{itemize}
  \item \textbf{Image} (input iconic) \hdots (multichannel-)image(-array) $\sim$ \texttt{HImageX / IObjectX} (byte, int2)
    Image to be filtered.
  \item \textbf{ImageWMedian} (output iconic) \hdots (multichannel-)image(-array) $\sim$ \texttt{HImageX / HUntypedObjectX} (byte, int2)
    Median filtered image.
\end{itemize}

\begin{itemize}
  \item \textbf{MaskType} (input control) \hdots \texttt{String / VARIANT}
    Type of median mask.
    Default Value : 'inner'
    List of values : \texttt{MaskType} $\in \{ \text{'inner'}, \text{'gauss'} \}$
  \item \textbf{MaskSize} (input control) \hdots \texttt{long / VARIANT}
    Mask size.
    Default Value : 3
    List of values : \texttt{MaskSize} $\in \{ 3 \}$
\end{itemize}

\textbf{Example}
\begin{verbatim}
read_image(Image,'fabrik')
median_weighted(Image,MedianWeighted,'gauss',3)
disp_image(MedianWeighted,WindowHandle).
\end{verbatim}

\textbf{Complexity}
For each pixel: $O(F \ast \log F)$ with $F$ = area of \texttt{MaskType}.

\textbf{Parallelization Information}
\texttt{MedianWeighted} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

\textbf{Possible Predecessors}
ReadImage

\textbf{Possible Successors}
Threshold, DynThreshold, RegionGrowing

\textbf{Alternatives}
MedianImage, TrimmedMean, SigmaImage

\textbf{References}
R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 319

\textbf{Module}
Image filters

\begin{verbatim}
[out] HImageX ImageMidrange HImageX.MidrangeImage ( [in] HRegionX Mask,
[in] VARIANT Margin )

void HOperatorSetX.MidrangeImage ( [in] IObjectX Image,
[in] IObjectX Mask, [out] HUntypedObjectX ImageMidrange,
[in] VARIANT Margin )
\end{verbatim}

Calculate the average of maximum and minimum inside any mask.

The operator \texttt{MidrangeImage} forms the average of maximum and minimum inside the indicated mask in the whole image. Several border treatments (\texttt{Margin}) can be chosen for filtering:

- \texttt{gray value} Pixels outside of the image edges are assumed to be constant (with the indicated gray value).
- \texttt{'continued'} Continuation of edge pixels.
- \texttt{'cyclic'} Cyclic continuation of image edges.
- \texttt{'mirrored'} Reflection of pixels at the image edges.
The indicated mask (= region of the mask image) is put over the image to be filtered in such a way that the center of the mask touches all pixels once.

### Parameter

- **Image** (input iconic) ...... (multichannel-)image(-array) \( \sim \) `HmageX / HOobjectX` (byte, int2, uint2, int4, real)
  - Image to be filtered.
- **Mask** (input iconic) ................. region \( \sim \) `HRegionX / HOobjectX` (byte)
  - Region serving as filter mask.
- **ImageMidrange** (output iconic) ...... (multichannel-)image(-array) \( \sim \) `HmageX / HUntypedObjectX` (byte, int2, uint2, int4, real)
  - Filtered output image.
- **Margin** (input control) .................. string \( \sim \) VARIANT (integer, real, string)
  - Border treatment.
  - **Default Value**: `mirrored`
  - **Suggested values**: Margin \( \in \{ 'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255 \}

### Example

```
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
midrange_image(Image,Region,Midrange,'mirrored')
disp_image(Midrange,WindowHandle).
```

### Complexity

For each pixel: \( O(\sqrt{F} * 5) \) with \( F \) = area of Mask.

### Result

If the parameter values are correct the operator **MidrangeImage** returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (`'noObjectResult',<Result>`). If necessary an exception handling is raised.

### Parallelization Information

**MidrangeImage** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

### Possible Predecessors

- ReadImage, DrawRegion, GenCircle, GenRectangle1

### Possible Successors

- Threshold, DynThreshold, RegionGROWING

### See also

- GenCircle, GenRectangle1, GrayErosionRect, GrayDilationRect, GrayRangeRect

### Alternatives

- SigmaImage

### References

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 319

### Module

- Image filters

---

```c
```

Smooth an image with an arbitrary rank mask.

The operator **RankImage** carries out a non-linear smoothing of the gray values of all input images (**Image**). The filter mask (**Mask**) is transmitted as a region. In contrast to many other filters you can choose an arbitrary shape,
e.g., by using operators like `GenCircle` or `DrawRegion`. The position of the mask region has no influence on the result; the center of gravity of the region is used as the reference point of the filter mask.

The specified mask is moved over the image to be filtered in such a way that the reference point of the mask touches all pixels once. At each position a histogram is calculated from the gray values of all pixels covered by the mask. By specifying **Rank** = 1 the lowest (= darkest) gray value appearing in the histogram is selected and entered as resulting gray value in the output image `ImageRank`; if **Rank** corresponds to the number of pixels of the filter mask, i.e., its area, the brightest gray value is selected. This behavior is identical to the erosion/dilation operators in gray morphology (`GrayErosion`, `GrayDilation`). If you use a rank that is equal to half of the pixels of the filter mask you get the same behavior as for the median filter (`MedianImage`).

You can use `RankImage` to eliminate noise, to eliminate structures with a given orientation (use `GenRectangle2` to create the mask region), or as an advanced gray morphologic operator that is more robust against noise. In this case you will not use 1 or the mask area as rank values, but a slightly higher or lower value, respectively.

Several border treatments can be chosen for filtering (**Margin**):

- **gray value**: Pixels outside of the image edges are assumed to be constant (with the indicated gray value).
- **‘continued’**: Continuation of edge pixels.
- **‘cyclic’**: Cyclic continuation of image edges.
- **‘mirrored’**: Reflection of pixels at the image edges.

---

**Parameter**

- **Image** (input iconic) . . . multichannel-image-array \( \leadsto HImageX / IHObjectX \) (byte, int2, uint2, int4, real)
  Image to be filtered.
- **Mask** (input iconic) . . . . . . . . . . region \( \leadsto HRegionX / IHObjectX \) (byte)
  Region serving as filter mask.
- **ImageRank** (output iconic) . . . multichannel-image-array \( \leadsto HImageX / HUntypedObjectX \) (byte, int2, uint2, int4, real)
  Filtered image.
- **Rank** (input control) . . . . . . . . . . . . . . integer \( \leadsto long / VARIANT \)
  Rank of the output gray value in the sorted sequence of input gray values inside the filter mask. Typical value (median): area(mask) / 2.
  **Default Value**: 5
  **Suggested values**: Rank \( \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31\} \)
  **Typical range of values**: \( 1 \leq \text{Rank} \leq 1 \)
  **Minimum Increment**: 1
  **Recommended Increment**: 2
- **Margin** (input control) . . . . . . . . . . . . . . string \( \leadsto VARIANT \) (integer, real, string)
  Border treatment.
  **Default Value**: ‘mirrored’
  **Suggested values**: Margin \( \in \{\text{‘mirrored’}, \text{‘cyclic’}, \text{‘continued’}, 0, 30, 60, 90, 120, 150, 180, 210, 240, 255\} \)

---

**Example**

- `read_image(Image,'fabrik')`
- `draw_region(Region,WindowHandle)`
- `rank_image(Image,Region,ImageRank,5,'mirrored')`
- `disp_image(ImageRank,WindowHandle)`.

---

**Complexity**

For each pixel: \( O(\sqrt{F} \times 5) \) with \( F \) = area of **Mask**.

---

**Result**

If the parameter values are correct the operator `RankImage` returns the value **TRUE**. The behavior in case of empty input (no input images available) is set via the operator `SetSystem` (`‘noObjectResult’`,<Result>). If necessary an exception handling is raised.

HALCON/COM Reference Manual, 2005-2-1
3.12. SMOOTHING

Parallelization Information

RankImage is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

ReadImage, DrawRegion, GenCircle, GenRectangle

Possible Successors

Threshold, DynThreshold, RegionGrowing

See also

GenCircle, GenRectangle, GrayErosionRect, GrayDilationRect

Alternatives

SigmaImage

References


Module

Image filters

Non-linear smoothing with the sigma filter.

The operator SigmaImage carries out a non-linear smoothing of the gray values of all input images (Image). All pixels are checked in a rectangular window (MaskHeight × MaskWidth). All pixels of the window which differ from the current pixel by less than Sigma are used for calculating the new pixel, which is the average of the chosen pixels. If all differences are larger than Sigma the gray value is adapted unchanged.

Attention

The filter is implemented for images of the 'byte' type only. If even values instead of odd values are given for MaskHeight or MaskWidth, the routine uses the next larger odd values instead (this way the center of the filter mask is always explicitly determined).

Parameter

- **Image** (input iconic) .... (multichannel-)image(-array) \( \rightsquigarrow \text{HImageX / IHObjectX} \) (byte, cyclic, int1, int2, uint2, int4, real)
  
  Image to be smoothed.

- **ImageSigma** (output iconic) .... (multichannel-)image(-array) \( \rightsquigarrow \text{HImageX / HUntypedObjectX} \) (byte, cyclic, int1, int2, uint2, int4, real)
  
  Smoothed image.

- **MaskHeight** (input control) ........................................... \text{extent.y} \( \rightsquigarrow \text{long / VARIANT} \)
  
  Height of the mask (number of lines).
  
  **Default Value**: 5
  
  **Suggested values**: MaskHeight \( \in \{3, 5, 7, 9, 11, 13, 15\} \)
  
  **Typical range of values**: \(3 \leq \text{MaskHeight} \leq 3\)
  
  **Minimum Increment**: 2
  
  **Recommended Increment**: 2
  
  **Restriction**: odd

- **MaskWidth** (input control) ........................................... \text{extent.x} \( \rightsquigarrow \text{long / VARIANT} \)
  
  Width of the mask (number of columns).
  
  **Default Value**: 5
  
  **Suggested values**: MaskWidth \( \in \{3, 5, 7, 9, 11, 13, 15\} \)
  
  **Typical range of values**: \(3 \leq \text{MaskWidth} \leq 3\)
  
  **Minimum Increment**: 2
  
  **Recommended Increment**: 2
  
  **Restriction**: odd

**HALCON 6.1.4**
\(\Sigma\) (input control) ..................................................... integer \(\rightarrow\) long / VARIANT
Max. deviation to the average.
Default Value: 3
Suggested values: \(\Sigma \in \{3, 5, 7, 9, 11, 20, 30, 50\}\)
Typical range of values: \(0 \leq \Sigma \leq 0\)
Minimum Increment: 1
Recommended Increment: 2

Example

\[
\text{read\_image(Image, 'fabrik')}
\text{sigma\_image(Image, ImageSigma, 5, 5, 3)}
\text{disp\_image(ImageSigma, WindowHandle)}.
\]

Complexity

For each pixel: \(O(\text{MaskHeight} \times \text{MaskWidth})\).

Result

If the parameter values are correct the operator \(\text{SigmaImage}\) returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \(\text{SetSystem('noObjectResult',<Result>)}\). If necessary an exception handling is raised.

Parallelization Information

\(\text{SigmaImage}\) is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

\(\text{ReadImage}\)

Possible Successors

\(\text{Threshold, DynThreshold, RegionGrowing}\)

See also

\(\text{SmoothImage, GaussImage, MeanImage}\)

Alternatives

\(\text{AnisotropeDiff, RankImage}\)

References

R. Haralick, L. Shapiro; "Computer and Robot Vision"; Addison-Wesley, 1992, Seite 325

Image filters

\[
[\text{out}] \text{HImageX ImageSmooth HImageX.SmoothImage ([in] String Filter, [in] double Alpha )}
\]

\[
\]

\(\text{SmoothImage}\) smooths gray images using recursive filters originally developed by Deriche and Shen and using the non-recursive Gaussian filter. The following filters can be chosen via the parameter \(\text{Filter}\):

\'derichel1', 'deriche2', 'shen' und 'gauss'.

The “filter width” (i.e., the range of the filter and thereby result of the filter) can be of any size. In the case that the Deriche or Shen is chosen it decreases by increasing the filter parameter \(\text{Alpha}\) and increases in the case of the Gauss filter (and \(\text{Alpha}\) corresponds to the standard deviation of the Gaussian function). An approximation of the appropriate size of the filterwidth \(\text{Alpha}\) is performed by the operator \(\text{InfoSmooth}\).

Non-recursive filters like the Gaussian filter are often implemented using filter-masks. In this case the runtime of the operator increases with increasing size of the filter mask. The runtime of the recursive filters remains constant; except the border treatment becomes a little bit more time consuming. The Gaussian filter becomes slow.
in comparison to the recursive ones but is in contrast to them isotropic (the filter ‘deriche2’ is only weakly direction sensitive). A comparable result of the smoothing is achieved by choosing the following values for the parameter:

\[
\begin{align*}
\text{Alpha('deriche2')} &= \frac{\text{Alpha('deriche1')}}{2} \\
\text{Alpha('shen')} &= \frac{\text{Alpha('deriche1')}}{2} \\
\text{Alpha('gauss')} &= \frac{1.77}{\text{Alpha('deriche1')}}
\end{align*}
\]

Parameter

- **Image** (input iconic) ...................... (multichannel-)image(-array) \sim HImageX / HObjectX (byte, uint2) Image to be smoothed.
- **ImageSmooth** (output iconic) ........ (multichannel-)image(-array) \sim HImageX / HUntypedObjectX (byte, uint2) Smoothed image.
- **Filter** (input control) ............................................. string \sim String / VARIANT Filter.
  - Default Value: ‘deriche2’
  - List of values: Filter \in \{‘deriche1’, ‘deriche2’, ‘shen’, ‘gauss’\}
- **Alpha** (input control) ............................................. real \sim double / VARIANT Filter parameter: small values cause strong smoothing (vice versa by using bei ‘gauss’).
  - Default Value: 0.5
  - Suggested values: Alpha \in \{0.1, 0.2, 0.3, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0\}
  - Typical range of values: 0.01 \leq Alpha \leq 0.01
  - Minimum Increment: 0.01
  - Recommended Increment: 0.1
  - Restriction: (Alpha > 0)

Example

info_smooth(‘deriche2’,0.5,Size,Coeffs)
smooth_image(Input,Smooth,’deriche2’,?)

Result

If the parameter values are correct the operator **SmoothImage** returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

**SmoothImage** is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors

ReadImage

Possible Successors

Threshold, DynThreshold, RegionGrowing

See also

InfoSmooth, MedianImage, SigmaImage, AnisotropeDiff

Alternatives

GaussImage, MeanImage, DerivateGauss

References


Module

Image filters
Smooth an image with an arbitrary rank mask.

The operator `TrimmedMean` carries out a non-linear smoothing of the gray values of all input images (`Image`). The filter mask (`Mask`) is passed in the form of a region. The average of `Number` gray values located near the median is calculated. Several border treatments can be chosen for filtering (`Margin`):

- `gray value` Pixels outside of the image edges are assumed to be constant (with the indicated gray value).
- `continued` Continuation of edge pixels.
- `cyclic` Cyclic continuation of image edges.
- `mirrored` Reflection of pixels at the image edges.

The indicated mask (= region of the mask image) is put over the image to be filtered in such a way that the center of the mask touches all pixels once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence according to their gray values. Thus, each of these sorted gray value sequences contains exactly as many gray values as the mask has pixels. If F is the area of the mask the average of these sequences is calculated as follows: The first (F - `Number`) gray values are ignored. Then the following `Number` gray values are summed up and divided by `Number`. Again the remaining (F - `Number`) gray values are ignored.

\[ \text{average} = \frac{\sum_{i=\text{F-Number+1}}^{\text{F}} \text{gray values}}{\text{Number}} \]

**Parameter**

- **Image** (input iconic) .... multichannel-image-array \( \rightsquigarrow \) `HIImageX / IHObjectX` (`byte`, `int2`, `uint2`, `int4`, `real`)
  - Image to be filtered.

- **Mask** (input iconic) ................................................................. region \( \rightsquigarrow \) `HRegionX / IHObjectX`
  - Image whose region serves as filter mask.

- **ImageTMean** (output iconic) ...... multichannel-image-array \( \rightsquigarrow \) `HIImageX / HUntypedObjectX` (`byte`, `int2`, `uint2`, `int4`, `real`)
  - Filtered output image.

- **Number** (input control) ................................................................. integer \( \rightsquigarrow \) `long / VARIANT`
  - Number of averaged pixels. Typical value: Surface(Mask) / 2.
  - **Default Value**: 5
  - **Suggested values**: `Number \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31\}`
  - **Typical range of values**: `1 \leq \text{Number} \leq 1`
  - **Minimum Increment**: 1
  - **Recommended Increment**: 2

- **Margin** (input control) ......................................................... string \( \rightsquigarrow \) `VARIANT` (integer, real, string)
  - Border treatment.
  - **Default Value**: `'mirrored'`
  - **Suggested values**: `Margin \in \{'mirrored', 'cyclic', 'continued', 0, 30, 60, 90, 120, 150, 180, 210, 240, 255\}`

**Example**

```c
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
trimmed_mean(Image,Region,TrimmedMean,5,'mirrored')
disp_image(TrimmedMean,WindowHandle).
```

**Result**

If the parameter values are correct the operator `TrimmedMean` returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator `SetSystem` (`'noObjectResult',<Result>`). If necessary an exception handling is raised.

**Parallelization Information**

`TrimmedMean` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

HALCON/COM Reference Manual, 2005-2-1
3.13. Texture

Possible Predecessors
ReadImage, DrawRegion, GenCircle, GenRectangle1

Possible Successors
Threshold, DynThreshold, RegionGrowing

See also
GenCircle, GenRectangle1, GrayErosionRect, GrayDilationRect

Alternatives
SigmaImage, MedianWeighted, MedianImage

References
R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 320

Image filters

3.13 Texture

[out] HImageX ImageDeviation
HImageX.DeviationImage ([in] long Width,
[in] long Height)

void HOperatorSetX.DeviationImage ([in] IHObjectX Image,
[out] HUntypedObjectX ImageDeviation, [in] VARIANT Width,
[in] VARIANT Height)

Calculate the standard deviation of gray values within rectangular windows.

DeviationImage calculates the standard deviation of gray values in the image Image within a rectangular mask of size (Height, Width). The resulting image is returned in ImageDeviation. To better use the range of gray values available in the output image, the result is multiplied by 2. If the parameters Height and Width are even, they are changed to the next larger odd value. At the image borders the gray values are mirrored.

Parameter

▶ Image (input iconic) …… (multichannel-)image(-array) $\sim$ HImageX / IHObjectX (byte, int4, real, int2, uint2)

Image for which the standard deviation is to be calculated.

▶ ImageDeviation (output iconic) …… image(-array) $\sim$ HImageX / HUntypedObjectX (byte, int4, real, int2, uint2)

Image containing the standard deviation.

▶ Width (input control) ……………………………………………. extent.x $\sim$ long / VARIANT

Width of the mask in which the standard deviation is calculated.

Default Value: 11
List of values: Width $\in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}$
Restriction: (Width $\&$ odd)

▶ Height (input control) …………………………………………. extent.y $\sim$ long / VARIANT

Height of the mask in which the standard deviation is calculated.

Default Value: 11
List of values: Height $\in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25\}$
Restriction: (Height $\&$ odd)

Example

read_image (Image, ‘fabrik’)
disp_image (Image, WindowHandle)
deviation_image (Image, Deviation, 9, 9)
disp_image (Deviation, WindowHandle).

Result

DeviationImage returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.
CHAPTER 3. FILTER

Parallelization Information

\textbf{DeviationImage} is reentrant and automatically parallelized (on tuple level, channel level).

Possible Successors

\textbf{DispImage}

See also \textbf{ConvolImage}, \textbf{TextureLaws}, \textbf{Intensity}

Alternatives

\textbf{EntropyImage}, \textbf{EntropyGray}

Module

Image filters

\begin{verbatim}
HImageX ImageEntropy HImageX.EntropyImage ([in] long Width, [in] long Height)

\end{verbatim}

Calculate the entropy of gray values within a rectangular window.

\textbf{EntropyImage} calculates the entropy of gray values in the image \textbf{Image} within a rectangular mask of size \textit{(Height, Width)}. The resulting image is returned in \textbf{ImageEntropy}, in which the entropy is multiplied by 32. If the parameters \textit{Height} and \textit{Width} are even, they are changed to the next larger odd value. At the image borders the gray values are mirrored.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Image} (input iconic) .............................. (multichannel-)image(-array) \rightsquigarrow HImageX / IHObjectX (byte)</td>
</tr>
<tr>
<td>\textbf{ImageEntropy} (output iconic) .............................. (multichannel-)image(-array) \rightsquigarrow HImageX / HUntypedObjectX (byte)</td>
</tr>
<tr>
<td>\textbf{Width} (input control) .......................................................... extent.x \rightsquigarrow long / VARIANT</td>
</tr>
<tr>
<td>Default Value : 9</td>
</tr>
<tr>
<td>Suggested values : Width \in {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>List of values : Width \in {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}</td>
</tr>
<tr>
<td>Restriction : ( (Width \land \text{odd}) )</td>
</tr>
<tr>
<td>\textbf{Height} (input control) .......................................................... extent.y \rightsquigarrow long / VARIANT</td>
</tr>
<tr>
<td>Default Value : 9</td>
</tr>
<tr>
<td>Suggested values : Height \in {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>List of values : Height \in {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}</td>
</tr>
<tr>
<td>Restriction : ( (Height \land \text{odd}) )</td>
</tr>
</tbody>
</table>

Example

\begin{verbatim}
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
entropy_image(Image,Entropy1,9,9)
disp_image(Entropy1,WindowHandle).
\end{verbatim}

Result

\textbf{EntropyImage} returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via \textit{SetSystem('noObjectResult',<Result>}). If necessary, an exception handling is raised.

Parallelization Information

\textbf{EntropyImage} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

\textbf{DispImage}
3.13. TEXTURE

See also EnergyGabor, EntropyGray

Alternatives EntropyGray

Module Image filters

Filter an image using a Laws texture filter.

TextureLaws applies one or more texture transformations (according to Laws) to an image. This is done by convolving the input image with one (or more) filter masks. The filters are:

9 different 3x3 matrices obtainable from the following three vectors:

\[ l = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \]
\[ e = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \]
\[ s = \begin{bmatrix} -1 & 2 \end{bmatrix} \]

25 different 5x5 matrices obtainable from the following five vectors:

\[ l = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} \]
\[ e = \begin{bmatrix} -1 & -2 & 0 & 2 & 1 \end{bmatrix} \]
\[ s = \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \end{bmatrix} \]
\[ r = \begin{bmatrix} 1 & -4 & 6 & -4 & 1 \end{bmatrix} \]
\[ w = \begin{bmatrix} -1 & 2 & 0 & -2 & 1 \end{bmatrix} \]

36 different 7x7 matrices obtainable from the following six vectors:

\[ l = \begin{bmatrix} 1 & 6 & 15 & 20 & 15 & 6 & 1 \end{bmatrix} \]
\[ e = \begin{bmatrix} -1 & -4 & -5 & 0 & 5 & 4 & 1 \end{bmatrix} \]
\[ s = \begin{bmatrix} -1 & -2 & 1 & 4 & 1 & -2 & -1 \end{bmatrix} \]
\[ r = \begin{bmatrix} -1 & -2 & -1 & 4 & -1 & -2 & -1 \end{bmatrix} \]
\[ w = \begin{bmatrix} -1 & 0 & 3 & 0 & -3 & 0 & 1 \end{bmatrix} \]
\[ o = \begin{bmatrix} -1 & 6 & -15 & 20 & -15 & 6 & -1 \end{bmatrix} \]

For most of the filters the resulting gray values must be modified by a Shift. This makes the different textures in the output image more comparable to each other, provided suitable filters are used.

The name of the filter is composed of the letters of the two vectors used, where the first letter denotes convolution in the column direction while the second letter denotes convolution in the row direction.

If more than one filter type is passed, a multi-channel image is returned for each (single-channel) input image, with each channel corresponding to a particular filter.

Parameter

- **Image** (input iconic) ..........(multichannel-)image(-array) \(\sim\) HImageX / HObjectX (byte, int2, uint2)
  Images to which the texture transformation is to be applied.

- **ImageTexture** (output iconic) .......(multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, int2, uint2)
  Texture images.
CHAPTER 3. FILTER

FilterTypes (input control) ......................................... string(-array) ~ VARIANT (string)
Desired filters (name or number).
Default Value: ‘el’

Shift (input control) .................................................. integer ~ long / VARIANT
Shift to reduce the gray value dynamics.
Default Value: 2
List of values: Shift ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

FilterSize (input control) ............................................ integer ~ long / VARIANT
Size of the filter kernel.
Default Value: 5
List of values: FilterSize ∈ {3, 5, 7}

Example

/* 2 dimensional pixel classification */
read_image(Image,’combine’)
open_window(0,0,-1,-1,’root’,’visible’,’’,WindowHandle)
disp_image(Image,WindowHandle)
texture_laws(Image,Texture1,’es’,2,5)
texture_laws(Image,Texture2,’le’,2,5)
mean_image(Texture1,H1,51,51)
mean_image(Texture2,H2,51,51)
fwrite_string(FileId,’mark desired image section’)
fnew_line(FileId)
set_color(WindowHandle,’green’)
draw_region(Region,WindowHandle)
reduce_domain(H1,Region,Foreground1)
reduce_domain(H2,Region,Foreground2)
histo_2dim(Region,Foreground1,Foreground2,Histo)
threshold(Histo,Characteristic_area,1,1000000)
set_color(WindowHandle,’blue’)
disp_region(Characteristic_area,WindowHandle)
class_2dim_sup(H1,H2,Characteristic_area,Seg,4,5)
set_color(WindowHandle,’red’)
disp_region(Seg,WindowHandle).

Result

TextureLaws returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via
SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

TextureLaws is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Successors

MeanImage, GaussImage, MedianImage, Histo2Dim, LearnNdimNorm, LearnNdimBox, Threshold

See also

Class2DimSup, ClassNdimNorm

Alternatives

Conv1Image

References


Module

Image filters
3.14 Wiener-Filter

```c
void HImageX.GenPsfDefocus ([in] long PSFwidth, [in] long PSFheight,
[in] double Blurring )
```

```c
void HOperatorSetX.GenPsfDefocus ([out] HUntypedObjectX Psf,
```

Generate an impulse response of an uniform out-of-focus blurring.

**GenPsfDefocus** generates an impulse response (spatial domain) of an uniform out-of-focus blurring and writes it into an image of HALCON image type ‘real’. **Blurring** specifies the extent of blurring by defining the ”"blur radius”” (out-of-focus blurring maps each image pixel on a small circle with a radius of **Blurring** - specified in ”"number of pixels””). If specified less than zero, the absolute value of **Blurring** is used. The result image of **GenPsfDefocus** encloses an spatial domain impulse response of the specified blurring. Its representation presumes the origin in the upper left corner. This results in the following disposition of an $N\times M$ sized image:

- first rectangle (“"left””): (image coordinates $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  - conforms to the fourth quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = 0..N/2$ and $y = 0..M/2$
- second rectangle (“"upper right””): (image coordinates $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  - conforms to the third quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = -N/2..1$ and $y = -1..M/2$
- third rectangle (“"lower right””): (image coordinates $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  - conforms to the first quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = 1..N/2$ and $y = M/2..0$
- fourth rectangle (“"lower left””): (image coordinates $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  - conforms to the second quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = -N/2..1$ and $y = M/2..1$

This representation conforms to that of the impulse response parameter of the HALCON-operator **WienerFilter**. So one can use **GenPsfDefocus** to generate an impulse response for Wiener filtering.

### Parameter

- **Psf** (output iconic) .............................................. image $\sim$ HImageX / HUntypedObjectX ( real )
  > Impulse response of uniform out-of-focus blurring.
- **PSFwidth** (input control) ............................................. integer $\sim$ long / VARIANT
  > Width of result image.
  > **Default Value**: 256
  > **Suggested values**: PSFwidth $\in \{128, 256, 512, 1024\}$
- **PSFheight** (input control) ............................................. integer $\sim$ long / VARIANT
  > Height of result image.
  > **Default Value**: 256
  > **Suggested values**: PSFheight $\in \{128, 256, 512, 1024\}$
- **Blurring** (input control) ............................................. real $\sim$ double / VARIANT
  > Degree of Blurring.
  > **Default Value**: 5.0
  > **Suggested values**: Blurring $\in \{1.0, 5.0, 10.0, 15.0, 18.0\}$

### Result

**GenPsfDefocus** returns TRUE if all parameters are correct.

**Parallelization Information**

**GenPsfDefocus** is reentrant and processed without parallelization.

**Possible Predecessors**

**SimulateMotion, GenPsfMotion**

**Possible Successors**

**SimulateDefocus, WienerFilter, WienerFilterNi**

**See also**

**SimulateDefocus, GenPsfMotion, SimulateMotion, WienerFilter, WienerFilterNi**

HALCON 6.1.4
Generates an impulse response (spatial domain) of a blurring caused by a relative motion between the object and the camera during exposure. The generated impulse response is output into an image of HALCON image type 'real'. PSFwidth and PSFheight define the width and height of the output image. The blurring motion moves along an even. Angle fixes its direction by specifying the angle between the motion direction and the x-axis (anticlockwise, measured in degrees). To specify different velocity behaviour five PSF prototypes can be generated. Type switches between the following prototypes:

1. reverse ramp (crude model for acceleration)
2. reverse trapezoid (crude model for high acceleration)
3. square pulse (exact model for constant velocity), this is default
4. forward trapezoid (crude model for deceleration)
5. forward ramp (crude model for high deceleration)

The blurring affects all part of the image uniformly. Blurring controls the extent of blurring. It specifies the number of pixels (lying one after another) that are affectected by the blurring. This number is determined by velocity of the motion and exposure time. If Blurring is a negative number, an adequate blurring in reverse direction is simulated. If Angle is a negative number, it is interpreted clockwise. If Angle exceeds 360 or falls below -360, it is transformed modulo(360) in an adequate number between [0..360] resp. [-360..0]. The result image of GenPsfMotion encloses an spatial domain impulse response of the specified blurring. Its representation presumes the origin in the upper left corner. This results in the following disposition of an \( N \times M \) sized image:

- first rectangle ("upper left"): (image coordinates \( x_b = 0..(N/2) - 1, y_b = 0..(M/2) - 1 \)) - conforms to the fourth quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = 0..N/2 \) and \( y = 0..M/2 \)
- second rectangle ("upper right"): (image coordinates \( x_b = N/2..N - 1, y_b = 0..(M/2) - 1 \)) - conforms to the third quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = -N/2..-1 \) and \( y = -1..-M/2 \)
- third rectangle ("lower left"): (image coordinates \( x_b = 0..(N/2) - 1, y_b = M/2..M - 1 \)) - conforms to the first quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = 1..N/2 \) and \( y = M/2..0 \)
- fourth rectangle ("lower right"): (image coordinates \( x_b = N/2..N - 1, y_b = M/2..M - 1 \)) - conforms to the second quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \( x = -N/2..-1 \) and \( y = M/2..1 \)

This representation conforms to that of the impulse response parameter of the HALCON-operator WienerFilter. So one can use GenPsfMotion to generate an impulse response for Wiener filtering a motion blurred image.
Parameter

- **Psf** (output iconic) .......................... image .workflow>HImageX / HUntypedObjectX (real)
  Impulse response of motion-blur.

- **PSFwidth** (input control) ........................ integer .workflow>long / VARIANT
  Width of impulse response image.
  Default Value: 256
  Suggested values: PSFwidth ∈ {128, 256, 512, 1024}

- **PSFheight** (input control) ........................ integer .workflow>long / VARIANT
  Height of impulse response image.
  Default Value: 256
  Suggested values: PSFheight ∈ {128, 256, 512, 1024}

- **Blurring** (input control) .......................... real .workflow>double / VARIANT
  Degree of motion-blur.
  Default Value: 20.0
  Suggested values: Blurring ∈ {5.0, 10.0, 20.0, 30.0, 40.0}

- **Angle** (input control) .......................... integer .workflow>long / VARIANT
  Angle between direction of motion and x-axis (anticlockwise).
  Default Value: 0
  Suggested values: Angle ∈ {0, 45, 90, 180, 270}

- **Type** (input control) .......................... integer .workflow>long / VARIANT
  PSF prototype resp. type of motion.
  Default Value: 3
  List of values: Type ∈ {1, 2, 3, 4, 5}

Result

GenPsfMotion returns TRUE if all parameters are correct.

Parallelization Information

GenPsfMotion is reentrant and processed without parallelization.

Possible Predecessors

GenPsfMotion, SimulateDefocus, GenPsfDefocus

Possible Successors

SimulateMotion, WienerFilter, WienerFilterNi

See also

SimulateMotion, SimulateDefocus, GenPsfDefocus, WienerFilter, WienerFilterNi

References


M. L"uckenhaus: “Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse”; Diplomarbeit; Technische Universit"at M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995.


Module

Wiener filter

```c
HImageX DefocusedImage HImageX.SimulateDefocus
([in] double Blurring )

void HOperatorSetX.SimulateDefocus ([in] IObjectX Image,
[out] HUntypedObjectX DefocusedImage, [in] VARIANT Blurring )
```

Simulate an uniform out-of-focus blurring of an image.

**SimulateDefocus** simulates out-of-focus blurring of an image. All parts of the image are blurred uniformly. **Blurring** specifies the extent of blurring by defining the “blur radius” (out-of-focus blurring maps each image pixel on a small circle with a radius of **Blurring** - specified in “number of pixels”). If specified less than null,
the absolute value of Blurring is used. Simulation of blurring is done by a convolution of the image with a blurring specific impulse response. The convolution is realized by multiplication in the Fourier domain.  

As the simulation of the blurring is realized based on the Fourier transform, image height and width must be a power of 2, e.g. 128, 256, 512,...

Parameter

▷ Image (input iconic) .......... image  \(\sim HImageX / IHObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real)

Image to blur.

▷ DefocusedImage (output iconic) ......... image  \(\sim HImageX / HUntypedObjectX\) (real)

Blurred image.

▷ Blurring (input control) ...................... real  \(\sim double / VARIANT\)

Degree of blurring.

Default Value : 5.0

Suggested values : Blurring \(\in\) \{1.0, 5.0, 10.0, 15.0, 18.0\}

Result

SimulateDefocus returns TRUE if all parameters are correct. If the input is empty SimulateDefocus returns with an error message.

Parallelization Information

SimulateDefocus is reentrant and processed without parallelization.

Possible Predecessors

GenPsfDefocus, SimulateMotion, GenPsfMotion

Possible Successors

WienerFilter, WienerFilterNi

See also

GenPsfDefocus, SimulateMotion, GenPsfMotion

References


M. L"uckenhaus:’’Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse’’; Diplomarbeit; Technische Universit"at M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995.

Module

Wiener filter

Simulation of (linearly) motion blur.

SimulateMotion simulates blurring caused by a relative motion between the object and the camera during exposure. The simulated motion moves along an even. Angle fixes its direction by specifying the angle between the motion direction and the x-axis (anticlockwise, measured in degrees). Simulation is done by a convolution of the image with a blurring specific impulse response. The convolution is realized by multiplication in the Fourier domain. SimulateMotion offers five prototypes of impulse responses conforming to different acceleration behaviours. Type allows to choose one of the following PSF prototypes:

1. reverse ramp (crude model for acceleration)
2. reverse trapezoid (crude model for high acceleration)
3. square pulse (exact model for constant velocity), this is default
4. forward trapezoid (crude model for deceleration)
5. forward ramp (crude model for high deceleration)

The simulated blurring affects all part of the image uniformly. **Blurring** controls the extent of blurring. It specifies the number of pixels (lying one after another) that are affected by the blurring. This number is determined by velocity of the motion and exposure time. If **Blurring** is a negative number, an adequate blurring in reverse direction is simulated. If **Angle** is a negative number, it is interpreted clockwise. If **Angle** exceeds 360 or falls below -360, it is transformed modulo(360) in an adequate number between [0..360] resp. [-360..0].

---

**Attention**

As the simulation of the blurring is realized based on the Fourier transform, image height and width must be a power of 2, e.g. 128, 128, 256, 512, ...

---

### Parameter

- **Image** (input iconic) ...... image  \( \sim HImageX / IObjectX (\text{byte}, \text{direction}, \text{cyclic}, \text{int1}, \text{int2}, \text{uint2}, \text{int4}, \text{real}) \)

  - image to be blurred.

- **MovedImage** (output iconic) ............... image  \( \sim HImageX / HUntypedObjectX (\text{real}) \)

  - motion blurred image.

- **Blurring** (input control) .......................................................... real  \( \sim \text{double} / \text{VARIANT} \)

  - extent of blurring.

  - Default Value: 20.0

  - Suggested values: **Blurring** \( \in \{5.0, 10.0, 20.0, 30.0, 40.0\} \)

- **Angle** (input control) .......................................................... integer  \( \sim \text{long} / \text{VARIANT} \)

  - Angle between direction of motion and x-axis (anti-clockwise).

  - Default Value: 0

  - Suggested values: **Angle** \( \in \{0, 45, 90, 180, 270\} \)

- **Type** (input control) .......................................................... integer  \( \sim \text{long} / \text{VARIANT} \)

  - impulse response of motion blur.

  - Default Value: 3

  - List of values: **Type** \( \in \{1, 2, 3, 4, 5\} \)

---

**SimulateMotion** returns TRUE if all parameters are correct. If the input is empty **SimulateMotion** returns with an error message.

---

**Parallelization Information**

**SimulateMotion** is reentrant and processed without parallelization.

---

**Possible Predecessors**

- GenPsfMotion, GenPsfMotion

---

**Possible Successors**

- SimulateDefocus, WienerFilter, WienerFilterNi

---

**See also**

- GenPsfMotion, SimulateDefocus, GenPsfDefocus

---

**References**


M. L"uckenhaus: "Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse"; Diplomarbeit; Technische Universit"at M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995.


---

**Module**

Wiener filter
**Image restoration by Wiener filtering.**

**WienerFilter** produces an estimate of the original image (= image without noise and blurring) by minimizing the mean square error between estimated and original image. **WienerFilter** can be used to restore images corrupted by noise and/or blurring (e.g. motion blur, atmospheric turbulence or out-of-focus blur). Method and realisation of this restoration technique bases on the following model: The corrupted image is interpreted as the output of a (disturbed) linear system. Functionality of a linear system is determined by its specific impulse response. So the convolution of original image and impulse response results in the corrupted image. The specific impulse response describes image acquisition and the occured degradations. In the presence of additive noise an additional noise term must be considered. So the corrupted image can be modeled as the result of

\[
[\text{convolution}(\text{impulse response, original image})] + \text{noise term}
\]

The noise term encloses two different terms describing image-dependent and image-independent noise. According to this model, two terms must be known for restoration by Wiener filtering:

1. degradation-specific impulse response
2. noise term

So **WienerFilter** needs a smoothed version of the input image to estimate the power spectral density of noise and original image. One can use one of the smoothing HALCON-filters (e.g. **EliminateMinMax**) to get this version. **WienerFilter** needs further the impulse response that describes the specific degradation. This impulse response (represented in spatial domain) must fit into an image of HALCON image type 'real'. There exist two HALCON-operators for generation of an impulse response for motion blur and out-of-focus (see **GenPsfMotion**, **GenPsfDefocus**). The representation of the impulse response presumes the origin in the upper left corner. This results in the following disposition of an \(N\times M\) sized image:

- first rectangle ("upper left"): (image coordinates \(x_b = 0..(N/2)−1, y_b = 0..(M/2)−1\))
  - conforms to the fourth quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \(x = 0..N/2\) and \(y = 0..−M/2\)
- second rectangle ("upper right"): (image coordinates \(x_b = N/2..N−1, y_b = 0..(M/2)−1\))
  - conforms to the third quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \(x = −N/2..−1\) and \(y = −1..−M/2\)
- third rectangle ("lower left"): (image coordinates \(x_b = 0..(N/2)−1, y_b = M/2..M−1\))
  - conforms to the first quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \(x = 0..N/2\) and \(y = M/2..0\)
- fourth rectangle ("lower right"): (image coordinates \(x_b = N/2..N−1, y_b = M/2..M−1\))
  - conforms to the second quadrant of the Cartesian coordinate system, encloses values of the impulse response at position \(x = −N/2..−1\) and \(y = M/2..1\)

As the Wiener filter is realized using the Fourier transform, only images with height and width fitting a power of 2 can be processed. **WienerFilter** works as follows:

- estimation of the power spectrum density of the original image by using the smoothed version of the corrupted image,
- estimation of the power spectrum density of each pixel by subtracting smoothed version from unsmoothed version,
- building the Wiener filter kernel with the quotient of power spectrum densities of noise and original image and with the impulse response,
- processing the convolution of image and Wiener filter frequency response.

The result image has got image type ‘real’.

**Attention**

Psf must be of image type ‘real’ and conform to Image and FilteredImage in image width and height.

---

HALCON/COM Reference Manual, 2005-2-1
Image restoration by Wiener filtering.

\textit{WienerFilterNi} (ni = noise-estimation integrated) produces an estimate of the original image (= image without noise and blurring) by minimizing the mean square error between estimated and original image. \textit{WienerFilter} can be used to restore images corrupted by noise and/or blurring (e.g. motion blur, atmospheric turbulence or out-of-focus blur). Method and realisation of this restoration technique bases on the following model:

The corrupted image is interpreted as the output of a (disturbed) linear system. Functionality of a linear system is determined by its specific impulse response. So the convolution of original image and impulse response results in the corrupted image. The specific impulse response describes image acquisition and the occured degradations. In the presence of additive noise an additional noise term must be considered. So the corrupted image can be modeled as the result of

\[
\text{convolution(impulse\_response, original\_image)} + \text{noise\_term}
\]

The noise term encloses two different terms describing image-dependent and image-independent noise. According to this model, two terms must be known for restoration by Wiener filtering:

1. degradation-specific impulse response
2. noise term

WienerFilterNi estimates the noise term as follows: The user defines a region that is suitable for noise estimation within the image (homogeneous as possible, as edges or textures aggravate noise estimation). After smoothing within this region by an (unweighted) median filter and subtracting smoothed version from unsmoothed, the average noise amplitude of the region is processed within WienerFilterNi. This amplitude together with the average gray value within the region allows estimating the quotient of the power spectral densities of noise and original image (in contrast to WienerFilter WienerFilterNi assumes a rather constant quotient within the whole image). The user can define width and height of the rectangular (median-)filter mask to influence the noise estimation (MaskWidth, MaskHeight). WienerFilterNi needs further the impulse response that describes the specific degradation. This impulse response (represented in spatial domain) must fit into an image of HALCON image type 'real'. There exist two HALCON-operators for generation of an impulse response for motion blur and out-of-focus (see GenPsfMotion, GenPsfDefocus). The representation of the impulse response presumes the origin in the upper left corner. This results in the following disposition of an $NxM$ sized image:

- first rectangle ("upper left"): (image coordinates $x = 0..(N/2) - 1$, $y = 0..(M/2) - 1$)
  - conforms to the fourth quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = 0..N/2$ and $y = 0..M/2$
- second rectangle ("upper right"): (image coordinates $x = N/2..N - 1$, $y = 0..(M/2) - 1$)
  - conforms to the third quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = -N/2..-1$ and $y = -1..-M/2$
- third rectangle ("lower left"): (image coordinates $x = 0..(N/2) - 1$, $y = M/2..M - 1$)
  - conforms to the first quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = 1..N/2$ and $y = M/2..0$
- fourth rectangle ("lower right"): (image coordinates $x = N/2..N - 1$, $y = M/2..M - 1$)
  - conforms to the second quadrant of the Cartesian coordinate system, encloses values of the impulse response at position $x = -N/2..-1$ and $y = M/2..1$

As the Wiener filter is realized using the Fourier transform, only images with height and width fitting a power of 2 can be processed. WienerFilter works as follows:

- estimating the quotient of the power spectrum densities of noise and original image,
- building the Wiener filter kernel with the quotient of power spectrum densities of noise and original image and with the impulse response,
- processing the convolution of image and Wiener filter frequency response.

The result image has got image type 'real'.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>(input iconic) . . . image $\sim H$ImageX / HObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real )</td>
</tr>
<tr>
<td>Psf</td>
<td>(input iconic) . . . . . . . . image $\sim H$ImageX / HObjectX ( real )</td>
</tr>
<tr>
<td>NoiseRegion</td>
<td>(input iconic) . . . region $\sim H$RegionX / HObjectX</td>
</tr>
<tr>
<td>RestoredImage</td>
<td>(output iconic) . . . . image $\sim H$ImageX / HUntypedObjectX ( real )</td>
</tr>
<tr>
<td>MaskWidth</td>
<td>(input control) . . . . integer $\sim$ long / VARIANT</td>
</tr>
</tbody>
</table>

Default Value: 3  
Suggested values: $\text{MaskWidth} \in \{3, 5, 7, 9\}$  
Typical range of values: $0 \leq \text{MaskWidth} \leq 0$

HALCON/COM Reference Manual, 2005-2-1
3.14. WIENER-FILTER

MaskHeight (input control) ............................................. integer  \rightarrow long / VARIANT
Height of filter mask.

Default Value : 3
Suggested values : MaskHeight ∈ \{3, 5, 7, 9\}
Typical range of values : 0 \leq MaskHeight \leq 0

WienerFilterNi returns TRUE if all parameters are correct. If the input is empty WienerFilterNi returns
with an error message.

Parallelization Information

WienerFilterNi is reentrant and processed without parallelization.

Possible Predecessors

GenPsfMotion, SimulateMotion, SimulateDefocus, GenPsfDefocus

See also

SimulateMotion, GenPsfMotion, SimulateDefocus, GenPsfDefocus

Alternatives

WienerFilter

References

M. L"uckenhaus: "Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse"; Diplomarbeit;
Technische Universit"at M"unchen, Institut f"ur Informatik; Lehrstuhl Prof. Radig; 1995


Module

Wiener filter
Chapter 4

Graphics

4.1 Drawing

Interactive moving of a region.

`DragRegion1` is used to move a region on the display by mouse. Calling `DragRegion1` turns the region visible as soon as the left mouse button is pressed. Therefore the region's edges are displayed only. As representation mode the mode 'not' (see `SetDraw`) is used during procedure's permanence. During the movement the cursor resides in the region's barycenter. If you move the mouse with pressed left mouse button, the depicted region follows - delayed - this movement. If you press the right mouse button you terminate `DragRegion1`. The depicted region disappears from the display. Output is a region which corresponds to the last position on the display. You may pass even several regions at once. Procedure `AffineTransImage` moves the gray values.

Attention

Gray values of regions are not moved. With moving the input region it is not sure whether the gray values of the output regions are filled reasonable. This may occur if the gray values of the input regions do not comprise the whole image.

Parameter

- `SourceRegion` (input iconic) ......................... region ~ HRegionX / IHObjectX
- `DestinationRegion` (output iconic) .................. region ~ HRegionX / HUntypedObjectX
- `WindowHandle` (input control) ...................... window ~ HWindowX / VARIANT

Example

draw_region(Obj,WindowHandle)
drag_region1(Obj,New,WindowHandle)
disp_region(New,WindowHandle)
position(Obj,_,Row1,Column1,_,_,_,_)
position(New,_,Row2,Column2,_,_,_,_)
disp_arrow(WindowHandle,Row1,Column1,Row2,Column2,1.0)
fwrite_string(['Transformation: ('Row2-Row1,'/',Column2-Column1.']})
fnew_line().

169
DragRegion1 returns TRUE, if a region is entered, the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised. You may determine the behavior after an empty input with SetSystem(:mm:’noObjectResult’,<Result>:).

Parallelization Information

DragRegion1 is reentrant, local, and processed without parallelization.

Possible Predecessors

Possible Successors

See also

Alternatives

Module

Interactive movement of a region with fixpoint specification.

You use DragRegion2 to move a region on the display by mouse. It corresponds to the procedure DragRegion1 with the difference, that the position of the mouse cursor can be determined.

Attention

Gray values of the regions are not moved. With moving the input region it is not sure whether the gray values of the output regions are filled reasonable. This may occur if the gray values of the input regions do not comprise the whole image.

Parameter

- SourceRegion (input iconic) .......................... region \sim HRegionX / IObjectX
  Regions to move.

- DestinationRegion (output iconic) ................region \sim HRegionX / HUntypedObjectX
  Moved regions.

- WindowHandle (input control) ...................... window \sim HWindowX / VARIANT
  Window_id.

- Row (input control) .................................. point.y \sim long / VARIANT
  Row index of the reference point.
  Default Value : 100
  Suggested values : Row \in \{0, 64, 128, 256, 512\}
  Typical range of values : 0 \leq Row \leq 0

- Column (input control) .............................. point.x \sim long / VARIANT
  Column index of the reference point.
  Default Value : 100
  Suggested values : Column \in \{0, 64, 128, 256, 512\}
  Typical range of values : 0 \leq Column \leq 0

Result

DragRegion2 returns TRUE, if a region is entered, the window is valid and the needed drawing mode (see
SetInsert) is available. If necessary, an exception handling is raised. You may determine the behavior after an empty input with SetSystem(::{'noObjectResult',<Result>;}).

Parallelization Information

DragRegion2 is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow

Possible Successors

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert, AffineTransImage

See also

SetInsert, SetDraw, AffineTransImage

Alternatives

GetMposition, MoveRegion, DragRegion1, DragRegion3

Module

System

Interactive movement of a region with restriction of positions.

You use DragRegion3 to move a region on the display by mouse. It corresponds to the procedure DragRegion2 with the enhancement, that all points are specified which can be entered by mouse. If you move the mouse outside of this area (MaskRegion), the region on the point with the smallest distance inside MaskRegion will be displayed.

Attention

The region’s gray values are not moved. With moving the input region it is not sure whether the gray values of the output regions are filled reasonable. This may occur if the gray values of the input regions do not comprise the whole image.

Parameter

- **SourceRegion** (input iconic) ........................................ region  \(\rightarrow\) HRegionX / IObjectX
  Regions to move.
- **MaskRegion** (input iconic) ........................................ region  \(\rightarrow\) HRegionX / IObjectX
  Points on which it is allowed for a region to move.
- **DestinationRegion** (output iconic) .......................... region  \(\rightarrow\) HRegionX / HUntypedObjectX
  Moved regions.
- **WindowHandle** (input control) ................................. window  \(\rightarrow\) HWindowX / VARIANT
  Window id.
- **Row** (input control) ............................................. point.y  \(\rightarrow\) long / VARIANT
  Row index of the reference point.

Default Value: 100

Suggested values: Row \(\in\) \{0, 64, 128, 256, 512\}

Typical range of values: 0 \(\leq\) Row \(\leq\) 0
> **Column** (input control) ............................... point.x \(\sim\) long / VARIANT
> Column index of the reference point.
> **Default Value:** 100
> **Suggested values:** Column \(\in\) \{0, 64, 128, 256, 512\}
> **Typical range of values:** \(0 \leq \text{Column} \leq 0\)

**Result**

DragRegion3 returns TRUE, if a region is entered, if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised. You may determine the behavior after an empty input with SetSystem(\{\{\{'noObjectResult', <Result>\}\}).

**Parallelization Information**

DragRegion3 is reentrant, local, and processed without parallelization.

**Possible Predecessors**

OpenWindow, GetMposition

**Possible Successors**

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert, AffineTransImage

**See also**

SetInsert, SetDraw, AffineTransImage

**Alternatives**

GetMposition, MoveRegion, DragRegion1, DragRegion2

**Module**

System

```
[out] double Row HWindowX.DrawCircle ([out] double Column,
[out] double Radius )

void HOperatorSetX.DrawCircle ([in] VARIANT WindowHandle,
[out] VARIANT Row, [out] VARIANT Column, [out] VARIANT Radius )
```

**Interactive drawing of a circle.**

**DrawCircle** produces the parameter for a circle created interactive by the user in the window.

To create a circle you have to press the mouse button at the location which is used as the center of that circle. While keeping the mouse button pressed, the **Radius**'s length can be modified through moving the mouse. After another mouse click in the created circle center you can move it. A clicking close to the circular arc you can modify the **Radius** of the circle. Pressing the right mouse button terminates the procedure. After terminating the procedure the circle is not visible in the window any longer.

**Parameter**

- **WindowHandle** (input control) ............................... window \(\sim\) HWindowX / VARIANT Window_id.
- **Row** (output control) ............................... circle.center.y \(\sim\) double / VARIANT Barycenter's row index.
- **Column** (output control) ............................... circle.center.x \(\sim\) double / VARIANT Barycenter's column index.
- **Radius** (output control) ............................... circle.radius \(\sim\) double / VARIANT Circle's radius.

**Example**

```c
read_image(Image, ‘affe’)
draw_circle(WindowHandle, Row, Column, Radius)
gen_circle(Circle, Row, Column, Radius, )
reduce_domain(Image, Circle, GrayCircle)
disp_image(GrayCircle, WindowHandle).
```
4.1. DRAWING 173

Result

DrawCircle returns TRUE if the window is valid and the needed drawing mode (see SetInsert) is available.

Parallelization Information

DrawCircle is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow

Possible Successors

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

See also

GenCircle, DrawRectangle1, DrawRectangle2, DrawPolygon, SetInsert

Alternatives

DrawCircleMod, DrawEllipse, DrawRegion

Module

System

---

Interactive drawing of a circle.

DrawCircleMod produces the parameter for a circle created interactive by the user in the window.

To create a circle are expected the coordinates RowIn and ColumnIn of the center of a circle with radius RadiusIn. After another mouse click in the created circle center you can move it. A clicking close to the circular arc you can modify the Radius of the circle. Pressing the right mousebutton terminates the procedure. After terminating the procedure the circle is not visible in the window any longer.

Parameter

- **WindowHandle** (input control) window \( \sim \) HWindowX / VARIANT Window_id.
- **RowIn** (input control) circle.center.y \( \sim \) double / VARIANT Row index of the center.
- **ColumnIn** (input control) circle.center.x \( \sim \) double / VARIANT Column index of the center.
- **RadiusIn** (input control) circle.radius1 \( \sim \) double / VARIANT Radius of the circle.
- **Row** (output control) circle.center.y \( \sim \) double / VARIANT Barycenter’s row index.
- **Column** (output control) circle.center.x \( \sim \) double / VARIANT Barycenter’s column index.
- **Radius** (output control) circle.radius \( \sim \) double / VARIANT Circle’s radius.

Example

read_image(Image,’affe’)
draw_circle_mod(WindowHandle,20,20,15,Row,Column,Radius)
gen_circle(Circle,Row,Column,Radius,)
reduce_domain(Image,Circle,GrayCircle)
disp_image(GrayCircle,WindowHandle).

HALCON 6.1.4
Result

DrawCircleMod returns TRUE if the window is valid and the needed drawing mode (see SetInsert) is available. If necessary, an exception handling is raised.

Parallelization Information

DrawCircleMod is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow

Possible Successors

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

See also

GenCircle, DrawRectangle1, DrawRectangle2, DrawPolygon, SetInsert

Alternatives

DrawCircle, DrawEllipse, DrawRegion

Module

System

Interactive drawing of an ellipse.

DrawEllipse returns the parameter for any orientated ellipse, which has been created interactively by the user in the window.

The created ellipse is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create an ellipse you have to determine the center of the ellipse with the left mouse button. Keeping the button pressed determines the length (Radius1) and the orientation (Phi) of the first half axis. In doing so a temporary default length for the second half axis is assumed, which may be modified afterwards on demand. After another mouse click in the center of the created ellipse you can move it. A mouse click close to a vertex “grips” it to modify the length of the appropriate half axis. You may modify the orientation only, if a vertex of the first half axis is gripped.

Pressing the right mouse button terminates the procedure. After terminating the procedure the ellipse is not visible in the window any longer.

Parameter

- WindowHandle (input control) .............................. window  ~ HWindowX / VARIANT Window_id.
- Row (output control) ............................... ellipse.center.y  ~ double / VARIANT Row index of the center.
- Column (output control) ............................... ellipse.center.x  ~ double / VARIANT Column index of the center.
- Phi (output control) ............................... ellipse.angle.rad  ~ double / VARIANT Orientation of the first half axis in radians.
- Radius1 (output control) ............................... ellipse.radius1  ~ double / VARIANT First half axis.
- Radius2 (output control) ............................... ellipse.radius2  ~ double / VARIANT Second half axis.

Example

read_image(Image, ‘affe’)
draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
gen_ellipse(Ellipse, Row, Column, Phi, Radius1, Radius2)
reduce_domain(Image, Ellipse, GrayEllipse)
sobel_amp(GrayEllipse, Sobel, ‘sum_abs’, 3)
disp_image(Sobel, WindowHandle).

**Result**

`DrawEllipse` returns `TRUE`, if the window is valid and the needed drawing mode (see `SetInsert`) is available. If necessary, an exception handling is raised.

**Parallelization Information**

`DrawEllipse` is reentrant, local, and processed without parallelization.

**Possible Predecessors**

`OpenWindow`

**Possible Successors**

`ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert`

**See also**

`GenEllipse, DrawRectangle1, DrawRectangle2, DrawPolygon, SetInsert`

**Alternatives**

`DrawEllipseMod, DrawCircle, DrawRegion`

**Module**

**System**

Interactive drawing of an ellipse.

`DrawEllipseMod` returns the parameter for any orientated ellipse, which has been created interactively by the user in the window.

The created ellipse is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create an ellipse are expected the parameters `RowIn`, `ColumnIn`, `PhiIn`, `Radius1In`, `Radius2In`. Keeping the button pressed determines the length (`Radius1`) and the orientation (`Phi`) of the first half axis. In doing so a temporary default length for the second half axis is assumed, which may be modified afterwards on demand. After another mouse click in the center of the created ellipse you can move it. A mouse click close to a vertex “grips” it to modify the length of the appropriate half axis. You may modify the orientation only, if a vertex of the first half axis is gripped.

Pressing the right mouse button terminates the procedure. After terminating the procedure the ellipse is not visible in the window any longer.

**Parameter**

- `WindowHandle` (input control) ......................... window ~ HWindowX / VARIANT WindowId.
- `RowIn` (input control) ................................. ellipse.center.y ~ double / VARIANT Row index of the barycenter.
- `ColumnIn` (input control) .............................. ellipse.center.x ~ double / VARIANT Column index of the barycenter.
CHAPTER 4. GRAPHICS

- **PhiIn** (input control) .......................... ellipse.angle.rad \(\sim\) double / VARIANT
  Orientation of the bigger half axis in radians.

- **Radius1In** (input control) .......................... ellipse.radius1 \(\sim\) double / VARIANT
  Bigger half axis.

- **Radius2In** (input control) .......................... ellipse.radius1 \(\sim\) double / VARIANT
  Smaller half axis.

- **Row** (output control) .......................... ellipse.center.y \(\sim\) double / VARIANT
  Row index of the center.

- **Column** (output control) .......................... ellipse.center.x \(\sim\) double / VARIANT
  Column index of the center.

- **Phi** (output control) .......................... ellipse.angle.rad \(\sim\) double / VARIANT
  Orientation of the first half axis in radians.

- **Radius1** (output control) .......................... ellipse.radius1 \(\sim\) double / VARIANT
  First half axis.

- **Radius2** (output control) .......................... ellipse.radius2 \(\sim\) double / VARIANT
  Second half axis.

--- Example ---

```cpp
read_image(Image,\'affe\')
draw_ellipse_mod(WindowHandle, RowIn, ColumnIn, PhiIn, Radius1In, Radius2In, Row, Column, Phi, Radius1, Radius2)
gen_ellipse(Ellipse, Row, Column, Phi, Radius1, Radius2)
reduce_domain(Image, Ellipse, GrayEllipse)
sobel_amp(GrayEllipse, Sobel, \'sum_abs\', 3)
disp_image(Sobel, WindowHandle).
```

--- Result ---

**DrawEllipseMod** returns TRUE, if the window is valid and the needed drawing mode (see **SetInsert**) is available. If necessary, an exception handling is raised.

--- Parallelization Information ---

**DrawEllipseMod** is reentrant, local, and processed without parallelization.

--- Possible Predecessors ---

OpenWindow

--- Possible Successors ---

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

--- See also ---

GenEllipse, DrawRectangle1, DrawRectangle2, DrawPolygon, SetInsert

--- Alternatives ---

DrawEllipse, DrawCircle, DrawRegion

--- Module ---

**System**

```cpp
[out] double Row1 HWindowX.DrawLine ([out] double Column1,
[out] double Row2, [out] double Column2 )
```

**DrawLine** returns the parameter for a line, which has been created interactively by the user in the window.

To create a line you have to press the left mouse button determining a start point of the line. While keeping the button pressed you may “drag” the line in any direction. After another mouse click in the middle of the created line you can move it. If you click on one end point of the created line, you may move this point. Pressing the right mouse button terminates the procedure.

HALCON/COM Reference Manual, 2005-2-1
After terminating the procedure the line is not visible in the window any longer.

--- Parameter ---

- **WindowHandle** (input control) .......................... window $\sim$ HWindowX / VARIANT Window.id.
- **Row1** (output control) .......................... line.begin.y $\sim$ double / VARIANT Row index of the first point of the line.
- **Column1** (output control) .......................... line.begin.x $\sim$ double / VARIANT Column index of the first point of the line.
- **Row2** (output control) .......................... line.end.y $\sim$ double / VARIANT Row index of the second point of the line.
- **Column2** (output control) .......................... line.end.x $\sim$ double / VARIANT Column index of the second point of the line.

--- Example ---

```c
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_line(WindowHandle,Row1,Column1,Row2,Column2)
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fwrite_string(['(',Row2,',',Column2,')'])
fnew_line().
```

--- Result ---

**DrawLine** returns TRUE, if the window is valid and the needed drawing mode (see **SetInsert**) is available. If necessary, an exception handling is raised.

--- Parallelization Information ---

**DrawLine** is reentrant, local, and processed without parallelization.

--- Possible Predecessors ---

**OpenWindow**

--- Possible Successors ---

**ReduceDomain,DispLine,SetColored,SetLineWidth,SetDraw,SetInsert**

--- See also ---

**DrawLineMod,GenRectangle1,DrawCircle,DrawEllipse,SetInsert**

--- Module ---

```c
[out] double Row1 HWindowX.DrawLineMod ([in] double Row1In,
[in] double Column1In, [in] double Row2In, [in] double Column2In,
[out] double Column1, [out] double Row2, [out] double Column2 )
```

**Draw a line.**

**DrawLineMod** returns the parameter for a line, which has been created interactively by the user in the window.

To create a line are expected the coordinates of the start point **Row1In, Column1In** and of the end point **Row2In,Column2In**. If you click on one end point of the created line, you may move this point. After another mouse click in the middle of the created line you can move it.
Pressing the right mousebutton terminates the procedure.

After terminating the procedure the line is not visible in the window any longer.

--- Parameter ---

- **WindowHandle** (input control)  
  Parameter: `window`  
  Type: `HWWindowX / VARIANT`  
  Description: Window id.

- **Row1In** (input control)  
  Parameter: `line.begin.y`  
  Type: `double / VARIANT`  
  Description: Row index of the first point of the line.

- **Column1In** (input control)  
  Parameter: `line.begin.x`  
  Type: `double / VARIANT`  
  Description: Column index of the first point of the line.

- **Row2In** (input control)  
  Parameter: `line.end.y`  
  Type: `double / VARIANT`  
  Description: Row index of the second point of the line.

- **Column2In** (input control)  
  Parameter: `line.end.x`  
  Type: `double / VARIANT`  
  Description: Column index of the second point of the line.

- **Row1** (output control)  
  Parameter: `line.begin.y`  
  Type: `double / VARIANT`  
  Description: Row index of the first point of the line.

- **Column1** (output control)  
  Parameter: `line.begin.x`  
  Type: `double / VARIANT`  
  Description: Column index of the first point of the line.

- **Row2** (output control)  
  Parameter: `line.end.y`  
  Type: `double / VARIANT`  
  Description: Row index of the second point of the line.

- **Column2** (output control)  
  Parameter: `line.end.x`  
  Type: `double / VARIANT`  
  Description: Column index of the second point of the line.

--- Example ---

```plaintext
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_line_mod(WindowHandle,10,20,55,124,Row1,Column1,Row2,Column2)
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fwrite_string(['(',Row2,',',Column2,')'])
fnew_line().
```

--- Result ---

`DrawLineMod` returns TRUE, if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

--- Parallelization Information ---

`DrawLineMod` is reentrant, local, and processed without parallelization.

--- Possible Predecessors ---

- `OpenWindow`

--- Possible Successors ---

- `ReduceDomain`, `DispLine`, `SetColored`, `SetLineWidth`, `SetDraw`, `SetInsert`

--- See also ---

- `GenCircle`, `DrawRectangle1`, `DrawRectangle2`

--- Alternatives ---

- `DrawLine`, `DrawEllipse`, `DrawRegion`

--- Module ---

System
Draw a point.

**DrawPoint** returns the parameter for a point, which has been created interactively by the user in the window.

To create a point you have to press the left mouse button. While keeping the button pressed you may “drag” the point in any direction. Pressing the right mouse button terminates the procedure.

After terminating the procedure the point is not visible in the window any longer.

### Parameter

- **WindowHandle** (input control) ................................. window  \(\sim\) HWindow / VARIANT WindowId.
- **Row** (output control) ................................. point.y  \(\sim\) double / VARIANT Row index of the point.
- **Column** (output control) ................................. point.x  \(\sim\) double / VARIANT Column index of the point.

### Example

```c
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_point(WindowHandle,Row1,Column1)
disp_line(WindowHandle,Row1-2,Column1,Row1+2,Column1)
disp_line(WindowHandle,Row1,Column1-2,Row1,Column1+2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fnew_line().
```

### Result

**DrawPoint** returns TRUE, if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

### Parallelization Information

**DrawPoint** is reentrant, local, and processed without parallelization.

### Possible Predecessors

OpenWindow

### Possible Successors

ReduceDomain, DispLine, SetColored, SetLineWidth, SetDraw, SetInsert

### See also

DrawPointMod, DrawCircle, DrawEllipse, SetInsert

### Module

System

---

```c
[out] double Row HWindowX.DrawLine ([out] double Column )

```

Draw a point.

**DrawPoint** returns the parameter for a point, which has been created interactively by the user in the window.
To create a point are expected the coordinates \texttt{RowIn} and \texttt{ColumnIn}. While keeping the button pressed you may “drag” the point in any direction. Pressing the right mouse button terminates the procedure.

After terminating the procedure the point is not visible in the window any longer.

---

**Parameter**

- \texttt{WindowHandle} (input control) \texttt{window} \sim \texttt{HWindowX / VARIANT Window_id}
- \texttt{RowIn} (input control) \texttt{point.y} \sim \texttt{double / VARIANT Row index of the point.}
- \texttt{ColumnIn} (input control) \texttt{point.x} \sim \texttt{double / VARIANT Column index of the point.}
- \texttt{Row} (output control) \texttt{point.y} \sim \texttt{double / VARIANT Row index of the point.}
- \texttt{Column} (output control) \texttt{point.x} \sim \texttt{double / VARIANT Column index of the point.}

---

**Example**

```scheme
get_system('width',Width)
gct_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,’affe’)
disp_image(Image,WindowHandle)
draw_point_mod(WindowHandle,Row1,Column1)
disp_line(WindowHandle,Row1-2,Column1,Row1+2,Column1)
disp_line(WindowHandle,Row1,Column1-2,Row1,Column1+2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fnew_line().
```

---

**Result**

\texttt{DrawPointMod} returns TRUE, if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

---

**Parallelization Information**

\texttt{DrawPointMod} is \texttt{reentrant}, \texttt{local}, and processed \texttt{without} parallelization.

---

**Possible Predecessors**

\texttt{OpenWindow}

---

**Possible Successors**

\texttt{ReduceDomain,DispLine,SetColored,SetLineWidth,SetDraw,SetInsert}

---

**See also**

\texttt{DrawPoint,DrawCircle,DrawEllipse,SetInsert}

---

**Module**

\texttt{System}

---

```scheme
[out] HRegionX PolygonRegion \texttt{HWindowX.DrawPolygon ( )}

void \texttt{HRegionX.DrawPolygon ([in] HWindowX WindowHandle )}

void \texttt{HOperatorSetX.DrawPolygon ([out] HUntypedObjectX PolygonRegion, [in] VARIANT WindowHandle )}
```

---

**Interactive drawing of a polygon row.**

\texttt{DrawPolygon} produces an image. The region of that image spans exactly the imagepoints entered interactively by mouse clicks (gray values remain undefined).

Painting in the output window happens while pressing the left mouse button. Releasing the left mouse button and repressing it at another position effects drawing a line between these two points. Pressing the right mouse button terminates the input. Painting uses that color which has been set by \texttt{SetColor, SetRgb}, etc.
To put gray values on the created PolygonRegion for further processing, you may use the procedure ReduceDomain.

Attention

The painted contour is not closed automatically, in particular it is not “filled up” either.
Output object’s gray values are not defined.

Parameter

PolygonRegion (output iconic) ...................... region \( \sim \) HRegionX / HUntypedObjectX Region, which encompasses all painted points.

WindowHandle (input control) ....................... window \( \sim \) HWindowX / VARIANT Window_id.

Example

draw_polygon(Polygon,WindowHandle)
convex(Polygon,Filled)
disp_region(Filled,WindowHandle).

Result

If the window is valid, DrawPolygon returns TRUE. If necessary, an exception handling is raised.

Parallelization Information

DrawPolygon is reentrant, local, and processed without parallelization.

OpenWindow

Possible Predecessors

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw

See also

ReduceDomain, FillUp, SetColor

Alternatives

DrawRegion, DrawCircle, DrawRectangle1, DrawRectangle2, Boundary

Module

System

\begin{verbatim}
[out] double Row1  HWindowX.DrawRectangle1  ( [out] double Column1,
  [out] double Row2,  [out] double Column2 )

void HOperatorSetX.DrawRectangle1 ( [in] VARIANT WindowHandle,
  [out] VARIANT Row1,  [out] VARIANT Column1,  [out] VARIANT Row2,
  [out] VARIANT Column2 )
\end{verbatim}

Draw a rectangle parallel to the coordinate axis.

DrawRectangle1 returns the parameter for a rectangle parallel to the coordinate axes, which has been created interactively by the user in the window.

To create a rectangle you have to press the left mouse button determining a corner of the rectangle. While keeping the button pressed you may “drag” the rectangle in any direction. After another mouse click in the middle of the created rectangle you can move it. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction to this side. If you click on one corner of the created rectangle, you may move this corner. Pressing the right mousebutton terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.

Parameter

WindowHandle (input control) ....................... window \( \sim \) HWindowX / VARIANT Window_id.

Row1 (output control) ......................... rectangle.origin.y \( \sim \) double / VARIANT Row index of the left upper corner.
DRAW_RECTANGLE1

- **Column1** (output control) \( \text{rectangle.origin.x} \) ~ double / VARIANT
  Column index of the left upper corner.
- **Row2** (output control) \( \text{rectangle.corner.y} \) ~ double / VARIANT
  Row index of the right lower corner.
- **Column2** (output control) \( \text{rectangle.corner.x} \) ~ double / VARIANT
  Column index of the right lower corner.

**Example**

```plaintext
get_system('width', Width)
get_system('height', Height)
set_part(WindowHandle, 0, 0, Width-1, Height-1)
read_image(Image, 'affe')
disp_image(Image, WindowHandle)
draw_rectangle1(WindowHandle, Row1, Column1, Row2, Column2)
disp_image(Image, WindowHandle)
fwrite_string(['Clipping = (', Row1, ',', Column1, ')'])
fwrite_string(['(', Row2, ',', Column2, ')'])
fnew_line()
```

**Result**

*DrawRectangle1* returns `TRUE` if the window is valid and the needed drawing mode (see *SetInsert*) is available. If necessary, an exception handling is raised.

**Parallelization Information**

*DrawRectangle1* is reentrant, local, and processed without parallelization.

**Possible Predecessors**

OpenWindow

**Possible Successors**

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

**See also**

GenRectangle1, DrawCircle, DrawEllipse, SetInsert

**Alternatives**

DrawRectangle1Mod, DrawRectangle2, DrawRegion

**Module**

System

```plaintext
[out] double Row1
HWINDOWX.DrawRectangle1Mod ([in] double Row1In,
[in] double Column1In, [in] double Row2In, [in] double Column2In,
[out] double Column1, [out] double Row2)
```

```plaintext
void HOPERATORSETX.DrawRectangle1Mod ([in] VARIANT WindowHandle,
[in] VARIANT Row1In, [in] VARIANT Column1In, [in] VARIANT Row2In,
[in] VARIANT Column2In, [out] VARIANT Row1, [out] VARIANT Column1,
[out] VARIANT Row2, [out] VARIANT Column2)
```

**Draw a rectangle parallel to the coordinate axis.**

*DrawRectangle1Mod* returns the parameter for a rectangle parallel to the coordinate axes, which has been created interactively by the user in the window.

To create a rectangle are expected the parameters `Row1In`, `Column1In.Row2In` und `Column2In`. After a mouse click in the middle of the created rectangle you can move it. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction to this side. If you click on one corner of the created rectangle, you may move this corner. Pressing the right mousebutton terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.
4.1. DRAWING

Parameter

- **WindowHandle** (input control) .......................... window \( \sim \) HWindowX / VARIANT
  Window \_id.
- **Row1In** (input control) ............................. rectangle.origin.y \( \sim \) double / VARIANT
  Row index of the left upper corner.
- **Column1In** (input control) ............................ rectangle.origin.x \( \sim \) double / VARIANT
  Column index of the left upper corner.
- **Row2In** (input control) ............................. rectangle.corner.y \( \sim \) double / VARIANT
  Row index of the right lower corner.
- **Column2In** (input control) ............................ rectangle.corner.x \( \sim \) double / VARIANT
  Column index of the right lower corner.
- **Row1** (output control) .............................. rectangle.origin.y \( \sim \) double / VARIANT
  Row index of the left upper corner.
- **Column1** (output control) ............................. rectangle.origin.x \( \sim \) double / VARIANT
  Column index of the left upper corner.
- **Row2** (output control) .............................. rectangle.corner.y \( \sim \) double / VARIANT
  Row index of the right lower corner.
- **Column2** (output control) ............................. rectangle.corner.x \( \sim \) double / VARIANT
  Column index of the right lower corner.

Example

```plaintext
get_system('width',Width)
get_system('height',Height)
set_part(WindowHandle,0,0,Width-1,Height-1)
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_rectangle1_mod(WindowHandle,Row1In,Column1In,Row2In,Column2In,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string([Clipping = ('',Row1,'','',Column1,'')])
fwrite_string([',(',Row2,',',Column2,')'])
fnew_line().
```

Result

**DrawRectangle1Mod** returns TRUE, if the window is valid and the needed drawing mode (see **SetInsert**) is available. If necessary, an exception handling is raised.

Parallelization Information

**DrawRectangle1Mod** is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow

Possible Successors

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert

See also

GenRectangle1, DrawCircle, DrawEllipse, SetInsert

Alternatives

DrawRectangle1, DrawRectangle2, DrawRegion

Module

System
Interactive drawing of any orientated rectangle.

\texttt{DrawRectangle2} returns the parameter for any orientated rectangle, which has been created interactively by the user in the window.

The created rectangle is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create a rectangle you have to press the left mouse button for the center of the rectangle. While keeping the button pressed you may dimension the length (\texttt{Length1}) and the orientation (\texttt{Phi}) of the first half axis. In doing so a temporary default length for the second half axis is assumed, which may be modified afterwards on demand. After another mouse click in the middle of the created rectangle, you can move it. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction to this side. You only can modify the orientation, if you grip a side perpendicular to the first half axis. Pressing the right mouse button terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.
Interactive drawing of any orientated rectangle.

The created rectangle is described by its center, its two half axes and the angle between the first half axis and the horizontal coordinate axis.

To create a rectangle are expected the parameters $RowIn$, $ColumnIn$, $PhiIn$, $Length1In$, $Length2In$. A click close to one side “grips” it to modify the rectangle’s dimension in perpendicular direction ($Length2$) to this side. You only can modify the orientation ($Phi$), if you grip a side perpendicular to the first half axis. After another mouse click in the middle of the created rectangle, you can move it. Pressing the right mouse button terminates the procedure.

After terminating the procedure the rectangle is not visible in the window any longer.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WindowHandle</td>
<td>Window id.</td>
</tr>
<tr>
<td>$RowIn$</td>
<td>Row index of the barycenter.</td>
</tr>
<tr>
<td>$ColumnIn$</td>
<td>Column index of the barycenter.</td>
</tr>
<tr>
<td>$PhiIn$</td>
<td>Orientation of the bigger half axis in radians.</td>
</tr>
<tr>
<td>$Length1In$</td>
<td>Bigger half axis.</td>
</tr>
<tr>
<td>$Length2In$</td>
<td>Smaller half axis.</td>
</tr>
<tr>
<td>$Row$</td>
<td>Row index of the barycenter.</td>
</tr>
<tr>
<td>$Column$</td>
<td>Column index of the barycenter.</td>
</tr>
<tr>
<td>$Phi$</td>
<td>Orientation of the bigger half axis in radians.</td>
</tr>
<tr>
<td>$Length1$</td>
<td>Bigger half axis.</td>
</tr>
<tr>
<td>$Length2$</td>
<td>Smaller half axis.</td>
</tr>
</tbody>
</table>

Result

$DrawRectangle2Mod$ returns TRUE, if the window is valid and the needed drawing mode (see $SetInsert$) is available. If necessary, an exception handling is raised.

Parallelization Information

$DrawRectangle2Mod$ is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow

Possible Successors

ReduceDomain, DispRegion, SetColored, SetLineWidth, SetDraw, SetInsert
Interactive drawing of a closed region.  

**DrawRegion** produces an image. The region of that image spans exactly the image region entered interactively by mouse clicks (gray values remain undefined). Painting happens in the output window while keeping the left mouse button pressed. The left mouse button even operates by clicking in the output window; through this a line between the previous clicked points is drawn. Clicking the right mouse button terminates input and closes the outline. Subsequently the image is "filled up". Also it contains the whole image area enclosed by the mouse.

Painting uses that color which has been set by **SetColor**, **SetRgb**, etc.

Attention  
The output object’s gray values are not defined.

Parameter  
- **Region** (output iconic)  
  Interactive created region.
- **WindowHandle** (input control)  
  Window id.

Example  
read_image(Image,’fabrik’)
disp_image(Image,WindowHandle)
draw_region(Region,WindowHandle)
reduce_domain(Image,Region,New)
regiongrowing(New,Segmente,5,5,6,50)
set_colored(WindowHandle,12)
disp_region(Segmente,WindowHandle).

Result  
If the window is valid, **DrawRegion** returns TRUE. If necessary, an exception handling is raised.

Parallelization Information  
**DrawRegion** is reentrant, local, and processed without parallelization.

Possible Predecessors  
**OpenWindow**

Possible Successors  
**ReduceDomain**, **DispRegion**, **SetColored**, **SetLineWidth**, **SetDraw**

See also  
**DrawPolygon**, **ReduceDomain**, **FillUp**, **SetColor**

Alternatives  
**DrawCircle**, **DrawEllipse**, **DrawRectangle1**, **DrawRectangle2**

Module  
**System**
## 4.2 Gnuplot

### `void HOperatorSetX.GnuplotClose ([in] VARIANT GnuplotFileID)`

Close all open gnuplot files or terminate an active gnuplot sub-process.

`GnuplotClose` closes all gnuplot files opened by `GnuplotOpenFile` or terminates the gnuplot sub-process created with `GnuplotOpenPipe`. In the latter case, all temporary files used to display images and control values are deleted. This means that `GnuplotClose` must be called after such a plot sequence. `GnuplotFileID` is the identifier of the corresponding gnuplot output stream.

- **Parameter**
  - `GnuplotFileID` (input control) \[\ldots\] \[\ldots\] \[\ldots\] `gnuplot_id ~ HOperatorSetX / VARIANT` Identifier for the gnuplot output stream.

- **Result**
  - `GnuplotClose` returns TRUE if `GnuplotFileID` is a valid gnuplot output stream. Otherwise, an exception handling is raised.

### `void HGnuplotX.GnuplotOpenFile ([in] String FileName)`

Open a gnuplot file for visualization of images and control values.

`GnuplotOpenFile` allows the output of images and control values in a format which can be later processed by gnuplot. The parameter `FileName` determines the base-name of the files to be created by calls to `GnuplotPlotImage`. `GnuplotOpenFile` generates a gnuplot control file with the name `<FileName>.gp`, in which the respective plot commands are written. Each image plotted by `GnuplotPlotImage` (or control values plotted by `GnuplotPlotCtrl`) creates a data file with the name `<FileName>.<Number>.dat`, where Number is the number of the plot in the current sequence. The generated control file can later be edited to create the desired effect. After the last plot `GnuplotClose` has to be called in order to close all open files. The corresponding identifier for the gnuplot output stream is returned in `GnuplotFileID`.

- **Parameter**
  - `FileName` (input control) \[\ldots\] \[\ldots\] `filename ~ String / VARIANT` Base name for control and data files.
  - `GnuplotFileID` (output control) \[\ldots\] \[\ldots\] `gnuplot_id ~ HGnuplotX / VARIANT` Identifier for the gnuplot output stream.

- **Result**
  - `GnuplotOpenFile` returns the value TRUE if the control file could be opened. Otherwise, an exception handling is raised.

### Parallelization Information

- `GnuplotClose` is processed completely exclusively without parallelization.

### Possible Predecessors

- `GnuplotOpenPipe`, `GnuplotOpenFile`, `GnuplotPlotImage`

### Possible Successors

- `GnuplotOpenPipe`, `GnuplotOpenFile`, `GnuplotPlotImage`

### Module

`System`
CHAPTER 4. GRAPHICS

See also
GnuplotOpenPipe, GnuplotClose, GnuplotPlotImage

Alternatives
GnuplotOpenPipe

Module
System

void HGnuplotX.GnuplotOpenPipe ( )
void HOperatorSetX.GnuplotOpenPipe ([out] VARIANT GnuplotFileID )

Open a pipe to a gnuplot process for visualization of images and control values.

GnuplotOpenPipe opens a pipe to a gnuplot sub-process with which subsequently images can be visualized as 3D-plots (GnuplotPlotImage) or control values can be visualized as 2D-plots (GnuplotPlotCtrl). The sub-process must be terminated after displaying the last plot by calling GnuplotClose. The corresponding identifier for the gnuplot output stream is returned in GnuplotFileID.

Attention
GnuplotOpenPipe is only implemented for Unix because gnuplot for Windows (wgnuplot) cannot be controlled by an external process.

Parameter

\[ \text{GnuplotFileID} \] (output control) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \text{gnuplot_id} \sim HGnuplotX / \text{VARIANT}
Identifier for the gnuplot output stream.

Result

GnuplotOpenPipe returns the value TRUE if the sub-process could be created. Otherwise, an exception handling is raised.

Parallelization Information
GnuplotOpenPipe is processed completely exclusively without parallelization.

Possible Successors
GnuplotPlotImage, GnuplotPlotCtrl, GnuplotClose

Alternatives
GnuplotOpenFile

Module
System

void HGnuplotX.GnuplotPlotCtrl ([in] VARIANT Values )

Plot control values using gnuplot.

GnuplotPlotCtrl displays a tuple of control values using gnuplot. If there is an active gnuplot sub-process (started with GnuplotOpenPipe), the image is displayed in a gnuplot window. Otherwise, the image is output to a file, which can be later read by gnuplot. In both cases the gnuplot output stream is identified by GnuplotFileID.

Parameter

\[ \text{GnuplotFileID} \] (input control) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \text{gnuplot_id} \sim HGnuplotX / \text{VARIANT}
Identifier for the gnuplot output stream.

\[ \text{Values} \] (input control) \ldots \ldots \ldots \ldots \ldots \ldots \ldots \text{number} \sim \text{VARIANT ( real, integer )}
Control values to be plotted (y-values).

Result

GnuplotPlotCtrl returns the value if GnuplotFileID is a valid gnuplot output stream, if the data file for
the current plot could be opened, and if only integer or floating point values were passed. Otherwise, an exception handling is raised.

--- Parallelization Information ---

GnuplotPlotCtrl is processed *completely exclusively* without parallelization.

--- Possible Predecessors ---

GnuplotOpenPipe, GnuplotOpenFile

--- Possible Successors ---

GnuplotClose

--- See also ---

GnuplotOpenPipe, GnuplotOpenFile, GnuplotClose

--- Module ---

System

--- Parameter ---

d > GnuplotFileID (input control) .................... gnuplot_id ~ HGnuplotX / VARIANT Identifier for the gnuplot output stream.

d > Function (input control) ....................... function_id ~ HFunction1dX / VARIANT (real, integer) Function to be plotted.

--- Result ---

GnuplotPlotCtrl returns TRUE if GnuplotFileID is a valid gnuplot output stream, and if the data file for the current plot could be opened. Otherwise, an exception handling is raised.

--- Parallelization Information ---

GnuplotPlotFunct1D is processed *completely exclusively* without parallelization.

--- Possible Predecessors ---

GnuplotOpenPipe, GnuplotOpenFile

--- Possible Successors ---

GnuplotClose

--- See also ---

GnuplotOpenPipe, GnuplotOpenFile, GnuplotClose

--- Alternatives ---

GnuplotPlotCtrl

--- Module ---

System

--- HALCON 6.1.4 ---
Visualize images using gnuplot.

`GnuplotPlotImage` displays an image as a 3D-plot using gnuplot. If there is an active gnuplot sub-process (started with `GnuplotOpenPipe`), the image is displayed in a gnuplot window. Otherwise, the image is output to a file, which can be later read by gnuplot. In both cases the gnuplot output stream is identified by `GnuplotFileID`. The parameters `SamplesX` and `SamplesY` determine the number of data points in the x- and y-direction, respectively, which gnuplot should use to display the image. They are the equivalent of the gnuplot variables samples and isosamples. The parameters `ViewRotX` and `ViewRotZ` determine the rotation of the plot with respect to the viewer. `ViewRotX` is the rotation of the coordinate system about the x-axis, while `ViewRotZ` is the rotation of the plot about the z-axis. These two parameters correspond directly to the first two parameters of the `set view` command in gnuplot. The parameter `Hidden3D` determines whether hidden surfaces should be removed. This is equivalent to the `set hidden3d` command in gnuplot. If a single image is passed to the operator, it is displayed in a separate plot. If multiple images are passed, they are displayed in the same plot.

Parameter

- **Image** (input iconic) …… image \( \sim HImageX / IHObjectX \) (byte, direction, cyclic, int1, int2, uint2, int4, real)
  - Image to be plotted.
- **GnuplotFileID** (input control) ……………. gnuplot_id \( \sim long / HGnuplotX / VARIANT \)
  - Identifier for the gnuplot output stream.
- **SamplesX** (input control) ………………………………… integer \( \sim long / VARIANT \)
  - Number of samples in the x-direction.
    - Default Value: 64
    - Typical range of values: \( 2 \leq \text{SamplesX} \leq 2 \)
    - Restriction: \( (\text{SamplesX} \geq 2) \)
- **SamplesY** (input control) ………………………………… integer \( \sim long / VARIANT \)
  - Number of samples in the y-direction.
    - Default Value: 64
    - Typical range of values: \( 2 \leq \text{SamplesY} \leq 2 \)
    - Restriction: \( (\text{SamplesY} \geq 2) \)
- **ViewRotX** (input control) ………………………………… number \( \sim \text{VARIANT} \) (integer, real)
  - Rotation of the plot about the x-axis.
    - Default Value: 60
    - Typical range of values: \( 0 \leq \text{ViewRotX} \leq 0 \)
    - Minimum Increment: 0.01
    - Recommended Increment: 10
    - Restriction: \( (0 \leq \text{ViewRotX}) \land (\text{ViewRotX} \leq 180) \)
- **ViewRotZ** (input control) ………………………………… number \( \sim \text{VARIANT} \) (integer, real)
  - Rotation of the plot about the z-axis.
    - Default Value: 30
    - Typical range of values: \( 0 \leq \text{ViewRotZ} \leq 0 \)
    - Minimum Increment: 0.01
    - Recommended Increment: 10
    - Restriction: \( (0 \leq \text{ViewRotZ}) \land (\text{ViewRotZ} \leq 360) \)
- **Hidden3D** (input control) ………………………………… string \( \sim \text{String} / \text{VARIANT} \)
  - Plot the image with hidden surfaces removed.
    - Default Value: 'hidden3d'
    - List of values: Hidden3D ∈ { 'hidden3d', 'nohidden3d' }
**4.3 LUT**

**GnuplotPlotImage** returns the value if *GnuplotFileID* is a valid gnuplot output stream, and if the data file for the current plot could be opened. Otherwise, an exception handling is raised.

**Parallelization Information**

**GnuplotPlotImage** is processed completely exclusively without parallelization.

**Possible Predecessors**

GnuplotOpenPipe, GnuplotOpenFile

**Possible Successors**

GnuplotClose

**See also**

GnuplotOpenPipe, GnuplotOpenFile, GnuplotClose

**Module**

System

---

**4.3 LUT**

| void **HWindowX.DispLut** ([in] long Row, [in] long Column, [in] long Scale ) |

**Graphical view of the look-up-table (lut).**

DispLut displays a graphical view of the look-up-table (lut) in the valid window. A look-up-table defines the transformation of image gray values to colors/gray levels on the screen. On most systems this can be modified.

DispLut creates a graphical view of the table assigned to the output window with the logical window number *WindowHandle* and displays it for every basic color (red, green, blue). *Row* and *Column* define the position of the centre of the graphic. *Scale* allows scaling of the graphic, whereas 1 means displaying all 256 values, 2 means displaying 128 values, 3 means displaying only 64 values, etc. Tables for monochrome-representations are displayed in the currently set color (see **SetColor**, **SetRgb**, etc.). Tables for displaying "false colors" are viewed with red, green and blue for each color component.

**Attention**

**DispLut** can only be used on hardware supporting look-up-tables for the output.

**Parameter**

- **WindowHandle** (input control) .......................... window ~ **HWindowX** / VARIANT Window identifier.
- **Row** (input control) ........................................... point.y ~ long / VARIANT Row of centre of the graphic.
  - Default Value : 128
  - Typical range of values : 0 ≤ Row ≤ 0
- **Column** (input control) ......................................... point.x ~ long / VARIANT Column of centre of the graphic.
  - Default Value : 128
  - Typical range of values : 0 ≤ Column ≤ 0
- **Scale** (input control) ..............................integer ~ long / VARIANT Scaling of the graphic.
  - Default Value : 1
  - List of values : Scale ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  - Typical range of values : 0 ≤ Scale ≤ 0

**Example**

```
set_lut(WindowHandle, 'color')
disp_lut(WindowHandle, 256, 256, 1)
get_mbutton(WindowHandle, _, _, _)
set_lut(WindowHandle, 'sqrt')
disp_lut(WindowHandle, 128, 128, 2).
```

HALCON 6.1.4
**DispLut** returns TRUE if the hardware supports a look-up-table, the window is valid and the parameters are correct. Otherwise an exception handling is raised.

---

**Parallelization Information**

**DispLut** is reentrant, local, and processed without parallelization.

---

**Possible Predecessors**

**SetLut**

---

**Possible Successors**

**SetLutStyle**, **SetLut**, **WriteLut**, **DispLut**

---

**See also**

**WriteLut**, **SetLut**, **GetLut**, **DispLut**

---

**Alternatives**

**SetFix**, **SetRgb**

---

**DispLut** returns TRUE if the window is valid. Otherwise an exception handling is raised.

---

**Parallelization Information**

**DispLut** is reentrant, local, and processed without parallelization.

---

**Possible Successors**

**SetLutStyle**, **SetLut**, **WriteLut**, **DispLut**

---

**See also**

**WriteLut**, **SetLut**, **GetLut**, **DispLut**

---

**Alternatives**

**SetFix**, **SetRgb**

---

Manipulate look-up-table (lut) interactively.

**DispLut** allows interactive manipulation of the look-up-table of the device currently displaying the output window.

By pressing and holding down the left mouse button one can change (from "left to right") the red-, green- and blue-intensity displayed in a 2 dimensional diagram with the gray values on the x-axis. The left mouse button also is used for choosing the color channel that should be changed. As an alternative, one can map pure gray levels (gray "color channel") to the gray values on the x-axis. The right mouse button is used for terminating the change-process. The modified look-up-table can be saved by **WriteLut** and reloaded later by **SetLut**. **GetLut** succeeding **DispLut** returns directly the RGB tuple of the look-up-table. These are suitable as input of **SetLut**.

---

**Attention**

**DispLut** can only be used on hardware supporting look-up-tables for the output and allow dynamic changing of the tables.

---

**Parameter**

> **WindowHandle** (input control) ................................. window 〜**HW**indowX / **V**ARIANT **WindowHandle**

---

**Example**

```c
void HWWindowX.DrawLut ( )

void HOperatorSetX.DrawLut ([in] VARIANT WindowHandle )
```

```c
Manipulate look-up-table (lut) interactively.

**DispLut** allows interactive manipulation of the look-up-table of the device currently displaying the output window.

By pressing and holding down the left mouse button one can change (from "left to right") the red-, green- and blue-intensity displayed in a 2 dimensional diagram with the gray values on the x-axis. The left mouse button also is used for choosing the color channel that should be changed. As an alternative, one can map pure gray levels (gray "color channel") to the gray values on the x-axis. The right mouse button is used for terminating the change-process. The modified look-up-table can be saved by **WriteLut** and reloaded later by **SetLut**. **GetLut** succeeding **DispLut** returns directly the RGB tuple of the look-up-table. These are suitable as input of **SetLut**.

---

**Attention**

**DispLut** can only be used on hardware supporting look-up-tables for the output and allow dynamic changing of the tables.

---

**Parameter**

> **WindowHandle** (input control) ................................. window 〜**HW**indowX / **V**ARIANT **WindowHandle**

---

**Example**

```c
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
draw_lut(WindowHandle)
write_lut(WindowHandle,'my_lut').
...
read_image(Image,'fabrik')
set_lut(WindowHandle,'my_lut').
```

---

**Result**

**DispLut** returns TRUE if the window is valid. Otherwise an exception handling is raised.

---

**Parallelization Information**

**DispLut** is reentrant, local, and processed without parallelization.

---

**Possible Successors**

**SetLutStyle**, **SetLut**, **WriteLut**, **DispLut**

---

**See also**

**WriteLut**, **SetLut**, **GetLut**, **DispLut**

---

**Alternatives**

**SetFix**, **SetRgb**

---

HALCON/COM Reference Manual, 2005-2-1
4.3. LUT

Get fixing of "'look-up-table'" (lut) for "'real color images'"

### Parameter

- **WindowHandle** (input control)  
  - window \(\sim\) *HWindowX / VARIANT*
  - Window identifier.

- **Mode** (output control)  
  - string \(\sim\) *String / VARIANT*
  - Mode of fixing.
  - **Default Value**: 'true'
  - **List of values**: Mode \(\in\) {'true', 'false'}

### Parallelization Information

GetFixedLut is reentrant, local, and processed without parallelization.

### Possible Successors

SetFixedLut

### System

Get current look-up-table (lut).

GetLut returns the name or the values of the look-up-table (lut) of the window, currently used by DispImage (or indirectly by DispRegion, etc.) for output. To set a look-up-table use SetLut. If the current table is a system table without any modification (by SetFix), the name of the table is returned. If it is a modified table, a table read from a file or a table for output with pseudo real colors, the RGB-values of the table are returned.

### Attention

### Parameter

- **WindowHandle** (input control)  
  - window \(\sim\) *HWindowX / VARIANT*
  - Window identifier.

- **LookUpTable** (output control)  
  - string \(\sim\) *VARIANT (integer, string )*
  - Name of look-up-table or tuple of RGB-values.

### Result

GetLut returns TRUE if the window is valid. Otherwise an exception handling is raised.

### Parallelization Information

GetLut is reentrant, local, and processed without parallelization.

### Possible Successors

DrawLut, SetLut

### See also

SetLut, DrawLut

### Alternatives

SetFix, GetPixel

### Module
Get modification parameters of look-up-table (lut).

GetLutStyle returns the values that were set with SetLutStyle. Default is:

- **Hue**: 0.0
- **Saturation**: 1.0
- **Intensity**: 1.0

Parameter:

- **WindowHandle** (input control)

Result:

GetLutStyle returns TRUE if the window is valid and the parameter is correct. Otherwise an exception handling is raised.

Parallelization Information:

GetLutStyle is reentrant, local, and processed without parallelization.

Possible Successors:

System

Query all available look-up-tables (lut).

QueryLut returns the names of all look-up-tables available on the current used device. These tables can be set with SetLut. An table named 'default' is always available.

Parameter:

- **WindowHandle** (input control)
- **LookUpTable** (output control)

Result:

QueryLut returns TRUE if a window is valid. Otherwise an exception handling is raised.

Parallelization Information:

QueryLut is reentrant, local, and processed without parallelization.
4.3. LUT

Possible Successors
SetLutStyle, SetLut, WriteLut, Displut

See also
SetLut

Module
System

void HWindowX.SetFixedLut ([in] String Mode )

void HOperatorSetX.SetFixedLut ([in] VARIANT WindowHandle, [in] VARIANT Mode )

Fix "'look-up-table'" (lut) for "'real color images'".

Parameter

▷ WindowHandle (input control) .......................... window  ~ HWindowX / VARIANT Window identifier.
▷ Mode (input control) .................................................. string  ~ String / VARIANT Mode of fixing.
   Default Value : 'true'
   List of values : Mode ∈ {'true', 'false'}

Parallelization Information
SetFixedLut is reentrant, local, and processed without parallelization.

Possible Predecessors

GetFixedLut

Module
System

void HWindowX.SetLut ([in] VARIANT LookUpTable )

void HOperatorSetX.SetLut ([in] VARIANT WindowHandle, [in] VARIANT LookUpTable )

Set "'look-up-table'" (lut).

SetLut sets look-up-table of the device (monitor) displaying the output window. A look-up-table defines the transformation of a "'gray value'" within an image into a gray value or color on the screen. It describes the screen gray value/color as a combination of red, green and blue for any image gray value (0..255) (so it is a 'table' to 'look up' the screen gray value/color for each image gray value: look-up-table). Transformation into screen-colors is performed in real-time at every time the screen is displayed new (typically this happens about 60 - 70 times per second). So it is possible to change the look-up-table to get a new look of images or regions. Please remind that not all machines support changing the look-up-table (e.g. monochrome resp. real color).

Look-up-tables within HALCON (and on a machine that supports 256 colors) are disposed into three areas:

S: system area resp. user area,
G: graphic colors,
B: image data.

Colors in S descend from applications that were active before starting HALCON and should not get lost. Graphic colors in G are used for operators such as DispRegion, DispCircle etc. and are set unique within all look-up-tables. A output in a graphic color has always got the same (color-)look, even if different look-up-tables are used. SetColor and SetRgb set graphic colors. Gray values resp. colors in B are used by DispImage to display an image. They can change according to the current look-up-table. There exist two exceptions to this concept:

- SetGray allows setting of colors of the area B for operators such as DispRegion,
- **SetFix** that allows modification of graphic colors.

For common monitors only one look-up-table can be loaded per screen. Whereas **SetLut** can be activated separately for each window. There is the following solution for this problem: It will always be activated the look-up-table that is assigned to the "active window" (a window is set into the state "active" by the window manager).

look-up-table can also be used with truecolor displays. In this case the look-up-table will be simulated in software. This means, that the look-up-table will be used each time an image is displayed.

WindowsNT specific: if the graphiccard is used in mode different from truecolor, you must display the image after setting the look-up-tape.

**QueryLut** lists the names of all look-up-tables. They differ from each other in the area used for gray values. Within this area the following behavior is defined:

gray value tables (1-7 image levels)

'**default**': Only the two basic colors (generally black and white) are used.

color tables (Real color, static gray value steps)

'**default**': Table proposed by the hardware.

gray value tables (256 colors)

'**default**': As 'linear'.

'**linear**': Linear increasing of gray values from 0 (black) to 255 (white).

'**inverse**': Inverse function of 'linear'.

'**sqr**': Gray values increase according to square function.

'**inv_sqr**': Inverse function of 'sqr'.

'**cube**': Gray values increase according to cubic function.

'**inv_cube**': Inverse function of 'cube'.

'**sqrt**': Gray values increase according to square-root function.

'**inv_sqrt**': Inverse Function of 'sqrt'.

'**cubic_root**': Gray values increase according to cubic-root function.

'**inv_cubic_root**': Inverse Function of 'cubic_root'.

color tables (256 colors)

'**color1**': Linear transition from red via green to blue.

'**color2**': Smooth transition from yellow via red, blue to green.

'**color3**': Smooth transition from yellow via red, blue, green, red to blue.

'**color4**': Smooth transition from yellow via red to blue.

'**three**': Displaying the three colors red, green and blue.

'**six**': Displaying the six basic colors yellow, red, magenta, blue, cyan and green.

'**twelve**': Displaying 12 colors.

'**twenty_four**': Displaying 24 colors.

'**rainbow**': Displaying the spectral colors from red via green to blue.

'**temperature**': Temperature table from black via red, yellow to white.

'**change1**': Color changement after every pixel within the table alternating the six basic colors.

'**change2**': Fivefold color changement from green via red to blue.

'**change3**': Threefold color changement from green via red to blue.
A look-up-table can be read from a file. Every line of such a file must contain three numbers in the range of 0 to 255, with the first number describing the amount of red, the second the amount of green and the third the amount of blue of the represented display color. The number of lines can vary. The first line contains information for the first gray value and the last line for the last value. If there are less lines than gray values, the available information values are distributed over the whole interval. If there are more lines than gray values, a number of (uniformly distributed) lines is ignored. The file-name must conform to "LookUpTable.lut". Within the parameter the name is specified without file extension. HALCON will search for the file in the current directory and after that in a specified directory (see SetSystem::'lutDir',<Pfad>::) ). It is also possible to call SetLut with a tuple of RGB-Values. These will be set directly. The number of parameter values must conform to the number of pixels currently used within the look-up-table.

Attention
SetLut can only be used with monitors supporting 256 gray levels/colors.

Parameter

WindowHandle (input control) ......................... window ~ HWindowX / VARIANT
Window identifier.

LookUpTable (input control) ............................string(-array) ~ VARIANT (integer, string)
Name of look-up-table, values of look-up-table (RGB) or file name.

Default Value : 'default'
Suggested values: LookUpTable ∈ {'default', 'linear', 'inverse', 'sqr', 'inv_sqr', 'cube', 'inv_cube', 'sqrt', 'inv_sqrt', 'cubic_root', 'inv_cubic_root', 'color1', 'color2', 'color3', 'color4', 'three', 'six', 'twelve', 'twenty_four', 'rainbow', 'temperature', 'cyclic_gray', 'cyclic_temperature', 'hsi', 'change1', 'change2', 'change3'}

Example

read_image(Image,'affe')
query_lut(WindowHandle,LUTs)
for(1,|LUTs|,
    set_lut(WindowHandle,LUTs[i])
    fwrite_string(['current table ',LUTs[i]])
    fnew_line()
    get_mbutton(WindowHandle,_,_,_)
    loop().

Result
SetLut returns TRUE if the hardware supports a look-up-table and the parameter is correct. Otherwise an exception handling is raised.

Parallelization Information
SetLut is reentrant, local, and processed without parallelization.

Possible Predecessors
QueryLut, DrawLut, GetLut

Possible Successors
WriteLut

See also
GetLut, QueryLut, DrawLut, SetFix, SetColor, SetRgb, SetHsi, WriteLut

Alternatives
DrawLut, SetFix, SetPixel

Module

System

void HWindowX.SetLutStyle ([in] double Hue, [in] double Saturation,
[in] double Intensity )

void HOperatorSetX.SetLutStyle ([in] VARIANT WindowHandle,

Changing the look-up-table (lut).
SetLutStyle changes the look-up-table (lut) of the device displaying the valid output window. It has got three parameters:

**Hue**: Rotation of color space, Hue = 1.9 conforms to a one-time rotation of the color space. No changement: Hue = 0.0 Complement colors: Hue = 0.5

**Saturation**: Changement of saturation, No changement: Saturation = 1.0 Gray value image: Saturation = 0.0

**Intensity**: Changement of intensity, No changement: Intensity = 1.0 Black image: Intensity = 0.0

Changement affects only the part of a look-up-table that is used for displayng images. The parameter of modification remain until the next call of SetLutStyle. Calling SetLut has got no effect on these parameters.

```
read_image(Image,’affe’)
set_lut(WindowHandle,’color’)
repeat(:::)
  get_mbutton(WindowHandle,Row,Column,Button)
  eval(Row/300.0,Saturation)
  eval(Column/512.0,Hue)
  set_lut_style(WindowHandle,Hue,Saturation,1.0)
until(Button = 1).
```

**Result**
SetLutStyle returns TRUE if the window is valid and the parameter is correct. Otherwise an exception handling is raised.

**Parallelization Information**
SetLutStyle is reentrant, local, and processed without parallelization.

**Possible Predecessors**
GetLutStyle

**Possible Successors**
SetLut

**See also**

**Alternatives**

SetLut, ScaleImage

**Module**
System
Write look-up-table (lut) as file.

WriteLut saves the look-up-table (resp. the part of it that is relevant for displaying image gray values) of the valid output window into a file named 'FileName.lut'. It can be read again later with SetLut.

Attention

WriteLut is only suitable for systems using 256 colors.

Parameter

- WindowHandle (input control) window \( \sim \) HWindowX / VARIANT Window identifier.
- FileName (input control) filename \( \sim \) String / VARIANT File name (of file containing the look-up-table).

Default Value: /tmp/lut

Example

read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_lut(WindowHandle)
write_lut(WindowHandle,'test_lut').

Result

DispLut returns TRUE if the window with the demanded properties (256 colors) is valid and the parameter (file-name) is correct. It returns FAIL if the specified file can’t be opened. Otherwise an exception handling is raised.

Parallelization Information

WriteLut is reentrant, local, and processed without parallelization.

Possible Predecessors

DrawLut, SetLut

See also

SetLut, DrawLut, SetPixel, GetPixel

Module

4.4 Mouse

Wait until a mouse button is pressed.

GetMbutton returns the coordinates of the mouse pointer in the output window and the mouse button pressed (Button):

1: Left button,
2: Middle button,
4: Right button.

The operator waits until a button is pressed in the output window. If more than one button is pressed, the sum of the individual buttons’ values is returned. The origin of the coordinate system is located in the left upper corner.
of the window. The row coordinates increase towards the bottom, while the column coordinates increase towards the right. For graphics windows, the coordinates of the lower right corner are (image height-1, image width-1) (see OpenWindow, ResetObjDb), while for text windows they are (window height-1, window width-1) (see OpenTextWindow).

GetMbutton only returns if a mouse button is pressed in the window.

Parameter

- WindowHandle (input control) ......................... window \( \sim \) HWindowX / VARIANT
  Window identifier.
- Row (output control) ................................. point.y \( \sim \) long / VARIANT
  Row coordinate of the mouse position in the window.
- Column (output control) .............................. point.x \( \sim \) long / VARIANT
  Column coordinate of the mouse position in the window.
- Button (output control) .............................. integer \( \sim \) long / VARIANT
  Mouse button(s) pressed.

Result

GetMbutton returns the value TRUE.

Parallelization Information

GetMbutton is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextwindow

See also

OpenWindow, OpenTextwindow

Alternatives

GetMposition

Module

System

[out] long Row HWindowX.GetMposition ([out] long Column,
[out] long Button )

void HOperatorSetX.GetMposition ([in] VARIANT WindowHandle,

Query the mouse position.

GetMposition returns the coordinates of the mouse pointer in the output window and the mouse button pressed. These values are returned regardless of the state of the mouse buttons (pressed or not pressed). If more than one button is pressed, the sum of the individual buttons’ values is returned. The possible values for Button are:

0: No button,
1: Left button,
2: Middle button,
4: Right button.

The origin of the coordinate system is located in the left upper corner of the window. The row coordinates increase towards the bottom, while the column coordinates increase towards the right. For graphics windows, the coordinates of the lower right corner are (image height-1, image width-1) (see OpenWindow, ResetObjDb), while for text windows they are (window height-1, window width-1) (see OpenTextWindow).

Attention

GetMposition fails (returns FAIL) if the mouse pointer is not located within the window. In this case, no values are returned.
4.4. MOUSE

Parameter

- **WindowHandle** (input control) .............................................. window ~ HWindowX / VARIANT
  Window identifier.
- **Row** (output control) ......................................................... point.y ~ long / VARIANT
  Row coordinate of the mouse position in the window.
- **Column** (output control) ..................................................... point.x ~ long / VARIANT
  Column coordinate of the mouse position in the window.
- **Button** (output control) ..................................................... integer ~ long / VARIANT
  Mouse button(s) pressed or 0.

Result

*GetMposition* returns the value TRUE. If the mouse pointer is not located within the window, FAIL is returned.

Parallelization Information

*GetMposition* is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextwindow

See also

Possible Successors

SetMshape

Alternatives

SetMshape, QueryMshape

Module

System

```
[out] String Cursor HWindowX.GetMshape ( )

void HOperatorSetX.GetMshape ([in] VARIANT WindowHandle,
[out] VARIANT Cursor )
```

Query the current mouse pointer shape.

*GetMshape* returns the name of the pointer shape set for the window. The mouse pointer shape can be used in the operator *SetMshape*.

Parameter

- **WindowHandle** (input control) .............................................. window ~ HWindowX / VARIANT
  Window identifier.
- **Cursor** (output control) ..................................................... string ~ String / VARIANT
  Mouse pointer name.

Result

*GetMshape* returns the value TRUE.

Parallelization Information

*GetMshape* is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextwindow, QueryMshape

Possible Successors

SetMshape

See also

SetMshape, QueryMshape

Module

System

HALCON 6.1.4
Query all available mouse pointer shapes.

QueryMshape returns the names of all available mouse pointer shapes for the window. These can be used in the operator SetMshape. If no mouse pointers are available, the empty tuple is returned.

\[
\text{[out]} \text{ VARIANT ShapeNames HWindowX.QueryMshape}() \\
\text{void HOperatorSetX.QueryMshape([in] VARIANT WindowHandle,} \\
\text{[out]} \text{ VARIANT ShapeNames )}
\]

Set the current mouse pointer shape.

SetMshape sets the shape of the mouse pointer for the window. A list of the names of all available mouse pointer shapes can be obtained by calling QueryMshape. The mouse pointer shape given by Cursor is used if the mouse pointer enters the window, irrespective of which window is the output window at present.

\[
\text{void HWindowX.SetMshape([in] String Cursor )} \\
\text{void HOperatorSetX.SetMshape([in] VARIANT WindowHandle,} \\
\text{[in]} \text{ VARIANT Cursor )}
\]
4.5 Output


Displays circular arcs in a window.

DispArc displays one or several circular arcs in the output window. An arc is described by its center point (CenterRow,CenterCol), the angle between start and end of the arc (Angle in radians) and the first point of the arc (BeginRow,BeginCol). The arc is displayed in clockwise direction. The parameters for output can be determined - as with the output of regions - with the procedures SetColor, SetGray, SetDraw, etc. It is possible to draw several arcs with one call by using tuple parameters. For the use of colors with several arcs, see SetColor.

Parameter

\textbf{WindowHandle} (input control) \hspace{1cm} \text{window $\sim$ HWindowX / VARIANT}\hspace{1cm} \text{Window identifier.}

\textbf{CenterRow} (input control) \hspace{1cm} \text{arc.center.y $\sim$ VARIANT (integer, real)} \hspace{1cm} \text{Row coordinate of center point.}

\textbf{Default Value} : 64

\textbf{Suggested values} : CenterRow $\in \{0, 64, 128, 256\}$

\textbf{Typical range of values} : $0 \leq \text{CenterRow} \leq 0$(lin)

\textbf{Minimum Increment} : 1

\textbf{Recommended Increment} : 1

\textbf{CenterCol} (input control) \hspace{1cm} \text{arc.center.x $\sim$ VARIANT (integer, real)} \hspace{1cm} \text{Column coordinate of center point.}

\textbf{Default Value} : 64

\textbf{Suggested values} : CenterCol $\in \{0, 64, 128, 256\}$

\textbf{Typical range of values} : $0 \leq \text{CenterCol} \leq 0$(lin)

\textbf{Minimum Increment} : 1

\textbf{Recommended Increment} : 1

\textbf{Angle} (input control) \hspace{1cm} \text{arc.angle.rad $\sim$ VARIANT (integer, real)} \hspace{1cm} \text{Angle between start and end of the arc (in radians).}

\textbf{Default Value} : 3.1415926

\textbf{Suggested values} : Angle $\in \{0.0, 0.785398, 1.570796, 3.1415926, 6.283185\}$

\textbf{Typical range of values} : $0.0 \leq \text{Angle} \leq 0.0$(lin)

\textbf{Minimum Increment} : 0.01

\textbf{Recommended Increment} : 0.1

\textbf{Restriction} : ($\text{Angle} > 0.0$)

\textbf{BeginRow} (input control) \hspace{1cm} \text{arc.begin.y(-array) $\sim$ VARIANT (integer, real)} \hspace{1cm} \text{Row coordinate of the start of the arc.}

\textbf{Default Value} : 32

\textbf{Suggested values} : BeginRow $\in \{0, 64, 128, 256\}$

\textbf{Typical range of values} : $0 \leq \text{BeginRow} \leq 0$(lin)

\textbf{Minimum Increment} : 1

\textbf{Recommended Increment} : 1

\textbf{BeginCol} (input control) \hspace{1cm} \text{arc.begin.x(-array) $\sim$ VARIANT (integer, real)} \hspace{1cm} \text{Column coordinate of the start of the arc.}

\textbf{Default Value} : 32

\textbf{Suggested values} : BeginCol $\in \{0, 64, 128, 256\}$

\textbf{Typical range of values} : $0 \leq \text{BeginCol} \leq 0$(lin)

\textbf{Minimum Increment} : 1

\textbf{Recommended Increment} : 1

Attention

The center point has to be within the window. The radius of the arc has be at least 2 pixel.
open_window(0,0,-1,-1,’root’,’visible’,’’,WindowHandle)
set_draw(WindowHandle,’fill’)
set_color(WindowHandle,’white’)
set_insert(WindowHandle,’not’)
Row = 100
Column = 100
disp_arc(WindowHandle,Row,Column,3.14,Row+10,Column+10)
close_window(WindowHandle).

Result
DispArc returns TRUE.

Parallelization Information
DispArc is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi

See also
OpenWindow, OpenTextwindow, SetColor, SetDraw, SetRgb, SetHsi

Alternatives
DispCircle, DispEllipse, DispRegion, GenCircle, GenEllipse

Module
System


Displays arrows in a window.
DispArc displays one or several arrows in the output window. An arrow is described by the coordinates of the start (Row1,Column1) and the end (Row2,Column2). An arrowhead is displayed at the end of the arrow. The size of the arrowhead is specified by the parameter Size. If the arrow consists of just one point (start = end) nothing is displayed. The procedures used to control the display of regions (e.g. SetDraw, SetColor, SetLineWidth) can also be used with arrows. Several arrows can be displayed with one call by using tuple parameters. For the use of colors with several arcs, see SetColor.

Attention
The start and the end of the arrows must fall within the window.

Parameter

- **WindowHandle** (input control) .......................... window ↦ HWindowX / VARIANT Window identifier.
- **Row1** (input control) ............................... .line.begin.y(-array) ↦ VARIANT( integer, real )
  Row index of the start.
  **Default Value** : 10.0
  **Suggested values** : Row1 ∈ {0.0, 64.0, 128.0, 256.0}
  **Typical range of values** : 0.0 ≤ Row1 ≤ 0.0(lin)
  **Minimum Increment** : 1.0
  **Recommended Increment** : 1.0
4.5. OUTPUT

- **Column1** (input control) .......................... line.begin.x(-array) ➔ VARIANT (integer, real)
  - Column index of the start.
  - Default Value: 10.0
  - Suggested values: Column1 ∈ {0.0, 64.0, 128.0, 256.0}
  - Typical range of values: 0.0 ≤ Column1 ≤ 0.0
  - Minimum Increment: 1.0
  - Recommended Increment: 1.0

- **Row2** (input control) .......................... line.end.y(-array) ➔ VARIANT (integer, real)
  - Row index of the end.
  - Default Value: 118.0
  - Suggested values: Row2 ∈ {0.0, 64.0, 128.0, 256.0}
  - Typical range of values: 0.0 ≤ Row2 ≤ 0.0
  - Minimum Increment: 1.0
  - Recommended Increment: 1.0

- **Column2** (input control) .......................... line.end.x(-array) ➔ VARIANT (integer, real)
  - Column index of the end.
  - Default Value: 118.0
  - Suggested values: Column2 ∈ {0.0, 64.0, 128.0, 256.0}
  - Typical range of values: 0.0 ≤ Column2 ≤ 0.0
  - Minimum Increment: 1.0
  - Recommended Increment: 1.0

- **Size** (input control) .......................... number ➔ VARIANT (integer, real)
  - Size of the arrowhead.
  - Default Value: 1.0
  - Suggested values: Size ∈ {1.0, 2.0, 3.0, 5.0}
  - Typical range of values: 0.0 ≤ Size ≤ 0.0
  - Minimum Increment: 1.0
  - Recommended Increment: 1.0
  - Restriction: (Size > 0.0)

---

**Example**

```plaintext
set_color(WindowHandle, ['red', 'green'])
disp_arrow(WindowHandle, [10,10], [10,10], [118,110], [118,118], 1.0).
```

---

**Result**

*DispArrow* returns TRUE.

**Parallelization Information**

*DispArrow* is reentrant, local, and processed without parallelization.

**Possible Predecessors**

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi

**See also**

OpenWindow, OpenTextWindow, SetColor, SetDraw, SetLineWidth

**Alternatives**

DispLine, GenRegionPolygon, DispRegion

**Module**

System

```plaintext

void HImageXDispChannel ([in] HWindowX WindowHandle,
[in] VARIANT Channel )

void HWindowXDispChannel ([in] HImageX MultichannelImage,
[in] VARIANT Channel )

void HOperatorSetXDispChannel ([in] IObjectX MultichannelImage,
[in] VARIANT WindowHandle, [in] VARIANT Channel )
```

Displays images with several channels.
**DispChannel** displays an image in the output window. It is possible to display several images with one call. In this case the images are displayed one after another. If the definition domains of the images overlap only the last image is visible. The parameter **Channel** defines the number of the channel that is displayed. For RGB-images the three color channels have to be used within a tuple parameter. For more information see **DispImage**.

**Parameter**

- **MultichannelImage** (input iconic) ...... image(-array) ~ HImageX / IHOBJECTX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  
  Multichannel images to be displayed.

- **WindowHandle** (input control) .............................................. window ~ HWindowX / VARIANT
  
  Window identifier.

- **Channel** (input control) .............................................. integer(-array) ~ VARIANT(integer)
  
  Number of channel or the numbers of the RGB-channels
  
  **Default Value** : 1
  
  **List of values** : Channel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

**Example**

```c
/* Transformation from rgb to gray */
read_image(Image, 'patras')
disp_color(Image, WindowHandle)
rgb1_to_gray(Image, GrayImage)
disp_image(GrayImage, WindowHandle).
```

**Result**

If the used images contain valid values and a correct output mode is set, **DispChannel** returns TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

**DispChannel** is reentrant, local, and processed without parallelization.

**Possible Predecessors**

OpenWindow, SetRgb, SetLut, SetHsi

**See also**

OpenWindow, OpenTextwindow, ResetObjDb, SetLut, DrawLut, DumpWindow

**Alternatives**

DispImage, DispColor

**System**

```c
void HWindowX.DispCircle ([in] VARIANT Row, [in] VARIANT Column,
[in] VARIANT Radius )

void HOperatorSetX.DispCircle ([in] VARIANT WindowHandle,
```

**Displays circles in a window.**

**DispCircle** displays one or several circles in the output window. A circle is described by the center (**Row**, **Column**) and the radius **Radius**. If the used coordinates are not within the window the circle is clipped accordingly.

The procedures used to control the display of regions (e.g. **SetDraw**, **SetGray**, **SetDraw**) can also be used with circles. Several circles can be displayed with one call by using tuple parameters. For the use of colors with several circles, see **SetColor**.

**Attention**

The center of the circle must be within the window.
4.5. OUTPUT

Parameter

- **WindowHandle** (input control) ................................. window \( \sim \) HWindowX / VARIANT
  Window identifier.

- **Row** (input control) ................................. circle.center.y(-array) \( \sim \) VARIANT (integer, real)
  Row index of the center.
  Default Value: 64
  Suggested values: \( \text{Row} \in \{0, 64, 128, 256\} \)
  Typical range of values: \( 0 \leq \text{Row} \leq 0(\text{lin}) \)
  Minimum Increment: 1
  Recommended Increment: 1

- **Column** (input control) ................................. circle.center.x(-array) \( \sim \) VARIANT (integer, real)
  Column index of the center.
  Default Value: 64
  Suggested values: \( \text{Column} \in \{0, 64, 128, 256\} \)
  Typical range of values: \( 0 \leq \text{Column} \leq 0(\text{lin}) \)
  Minimum Increment: 1
  Recommended Increment: 1

- **Radius** (input control) ................................. circle.radius(-array) \( \sim \) VARIANT (integer, real)
  Radius of the circle.
  Default Value: 64
  Suggested values: \( \text{Radius} \in \{0, 64, 128, 256\} \)
  Typical range of values: \( 0 \leq \text{Radius} \leq 0(\text{lin}) \)
  Minimum Increment: 1
  Recommended Increment: 1
  
  Restriction: \((\text{Radius} > 0.0)\)

Example

```plaintext
open_window(0,0,-1,-1,'root','visible',' ',WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
set_insert(WindowHandle,'not')
repeat()
    get_mbutton(WindowHandle,Row,Column,Button)
    disp_circle(WindowHandle,Row,Column,(Row + Column) mod 50)
until(Button = 1)
close_window(WindowHandle).
```

Result

**DispCircle** returns TRUE.

Parallelization Information

**DispCircle** is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi

See also

OpenWindow, OpenTextwindow, SetColor, SetDraw, SetRgb, SetHsi

Alternatives

DispEllipse, DispRegion, GenCircle, GenEllipse

Module

System
Displays a color (RGB) image

DispColor displays the three channels of a color image in the output window. The channels are ordered in the sequence (red, green, blue). DispColor can be simulated by DispChannel.

Attention

Due to the restricted number of available colors the color appearance is usually different from the original.

Parameter

- **ColorImage** (input iconic) ...... image \(\sim\) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dxf)
  
  Color image to display.

- **WindowHandle** (input control) ......................... window \(\sim\) HWindowX / VARIANT Window identifier.

Result

If the used image contains valid values and a correct output mode is set, DispColor returns TRUE. Otherwise an exception handling is raised.

Parallelization Information

DispColor is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, SetRgb, SetLut, SetHsi

See also

DispImage, OpenWindow, OpenTextwindow, ResetObjDb, SetLut, DrawLut, DumpWindow

Alternatives

DispChannel, DispObj

System

Displays a noise distribution.

DispDistribution displays a distribution in the window. The parameters are the same as in SetPaint(WindowHandle,‘histogram’) or GenRegionHisto. Noise distributions can be generated with operations like GaussDistribution or NoiseDistributionMean.

Parameter

- **WindowHandle** (input control) ......................... window \(\sim\) HWindowX / VARIANT Window identifier.

- **Distribution** (input control) ......................... real \(\sim\) VARIANT(real)
  
  Gray value distribution (513 values).

- **Row** (input control) ................................. point.y \(\sim\) long / VARIANT
  
  Row index of center.

  Default Value : 256

  Suggested values : Row \(\in\) \{0, 64, 128, 256\}

  Typical range of values : \(0 \leq\) Row \(\leq\) 0(lin)

  Minimum Increment : 1

  Recommended Increment : 10
### 4.5. OUTPUT

#### Column (input control)

- **point.x** \(\sim\) long / VARIANT
- **Default Value**: 256
- **Suggested values**: Column \(\in\) \{0, 64, 128, 256\}
- **Typical range of values**: 0 \(\leq\) Column \(\leq\) 0(lin)
- **Minimum Increment**: 1
- **Recommended Increment**: 10

- **Column index of center.**

#### Scale (input control)

- **Size of display.**
- **Default Value**: 1
- **Suggested values**: Scale \(\in\) \{1, 2, 3, 4, 5, 6\}

---

**Example**

```c
open_window(0,0,-1,-1,'root','visible',''WindowHandle)
set_draw(WindowHandle,'fill')
set_color(WindowHandle,'white')
set_insert(WindowHandle,'not')
read_image(Image,'affe')
draw_region(Region,WindowHandle)
normal_distribution_mean(Region,Image,21,Distribution)
disp_distribution (WindowHandle,Distribution,100,100,3).
```

---

**Parallelization Information**

- **DispDistribution** is reentrant, local, and processed without parallelization.

**Possible Predecessors**

- OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi, NoiseDistributionMean, GaussDistribution

**See also**

- GenRegionHisto, SetPaint, GaussDistribution, NoiseDistributionMean

**Module**

---

```c
void HWindowXDispEllipse ([in] VARIANT CenterRow,
[in] VARIANT CenterCol, [in] VARIANT Phi, [in] VARIANT Radius1,
[in] VARIANT Radius2 )

void HOperatorSetXDispEllipse ([in] VARIANT WindowHandle,
[in] VARIANT CenterRow, [in] VARIANT CenterCol, [in] VARIANT Phi,
[in] VARIANT Radius1, [in] VARIANT Radius2 )
```

- **Displays ellipses.**

**DispEllipse** displays one or several ellipses in the output window. An ellipse is described by the center (CenterRow, CenterCol), the orientation Phi (in radians) and the radii of the major and the minor axis (Radius1 and Radius2).

The procedures used to control the display of regions (e.g. SetDraw, SetGray, SetDraw) can also be used with ellipses. Several ellipses can be displayed with one call by using tuple parameters. For the use of colors with several ellipses, see SetColor.

---

**Attention**

- The center of the ellipse must be within the window.

---

**Parameter**

- **WindowHandle** (input control) \(\sim\) HWindowX / VARIANT
- **Window identifier.**
CenterRow (input control) .......................... ellipse.center.y(-array)  \sim \text{VARIANT (integer)}
Row index of center.
Default Value : 64
Suggested values : CenterRow \in \{0, 64, 128, 256\}
Typical range of values : 0 \leq \text{CenterRow} \leq 0\text{(lin)}
Minimum Increment : 1
Recommended Increment : 10

CenterCol (input control) .......................... ellipse.center.x(-array)  \sim \text{VARIANT (integer)}
Column index of center.
Default Value : 64
Suggested values : CenterCol \in \{0, 64, 128, 256\}
Typical range of values : 0 \leq \text{CenterCol} \leq 0\text{(lin)}
Minimum Increment : 1
Recommended Increment : 10

Phi (input control) ................................. ellipse.angle.rad(-array)  \sim \text{VARIANT (integer, real)}
Orientation of the ellipse in radians
Default Value : 0.0
Suggested values : \Phi \in \{0.0, 0.785398, 1.570796, 3.1415926, 6.283185\}
Typical range of values : 0.0 \leq \Phi \leq 0.0\text{(lin)}
Minimum Increment : 0.01
Recommended Increment : 0.1

Radius1 (input control) ............................. ellipse.radius1(-array)  \sim \text{VARIANT (integer, real)}
Radius of major axis.
Default Value : 24.0
Suggested values : Radius1 \in \{0.0, 64.0, 128.0, 256.0\}
Typical range of values : 0.0 \leq \text{Radius1} \leq 0.0\text{(lin)}
Minimum Increment : 1.0
Recommended Increment : 10.0

Radius2 (input control) ............................. ellipse.radius2(-array)  \sim \text{VARIANT (integer, real)}
Radius of minor axis.
Default Value : 14.0
Suggested values : Radius2 \in \{0.0, 64.0, 128.0, 256.0\}
Typical range of values : 0.0 \leq \text{Radius2} \leq 0.0\text{(lin)}
Minimum Increment : 1.0
Recommended Increment : 10.0

Example

set_color(WindowHandle, 'red')
draw_region(MyRegion,WindowHandle)
elliptic_axis(MyRegionRa,Rb,Phi)
area_center(MyRegion,_,Row,Column)
disp_ellipse(WindowHandle,Row,Column,Phi,Ra,Rb).

Result
DispEllipse returns TRUE, if the parameters are correct. Otherwise an exception handling is raised.

Parallelization Information
DispEllipse is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi, EllipticAxis, AreaCenter

See also
OpenWindow, OpenTextwindow, SetColor, SetRgb, SetHsi, SetDraw, SetLineWidth

Alternatives
DispCircle, DispRegion, GenEllipse, GenCircle

Module
System

HALCON/COM Reference Manual, 2005-2-1
Displays gray value images.

DispImage displays the gray values of an image in the output window. The gray value pixels of the definition domain (SetComprise(::WindowHandle,'object':)) or of the whole image (SetComprise(::WindowHandle,'image':)) are used. Restriction to the definition domain is the default.

For the display of gray value images the number of gray values is usually reduced. This is due to the fact that colors have to be reserved for the display of graphics (e.g. SetColor) and the window manager. Also depending on the number of bitplanes on the used output device often less than 256 colors (eight bitplanes) are available. The number of "colors" actually reserved for the display of gray values can be queried by GetSystem. Before opening the first window this value can be modified by SetSystem. For instance for 8 bitplanes 200 real gray values are the default.

The reduction of the number of gray values does not pose problems as long as only gray value information is displayed, humans cannot distinguish 256 different shades of gray. If certain gray values are used for the representation of region information (which is not the style commonly used in HALCON), confusions might be the result, since different numerical values are displayed on the screen with the same gray value. The procedure LabelToRegion should be used on these images in order to transform the label data into HALCON objects.

If images of type 'int2', 'int4', 'real' or 'complex' are displayed, the smallest and largest gray value is computed. Afterwards the pixel data is rescaled according to the number of available gray values (depending on the output device. e.g. 200). It is possible that some pixels have a very different value than the other pixels. This might lead to the display of an (almost) completely white or black image. In order to decide if the current image is a binary image MinMaxGray can be used. If necessary the image can be transformed or converted by ScaleImage and ConvertImageType before it is displayed.

Attention

If a wrong output mode was set by SetPaint, the error will be reported when DispImage is used.

Parameter

▷ Image (input iconic) ...... image  \rightarrow HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Gray value image to display.

▷ WindowHandle (input control) .............................................. window  \rightarrow HWindowX / VARIANT

Window identifier.

/* Output of a gray image: */
read_image(Image1,'affe')
disp_image(Image1,WindowHandle).

Result

If the used image contains valid values and a correct output mode is set, DispImage returns TRUE. Otherwise an exception handling is raised.

Parallelization Information

DispImage is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, SetRgb, SetLut, SetHsi, ScaleImage, ConvertImageType, MinMaxGray

See also

OpenWindow, OpenTextwindow, ResetObjDb, SetComprise, SetPaint, SetLut, DrawLut, PaintGray, ScaleImage, ConvertImageType, DumpWindow

Alternatives

DispObj, DispColor

Module

System

HALCON 6.1.4


Draws lines in a window.

DispLine displays one or several lines in the output window. A line is described by the coordinates of the start (Row1,Column1) and the coordinates of the end (Row2,Column2). The procedures used to control the display of regions (e.g. SetColor, SetGray, SetDraw, SetLineWidth) can also be used with lines. Several lines can be displayed with one call by using tuple parameters. For the use of colors with several lines, see SetColor.

Attention

The starting points and the ending points of the lines must be in the window.

Parameter

WindowHandle (input control) ......................... window HWindowX / VARIANT
Window identifier.

Row1 (input control) ......................... line.begin.y(-array) VARIANT (real)
Row index of the start.
Default Value : 32
Suggested values : Row1 ∈ {0, 64, 128, 256, 511}
Typical range of values : 0 ≤ Row1 ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 10

Column1 (input control) ......................... line.begin.x(-array) VARIANT (real)
Column index of the start.
Default Value : 32
Suggested values : Column1 ∈ {0, 64, 128, 256, 511}
Typical range of values : 0 ≤ Column1 ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 10

Row2 (input control) ......................... line.end.y(-array) VARIANT (real)
Row index of end.
Default Value : 64
Suggested values : Row2 ∈ {0, 64, 128, 256, 511}
Typical range of values : 0 ≤ Row2 ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 10

Column2 (input control) ......................... line.end.x(-array) VARIANT (real)
Column index of end.
Default Value : 64
Suggested values : Column2 ∈ {0, 64, 128, 256, 511}
Typical range of values : 0 ≤ Column2 ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 10

Example

/* Prozedur zur Ausgabe der Kontur eines Rechtecks: */
disp_rectangle1_margin(WindowHandle,Row1,Column1,Row2,Column2):
disp_line(WindowHandle,Row1,Column1,Row1,Column2)
disp_line(WindowHandle,Row1,Column2,Row2,Column2)
disp_line(WindowHandle,Row2,Column2,Row2,Column1)
disp_line(WindowHandle,Row2,Column1,Row1,Column1).

DispLine returns TRUE.
Parallelization Information
DispLine is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

See also
OpenWindow, OpenTextwindow, SetColor, SetRgb, SetHsi, SetInsert, SetLineWidth

Alternatives
DispArrow, DispRectangle1, DispRectangle2, DispRegion, GenRegionPolygon, GenRegionPoints

Module
System

void HObjectX.DispObj ([in] HWindowX WindowHandle )
void HWindowX.DispObj ([in] IHObjectX Object )
void HOperatorSetX.DispObj ([in] IHObjectX Object , [in] VARIANT WindowHandle )

Displays image objects (image, region, XLD).
DispObj displays objects depending of their kind. DispObj is equivalent to DispImage for one channel images, equivalent to DispColor for three channel images, equivalent to DispRegion for regions and equivalent to DispXld for XLDs.

Parameter

▷ Object (input iconic) ................................................................. object(-array) ~ IHObjectX
Image object to be displayed.
▷ WindowHandle (input control) ................................. window ~ HWindowX / VARIANT
Window identifier.

Example

/* Output of a gray image: */
read_image(Image1,’affe’)
disp_obj(Image1,WindowHandle)
threshold(Image,Region,0,128)
disp_obj(Region,WindowHandle)

Result
If the used object is valid and a correct output mode is set, DispObj returns TRUE. Otherwise an exception handling is raised.

Parallelization Information
DispObj is reentrant, local, and processed without parallelization.
Displays a polyline.

`DispPolygon` displays a polyline with the row coordinates `Row` and the column coordinates `Column` in the output window. The parameters `Row` and `Column` have to be provided as tuples. Straight lines are drawn between the given points. The start and the end of the polyline are not connected.

The procedures used to control the display of regions (e.g. `SetColor`, `SetGray`, `SetDraw`, `SetLineWidth`) can also be used with polylines.

Attention

The given coordinates must lie within the window.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Row</code></td>
<td>Row index</td>
</tr>
<tr>
<td><code>Column</code></td>
<td>Column index</td>
</tr>
</tbody>
</table>

Default Value: `Row ∈ {0, 64, 128, 256, 511}`  
Suggested values: `0 ≤ Row ≤ 0(lin)`  
Minimum Increment: 1  
Recommended Increment: 10

Default Value: `Column ∈ {0, 64, 128, 256, 511}`  
Suggested values: `0 ≤ Column ≤ 0(lin)`  
Minimum Increment: 1  
Recommended Increment: 10

Result

`DispPolygon` returns TRUE.

Parallelization Information

`DispPolygon` is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

See also

OpenWindow, OpenTextwindow, SetColor, SetRgb, SetHsi, SetInsert, SetLineWidth

Alternatives

DispLine, GenRegionPolygon, DispRegion

Module

System
regions (e.g. `SetColor`, `SetGray`, `SetDraw`, `SetLineWidth`) can also be used with rectangles. Several rectangles can be displayed with one call by using tuple parameters.

---

**Parameter**

- **`WindowHandle`** (input control)  
  window $\sim$ `HWindowX` / `VARIANT`  
  Window identifier.

- **`Row1`** (input control)  
  rectangle.origin.y(-array) $\sim$ `VARIANT` (integer, real)  
  Row index of the upper left corner.
  - **Default Value**: 16  
  - **Suggested values**: $Row1 \in \{0, 64, 128, 256, 511\}$  
  - **Typical range of values**: $0 \leq Row1 \leq 0(lin)$  
  - **Minimum Increment**: 1  
  - **Recommended Increment**: 10

- **`Column1`** (input control)  
  rectangle.origin.x(-array) $\sim$ `VARIANT` (integer, real)  
  Column index of the upper left corner.
  - **Default Value**: 16  
  - **Suggested values**: $Column1 \in \{0, 64, 128, 256, 511\}$  
  - **Typical range of values**: $0 \leq Column1 \leq 0(lin)$  
  - **Minimum Increment**: 1  
  - **Recommended Increment**: 10

- **`Row2`** (input control)  
  rectangle.corner.y(-array) $\sim`VARIANT` (integer, real)  
  Row index of the lower right corner.
  - **Default Value**: 48  
  - **Suggested values**: $Row2 \in \{0, 64, 128, 256, 511\}$  
  - **Typical range of values**: $0 \leq Row2 \leq 0(lin)$  
  - **Minimum Increment**: 1  
  - **Recommended Increment**: 10  
  - **Restriction**: $(Row2 \geq Row1)$

- **`Column2`** (input control)  
  rectangle.corner.x(-array) $\sim`VARIANT` (integer, real)  
  Column index of the lower right corner.
  - **Default Value**: 80  
  - **Suggested values**: $Column2 \in \{0, 64, 128, 256, 511\}$  
  - **Typical range of values**: $0 \leq Column2 \leq 0(lin)$  
  - **Minimum Increment**: 1  
  - **Recommended Increment**: 10  
  - **Restriction**: $(Column2 \geq Column1)$

---

**Example**

```plaintext
set_color(WindowHandle,'green')
draw_region(MyRegion,WindowHandle)
smallest_rectangle1(MyRegion,R1,C1,R2,C2)
disp_rectangle1(WindowHandle,R1,C1,R2,C2).
```

---

**Result**

`DispRectangle1` returns TRUE.

---

**Parallelization Information**

`DispRectangle1` is reentrant, local, and processed without parallelization.

---

**Possible Predecessors**

OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

---

**See also**

OpenWindow, OpenTextwindow, SetColor, SetDraw, SetLineWidth

---

**Alternatives**

`DispRectangle2`, `GenRectangle1`, `DispRegion`, `DispLine`, `SetShape`

---

**Module**

**System**

HALCON 6.1.4
Displays arbitrarily oriented rectangles.

DispRectangle2 draws one or several arbitrarily oriented rectangles in the output window. A rectangle is described by the center (CenterRow,CenterCol), the orientation Phi (in radians) and half the lengths of the edges Length1 and Length2. The procedures used to control the display of regions (e.g. SetDraw, SetGray, SetDraw) can also be used with rectangles. Several rectangles can be displayed with one call by using tuple parameters. For the use of colors with several rectangles, see SetColor.

Attention The center must lie within the window boundaries.

Parameter

- **WindowHandle** (input control) .......................... window \( \sim \) HWindowX / VARIANT Window identifier.
- **CenterRow** (input control) ......................... rectangle2.center.y(-array) \( \sim \) VARIANT (integer, real ) Row index of the center.
  - Default Value : 48
  - Suggested values : CenterRow \( \in \{0, 64, 128, 256, 511\} \)
  - Typical range of values : \( 0 \leq \text{CenterRow} \leq 0(\text{lin}) \)
  - Minimum Increment : 1
  - Recommended Increment : 10
- **CenterCol** (input control) ...................... rectangle2.center.x(-array) \( \sim \) VARIANT (integer, real ) Column index of the center.
  - Default Value : 64
  - Suggested values : CenterCol \( \in \{0, 64, 128, 256, 511\} \)
  - Typical range of values : \( 0 \leq \text{CenterCol} \leq 0(\text{lin}) \)
  - Minimum Increment : 1
  - Recommended Increment : 10
- **Phi** (input control) .......................... rectangle2.angle.rad(-array) \( \sim \) VARIANT (integer, real ) Orientation of rectangle in radians.
  - Default Value : 0.0
  - Suggested values : Phi \( \in \{0.0, 0.785398, 1.570796, 3.1415926, 6.283185\} \)
  - Typical range of values : \( 0.0 \leq \text{Phi} \leq 0.0(\text{lin}) \)
  - Minimum Increment : 0.01
  - Recommended Increment : 0.1
- **Length1** (input control) .......................... rectangle2.hwidth(-array) \( \sim \) VARIANT (integer, real ) Half of the length of the longer side.
  - Default Value : 48
  - Suggested values : Length1 \( \in \{0, 64, 128, 256, 511\} \)
  - Typical range of values : \( 0 \leq \text{Length1} \leq 0(\text{lin}) \)
  - Minimum Increment : 1
  - Recommended Increment : 10
- **Length2** (input control) .......................... rectangle2.hheight(-array) \( \sim \) VARIANT (integer, real ) Half of the length of the shorter side.
  - Default Value : 32
  - Suggested values : Length2 \( \in \{0, 64, 128, 256, 511\} \)
  - Typical range of values : \( 0 \leq \text{Length2} \leq 0(\text{lin}) \)
  - Minimum Increment : 1
  - Recommended Increment : 10
  - Restriction : \( \text{Length2} < \text{Length1} \)

Example

HALCON/COM Reference Manual, 2005-2-1
set_color(WindowHandle,'green')
draw_region(MyRegion:WindowHandle)
elliptic_axis(MyRegion,Ra,Rb,Phi)
area_center(MyRegion,_,Row,Column)
disp_rectangle2(WindowHandle,Row,Column,Phi,Ra,Rb).

Result

DispRectangle2 returns TRUE, if the parameters are correct. Otherwise an exception handling is raised.

Parallelization Information

DispRectangle2 is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, SetRgb, SetLut, SetHsi, SetDraw, SetColor, SetColored, SetLineWidth

See also

OpenWindow, OpenTextwindow, DispRegion, SetColor, SetDraw, SetLineWidth

Alternatives

DispRegion, GenRectangle2, DispRectangle1, SetShape

Module

System

void HRegionX.DispRegion ([in] HWindowX WindowHandle )
void HWindowX.DispRegion ([in] HRegionX DispRegions )
void HOperatorSetX.DispRegion ([in] IObjectX DispRegions, [in] VARIANT WindowHandle )

Displays regions in a window.

DispRegion displays the regions in DispRegions in the output window. The parameters for output can be set with the procedures SetColor, SetGray, SetDraw, SetLineWidth, etc.

The color(s) for the display of the regions are determined with SetColor, SetRgb, SetGray or SetColored. If more than one region is displayed and more than one color is set, the colors are assigned in a cyclic way to the regions.

The form of the region for output can be modified by SetPaint (e.g. encompassing circle, convex hull). The command SetDraw determines if the region is filled or only the boundary is drawn. If only the boundary is drawn, the thickness of the boundary will be determined by SetLineWidth and the style by SetLineStyle.

Parameter

- DispRegions (input iconic) region(-array) → HRegionX / IObjectX Regions to display.
- WindowHandle (input control) window → HWindowX / VARIANT Window identifier.

Example

/* Output with 12 colors: */
set_colored(WindowHandle,12)
disp_region(SomeSegments,WindowHandle).

/* Symbolic representation: */
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'red')
set_shape(WindowHandle,'ellipse')
disp_region(SomeSegmentsWindowHandle).

/* Representation of a margin with pattern: */
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'blue')
set_line_style(WindowHandle,[12,3])
disp_region(Segments,WindowHandle).

Result
DispRegion returns TRUE.

Parallelization Information
DispRegion is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, SetRgb, SetLut, SetHsi, SetShape, SetLineStyle, SetInsert, SetFix,
SetDraw, SetColor, SetColored, SetLineWidth

See also
OpenWindow, OpenTextwindow, SetColor, SetColored, SetDraw, SetShape, SetPaint,
SetGray, SetRgb, SetHsi, SetPixel, SetLineWidth, SetLineStyle, SetInsert, SetFix,
PaintRegion, DumpWindow

Alternatives
DispObj, DispArrow, DispLine, DispCircle, DispRectangle1, DispRectangle2,
DispEllipse

Module
System

void HXLDX.DispXld ([in] HWindowX WindowHandle )
void HWindowX.DispXld ([in] IHXLDX XLDObject )
void HOperatorSetX.DispXld ([in] IHObjectX XLDObject, [in] VARIANT WindowHandle )

Display an XLD object.
DispXld serves to display an XLD object of arbitrary type.

Parameter

- **XLDObject** (input iconic) .......................................................... xld  \( \sim \) IHXLDX / IHObjectX
XLD object to display.
- **WindowHandle** (input control) .......................... window  \( \sim \) HWindowX / VARIANT
Window id.

Parallelization Information
DispXld is local and processed completely exclusively without parallelization.

See also
DispImage, DispRegion, DispChannel, DispColor, DispLine, DispArc

Module

4.6 Parameters

[out] String Mode HInfoX.GetComprise ([in] HWindowX WindowHandle )
void HOperatorSetX.GetComprise ([in] VARIANT WindowHandle, [out] VARIANT Mode )

Get the output treatment of an image matrix.
GetComprise returns the output mode of grayvalues in the window WindowHandle that is used by
DispImage and DispColor. The output mode defines whether only the grayvalues of objects are displayed
or the whole image is displayed. The query is used for temporary mode settings, i.e., the current mode is queried,
then overwritten with (SetComprise) and finally reset.
4.6. PARAMETERS

**Attention**

**Parameter**

▷ **WindowHandle** (input control) window ⇒ HWindowX / VARIANT

▷ **Mode** (output control) string ⇒ String / VARIANT

**Result**

GetComprise returns TRUE if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

GetComprise is reentrant and processed without parallelization.

**Possible Successors**

SetComprise, DispImage, DispImage

See also

SetComprise, DispImage, DispColor

**Module**

System

```c
[out] String Mode HWindowX.GetDraw ( )

void HOperatorSetX.GetDraw ([in] VARIANT WindowHandle,
[out] VARIANT Mode )
```

Get the current region fill mode.

GetDraw returns the region fill mode of the output window. It is used by operators as DispRegion, DispCircle, DispArrow, DispRectangle1, DispRectangle2 etc. The region fill mode is set with SetDraw.

**Attention**

**Parameter**

▷ **WindowHandle** (input control) window ⇒ HWindowX / VARIANT

▷ **Mode** (output control) string ⇒ String / VARIANT

**Result**

GetDraw returns TRUE, if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

GetDraw is reentrant and processed without parallelization.

**Possible Successors**

SetDraw, DispRegion

See also

SetDraw, DispRegion, SetPaint

**Module**

System

```c
[out] String Mode HWindowX.GetFix ( )

void HOperatorSetX.GetFix ([in] VARIANT WindowHandle,
[out] VARIANT Mode )
```

Get mode of fixing of current look-up-table (lut).
Use \texttt{GetFix} to get mode of fixing of current look-up-table (look-up-table of valid window) set before by \texttt{SetFix}.

\begin{verbatim}
\begin{verbatim}
\textbf{Attention}  Parameter
\end{verbatim}
\end{verbatim}
\begin{itemize}
  \item \texttt{WindowHandle} (input control) \texttt{window \rightarrow HWindowX / VARIANT}
  \texttt{Window identifier}.
  \item \texttt{Mode} (output control) \texttt{string \rightarrow String / VARIANT}
  \texttt{Current Mode of fixing}.
\end{itemize}

\textbf{Result}  GetFix returns \texttt{TRUE} if the window is valid. Otherwise an exception handling is raised.

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\textbf{Parallelization Information}  GetFix is \texttt{reentrant}, \texttt{local}, and processed \texttt{without} parallelization.
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\textbf{Possible Successors}  SetFix, SetPixel, SetRgb
\end{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\textbf{Module}  System
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\begin{verbatim}
\begin{verbatim}
\end{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\begin{verbatim}
\end{verbatim}
\begin{verbatim}
\end{verbatim}

\beg...
4.6. PARAMETERS

```c
void HRegionX.GetIcon ([in] HWindowX WindowHandle )
[out] HRegionX Icon HWindowX.GetIcon ( )

void HOperatorSetX.GetIcon ([out] HUntypedObjectX Icon, [in] VARIANT WindowHandle )
```

Query the icon for region output

`GetIcon` queries the icon that was set with `SetIcon`.

Parameter

- **Icon** (output iconic)  
  region `~` HRegionX / HUntypedObjectX
  Icon for the regions center of gravity.

- **WindowHandle** (input control)  
  window `~` HWindowX / VARIANT
  Window id.

Result

`GetIcon` always returns TRUE.

Parallelization Information

`GetIcon` is reentrant and processed without parallelization.

Possible Predecessors

`SetIcon`

Possible Successors

DispRegion

System

```c
[out] String Mode HWindowX.GetInsert ( )

void HOperatorSetX.GetInsert ([in] VARIANT WindowHandle, [out] VARIANT Mode )
```

Get the current display mode.

`GetInsert` returns the display mode of the output window. It is used by procedures like `DispRegion, DispLine, DispRectangle`, etc. The mode is set with `SetInsert`. Possible values for `Mode` can be queried with the procedure `QueryInsert`.

Parameter

- **WindowHandle** (input control)  
  window `~` HWindowX / VARIANT
  Window id.

- **Mode** (output control)  
  string `~` String / VARIANT
  Display mode.

Result

`GetInsert` returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

`GetInsert` is reentrant and processed without parallelization.

Possible Predecessors

`QueryInsert`

Possible Successors

`SetInsert, DispImage`

See also

`SetInsert, QueryInsert, DispRegion, DispLine`

System

HALCON 6.1.4
Get the current approximation error for contour display.

GetLineApprox returns a parameter that controls the approximation error for region contour display in the window. It is used by the procedure DispRegion. Approximation controls the polygon approximation for contour display (0 ⇔ no approximation). Approximation is only important for displaying the contour of objects, especially if a line style was set with SetLineStyle.

Attention

Parameter

- WindowHandle (input control) window → HWindowX / VARIANT Window id.
- Approximation (output control) Current approximation error for contour display.

Result

GetLineApprox returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLineApprox is reentrant and processed without parallelization.

Possible Successors

SetLineApprox, SetLineStyle, DispRegion

See also

GetRegionPolygon, SetLineApprox, SetLineStyle, DispRegion

Module

System

Get the current graphic mode for contours.

GetLineStyle returns the display mode for contours when displaying regions. It is used by procedures like DispRegion, DispLine, DispPolygon, etc. Style is set with the procedure SetLineStyle. Style is only important for displaying the contour of objects, especially if a line style was set with SetLineStyle.

Attention

Parameter

- WindowHandle (input control) window → HWindowX / VARIANT Window id.
- Style (output control) Template for contour display.

Result

GetLineStyle returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLineStyle is reentrant, local, and processed without parallelization.

See also

SetLineStyle, DispRegion

Module

System
Get the current line width for contour display.

GetLineWidth returns the line width for region display in the window. It is used by procedures like DispRegion, DispLine, DispPolygon, etc. Width is set with the procedure SetLineWidth. Width is only important for displaying the contour of objects.

Attention

Parameter

▷ WindowHandle (input control) ........................................ window \( \sim \) HWindowX / VARIANT WindowId.
▷ Width (output control) ..................................................... integer \( \sim \) long / VARIANT
  Current line width for contour display.

Result

GetLineWidth returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetLineWidth is reentrant and processed without parallelization.

Possible Successors

SetLineWidth, SetLineStyle, DispRegion

See also

SetLineWidth, DispRegion

Module

System

Get the current display mode for grayvalues.

GetPaint returns the display mode for grayvalues in the window. Mode is used by the procedure DispImage. GetPaint is used for temporary changes of the grayvalue display mode. The current value is queried, then changed (with procedure SetPaint) and finally the old value is written back. The available modes can be viewed with the procedure QueryPaint. Mode is the name of the display mode. If a mode can be customized with parameters, the parameter values are passed in a tuple after the mode name. The order of values is the same as in SetPaint.

Attention

Parameter

▷ WindowHandle (input control) ........................................ window \( \sim \) HWindowX / VARIANT WindowId.
▷ Mode (output control) ..................................................... string \( \sim \) VARIANT( string, integer )
  Name and parameter values of the current display mode.

Result

GetPaint returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetPaint is reentrant and processed without parallelization.

Possible Predecessors

QueryPaint
Get the image part.

GetPart returns the upper left and lower right corner of the image part shown in the window. The image part can be changed with the procedure SetPart (Default is the whole image).

Attention

Parameter

- **WindowHandle** (input control) \( \text{window} \sim \text{HWWindowX/VARIANT Window.id.} \)
- **Row1** (output control) \( \text{rectangle.origin.y} \sim \text{long/VARIANT} \)
- **Column1** (output control) \( \text{rectangle.origin.x} \sim \text{long/VARIANT} \)
- **Row2** (output control) \( \text{rectangle.corner.y} \sim \text{long/VARIANT} \)
- **Column2** (output control) \( \text{rectangle.corner.x} \sim \text{long/VARIANT} \)

Result

GetPart returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetPart is reentrant and processed without parallelization.

Possible Successors

SetPart, DispRegion, DispImage

See also

SetPart, DispImage, DispRegion, DispColor

Module

System

Get the current interpolation mode for grayvalue display.

GetPartStyle returns the interpolation mode used for displaying an image part in the window. An interpolation takes place if the output window is larger than the image format or the image output format (see SetPart).

HALCON supports three interpolation modes:

- **0** no interpolation (low quality, very fast).
4.6. PARAMETERS

1 unweighted interpolation (average quality and computation time)
2 weighted interpolation (high quality, slow)

The current mode can be changed with SetPartStyle.

Attention Parameter

WindowHandle (input control) .......................... window ～ HWindowX / VARIANT

Style (output control) ................................. integer ～ long / VARIANT

List of values : Style ∈ \{0, 1, 2\}

GetPartStyle returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information
GetPartStyle is reentrant and processed without parallelization.

Possible Successors
SetPartStyle, DispRegion, DispImage

See also
SetPartStyle, SetPart, DispImage, DispColor

Module

Get the current color lookup table index.

GetPixel returns the internal coding of the output grayvalue or color, respectively, for the window. If the output mode is set to color(s) or grayvalue(s) (see SetColor or SetGray), then the color- or grayvalues are transformed for internal use. The internal code is then used for (physical) screen display. The transformation depends on the mapping characteristics and the condition of the output device and can be different in different program runs. Don’t confuse the term “pixel” with the term “pixel” in image processing (the other procedure is GetGrayval). Here a pixel is meant to be the color lookup table index.

With GetPixel it is possible to save the output mode without knowing whether colors or grayvalues are used. Pixel is set with the procedure SetPixel.

Attention Parameter

WindowHandle (input control) .......................... window ～ HWindowX / VARIANT

Pixel (output control) ................................. string ～ VARIANT (integer)

Index of the current color lookup table.

GetPartStyle returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information
GetPixel is reentrant and processed without parallelization.

Possible Successors
SetPixel, DispRegion, DispImage

See also
SetPixel, SetFix
Get the current color in RGB-coding.

GetRgb returns the output colors or grayvalues, respectively, for the output window. They are defined by the three color components red, green and blue.

GetRgb is like GetPixel but returns the entries of the color lookup table rather than the indices. The values returned by GetRgb can be set with SetRgb.

Parameter

- **WindowHandle** (input control) .................. window \( \sim \) HWindowX / VARIANT WindowId.
- **Red** (output control) .......................... integer \( \sim \) VARIANT (integer)
  The current color’s red value.
- **Green** (output control) ......................... integer \( \sim \) VARIANT (integer)
  The current color’s green value.
- **Blue** (output control) .......................... integer \( \sim \) VARIANT (integer)
  The current color’s blue value.

Result

GetRgb returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetRgb is reentrant and processed without parallelization.

Possible Successors

SetRgb, DispRegion, DispImage

See also

SetRgb

Module

Get the current region output shape.

GetShape returns the shape in which regions are displayed. The available shapes can be queried with QueryShape and then changed with SetShape.

Parameter

- **WindowHandle** (input control) .................. window \( \sim \) HWindowX / VARIANT WindowId.
- **DisplayShape** (output control) .................. string \( \sim \) String / VARIANT
  Current region output shape.
4.6. PARAMETERS

Result

GetShape returns TRUE if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

GetShape is reentrant and processed without parallelization.

Possible Predecessors

QueryShape

Possible Successors

SetShape, DispRegion

See also

SetShape, QueryShape, DispRegion

Module

System

VARIANT Colors HWindowX.QueryAllColors ( )

void HOperatorSetX.QueryAllColors ([in] VARIANT WindowHandle,
[in out] VARIANT Colors )

Query all color names.

QueryAllColors returns the names of all colors that are known to HALCON. That doesn’t mean, that these colors are available for specific screens. On some screens there may only be a subset of colors available (see QueryColor). Before opening the first window, SetSystem can be used to define which and how many colors should be used. The HALCON colors are used to display regions (DispRegion, DispPolygon, DispCircle, etc.). They can be defined with SetColor.

Attention

Parameter

- WindowHandle (input control) window ~ HWindowX / VARIANT Window_id.
- Colors (output control) string ~ VARIANT(string) Color names.

Example

query_all_colors(WindowHandle,Colors)
<interactive selection from Colors provide ActColors> >
set_system(’graphic_colors’,ActColors)
open_window(0,0,1,1,’root’,’invisible’,’,WindowHandle)
query_color(WindowHandle,F)
close_window(WindowHandle)
fwrite_string([’Setting Colors: ’,F]).

Result

QueryAllColors always returns TRUE.

Parallelization Information

QueryAllColors is reentrant, local, and processed without parallelization.

Possible Successors

SetSystem, SetColor, DispRegion

See also

QueryColor, SetSystem, SetColor, DispRegion, OpenWindow, OpenTextwindow

Module

System

HALCON 6.1.4
QueryColor returns the names of all colors that are usable for region output (DispRegion, DispPolygon, DispCircle, etc.). On a b/w screen QueryColor returns 'black' and 'white'. These two ''colors'' are displayable on any screen. In addition to 'black' and 'white' several grayvalues (e.g. 'dim gray') are returned on screens capable of grayvalues. A list of all displayable colors is returned for screens with color lookup table. The returned tuple of colors begins with b/w, followed by the three primaries ('red','green','blue') and several grayvalues. Before opening the first window it is furthermore possible to define the color list with SetSystem ('graphicColors',...):. QueryAllColors(::WindowHandle:Colors ) returns a list of all available colors for the SetSystem (::'graphicColors',...:) call. For screens with truecolor output the same list is returned by QueryColor. The list of available colors (to HALCON ) must not be confused with the list of displayable colors. For screens with truecolor output the available colors are only a small subset of the displayable colors. Colors that are not directly available to HALCON can be chosen manually with SetRgb or SetHsi. If colors are chosen that are known to HALCON but cannot be displayed, HALCON can choose a similar color. To use this feature, SetCheck (::'~color':) must be set.

--- Attention ---

Parameter

- WindowHandle (input control) ...................... window ~ HWindowX / VARIANT Window;id.
- Colors (output control) ............................ string ~ VARIANT ( string ) Color names.

Example

open_window(0,0,-1,-1,'root','invisible',' ',WindowHandle)
query_color(WindowHandle,Colors)
close_window(WindowHandle)
fwrite_string (['Displayable colors: ',Farben]).

Result

QueryColor returns TRUE, if the window is valid. Otherwise an exception handling is raised.

--- Parallelization Information ---

QueryColor is reentrant, local, and processed without parallelization.

--- Possible Successors ---

SetColor,DispRegion

See also

QueryAllColors,SetColor,DispRegion,OpenWindow,OpenTextwindow

--- Module ---

System
4.6. PARAMETERS

Parameter

▷ **PossibleNumberOfColors** (output control) ................................integer ~ VARIANT ( integer )
  Tuple of the possible numbers of colors.

Example

regiongrowing(Image, Seg, 5, 5, 6, 100)
query_colored(Colored)
set_colored(WindowHandle, Colors[1])
disp_region(Seg, WindowHandle).

Result

QueryColored always returns TRUE.

Parallelization Information

QueryColored is reentrant and processed without parallelization.

Possible Successors

SetColored, SetColor, DispRegion

See also

SetColored, SetColor

Alternatives

QueryColor

Module

System

```
[out] VARIANT Grayval HWindowX.QueryGray ()
void HOperatorSetX.QueryGray ([in] VARIANT WindowHandle,
[out] VARIANT Grayval )
```

Query the displayable grayvalues.

**QueryGray** returns all grayvalues that are used for grayvalue output (**DispImage**) and that can be reproduced exactly in the window. They can be set with the **SetGray** call. The number of displayable grayvalues can be set with **SetSystem(::'numGray*', ...)** before opening the first window.

Attention

Parameter

▷ **WindowHandle** (input control) ................................................. window ~ HWindowX / VARIANT Window_id.
▷ **Grayval** (output control) ..................................................... integer ~ VARIANT( integer )
  Tuple of all displayable grayvalues.

Result

QueryGray returns TRUE, if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

**QueryGray** is reentrant, local, and processed without parallelization.

Possible Successors

SetGray, DispRegion

See also

SetGray, DispImage

Module

System
CHAPTER 4. GRAPHICS

[out] VARIANT Mode HWindowX.QueryInsert ( )

void HOperatorSetX.QueryInsert ([in] VARIANT WindowHandle,
[out] VARIANT Mode )

Query the possible graphic modes.

QueryInsert returns the possible modes pixels can be displayed in the output window. New pixels may e.g. overwrite old ones. In most of the cases there is a functional relationship between old and new values.

Possible display functions:

'copy': overwrite displayed pixels
'xor': display old ‘xor’ new pixels
'complement': complement displayed pixels

‘“copy’’ is always available.

Parameter

- WindowHandle (input control) window ~ HWindowX / VARIANT WindowId.
- Mode (output control) string ~ VARIANT( string )

Result

QueryInsert returns TRUE, if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

QueryInsert is reentrant, local, and processed without parallelization.

Possible Successors

SetInsert, DispRegion

See also

SetInsert, GetInsert

Module

System

[out] long Min HInfoX.QueryLineWidth ([out] long Max )

void HOperatorSetX.QueryLineWidth ([out] VARIANT Min,
[out] VARIANT Max )

Query the possible line widths.

QueryLineWidth returns the minimal (Min) and maximal (Max) values of widths of region border which can be displayed. Setting of the border width is done with SetLineWidth. Border width is used by operators like DispRegion, DispLine, DispCircle, DispRectangle1, DispRectangle2 etc. if the drawing mode is ‘‘margin’’ ( SetDraw( WindowHandle,’margin’)).

Parameter

- Min (output control) integer ~ long / VARIANT
- Max (output control) integer ~ long / VARIANT

Displayable minimum width.

Displayable maximum width.
4.6. PARAMETERS

**QueryLineWidth** returns TRUE if the window is valid. Otherwise an exception handling is raised.

---

**Parallelization Information**

**QueryLineWidth** is reentrant and processed without parallelization.

---

**Possible Successors**

GetLineWidth, SetLineWidth, SetLineStyle, DispLine

---

**Possible Successors**

DispCircle, DispLine, DispRectangle1, DispRectangle2, DispRegion, SetLineWidth, GetLineWidth, SetLineStyle

---

**See also**

DispCircle, DispLine, DispRectangle1, DispRectangle2, DispRegion, SetLineWidth, GetLineWidth, SetLineStyle

---

**Module**

System

```c
[out] VARIANT Mode HWindowX.QueryPaint ( )

void HOperatorSetX.QueryPaint ([in] VARIANT WindowHandle,
[out] VARIANT Mode )
```

Query the grayvalue display modes.

**QueryPaint** returns the names of all grayvalue display modes (e.g. 'gray', '3D-plot', 'contourline', etc.) for the output window. These modes are used by **SetPaint**. **QueryPaint** only returns the names of the display values, not the additional parameters that may be necessary for some modes.

---

**Attention**

---

**Parameter**

- **WindowHandle** (input control) window \( \sim \) HWindowX / VARIANT Window\_id.
- **Mode** (output control) string \( \sim \) VARIANT ( string )

---

**Result**

**QueryPaint** returns TRUE if the window is valid. Otherwise an exception handling is raised.

---

**Parallelization Information**

**QueryPaint** is reentrant, local, and processed without parallelization.

---

**Possible Successors**

GetPaint, SetPaint, DispImage

---

**See also**

SetPaint, GetPaint, DispImage

---

**Module**

System

```c
[out] VARIANT DisplayShape HInfoX.QueryShape ( )

void HOperatorSetX.QueryShape ([out] VARIANT DisplayShape )
```

Query the region display modes.

**QueryShape** returns the names of all region display modes (e.g. 'original', 'circle', 'rectangle1', 'rectangle2', 'ellipse', etc.) for the window. They are used by **SetShape**.

---

**Attention**

---

**Module**

System
Parameter

- **DisplayShape** (output control) ............................................................ string  \(\sim\) VARIANT (string )

  region display mode names.

Result

**QueryShape** returns TRUE, if the window is valid. Otherwise an exception handling is raised.

Parallelization Information

**QueryShape** is reentrant and processed without parallelization.

Possible Successors

**GetShape**, **SetShape**, **DispRegion**

See also

**SetShape**, **GetShape**, **DispRegion**

Module

**System**

```
HWindowX.SetColor ([in] VARIANT Color )
HOperatorSetX.SetColor ([in] VARIANT WindowHandle, [in] VARIANT Color )
```

Set output color.

**SetColor** defines the colors for region output in the window. The available colors can be queried with the procedure **QueryColor**. The "colors" 'black' and 'white' are available for all screens. If colors are used that are not displayable on the screen, HALCON can choose a similar, displayable color of the output. For this, **SetCheck()** must be called. Furthermore, the list of available colors can be set with the procedure **SetSystem()**. That must be done before opening the first output window.

If only a single color is passed, all output is in this color. If a tuple of colors is passed, the output color of regions is modulo to the number of colors. In the example below, the first circle is displayed red, the second in green and the third in red again. HALCON always begins output with the first color passed. Note, that the number of output colors depends on the number of objects that are displayed in one procedure call. If only single objects are displayed, they always appear in the first color, even if the consist of more than one connected components.

The defined colors are used until **SetColor**, **SetPixel**, **SetRgb**, **SetHsi** or **SetGray** is called again.

Colors are defined separately for each window. They can only be changed for the valid window.

**Color** is used in procedures with region output like **DispRegion**, **DispLine**, **DispRectangle1**, **DispArrow** etc. It is also used by procedures with grayvalue output in certain output modes (e.g. '3D-plot','histogram', 'contourline', etc. See **SetPaint**).

Attention

```
Parameter

- **WindowHandle** (input control) ................................. window  \(\sim\) HWindowX / VARIANT WindowId.

- **Color** (input control) .................................................. string(-array)  \(\sim\) VARIANT (string )

  Output color names.

  Default Value: 'white'

Example

set_color(WindowHandle, ['red','green'])
disp_circle(WindowHandle, [100,200,300],[200,300,100],[100,100,100]).

Result

**SetColor** returns TRUE if the window is valid and the passed colors are displayable on the screen. Otherwise an exception handling is raised.

Parallelization Information

**SetColor** is reentrant, local, and processed without parallelization.
4.6. PARAMETERS

Possible Predecessors

QueryColor

Possible Successors

DispRegion

See also

GetRgb, DispRegion, SetFix, SetPaint

Alternatives

SetRgb, SetHsi

Module

System

void HWindowX.SetColored ([in] long NumberOfColors )

void HOperatorSetX.SetColored ([in] VARIANT WindowHandle, [in] VARIANT NumberOfColors )

Set multiple output colors.

SetColored is a shortcut for certain SetColor calls. It allows the user to display a region set in different colors. NumberOfColors defines the number of colors that are used. Valid values for NumberOfColors can be queried with QueryColored. Furthermore, the list of available colors can be set with the procedure SetSystem(::‘graphicColors’,…:). This must be done before opening the first output window.

Attention

Parameter

WindowHandle (input control) ……………. window ~ HWindowX / VARIANT Window_id.

NumberOfColors (input control) …………… integer ~ long / VARIANT Number of output colors.

Default Value: 12

List of values: NumberOfColors ∈ {3, 6, 12}

Result

SetColored returns TRUE if NumberOfColors is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetColored is reentrant, local, and processed without parallelization.

Possible Predecessors

QueryColored, SetColor

Possible Successors

DispRegion

See also

QueryColored, SetColor, DispRegion

Module

System

void HWindowX.SetComprise ([in] String Mode )

void HOperatorSetX.SetComprise ([in] VARIANT WindowHandle, [in] VARIANT Mode )

Define the image matrix output clipping.

SetComprise defines the image matrix output clipping. If Mode is set to ’object’, only grayvalues belonging to the output object are displayed. If set to ’image’, the whole image matrix is displayed. Default is ’object’.
Attention

If `Mode` was set to 'image', undefined grayvalues may be displayed. Depending on the context they are black or can have random content. See the examples.

**Parameter**

- **WindowHandle** (input control) ..........................  window \(\sim\) `HWindowX / VARIANT WindowId`
- **Mode** (input control) ........................................................... string \(\sim\) `String / VARIANT`

**Default Value**: 'object'

**List of values**: \(\text{Mode} \in \{\text{`image'}, \text{`object'}\}\)

**Example**

```c
open_window(0,0,-1,-1,'root','visible' ,"",WindowHandle)
read_image(Image,'fabrik')
threshold(Image,Seg,100,255)
set_system(‘init_new_image’,’false’)
sobel_amp(Image,Sob,’sum_abs’,3)
disp_image(Sob,WindowHandle)
get_comprise(Mode)
fwrite_string([‘Current mode for gray values: ’,Mode])
fnew_line()
set_comprise(WindowHandle,’image’)
get_mbutton(WindowHandle,_,_,_)
disp_image(Sob,WindowHandle)
fwrite_string([‘Current mode for gray values: image’])
fnew_line().
```

**Result**

`SetComprise` returns TRUE if `Mode` is correct and the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

`SetComprise` is reentrant and processed without parallelization.

**Possible Predecessors**

`GetComprise`

**Possible Successors**

`DispImage`

See also `GetComprise`, `DispImage`, `DispColor`

**Module**

`System`

```c
void HWindowX.SetDraw ([in] String Mode )
void HOperatorSetX.SetDraw ([in] VARIANT WindowHandle,
[in] VARIANT Mode )
```

Define the region fill mode.

`SetDraw` defines the region fill mode. If `Mode` is set to 'fill', output regions are filled, if set to 'margin', only contours are displayed. Setting `Mode` only affects the valid window. It is used by procedures with region output like `DispRegion`, `DispCircle`, `DispRectangle1`, `DispRectangle2`, `DispArrow` etc. It is also used by procedures with grayvalue output for some grayvalue output modes (e.g. 'histogram', see `SetPaint`). If the mode is 'margin', the contour can be affected with `SetLineWidth`, `SetLineApprox` and `SetLineStyle`

**Attention**

If the output mode is 'margin' and the line width is more than one, objects may not be displayed.
4.6. PARAMETERS

Parameter

- **WindowHandle** (input control) window \( \sim HWindowX / \text{VARIANT WindowId} \)
- **Mode** (input control) string \( \sim \text{String / VARIANT} \)

Fill mode for region output.

Default Value: ‘fill’
List of values: Mode \( \in \{\text{‘fill’}, \text{‘margin’}\} \)

Result

SetDraw returns TRUE if Mode is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetDraw is reentrant, local, and processed without parallelization.

Possible Predecessors

GetDraw, DispRegion

Possible Successors

DispImage, SetLineWt, SetLineStyle

See also

GetDraw, DispRegion, SetPaint, DispImage, SetLineWt, SetLineStyle

Module

System

```c
void HWindowX.SetFix ([in] String Mode )

void HOperatorSetX.SetFix ([in] VARIANT WindowHandle, [in] VARIANT Mode )
```

Set fixing of “look-up-table” (lut)

Behaviour for Mode = ’true’: SetFix fixes that pixel lastly ascertained by one of the operators SetGray, SetColor, SetHsi or SetRgb (Remark: Here a pixel is the index within the current look-up-table). To assign a new color to a fixed pixel set a color or gray value by using SetColor, SetRgb, SetHsi or SetGray. This makes it possible to define any color (SetColor), any gray value (SetGray) and any color combination (SetRgb, SetHsi) at any position of the look-up-table.

Mode set to ‘false’ reset the fixing. To modify or create a look-up-table process SetPixel, SetFix (::WindowHandle,’true’), SetRgb and SetFix (::WindowHandle,’false’) one after another.

Attention

As a side effect SetFix can change colors of “non-HALCON windows”.

Parameter

- **WindowHandle** (input control) window \( \sim HWindowX / \text{VARIANT WindowId} \)
- **Mode** (input control) string \( \sim \text{String / VARIANT} \)

Mode of fixing.

Default Value: ‘true’
List of values: Mode \( \in \{\text{‘true’}, \text{‘false’}\} \)

Result

SetFix returns TRUE if the window is valid, the hardware supports a look-up-table and all parameters are correct. Otherwise an exception handling is raised.

Parallelization Information

SetFix is reentrant, local, and processed without parallelization.

Possible Predecessors

GetFix

Possible Successors

SetPixel, SetRgb
Define grayvalues for region output.

*SetGray* defines the grayvalues for region output. Grayvalues are defined as the range of the color lookup table that is used for grayvalue output with *DispImage* in conjunction with *SetPaint* (::WindowHandle,'gray'). These entries can be modified by *SetLut*. So a 'grayvalue' is the color in which a pixel with the same value is displayed (not necessarily really gray). In general, when changing the color lookup table with *SetLut*, the colors of the displayed image will change too.

If a grayvalue is needed as a color for image output (i.e. no color changes with *SetLut* are possible), it can be set with *SetColor* (::WindowHandle,'gray').

If only a single grayvalue is passed, all output will take place in that grayvalue. If a tuple of grayvalues is passed, all output will take place in grayvalues modulo the number of tuple elements. In the example below, the first circle is displayed with grayvalue 100, the second with 200 and the third with 100 again. Every output procedure starts with the first grayvalue. Note, that the number of output grayvalues depends on the number of objects that are displayed in one procedure call. If only single objects are displayed, they always appear in the first grayvalue, even if the consist of more than one connected components.

When the procedures *SetGray*, *SetColor*, *SetRgb*, *SetHsi* are called, the overwrite the existing values. If not all grayvalues are displayable on the output device, the number range of *GrayValues* (0..255) is dithered to the range of displayable grayvalues. In any case 0 is displayed as black and 255 as white. The displayable grayvalues can be queried with the procedure *QueryGray*. Furthermore, the number of actually displayed grayvalues can be changed with *SetSystem* (::'numGray*',...:). This must be done before opening the first window. With *SetCheck* (::'˜color':) error messages can be suppressed if a grayvalue can’t be displayed on the screen. In that case, a similar grayvalue is displayed.

### Attention

#### Parameter

- **WindowHandle** (input control) .......................... window  ~ HWindowX / VARIANT Window.id.
- **GrayValues** (input control) ......................... integer(-array)  ~ VARIANT( integer )
  Grayvalues for region output.
  **Default Value** : 255
  **Suggested values** : GrayValues ∈ {0, 1, 2, 10, 16, 32, 64, 100, 120, 128, 250, 251, 252, 253, 254, 255}
  **Typical range of values** : 0 ≤ GrayValues ≤ 0

#### Example

```plaintext
set_gray(WindowHandle, [100, 200])
disp_circle(WindowHandle, [100, 200, 300], [200, 300, 100], [100, 100, 100]).
```

### Parallelization Information

*SetGray* is reentrant, local, and processed without parallelization.

### Possible Successors

*DispRegion*  

### See also

*GetPixel, SetColor*  

### Module

System

HALCON/COM Reference Manual, 2005-2-1
Define output colors (HSI-coded).

SetHsi sets the region output color(s)/grayvalue(s) for the valid window. Colors are passed as Hue, Saturation, and Intensity. Transformation from HSI to RGB is done with:

\[
\begin{align*}
H &= \frac{2\pi \text{Hue}}{255} \\
I &= \frac{\sqrt{6} \text{Intensity}}{255} \\
M_1 &= \frac{\sin(H) \text{Saturation}}{255\sqrt{6}} \\
M_2 &= \frac{\cos(H) \text{Saturation}}{255\sqrt{2}} \\
R &= \frac{2M_1 + I}{4\sqrt{6}} \\
G &= \frac{-M_1 + M_2 + I}{4\sqrt{6}} \\
B &= \frac{-M_1 - M_2 + I}{4\sqrt{6}} \\
\end{align*}
\]

Red = \(R \times 255\)
Green = \(G \times 255\)
Blue = \(B \times 255\)

If only one combination is passed, all output will take place in that color. If a tuple of colors is passed, the output color of regions and geometric objects is modulo to the number of colors. HALCON always begins output with the first color passed. Note, that the number of output colors depends on the number of objects that are displayed in one procedure call. If only single objects are displayed, they always appear in the first color, even if the consist of more than one connected components.

Selected colors are used until the next call of SetColor, SetPixel, SetRgb or SetGray. Colors are relevant to windows, i.e. only the colors of the valid window can be set. Region output colors are used by operators like DispRegion, DispLine, DispRectangle1, DispRectangle2, DispArrow, etc. It is also used by procedures with grayvalue output in certain output modes (e.g. ’3D-plot’, ’histogram’, ’contourline’, etc. See SetPaint).

Attention

The selected intensities may not be available for the selected hues. In that case, the intensities will be lowered automatically.

Parameter

- WindowHandle (input control) ............................. window \(\sim\) HWindowX / VARIANT WindowId.
- Hue (input control) .......................... integer(-array) \(\sim\) VARIANT( integer )
  Hue for region output.
  Default Value : 30
  Typical range of values : \(0 \leq \text{Hue} \leq 0\)
  Restriction : \((0 \leq \text{Hue}) \land (\text{Hue} \leq 255)\)
- Saturation (input control) ........................... integer(-array) \(\sim\) VARIANT( integer )
  Saturation for region output.
  Default Value : 255
  Typical range of values : \(0 \leq \text{Saturation} \leq 0\)
  Restriction : \((0 \leq \text{Saturation}) \land (\text{Saturation} \leq 255)\)
- Intensity (input control) ............................. integer(-array) \(\sim\) VARIANT( integer )
  Intensity for region output.
  Default Value : 84
  Typical range of values : \(0 \leq \text{Intensity} \leq 0\)
  Restriction : \((0 \leq \text{Intensity}) \land (\text{Intensity} \leq 255)\)

Result

SetHsi returns TRUE if the window is valid and the output colors are displayable. Otherwise an exception handling is raised.

HALCON 6.1.4
Parallelization Information

SetHsi is reentrant, local, and processed without parallelization.

Possible Predecessors

GetHsi

Possible Successors

DispRegion

See also

GetHsi, GetPixel, TransFromRgb, TransToRgb, DispRegion

Module

System

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRegionX.SetIcon</td>
<td>[in] HWindowX.WindowHandle</td>
</tr>
<tr>
<td>HWindowX.SetIcon</td>
<td>[in] HRegionX.Icon</td>
</tr>
<tr>
<td>HOperatorSetX.SetIcon</td>
<td>[in] IHObjectX.Icon, [in] VARIANT WindowHandle</td>
</tr>
</tbody>
</table>

Icon definition for region output.

SetIcon defines an icon for region output (DispRegion). It is displayed in the regions center of gravity. The use of this icon is activated with SetShape.

Parameter

- **Icon** (input iconic)                      
  - region ∼ HRegionX / IHObjectX Icon for center of gravity.
- **WindowHandle** (input control)           
  - window ∼ HWindowX / VARIANT Window id.

Parallelization Information

SetIcon is reentrant and processed without parallelization.

Possible Predecessors

GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion

Possible Successors

SetShape, DispRegion

Module

System

<table>
<thead>
<tr>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>HWindowX.SetInsert</td>
<td>[in] String Mode</td>
</tr>
<tr>
<td>HOperatorSetX.SetInsert</td>
<td>[in] VARIANT WindowHandle, [in] VARIANT Mode</td>
</tr>
</tbody>
</table>

Define the pixel output function.

SetInsert defines the function, with which pixels are displayed in the output window. It is e.g. possible for a pixel to overwrite the old value. In most of the cases there is a functional relationship between old and new values. The definition value is only valid for the valid window. Output procedures that honor Mode are e.g. DispRegion, DispPolygon, DispCircle.

Possible display functions are:

- `'copy'`: overwrite displayed pixels
- `'xor'`: display old "xor" new pixels
- `'complement'`: complement displayed pixels
There may not be all functions available, depending on the physical display. However, "copy" is always available.

Attention

Parameter

- **WindowHandle** (input control)  
  
  *window ~ HWindowX / VARIANT WindowId.* 

- **Mode** (input control)  
  
  *Name of the display function.*
  
  Default Value: 'copy'
  
  List of values: Mode ∈ {'copy', 'xor', 'complement'}

Result

*SetInsert* returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

*SetInsert* is reentrant, local, and processed without parallelization.

Possible Predecessors

QueryInsert, GetInsert

Possible Successors

DispRegion

See also

GetInsert, QueryInsert

Module

System

```c
void HWindowX.SetLineApprox ([in] long Approximation )

void HOperatorSetX.SetLineApprox ([in] VARIANT WindowHandle, [in] VARIANT Approximation )
```

Define the approximation error for contour display.

*SetLineApprox* defines the approximation error for region contour display in the window. *Approximation* values greater than zero cause a polygon approximation ≈ smoothing (with a maximum polygon/contour deviation of *Approximation* pixel). The approximation algorithm is the same as in *GetRegionPolygon*. 

*SetLineApprox* is important for contour output via *SetLineStyle*.

Attention

Parameter

- **WindowHandle** (input control)  
  
  *window ~ HWindowX / VARIANT WindowId.* 

- **Approximation** (input control)  
  
  *Maximum deviation from the original contour.*
  
  Default Value: 0
  
  Restriction: (Approximation ≥ 0)

Example

```c
/* Calling */
set_line_approx(WindowHandle, Approximation)
set_draw(WindowHandle, 'margin')
disp_region(Obj, WindowHandle).

/* correspond with */
get_region_polygon(Obj, Approximation, Row, Col)
disp_polygon(WindowHandle, Row, Col).
```

HALCON 6.1.4
SetLineApprox returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

SetLineApprox is reentrant and processed without parallelization.

Possible Predecessors

GetLineApprox

Possible Successors

DispRegion

See also

GetLineApprox, SetLineStyle, SetDraw, DispRegion

Alternatives

GetRegionPolygon, DispPolygon

Module

System

#define a contour output pattern.

SetLineStyle defines the output pattern of region contours. The information is used by procedures like DispRegion, DispLine, DispPolygon etc. The current value can be queried with GetLineStyle. Style contains up to five pairs of values. The first value is the length of the visible contour part, the second is the length of the invisible part. The value pairs are used cyclically for contour output.

Attention

SetLineStyle does an implicit polygon approximation (see SetLineApprox(::WindowHandle,3:)). It is only possible to enlarge it with SetLineApprox.

Parameter

- WindowHandle (input control) ..................................... window ~ HWindowX / VARIANT Window
- Style (input control) .............................................. integer ~ VARIANT( integer ) Contour pattern.

Default Value : []

Example

* stroke line: X-Windows
  set_line_style( WindowHandle, [20,7] )
* point-stroke line: X-Windows
  set_line_style( WindowHandle, [20,7,3,7] )
* passing line (standard)
  set_line_style( WindowHandle, [] )

Result

SetLineStyle returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetLineStyle is reentrant, local, and processed without parallelization.

Possible Predecessors

GetLineStyle
4.6. PARAMETERS

Possible Successors

DispRegion

See also

GetLineStyle, SetLineApprox, DispRegion

Module

System

---

void HWindowX.SetLineWidth ([in] long Width )

void HOperatorSetX.SetLineWidth ([in] VARIANT WindowHandle, [in] VARIANT Width )

Define the line width for region contour output.

SetLineWidth defines the line width (in pixel) in which a region contour is displayed (e.g. with DispRegion, DispLine, DispPolygon, etc.). The procedure GetLineWidth returns the current value for the window. Some output devices may not allow to change the contour width. If it is possible for the current device, it can be queried with QueryLineWidth.

Attention

The line width is important if the output mode was set to ‘margin’ (see SetDraw). If the line width is greater than one, regions may not always be displayed correctly.

Parameter

▷ WindowHandle (input control) .............................................. window ~ HWindow / VARIANT WindowId.
▷ Width (input control) ........................................................... integer ~ long / VARIANT
Line width for region output in contour mode.
Default Value : 1
Restriction : ((Width ≥ 1) ∧ (Width ≤ 2000))

Result

SetLineWidth returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information

SetLineWidth is reentrant and processed without parallelization.

Possible Predecessors

QueryLineWidth, GetLineWidth

Possible Successors

DispRegion

See also

GetLineWidth, QueryLineWidth, SetDraw, DispRegion

Module

System

---

void HWindowX.SetPaint ([in] VARIANT Mode )

void HOperatorSetX.SetPaint ([in] VARIANT WindowHandle, [in] VARIANT Mode )

Define the grayvalue output mode.

SetPaint defines the output mode for grayvalue display (single- or multichannel) in the window. The mode is used by DispImage.

This page describes the different modes, that can be used for grayvalue output. It should be noted, that the mode ‘default’ is the most suitable.
The hardware characteristics determine how grayvalues can be displayed. On a screen with one to seven bit planes, only binary data can be displayed. On screens with at least eight bit planes, it is possible to display multiple grayvalues. For binary displays HALCON includes algorithms with dithering matrix (fast, but low resolution), minimal error (good, but slow) and thresholding. Using the thresholding algorithm, the threshold can be passed as a second parameter (a tuple with the string 'threshold' and the actual threshold, e.g.: ['threshold', 100]). Displays with eight bit planes use approximately 200 grayvalues for output. Of course it is still possible to use a binary display on those displays.

A different way to display grayvalues is the histogram (mode: 'histogram'). This mode has two additional parameter values: Row (second value) and column (third value). They denote row and column of the histogram center for positioning on the screen. The scale factor (fourth value) determines the histogram size: a scale factor of 1 distinguishes 256 grayvalues, 2 distinguishes 128 grayvalues, and so on. The four values are passed as a tuple, e.g. ['histogram', 256,256,1]. If only the first value is passed ('histogram'), the other values are set to defaults or last values, respectively. For histogram computation see GrayHisto. Histogram output honors the same parameters as procedures like DispRegion etc. (e.g. SetColor, SetDraw, etc.)

Yet another mode is the display of relative frequencies of the number of connection components ("component_histogram"). For informations on computing the component histogram see ShapeHistoAll. Positioning and resolution are exactly as in the mode 'histogram'.

In mode 'mean', all object regions are displayed in their mean grayvalue.

The modes 'row' and 'column' allow the display of lines or columns, respectively. The position (line- and columnindex) is passed with the second parameter value. The third parameter value is the scale factor in percent (100 means 1 pixel per grayvalue, 50 means one pixel per two grayvalues). Gray images can also be interpreted as 3d data, depending on the grayvalue. To view these 3d plots, select the modes 'contourline', '3D-plot' or '3D-plot_hidden'.

Two-channel images can be viewed as 'vectorfield'. Three-channel images are interpreted as RGB images. They can be displayed in three different modes. Two of them can be optimized by Floyd-Steinberg dithering.

Parameters for modes that need more than one parameter can be passed the following ways:

- Only the name of the mode is passed: the defaults or the last values are used, respectively. Example: SetPaint(::WindowHandle,'contourline':)
- All values are passed: all output characteristics can be set. Example: SetPaint (::WindowHandle,['contourline',10,1]):
- Only the first n values are passed: only the passed values are changed. Example: SetPaint (::WindowHandle,['contourline',10]):
- Some of the values are replaced by an asterisk ('*'): The value of the replaced parameters is not changed. Example: SetPaint (::WindowHandle,['contourline','*',1]):

If the current mode is 'default', HALCON chooses a suitable algorithm for the output of 2- and 3-channel images. No SetPaint call is necessary in this case.

Apart from SetPaint there are other procedures that affect the output of grayvalues. The most important of them are SetPart, SetPartStyle, SetLut and SetLutStyle. Some output modes display grayvalues using region output (e.g. 'histogram','contourline','3D-plot', etc.). In these modes, parameters set with SetColor, SetRgb, SetHsi, SetPixel, SetShape, SetLineWidth and SetInsert influence grayvalue output. This can lead to unexpected results when using SetShape(::{:convex}:) and SetPaint (::{:WindowHandle,'histogram'}). Here the convex hull of the histogram is displayed.

Modes:

- 'default' optimal display on given hardware
- 'gray' grayvalue output
- 'mean' mean grayvalue
- 'dither4_1' binary image, dithering matrix 4x4
- 'dither4_2' binary image, dithering image 4x4
- 'dither4_3' binary image, dithering matrix 4x4

HALCON/COM Reference Manual, 2005-2-1
4.6. PARAMETERS

'dither8_1'  binary image, dithering matrix 8x8
'floyd_steinberg'  binary image, optimal grayvalue simulation

['threshold',Threshold ]
  'threshold'  binary image, threshold: 128 (default)
  ['threshold',200 ] binary image, any threshold: (here: 200)

['histogram',Line,Column,Scale ]
  'histogram'  grayvalue output as histogram.
  position default: max. size, in the window center
  ['histogram',256,256,2 ] grayvalue output as histogram, any parameter values.
     positioning: window center (here (256,256))
     size: (here 2, half the max. size)

['component_histogram',Line,Column,Scale ]
  'component_histogram'  output as histogram of the connection components.
  Positioning: default
  ['component_histogram',256,256,1 ] output as histogram of the connection components.
     Positioning: (here (256, 256))
     Scaling: (here 1, max. size)

['row',Line,Scale ]
  'row'  output of the grayvalue profile along the given line.
  line: image center (default)
  Scaling: 50
  ['row',100,20 ] output of the grayvalue profile of line 100 with a scaling of 0.2 (20

['column',Column,Scale ]
  'column'  output of the grayvalue profile along the given column.
  column: image center (default)
  Scaling: 50
  ['column',100,20 ] output of the grayvalue profile of column 100 with a scaling of 0.2 (20

['contourline',Step,Colored ]
  'contourline'  grayvalue output as contour lines: the grayvalue difference per line is defined with
  the parameter 'Step' (default: 30, i.e. max. 8 lines for 256 grayvalues). The line can be displayed
  in a given color (see set_color) or in the grayvalue they represent. This behaviour is defined with
  the parameter 'Colored' (0 = color, 1 = grayvalues). Default is color.
  ['contourline',15,1 ] grayvalue output as contour lines with a step of 15 and gray output.

['3D-plot', Step, Colored, EyeHeight, EyeDistance, ScaleGray, LinePos, ColumnPos]
  '3D-plot'  grayvalues are interpreted as 3d data: the greater the value, the 'higher' the assumed
  mountain. Lines with step 2 (second parameter value) are drawn along the x- and y-axes. The third
  parameter (Colored) determines, if the output should be in color (default) or grayvalues. To define
  the projection of the 3d data, use the parameters EyeHeight and EyeDistance. The projection parameters
  take values from 0 to 255. ScaleGray defines a factor, by which the grayvalues are multiplied for
  'height' interpretation (given in percent. 100EyeHeight and EyeDistance the image can be shifted
  out of place. Use RowPos and ColumnPos to move the whole output. Values from -127 to 127 are
  possible.
  ['3D-plot', 5, 1, 110, 160, 150, 70, -10 ] line step: 5 pixel
  Colored: yes (1)
  EyeHeight: 110
  EyeDistance: 160
  ScaleGray: 1.5 (150)
  RowPos: 70 pixel down
  ColumnPos: 10 pixel right

['3D-plot_hidden', Step, Colored, EyeHeight, EyeDistance, ScaleGray, LinePos, ColumnPos]
  '3D-plot_hidden'  like '3D-plot', but computes hidden lines.

• Two-channel images:
  'default'  output as vector field.
  ['vectorfield', Step, MinLength, ScaleLength ]
'vectorfield’ output of the two channels as displacement vector field. Procedures like optical_flow* produce images of pixel displacements (first channel: x-displacement, second channel: y-displacement). In this mode, an arrow is drawn for each vector. The parameter Step has to be set in correspondence to optical_flow*. Short vectors can be suppressed by the third parameter value (MinLength). The fourth parameter value scales the vector length. Vectors, that exceed the window boundaries are nor drawn.

['vectorfield',16,2,3 ] Output of every 16. vector, that is longer than 2 pixel. Each vector is multiplied by 3 for output.

- Three-channel images:
  
  'default’ output as RGB image with 'median_cut'.
  
  'television' color addition algorithm for RGB images: (three components necessary for DispImage). Images are displayed via a fixed color lookup table. Fast, but non-optimal color resolution. Only recommended on bright screens.
  
  'grid_scan’ grid-scan algorithm for RGB images (three components necessary for DispImage). An optimized color lookup table is generated for each image. Slower than 'television'. Disadvantages: Hard color boundaries (no dithering). Different color lookup table for every image.
  
  'grid_scan_floyd_steinberg’ grid-scan with Floyd-Steinberg dithering for smooth color boundaries.
  
  
  'median_cut_floyd_steinberg’ median-cut algorithm with Floyd-Steinberg dithering for smooth color boundaries.

Attention

- Display of color images ('television’, ‘grid_scan', etc.) changes the color lookup tables.
  
  - If a wrong color mode is set, the error message may appear not until the DispImage call.
  
  - Grayvalue output may be influenced by region output parameters. This can yield unexpected results.

Parameter

<table>
<thead>
<tr>
<th>WindowHandle (input control)</th>
<th>window</th>
<th>HWindowX / VARIANT Window.id.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode (input control)</td>
<td>string</td>
<td>VARIANT ( integer, string )</td>
</tr>
<tr>
<td>Default Value: 'default'</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| List of values: Mode ∈ {'default', 'histogram', 'row', 'column', 'contourline', '3D-plot', '3D-plot_hidden', '3D-plot_point' }

Example

read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','.','WindowHandle)
query_paint(WindowHandleModi)
fwrite_string([‘available gray value modes: ',Modi])
fnew_line()
disp_image(Image,WindowHandle)
get_mbutton(WindowHandle,_,_,_) set_color(WindowHandle,'red')
set_draw(WindowHandle,'margin')
set_paint(WindowHandle,'histogram')
disp_image(Image,WindowHandle)
set_color(WindowHandle,'blue')
set_paint(WindowHandle,[‘histogram’,'100,100,3'])
disp_image(Image,WindowHandle)
set_color(WindowHandle,'yellow')

HALCON/COM Reference Manual, 2005-2-1
set_paint(WindowHandle, ['row', 100])
disp_image(Image, WindowHandle)
get_mbutton(WindowHandle, _, _, _)
clear_window(WindowHandle)
set_paint(WindowHandle, ['contourline', 10, 1])
disp_image(Image, WindowHandle)
set_lut(WindowHandle, 'color')
get_mbutton(WindowHandle, _, _, _)
clear_window(WindowHandle)
set_part(WindowHandle, 100, 100, 300, 300)
set_paint(WindowHandle, '3D-plot')
disp_image(Image, WindowHandle).

Result
SetPaint returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

Parallelization Information
SetPaint is reentrant, local, and processed without parallelization.

Possible Predecessors
QueryPaint, GetPaint

Possible Successors
DispImage

See also
GetPaint, QueryPaint, DispImage, SetShape, SetRgb, SetColor, SetGray

Module


Modify the displayed image part.
SetPart modifies the image part that is displayed in the window. (Row1, Column1) denotes the upper left corner and (Row2, Column2) the lower right corner of the image part to display. The changed values are used by grayvalue output operators (DispImage, DispColor) as well as region output operators (DispRegion).

If only part of an image is displayed, it will be zoomed to full window size. The zooming interpolation method can be set with SetPartStyle. GetPart returns the values of the image part to display.

Beside setting the image part directly, the following special modes are supported:

Row1 = Column1 = Row2 = Column2 = -1: The window size is choosen as the image part, i.e. no zooming of the image will be performed.

Row1, Column1 > -1 and Row2 = Column2 = -1: The size of the last displayed image (in this window) is choosen as the image part, i.e. the image can completely be displayed in the image. For this the image will be zoomed if necessary.

Attention
After a call of SetPart, some operators like DrawRegion, DrawEllipse, etc. can no longer be used.

Parameter

> WindowHandle (input control) ........................................... window ~ HWindowX / VARIANT Window_id.
CHAPTER 4. GRAPHICS

- **Row1** (input control) .......................... rectangle.origin.y \( \sim \) long / VARIANT
  Row of the upper left corner of the chosen image part.
  
  **Default Value:** 0

- **Column1** (input control) .......................... rectangle.origin.x \( \sim \) long / VARIANT
  Column of the upper left corner of the chosen image part.
  
  **Default Value:** 0

- **Row2** (input control) .......................... rectangle.corner.y \( \sim \) long / VARIANT
  Row of the lower right corner of the chosen image part.
  
  **Default Value:** -1
  
  **Restriction:** \((Row2 \geq Row1) \lor (Row2 = -1)\)

- **Column2** (input control) .......................... rectangle.corner.x \( \sim \) long / VARIANT
  Column of the lower right corner of the chosen image part.
  
  **Default Value:** -1
  
  **Restriction:** \((Column2 \geq Column1) \lor (Column2 = -1)\)

---

```plaintext
get_system('width', Width)
get_system('height', Height)
set_part(WindowHandle, 0, 0, Height-1, Width-1)
disp_image(Image, WindowHandle)
draw_rectangle1(WindowHandle: Row1, Column1, Row2, Column2)
set_part(WindowHandle, Row1, Column1, Row2, Column2)
disp_image(Image, WindowHandle).
```

---

**Example**

**Result**

*SetPart* returns TRUE if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

*SetPart* is reentrant and processed without parallelization.

**Possible Predecessors**

*GetPart*

**Possible Successors**

*SetPartStyle, DispImage, DispRegion*

**See also**

*GetPart, SetPartStyle, DispRegion, DispImage, DispColor*

**Alternatives**

*AffineTransImage*

**Module**

**System**

```plaintext
void HWindowX.SetPartStyle ([in] long Style )
void HOperatorSetX.SetPartStyle ([in] VARIANT WindowHandle,
[in] VARIANT Style )
```

Define an interpolation method for grayvalue output.

*SetPartStyle* defines the interpolation method to zoom an image part which is displayed in the window.
Interpolation takes place, if the output window has different size than the image to display (e.g. after a call to *SetPart* or a window resize). Three modes are supported:

- **0** no interpolation (low quality, very fast).
- **1** unweighted interpolation (medium quality and run time)
- **2** weighted interpolation (high quality, slow)
The current value can be queried with \texttt{GetPartStyle}.

\begin{verbatim}
Attention Parameter
\end{verbatim}

\begin{itemize}
\item \texttt{WindowHandle} (input control) \hspace*{1cm} window \sim HWindowX / VARIANT WindowId.
\item \texttt{Style} (input control) \hspace*{1cm} \texttt{Style} \in \{0, 1, 2\}
\end{itemize}

\begin{verbatim}
Result
\end{verbatim}

\texttt{SetPartStyle} returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

\begin{verbatim}
Parallelization Information
\end{verbatim}

\texttt{SetPartStyle} is reentrant and processed without parallelization.

\begin{verbatim}
Possible Predecessors
\end{verbatim}

\texttt{GetPartStyle}, \texttt{SetPart}, \texttt{DispImage}, \texttt{DispRegion}

\begin{verbatim}
Possible Successors
\end{verbatim}

\texttt{SetPartStyle}, \texttt{GetPartStyle}, \texttt{DispImage}, \texttt{DispRegion}

\begin{verbatim}
See also
\end{verbatim}

\texttt{SetPart}, \texttt{DispImage}, \texttt{DispColor}

\begin{verbatim}
Alternatives
\end{verbatim}

\texttt{AffineTransImage}

\begin{verbatim}
Module
\end{verbatim}

\texttt{System}

\begin{verbatim}
void HWindowX.SetPixel ([in] VARIANT Pixel )
void HOperatorSetX.SetPixel ([in] VARIANT WindowHandle, [in] VARIANT Pixel )
\end{verbatim}

\begin{verbatim}
Define a color lookup table index.
\end{verbatim}

\texttt{SetPixel} sets pixel values: colors (\texttt{SetColor}, \texttt{SetRgb}, etc.) and grayvalues (\texttt{SetGray}) are coded together into a number, called pixel. This 'pixel' is an index in the color lookup table. It ranges from 0 to 1 in b/w images and 0 to 255 color images with 8 bit planes. It is different from the 'pixel' ('picture element') in image processing. Therefore HALCON distinguishes between pixel and image element (or grayvalue).

The current value can be queried with \texttt{GetPixel}.

\begin{verbatim}
Attention Parameter
\end{verbatim}

\begin{itemize}
\item \texttt{WindowHandle} (input control) \hspace*{1cm} window \sim HWindowX / VARIANT WindowId.
\item \texttt{Pixel} (input control) \hspace*{1cm} \texttt{Pixel} \in \{0, 1, 2\}
\end{itemize}

\begin{verbatim}
Result
\end{verbatim}

\texttt{SetPixel} returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

\begin{verbatim}
Parallelization Information
\end{verbatim}

\texttt{SetPixel} is reentrant, local, and processed without parallelization.
Set the color definition via RGB values.

SetRgb sets the output color(s) or the grayvalues, respectively, for region output for the window. The colors are defined with the red, green and blue components. If only one combination is passed, all output takes place in that color. If a tuple is passed, region output and output of geometric objects takes place modulo the passed colors.

For every call of an output procedure, output is started with the first color. If only one object is displayed per call, it will always be displayed in the first color. This is even true for objects with multiple connection components. If multiple objects are displayed per procedure call, multiple colors are used. The defined colors are used until SetColor, SetPixel, SetRgb or SetGray is called again. The values are used by procedures like DispRegion, DispLine, DispRectangle1, DispRectangle2, DispArrow, etc.

Attention
If a passed is not available, an exception handling is raised. If SetCheck(:,:,"color") was called before, HALCON uses a similar color and suppresses the error.

Parameter

- **WindowHandle** (input control) 
  - (input control) window ~ HWindowX / VARIANT Window id.

- **Red** (input control) 
  - (input control) integer(-array) ~ VARIANT( integer ) Red component of the color.
  - Default Value : 255
  - Typical range of values : 0 ≤ Red ≤ 0
  - Restriction : ((0 ≤ Red) ∧ (Red ≤ 255))

- **Green** (input control) 
  - (input control) integer(-array) ~ VARIANT( integer ) Green component of the color.
  - Default Value : 0
  - Typical range of values : 0 ≤ Green ≤ 0
  - Restriction : ((0 ≤ Green) ∧ (Green ≤ 255))

- **Blue** (input control) 
  - (input control) integer(-array) ~ VARIANT( integer ) Blue component of the color.
  - Default Value : 0
  - Typical range of values : 0 ≤ Blue ≤ 0
  - Restriction : ((0 ≤ Blue) ∧ (Blue ≤ 255))

Result
SetRgb returns TRUE if the window is valid and all passed colors are available and displayable. Otherwise an exception handling is raised.

Parallelization Information
SetRgb is reentrant, local, and processed without parallelization.
4.6. PARAMETERS

SetFix, DispRegion

SetHsi, SetColor, SetGray

Module System

void HWindowX.SetShape ( [in] String Shape )

void HOperatorSetX.SetShape ( [in] VARIANT WindowHandle, [in] VARIANT Shape )

Define the region output shape.

SetShape defines the shape for region output. It is only valid for the window with the logical window number WindowHandle. The output shape is used by DispRegion. The available shapes can be queried with QueryShape.

Available modes:

‘original’: The shape is displayed unchanged. Nevertheless modifications via parameters like set_line_width or set_line_approx can take place. This is also true for all other modes.

‘outer_circle’: Each region is displayed by the smallest surrounding circle. (See SmallestCircle.)

‘inner_circle’: Each region is displayed by the largest included circle. (See InnerCircle.)

‘ellipse’: Each region is displayed by an ellipse with the same moments and orientation. (See EllipticAxis.)

‘rectangle1’: Each region is displayed by the smallest surrounding rectangle parallel to the coordinate axes. (See SmallestRectangle1.)

‘rectangle2’: Each region is displayed by the smallest surrounding rectangle. (See SmallestRectangle2.)

‘convex’: Each region is displayed by its convex hull. (See Convexity.)

‘icon’ Each region is displayed by the icon set with SetIcon in the center of gravity.

Attention

Caution is advised for grayvalue output procedures with output parameter settings that use region output, e.g., DispImage with SetPaint (::WindowHandle, ‘histogram’:) and SetShape (::WindowHandle, ‘convex’:). In that case the convex hull of the grayvalue histogram is displayed.

Parameter

> WindowHandle (input control) ........................................... window ~ HWindowX / VARIANT Window_id.

> Shape (input control) ......................................................... string ~ String / VARIANT Region output mode.

Default Value: ‘original’


Example

read_image (Image, ‘fabrik’)
regiongrowing (Image, Seg, 5, 5, 6, 100)
set_colored (WindowHandle, 12)
set_shape (WindowHandle, ‘rectangle2’)
disp_region (Seg, WindowHandle).

Result

SetShape returns TRUE if the parameter is correct and the window is valid. Otherwise an exception handling is raised.

HALCON 6.1.4
SetShape is reentrant and processed without parallelization.

Possible Predecessors
SetIcon, QueryShape, GetShape

Possible Successors
DispRegion

See also
GetShape, QueryShape, DispRegion

Module
System

4.7 Text

Get the current font.

GetFont queries the name of the font used in the output window. The font is used by the operators WriteString, ReadString etc. The font is set by the operator SetFont. Text windows as well as windows for image display use fonts. Both types of windows have a default font that can be modified with SetSystem (‘defaultFont’, Fontname) prior to opening the window. A list of all available fonts can be obtained using QueryFont.

Parameter

\[ \text{WindowHandle} \text{ (input control) \quad \text{window} \; \sim \; \text{HWindowX / VARIANT Window identifier.}} \]

\[ \text{Font} \text{ (output control) \quad \text{string} \; \sim \; \text{String / VARIANT Name of the current font.}} \]

Example

get_font(WindowHandle, CurrentFont)
sset_font(WindowHandle, MyFont)
write_string(WindowHandle, ['The name of my Font is:', Myfont])
nnew_line(WindowHandle)
sset_font(WindowHandle, CurrentFont)

Result

GetFont returns TRUE.
Get the spatial size of a string.

GetStringExtents queries width and height of the output size of a string using the font of the window. In addition the extension above and below the current baseline for writing is returned (Ascent bzw. Descent).

The sizes are measured in the coordinate system of the window (for text windows in pixels). Using GetStringExtents it is possible to determine text output and input independently from the used font. The conversion from integer numbers and floating point numbers to text strings is the same as in WriteString.

**Attention**

- **WindowHandle** (input control) – window \( \sim \) HWindowX / VARIANT Window identifier.
- **Values** (input control) – string(-array) \( \sim \) VARIANT( string, real, integer ) Values to consider.
- **Ascent** (output control) – integer \( \sim \) long / VARIANT Maximum height above baseline for writing.
- **Descent** (output control) – integer \( \sim \) long / VARIANT Maximum extension below baseline for writing.
- **Width** (output control) – integer \( \sim \) long / VARIANT Text width.
- **Height** (output control) – integer \( \sim \) long / VARIANT Text height.

**Result**

GetStringExtents returns TRUE if the window is valid. Otherwise an exception handling is raised.

**Parallelization Information**

GetStringExtents is reentrant, local, and processed without parallelization.

**Possible Predecessors**

OpenWindow, OpenTextwindow, SetFont

**Possible Successors**

SetTposition, WriteString, ReadString, ReadChar

**See also**

SetTposition, SetFont

**Module**

System

Get cursor position.

GetTposition queries the current position of the text cursor in the output window. The position is measured in the coordinate system of the window (in pixels for text windows). The next output of text in this window starts at the cursor position. The left end of the baseline for writing the next string (not considering descenders) is placed on this position. The position is changed by the output or input of text (WriteString, ReadString) or by an explicit change of position by (SetTposition, NewLine).
If the output text does not fit completely into the window, an exception handling is raised. This can be avoided by \texttt{SetCheck(’\textbackslash n text’)}.

\begin{itemize}
  \item \textbf{WindowHandle} (input control) ........................................... window $\rightsquigarrow$ \texttt{HWindowX / VARIANT}
    Window identifier.
  \item \textbf{Row} (output control) .......................................................... point.y $\rightsquigarrow$ \texttt{long / VARIANT}
    Row index of text cursor position.
  \item \textbf{Column} (output control) ...................................................... point.x $\rightsquigarrow$ \texttt{long / VARIANT}
    Column index of text cursor position.
\end{itemize}

\texttt{GetTposition} returns TRUE if the window is valid. Otherwise an exception handling is raised.

\begin{itemize}
  \item \textbf{Result}
\end{itemize}

\texttt{GetTposition} is \textit{reentrant, local}, and processed \textit{without} parallelization.

\begin{itemize}
  \item \textbf{Possible Predecessors} \texttt{OpenWindow, OpenTextwindow, SetFont}
  \item \textbf{Possible Successors} \texttt{SetTposition, WriteString, ReadString, ReadChar}
\end{itemize}

\texttt{See also} \texttt{SetTshape, QueryTshape, WriteString, ReadString}

\begin{itemize}
  \item \textbf{Module} \texttt{System}
\end{itemize}

\texttt{[out] String TextCursor HWindowX.GetTshape ( )}

\begin{itemize}
  \item \textbf{void} \texttt{HOperatorSetX.GetTshape ([in] VARIANT WindowHandle, [out] VARIANT TextCursor )}
\end{itemize}

\textit{Get the shape of the text cursor.}

\texttt{GetTshape} queries the shape of the text cursor for the output window. A new cursor shape is set by the operator \texttt{SetTshape}.

A text cursor marks the current position for text output (which can also be invisible). It is different from the mouse cursor (although both will be called ’’cursor’’ if the context makes misconceptions impossible). The available shapes for the text cursor can be queried with \texttt{QueryTshape}.

\begin{itemize}
  \item \textbf{Parameter}
\end{itemize}

\begin{itemize}
  \item \textbf{WindowHandle} (input control) ........................................... window $\rightsquigarrow$ \texttt{HWindowX / VARIANT}
    Window identifier.
  \item \textbf{TextCursor} (output control) ................................................ string $\rightsquigarrow$ \texttt{String / VARIANT}
    Name of the current text cursor.
\end{itemize}

\texttt{GetTshape} returns TRUE if the window is valid. Otherwise an exception handling is raised.

\begin{itemize}
  \item \textbf{Parallelization Information}
\end{itemize}

\texttt{GetTshape} is \textit{reentrant, local}, and processed \textit{without} parallelization.

\begin{itemize}
  \item \textbf{Possible Predecessors} \texttt{OpenWindow, OpenTextwindow, SetFont}
  \item \textbf{Possible Successors} \texttt{SetTshape, SetTposition, WriteString, ReadString, ReadChar}
\end{itemize}

\texttt{See also} \texttt{SetTshape, QueryTshape, WriteString, ReadString}

\begin{itemize}
  \item \textbf{Module} \texttt{System}
\end{itemize}
### 4.7. TEXT

<table>
<thead>
<tr>
<th>void HWindowX.NewLine ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.NewLine ([in] VARIANT WindowHandle)</td>
</tr>
</tbody>
</table>

Set the position of the text cursor to the beginning of the next line.

**NewLine** sets the position of the text cursor to the beginning of the next line. The new position depends on the current font. The left end of the baseline for writing the following text string (not considering descenders) is placed on this position.

If the next line does not fit into the window the content of the window is scrolled by the height of one line in the upper direction. In order to reach the correct new cursor position the font used in the next line must be set before **NewLine** is called. The position is changed by the output or input of text (**WriteString, ReadString**) or by an explicit change of position by (**SetTposition**).

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
</table>

▷ **WindowHandle** (input control) ........................ window ~ HWindowX / VARIANT Window identifier.

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
</table>

**NewLine** returns TRUE if the window is valid. Otherwise an exception handling is raised.

<table>
<thead>
<tr>
<th>Parallelization Information</th>
</tr>
</thead>
</table>

**NewLine** is **reentrant** and processed **without** parallelization.

<table>
<thead>
<tr>
<th>Possible Predecessors</th>
</tr>
</thead>
</table>

OpenWindow, OpenTextWindow, SetFont, WriteString

<table>
<thead>
<tr>
<th>See also</th>
</tr>
</thead>
</table>

WriteString, SetFont

<table>
<thead>
<tr>
<th>Alternatives</th>
</tr>
</thead>
</table>

GetTposition, GetStringExtents, SetTposition, MoveRectangle

<table>
<thead>
<tr>
<th>Module</th>
</tr>
</thead>
</table>

System

<table>
<thead>
<tr>
<th>[out] VARIANT Font HWindowX.QueryFont ( )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.QueryFont ([in] VARIANT WindowHandle, [out] VARIANT Font)</td>
</tr>
</tbody>
</table>

Query the available fonts.

**QueryFont** queries the fonts available for text output in the output window. They can be set with the operator **SetFont**. Fonts are used by the operators **WriteString, ReadChar, ReadString** and **NewLine**.

<table>
<thead>
<tr>
<th>Attention</th>
</tr>
</thead>
</table>

For different machines the available fonts may differ a lot. Therefore **QueryFont** will return different fonts on different machines.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
</table>

▷ **WindowHandle** (input control) ........................ window ~ HWindowX / VARIANT Window identifier.

▷ **Font** (output control) ............................... string ~ VARIANT ( string ) Tupel with available font names.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>

open_window(0,0,-1,-1,'root','visible',' ',WindowHandle)
set_check('˜text')
query_font(WindowHandle,Fontlist)
set_color(WindowHandle,'white')
for i=0 to |Fontlist|-1 by 1
    set_font(WindowHandle,Fontlist[i])
write_string(WindowHandle,Fontlist[i])
new_line(WindowHandle)
endfor

Result

Parallelization Information
QueryFont is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, OpenTextwindow

Possible Successors
SetFont, WriteString, ReadString, ReadChar

See also
SetFont, WriteString, ReadString, ReadChar, NewLine

Module

[out] VARIANT TextCursor HWindowX.QueryTshape()

void HOperatorSetX.QueryTshape ([in] VARIANT WindowHandle,
[out] VARIANT TextCursor )

Query all shapes available for text cursors.

QueryTshape queries the available shapes of text cursors for the output window. The retrieved shapes can be used by the operator SetTshape.

Parameter

WindowHandle (input control) window ~ HWindowX / VARIANT identifier.

TextCursor (output control) string ~ VARIANT (string)

Names of the available text cursors.

Result

QueryTshape returns TRUE.

Parallelization Information
QueryTshape is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, OpenTextwindow

Possible Successors
SetTshape, WriteString, ReadString

See also
SetTshape, GetShape, SetTposition, WriteString, ReadString

Module

[out] String Char HWindowX.ReadChar ([out] String Code )

void HOperatorSetX.ReadChar ([in] VARIANT WindowHandle,
[out] VARIANT Char, [out] VARIANT Code )

Read a character from a text window.

ReadChar reads a character from the keyboard in the input window (= output window). If the character is printable it is returned in Char. If a control key has been pressed, this will be indicated by the value of Code. Some important keys are recognizable by this value. Possible values are:
'character': printable character
'left': cursor left
'right': cursor right
'up': cursor up
'down': cursor down
'insert': insert
'none': none of these keys

The window has to be a text window.

Parameter

- **WindowHandle** (input control) \(\Rightarrow HWindowX / \text{VARIANT}\) Window identifier.
- **Char** (output control) \(\Rightarrow \text{String} / \text{VARIANT}\) Input character (if it is not a control character).
- **Code** (output control) \(\Rightarrow \text{String} / \text{VARIANT}\) Code for input character.

Result

ReadChar returns TRUE if the text window is valid. Otherwise an exception handling is raised.

Parallelization Information

ReadChar is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenTextwindow, SetFont, WriteString, SetFont

See also

Alternatives

ReadString, FreadChar, FreadString

Module

System

```
[out] String OutString HWindowX.ReadString ([in] String InString,
[in] long Length )
```

```
void HOperatorSetX.ReadString ([in] VARIANT WindowHandle,
```

Read a string in a text window.

**ReadString** reads a string with a predetermined maximum size (Length) from the keyboard in the input window (= output window). The string is read from the current position of the text cursor using the current font. The maximum size has to be small enough to keep the string within the right window boundary. A default string which can be edited or simply accepted by the user may be provided. After text input the text cursor is positioned at the end of the edited string. Commands for editing:

- **RETURN** finish input
- **BACKSPACE** delete the character on the left side of the cursor and move the cursor to this position.

Parameter

- **WindowHandle** (input control) \(\Rightarrow HWindowX / \text{VARIANT}\) Window identifier.
- **InString** (input control) \(\Rightarrow \text{String} / \text{VARIANT}\) Default string (visible before input).
  Default Value: ”
CHAPTER 4. GRAPHICS

- **Length** (input control) ................................................ integer \( \sim long / \text{VARIANT} \)
  Maximum number of characters.
  **Default Value**: 32
  **Restriction**: \( (\text{Length} > 0) \)

- **OutString** (output control) ........................................... string \( \sim \text{String} / \text{VARIANT} \)
  Read string.

**Result**

**ReadString** returns TRUE if the text window is valid and a string of maximal length fits within the right window boundary. Otherwise an exception handling is raised.

---

**Parallelization Information**

**ReadString** is reentrant, local, and processed without parallelization.

---

**Possible Predecessors**

OpenTextwindow, SetFont

---

See also

SetTitleposition, NewLine, OpenTextwindow, SetFont, SetColor

---

Alternatives

ReadChar, FreadString, FreadChar

---

Module

---

System

---

```
void HWindowX.SetFont ([in] String Font )
void HOperatorSetX.SetFont ([in] VARIANT WindowHandle, [in] VARIANT Font )
```

**Set the font used for text output.**

*SetFont* sets the font for the output window. The font is used by the operators *WriteString, ReadString* etc. Text windows as well as windows for image display use fonts. A default font is assigned when a window is opened. This font can be changed with *SetFont*. On UNIX systems all available fonts can be queried with *QueryFont*. The default font can be modified with *SetSystem(‘defaultFont’,Fontname)*. Fonts are not used for file operations.

The syntax for the specification of a font (in *Font*) differs for UNIX and Windows NT environments: In Windows NT a string with the following components is used:

- `-FontName-Height-Width-Italic-Underlined-Strikeout- [Bold-][Charset-]`

where “Italic”, “Underlined”, “Strikeout” and “Bold” can take the values 1 and 0 to activate or deactivate the corresponding feature. “Charset” can be used to select the character set, if it differs from the default one. You can use the names of the defines or the integer value.

All parameters beside “FontName” and “Height” are optional. But at the begin and end a Minus is suspected. To use the default setting, a * can be used instead. -Arial-10-*-*-*-*-*

Please refer to the Windows NT documentation for a detailed discussion.

---

**Attention**

For different machines the available fonts may differ a lot. Therefore it is suggested to use wildcards, tables of fonts and/or the operator *QueryFont*.

---

**Parameter**

- **WindowHandle** (input control) ................................ window \( \sim \text{HWindowX} / \text{VARIANT} \)
  Window identifier.

- **Font** (input control) ................................................. string \( \sim \text{String} / \text{VARIANT} \)
  Name of new font.

---

**Example**
```c
// UNIX
set_font(WindowHandle,'-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*:*
SetTshape is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, OpenTextwindow

Possible Successors
SetTshape, WriteString, ReadString

See also
GetTshape, QueryTshape, WriteString

Alternatives
NewLine

Module
System

void HWindowX.SetTshape ([in] String TextCursor )
void HOperatorSetX.SetTshape ([in] VARIANT WindowHandle, [in] VARIANT TextCursor )

Set the shape of the text cursor.

SetTshape sets the shape and the display mode of the text cursor of the output window. A text cursor marks the current position for text output. It is different from the mouse cursor (although both will be called 'cursor', if the context makes misconceptions impossible). The available shapes for the text cursor can be queried with QueryTshape.

Parameter

▷ WindowHandle (input control) ......................... window ↷ HWindowX / VARIANT Window identifier.
▷ TextCursor (input control) .......................... string ↷ String / VARIANT Name of cursor shape.

Default Value: 'invisible'

Result

GetTshape returns TRUE if the window is valid and the given cursor shape is defined for this window. Otherwise an exception handling is raised.

Parallelization Information
SetTshape is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, OpenTextwindow, QueryTshape, GetTshape

Possible Successors
WriteString, ReadString

See also
GetTshape, QueryTshape, WriteString, ReadString

Module
System

void HWindowX.WriteString ([in] VARIANT String )
void HOperatorSetX.WriteString ([in] VARIANT WindowHandle, [in] VARIANT String )

Print text in a window.

WriteString prints String in the output window starting at the current cursor position. The output text has to fit within the right window boundary (the width of the string can be queried by GetStringExtents).

HALCON/COM Reference Manual, 2005-2-1
The font currently assigned to the window will used. The text cursor is positioned at the end of the text. 
WriteString can output all three types of data used in HALCON. The conversion to a string is guided by the following rules:

- strings are not converted.
- integer numbers are converted without any spaces before or after the number.
- floating numbers are printed (if possible) with a floating point and without an exponent.
- the resulting strings are concatenated without spaces.

For buffering of texts see SetSystem with the flag 'flush_graphic'.

Attention

If clipping at the window boundary is desired, exceptions can be switched off by SetCheck('text').

Parameter

- WindowHandle (input control) ........................................... window ~ HWindowX / VARIANT Window identifier.
- String (input control) .................................................. string(-array) ~ VARIANT ( integer, real, string ) Tuple of output values (all types).
- Default Value : 'hello'

Result

WriteString returns TRUE if the window is valid and the output text fits within the current line (see SetCheck). Otherwise an exception handling is raised.

Parallelization Information

WriteString is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextwindow, SetFont, GetStringExtents

See also

SetTposition, GetStringExtents, OpenTextwindow, SetFont, SetSystem, SetCheck

Alternatives

FWriteString

Module

4.8 Window

void HWindowX.ClearRectangle ([in] VARIANT Row1, [in] VARIANT Column1,
[in] VARIANT Row2, [in] VARIANT Column2 )

void HOperatorSetX.ClearRectangle ([in] VARIANT WindowHandle,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2 )

Delete a rectangle on the output window.

ClearRectangle deletes all entries in the rectangle which is defined through the upper left corner (Row1,Column1) and the lower right corner (Row2,Column2). Deletion signifies that the specified rectangle is set to the background color (see OpenWindow, OpenTextwindow).

If you want to delete more than one rectangle, you may pass several rectangles, i.e., the parameters Row1, Column1, Row2 and Column2 are tuple.
CHAPTER 4. GRAPHICS

Parameter

- **WindowHandle** (input control)  
  Window identifier.

- **Row1** (input control)  
  Line index of upper left corner.
  
  Default Value: 10
  Typical range of values: $0 \leq \text{Row1} \leq 0$ (lin)
  Minimum Increment: 1
  Recommended Increment: 1

- **Column1** (input control)  
  Column index of upper left corner.
  
  Default Value: 10
  Typical range of values: $0 \leq \text{Column1} \leq 0$ (lin)
  Minimum Increment: 1
  Recommended Increment: 1

- **Row2** (input control)  
  Row index of lower right corner.
  
  Default Value: 118
  Typical range of values: $0 \leq \text{Row2} \leq 0$ (lin)
  Minimum Increment: 1
  Recommended Increment: 1
  Restriction: $(\text{Row2} > \text{Row1})$

- **Column2** (input control)  
  Column index of lower right corner.
  
  Default Value: 118
  Typical range of values: $0 \leq \text{Column2} \leq 0$ (lin)
  Minimum Increment: 1
  Recommended Increment: 1
  Restriction: $(\text{Column2} \geq \text{Column1})$

Example

/* "Interaktives" Löschen eines Rechtecks im Ausgabefenster: */
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
clear_rectangle(WindowHandle,L1,C1,L2,C2).

Result

If an output window exists and the specified parameters are correct, `ClearRectangle` returns TRUE. If necessary an exception handling is raised.

Parallelization Information

`ClearRectangle` is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, SetRgb, SetHsi, DrawRectangle1

See also

OpenWindow, OpenTextwindow

Alternatives

ClearWindow, DispRectangle1

Module

System

```c
void HWindowX.ClearWindow ( )
void HOperatorSetX.ClearWindow ([in] VARIANT WindowHandle )
```

Delete an output window.

HALCON/COM Reference Manual, 2005-2-1
**4.8. WINDOW**

**ClearWindow** deletes all entries in the output window. The window (background and edge) is reset to its original state. Parameters assigned to this window (e.g., with **SetColor**, **SetPaint**, etc.) remain unmodified.

- **Parameter**
  - `WindowHandle` (input control) \[HWindowX / VARIANT\]
    - Window identifier.

- **Example**
  - `clear_window(WindowHandle)`.

- **Result**
  - If the output window is valid **ClearWindow** returns TRUE. If necessary an exception handling is raised.

- **Parallelization Information**
  - **ClearWindow** is reentrant, local, and processed without parallelization.

- **Possible Predecessors**
  - OpenWindow, OpenTextwindow

- **See also**
  - **OpenWindow**, **OpenTextwindow**

**ClearRectangle**, **DispRectangle1**

**Module**

**System**

```c
void HOperatorSetX.CloseWindow ([in] VARIANT WindowHandle )
```

*Close an output window.*

**CloseWindow** closes a window which have been opened by **OpenWindow** or by **OpenTextwindow**. Afterwards the output device or the window area, respectively, is ready to accept new calls of **OpenWindow** or **OpenTextwindow**.

- **Parameter**
  - `WindowHandle` (input control) \[HWindowX / VARIANT\]
    - Window identifier.

- **Result**
  - If the output window is valid **CloseWindow** returns TRUE. If necessary an exception handling is raised.

- **Parallelization Information**
  - **CloseWindow** is reentrant, local, and processed without parallelization.

- **Possible Predecessors**
  - OpenWindow, OpenTextwindow

- **See also**
  - **OpenWindow**, **OpenTextwindow**

**System**

```c
```

- Copy all pixels within rectangles between output windows.

```c
```

*Copy all pixels within rectangles between output windows.*
**CopyRectangle** copies all pixels from the specified window with the logical window number **WindowHandleSource** in the specified window with the logical window number **WindowHandleDestination**. It copies pixels which reside inside a rectangle which is specified by parameters **Row1**, **Column1**, **Row2** and **Column2**. The target position is specified through the upper left corner of the rectangle (**DestRow**, **DestColumn**).

If you want to move more than one rectangle, you may pass them at once (in form of the tuple mode).

You may use **CopyRectangle** to copy edited graphics from an "invisible" window in a visible window. Therefore a window with the option 'buffer' is opened. The graphics is then displayed in this window and is copied in a visible window afterwards. The advantage of this strategy is, that **CopyRectangle** is much more rapid than output procedures as, e.g., **DispChannel**. This means a particular advantage while using demo programs. You could even realise short "clips": you have to create for every image of a sequence a window of a 'buffer' type and pass the data into it. Output is the image sequence whereat all buffers are copied one after another in a visible window.

Attention

Both windows have to reside on the same computer.

Parameter

- **WindowHandleSource** (input control) \(\rightarrow HWindowX \) / VARIANT
  Number of the source window.
- **WindowHandleDestination** (input control) \(\rightarrow HWindowX \) / VARIANT
  Number of the destination window.
- **Row1** (input control) \(\rightarrow rectangle\).origin.y(-array) \(\sim\) VARIANT (integer)
  Row index of upper left corner in the source window.
  Default Value : 0
  Typical range of values : \(0 \leq \text{Row1} \leq 0\) (lin)
  Minimum Increment : 1
  Recommended Increment : 1
- **Column1** (input control) \(\rightarrow rectangle\).origin.x(-array) \(\sim\) VARIANT (integer)
  Column index of upper left corner in the source window.
  Default Value : 0
  Typical range of values : \(0 \leq \text{Column1} \leq 0\) (lin)
  Minimum Increment : 1
  Recommended Increment : 1
- **Row2** (input control) \(\rightarrow rectangle\).corner.y(-array) \(\sim\) VARIANT (integer)
  Row index of lower right corner in the source window.
  Default Value : 128
  Typical range of values : \(0 \leq \text{Row2} \leq 0\) (lin)
  Minimum Increment : 1
  Recommended Increment : 1
  Restriction : \((\text{Row2} \geq \text{Row1})\)
- **Column2** (input control) \(\rightarrow rectangle\).corner.x(-array) \(\sim\) VARIANT (integer)
  Column index of lower right corner in the source window.
  Default Value : 128
  Typical range of values : \(0 \leq \text{Column2} \leq 0\) (lin)
  Minimum Increment : 1
  Recommended Increment : 1
  Restriction : \((\text{Column2} \geq \text{Column1})\)
- **DestRow** (input control) \(\rightarrow point\).y(-array) \(\sim\) VARIANT (integer)
  Row index of upper left corner in the target window.
  Default Value : 0
  Typical range of values : \(0 \leq \text{DestRow} \leq 0\) (lin)
  Minimum Increment : 1
  Recommended Increment : 1
4.8. WINDOW

- **DestColumn** (input control) .......................... point.x(-array) -> VARIANT (integer)
  - Column index of upper left corner in the target window.
  - **Default Value:** 0
  - **Typical range of values:** 0 ≤ DestColumn ≤ 0
  - **Minimum Increment:** 1
  - **Recommended Increment:** 1

---

**Example**

```plaintext
read_image(Image,'affe')
open_window(0,0,-1,-1,'root','buffer','',WindowHandle)
disp_image(Image,WindowHandle)
open_window(0,0,-1,-1,'root','visible','',WindowHandleDestination)
repeat()
get_mbutton(WindowHandleDestination,Row,Column,Button)
copy_rectangle(BufferID,WindowHandleDestination,20,90,120,390,Row,Column)
until(Button = 1)
close_window(WindowHandleDestination)
close_window(WindowHandle)
clear(Image).
```

---

**Result**

If the output window is valid and if the specified parameters are correct, **CloseWindow** returns TRUE. If necessary an exception handling is raised.

---

**Parallelization Information**

**CopyRectangle** is reentrant, local, and processed without parallelization.

---

**Possible Predecessors**

OpenWindow, OpenTextwindow

**Possible Successors**

CloseWindow

---

**See also**

OpenWindow, OpenTextwindow

---

**Alternatives**

MoveRectangle, SlideImage

---

**Module**

System

---

```plaintext
void HWindowX.DumpWindow ([in] VARIANT Device, [in] String FileName )
void HOperatorSetX.DumpWindow ([in] VARIANT WindowHandle, 
```

Write the window content to a file.

**DumpWindow** writes the content of the window to a file. You may continue to process this file by convenient printers or other programs. The content of a display is prepared for each special device (**Device**), i.e., it is formatted in a manner, that you may print this file directly or it can be processed furthermore by a graphical program.

To transform gray values the current color table of the window is used, i.e., the values of **SetLutStyle** remain unconsidered.

The gray values of the display have in general a poorer resolution than the original gray values of the output images. This results in a reduced resolution (default setting are, e.g., 200 gray levels with 8 bit planes) for the representation of gray values (see also **DispChannel**). You may request the amount of actual disposable gray levels by **GetSystem**. To modify them before opening the first window you may use **SetSystem**.

Possible values for **Device**

- **'postscript'**: PostScript - file. Filetermination: `<Filename>.ps`
'postscript', Width, Height: PostScript - file with specification of the output size. Width and Height refer to the size. In this case a tuple with three values as Device is passed. File termination: <FileName>.ps
'tiff': TIFF - file, 1 byte per pixel incl. current color table or 3 bytes per sample (dependent on VGA card), uncompressed. File termination: <FileName>.tiff
'bmp': Windows-BMP format, RGB image, 3 bytes per pixel. The color resolution depends on the VGA card. File extension: '.bmp'
'jpeg': JPEG format, with lost of information; together with the format string the quality value determining the compression rate can be provided: e.g., 'jpeg 30'.
'png': PNG format (lossless compression); together with the format string, a compression level between 0 and 9 can be specified, where 0 corresponds to no compression and 9 to the best possible compression. Alternatively, the compression can be selected with the following strings: 'best', 'fastest', and 'none'. Hence, examples for correct parameters are 'png', 'png 7', and 'png none'. Images of type byte and uint2 can be stored in PNG files. If an image with a reduced domain is written, the region is stored as the alpha channel, where the points within the domain are stored as the maximum gray value of the image type and the points outside the domain are stored as the gray value 0. If an image with a full domain is written, no alpha channel is stored.

Attention
Under X Windows, the HALCON window must be completely visible on the root window, because otherwise the contents of the window cannot be read due to limitations in X Windows. If larger graphical displays are to be written to a file, the window type 'pixmap' can be used.

Parameter

- **WindowHandle** (input control) ............................. window ~ HWindowX / VARIANT Window identifier.
- **Device** (input control) ................................. string(-array) ~ VARIANT( string, integer ) Name of the target device or of the graphic format.
  Default Value: 'postscript'
  List of values: Device ∈ { 'postscript', 'tiff', 'bmp', 'jpeg', 'png', 'jpeg 100', 'jpeg 80', 'jpeg 60', 'jpeg 40', 'jpeg 20', 'png best', 'png fastest', 'png none' }
- **FileName** (input control) ................................. filename ~ String / VARIANT File name (without extension).
  Default Value: 'halcon_dump'

Example

/* PostScript - Dump von Bild und eingeblendeten Regionen: */
disp_image(Image,WindowHandle)
set_colored(WindowHandle,12)
disp_region(Regions,WindowHandle)
dump_window(WindowHandle,'postscript','/tmp/halcon_dump')
system_call('lp -d ps /tmp/halcon_dump.ps').

Result
If the appropriate window is valid and the specified parameters are correct DumpWindow returns TRUE. If necessary an exception handling is raised.

Parallelization Information
DumpWindow is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextwindow, DispRegion

Possible Successors
SystemCall

See also
OpenWindow, OpenTextwindow, SetSystem

Module

System

HALCON/COM Reference Manual, 2005-2-1
Get the operating system window handle.

GetOsWindowHandle returns the operating system window handle of the HALCON window WindowHandle in OWindowHandle. Under UNIX, additionally the operating system display handle is returned in OSDisplayHandle. The operating system window handle can be used to access the window using functions from the operating system, e.g., to draw in a user-defined manner into the window. Under Windows, OWindowHandle can be cast to a variable of type HWND. Under UNIX systems, OWindowHandle can be cast into a variable of type Window, while OSDisplayHandle can be cast into a variable of type Display.

Parameter

Output control.

Result

If the window is valid GetOsWindowHandle returns TRUE. Otherwise, an exception handling is raised.

Parallelization Information

GetOsWindowHandle is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextWindow

Module

System

Get window characteristics.

The operator GetWindowAttr can be used to read characteristics of graphics windows that were set using SetWindowAttr. The following parameters of a window may be queried:

'border_width' Width of the window border in pixels.
'border_color' Color of the window border.
'background_color' Background color of the window.
>window_title' Name of the window in the titlebar.

Parameter

Attribute name.

Result

If the parameters are correct GetWindowAttr returns TRUE. If necessary an exception handling is raised.

Parallelization Information

GetWindowAttr is reentrant, local, and processed without parallelization.
CHAPTER 4. GRAPHICS

Possible Predecessors
OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextwindow

See also
OpenWindow, SetWindowAttr

Module

System

Information about a window’s size and position.

**GetWindowExtents** returns the position of the upper left corner, as well as width and height of the output window.

```
[out] long Row HWindowX.GetWindowExtents ([out] long Column,
[out] long Width, [out] long Height )
```

```
void HOperatorSetX.GetWindowExtents ([in] VARIANT WindowHandle,
[out] VARIANT Row, [out] VARIANT Column, [out] VARIANT Width,
[out] VARIANT Height )
```

Attention
Size and position of a window may be modified by the window manager, without explicit instruction in the program. Therefore the values which are returned by **GetWindowExtents** may change cause of side effects.

Parameter

- **WindowHandle** (input control) ................. window \( \sim \) HWindowX / VARIANT
  Window identifier.
- **Row** (output control) ................................ rectangle.origin.y \( \sim \) long / VARIANT
  Row index of upper left corner of the window.
- **Column** (output control) .......................... rectangle.origin.x \( \sim \) long / VARIANT
  Column index of upper left corner of the window.
- **Width** (output control) ............................ rectangle.extent.x \( \sim \) long / VARIANT
  Window width.
- **Height** (output control) .......................... rectangle.extent.y \( \sim \) long / VARIANT
  Window height.

Example

```
open_window(100,100,200,200,'root','visible',''",WindowHandle)
fwrite_string('Move the window with the mouse!')
fnew_line()
repeat()
get_mbutton(WindowHandle,_,_,Button)
get_window_extents(WindowHandle,Row,Column,Width,Height)
fwrite(['('Row,',',Column,')'])
fnew_line()
until(Button = 4).
```

Result
If the window is valid **GetWindowExtents** returns TRUE. If necessary an exception handling is raised.

Parallelization Information
**GetWindowExtents** is reentrant, local, and processed without parallelization.

Possible Predecessors
OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextwindow

See also
SetWindowExtents, OpenWindow, OpenTextwindow

Module

System
Access to a window’s pixel data.

GetWindowPointer3 enables (in some window systems) the direct access to the bitmap. Result values are the three pointers on the color extracts of a 24-bit window (ImageRed, ImageGreen, ImageBlue), as well as the window size (Width, Height). In the language C the type of the image points is unsigned char.

Attention

GetWindowPointer3 is usable only for window type ‘pixmap’.

Parameter

- **WindowHandle** (input control) \( \text{window} \sim \text{HWindowX} / \text{VARIANT} \)
  Window identifier.
- **ImageRed** (output control) \( \text{ImageRed} \sim \text{long} / \text{VARIANT} \)
  Pointer on red channel of pixel data.
- **ImageGreen** (output control) \( \text{ImageGreen} \sim \text{long} / \text{VARIANT} \)
  Pointer on green channel of pixel data.
- **ImageBlue** (output control) \( \text{ImageBlue} \sim \text{long} / \text{VARIANT} \)
  Pointer on blue channel of pixel data.
- **Width** (output control) \( \text{Width} \sim \text{long} / \text{VARIANT} \)
  Length of an image line.
- **Height** (output control) \( \text{Height} \sim \text{long} / \text{VARIANT} \)
  Number of image lines.

Result

If a window of type ‘pixmap’ exists and it is valid GetWindowPointer3 returns TRUE. If necessary an exception handling is raised.

Parallelization Information

GetWindowPointer3 is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextwindow

See also

OpenWindow, SetWindowType

Module

System

Get the window type.

GetWindowType determines the type or the graphical software, respectively, of the output device for the window. You may query the available types of output devices with procedure QueryWindowType. A reasonable use for GetWindowType might be in the field of the development of machine independent software. Possible values are:

‘X-Window’ X-Window Version 11.

‘pixmap’ Windows are not shown, but managed in memory. By this means HALCON programs can be ported on computers without a graphical display.

‘PostScript’ Objects are output to a PostScript File.
CHAPTER 4. GRAPHICS

Parameter ⊲ WindowHandle (input control) ........................................... window  ~ HWindowX / VARIANT
Window identifier.

Parameter ⊲ WindowType (output control) ............................................. string  ~ String / VARIANT
Window type

Example

open_window(100,100,200,200,’root’,’visible’,’,’,WindowHandle)
get_window_type(WindowHandle,WindowType)
fwrite_string([’Window type: ’,WindowType])
fnew_line().

Result

If the window is valid GetWindowType returns TRUE. If necessary an exception handling is raised.

Parallelization Information

GetWindowType is reentrant, local, and processed without parallelization.

Possible Predecessors

OpenWindow, OpenTextwindow

See also

QueryWindowType, SetWindowType, GetWindowPointer3, OpenWindow, OpenTextwindow

Module

System

Copy inside an output window.

MoveRectangle copies all entries in the rectangle (Row1,Column1), (Row2,Column2) of the output window
to a new position inside the same window. This position is determined by the upper left corner (DestRow,
DestColumn). Regions of the window, which are "uncovered" through moving the rectangle, are set to the
color of the background.

If you want to move several rectangles at once, you may pass parameters in form of tupels.

Parameter

▷ WindowHandle (input control) ........................................... window  ~ HWindowX / VARIANT
Window identifier.

▷ Row1 (input control) ............................................... rectangle.origin.y(-array)  ~ VARIANT ( integer )
Row index of upper left corner of the source rectangle.
Default Value : 0
Typical range of values : 0 ≤ Row1 ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 1

▷ Column1 (input control) ............................................... rectangle.origin.x(-array)  ~ VARIANT ( integer )
Column index of upper left corner of the source rectangle.
Default Value : 0
Typical range of values : 0 ≤ Column1 ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 1
4.8. WINDOW

Row2 (input control) .......................................................... rectangle.corner.y(-array) \sim \text{VARIANT (integer)}
Row index of lower right corner of the source rectangle.
Default Value : 64
Typical range of values : $0 \leq \text{Row2} \leq 0$ (lin)
Minimum Increment : 1
Recommended Increment : 1

\textbf{Column2} (input control) ........................................ rectangle.corner.x(-array) \sim \text{VARIANT (integer)}
Column index of lower right corner of the source rectangle.
Default Value : 64
Typical range of values : $0 \leq \text{Column2} \leq 0$ (lin)
Minimum Increment : 1
Recommended Increment : 1

\textbf{DestRow} (input control) .......................................... point.y(-array) \sim \text{VARIANT (integer)}
Row index of upper left corner of the target position.
Default Value : 64
Typical range of values : $0 \leq \text{DestRow} \leq 0$ (lin)
Minimum Increment : 1
Recommended Increment : 1

\textbf{DestColumn} (input control) ................................. point.x(-array) \sim \text{VARIANT (integer)}
Column index of upper left corner of the target position.
Default Value : 64
Typical range of values : $0 \leq \text{DestColumn} \leq 0$ (lin)
Minimum Increment : 1
Recommended Increment : 1

\begin{verbatim}
Example

/* "Interaktives" Kopieren eines Rechtecks im Ausgabefenster: */
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
get_mbutton(WindowHandle,LN,CN,Button)
move_rectangle(WindowHandle,L1,C1,L2,C2,LN,CN).
\end{verbatim}

\textbf{Result}

If the window is valid and the specified parameters are correct \texttt{MoveRectangle} returns TRUE. If necessary an exception handling is raised.

\textbf{Parallelization Information}

\texttt{MoveRectangle} is reentrant, local, and processed without parallelization.

\textbf{Possible Predecessors}

OpenWindow, OpenTextwindow

See also

OpenWindow, OpenTextwindow

\textbf{Alternatives}

CopyRectangle

\textbf{Module}

\textbf{System}

\begin{verbatim}
void HWindowX.NewExternWindow ([in] long WINHWnd, [in] long WINHDC,

void HOperatorSetX.NewExternWindow ([in] VARIANT WINHWnd,
[in] VARIANT WINHDC, [in] VARIANT Row, [in] VARIANT Column,
[in] VARIANT Width, [in] VARIANT Height, [out] VARIANT WindowHandle )
\end{verbatim}

Create a virtual graphics window under Windows NT.
\texttt{NewExternWindow} opens a new virtual window. Virtual means that a new window will not be created, but the window whose Windows NT handle is given in the parameter \texttt{WINHWnd} is used to perform output of gray value
data, regions, graphics as well as to perform textual output. Visualization parameters for the output of data can be done either using HALCON commands or by the appropriate Windows NT commands.

Example: setting of the drawing color:

HALCON:
```
set_color(WindowHandle,"green");
disp_region(WindowHandle,region);
```

Windows NT:
```
HPEN* penold;
HPEN penGreen = CreatePen(PS_SOLID,1,RGB(0,255,0));
pen = (HPEN*)SelectObject(WINHDC,penGreen);
disp_region(WindowHandle,region);
```

Interactive operators, for example `DrawRegion`, `DrawCircle` or `GetMbutton` cannot be used in this window. The following operators can be used:

- Output of gray values: `SetPaint`, `SetComprise`, (`SetLut` and `SetLutStyle` after output)
- Regions: `SetColor`, `SetRgb`, `SetHsi`, `SetGray`, `SetPixel`, `SetShape`, `SetLineWidth`, `SetInsert`, `SetLineStyle`, `SetDraw`
- Image part: `SetPart`
- Text: `SetFont`

You may query current set values by calling procedures like `GetShape`. As some parameters are specified through the hardware (Resolution/Colors), you may query current available resources by calling operators like `QueryColor`.

The parameter `WINHWnd` is used to pass the window handle of the Windows NT window, in which output should be done. The parameter `WINHDC` is used to pass the device context of the window `WINHWnd`. This device context is used in the output routines of HALCON.

The origin of the coordinate system of the window resides in the upper left corner (coordinates: (0,0)). The row index grows downward (maximum: `Height` - 1), the column index grows to the right (maximal: `Width` - 1).

You may use the value `-1` for parameters `Width` and `Height`. This means, that the corresponding value is chosen automatically. In particular, this is important if the aspect ratio of the pixels is not 1.0 (see `SetSystem`). If one of the two parameters is set to `-1`, it will be chosen through the size which results out of the aspect ratio of the pixels. If both parameters are set to `-1`, they will be set to the current image format.

The position and size of a window may change during runtime of a program. This may be achieved by calling `SetWindowExtents`, but also through external influences (window manager). For the latter case the procedure `SetWindowExtents` is provided.

Opening a window causes the assignment of a default font. It is used in connection with procedures like `WriteString` and you may change it by performing `SetFont` after calling `OpenWindow`. On the other hand, you have the possibility to specify a default font by calling `SetSystem` (`::'defaultFont',<Fontname>::`) before opening a window (and all following windows; see also `QueryFont`).

You may set the color of graphics and font, which is used for output procedures like `DispRegion` or `DispCircle`, by calling `SetRgb`, `SetHsi`, `SetGray`, `SetPixel`, `SetShape` or `SetLineWidth`. Calling `SetInsert` specifies how graphics is combined with the content of the image repeat memory. Thereto you may achieve by calling, e.g., `SetInsert` ( `::'not':`) to eliminate the font after writing text twice at the same position.

The content of the window is not saved, if other windows overlap the window. This must be done in the program code that handles the Windows NT window in the calling program.

For graphical output ( `DispImage`, `DispRegion`, etc.) you may adjust the window by calling procedure `SetPart` in order to represent a logical clipping of the image format. In particular this implies that only this part (appropriately scaled) of images and regions is displayed. Before you close your window, you have to close the HALCON-window.

Steps to use new _extern_window:

**Creation:**
- Create a Windows-window.
4.8. WINDOW

- Call newExternWindow with the WINHWnd of the above created window.

Use:  
- Before drawing in the window you have to call the method setWindowDC. This ensures that the halcon drawing routines use the right DC. After drawing call again setWindowDC, but this time with the address of a long set to zero, this ensures that HALCON can delete the created graphic objects.

Destroy: 
- Call closeWindow.

Attention
Note that parameters as Row, Column, Width and Height are constrained through the output device, i.e., the size of the Windows NT desktop.

Parameter

- **WINHWnd** (input control) .......................... integer \( \sim \) long / VARIANT  
  Windows windowhandle of a previously created window.
  
  **Restriction:** \((WINHWnd \neq 0)\)

- **WINHDC** (input control) .......................... integer \( \sim \) long / VARIANT  
  Device context of WINHWnd.
  
  **Restriction:** \((WINHDC \neq 0)\)

- **Row** (input control) .......................... rectangle.origin.y \( \sim \) long / VARIANT  
  Row coordinate of upper left corner.
  
  **Default Value:** 0
  
  **Restriction:** \((Row \geq 0)\)

- **Column** (input control) .......................... rectangle.origin.x \( \sim \) long / VARIANT  
  Column coordinate of upper left corner.
  
  **Default Value:** 0
  
  **Restriction:** \((Column \geq 0)\)

- **Width** (input control) .......................... rectangle.extent.x \( \sim \) long / VARIANT  
  Width of the window.
  
  **Default Value:** 512
  
  **Restriction:** \(((Width > 0) \lor (Width = -1))\)

- **Height** (input control) .......................... rectangle.extent.y \( \sim \) long / VARIANT  
  Height of the window.
  
  **Default Value:** 512
  
  **Restriction:** \(((Height > 0) \lor (Height = -1))\)

- **WindowHandle** (output control) ...................... window \( \sim \) HWindowX / VARIANT  
  Window identifier.

Result
If the values of the specified parameters are correct NewExternWindow returns TRUE. If necessary, an exception is raised.

Parallelization Information
NewExternWindow is reentrant, local, and processed without parallelization.

Possible Predecessors
ResetObjDb

Possible Successors
SetColor, QueryWindowType, getWindowType, SetWindowType, GetMposition, SetTposition, SetTshape, SetWindowExtents, GetWindowExtents, QueryColor, SetCheck, SetSystem

See also
OpenWindow, DispRegion, DispImage, DispColor, SetLut, QueryColor, SetColor, SetRgb, SetHsi, SetPixel, SetGray, SetPart, SetPartStyle, QueryWindowType, GetWindowType, SetWindowType, GetMposition, SetTposition, SetWindowExtents, GetWindowExtents, SetWindowAttr, SetCheck, SetSystem

Alternatives
OpenWindow, OpenTextwindow

Module
System
Open a textual window.

**OpenTextwindow** opens a new textual window, which can be used to perform textual input and output, as well as to perform output of images. All output (**WriteString**, **ReadString**, **DispRegion**, etc.) is redirected to this window, if the same logical window number **WindowHandle** is used.

Besides the mouse cursor textual windows possess also a textual cursor which indicates the current writing position (more exactly: the lower left corner of the output string without consideration of descenders). Its position is indicated through an underscore or another type (the indication of this position may also be disabled (= default setting); cf. **SetTshape**). You may set or query the position by calling the procedures **SetTposition** or **GetTposition**.

After you opened a textual window the position of the cursor is set to (H,0). Whereby H signifies the height of the default font less the descenders. But the cursor is not shown. Hence the output starts for writing in the upper left corner of the window.

You may query the colors of the background and the image edges by calling **QueryColor**. In the same way you may use **QueryFont** in a window of type 'invisible'. During output (**WriteString**) you may set the clipping of text out of the window edges by calling **SetCheck**(::'text'). This disables the creation of error messages, if text passes over the edge of the window.

The origin of the coordinate system of the window resides in the upper left corner (coordinates: (0,0)). The row index grows downward (maximal: **Height**-1), the column index grows to the right (maximal: **Width**-1).

The parameter **Machine** indicates the name of the computer, which has to open the window. In case of a X-window, TCP-IP only sets the name, DEC-Net sets in addition a colon behind the name. The ”’server’” or the ”’screen’”, respectively, are not specified. If the empty string is passed the environment variable DISPLAY is used. It indicates the target computer. At this the name is indicated in common syntax

```
<Host>:0.0
```

Position and size of a window may change during runtime of a program. This may be achieved by calling **SetWindowExtents**, but also through external interferences (window manager). In the latter case the procedure **SetWindowExtents** is provided.

Opening a window causes the assignment of a called default font. It is used in connection with procedures like **WriteString** and you may overwrite it by performing **SetFont** after calling **OpenTextwindow**. On the other hand you have the possibility to specify a default font by calling **SetSystem**(::'defaultFont',<Fontname>:)) before opening a window (and all following windows; see also **QueryFont**).

You may set the color of the font (**WriteString**, **ReadString**) by calling **SetColor**, **SetRgb**, **SetHsi**, **SetGray** or **SetPixel**. Calling **SetInsert** specifies how the text or the graphics, respectively, is combined with the content of the image repeat memory. So you may achieve by calling, e.g., **SetInsert**(::'not:') to eliminate the font after writing text twice at the same position.

Normally every output (e.g., **WriteString**, **DispRegion**, **DispCircle**, etc.) in a window is terminated by a ”’flush’”. This causes the data to be fully visible on the display after termination of the output procedure. But this is not necessary in all cases, in particular if there are permanently output tasks or there is a mouse procedure active. Therefore it is more favorable (i.e., more rapid) to store the data until sufficient data is available. You may stop this behavior by calling **SetSystem**(::'flushGraphic',’false’).

The content of windows is saved (in case it is supported by special driver software); i.e., it is preserved, also if the window is hidden by other windows. But this is not necessary in all cases: If you use a textual window, e.g., as a parent window for other windows, you may suppress the security mechanism for it and save the
necessary memory at the same moment. You achieve this before opening the window by calling `SetSystem (::'backingStore', 'false');`.

Difference: graphical window - textual window

- In contrast to graphical windows (OpenWindow) you may specify more parameters (color, edge) for a textual window while opening it.
- You may use textual windows only for input of user data (ReadString).
- Using textual windows, the output of images, regions and graphics is "clipped" at the edges. Whereas during the use of graphical windows the edges are "zoomed".
- The coordinate system (e.g., with GetMbutton or GetMposition) consists of display coordinates independently of image size. The maximum coordinates are equal to the size of the window minus 1. In contrast to this, graphical windows (OpenWindow) use always a coordinate system, which corresponds to the image format.

The parameter Mode specifies the mode of the window. It can have following values:

'visible': Normal mode for textual windows: The window is created according to the parameters and all inputs and outputs are possible.

'invisible': Invisible windows are not displayed in the display. Parameters like Row, Column, BorderWidth, BorderColor, BackgroundColor and FatherWindow do not have any meaning. Output to these windows has no effect. Input (ReadString, mouse, etc.) is not possible. You may use these windows to query representation parameter for an output device without opening a (visible) window. General queries are, e.g., QueryColor and GetStringExtents.

'transparent': These windows are transparent: the window itself is not visible (edge and background), but all the other operations are possible and all output is displayed. Parameters like BorderColor and BackgroundColor do not have any meaning. A common use for this mode is the creation of mouse sensitive regions.

'buffer': These are also not visible windows. The output of images, regions and graphics is not visible on the display, but is stored in memory. Parameters like Row, Column, BorderWidth, BorderColor, BackgroundColor and FatherWindow do not have any meaning. You may use buffer windows, if you prepare output (in the background) and copy it finally with CopyRectangle in a visible window. Another usage might be the rapid processing of image regions during interactive manipulations. Textual input and mouse interaction are not possible in this mode.

**Attention**

You have to keep in mind that parameters like Row, Column, Width and Height are restricted by the output device. Is a father window (FatherWindow <> 'root') specified, then the coordinates are relative to this window.

---

**Parameter**

- **Row** (input control) ................................. rectangle.origin.y  ~> long / VARIANT
  - Row index of upper left corner.
  - Default Value : 0
  - Minimum Increment : 1
  - Recommended Increment : 1
  - Restriction : (Row ≥ 0)

- **Column** (input control) ............................. rectangle.origin.x  ~> long / VARIANT
  - Column index of upper left corner.
  - Default Value : 0
  - Minimum Increment : 1
  - Recommended Increment : 1
  - Restriction : (Column ≥ 0)

- **Width** (input control) .............................. rectangle.extent.x  ~> long / VARIANT
  - Window’s width.
  - Default Value : 256
  - Minimum Increment : 1
  - Recommended Increment : 1
  - Restriction : (Width > 0)
CHAPTER 4. GRAPHICS

- **Height** (input control) ............................................................ \( \text{rectangle.extent.y} \sim \text{long} / \text{VARIANT} \\
Window’s height.
  
  Default Value: 256
  
  (lin) Minimum Increment: 1
  
  Recommended Increment: 1
  
  Restriction: \((\text{Height} > 0)\)

- **BorderWidth** (input control) .................................................. \( \text{integer} \sim \text{long} / \text{VARIANT} \\
Window border’s width.
  
  Default Value: 2
  
  (lin) Minimum Increment: 1
  
  Recommended Increment: 1
  
  Restriction: \((\text{BorderWidth} \geq 0)\)

- **BorderColor** (input control) .................................................. \( \text{string} \sim \text{String} / \text{VARIANT} \\
Window border’s color.
  
  Default Value: ‘white’

- **BackgroundColor** (input control) ............................................. \( \text{string} \sim \text{String} / \text{VARIANT} \\
Background color.
  
  Default Value: ‘black’

- **FatherWindow** (input control) .............................................. \( \text{integer} \sim \text{VARIANT} (\text{integer}, \text{string}) \\
Logical number of the father window. For the display as father you may specify ‘root’ or 0.
  
  Default Value: 0
  
  Restriction: \((\text{FatherWindow} \geq 0)\)

- **Mode** (input control) ............................................................ \( \text{string} \sim \text{String} / \text{VARIANT} \\
Window mode.
  
  Default Value: ‘visible’
  
  List of values: \(\text{Mode} \in \{‘visible’, ‘invisible’, ‘transparent’, ‘buffer’\}\)

- **Machine** (input control) ....................................................... \( \text{string} \sim \text{String} / \text{VARIANT} \\
Computer name, where the window has to be opened or empty string.
  
  Default Value: ”

- **WindowHandle** (output control) ............................................. \( \text{window} \sim \text{HWindowX} / \text{VARIANT} \\
Window identifier.

---

**Example**

```plaintext
open_textwindow(0,0,900,600,1,'black','slate blue','root','visible','WindowHandle)
open_textwindow(10,10,300,580,3,'red','blue',Father,'visible','WindowHandle)
open_window(10,320,570,580,Father,'visible','WindowHandle)
set_color(WindowHandle,'red')
read_image(Image,'affe')
disp_image(Image,WindowHandle)
```

---

**Result**

If the values of the specified parameters are correct **OpenTextwindow** returns TRUE. If necessary an exception handling is raised.

---

**Parallelization Information**

**OpenTextwindow** is **reentrant**, **local**, and processed **without** parallelization.

---

**Possible Predecessors**

**ResetObjDb**

HALCON/COM Reference Manual, 2005-2-1
Open a graphics window.

OpenWindow opens a new window, which can be used to perform output of gray value data, regions, graphics as well as to perform textual output. All output (DispRegion, DispImage, etc.) is redirected to this window, if the same logical window number WindowHandle is used.

The background of the created window is set to black in advance and it has a white border, which is 2 pixels wide (see also SetWindowAttr(::'borderWidth',<Breite>):).

Certain parameters used for the editing of output data are assigned to a window. These parameters are considered during the output itself (e.g., with DispImage or DispRegion). They are not specified by an output procedure, but by "configuration procedures". If you want to set, e.g., the color red for the output of regions, you have to call SetColor(::WindowHandle,'red':) before calling DispRegion. These parameters are always set for the window with the logical window number WindowHandle and remain assigned to a window as long as they will be overwritten. You may use the following configuration procedures:

- Output of gray values: SetPaint, SetComprise, (SetLut and SetLutStyle after output)
- Regions: SetColor, SetRgb, SetHsi, SetGray, SetPixel, SetShape, SetLineWidth, SetInsert, SetLineStyle, SetDraw
- Image clipping: SetPart
- Text: SetFont

You may query current set values by calling procedures like GetShape. As some parameters are specified through the hardware (Resolution/Colors), you may query current available resources by calling QueryColor.

The origin of the coordinate system of the window resides in the upper left corner (coordinates: (0,0)). The row index grows downward (maximal: Height-1), the column index grows to the right (maximal: Width-1). You have to keep in mind, that the range of the coordinate system is independent of the window size. It is specified only through the image format (see ResetObjDb).

The parameter Machine indicates the name of the computer, which has to open the window. In case of a X-window, TCP-IP only sets the name, DEC-Net sets in addition a colon behind the name. The "’server’" resp. the "’screen’" are not specified. If the empty string is passed the environment variable DISPLAY is used. It indicates the target computer. At this the name is indicated in common syntax

```
<Host>:0.0
```

You may use the value "’-1’" for parameters Width and Height. This means, that the according value has to be specified automatically. In particular this is of importance, if the proportion of pixels is not 1.0 (see SetSystem):
Is one of the two parameters set to "-1", it will be specified through the size which results out of the proportion of pixels. Are both parameters set to "-1", they will be set to the maximum image format, which is currently used (further information about the currently used maximum image format can be found in the description of GetSystem using "width" or "height").

Position and size of a window may change during runtime of a program. This may be achieved by calling SetWindowExtents, but also through external interferences (window manager). In the latter case the procedure SetWindowExtents is provided.

Opening a window causes the assignment of a called default font. It is used in connection with procedures like WriteString and you may overwrite it by performing SetFont after calling OpenWindow. On the other hand you have the possibility to specify a default font by calling SetSystem (::'defaultFont',<Fontname>;) before opening a window (and all following windows; see also QueryFont).

You may set the color of graphics and font, which is used for output procedures like DispRegion or DispCircle, by calling SetRgb, SetHsi, SetGray or SetPixel. Calling SetInsert specifies how graphics is combined with the content of the image repeat memory. Thereto you may achieve by calling, e.g., SetInsert(::'not:') to eliminate the font after writing text twice at the same position.

Normally every output (e.g., DispImage, DispRegion, DispCircle, etc.) in a window is terminated by a called "flush". This causes the data to be fully visible on the display after termination of the output procedure. But this is not necessary in all cases, in particular if there are permanently output tasks or if there is a mouse procedure active. Therefore it is more favorable (i.e., more rapid) to store the data until sufficient data is available. You may stop this behavior by calling SetSystem (::'flushGraphic','false':).

The content of windows is saved (in case it is supported by special driver software); i.e., it is preserved, also if the window is hidden by other windows. But this is not necessary in all cases: If the content of a window is built up permanently new (CopyRectangle), you may suppress the security mechanism for that and hence you can save the necessary memory. This is done by calling SetSystem (::'backingStore','false':) before opening a window. In doing so you save not only memory but also time to compute. This is significant for the output of video clips (see CopyRectangle).

For graphical output (DispImage, DispRegion, etc.) you may adjust the window by calling procedure SetPart in order to represent a logical clipping of the image format. In particular this implicates that you obtain this clipping (with appropriate enlargement) of images and regions only.

**Difference: graphical window - textual window**

- Using graphical windows the layout is not as variable as concerned to textual windows.
- You may use textual windows for the input of user data only (ReadString).
- During the output of images, regions and graphics a "zooming" is performed using graphical windows: Independent on size and side ratio of the window images are transformed in that way, that they are displayed in the window by filling it completely. On the opposite side using textual windows the output does not care about the size of the window (only if clipping is necessary).
- Using graphical windows the coordinate system of the window corresponds to the coordinate system of the image format. Using textual windows, its coordinate system is always equal to the display coordinates independent on image size.

The parameter Mode determines the mode of the window. It may have following values:

**'visible'**: Normal mode for graphical windows: The window is created according to the parameters and all input and output are possible.

**'invisible'**: Invisible windows are not displayed in the display. Parameters like Row, Column and FatherWindow do not have any meaning. Output to these windows has no effect. Input (ReadString, mouse, etc.) is not possible. You may use these windows to query representation parameter for an output device without opening a (visible) window. Common queries are, e.g., QueryColor and GetStringExtents.

**'transparent'**: These windows are transparent: the window itself is not visible (edge and background), but all the other operations are possible and all output is displayed. A common use for this mode is the creation of mouse sensitive regions.
4.8. WINDOW

'buffer': These are also not visible windows. The output of images, regions, and graphics is not visible on the display, but is stored in memory. Parameters like Row, Column, and FatherWindow do not have any meaning. You may use buffer windows, if you prepare output (in the background) and copy it finally with CopyRectangle in a visible window. Another usage might be the rapid processing of image regions during interactive manipulations. Textual input and mouse interaction are not possible in this mode.

Attention
You may keep in mind that parameters as Row, Column, Width and Height are constrained by the output device. If you specify a father window (FatherWindow <> 'root') the coordinates are relative to this window.

Parameter

▷ Row (input control) .......................................................... rectangle.origin.y  ~ long / VARIANT
Row index of upper left corner.
Default Value : 0
Minimum Increment : 1
Recommended Increment : 1
Restriction : (Row ≥ 0)

▷ Column (input control) ......................................................... rectangle.origin.x  ~ long / VARIANT
Column index of upper left corner.
Default Value : 0
Minimum Increment : 1
Recommended Increment : 1
Restriction : (Column ≥ 0)

▷ Width (input control) .......................................................... rectangle.extent.x  ~ long / VARIANT
Width of the window.
Default Value : 256
Minimum Increment : 1
Recommended Increment : 1
Restriction : ((Width > 0) ∨ (Width = -1))

▷ Height (input control) ......................................................... rectangle.extent.y  ~ long / VARIANT
Height of the window.
Default Value : 256
Minimum Increment : 1
Recommended Increment : 1
Restriction : ((Height > 0) ∨ (Height = -1))

▷ FatherWindow (input control) .............................................. integer  ~ VARIANT (integer, string)
Logical number of the father window. To specify the display as father you may enter 'root' or 0.
Default Value : 0
Restriction : (FatherWindow ≥ 0)

▷ Mode (input control) ......................................................... string  ~ String / VARIANT
Window mode.
Default Value : 'visible'
List of values: Mode ∈ {'visible', 'invisible', 'transparent', 'buffer'}

▷ Machine (input control) ..................................................... string  ~ String / VARIANT
Name of the computer on which you want to open the window. Otherwise the empty string.
Default Value : ''

▷ WindowHandle (output control) ............................................. window  ~ HWindowX / VARIANT
Window identifier.

Example

open_window(0, 0, 400, -1, 'root', 'visible', '', WindowHandle)
read_image(Image, 'fabrik')
disp_image(Image, WindowHandle)
write_string(WindowHandle, 'File, fabrik.ima')
new_line(WindowHandle)
get_mbutton(WindowHandle, _, _, _)
set_lut(WindowHandle, 'temperature')
set_color(WindowHandle, 'blue')

HALCON 6.1.4
write_string(WindowHandle,'temperature')
new_line(WindowHandle)
write_string(WindowHandle,'Draw Rectangle')
new_line(WindowHandle)
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2)
set_part(Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
new_line(WindowHandle).

Result
If the values of the specified parameters are correct OpenWindow returns TRUE. If necessary an exception handling is raised.

Parallelization Information
OpenWindow is reentrant, local, and processed without parallelization.

ResetObjDb
Possible Predecessors
Possible Successors
SetColor,QueryWindowType,GetWindowType,SetWindowType,GetMposition,
SetPosition,SetTshape,SetWindowExtents,GetWindowExtents,QueryColor,SetCheck,
SetSystem
See also
DispRegion,DispImage,DispColor,SetLut,QueryColor,SetColor,SetRgb,SetHsi,
SetPixel,SetGray,SetPart,SetPartStyle,QueryWindowType,GetWindowType,
SetWindowType,GetMposition,SetTposition,SetWindowExtents,GetWindowExtents,
SetWindowAttr,SetCheck,SetSystem
Alternatives
OpenTextwindow
Module
System

[out] VARIANT WindowTypes HInfoX.QueryWindowType ( )
[out] VARIANT WindowTypes HSystemX.QueryWindowType ( )
void HOperatorSetX.QueryWindowType ([out] VARIANT WindowTypes )

Query all available window types.
QueryWindowType returns a tuple which contains all devices or software systems, respectively, which are used to display image objects. You may use QueryWindowType usefully while developing machine independent programs. Possible values are:

‘X-Window’ X-Window Version 11.
‘pixmap’ Windows are not displayed, but managed in memory. In this manner it is possible to port HALCON programs to computers without graphical display.
‘PostScript’ Objects are output to a PostScript File.
Set window characteristics.
You may use `SetWindowAttr` to set specific characteristics of graphics windows. With it you may modify the
following default parameters of a window:

'border_width' Width of the window border in pixels.
'border_color' Color of the window border.
'background_color' Background color of the window.
'window_title' Name of the window in the titlebar.

Attention
You have to call `SetWindowAttr` before calling `OpenWindow`.

Parameter

- `AttributeName` (input control) 
  Name of the attribute that should be modified.
  List of values: `AttributeName ∈ {"border_width", "border_color", "background_color", "window_title"}`

- `AttributeValue` (input control)
  Value of the attribute that should be set.
  List of values: `AttributeValue ∈ {0, 1, 2, "white", "black", "MyName", "default"}`

If the parameters are correct `SetWindowAttr` returns TRUE. If necessary an exception handling is raised.

Parallelization Information
`SetWindowAttr` is reentrant, local, and processed without parallelization.

Possible Predecessors
`OpenWindow, SetDraw, SetColor, SetColored, SetLineWidth, OpenTextwindow`

See also
`OpenWindow, GetWindowAttr`

Set the device context of a virtual graphics window (Windows NT).
`SetWindowDc` sets the device context of a window previously opened with `NewExternWindow`. All output (`DispRegion`, `DispImage`, etc.) is done in the window with this device context.

The parameter `WINHDC` contains the device context of the window in which HALCON should output its data. This device context is used in all output routines of HALCON.

Attention
The window `WindowHandle` has to be created with `NewExternWindow` beforehand.
CHAPTER 4. GRAPHICS

Parameter

- **WindowHandle** (input control) 
  window identifier.
- **WINHDC** (input control) 
  devicecontext of WINHWnd.

Restriction: \((WINHDC \neq 0)\)

Example

```plaintext
hWnd = createWINDOW(...) 
newExternWindow(hwnd, hdc, 0, 0, 400, -1, WindowHandle) 
set_device_context(WindowHandle, hdc) 
read_image(Image, 'fabrik') 
disp_image(Image, WindowHandle) 
write_string(WindowHandle, 'File, fabrik ima') 
new_line(WindowHandle) 
get_mbutton(WindowHandle, _, _, _) 
set_color(WindowHandle, 'temperature') 
write_string(WindowHandle, 'temperature') 
new_line(WindowHandle) 
write_string(WindowHandle, 'Draw Rectangle') 
new_line(WindowHandle) 
draw_rectangle1(WindowHandle, Row1, Column1, Row2, Column2) 
set_part(Row1, Column1, Row2, Column2) 
disp_image(Image, WindowHandle) 
new_line(WindowHandle) .
```

Result

If the values of the specified parameters are correct, `SetWindowDc` returns TRUE. If necessary, an exception is raised.

Parallelization Information

`SetWindowDc` is reentrant, local, and processed without parallelization.

Possible Predecessors

NewExternWindow, DispImage, DispRegion

Possible Successors

NewExternWindow, DispRegion, DispImage, DispColor, SetLut, QueryColor, SetColor, SetRgb, SetHsi, SetPixel, SetGray, SetPart, SetPartStyle, QueryWindowType, GetType, SetWindowType, GetMposition, SetPosition, SetWindowExtents, GetWindowType, SetWindowAttr, SetCheck, SetSystem

Module

```plaintext
void HWindowX.SetWindowExtents ([in] long Row, [in] long Column,  
  [in] long Width, [in] long Height )
void HOperatorSetX.SetWindowExtents ([in] VARIANT WindowHandle,  
  [in] VARIANT Height )
```

Modify position and size of a window.

`SetWindowExtents` positions the upper left corner of the output window at \((Row, Column)\) and changes the size of the window to \(Width\) and \(Height\) at the same time.

HALCON/COM Reference Manual, 2005-2-1
If you modify the size of a window an adaptation of the displayed date to the new format is not processed automatically. This has to be done by the program in performing another output of these data.

Parameter

- **WindowHandle** (input control) \( \leadsto HWindowX / \text{VARIANT} \)
  - Window identifier.

- **Row** (input control) \( \leadsto \text{long} / \text{VARIANT} \)
  - Row index of upper left corner in target position.
    - Default Value: 0
    - Minimum Increment: 1
    - Recommended Increment: 1

- **Column** (input control) \( \leadsto \text{long} / \text{VARIANT} \)
  - Column index of upper left corner in target position.
    - Default Value: 0
    - Minimum Increment: 1
    - Recommended Increment: 1

- **Width** (input control) \( \leadsto \text{long} / \text{VARIANT} \)
  - Width of the window.
    - Default Value: 512
    - Minimum Increment: 1
    - Recommended Increment: 1

- **Height** (input control) \( \leadsto \text{long} / \text{VARIANT} \)
  - Height of the window.
    - Default Value: 512
    - Minimum Increment: 1
    - Recommended Increment: 1

Result

If the window is valid and the parameters are correct, `SetWindowExtents` returns TRUE. If necessary an exception handling is raised.

Parallelization Information

`SetWindowExtents` is reentrant, local, and processed without parallelization.

Possible Predecessors

`OpenWindow, OpenTextwindow`

See also

`GetWindowExtents, OpenWindow, OpenTextwindow`

Module

`System`

```c
void HSystemX.SetWindowType ([in] String WindowType )
void HOperatorSetX.SetWindowType ([in] VARIANT WindowType )
```

Specify a window type.

`SetWindowType` determines on which type of output device the output is going to be displayed. This specification is going to be used by procedure `OpenWindow` while opening the windows. You may open different windows on different types of output devices. Therefore you have to specify the wanted type before opening. You may request the available types of output devices by calling procedure `QueryWindowType`. Possible values are:

- ’X-Window’ X-Window Version 11.
- ’WIN32-Window’ Microsoft Windows.
- ’pixmap’ Windows are not displayed, but managed in memory only. In this manner you may port HALCON programs to computers without graphical display.
- ’PostScript’ Objects are output to a PostScript File.
A useful usage of `SetWindowType` could be the development of machine independent programs.

```
Parameter

▷ WindowType (input control) .............................................. string ~ String / VARIANT
  Name of the window type which has to be set.
  Default Value: 'X-Window'
  List of values: WindowType ∈ {'X-Window', 'WIN32-Window', 'pixmap', 'PostScript'}
```

If the type of the output device is available, then `SetWindowType` returns TRUE. If necessary an exception handling is raised.

```
Possible Predecessors

OpenWindow, OpenTextwindow
```

```
See also

OpenWindow, OpenTextwindow, QueryWindowType, GetWindowType
```

```
Module

void HWindowX.SlideImage ([in] HWindowX WindowHandleSource1,
  [in] HWindowX WindowHandleSource2 )

void HOperatorSetX.SlideImage ([in] VARIANT WindowHandleSource1,
  [in] VARIANT WindowHandleSource2, [in] VARIANT WindowHandle )
```

Interactive output from two window buffers.

SlideImage divides the window horizontal in two logical areas dependent of the mouse position. The content of the first indicated window is copied in the upper area, the content of the second window is copied in the lower area. If you press the left mouse button you may scroll the delimitation between the two areas (you may move the mouse outside the window, too. In doing so the position of the mouse relative to the window determines the borderline).

Pressing the right mouse button in the window terminates the procedure SlideImage.

A useful application of procedure SlideImage might be the visualisation of the effect of a filtering operation for an image. The output is directed to the current set window (WindowHandle).

```
Attention

The three windows must have the same size and have to reside on the same computer.
```

```
Parameter

▷ WindowHandleSource1 (input control) ............................... window ~ HWindowX / VARIANT
  Logical window number of the "upper window".

▷ WindowHandleSource2 (input control) ............................... window ~ HWindowX / VARIANT
  Logical window number of the "lower window".

▷ WindowHandle (input control) ................................. window ~ HWindowX / VARIANT
  Window identifier.
```

```
Example

read_image (Image, 'fabrik')
read_image (Image, 'fabrik')
sobel_amp (Image, Amp, 'sum_abs', 3)
open_window (0, 0, -1, -1, 'root', 'buffer', 'visible', WindowHandle)
disp_image (Amp, WindowHandle)
sobel_dir (Image, Dir, 'sum_abs', 3)
open_window (0, 0, -1, -1, 'root', 'buffer', 'visible', WindowHandle)
disp_image (Dir, WindowHandle)
open_window (0, 0, -1, -1, 'root', 'visible', 'visible', WindowHandle)
slide_image (Puffer1, Puffer2, WindowHandle).
```
4.8. WINDOW

**Result**

If the both windows exist and one of these windows is valid, `SlideImage` returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

`SlideImage` is reentrant, local, and processed without parallelization.

**Possible Predecessors**

OpenWindow, OpenTextwindow

See also

OpenWindow, OpenTextwindow, MoveRectangle

**Alternatives**

CopyRectangle, GetMposition

**Module**

System
Chapter 5

Image

5.1 Access

Access the gray values of an image object.

The operator Grayval is a tuple of floating point numbers, integer respectively, which returns the gray values of several pixels of Image. The line coordinates of the pixels are in the tuple Row, the columns in Column.

Attention

The type of the values of Grayval depends on the type of the gray values. Gray values which do not belong to the image can also be accessed. The state of these gray values is not ascertained. The operator GetGrayval involves a lot of work. It is not suitable for programming image processing operations such as filters. In this case it is more useful to use the procedure GetImagePointer1 or to directly use the C interface for integrating own procedures.

Parameter

- Image (input iconic) ...... image \( \sim HImageX / IObjectX (\text{byte, direction, cyclic, int1, int2, uint2, int4, real, complex}) \)
  
  Image whose gray value is to be accessed.

- Row (input control) ................................................. point.y(-array) \( \sim \text{VARIANT (integer)} \)
  
  Line numbers of pixels to be viewed.
  
  **Default Value:** 0
  
  **Suggested values:** \( \text{Row} \in \{0, 64, 128, 256, 512, 1024\} \)
  
  **Typical range of values:** \( 0 \leq \text{Row} \leq 0(\text{lin}) \)
  
  **Minimum Increment:** 1
  
  **Recommended Increment:** 1
  
  **Restriction:** \((\text{Row} \land (\text{Image} < \text{height}))\)

- Column (input control) ................................................. point.x(-array) \( \sim \text{VARIANT (integer)} \)
  
  Column numbers of pixels to be viewed.
  
  **Default Value:** 0
  
  **Suggested values:** \( \text{Column} \in \{0, 64, 128, 256, 512, 1024\} \)
  
  **Typical range of values:** \( 0 \leq \text{Column} \leq 0(\text{lin}) \)
  
  **Minimum Increment:** 1
  
  **Recommended Increment:** 1
  
  **Restriction:** \((\text{Column} \land (\text{Image} < \text{width}))\)
  
  **Number of elements:** \((\text{Column} = \text{Row})\)

- Grayval (output control) ........................................... grayval(-array) \( \sim \text{VARIANT (integer, real)} \)
  
  Gray values of indicated pixels.
  
  **Number of elements:** \((\text{Grayval} = \text{Row})\)
CHAPTER 5. IMAGE

If the state of the parameters is correct the operator GetGrayval returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator SetSystem (‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

GetGrayval is reentrant and processed without parallelization.

Possible Predecessors

ReadImage

See also

SetGrayval

Alternatives

GetImagePointer1

Module

Image / region / XLD management

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic) ...... image $\sim$ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)</td>
</tr>
<tr>
<td><strong>Pointer</strong> (output control) .............................................. pointer $\sim$ long / VARIANT Pointer to the image data in the HALCON databank.</td>
</tr>
<tr>
<td><strong>Type</strong> (output control) ...................................................... string $\sim$ String / VARIANT Type of image.</td>
</tr>
<tr>
<td><strong>Width</strong> (output control) .................................................. extent.x $\sim$ long / VARIANT Width of image.</td>
</tr>
<tr>
<td><strong>Height</strong> (output control) .................................................. extent.y $\sim$ long / VARIANT Height of image.</td>
</tr>
</tbody>
</table>

**Result**

The operator GetImagePointer1 returns the value TRUE if exactly one image was passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem (‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

GetImagePointer1 is reentrant and processed without parallelization.

Possible Predecessors

ReadImage

Access the pointer of a channel.

The operator GetImagePointer1 returns a C pointer to the first channel of the image Image. Additionally the image type (Type $=$ ‘byte’, ‘integer’, ‘float’ etc.) and the image size (width and height) are returned. Consequently a direct access to the image data in the HALCON databank from the HALCON host language via the pointer is possible. An image is stored in HALCON as a vector of image lines.

Attention

The operator GetImagePointer1 should only be used for entry into newly created images, since otherwise the gray values of other images might be overwritten (see relational structure).

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic) ...... image $\sim$ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)</td>
</tr>
<tr>
<td><strong>Pointer</strong> (output control) .............................................. pointer $\sim$ long / VARIANT Pointer to the image data in the HALCON databank.</td>
</tr>
<tr>
<td><strong>Type</strong> (output control) ...................................................... string $\sim$ String / VARIANT Type of image.</td>
</tr>
<tr>
<td><strong>Width</strong> (output control) .................................................. extent.x $\sim$ long / VARIANT Width of image.</td>
</tr>
<tr>
<td><strong>Height</strong> (output control) .................................................. extent.y $\sim$ long / VARIANT Height of image.</td>
</tr>
</tbody>
</table>

**Result**

The operator GetImagePointer1 returns the value TRUE if exactly one image was passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem (‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

GetImagePointer1 is reentrant and processed without parallelization.

Possible Predecessors

ReadImage
5.1. ACCESS

See also PaintRegion, PaintGray

Alternatives SetGrayval, GetGrayval, GetImagePointer3

Image / region / XLD management

Access to the image data pointer and the image data inside the smallest rectangle of the domain of the input image. The operator GetImagePointer1Rect returns the pointer PixelPointer which points to the beginning of the image data inside the smallest rectangle of the domain of Image. VerticalPitch corresponds to the width of the input image Image multiplied with the number of bytes per pixel (HorizontalBitPitch/8). Width and Height correspond to the size of the smallest rectangle of the input region. HorizontalBitPitch is the horizontal distance (in bits) between two neighbouring pixels. BitsPerPixel is the number of used bits per pixel. GetImagePointer1Rect is symmetrical to GenImage1Rect.

Attention The operator GetImagePointer1Rect should only be used for entry into newly created images, since otherwise the gray values of other images might be overwritten (see relational structure).

Parameter

- **Image** (input iconic) image ~ HImageX / IHOBJECTX (byte, uint2, int4) Input image (Himage).
- **PixelPointer** (output control) pointer ~ long / VARIANT Pointer to the image data.
- **Width** (output control) extent.x ~ long / VARIANT Width of the output image.
- **Height** (output control) extent.y ~ long / VARIANT Height of the output image.
- **VerticalPitch** (output control) Width(input image)*(HorizontalBitPitch/8). integer ~ long / VARIANT Distance between two neighbouring pixels in bits.
- **HorizontalBitPitch** (output control) integer ~ long / VARIANT Distance between two neighbouring pixels in bits. Default Value: 8 List of values: HorizontalBitPitch ∈ {8, 16, 32}
- **BitsPerPixel** (output control) integer ~ long / VARIANT Number of used bits per pixel. Default Value: 8 List of values: BitsPerPixel ∈ {8, 16, 32}

Result

The operator GetImagePointer1Rect returns the value TRUE if exactly one image was passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem (‘noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

GetImagePointer1Rect is reentrant and processed without parallelization.

Possible Predecessors

ReadImage, GenImage1Rect
Access the pointers of a colored image.

The operator `GetImagePointer3` returns a C pointer to the three channels of a colored image (`ImageRGB`). Additionally the image type (`Type = 'byte', 'int2', 'float' etc.`) and the image size (`Width` and `Height`) are returned. Consequently a direct access to the image data in the HALCON databank from the HALCON host language via the pointer is possible. An image is stored in HALCON as a vector of image lines. The three channels must have the same pixel type and the same size.

**Attention**

Only one image can be passed. The operator `GetImagePointer3` should only be used for entry into newly created images, since otherwise the gray values of other images might be overwritten (see relational structure).

### Parameter

- **ImageRGB** (input iconic) ...... image \(\sim\) `HImageX / IHObjectX (byte, direction, cyclic, int1, uint2, int4, real, complex, dvl)`
  - Input image.
- **PointerRed** (output control) .......................................................... pointer \(\sim\) long / VARIANT
  - Pointer to the pixels of the first channel.
- **PointerGreen** (output control) .......................................................... pointer \(\sim\) long / VARIANT
  - Pointer to the pixels of the second channel.
- **PointerBlue** (output control) ............................................................ pointer \(\sim\) long / VARIANT
  - Pointer to the pixels of the third channel.
- **Type** (output control) ................................................................. string \(\sim\) String / VARIANT
  - Type of image.
  - **List of values:** `Type \in \{'int1', 'int2', 'uint2', 'int4', 'byte', 'real', 'direction', 'cyclic', 'complex', 'dvl', 'lut'\}`
- **Width** (output control) ............................................................... extent.x \(\sim\) long / VARIANT
  - Width of image.
- **Height** (output control) .............................................................. extent.y \(\sim\) long / VARIANT
  - Height of image.

### Result

The operator `GetImagePointer3` returns the value TRUE if exactly one image is passed. The behavior in case of empty input (no input images available) is set via the operator `SetSystem ('noObjectResult',<Result>). If necessary an exception handling is raised.

**Parallelization Information**

`GetImagePointer3` is reentrant and processed without parallelization.

**Possible Predecessors**

- `ReadImage`

**See also**

- `PaintRegion`, `PaintGray`, `GenImage1Rect`
- `SetGrayval`, `GetGrayval`, `GetImagePointer3`, `GetImagePointer1`
- `SetSystem` ('noObjectResult',<Result>)
5.1. ACCESS

Alternatives

SetGrayval, GetGrayval, GetImagePointer

Module

Image / region / XLD management

[out] long HImageX.GetImageTime ([out] long Second,
[out] long Minute, [out] long Hour, [out] long Day, [out] long YDay,
[out] long Month, [out] long Year )

void HOperatorSetX.GetImageTime ([in] IHObjectX Image,
[out] VARIANT MSecond, [out] VARIANT Second, [out] VARIANT Minute,
[out] VARIANT Hour, [out] VARIANT Day, [out] VARIANT YDay,
[out] VARIANT Month, [out] VARIANT Year )

Request time at which the image was created.

The operator GetImageTime returns the time at which the image was created.

Parameter

▷ Image (input iconic) ...... image 〜 HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4,
real, complex, dvf )

Input image.

▷ MSecond (output control) ........................................ integer 〜 long / VARIANT
Milliseconds (0..999).

▷ Second (output control) ............................................ integer 〜 long / VARIANT
Seconds (0..59).

▷ Minute (output control) ............................................ integer 〜 long / VARIANT
Minutes (0..59).

▷ Hour (output control) ............................................. integer 〜 long / VARIANT
Hours (0..11).

▷ Day (output control) ............................................. integer 〜 long / VARIANT
Day of the month (1..31).

▷ YDay (output control) ........................................... integer 〜 long / VARIANT
Day of the year (1..365).

▷ Month (output control) ......................................... integer 〜 long / VARIANT
Month (1..12).

▷ Year (output control) ............................................ integer 〜 long / VARIANT
Year (xxxx).

Result

The operator GetImageTime returns the value TRUE if exactly one image was passed. The behavior in case of empty input (no input images available) is set via the operator SetSystem ('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

GetImageTime is reentrant, local, and processed without parallelization.

Possible Predecessors

ReadImage, GrabImage

See also

CountSeconds

Module

Image / region / XLD management

HALCON 6.1.4
5.2 Channel

<table>
<thead>
<tr>
<th>[out] HImageX Image</th>
<th>HImageX.AccessChannel ([in] long Channel )</th>
</tr>
</thead>
</table>

Access a channel of a multichannel image.

The operator AccessChannel accesses a channel of the (multichannel) input image. The result is a one-channel image. The definition domain of the input is adopted. The channels are numbered from 1 to n. The number of channels can be determined via the operator CountChannels.

Parameter

- **MultiChannelImage** (input iconic) ...... multichannel-image-array \(\sim\) HImageX / IOBJECTX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVF )
  - Multichannel image.
- **Image** (output iconic) ...... image \(\sim\) HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVF )
  - One channel of MultiChannelImage.
- **Channel** (input control) ................................. channel \(\sim\) long / VARIANT
  - Index of channel to be accessed.
  - Default Value : 1
  - Suggested values : Channel \(\in\) \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}

Parallelization Information

AccessChannel is reentrant and processed without parallelization.

Possible Predecessors

- CountChannels

Possible Successors

- DispImage
- CountChannels

See also

Alternatives

- Decompose2, Decompose3, Decompose4, Decompose5

Module

Image / region / XLD management

<table>
<thead>
<tr>
<th>[out] HImageX ImageExtended</th>
<th>HImageX.AppendChannel ([in] HImageX Image )</th>
</tr>
</thead>
</table>

Append additional matrices (channels) to the image.

The operator AppendChannel appends the matrices of the image Image to the matrices of MultiChannelImage. The result is an image containing as many matrices (channels) as MultiChannelImage and Image combined. The definition domain of the output image is calculated as the average of the definition domains of both input images.

Parameter

- **MultiChannelImage** (input iconic) ...... image \(\sim\) HImageX / IOBJECTX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVF )
  - Multichannel image.
- **Image** (input iconic) ...... image \(\sim\) HImageX / IOBJECTX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVF )
  - Image to be appended.

HALCON/COM Reference Manual, 2005-2-1
5.2. CHANNEL

ImageExtended (output iconic) ...... multichannel-image-array ∼ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Image appended by Image.

Parallelization Information

AppendChannel is reentrant and processed without parallelization.

Possible Successors

DispImage

Alternatives

Compose2, Compose3, Compose4, Compose5

Module

Image / region / XLD management

[out] HImageX MultiChannelImage HImageX.ChannelsToImage ( )

void HOperatorSetX.ChannelsToImage ([in] IOBJECTX Images,
[out] HUntypedObjectX MultiChannelImage )

Convert one-channel images into a multichannel image

The operator ChannelsToImage converts several one-channel images into a multichannel image. The new definition domain is the average of the definition domains of the input images.

Parameter

▷ Images (input iconic) ...... image ∼ HImageX / IOBJECTX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

One-channel images to be combined into a one-channel image.

▷ MultiChannelImage (output iconic) ...... multichannel-image-array ∼ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Multichannel image.

Parallelization Information

ChannelsToImage is reentrant, local, and processed without parallelization.

Possible Successors

CountChannels, DispImage

Module

Image / region / XLD management

[out] HImageX MultiChannelImage HImageX.Compose2 ([in] HImageX Image2 )

void HOperatorSetX.Compose2 ([in] IOBJECTX Image1,
[in] IOBJECTX Image2, [out] HUntypedObjectX MultiChannelImage )

Convert two images into a two-channel image.

The operator Compose2 converts 2 one-channel images into a 2-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

Parameter

▷ Image1 (input iconic) ...... image(-array) ∼ HImageX / IOBJECTX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Input image 1.

▷ Image2 (input iconic) ...... image(-array) ∼ HImageX / IOBJECTX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Input image 2.
**MultiChannelImage** (output iconic) ...... multichannel-image-array \(\sim\) HImageX / HUntypedObjectX

Multichannel image.

---

**Parallelization Information**

**Compose2** is reentrant and automatically parallelized (on tuple level).

---

**Possible Successors**

DispImage

---

See also

Decompose2

---

Alternatives

AppendChannel

---

Module

Image / region / XLD management

---

Convert 3 images into a three-channel image.

The operator **Compose3** converts 3 one-channel images into a 3-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

---

**Parameter**

- **Image1** (input iconic) ...... image-array \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Input image 1.

- **Image2** (input iconic) ...... image-array \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Input image 2.

- **Image3** (input iconic) ...... image-array \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Input image 3.

- **MultiChannelImage** (output iconic) ...... multichannel-image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Multichannel image.

---

**Parallelization Information**

**Compose3** is reentrant and automatically parallelized (on tuple level).

---

**Possible Successors**

DispImage

---

See also

Decompose3

---

Alternatives

AppendChannel

---

Module

Image / region / XLD management
Convert 4 images into a four-channel image.

The operator \texttt{Compose4} converts 4 one-channel images into a 4-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

\begin{verbatim}
\end{verbatim}

Parameter

\begin{itemize}
  \item \texttt{Image1} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image2} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image3} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image4} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
\end{itemize}

Multichannel image.

Parallelization Information

\texttt{Compose4} is reentrant and automatically parallelized (on tuple level).

Possible Successors

\texttt{DispImage}

See also

\texttt{Decompose4}

Alternatives

\texttt{AppendChannel}

Module

Image / region / XLD management

Convert 5 images into a five-channel image.

The operator \texttt{Compose5} converts 5 one-channel images into a 5-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

\begin{verbatim}
\end{verbatim}

Parameter

\begin{itemize}
  \item \texttt{Image1} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image2} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image3} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image4} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  \item \texttt{Image5} (input iconic) \ldots \ldots image(-array) \Rightarrow HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
\end{itemize}

HALCON 6.1.4
CHAPTER 5. IMAGE

- **Image2** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 2.

- **Image3** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 3.

- **Image4** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 4.

- **Image5** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 5.

- **MultiChannelImage** (output iconic) ...... multichannel-image-array  \(\sim\)  \(HImageX / HUntypedObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Multichannel image.

---

**Parallelization Information**

*Compose5* is reentrant and automatically parallelized (on tuple level).

---

**Possible Successors**

- **DispImage**
- **Decompose5**

---

**Alternatives**

- **AppendChannel**

---

**Module**

Image / region / XLD management

---

```plaintext
```

Convert 6 images into a six-channel image.

The operator **Compose6** converts 6 one-channel images into a 6-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

---

**Parameter**

- **Image1** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 1.

- **Image2** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 2.

- **Image3** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 3.

- **Image4** (input iconic) ...... image(-array)  \(\sim\)  \(HImageX / IObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image 4.
5.2 CHANNEL

- **Image5** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  
  Input image 5.

- **Image6** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  
  Input image 6.

- **MultiChannelImage** (output iconic) ...... multichannel-image-array ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, int4, real, complex, dVf)

  Multichannel image.

  **Parallelization Information**

  **Compose6** is reentrant and automatically parallelized (on tuple level).

  **Possible Successors**

  **DispImage**

  **See also** **Decompose6**

  **Alternatives**

  **AppendChannel**

  **Module**

  Image / region / XLD management

  Convert 7 images into a seven-channel image.

  The operator **Compose7** converts 7 one-channel images into a 7-channel image. The definition domain is calculated as the intersection of the definition domains of the input images.

  **Parameter**

  - **Image1** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
    
    Input image 1.

  - **Image2** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
    
    Input image 2.

  - **Image3** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
    
    Input image 3.

  - **Image4** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
    
    Input image 4.

  - **Image5** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
    
    Input image 5.

  - **Image6** (input iconic) ...... image(-array) ~ HImageX / IOObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
    
    Input image 6.
Image7 (input iconic) ...... image(array) ～ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Input image 7.

MultiChannelImage (output iconic) ...... multichannel-image-array ～ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Multichannel image.

Compose7 is reentrant and automatically parallelized (on tuple level).

Possible Successors

DispImage

Decompose7

See also

Alternatives

AppendChannel

Module

Image / region / XLD management

--------- Parallelization Information

CountChannels is reentrant and automatically parallelized (on tuple level).

--------- Parameter

MultiChannelImage (input iconic) ...... (multichannel-)image(array) ～ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

One- or multichannel image.

Channels (output control) ......................... integer(array) ～ VARIANT(integer)

Number of channels.

--------- Parallelization Information

CountChannels is reentrant and processed without parallelization.

--------- Possible Successors

AccessChannel, AppendChannel, DispImage

AppendChannel, AccessChannel

--------- Module

Image / region / XLD management

--------- Parameters

[out] VARIANT Channels HImageX.CountChannels ( )

void HOperatorSetX.CountChannels ([in] IHObjectX MultiChannelImage, [out] VARIANT Channels )

Count channels of image.
The operator CountChannels counts the number of channels of all input images.

--------- Parallelization Information

Decompose2 is reentrant and automatically parallelized (on tuple level).

--------- Possible Successors

AccessChannel, AppendChannel, DispImage

AppendChannel, AccessChannel

--------- Module

Image / region / XLD management

--------- Parameters

[out] HImageX Image1 HImageX.Decompose2 ([out] HImageX Image2 )

void HOperatorSetX.Decompose2 ([in] IHObjectX MultiChannelImage, [out] HUntypedObjectX Image1, [out] HUntypedObjectX Image2 )

Convert a two-channel image into two images.
The operator Decompose2 converts a 2-channel image into two one-channel images with the same definition domain.
5.2. CHANNEL

Parameter

- **MultiChannelImage** (input iconic) ...... multichannel-image-array ~ HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Multichannel image.

- **Image1** (output iconic) ...... image(-array) ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Output image 1.

- **Image2** (output iconic) ...... image(-array) ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Output image 2.

Parallelization Information

Decompose2 is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

**CountChannels**

Possible Successors

**DispImage**

See also

**Compose2**

Alternatives

**AccessChannel, ImageToChannels**

Module

Image / region / XLD management

---

Convert a three-channel image into three images.

The operator **Decompose3** converts a 3-channel image into three one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) ...... multichannel-image-array ~ HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Multichannel image.

- **Image1** (output iconic) ...... image(-array) ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Output image 1.

- **Image2** (output iconic) ...... image(-array) ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Output image 2.

- **Image3** (output iconic) ...... image(-array) ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

  Output image 3.

Parallelization Information

Decompose3 is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

**CountChannels**

---
Convert a four-channel image into four images.

The operator `Decompose4` converts a 4-channel image into four one-channel images with the same definition domain.

**Parameter**

- **MultiChannelImage** (input iconic) ...... multichannel-image-array \(\sim \) `HImageX / IHObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Multichannel image.

- **Image1** (output iconic) ...... image(-array) \(\sim \) `HImageX / HUntypedObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Output image 1.

- **Image2** (output iconic) ...... image(-array) \(\sim \) `HImageX / HUntypedObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Output image 2.

- **Image3** (output iconic) ...... image(-array) \(\sim \) `HImageX / HUntypedObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Output image 3.

- **Image4** (output iconic) ...... image(-array) \(\sim \) `HImageX / HUntypedObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Output image 4.

**Parallelization Information**

`Decompose4` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- `CountChannels`

**Possible Successors**

- `DispImage`

**See also**

- `Compose4`

**Alternatives**

- `AccessChannel`, `ImageToChannels`

**Module**

Image / region / XLD management
Convert a five-channel image into five images.

The operator `Decompose5` converts a 5-channel image into five one-channel images with the same definition domain.

Parameter

- **MultiChannelImage** (input iconic) . . . . . . multichannel-image-array . . . . . . multichannel-image-array
  - `HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)`
  - Multichannel image.

- **Image1** (output iconic) . . . . . . image(-array) . . . . . . image(-array)
  - `HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)`
  - Output image 1.

- **Image2** (output iconic) . . . . . . image(-array) . . . . . . image(-array)
  - `HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)`
  - Output image 2.

- **Image3** (output iconic) . . . . . . image(-array) . . . . . . image(-array)
  - `HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)`
  - Output image 3.

- **Image4** (output iconic) . . . . . . image(-array) . . . . . . image(-array)
  - `HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)`
  - Output image 4.

- **Image5** (output iconic) . . . . . . image(-array) . . . . . . image(-array)
  - `HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)`
  - Output image 5.

Parallelization Information

`Decompose5` is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- `CountChannels`

Possible Successors

- `DispImage`

See also

- `Compose5`

Alternatives

- `AccessChannel`, `ImageToChannels`

Module

Image / region / XLD management
The operator Decompose6 converts a 6-channel image into six one-channel images with the same definition domain.

**Parameter**

- **MultiChannelImage** (input iconic) ...... multichannel-image-array \(\sim\) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Multichannel image.

- **Image1** (output iconic) ...... image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Output image 1.

- **Image2** (output iconic) ...... image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Output image 2.

- **Image3** (output iconic) ...... image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Output image 3.

- **Image4** (output iconic) ...... image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Output image 4.

- **Image5** (output iconic) ...... image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Output image 5.

- **Image6** (output iconic) ...... image-array \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dfv)

  Output image 6.

**Parallelization Information**

Decompose6 is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

CountChannels

**Possible Successors**

DispImage

**See also**

Compose6

**Alternatives**

AccessChannel, ImageToChannels

**Module**

Image / region / XLD management

```
[out] HImageX Image1, HImageX Decompose7 ([out] HImageX Image2,
[out] HImageX Image3, [out] HImageX Image4, [out] HImageX Image5,
[out] HImageX Image6, [out] HImageX Image7)

void HOperatorSetX.Decompose7 ([in] IObjectX MultiChannelImage,
[out] HUntypedObjectX Image1, [out] HUntypedObjectX Image2,
[out] HUntypedObjectX Image3, [out] HUntypedObjectX Image4,
[out] HUntypedObjectX Image5, [out] HUntypedObjectX Image6,
[out] HUntypedObjectX Image7)
```

Convert a seven-channel image into seven images.

The operator Decompose7 converts a 7-channel image into seven one-channel images with the same definition domain.
5.2. CHANNEL

Parameter

» MultiChannelImage (input iconic) ...... multichannel-image-array \(\sim\) HImageX / IHOBJECTX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Multichannel image.

» Image1 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 1.

» Image2 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 2.

» Image3 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 3.

» Image4 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 4.

» Image5 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 5.

» Image6 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 6.

» Image7 (output iconic) ...... image(-array) \(\sim\) HImageX / HUnTypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Output image 7.

Parallelization Information

Decompose7 is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

CountChannels

Possible Successors

DispImage

See also

Compose7

Alternatives

AccessChannel, ImageToChannels

Module

Image / region / XLD management

[out] HImageX Images HImageX.ImageToChannels ( )

void HOperatorSetX.ImageToChannels ([in] IHOBJECTX MultiChannelImage, [out] HUntypedObjectX Images )

Convert a multichannel image into One-channel images

The operator ImageToChannels generates a one-channel image for each channel of the multichannel image in MultiChannelImage. The definition domains are adopted from the input image. As many images are created as MultiChannelImage has channels.

Parameter

» MultiChannelImage (input iconic) ...... multichannel-image-array \(\sim\) HImageX / IHOBJECTX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)

Multichannel image to be decomposed.
Generated one-channel images.

Parallelization Information

ImageToChannels is reentrant and processed without parallelization.

Possible Predecessors

CountChannels

Possible Successors

DispImage

Alternatives

AccessChannel, Decompose2, Decompose3, Decompose4, Decompose5

Module

Image / region / XLD management

5.3 Creation

```
[out] HImageX DupImage HImageX.CopyImage ( )
void HOperatorSetX.CopyImage ([in] IHObjectX Image,
[out] HUntypedObjectX DupImage )
```

Copy an image and allocate new memory for it.

CopyImage copies the first channel of the input image into a new (single-channel) image with the same domain as the input image. In contrast to HALCON operators such as CopyObj, the image is copied into a newly allocated memory segment. This can be used to modify the gray values of the new image, for example (see GetImagePointer1).

Parameter

▷ Image (input iconic) ...... image  \(\sim\) HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Image to be copied.

▷ DupImage (output iconic) ...... image  \(\sim\) HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Copied image.

Parallelization Information

CopyImage is reentrant and processed without parallelization.

Possible Predecessors

ReadImage, GenImageConst

Possible Successors

SetGrayval, GetImagePointer1

See also

GetImagePointer1

Alternatives

SetGrayval, PaintGray, GenImageConst, GenImageProto

Module

Basic operators

```
void HImageX.GenImage1 ([in] String Type, [in] long Width,
[in] long Height, [in] long PixelPointer )

void HOperatorSetX.GenImage1 ([out] HUntypedObjectX Image,
[in] VARIANT Type, [in] VARIANT Width, [in] VARIANT Height,
[in] VARIANT PixelPointer )
```

Create an image from a pointer to the pixels.
The operator `GenImage1` creates an image of the size \( \text{Width} \times \text{Height} \). The pixels in `PixelPointer` are stored line-sequentially. The type of the given pixels (`PixelPointer`) must correspond to `Type`. The storage for the new image is newly created by HALCON. Thus, the storage on the `PixelPointer` can be released after the call. Since the type of the parameter `PixelPointer` is generic (long) a cast has to be used for the call.

---

**Parameter**

- **Image** (output iconic) ….. image \( \sim \) `HImageX / HUntypedObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real)

  Created image with new image matrix.

- **Type** (input control) ………………………………………………………..string \( \sim \) `String / VARIANT`

  Pixel type.
  
  - Default Value : ’byte’
  - List of values : `Type \in \{’byte’, ’direction’, ’cyclic’, ’int1’, ’int2’, ’uint2’, ’int4’, ’real’\}`

- **Width** (input control) ………………………………………………………..extent.x \( \sim \) `long / VARIANT`

  Width of image.
  
  - Default Value : 512
  - Suggested values : `Width \in \{128, 256, 512, 1024\}`
  - Typical range of values : \(1 \leq \text{Width} \leq 1\) (lin)
  - Minimum Increment : 1
  - Recommended Increment : 10
  - Restriction : \((\text{Width} \geq 1)\)

- **Height** (input control) ………………………………………………………..extent.y \( \sim \) `long / VARIANT`

  Height of image.
  
  - Default Value : 512
  - Suggested values : `Height \in \{128, 256, 512, 1024\}`
  - Typical range of values : \(1 \leq \text{Height} \leq 1\) (lin)
  - Minimum Increment : 1
  - Recommended Increment : 10
  - Restriction : \((\text{Height} \geq 1)\)

- **PixelPointer** (input control) ……………………………………………………integer \( \sim \) `long / VARIANT`

  Pointer to first gray value.

---

**Result**

If the parameter values are correct, the operator `GenImage1` returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

`GenImage1` is reentrant and processed without parallelization.

---

**Possible Predecessors**

`GenImageConst, GetImagePointer1`

---

See also

`ReduceDomain, PaintGray, PaintRegion, SetGrayval`

---

**Alternatives**

`GenImage3, GenImageConst, GetImagePointer1`

---

**Module**

Image / region / XLD management

---

```
void HImageX.GenImage1Extern ([in] String Type, [in] long Width,

void HOperatorSetX.GenImage1Extern ([out] HUntypedObjectX Image,
[in] VARIANT Type, [in] VARIANT Width, [in] VARIANT Height,
```

Create an image from a pointer on the pixels with storage management.
The operator `GenImage1Extern` creates an image of the size `Width × Height`. The pixels in `PixelPointer` are stored line-sequentially. The type of the given pixels (`PixelPointer`) must correspond to `Type`. Since the type of the parameter `PixelPointer` is generic (long) a cast must be used for the call.

The memory for the new image is not newly allocated by HALCON, contrary to `GenImage1`, and thus is not copied either. This means that the memory space that `PixelPointer` points to must be released by deleting the object `Image`. This is done by the procedure `ClearProc` provided by the caller. This procedure must have the following signature:

```c
void ClearProc(void* ptr);
```

It is called when deleting `Image`. If the memory shall not be released (in the case of frame grabbers or static memory) a procedure “without trunk” or the NULL-Pointer can be passed. Analogous to the parameter `PixelPointer` the pointer has to be passed to the procedure by casting it to long.

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Suggested values</th>
<th>Typical range of values</th>
<th>Minimum Increment</th>
<th>Recommended Increment</th>
<th>Restriction</th>
</tr>
</thead>
</table>
| `Image` (output iconic) | image

Created HALCON image.

| `Type` (input control) | string

Pixel type.

Default Value: 'byte'


| `Width` (input control) | long

Width of image.

Default Value: 512

Suggested values: `Width ∈ {128, 256, 512, 1024}`

Typical range of values: `1 ≤ Width ≤ 1(lin)`

Minimum Increment: 1

Recommended Increment: 10

Restriction: `(Width ≥ 1)`

| `Height` (input control) | long

Height of image.

Default Value: 512

Suggested values: `Height ∈ {128, 256, 512, 1024}`

Typical range of values: `1 ≤ Height ≤ 1(lin)`

Minimum Increment: 1

Recommended Increment: 10

Restriction: `(Height ≥ 1)`

| `PixelPointer` (input control) | integer

Pointer to the first gray value.

| `ClearProc` (input control) | integer

Pointer to the procedure re-releasing the memory of the image when deleting the object.

Default Value: 0

### Result

The operator `GenImage1Extern` returns the value TRUE if the parameter values are correct. Otherwise an exception handling is raised.

### Parallelization Information

`GenImage1Extern` is reentrant and processed without parallelization.

### See also

`ReduceDomain, PaintGray, PaintRegion, SetGrayval`

### Alternatives

`GenImage1, GenImageConst, GetImagePointer1`

### Module

Image / region / XLD management
Create an image with a rectangular domain from a pointer on the pixels (with storage management).

The operator `GenImage1Rect` creates an image of size \((\text{VerticalPitch}/(\text{HorizontalBitPitch} / 8)) \times \text{Height}\). The pixels pointed to by `PixelPointer` are stored line by line. Since the type of the parameter `PixelPointer` is generic (long) a cast must be used for the call. `VerticalPitch` determines the distance (in bytes) between pixel \(m\) in row \(n\) and pixel \(m\) in row \(n+1\) inside of memory. All rows of the 'input image' have the same vertical pitch. The width of the output image equals \(\text{VerticalPitch} / (\text{HorizontalBitPitch} / 8)\). The height of input and output image are equal. The domain of the output image `Image` is a rectangle of the size \(\text{Width} \times \text{Height}\). The parameter `HorizontalBitPitch` is the horizontal distance (in bits) between two neighbouring pixels. `BitsPerPixel` is the number of used bits per pixel.

If `DoCopy` is set 'true', the image data pointed to by `PixelPointer` is copied and memory for the new image is newly allocated by HALCON. Else the image data is not duplicated and the memory space that points to must be released when deleting the object `Image`. This is done by the procedure `ClearProc` provided by the caller. This procedure must have the following signature:

```c
void ClearProc(void* ptr);
```

It is called when deleting `Image`. If the memory shall not be released (in the case of frame grabbers or static memory) a procedure "without trunk" or the NULL-pointer can be passed. Analogously to the parameter `PixelPointer` the pointer has to be passed to the procedure by casting it to long. If `DoCopy` is 'true' then `ClearProc` is irrelevant. The operator `GenImage1Rect` is symmetrical to `GetImagePointer1Rect`.

**Parameter**

- **Image** (output iconic) ................. image \(\sim \text{HImageX} / \text{HUntypedObjectX} (\text{byte, uint2, int4})\)
  
  Created HALCON image.

- **PixelPointer** (input control) .................................................... integer \(\sim\) long / VARIANT
  
  Pointer to the first pixel.

- **Width** (input control) ......................................................... extent.x \(\sim\) long / VARIANT
  
  Width of the image.

  **Default Value**: 512

  **Suggested values**: \(\text{Width} \in \{128, 256, 512, 1024\}\)

  **Typical range of values**: \(1 \leq \text{Width} \leq 1\text{(lin)}\)

  **Minimum Increment**: 1

  **Recommended Increment**: 10

  **Restriction**: \((\text{Width} \geq 1)\)

- **Height** (input control) ......................................................... extent.y \(\sim\) long / VARIANT
  
  Height of the image.

  **Default Value**: 512

  **Suggested values**: \(\text{Height} \in \{128, 256, 512, 1024\}\)

  **Typical range of values**: \(1 \leq \text{Height} \leq 1\text{(lin)}\)

  **Minimum Increment**: 1

  **Recommended Increment**: 10

  **Restriction**: \((\text{Height} \geq 1)\)

- **VerticalPitch** (input control) .................................................. integer \(\sim\) long / VARIANT
  
  Distance (in bytes) between pixel \(m\) in row \(n\) and pixel \(m\) in row \(n+1\) of the 'input image'.

  **Restriction**: \((\text{VerticalPitch} \geq (\text{Width} \cdot (\text{HorizontalBitPitch}/8)))\)

- **HorizontalBitPitch** (input control) ........................................... integer \(\sim\) long / VARIANT
  
  Distance between two neighbouring pixels in bits.

  **Default Value**: 8

  **List of values**: \(\text{HorizontalBitPitch} \in \{8, 16, 32\}\)
BitsPerPixel (input control) ........................................ integer  \sim long / VARIANT
Number of used bits per pixel.
   Default Value : 8
   List of values: BitsPerPixel \in \{8, 9, 10, 11, 12, 13, 14, 15, 16, 32\}
   Restriction : (BitsPerPixel \leq HorizontalBitPitch)

DoCopy (input control) ............................................ string  \sim String / VARIANT
Copy image data.
   Default Value : 'false'
   Suggested values : DoCopy \in \{ 'true', 'false' \}

ClearProc (input control) ........................................ integer  \sim long / VARIANT
Pointer to the procedure releasing the memory of the image when deleting the object.
   Default Value : 0

-------------------- Result ---------------------
The operator GenImage1Rect returns the value TRUE if the parameter values are correct. Otherwise an exception handling is raised.

-------------------- Parallelization Information ---------------------
GenImage1Rect is reentrant and processed without parallelization.

-------------------- Possible Successors ---------------------
GetImagePointer1Rect
See also

-------------------- Alternatives ---------------------
GenImage1, GenImage1Extern

Module

Image / region / XLD management

```
void HImageX.GenImage3 ([in] String Type, [in] long Width,
[in] long PixelPointerBlue )

void HOperatorSetX.GenImage3 ([out] HUntypedObjectX ImageRGB,
[in] VARIANT Type, [in] VARIANT Width, [in] VARIANT Height,
[in] VARIANT PixelPointerRed, [in] VARIANT PixelPointerGreen,
[in] VARIANT PixelPointerBlue )
```

Create an image from three pointers to the pixels (red/green/blue).
The operator GenImage3 creates a three-channel image of the size Width \times Height. The pixels in PixelPointerRed, PixelPointerGreen and PixelPointerBlue are stored line-sequentially. The type of the given pixels (PixelPointerRed etc.) must correspond to the name of the pixels (Type). The storage for the new image is newly created by HALCON. Thus, it can be released after the call. Since the type of the parameters (PixelPointerRed etc.) is generic (long) a “cast” must be used for the call.

-------------------- Attention ---------------------

-------------------- Parameter ---------------------

ImageRGB (output iconic) ...... image  \sim HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real )
Created image with new image matrix.

Type (input control) ............................................. string  \sim String / VARIANT
Pixel type.
   Default Value : 'byte'
   List of values: Type \in \{ 'byte', 'direction', 'cyclic', 'int1', 'int2', 'uint2', 'int4', 'real' \}
5.3. CREATION

- **Width** (input control) 
  Width of image.
  
  Default Value: 512
  
  Suggested values: \( \text{Width} \in \{128, 256, 512, 1024\} \)
  
  Typical range of values: \( 1 \leq \text{Width} \leq 1(\text{lin}) \)
  
  Minimum Increment: 1
  
  Recommended Increment: 10

- **Height** (input control) 
  Height of image.
  
  Default Value: 512
  
  Suggested values: \( \text{Height} \in \{128, 256, 512, 1024\} \)
  
  Typical range of values: \( 1 \leq \text{Height} \leq 1(\text{lin}) \)
  
  Minimum Increment: 1
  
  Recommended Increment: 10

- **PixelPointerRed** (input control) 
  Pointer to first red value (channel 1).

- **PixelPointerGreen** (input control) 
  Pointer to first green value (channel 2).

- **PixelPointerBlue** (input control) 
  Pointer to first blue value (channel 3).

Result

If the parameter values are correct, the operator **GenImage3** returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

**GenImage3** is reentrant and processed without parallelization.

Possible Predecessors

**GenImageConst**, **GetImagePointer1**

Possible Successors

**DispColor**

See also

**ReduceDomain**, **PaintGray**, **PaintRegion**, **SetGrayval**, **GetImagePointer1**, **Decompose3**

Alternatives

**GenImage1**, **Compose3**, **GenImageConst**

Module

Image / region / XLD management

```c
void HImageX.GenImageConst ([in] String Type, [in] long Width, [in] long Height )

```

Create an image with constant gray value.

The operator **GenImageConst** creates an image of the indicated size. The height and width of the image are determined by **Height** and **Width**. HALCON supports the following image types:

- **'byte'** 1 byte per pixel (0..255)
- **'int1'** 1 byte per pixel (-127..127)
- **'int2'** 2 bytes per pixel (-32767..32767)
- **'uint2'** 2 bytes per pixel (0..65535)
- **'int4'** 4 bytes per pixel (-2147483647..2147483647)
- **'real'** 4 bytes per pixel, floating point
- **'complex'** two matrices of the type **real**
'dvf' two matrices of the type int1
'dir' 1 byte per pixel (0..180)
cyclic' 1 byte per pixel; cyclic arithmetics (0..255).

The default value 0 is set via the operator SetSystem('initNewImage',<true/false>).

Attention

Parameter

Image (output iconic) ...... image ~ HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Created image with new image matrix.

Type (input control) ......................................................... string ~ String / VARIANT

Pixel type

Default Value : 'byte'
List of values : Type ∈ {'byte', 'direction', 'cyclic', 'int1', 'int2', 'uint2', 'int4', 'real', 'complex', 'dvf'}

Width (input control) .................................................. extent.x ~ long / VARIANT

Width of image.

Default Value : 512
Suggested values : Width ∈ {128, 256, 512, 1024}
Typical range of values : 1 ≤ Width ≤ 1(lin)
Minimum Increment : 1
Recommended Increment : 10
Restriction : (Width ≥ 1)

Height (input control) .................................................. extent.y ~ long / VARIANT

Height of image.

Default Value : 512
Suggested values : Height ∈ {128, 256, 512, 1024}
Typical range of values : 1 ≤ Height ≤ 1(lin)
Minimum Increment : 1
Recommended Increment : 10
Restriction : (Height ≥ 1)

If the parameter values are correct, the operator GenImageConst returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

GenImageConst is reentrant and processed without parallelization.

Possible Successors

PaintRegion, ReduceDomain, GetImagePointer1, CopyObj

See also

ReduceDomain, PaintGray, PaintRegion, SetGrayval, GetImagePointer1

Alternatives

GenImage1, GenImage3

Module

Image / region / XLD management

void HImageX.GenImageGrayRamp ([in] double Alpha, [in] double Beta,
in long Height )

void HOperatorSetX.GenImageGrayRamp
([out] HUntypedObjectX ImageGrayRamp, [in] VARIANT Alpha, [in] VARIANT Beta,
in VARIANT Height )

Create a gray value ramp.
The operator \texttt{GenImageGrayRamp} creates a gray value ramp according to the following equation:

\[
\text{ImageGrayRamp}^\prime(r,c) = \text{Alpha}(r - \text{Row}) + \text{Beta}(c - \text{Column}) + \text{Mean}
\]

The size of the image is determined by \texttt{Width} and \texttt{Height}. The gray values are of the type \texttt{byte}. Gray values outside the valid area are clipped.

\begin{itemize}
\item \textbf{ImageGrayRamp} (output iconic) \hspace{1cm} \textit{image} \sim \texttt{HImageX / HUntypedObjectX (byte)} \hspace{1cm} Created image with new image matrix.
\item \textbf{Alpha} (input control) \hspace{1cm} \textit{number} \sim \texttt{double / VARIANT} \hspace{1cm} Gradient in line direction.
\begin{itemize}
\item Default Value : 1.0
\item Suggested values : \texttt{Alpha} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}
\item Minimum Increment : 0.000001
\item Recommended Increment : -0.005
\end{itemize}
\item \textbf{Beta} (input control) \hspace{1cm} \textit{number} \sim \texttt{double / VARIANT} \hspace{1cm} Gradient in column direction.
\begin{itemize}
\item Default Value : 1.0
\item Suggested values : \texttt{Beta} \in \{-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0\}
\item Minimum Increment : 0.000001
\item Recommended Increment : -0.005
\end{itemize}
\item \textbf{Mean} (input control) \hspace{1cm} \textit{number} \sim \texttt{double / VARIANT} \hspace{1cm} Mean gray value.
\begin{itemize}
\item Default Value : 128
\item Suggested values : \texttt{Mean} \in \{0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 255\}
\item Minimum Increment : 1
\item Recommended Increment : 10
\end{itemize}
\item \textbf{Row} (input control) \hspace{1cm} \textit{point.y} \sim \texttt{long / VARIANT} \hspace{1cm} Line index of reference point.
\begin{itemize}
\item Default Value : 256
\item Suggested values : \texttt{Row} \in \{128, 256, 512, 1024\}
\item Minimum Increment : 1
\item Recommended Increment : 10
\end{itemize}
\item \textbf{Column} (input control) \hspace{1cm} \textit{point.x} \sim \texttt{long / VARIANT} \hspace{1cm} Column index of reference point.
\begin{itemize}
\item Default Value : 256
\item Suggested values : \texttt{Column} \in \{128, 256, 512, 1024\}
\item Minimum Increment : 1
\item Recommended Increment : 10
\end{itemize}
\item \textbf{Width} (input control) \hspace{1cm} \textit{extent.x} \sim \texttt{long / VARIANT} \hspace{1cm} Width of image.
\begin{itemize}
\item Default Value : 512
\item Suggested values : \texttt{Width} \in \{128, 256, 512, 1024\}
\item Typical range of values : \(1 \leq \text{Width} \leq 1\) (lin)
\item Minimum Increment : 1
\item Recommended Increment : 10
\item Restriction : \((\text{Width} \geq 1)\)
\end{itemize}
\item \textbf{Height} (input control) \hspace{1cm} \textit{extent.y} \sim \texttt{long / VARIANT} \hspace{1cm} Height of image.
\begin{itemize}
\item Default Value : 512
\item Suggested values : \texttt{Height} \in \{128, 256, 512, 1024\}
\item Typical range of values : \(1 \leq \text{Height} \leq 1\) (lin)
\item Minimum Increment : 1
\item Recommended Increment : 10
\item Restriction : \((\text{Height} \geq 1)\)
\end{itemize}
\end{itemize}
If the parameter values are correct, GenImageGrayRamp returns the value TRUE. Otherwise an exception handling is raised.

GenImageGrayRamp is reentrant and processed without parallelization.

Possible Predecessors

MomentsGrayPlane

Possible Successors

Parallelization Information

Parallelization Information

Possible Predecessors

Parallelization Information

Possible Successors

See also

MomentGrayPlane, ReduceDomain, GetImagePointer1, CopyObj

See also

ReduceDomain, PaintGray

Alternatives

GenImage1

Module

Image / region / XLD management

---

**Result**

If the parameter values are correct, GenImageGrayRamp returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

GenImageGrayRamp is reentrant and processed without parallelization.

---

**Parameter**

- **Image** (input iconic) ...... image  ~ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  - Image, whose gray values are to be cleared.
- **ImageCleared** (output iconic) ...... image  ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  - Image with constant gray value.
- **Grayval** (input control) .................. number  ~ VARIANT (integer, real)
  - Gray value to be used for the output image.

**Default Value**:

0

**Suggested values**:

Grayval \(\in\) \{0, 1, 2, 5, 10, 16, 32, 64, 128, 253, 254, 255\}

**Result**

GenImageProto returns TRUE if all parameters are correct. If necessary, an exception is raised.

---

**Parallelization Information**

GenImageProto is reentrant and processed without parallelization.

---

**Possible Predecessors**

TestObjDef

---

**See also**

GenImagePointer1

---

**Alternatives**

SetGrayval, PaintGray, GenImageConst, CopyObj

---

**Module**

Basic operators
5.3. Creation

Create a curved gray surface with first order polynomial.

The operator `GenImageSurfaceSecondOrder` creates a curved gray value surface according to the following equation:

\[ \text{ImageSurface}(r, c) = \text{Alpha}(r - \text{Row}) + \text{Beta}(c - \text{Col}) + \text{Gamma} \]

The size of the image is determined by `Width` and `Height`. The gray values are of the type `Type`. Gray values outside the valid area are clipped.

```plaintext

```

Parameter

- **ImageSurface** (output iconic) ..........image ∼ HImageX / HUntypedObjectX (byte, uint2, real )
  Created image with new image matrix.

- **Type** (input control) ......................................................string ∼ String / VARIANT
  Pixel type.
  Default Value: ’byte’
  List of values: Type ∈ {’byte’, ’uint2’, ’real’}

- **Alpha** (input control) ..............................................number ∼ double / VARIANT
  First order coefficient in vertical direction.
  Default Value: 1.0
  Suggested values: Alpha ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  Minimum Increment: 0.000001
  Recommended Increment: -0.005

- **Beta** (input control) ..............................................number ∼ double / VARIANT
  First order coefficient in horizontal direction.
  Default Value: 1.0
  Suggested values: Beta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  Minimum Increment: 0.000001
  Recommended Increment: -0.005

- **Gamma** (input control) ............................number ∼ double / VARIANT
  Zero order coefficient
  Default Value: 1.0
  Suggested values: Gamma ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
  Minimum Increment: 0.000001
  Recommended Increment: -0.005

- **Row** (input control) ............................number ∼ double / VARIANT
  Line coordinate of the apex of the surface
  Default Value: 256.0
  Suggested values: Row ∈ {0.0, 128.0, 256.0, 512.0}
  Minimum Increment: 0.000001
  Recommended Increment: -0.005

- **Col** (input control) ............................number ∼ double / VARIANT
  Column coordinate of the apex of the surface
  Default Value: 256.0
  Suggested values: Col ∈ {0.0, 128.0, 256.0, 512.0}
  Minimum Increment: 0.000001
  Recommended Increment: -0.005
CHAPTER 5. IMAGE

- **Width** (input control) .................................................. extent.x ~ long / VARIANT
  
  Width of image.
  
  Default Value : 512
  
  Suggested values : Width ∈ {128, 256, 512, 1024}
  
  Typical range of values : 1 ≤ Width ≤ 1(lin)
  
  Minimum Increment : 1
  
  Recommended Increment : 10
  
  Restriction : (Width ≥ 1)

- **Height** (input control) .................................................. extent.y ~ long / VARIANT
  
  Height of image.
  
  Default Value : 512
  
  Suggested values : Height ∈ {128, 256, 512, 1024}
  
  Typical range of values : 1 ≤ Height ≤ 1(lin)
  
  Minimum Increment : 1
  
  Recommended Increment : 10
  
  Restriction : (Height ≥ 1)

**Result**

If the parameter values are correct `GenImageSurfaceFirstOrder` returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

`GenImageSurfaceFirstOrder` is reentrant, local, and processed without parallelization.

See also `GenImageGrayRamp, GenImageSurfaceSecondOrder`

**Module**

Image / region / XLD management

```c
void HImageX.GenImageSurfaceSecondOrder ([in] String Type,
[in] long Width, [in] long Height )
```

```c
void HOperatorSetX.GenImageSurfaceSecondOrder
([out] HUntypedObjectX ImageSurface, [in] VARIANT Type, [in] VARIANT Alpha,
[in] VARIANT Beta, [in] VARIANT Gamma, [in] VARIANT Delta,
[in] VARIANT Width, [in] VARIANT Height )
```

Create a curved gray surface with second order polynomial.

The operator `GenImageSurfaceSecondOrder` creates a curved gray value surface according to the following equation:

\[
\text{ImageSurface}(r,c) = \text{Alpha}(r-\text{Row})^2 + \text{Beta}(c-\text{Col})^2 + \text{Gamma}(r-\text{Row})(c-\text{Col}) + \text{Delta}(r-\text{Row}) + \text{Epsilon}(c-\text{Col})
\]

The size of the image is determined by `Width` and `Height`. The gray values are of the type `Type`. Gray values outside the valid area are clipped.

**Parameter**

- **ImageSurface** (output iconic) ...................... image ~ HImageX / HUntypedObjectX (byte, uint2, real )
  
  Created image with new image matrix.

- **Type** (input control) .................................................. string ~ String / VARIANT
  
  Pixel type.
  
  Default Value : 'byte'
  
  List of values : Type ∈ {'byte', 'uint2', 'real'}
5.3. Creation

- **Alpha** (input control) .......................... number  ~ double / VARIANT
  Second order coefficient in vertical direction.
  Default Value : 1.0
  Suggested values : Alpha ∈ \{-2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Beta** (input control) .......................... number  ~ double / VARIANT
  Second order coefficient in horizontal direction.
  Default Value : 1.0
  Suggested values : Beta ∈ \{-2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Gamma** (input control) .......................... number  ~ double / VARIANT
  Mixed second order coefficient.
  Default Value : 1.0
  Suggested values : Gamma ∈ \{-2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Delta** (input control) .......................... number  ~ double / VARIANT
  First order coefficient in vertical direction.
  Default Value : 1.0
  Suggested values : Delta ∈ \{-2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Epsilon** (input control) .......................... number  ~ double / VARIANT
  First order coefficient in horizontal direction.
  Default Value : 1.0
  Suggested values : Epsilon ∈ \{-2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Zeta** (input control) .......................... number  ~ double / VARIANT
  Zero order coefficient
  Default Value : 1.0
  Suggested values : Zeta ∈ \{-2.0, -1.0, -0.5, 0.0, 0.5, 1.0, 2.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Row** (input control) .......................... number  ~ double / VARIANT
  Line coordinate of the apex of the surface
  Default Value : 256.0
  Suggested values : Row ∈ \{0.0, 128.0, 256.0, 512.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Col** (input control) .......................... number  ~ double / VARIANT
  Column coordinate of the apex of the surface
  Default Value : 256.0
  Suggested values : Col ∈ \{0.0, 128.0, 256.0, 512.0\}
  Minimum Increment : 0.000001
  Recommended Increment : -0.005

- **Width** (input control) .......................... extent.x  ~ long / VARIANT
  Width of image.
  Default Value : 512
  Suggested values : Width ∈ \{128, 256, 512, 1024\}
  Typical range of values : \(1 \leq \text{Width} \leq 1\) (lin)
  Minimum Increment : 1
  Recommended Increment : 10
  Restriction : (Width ≥ 1)
CHAPTER 5. IMAGE

- **Height** (input control) .................................................. extent.y $\sim$ long / VARIANT
  Height of image.
  Default Value: 512
  Suggested values: $\text{Height} \in \{128, 256, 512, 1024\}$
  Typical range of values: $1 \leq \text{Height} \leq 1$ (lin)
  Minimum Increment: 1
  Recommended Increment: 10
  Restriction: ($\text{Height} \geq 1$)

Result

If the parameter values are correct `GenImageSurfaceSecondOrder` returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

`GenImageSurfaceSecondOrder` is reentrant, local, and processed without parallelization.

See also `GenImageGrayRamp`, `GenImageSurfaceFirstOrder`

Module

Image / region / XLD management

```c
[out] HImageX BinImage HRegionX.RegionToBin ([in] long ForegroundGray,
   [in] long BackgroundGray, [in] long Width, [in] long Height )

void HOperatorSetX.RegionToBin ([in] IObjectX Region,
   [out] HUntypedObjectX BinImage, [in] VARIANT ForegroundGray,
```

Convert a region into a binary byte-image.

`RegionToBin` converts the input region given in `Region` into a byte-image and assigns a gray value of `ForegroundGray` to all pixels in the region. If the input region is larger than the generated image, it is clipped at the image borders. The background is set to `BackgroundGray`.

- **Region** (input iconic) ............................................. region(-array) $\sim$ HRegionX / IObjectX
  Regions to be converted.

- **BinImage** (output iconic) ...................................... image $\sim$ HImageX / HUntypedObjectX (byte)
  Result image of dimension $\text{Width} \times \text{Height}$ containing the converted regions.

- **ForegroundGray** (input control) .............................. integer $\sim$ long / VARIANT
  Gray value in which the regions are displayed.
  Default Value: 255
  Suggested values: $\text{ForegroundGray} \in \{0, 1, 50, 100, 128, 150, 200, 254, 255\}$
  Typical range of values: $0 \leq \text{ForegroundGray} \leq 0$ (lin)
  Recommended Increment: 1

- **BackgroundGray** (input control) .............................. integer $\sim$ long / VARIANT
  Gray value in which the background is displayed.
  Default Value: 0
  Suggested values: $\text{BackgroundGray} \in \{0, 1, 50, 100, 128, 150, 200, 254, 255\}$
  Typical range of values: $0 \leq \text{BackgroundGray} \leq 0$ (lin)
  Recommended Increment: 1

- **Width** (input control) ............................... extent.x $\sim$ long / VARIANT
  Width of the image to be generated.
  Default Value: 512
  Suggested values: $\text{Width} \in \{256, 512, 1024\}$
  Typical range of values: $1 \leq \text{Width} \leq 1$ (lin)
  Minimum Increment: 1
  Recommended Increment: 16
  Restriction: ($\text{Width} \geq 1$)
5.3. CREAT ION

Height (input control) .................................................. extent.y $\sim$ long / VARIANT
Height of the image to be generated.

Default Value : 512
Suggested values : $\text{Height} \in \{256, 512, 1024\}$
Typical range of values : $1 \leq \text{Height} \leq 1\text{lin}$
Minimum Increment : 1
Recommended Increment : 16
Restriction : $(\text{Height} \geq 1)$

$O(2 \ast \text{Height} \ast \text{Width})$. Result

RegionToBin always returns TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

RegionToBin is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, Connection, Regiongrowing, Pouring

Possible Successors

GetGrayval

See also

GenImageProto, PaintGray

Alternatives

RegionToLabel, PaintRegion, SetGrayval

Module

Image / region / XLD management

Convert regions to a label image.

RegionToLabel converts the input regions into a label image according to their index (1..n), i.e., the first region is painted with the gray value 1, the second the gray value 2, etc. Only positive gray values are used. For byte-images the index is entered modulo 256.

Regions larger than the generated image are clipped appropriately. If regions overlap the regions with the higher image are entered (i.e., they are painted in the order in which they are contained in the input regions). If so desired, the regions can be made non-overlapping by calling ExpandRegion.

The background, i.e., the area not covered by any regions, is set to 0. This can be used to test in which image range no region is present.

Parameter

$\triangleright$ Region (input iconic) ...................................... region(-array) $\sim$ HRegionX / IHObjectX
Regions to be converted.

$\triangleright$ ImageLabel (output iconic) .......................... image $\sim$ HImageX / HUntypedObjectX (byte, int2, int4)
Result image of dimension Width $\times$ Height containing the converted regions.

$\triangleright$ Type (input control) ................................................ string $\sim$ String / VARIANT
Pixel type of the result image.

Default Value : 'int2'
List of values : $\text{Type} \in \{\text{byte}, \text{int2}, \text{int4}\}$
**Width** (input control) .......................................................... extent.y ~ long / VARIANT
Width of the image to be generated.
Default Value: 512
Suggested values: Width ∈ {64, 128, 256, 512, 1024}
Typical range of values: 1 ≤ Width ≤ 1(lin)
Minimum Increment: 1
Recommended Increment: 16
Restriction: (Width ≥ 1)

**Height** (input control) .......................................................... extent.x ~ long / VARIANT
Height of the image to be generated.
Default Value: 512
Suggested values: Height ∈ {64, 128, 256, 512, 1024}
Typical range of values: 1 ≤ Height ≤ 1(lin)
Minimum Increment: 1
Recommended Increment: 16
Restriction: (Height ≥ 1)

Complexity

\[ O(2 \cdot \text{Height} \cdot \text{Width}) \].

Result

RegionToLabel always returns TRUE. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`. If necessary, an exception handling is raised.

Parallelization Information

RegionToLabel is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection, ExpandRegion

Possible Successors

GetGrayval, GetImagePointer1

See also

LabelToRegion

Alternatives

RegionToBin, PaintRegion

Module

Image / region / XLD management

```
[out] HImageX ImageMean HRegionX.RegionToMean ([in] HImageX Image )
void HOperatorSetX.RegionToMean ([in] IObjectX Regions, [in] IObjectX Image, [out] HUntypedObjectX ImageMean )
```

Paint regions with their average gray value.

RegionToMean returns an image in which the regions Regions are painted with their average gray value based on the image Image. This operator is mainly intended to visualize segmentation results.

Parameter

- **Regions** (input iconic) ......................................................... region(-array) ~ HRegionX / IObjectX
  Input regions.
- **Image** (input iconic) ......................................................... image ~ HImageX / IObjectX (byte )
  Original gray-value image.
- **ImageMean** (output iconic) .............................................. image ~ HImageX / HUntypedObjectX (byte )
  Result image with painted regions.

Example

```
read_image(Image,'fabrik')
```

HALCON/COM Reference Manual, 2005-2-1
5.4 Domain

Add gray values to regions.
The operator AddChannels adds the gray values from Image to the regions in Regions. All channels of Image are adopted. The definition domain is calculated as the average of the definition domain of the image with the region. Thus the new definition domain can be a subset of the input region. The size of the matrix is not changed.

Parameter

- **Regions** (input iconic) \(\text{region}(-\text{array})\)\(\sim\) HRegionX / IObjectX
  - Input regions (without gray values).
- **Image** (input iconic) \(\text{(multichannel-)image}\)\(\sim\) HImageX / IObjectX
  - Gray image for regions.
- **GrayRegions** (output iconic) \(\text{image}(-\text{array})\)\(\sim\) HImageX / HUntypedObjectX
  - Regions with gray values (also gray images).

**Number of elements:** \(\text{Regions} = \text{GrayRegions}\)

AddChannels is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- Threshold, RegionGrowing, GenCircle, DrawRegion

**Possible Successors**

- Threshold, RegionGrowing, GetDomain

**Alternatives**

- ChangeDomain, ReduceDomain

**Parallelization Information**

AddChannels is reentrant and automatically parallelized (on tuple level, channel level).
### CHAPTER 5. IMAGE

Module

Image / region / XLD management

```c
[out] HImageX ImageNew HImageX.ChangeDomain ([in] HRegionX NewDomain )
```

```c
void HOperatorSetX.ChangeDomain ([in] IHObjectX Image,
```

**Change definition domain of an image.**

The operator `ChangeDomain` uses the indicated region as new definition domain. Unlike the operator `ReduceDomain` it does not form the intersection of the previous definition domain. This can lead to errors particularly when the region is larger than the image matrix. The size of the matrix is not changed.

---

**Attention**

Due to running time the transferred region is not checked for consistency (i.e., whether it fits with the image matrix). Incorrect regions lead to system hang-ups during subsequent operations.

---

**Parameter**

- **Image** (input iconic) ...... image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )
  
  Input image.

- **NewDomain** (input iconic) ..................................................region \(\sim\) HRegionX / IHObjectX
  
  New definition domain.

- **ImageNew** (output iconic) ...... image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )
  
  Image with new definition domain.

---

**Parallelization Information**

`ChangeDomain` is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

GetDomain

---

**See also**

FullDomain, GetDomain, Intersection

---

**Alternatives**

ReduceDomain

---

Image / region / XLD management

```c
[out] HImageX ImageFull HImageX.FullDomain ( )
```

```c
void HOperatorSetX.FullDomain ([in] IHObjectX Image,
[out] HUntypedObjectX ImageFull )
```

**Expand the domain of an image to maximum.**

The operator `FullDomain` enters a rectangle with the edge length of the image as new definition domain. This means that all pixels of the matrix are included in further operations. Thus the same definition domain is obtained as by reading or generating an image. The size of the matrix is not changed.

---

**Parameter**

- **Image** (input iconic) ...... image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )
  
  Input image.

- **ImageFull** (output iconic) ...... image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )
  
  Image with maximum definition domain.

---

HALCON/COM Reference Manual, 2005-2-1
5.4. DOMAIN

Parallelization Information

FullDomain is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

GetDomain

See also

GetDomain, GenRectangle1

Alternatives

ChangeDomain, ReduceDomain

Module

Image / region / XLD management

Get the domain of an image.

The operator GetDomain returns the definition domains of all input images as a region.

Parameter

▷ Image (input iconic) ...... image(-array) \sim HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Input images.

▷ Domain (output iconic) ..................................region(-array) \sim HRegionX / HUntypedObjectX

Definition domains of input images.

Parallelization Information

GetDomain is reentrant and automatically parallelized (on tuple level).

Possible Successors

ChangeDomain, ReduceDomain, FullDomain

See also

GetDomain, ChangeDomain, ReduceDomain, FullDomain

Module

Image / region / XLD management

Reduce the domain of an image to a rectangle.

The operator Rectangle1Domain reduces the definition domain of the given image to the specified rectangle. The old domain of the input image is ignored. The size of the matrix is not changed.

Parameter

▷ Image (input iconic) ...... image(-array) \sim HImageX / IHObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Input image.

▷ ImageReduced (output iconic) ...... image(-array) \sim HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )

Image with reduced definition domain.
Parallelization Information

ReduceDomain is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

GetDomain

See also

FullDomain, GetDomain, Intersection

Alternatives

ChangeDomain, ReduceDomain, AddChannels

Module

Image / region / XLD management

Reduce the domain of an image.

The operator ReduceDomain reduces the definition domain of the given image to the indicated region. The new definition domain is calculated as the intersection of the old definition domain with the region. Thus, the new definition domain can be a subset of the region. The size of the matrix is not changed.

Parameter

Image (input iconic) ...... image(-array) ~ HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Input image.

Region (input iconic) ...........................................region ~ HRegionX / IHObjectX

New definition domain.

ImageReduced (output iconic) ...... image(-array) ~ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)

Image with reduced definition domain.
See also
FullDomain, GetDomain, Intersection
Alternatives
ChangeDomain, Rectangle1Domain, AddChannels
Module

5.5 Features

Compute the area and center of gravity of a region in a gray value image.

AreaCenterGray computes the area and center of gravity of the regions Regions that have gray values which are defined by the image Image. This operator is similar to AreaCenter, but in contrast to that operator, the gray values of the image are taken into account while computing the area and center of gravity.

The area $A$ of a region $R$ in the image with the gray values $g(r,c)$ is defined as

$$A = \sum_{(r,c) \in R} g(r,c).$$

This means that the area is defined by the volume of the gray value function $g(r,c)$. The center of gravity is defined by the first two normalized moments of the gray values $g(r,c)$, i.e., by $(m_{1,0}, m_{0,1})$, where

$$m_{p,q} = \frac{1}{A} \sum_{(r,c) \in R} r^p c^q g(r,c).$$

Parameter

- **Regions** (input iconic) ……………region(-array) → HRegionX / IObjectX
  Region(s) to be examined.
- **Image** (input iconic) …… image → HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  Gray value image.
- **Area** (output control) ……………real(-array) → VARIANT (real)
  Gray value volume of the region.
- **Row** (output control) ……………point.y(-array) → VARIANT (real)
  Row coordinate of the gray value center of gravity.
- **Column** (output control) ………….point.x(-array) → VARIANT (real)
  Column coordinate of the gray value center of gravity.

Result

AreaCenterGray returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception handling is raised.

Parallelization Information

AreaCenterGray is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

See also
AreaCenterXld, EllipticAxisGray

Alternatives

AreaCenter
Calculate a co-occurrence matrix and derive gray value features thereof.

The call of CoocFeatureImage corresponds to the consecutive execution of the operators GenCoocMatrix and CoocFeatureMatrix. If several direction matrices of the co-occurrence matrix are to be evaluated consecutively, it is more efficient to generate the matrix via GenCoocMatrix and then call the operator CoocFeatureMatrix for the resulting matrix. The parameter Direction transfers the direction of the neighborhood in angle or 'mean'. In the case of 'mean' the mean value is calculated in all four directions.

### Parameter

- **Regions** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . region(-array) ~ HRegionX / IObjectX
  Region to be examined.

- **Image** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . image ~ HImageX / IObjectX (byte)
  Corresponding gray values.

- **LdGray** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . integer ~ long / VARIANT
  Number of gray values to be distinguished (2\(^\text{LdGray}\)).
  Default Value: 6
  List of values: LdGray ∈ \{1, 2, 3, 4, 5, 6, 7, 8\}

- **Direction** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . integer ~ VARIANT (integer, string)
  Direction in which the matrix is to be calculated.
  Default Value: 0
  List of values: Direction ∈ \{0, 45, 90, 135, 'mean'\}

- **Energy** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . real(-array) ~ VARIANT (real)
  Gray value energy.

- **Correlation** (output control) . . . . . . . . . . . . . . . . . . . . . . . real(-array) ~ VARIANT (real)
  Correlation of gray values.

- **Homogeneity** (output control) . . . . . . . . . . . . . . . . . . . . . . real(-array) ~ VARIANT (real)
  Local homogeneity of gray values.

- **Contrast** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . real(-array) ~ VARIANT (real)
  Gray value contrast.

### Result

The operator CoocFeatureImage returns the value TRUE if an image with defined gray values (byte) is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator SetSystem(':noObjectResult',<Result>::), the behavior in case of empty region is set via SetSystem(':emptyRegionResult',<Result>::). If necessary an exception handling is raised.

### Parallelization Information

CoocFeatureImage is reentrant and automatically parallelized (on tuple level).

### Possible Predecessors

- GenCoocMatrix

### See also

- Intensity, MinMaxGray, EntropyGray, SelectGray

### Alternatives

- CoocFeatureMatrix

### Module

Image filters

HALCON/COM Reference Manual, 2005-2-1
Calculate gray value features from a co-occurrence matrix.

The procedure calculates from a co-occurrence matrix (\texttt{CoocMatrix}) the energy (\texttt{Energy}), correlation (\texttt{Correlation}), local homogeneity (\texttt{Homogeneity}) and contrast (\texttt{Contrast}).

The operator \texttt{CoocFeatureMatrix} calculates the gray value features from the part of the input matrix generated by \texttt{GenCoocMatrix} corresponding to the direction matrix indicated by the parameters \texttt{LdGray} and \texttt{Direction} according to the following formulae:

\textbf{Energy:}

\[
\text{Energy} = \sum_{i,j=0}^{\text{width}} c_{ij}^2
\]

(Measure for image homogeneity)

\textbf{Correlation:}

\[
\text{Correlation} = \frac{\sum_{i,j=0}^{\text{width}} (i - u_x)(j - u_y)c_{ij}}{s_x s_y}
\]

(Measure for gray value dependencies)

\textbf{Local homogeneity:}

\[
\text{Homogeneity} = \sum_{i,j=0}^{\text{width}} \frac{1}{1 + (i-j)^2} c_{ij}
\]

\textbf{Contrast:}

\[
\text{Contrast} = \sum_{i,j=0}^{\text{width}} (i - j)^2 c_{ij}
\]

(Measure for the size of the intensity differences)

where

- \texttt{width} = Width of \texttt{CoocMatrix}
- \texttt{c}_{ij} = Entry of co-occurrence matrix
- \texttt{u}_x = \sum_{i,j=0}^{\text{width}} i * c_{ij}
- \texttt{u}_y = \sum_{i,j=0}^{\text{width}} j * c_{ij}
- \texttt{s}_x^2 = \sum_{i,j=0}^{\text{width}} (i - \texttt{u}_x)^2 * c_{ij}
- \texttt{s}_y^2 = \sum_{i,j=0}^{\text{width}} (i - \texttt{u}_y)^2 * c_{ij}

Attention: The region of the input image is disregarded.

Parameter

\begin{itemize}
\item \texttt{CoocMatrix} (input iconic) \texttt{image} \sim \texttt{HImageX / IObjectX (real)} Co-occurrence matrix.
\item \texttt{Energy} (output control) \texttt{real} \sim \texttt{double / VARIANT} Homogeneity of the gray values.
\item \texttt{Correlation} (output control) \texttt{real} \sim \texttt{double / VARIANT} Correlation of gray values.
\item \texttt{Homogeneity} (output control) \texttt{real} \sim \texttt{double / VARIANT} Local homogeneity of gray values.
\item \texttt{Contrast} (output control) \texttt{real} \sim \texttt{double / VARIANT} Gray value contrast.
\end{itemize}
The operator \texttt{CoocFeatureMatrix} returns the value \texttt{TRUE} if an image with defined gray values is passed and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(::'noObjectResult',\langle Result\rangle:)}. If necessary an exception handling is raised.

\textbf{Parallelization Information} \texttt{CoocFeatureMatrix} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Predecessors} \texttt{GenCoocMatrix}

See also \texttt{Intensity}, \texttt{MinMaxGray}, \texttt{EntropyGray}, \texttt{SelectGray}

\textbf{Alternatives} \texttt{CoocFeatureImage}

\textbf{Module} \texttt{Image filters}
Determine the entropy and anisotropy of images.

The operator `EntropyGray` creates the histogram of relative frequencies of the gray values in the input image and calculates from these frequencies the entropy and the anisotropy coefficient for each region from `Regions` according to the following formulae:

Entropy:

\[ Entropy = - \sum_{i=0}^{255} rel[i] \cdot \log_2(rel[i]) \]

Anisotropy coefficient:

\[ Anisotropy = \frac{\sum_{i=0}^{k} rel[i] \cdot \log_2(rel[i])}{Entropy} \]

where

- `rel[i]` = Histogram of relative gray value frequencies
- `i` = Gray value of input image (0...255)
- `k` = Smallest possible gray value with \( \sum_{i=0}^{k} rel[i] \geq 0.5 \)

**Parameter**

- `Regions` (input iconic) 
  Region(-array) \( \sim HRegionX / IObjectX \)
  Regions where the features are to be determined.

- `Image` (input iconic) 
  Image (-array) \( \sim HIImageX / IObjectX (byte) \)
  Gray value image.

- `Entropy` (output control) 
  Information content (entropy) \( \sim \) VARIANT (real)
  Measure of the symmetry of gray value distribution.

**Complexity**

If \( F \) is the area of the region the runtime complexity is \( O(F + 255) \).

**Result**

The operator `EntropyGray` returns the value TRUE if an image with defined gray values is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(::'noObjectResult',<Result>)`, the behavior in case of empty region is set via `SetSystem(::'emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`EntropyGray` is reentrant and automatically parallelized (on tuple level).

**See also**

`EntropyImage`, `GrayHisto`, `GrayHistoAbs`, `FuzzyEntropy`, `FuzzyPerimeter`

**Alternatives**

`SelectGray`

**Module**

Image filters
Calculate gray value moments and approximation by a first order surface.

The operator \texttt{FitSurfaceFirstOrder} calculates the gray value moments and the parameters of the approximation of the gray values by a first order surface. The calculation is done by minimizing the distance between the gray values and the surface. A first order surface is described by the following formula:

\[
\text{Image}'(r,c) = \text{Alpha}(r - r_{\text{center}}) + \text{Beta}(c - c_{\text{center}}) + \text{Gamma}
\]

\(r_{\text{center}}\) and \(c_{\text{center}}\) are the coordinates of the center of the input region. By the minimization process the parameters from \(\text{Alpha}\) to \(\text{Gamma}\) is calculated.

The algorithm used for the fitting can be selected via \texttt{Algorithm}:

- \texttt{'regression'} Standard 'least squares' line fitting.
- \texttt{'huber'} Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Huber.
- \texttt{'tukey'} Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Tukey.

The parameter \texttt{ClippingFactor} (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for \texttt{ClippingFactor} the more outliers are detected. The detection of outliers is repeated. The parameter \texttt{Iterations} specifies the number of iterations. In the modus 'regression' this value is ignored.

- \texttt{Regions} (input iconic) \(\text{region(-array)}\) \(\text{HRegionX} / \text{IHObjectX}\) Regions to be checked.
- \texttt{Image} (input iconic) \(\text{image}\) \(\text{HImageX} / \text{IHObjectX}\) Corresponding gray values.
- \texttt{Algorithm} (input control) \(\text{string}\) \(\text{String} / \text{VARIANT}\) Algorithm for the fitting.
  - Default Value: \texttt{'regression'}
  - List of values: \texttt{Algorithm} \(\in \{\text{'regression'}, \text{'huber'}, \text{'tukey'}\}\)
- \texttt{Iterations} (input control) \(\text{integer}\) \(\text{long} / \text{VARIANT}\) Maximum number of iterations (unused for 'regression').
  - Default Value: 5
  - Restriction: \((\text{Iterations} \geq 0)\)
- \texttt{ClippingFactor} (input control) \(\text{real(-array)}\) \(\text{double} / \text{VARIANT}\) Clipping factor for the elimination of outliers.
  - Default Value: 2.0
  - List of values: \texttt{ClippingFactor} \(\in \{1.0, 1.5, 2.0, 2.5, 3.0\}\)
  - Restriction: \((\text{ClippingFactor} > 0)\)
- \texttt{Alpha} (output control) \(\text{real(-array)}\) \(\text{VARIANT} / \text{real}\) Parameter Alpha of the approximating surface.
- \texttt{Beta} (output control) \(\text{real(-array)}\) \(\text{VARIANT} / \text{real}\) Parameter Beta of the approximating surface.
- \texttt{Gamma} (output control) \(\text{real(-array)}\) \(\text{VARIANT} / \text{real}\) Parameter Gamma of the approximating surface.

\textbf{Result}

The operator \texttt{FitSurfaceSecondOrder} returns the value \texttt{TRUE} if an image with the defined gray values \(\text{byte}\) is entered and the parameters are correct. If necessary an exception handling is raised.
FitSurfaceFirstOrder is reentrant and automatically parallelized (on tuple level).

See also MomentsGrayPlane, FitSurfaceSecondOrder

Image filters

Calculate gray value moments and approximation by a second order surface.

The operator FitSurfaceSecondOrder calculates the gray value moments and the parameters of the approximation of the gray values by a second order surface. The calculation is done by minimizing the distance between the gray values and the surface. A second order surface is described by the following formula:

\[
\text{Image}(r,c) = \text{Alpha}(r-r_{center})^2 + \text{Beta}(c-c_{center})^2 + \text{Gamma}(r-r_{center})(c-c_{center}) + \text{Delta}(r-r_{center}) + \text{Epsilon}(c-c_{center}) + \text{Zeta}.
\]

\[ r_{center} \text{ and } c_{center} \text{ are the coordinates of the center of the input region. By the minimization process the parameters from } \text{Alpha} \text{ to } \text{Zeta} \text{ is calculated.} \]

The algorithm used for the fitting can be selected via Algorithm:

- **regression**: Standard 'least squares' fitting.
- **huber**: Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Huber.
- **tukey**: Weighted 'least squares' fitting, where the impact of outliers is decreased based on the approach of Tukey.

The parameter ClippingFactor (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for ClippingFactor the more outliers are detected. The detection of outliers is repeated. The parameter Iterations specifies the number of iterations. In the mode 'regression' this value is ignored.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Regions</strong> (input iconic)</td>
</tr>
<tr>
<td><strong>Image</strong> (input iconic)</td>
</tr>
<tr>
<td><strong>Algorithm</strong> (input control)</td>
</tr>
</tbody>
</table>

**Default Value**: 'regression'

**List of values**: Algorithm \( \in \{ \text{"regression", "tukey", "huber"} \} \)

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Iterations</strong> (input control)</td>
</tr>
</tbody>
</table>

**Default Value**: 5

**Restriction**: \((\text{Iterations} \geq 0)\)

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ClippingFactor</strong> (input control)</td>
</tr>
</tbody>
</table>

**Default Value**: 2.0

**List of values**: ClippingFactor \( \in \{ 1.0, 1.5, 2.0, 2.5, 3.0 \} \)

**Restriction**: \((\text{ClippingFactor} > 0)\)
CHAPTER 5. IMAGE

▸ **Alpha** (output control) .................................................. real(-array) ⇐ VARIANT (real)
Parameter Alpha of the approximating surface.

▸ **Beta** (output control) .................................................. real(-array) ⇐ VARIANT (real)
Parameter Beta of the approximating surface.

▸ **Gamma** (output control) .................................................. real(-array) ⇐ VARIANT (real)
Parameter Gamma of the approximating surface.

▸ **Delta** (output control) .................................................. real(-array) ⇐ VARIANT (real)
Parameter Deltaa of the approximating surface.

▸ **Epsilon** (output control) .................................................. real(-array) ⇐ VARIANT (real)
Parameter Epsilon of the approximating surface.

▸ **Zeta** (output control) .................................................. real(-array) ⇐ VARIANT (real)
Parameter Zeta of the approximating surface.

**Result**
The operator *FitSurfaceSecondOrder* returns the value TRUE if an image with the defined gray values (byte) is entered and the parameters are correct. If necessary an exception handling is raised.

**Parallelization Information**

*FitSurfaceSecondOrder* is reentrant and automatically parallelized (on tuple level).

See also

*MomentsGrayPlane, FitSurfaceFirstOrder*

**Module**

Image filters

|------|---------|---------|-----------------------|------|----------------|------|------|------|------|------|------|


Determine the fuzzy entropy of regions.

*FuzzyEntropy* calculates the fuzzy entropy of a fuzzy set. To do so, the image is regarded as a fuzzy set. The entropy then is a measure of how well the image approximates a white or black image. It is defined as follows:

\[
H(X) = \frac{1}{MN} \sum_{l} T_{e}(l) h(l)
\]

where \( M \times N \) is the size of the image, and \( h(l) \) is the histogram of the image. Furthermore,

\[
T_{e}(l) = -\mu(l) \ln \mu(l) - (1 - \mu(l)) \ln(1 - \mu(l))
\]

Here, \( u(x|m,n) \) is a fuzzy membership function defining the fuzzy set (see *FuzzyPerimeter*). The same restrictions hold as in *FuzzyPerimeter*.

**Parameter**

▸ **Regions** (input iconic) .............................................. region(-array) ⇐ HRegionX / IObjectX
Regions for which the fuzzy entropy is to be calculated.

▸ **Image** (input iconic) ................................................... image ⇐ HImageX / IObjectX (byte)
Input image containing the fuzzy membership values.

▸ **Apar** (input control) .................................................. integer ⇐ long / VARIANT
Start of the fuzzy function.

**Default Value** : 0

**Suggested values** : \( Apar \in \{0, 5, 10, 20, 50, 100\} \)

**Typical range of values** : \( 0 \leq Apar \leq 0(\text{lin}) \)

**Minimum Increment** : 1

**Recommended Increment** : 5
5.5. FEATURES

- \textbf{Cpar} (input control) \hspace{1cm} \text{integer} \sim \text{long / VARIANT}

End of the fuzzy function.

\textbf{Default Value:} 255

\textbf{Suggested values:} $Cpar \in \{50, 100, 150, 200, 220, 255\}$

\textbf{Typical range of values:} $0 \leq Cpar \leq 0$(in)

\textbf{Minimum Increment:} 1

\textbf{Recommended Increment:} 5

\textbf{Restriction:} $(Apar \leq Cpar)$

- \textbf{Entropy} (output control) \hspace{1cm} \text{real(-array)} \sim \text{VARIANT (real)}

Fuzzy entropy of a region.

\texttt{/* To find a Fuzzy Entropy from an Image */}
\texttt{read_image(Image,’affe’)}
\texttt{fuzzy_entropy(Trans,Trans,0,255,Entro).}

\textbf{Result}

The operator \texttt{FuzzyEntropy} returns the value \texttt{TRUE} if the parameters are correct. Otherwise an exception is raised.

\textbf{Parallelization Information}

\texttt{FuzzyEntropy} is reentrant and automatically parallelized (on tuple level).

\textbf{See also}

\texttt{FuzzyPerimeter}

\textbf{References}


\textbf{Module}

\texttt{Image filters}


\texttt{Calculate the fuzzy perimeter of a region.}

The operator \texttt{FuzzyPerimeter} is used to determine the differences of fuzzy membership between an image point and its neighbor points. The right and lower neighbor are taken into account. The fuzzy perimeter is then defined as follows:

$$p(X) = \sum_{m=1}^{M-1} \sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m,n+1})| + \sum_{m=1}^{M-1} \sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m+1,n})|$$

where $M \times N$ is the size of the image, and $\mu(x(m,n))$ is the fuzzy membership function (i.e., the input image). This implementation uses Zadeh’s Standard-S function, which is defined as follows:

$$\mu_X(x) = \begin{cases} 
0, & x \leq a \\
2 \left( \frac{x-a}{c-a} \right)^2, & a < x \leq b \\
1 - 2 \left( \frac{x-c}{c-a} \right)^2, & b < x \leq c \\
1, & c < x 
\end{cases}$$

HALCON 6.1.4
The parameters $a$, $b$ and $c$ obey the following restrictions: $b = \frac{4}{a}$ is the inflection point of the function, $\Delta b = b - a = c - b$ is the bandwidth, and for $x = b \mu(x) = 0.5$ holds. In FuzzyPerimeter, the parameters $Apar$ and $Cpar$ are defined as follows: $b = \frac{Apar + Cpar}{2}$.

- **Regions** (input iconic) region(-array) $\rightarrow$ HRegionX / IHOBJECTX
  Regions for which the fuzzy perimeter is to be calculated.

- **Image** (input iconic) image $\rightarrow$ HIMAGEX / IHOBJECTX (byte)
  Input image containing the fuzzy membership values.

- **Apar** (input control) integer $\rightarrow$ long / VARIANT
  Start of the fuzzy function.
  Default Value: 0
  Suggested values: $Apar \in \{0, 5, 10, 20, 50, 100\}$
  Typical range of values: $0 \leq Apar \leq 0(lin)$
  Minimum Increment: 1
  Recommended Increment: 5

- **Cpar** (input control) integer $\rightarrow$ long / VARIANT
  End of the fuzzy function.
  Default Value: 255
  Suggested values: $Cpar \in \{50, 100, 150, 200, 220, 255\}$
  Typical range of values: $0 \leq Cpar \leq 0(lin)$
  Minimum Increment: 1
  Recommended Increment: 5
  Restriction: $(Apar \leq Cpar)$

- **Perimeter** (output control) real(-array) $\rightarrow$ VARIANT (real)
  Fuzzy perimeter of a region.

----------

/* To find a Fuzzy Entropy from an Image */
read_image(Image,’affe’)
fuzzy_perimeter(Trans,Trans,0,255,Per).

----------

The operator FuzzyPerimeter returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

----------

Parallelization Information

FuzzyPerimeter is reentrant and automatically parallelized (on tuple level).

----------

See also

----------

References


----------

Module

Image filters

```
[out] HImageX Matrix HRegionX.GenCoocMatrix ([in] HImageX Image,

void HOperatorSetX.GenCoocMatrix ([in] IOBJECTX Regions,
[in] VARIANT Direction )
```

Calculate the co-occurrence matrix of a region in an image.

The operator GenCoocMatrix determines from the input regions how often the gray values $i$ and $j$ are located next to each other in a certain direction ($0, 45, 90, 135$ degrees), stores this number in the co-occurrence matrix at
the locations \((i,j)\) and \((j,i)\) (the matrix is symmetrical), and finally scale the matrix with the number of entries. \(\text{LdGray}\) indicates the number of gray values to be distinguished (namely \(2^{\text{LdGray}}\)).

Example (\(\text{LdGray} = 2\), i.e. 4 gray values are distinguished):

<table>
<thead>
<tr>
<th>Input image with gray values:</th>
<th>Co-occurrence matrix (not scaled)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 3</td>
<td>2 0 0 1 0 1 1 0</td>
</tr>
<tr>
<td>1 1 2</td>
<td>0 2 2 0 1 0 1 1</td>
</tr>
<tr>
<td>1 2 3</td>
<td>0 2 0 1 1 0 0</td>
</tr>
<tr>
<td></td>
<td>1 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>0 2 0 0 0 1 0 0</td>
</tr>
<tr>
<td></td>
<td>2 2 1 0 1 2 0 1</td>
</tr>
<tr>
<td></td>
<td>0 1 0 2 0 0 2 0</td>
</tr>
<tr>
<td></td>
<td>0 0 2 0 0 1 0 0</td>
</tr>
</tbody>
</table>

\[\begin{pmatrix}
0 & 0 & 3 & 2 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\
1 & 1 & 2 & 0 & 2 & 2 & 0 & 1 & 0 & 1 & 1 \\
1 & 2 & 3 & 0 & 2 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 &  &  &  \\
0 & 2 & 0 & 0 & 0 & 1 & 0 & 0 &  &  &  \\
2 & 2 & 1 & 0 & 1 & 2 & 0 & 1 &  &  &  \\
0 & 1 & 0 & 2 & 0 & 0 & 2 & 0 &  &  &  \\
0 & 0 & 2 & 0 & 0 & 1 & 0 & 0 &  &  & \\
\end{pmatrix}\]

\[\text{Parameter}\]

- **Regions** (input iconic) \(\Rightarrow H\text{RegionX} / H\text{ObjectX}\)
  Region to be checked.

- **Image** (input iconic) \(\Rightarrow H\text{ImageX} / H\text{ObjectX}\) (byte)
  Image providing the gray values.

- **Matrix** (output iconic) \(\Rightarrow H\text{ImageX} / H\text{UntypedObjectX}\) (real)
  Co-occurrence matrix (matrices).

- **LdGray** (input control) \(\Rightarrow\) integer \(\Rightarrow long /\text{VARIANT}\)
  Number of gray values to be distinguished (\(2^{\text{LdGray}}\)).

  **Default Value**: 6
  **List of values**: \(\text{LdGray} \in \{1, 2, 3, 4, 5, 6, 7, 8\}\)
  **Typical range of values**: \(1 \leq \text{LdGray} \leq 8\text{(lin)}\)
  **Minimum Increment**: 1
  **Recommended Increment**: 1

- **Direction** (input control) \(\Rightarrow\) integer \(\Rightarrow long /\text{VARIANT}\)
  Direction of neighbor relation.

  **Default Value**: 0
  **List of values**: \(\text{Direction} \in \{0, 45, 90, 135\}\)

\[\text{Result}\]

The operator \texttt{GenCoocMatrix} returns the value TRUE if an image with defined gray values is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(‘noObjectResult’,<Result>:)}; the behavior in case of empty region is set via \texttt{SetSystem(‘emptyRegionResult’,<Result>:)}. If necessary an exception handling is raised.

\[\text{Parallelization Information}\]

\texttt{GenCoocMatrix} is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- \texttt{DrawRegion}, \texttt{GenCircle}, \texttt{GenEllipse}, \texttt{GenRectangle1}, \texttt{GenRectangle2}, \texttt{Threshold}, \texttt{ErosionCircle}, \texttt{GaussImage}, \texttt{SmoothImage}, \texttt{SubImage}

**See also**

- \texttt{CoocFeatureMatrix}

**Alternatives**

- \texttt{CoocFeatureImage}

**Module**

Image filters
**CHAPTER 5. IMAGE**

```c
[out] VARIANT AbsoluteHisto HRegionX.GrayHisto ([in] HImageX Image,
[out] VARIANT RelativeHisto )
```

The operator **GrayHisto** calculates for the image (**Image**) within **Regions** the absolute (**AbsoluteHisto**) and relative (**RelativeHisto**) histogram of the gray values.

Both histograms are tuples of 256 values, which — beginning at 0 — contain the frequencies of the individual gray values of the image.

**AbsoluteHisto** indicates the absolute frequencies of the gray values in integers, and **RelativeHisto** indicates the relative, i.e. the absolute frequencies divided by the area of the image as floating point numbers.

**real**, **int2**, **uint2**, and **int4**-images are transformed into **byte**-images (first the largest and smallest gray value in the image are determined, and then the original gray values are mapped linearly into the area 0..255) and then processed as mentioned above. The histogram can also be returned directly as a graphic via the operators **SetPaint** (**::WindowHandle,'histogram':**) and **DispImage**.

---

**Attention**

Real, int2, uint2, and int4 images are reduced to 256 gray values.

---

**Parameter**

- **Regions** (input iconic) ................................. region(-array)  HRegionX / IObjectX
  Region in which the histogram is to be calculated.
- **Image** (input iconic) .............................. image  HImageX / IObjectX ( byte, cyclic, direction, int1, int2, uint2, int4, real )
  Image the gray value distribution of which is to be calculated.
- **AbsoluteHisto** (output control) ........................... histogram  VARIANT ( integer )
  Absolute frequencies of the gray values.
- **RelativeHisto** (output control) ............................ histogram  VARIANT ( real )
  Frequencies, normalized to the area of the region.

---

**Complexity**

If **F** is the area of the region the runtime complexity is \( O(F + 255) \).

---

**Result**

The operator **GrayHisto** returns the value TRUE if the image has defined gray values and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (**::'noObjectResult','<Result>:**) , the behavior in case of empty region is set via **SetSystem** (**::'emptyRegionResult','<Result>:**) . If necessary an exception handling is raised.

---

**Parallelization Information**

**GrayHisto** is **reentrant** and processed **without** parallelization.

---

**Possible Successors**

**HistToThresh**, **GenRegionHisto**

---

**See also**

**SetPaint**, **DispImage**, **HistTo2Dim**, **ScaleImageMax**, **EntropyGray**

---

**Alternatives**

**MinMaxGray**, **Intensity**, **GrayHistoAbs**

---

**Module**

Image filters
5.5. Features

Calculate the gray value distribution.

The operator \texttt{GrayHistoAbs} calculates for the image (\texttt{Image}) within \texttt{Regions} the absolute (\texttt{AbsoluteHisto}) histogram of the gray values.

The parameter \texttt{Quantization} defines, how many frequencies of neighbored gray values are added for one frequency value. The resulting histogram \texttt{AbsoluteHisto} is a tuple, whose indices are mapped on the gray values of the input image \texttt{Image} and whose elements contain the frequencies of the gray values. The indices $i$ of the frequency value are calculated from the gray values $g$ and the quantisation $q$ as follows:

$$i = \left\lceil \frac{g + 0.5}{q} \right\rceil$$

for unsigned image types,

$$i = \left\lceil \frac{g - (\text{MIN} - 0.5)}{q} \right\rceil$$

for signed image types,

whereas \texttt{MIN} denotes the minimal gray value, e.g., -128 for an \texttt{int1} image type. Therefore, the size of the tuple results from the ratio of the full domain of gray values and the quantisation, e.g. for images of \texttt{int2} in $[\frac{-32768}{32767}] = 21846$. The origin gray value of the signed image types \texttt{int1} resp. \texttt{int2} is mapped on the index 128 resp. 32768, negative resp. positive gray values have smaller resp. greater indices.

The histogram can also be returned directly as a graphic via the operators \texttt{SetPaint (::WindowHandle, ‘histogram’)} and \texttt{DispImage}.

### Parameter

- **Regions** (input iconic) .............. \texttt{region(-array) \sim HRegionX / IHOBJECTX} Region in which the histogram is to be calculated.
- **Image** (input iconic) .............. \texttt{image \sim HIMAGEX / IHOBJECTX} (byte, cyclic, direction, int1, int2, uint2) Image the gray value distribution of which is to be calculated.
- **Quantization** (input control) ........ .......... \texttt{histogram \sim VARIANT(integer, real)} Quantization of the gray values.

**Default Value:** 1.0

**List of values:** \texttt{Quantization \in \{1.0, 2.0, 3.0, 5.0, 10.0\}}

**Restriction:** ($\text{Quantization} \geq 1.0$)

- **AbsoluteHisto** (output control) ............. \texttt{histogram \sim VARIANT(integer)} Absolute frequencies of the gray values.

### Result

The operator \texttt{GrayHistoAbs} returns the value \texttt{TRUE} if the image has defined gray values and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem (::’noObjectResult’,<Result>:)}, the behavior in case of empty region is set via \texttt{SetSystem (::’emptyRegionResult’,<Result>:)}. If necessary an exception handling is raised.

### Parallelization Information

\texttt{GrayHistoAbs} is \texttt{reentrant} and processed \texttt{without} parallelization.

### Possible Successors

\texttt{HistToThresh, GenRegionHisto}

### See also

\texttt{SetPaint, DispImage, HistTo2Dim, ScaleImageMax, EntropyGray}

### Alternatives

\texttt{MinMaxGray, Intensity, GrayHisto}

### Module

Image filters

HALCON 6.1.4
Calculate horizontal and vertical gray-value projections.

GrayProjections calculates the horizontal and vertical gray-value projections, i.e., the mean values in the horizontal and vertical direction of the gray values of the input image Image within the input region Region.

If Mode = 'simple' is selected the projection is performed in the direction of the coordinate axes of the image, i.e.:

\[
\text{HorProjection}(r) = \sum_{(r+r', c+c') \in \text{Region}} \text{Image}(r+r', c+c')
\]

\[
\text{VertProjection}(c) = \sum_{(r+r', c+c') \in \text{Region}} \text{Image}(r+r', c+c')
\]

Here, \((r', c')\) denotes the upper left corner of the smallest enclosing axis-parallel rectangle of the input region (see SmallestRectangle1). Hence, the horizontal projection returns a one-dimensional function that reflects the vertical gray value changes. Likewise, the vertical projection returns a function that reflects the horizontal gray value changes.

If Mode = 'rectangle' is selected the projection is performed in the direction of the major axes of the smallest enclosing rectangle of arbitrary orientation of the input region (see SmallestRectangle2). Here, the horizontal projection direction corresponds to the smaller axis, while the vertical direction corresponds to the larger axis. In this mode, all gray values within the smallest enclosing rectangle of arbitrary orientation of the input region are used to compute the projections.
The operator **Histo2Dim** calculates the 2-dimensional histogram of two images within **Regions**. The gray values of channel 1 (**ImageCol**) are interpreted as row index, those of channel 2 (**ImageRow**) as column index. The gray value at one point \( P(g_1, g_2) \) in the output image **Histo2Dim** indicates the frequency of the gray value combination \((g_1, g_2)\) with \(g_1\) indicating the line index and \(g_2\) the column index.

### Parameter

- **Regions** (input iconic) ................. region(-array) \( \sim \) **HRegionX** / **IObjectX** Region in which the histogram is to be calculated.
- **ImageCol** (input iconic) ................. image \( \sim \) **HImageX** / **IObjectX** (byte, direction, cyclic, int1) Channel 1.
- **ImageRow** (input iconic) ................. image \( \sim \) **HImageX** / **IObjectX** (byte, direction, cyclic, int1) Channel 2.
- **Histo2Dim** (output iconic) .............. image \( \sim \) **HImageX** / **HUntypedObjectX** (int4) Histogram to be calculated.

### Example

```plaintext
read_image(Image,’affe’)
texture_laws(Image,Texture,’el’,1,5)
draw_region(Region,WindowHandle)
histo_2dim(Region,Texture,Image,Histo2Dim)
disp_image(Histo2Dim,WindowHandle).
```

### Complexity

If \( F \) is the plane of the region, the runtime complexity is \( O(F + 256^2) \).

### Result

The operator **Histo2Dim** returns the value TRUE if both images have defined gray values. The behavior in case of empty input (no input images available) is set via the operator **SetSystem** (::’noObjectResult’,<Result>:), the behavior in case of empty region is set via **SetSystem** (::’emptyRegionResult’,<Result>:). If necessary an exception handling is raised.

### Parallelization Information

**Histo2Dim** is **reentrant** and processed **without** parallelization.

### Possible Predecessors

**Decompose3**, **Decompose2**, **DrawRegion**

### Possible Successors

**Threshold**, **Class2DimSup**, **Pouring**, **LocalMax**, **GraySkeleton**

### See also

**GetGrayval**

### Alternatives

**GrayHisto**, **GrayHistoAbs**

### Module

**Image filters**

```plaintext
[out] VARIANT Mean **HRegionX.Intensity** ([in] **HImageX** Image,
  [out] VARIANT Deviation )

void **HOperatorSetX.Intensity** ([in] **IObjectX** Regions,
  [in] **IObjectX** Image, [out] VARIANT Mean, [out] VARIANT Deviation )
```

Calculate the mean and deviation of gray values.

The operator **Intensity** calculates the mean and the deviation of the gray values in the input image within **Regions**. If \( R \) is a region, \( p \) a pixel from \( R \) with the gray value \( g(p) \) and \( F \) the plane \( (F = |R|) \), the features are defined by:

\[
\text{Mean} := \frac{\sum_{p \in R} g(p)}{F}
\]
\[ \text{Deviation} := \sqrt{\frac{\sum_{p \in R} (g(p) - \text{Mean})^2}{F}} \]

**Attention**

The calculation of \textit{Deviation} does not follow the usual definition if the region of the image contains only one pixel. In this case 0.0 is returned.

---

### Parameter

- **Regions** (input iconic) \(\text{region(-array)} \sim \text{HRegionX/IOObjectX}\)
  
  Regions the features of which are to be calculated.

- **Image** (input iconic) \(\text{image} \sim \text{HImageX/IOObjectX}\)
  
  Gray value image.

- **Mean** (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\)
  
  Mean gray value of a region.

- **Deviation** (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\)
  
  Deviation of gray values within a region.

---

### Complexity

If \(F\) is the area of the region, the runtime complexity is \(O(F)\).

---

### Result

The operator \textit{Intensity} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \textit{SetSystem}::{‘noObjectResult’,<Result>:}, the behavior in case of empty region is set via \textit{SetSystem}::{‘emptyRegionResult’,<Result>:}. If necessary an exception handling is raised.

---

### Parallelization Information

\textit{Intensity} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

---

### Possible Successors

- **Threshold**

---

### See also

- MeanImage, MeanImage, GrayHisto, GrayHistoAbs

---

### Alternatives

- SelectGray, MinMaxGray

---

### Module

**Image filters**

---

```plaintext

```

\textit{Determine the minimum and maximum gray values within regions.}

The operator \textit{MinMaxGray} creates the histogram of the absolute frequencies of the gray values within \textit{Regions} in the input image \textit{Image} (see \textit{GrayHisto}) and calculates the number of pixels \textit{Percent} corresponding to the area of the input image. Then it goes inwards on both sides of the histogram by this number of pixels and determines the smallest and the largest gray value:

\[ \text{e.g.} \quad \text{Area} = 60, \text{percent} = 5, \text{i.e.} 3 \text{ pixels} \]

\[ \text{histogram} = [2,8,0,7,13,0,0,\ldots,0,10,10,5,3,1,1] \]

\[ \Rightarrow \text{Maximum} = 255, \text{Minimum} = 0, \text{Range} = 255 \]

\textit{MinMaxGray} returns: \text{Maximum} = 253, \text{Minimum} = 1, \text{Range} = 252

If \textit{Percent} is set at 50, \text{Min} = \text{Max} = \text{Median}. If \textit{Percent} is 0 no histogram is calculated in order to enhance the runtime.

HALCON/COM Reference Manual, 2005-2-1
In case of int2, uint2, int4- and real-images **Percent** has to be 0.

**Parameter**

- **Regions** (input iconic) ... region(-array) \( \sim H\text{Region}X / IH\text{Object}X \)
  Regions, the features of which are to be calculated.

- **Image** (input iconic) ... image \( \sim H\text{Image}X / IH\text{Object}X \) (byte, direction, cyclic, int1, int2, uint2, int4, real)
  Gray value image.

- **Percent** (input control) ... number \( \sim \) VARIANT (integer, real)
  Percentage below (above) the absolute maximum (minimum).

  **Default Value**: 0
  **Suggested values**: Percent \( \in \{0, 1, 2, 5, 7, 10, 15, 20, 30, 40, 50\} \)
  **Restriction**: \((0 \leq \text{Percent}) \land (\text{Percent} \leq 50)\)

- **Min** (output control) ... real(-array) \( \sim \) VARIANT (real)
  “Minimum” gray value.

  **Restriction**: \((\text{Max} \geq \text{Min})\)

- **Max** (output control) ... real(-array) \( \sim \) VARIANT (real)
  “Maximum” gray value.

- **Range** (output control) ... real(-array) \( \sim \) VARIANT (real)
  Difference between Max and Min.

  **Restriction**: \((\text{Range} \geq 0)\)

**Example**

/* Threshold segmentation with training region: */
read_image(Image,’fabrik’)
draw_region(Region,WindowHandle)
min_max_gray(Region,Image,5,Min,Max,_)"
CHAPTER 5. IMAGE

Calculate gray value moments and approximation by a plane.

The operator `MomentsGrayPlane` calculates the gray value moments and the parameters of the approximation of the gray values by a plane. The calculation is carried out according to the following formula:

\[
\text{MRow} = \frac{1}{F^2} \sum_{(r,c) \in \text{Regions}} (r - \bar{r})(\text{Image}(r,c) - \text{Mean}) \\
\text{MCol} = \frac{1}{F^2} \sum_{(r,c) \in \text{Regions}} (c - \bar{c})(\text{Image}(r,c) - \text{Mean})
\]

\[
\text{Alpha} = \frac{\text{MRow} \cdot m_{02} - m_{11} \cdot \text{MCol}}{F} \\
\text{Beta} = \frac{m_{20} \cdot \text{MCol} \cdot F - \text{MRow} \cdot m_{11}}{m_{20} \cdot m_{02} - m_{11}^2}
\]

where \( F \) is the plane, \( \bar{r}, \bar{c} \) the center, and \( m_{11}, m_{20}, \text{and} m_{02} \) the scaled moments of \( \text{Regions} \).

The parameters `Alpha`, `Beta` and `Mean` describe a plane above the region:

\[
\text{Image}'(r,c) = \text{Alpha}(r - \bar{r}) + \text{Beta}(c - \bar{c}) + \text{Mean}
\]

Thus `Alpha` indicates the gradient in the direction of the line axis ("down"), `Beta` the gradient in the direction of the column axis (to the “right”).

The operator `MomentsGrayPlane` returns the value TRUE if an image with the defined gray values (byte) is entered and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator `SetSystem(::'noObjectResult',<Result>:)`, the behavior in case of empty region is set via `SetSystem(::'emptyRegionResult',<Result>:)`. If necessary an exception handling is raised.

Parallelization Information

`MomentsGrayPlane` is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

`DrawRegion, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, Threshold, Regiongrowing`

See also

`Intensity, MomentsRegion2Nd`
Calculate the deviation of the gray values from the approximating image plane.

The operator `PlaneDeviation` calculates the deviation of the gray values in `Image` from the approximation of the gray values through a plane. Contrary to the standard deviation in case of `Intensity` slanted gray value planes also receive the value zero. The gray value plane is calculated according to `GenImageGrayRamp`.

If $F$ is the plane, $\alpha, \beta, \mu$ the parameters of the image plane and $(r', c')$ the center, `Deviation` is defined by:

$$
\text{Deviation} = \sqrt{\frac{\sum_{(r,c) \in \text{Regions}} ((\alpha(r - r') + \beta(c - c') + \mu) - \text{Image}(r,c))^2}{F}}.
$$

**Attention**

It should be noted that the calculation of `Deviation` does not follow the usual definition. It is defined to return the value 0.0 for an image with only one pixel.

**Parameter**

- **Regions** (input iconic) ............................... `region(-array) \sim \text{HRegionX / IHObjectX Regions, of which the plane deviation is to be calculated.}`
- **Image** (input iconic) ............................... `image \sim \text{HImageX / IHObjectX ( byte, cyclic ) Gray value image.}`
- **Deviation** (output control) ............................. `real(-array) \sim \text{VARIANT ( real ) Deviation of the gray values within a region.}`

**Complexity**

If $F$ is the area of the region the runtime complexity amounts to $O(F)$.

**Result**

The operator `PlaneDeviation` returns the value `TRUE` if `Image` is of the type `byte`. The behavior in case of empty input (no input images available) is set via the operator `SetSystem (::'noObjectResult',<Result>:)`, the behavior in case of empty region is set via `SetSystem (::'emptyRegionResult',<Result>:)`. If necessary an exception handling is raised.

**Parallelization Information**

`PlaneDeviation` is **reentrant** and automatically **parallelized** (on tuple level).

**See also**

`MomentsGrayPlane`

**Alternatives**

`Intensity, GenImageGrayRamp, SubImage`

**Module**

`Image filters`
Select regions based on gray value features.

The operator `SelectGray` has a number of regions (Regions) as input. For each of these regions the features (Features) are calculated. If each (Operation = 'and') or at least one (Operation = 'or') of the calculated features is within the limits determined by the parameter, the region is transferred (duplicated) into the output. The parameter `Image` contains an image which returns the gray values for calculating the features.

Condition:

\[ \text{Min}, \leq \text{Features}_{\text{Regions,Image}} \leq \text{Max}, \]

Possible values for `Features`:

- `'area'` Gray value volume of region (see `AreaCenterGray`)
- `'row'` Row index of the center of gravity (see `AreaCenterGray`)
- `'column'` Column index of the center of gravity (see `AreaCenterGray`)
- `'ra'` Major axis of equivalent ellipse (see `EllipticAxisGray`)
- `'rb'` Minor axis of equivalent ellipse (see `EllipticAxisGray`)
- `'phi'` Orientation of equivalent ellipse (see `EllipticAxisGray`)
- `'min'` Minimum gray value (see `MinMaxGray`)
- `'max'` Maximum gray value (see `MinMaxGray`)
- `'mean'` Mean gray value (see `Intensity`)
- `'deviation'` Deviation of gray values (see `Intensity`)
- `'plane deviation'` Deviation from the approximating plane (see `PlaneDeviation`)
- `'anisotropy'` Anisotropy (see `EntropyGray`)
- `'entropy'` Entropy (see `EntropyGray`)
- `'fuzzy entropy'` Fuzzy entropy of region (see `FuzzyEntropy`, with a fuzzy function from Apar=0 to Cpar=255)
- `'fuzzy perimeter'` Fuzzy perimeter of region (see `FuzzyPerimeter`, with a fuzzy function from Apar=0 to Cpar=255)
- `'moments row'` Mixed moments along a row (see `MomentsGrayPlane`)
- `'moments column'` Mixed moments along a column (see `MomentsGrayPlane`)
- `'alpha'` Approximating plane, parameter Alpha (see `MomentsGrayPlane`)
- `'beta'` Approximating plane, parameter Beta (see `MomentsGrayPlane`)

Attention

If only one feature is used the value of `Operation` is meaningless. Several features are processed in the order in which they are entered.

**Parameter**

- `Regions` (input iconic) ............................. region ↝ `HRegionX / IObjectName` Regions to be examined.
- `Image` (input iconic) .......................... image ↝ `HImageX / IObjectName` (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Gray value image.
- `SelectedRegions` (output iconic) ...................... region ↝ `HRegionX / HUntypedObjectX` Regions having features within the limits.
5.5. FEATURES

Features (input control) .............................. string(-array)  \sim   VARIANT ( string )
Names of the features.
Default Value : 'mean'
List of values : Features ∈ {'area', 'row', 'column', 'ra', 'rb', 'phi', 'min', 'max', 'mean', 'deviation', 'plane_deviation', 'anisotropy', 'entropy', 'fuzzy_entropy', 'fuzzy_perimeter', 'moments_row', 'moments_column', 'alpha', 'beta'}

Operation (input control) .............................. string  \sim   String / VARIANT
Logical connection of features.
Default Value : 'and'
List of values : Operation ∈ {'and', 'or'}

Min (input control) .............................. number(-array)  \sim   VARIANT ( integer, real )
Lower limit(s) of features.
Default Value : 128.0
Suggested values : Min ∈ {0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0}

Max (input control) .............................. number(-array)  \sim   VARIANT ( integer, real )
Upper limit(s) of features.
Default Value : 255.0
Suggested values : Max ∈ {0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0}

Complexity
If \( F \) is the area of the region and \( N \) the number of features the runtime complexity is \( O(F \times N) \).

Result
The operator \texttt{SelectGray} returns the value TRUE if the input image has the defined gray values and the parameters are correct. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(::'noObjectResult',<Result>:)}, the behavior in case of empty region is set via \texttt{SetSystem(::'emptyRegionResult',<Result>:)}. If necessary an exception handling is raised.

Parallelization Information
\texttt{SelectGray} is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Connection, MeanImage, EntropyImage, SobelAmp, MedianSeparate

Possible Successors
SelectShape, SelectGray, ShapeTrans, ReduceDomain, CountObj

See also
DeviationImage, EntropyGray, Intensity, MeanImage, MinMaxGray, SelectObj

Module

Image filters

\[ \texttt{HRegionX.ShapeHistoAll} ([\texttt{in}] \texttt{HImageX Image}, [\texttt{in}] \texttt{String Feature}, [\texttt{out}] \texttt{VARIANT AbsoluteHisto}) \]

\[ \texttt{HOperatorSetX.ShapeHistoAll} ([\texttt{in}] \texttt{IObjectX Region}, [\texttt{in}] \texttt{IObjectX Image}, [\texttt{in}] \texttt{VARIANT Feature}, [\texttt{out}] \texttt{VARIANT AbsoluteHisto}, [\texttt{out}] \texttt{VARIANT RelativeHisto}) \]

Determine a histogram of features along all threshold values.
The operator \texttt{ShapeHistoAll} carries out 255 threshold operations within \texttt{Region} with the gray values of \texttt{Image}. The entry \( i \) in the histogram corresponds to the number of connected components/holes of this image segmented with the threshold \( i \) (Feature = 'connected_components', 'holes') or the mean value of the feature values of the regions segmented in this way (Feature = 'convexity', 'compactness', 'anisometry'), respectively.
The histogram can also be displayed directly as a graphic via the operators \texttt{SetPaint (::WindowHandle,'componentHistogram')}: and \texttt{DispImage}.

Attention
The operator \texttt{ShapeHistoAll} expects a region and exactly one gray value image as input. Because of the power of this operator the runtime of \texttt{ShapeHistoAll} is relatively large!
CHAPTER 5. IMAGE

Parameter

- **Region** (input iconic) .......................... region \( HRegionX / IObjectX \)
  Region in which the features are to be examined.

- **Image** (input iconic) .......................... image \( HImageX / IObjectX \) (byte)
  Gray value image.

- **Feature** (input control) .......................... string \( String / VARIANT \)
  Feature to be examined.

  **Default Value:** 'connected components'

  **List of values:** Feature \( \in \{ \text{connected components}, \text{convexity}, \text{compactness}, \text{anisometry}, \text{holes} \} \)

- **AbsoluteHisto** (output control) .......................... histogram \( VARIANT \) (integer, real)
  Absolute distribution of the feature.

- **RelativeHisto** (output control) .......................... histogram \( VARIANT \) (real)
  Relative distribution of the feature.

Example

```
/* Simulation von shape_hist_all mit Merkmal ‘connected_components’: */
my_shape_hist_all(Region,Image,AbsHisto,RelHisto):
  reduce_domain(Region,Image,RegionGray)
  for(0,255,i)
    threshold(RegionGray,Seg,i,255)
    connect_and_holes(Seg,AbsHisto[i],_)
    clear_obj([Seg,H])
  end_for()
  eval(0,Sum)
  for(0,255,i)
    eval(Sum+AbsHisto[i],Sum)
  end_for()
  for(0,255,i)
    eval(AbsHisto[i]/Sum,RelHisto[i])
  end_for()
```

Complexity

If \( F \) is the area of the input region and \( N \) the mean number of connected components the runtime complexity is \( O(255(F + \sqrt{F} \sqrt{N})) \).

Result

The operator **ShapeHistAll** returns the value TRUE if an image with the defined gray values is entered. The behavior in case of empty input (no input images) is set via the operator **SetSystem** (::'noObjectResult',<Result>);, the behavior in case of empty region is set via **SetSystem** (::'emptyRegionResult',<Result>);. If necessary an exception handling is raised.

Parallelization Information

**ShapeHistAll** is reentrant and processed without parallelization.

Possible Successors

**HistoToThresh, Threshold, GenRegionHisto**

See also

**Connection, Convexity, Compactness, ConnectAndHoles, EntropyGray, GrayHisto, SetPaint, CountObj**

Alternatives

**ShapeHistPoint**

Module

Image filters
Determine a histogram of features along all threshold values.

Like ShapeHistoAll the operator ShapeHistoPoint carries out 255 threshold value operations within Region with the gray values of Image. Contrary to ShapeHistoAll only the segmented region containing the pixel (Row, Column) is taken into account here. The entry \( i \) in the histogram then corresponds to the number of holes of this region segmented with the threshold \( i \) (Feature = 'holes') or the feature value of the region (Feature = 'convexity', 'compactness', 'anisometry'), respectively.

The histogram can also be displayed directly as a graphic via the operators SetPaint (:WindowHandle,'componentHistogram':) and DispImage.

### Parameter

- **Region** (input iconic) Region in which the features are to be examined.
- **Image** (input iconic) Gray value image.
- **Feature** (input control) Feature to be examined.
  - Default Value: 'convexity'
  - List of values: Feature \( \in \{ \text{convexity, compactness, anisometry, holes} \} \)
- **Row** (input control) Row of the pixel which the region must contain.
  - Default Value: 256
  - Suggested values: Row \( \in \{ 10, 50, 100, 200, 300, 400 \} \)
- **Column** (input control) Column of the pixel which the region must contain.
  - Default Value: 256
  - Suggested values: Column \( \in \{ 10, 50, 100, 200, 300, 400 \} \)
- **AbsoluteHisto** (output control) Absolute distribution of the feature.
- **RelativeHisto** (output control) Relative distribution of the feature.

### Result

The operator ShapeHistoPoint returns the value TRUE if an image with defined gray values is entered. The behavior in case of empty input (no input images available) is set via the operator SetSystem (::'noObjectResult',<Result>:), the behavior in case of empty region is set via SetSystem (::'emptyRegionResult',<Result>:). If necessary an exception handling is raised.

### Parallelization Information

ShapeHistoPoint is reentrant and processed without parallelization.

### Possible Predecessors

GetMbutton, AreaCenter

### Possible Successors

HistoToThresh, Threshold, GenRegionHisto

### See also

Connection, ConnectAndHoles, Convexity, Compactness, SetPaint

### Alternatives

ShapeHistoAll
5.6 Format

```c
[out] HImageX ImagePart HImageX.ChangeFormat ([in] long Width,
[in] long Height )
```

```c
void HOperatorSetX.ChangeFormat ([in] IObjectX Image,
```

Change image size.

The operator ChangeFormat increases or decreases the size of the input images to the indicated height or width, respectively. If the image is reduced, parts are cut off at the “right” or “lower” edge of the image, respectively. If the image is enlarged, the additional areas are set to 0. The definition domain of the new image is equal to the domain of the input image, clipped to the size of the new image. No zooming is carried out.

---

**Parameter**

- **Image** (input iconic) ...... image(-array) \( \sim \) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  
  Input image.

- **ImagePart** (output iconic) ...... image(-array) \( \sim \) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dVf)
  
  Image with new format.

- **Width** (input control) .......................................................... extent.x \( \sim \) long / VARIANT
  
  Width of new image.

  **Default Value**: 512

  **Suggested values**: Width \( \in \) \{32, 64, 128, 256, 512, 768, 1024\}

- **Height** (input control) .......................................................... extent.y \( \sim \) long / VARIANT
  
  Height of new image.

  **Default Value**: 512

  **Suggested values**: Height \( \in \) \{32, 64, 128, 256, 512, 525, 1024\}

---

**Parallelization Information**

ChangeFormat is reentrant and automatically parallelized (on tuple level).

---

**Possible Successors**

- DispImage

---

See also ZoomImageSize, ZoomImageFactor

---

**Alternatives**

- CropPart

---

**Module**

Image / region / XLD management

---

```c
[out] HImageX ImagePart HImageX.CropDomain ( )
```

```c
void HOperatorSetX.CropDomain ([in] IObjectX Image,
[out] HUntypedObjectX ImagePart )
```

Cut out of defined gray values.

The operator CropDomain cuts a rectangular area from the input images. This rectangle is the smallest surrounding rectangle of the domain of the input image. The new definition domain includes all pixels of the new image. The new image matrix has the size of the rectangle.

---

**Parameter**

- **Image** (input iconic) ...... (multichannel-)image(-array) \( \sim \) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Input image.
CropDomainRel cuts a rectangular area from the input images. The area is determined by the surrounding rectangle of the domain of the input image. The rectangle can be influenced by the control parameters to modify at the top (Top), at the left (Left), at the bottom (Bottom), and at the right (Right). Positive values result in a smaller, negative values in a larger size. If all parameters are set to zero, the region remains unchanged.

**Parameter**

- **Image** (input iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Input image.

- **ImagePart** (output iconic) …… (multichannel-)image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Image area.

- **Top** (input control) .......................................................... integer \(\sim\) long / VARIANT
  
  Number of rows clipped at the top.

  *Default Value*: -1

  *Suggested values*: Top \(\in\) \{-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20\}

- **Left** (input control) .......................................................... integer \(\sim\) long / VARIANT
  
  Number of columns clipped at the left.

  *Default Value*: -1

  *Suggested values*: Left \(\in\) \{-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20\}

- **Bottom** (input control) ...................................................... integer \(\sim\) long / VARIANT
  
  Number of rows clipped at the bottom.

  *Default Value*: -1

  *Suggested values*: Bottom \(\in\) \{-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20\}

- **Right** (input control) ...................................................... integer \(\sim\) long / VARIANT
  
  Number of columns clipped at the right.

  *Default Value*: -1

  *Suggested values*: Right \(\in\) \{-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20\}

**Result**

CropDomainRel returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty
input region via `SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

### Parallelization Information
`CropDomainRel` is reentrant and automatically parallelized (on tuple level, channel level).

### Possible Predecessors
ReduceDomain, Threshold, Connection, RegionGrowing, Pouring

**See also**

SmallestRectangle1, Intersection, GenRectangle1, ClipRegion

### Alternatives

CropDomain, CropRectangle1

### Module

Image / region / XLD management

```plaintext

```

Cut out a rectangular image area.

The operator `CropPart` cuts a rectangular area from the input images. The area is indicated by a rectangle (upper left corner and size). The area must be within the image. The definition domain includes all pixels of the new image. The new image matrix has the size of a rectangle.

### Parameter

- **Image** (input iconic) ... (multichannel-)image(-array) \(\sim\) `HImageX / IHObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Input image.

- **ImagePart** (output iconic) ... (multichannel-)image(-array) \(\sim\) `HImageX / HUntypedObjectX` (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Image area.

- **Row** (input control) ...................................................... `rectangle.origin.y \sim long / VARIANT`
  
  Line index of upper left corner of image area.

  **Default Value**: 100

  **Suggested values**: `Row \in \{10, 20, 50, 100, 200, 300, 500\}`

  **Typical range of values**: `0 \leq Row \leq 0`

- **Column** (input control) ...................................................... `rectangle.origin.x \sim long / VARIANT`
  
  Column index of upper left corner of image area.

  **Default Value**: 100

  **Suggested values**: `Column \in \{10, 20, 50, 100, 200, 300, 500\}`

  **Typical range of values**: `0 \leq Column \leq 0`

- **Width** (input control) ...................................................... `rectangle.extent.x \sim long / VARIANT`
  
  Width of new image.

  **Default Value**: 128

  **Suggested values**: `Width \in \{32, 64, 128, 256, 512, 768\}`

  **Typical range of values**: `0 \leq Width \leq 0`

- **Height** (input control) ...................................................... `rectangle.extent.y \sim long / VARIANT`
  
  Height of new image.

  **Default Value**: 128

  **Suggested values**: `Height \in \{32, 64, 128, 256, 512, 525\}`

  **Typical range of values**: `0 \leq Height \leq 0`

### Parallelization Information

`CropPart` is reentrant and automatically parallelized (on tuple level).
Possible Successors

DispImage

See also

ZoomImageSize, ZoomImageFactor

Alternatives

CropRectangle1, CropDomain, ChangeFormat, ReduceDomain

Module

Image / region / XLD management

Cut out a rectangular image area.

The operator CropRectangle1 cuts a rectangular area from the input images. The area is indicated by a rectangle (upper left and lower right corner). The area must be within the image. The definition domain includes all pixels of the new image. The new image matrix has the size of a rectangle.

Parameter

- **Image** (input iconic) ...... (multichannel-)image(-array) \(\Rightarrow\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Input image.

- **ImagePart** (output iconic) ...... (multichannel-)image(-array) \(\Rightarrow\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  
  Image area.

- **Row1** (input control) ....................... rectangle.origin.y \(\Rightarrow\) long / VARIANT
  
  Line index of upper left corner of image area.

  Default Value : 100

  Suggested values : Row1 \(\in\) \{10, 20, 50, 100, 200, 300, 500\}

  Typical range of values : \(0 \leq \text{Row1} \leq 0\)

- **Column1** (input control) ....................... rectangle.origin.x \(\Rightarrow\) long / VARIANT
  
  Column index of upper left corner of image area.

  Default Value : 100

  Suggested values : Column1 \(\in\) \{10, 20, 50, 100, 200, 300, 500\}

  Typical range of values : \(0 \leq \text{Column1} \leq 0\)

- **Row2** (input control) ....................... rectangle.corner.y \(\Rightarrow\) long / VARIANT
  
  Line index of lower right corner of image area.

  Default Value : 200

  Suggested values : Row2 \(\in\) \{10, 20, 50, 100, 200, 300, 500\}

  Typical range of values : \(0 \leq \text{Row2} \leq 0\)

- **Column2** (input control) ....................... rectangle.corner.x \(\Rightarrow\) long / VARIANT
  
  Column index of lower right corner of image area.

  Default Value : 200

  Suggested values : Column2 \(\in\) \{10, 20, 50, 100, 200, 300, 500\}

  Typical range of values : \(0 \leq \text{Column2} \leq 0\)

Parallelization Information

CropRectangle1 is reentrant and automatically parallelized (on tuple level).
TileChannles tiles an image consisting of multiple channels into a large single-channel image. The input image Image contains Num images of the same size, which are stored in the individual channels. The output image TiledImage contains a single channel image, where the Num input channels have been tiled into NumColumns columns. In particular, this means that TileChannels cannot tile color images. For this purpose, TileImages can be used. The parameter TileOrder determines the order in which the images are copied into the output in the cases in which this is not already determined by NumColumns (i.e., if NumColumns $\neq 1$ and NumColumns $\neq$ Num). If TileOrder = 'horizontal' the images are copied in the horizontal direction, i.e., the second channel of Image will be to the right of the first channel. If TileOrder = 'vertical' the images are copied in the vertical direction, i.e., the second channel of Image will be below the first channel. The domain of TiledImage is obtained by copying the domain of Image to the corresponding locations in the output image. If Num is not a multiple of NumColumns the output image will have undefined gray values in the lower right corner of the image. The output domain will reflect this.

Attention

Parameter

- **Image** (input iconic) ...... multichannel-image-array $\sim$ HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  - Input image.
- **TiledImage** (output iconic) ...... image(-array) $\sim$ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  - Tiled output image.
- **NumColumns** (input control) .......................................................integer $\sim$ long / VARIANT
  - Number of columns to use for the output image.
    - **Default Value:** 1
    - **Suggested values:** NumColumns $\in$ \{1, 2, 3, 4, 5, 6, 7\}
    - **Restriction:** (NumColumns $\geq$ 1)
- **TileOrder** (input control) .....................................................string $\sim$ String / VARIANT
  - Order of the input images in the output image.
    - **Default Value:** 'vertical'
    - **List of values:** TileOrder $\in$ {'horizontal', 'vertical'}

Example

/* Grab 5 single-channel images and stack them vertically. */
gen_rectangle1 (Image, 0, 0, Height-1, Width-1)
for I := 1 to 5 by 1
  grab_image_async (ImageGrabbed, FGHandle, -1)
  append_channel (Image, ImageGrabbed, Image)
endfor
tile_channels (Image, TiledImage, 1, 'vertical')

Result

TileChannels returns TRUE if all parameters are correct and no error occurs during execution. If the input
is empty the behavior can be set via `SetSystem(::'noObjectResult',<Result>;)`. If necessary, an exception handling is raised.

Parallelization Information

*TileChannels* is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

*AppendChannel*

See also

*ChangeFormat, CropPart, CropRectangle1*

Alternatives

*TileImages, TileImagesOffset*

Module

Image / region / XLD management

```

```

TileImages tiles multiple input image objects, which must contain the same number of channels, into a large image. The input image object *Images* contains *Num* images, which may be of different size. The output image *TiledImage* contains as many channels as the input images. In the output image the *Num* input images have been tiled into *NumColumns* columns. Each tile has the same size, which is determined by the maximum width and height of all input images. If an input image is smaller than the tile size it is copied to the center of the respective tile. The parameter *TileOrder* determines the order in which the images are copied into the output in the cases in which this is not already determined by *NumColumns* (i.e., if *NumColumns* != 1 and *NumColumns* != *Num*).

If *TileOrder* = 'horizontal' the images are copied in the horizontal direction, i.e., the second image of *Images* will be to the right of the first image. If *TileOrder* = 'vertical' the images are copied in the vertical direction, i.e., the second image of *Images* will be below the first image. The domain of *TiledImage* is obtained by copying the domains of *Images* to the corresponding locations in the output image. If *Num* is not a multiple of *NumColumns* the output image will have undefined gray values in the lower right corner of the image. The output domain will reflect this.

Attention

Parameter

- **Images** (input iconic) ...... (multichannel-)image  ∼  HImageX / IObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real )
  
  Input images.

- **TiledImage** (output iconic) ...... (multichannel-)image  ∼  HImageX / HUntypedObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real )
  
  Tiled output image.

- **NumColumns** (input control) ............................................integer  ∼  long / VARIANT
  
  Number of columns to use for the output image.

  Default Value : 1

  Suggested values : NumColumns ∈ {1, 2, 3, 4, 5, 6, 7}

  Restriction : (NumColumns ≥ 1)

- **TileOrder** (input control) ..................................................string  ∼  String / VARIANT
  
  Order of the input images in the output image.

  Default Value : 'vertical'

  List of values : TileOrder ∈ {'horizontal', 'vertical'}
/* Grab 5 (multi-channel) images and stack them vertically. */
gen_empty_obj (Images)
for I := 1 to 5 by 1
    grab_image_async (ImageGrabbed, FGHandle, -1)
    concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images (Images, TiledImage, 1, 'vertical')

TileImages returns TRUE if all parameters are correct and no error occurs during execution. If the input is empty the behavior can be set via SetSystem(‘noObjectResult’,<Result>:). If necessary, an exception handling is raised.

Parallelization Information
TileImages is reentrant and automatically parallelized (on channel level).

Possible Predecessors
AppendChannel
See also
ChangeFormat, CropPart, CropRectangle1

Alternatives
TileChannels, TileImagesOffset

Module
Image / region / XLD management

TileImages Offset tiles multiple input image objects, which must contain the same number of channels, into a large image. The input image object Images contains Num images, which may be of different size. The output image TiledImage contains as many channels as the input images. The size of the output image is determined by the parameters Width and Height. The position of the upper left corner of the input images in the output images is determined by the parameters OffsetRow and OffsetCol. Both parameters must contain exactly Num values. Optionally, each input image can be cropped to an arbitrary rectangle that is smaller than the input image. To do so, the parameters Row1, Col1, Row2, and Col2 must be set accordingly. If any of these four parameters is set to -1, the corresponding input image is not cropped. In any case, all four parameters must contain Num values. If the input images are cropped the position parameters OffsetRow and OffsetCol refer to the upper left corner of the cropped image. If the input images overlap each other in the output image (while taking into account their respective domains), the image with the higher index in Images overwrites the image data of the image with the lower index. The domain of TiledImage is obtained by copying the domains of Images to the corresponding locations in the output image.

Attention
If the input images all have the same size and tile the output image exactly, the operator TileImages usually will be slightly faster.
Parameter

- **Images** (input iconic) \[\ldots\] (multichannel-)image(-array) \[\sim\ HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)

  Input images.

- **TiledImage** (output iconic) \[\ldots\] (multichannel-)image \[\sim\ HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)

  Tiled output image.

- **OffsetRow** (input control) \[\ldots\] point.y(-array) \[\sim\ \text{VARIANT} (integer)

  Row coordinate of the upper left corner of the input images in the output image.

  **Default Value:** 0

  **Suggested values:** OffsetRow \[\in\{0, 50, 100, 150, 200, 250\}

- **OffsetCol** (input control) \[\ldots\] point.x(-array) \[\sim\ \text{VARIANT} (integer)

  Column coordinate of the upper left corner of the input images in the output image.

  **Default Value:** 0

  **Suggested values:** OffsetCol \[\in\{0, 50, 100, 150, 200, 250\}

- **Row1** (input control) \[\ldots\] rectangle.origin.y(-array) \[\sim\ \text{VARIANT} (integer)

  Row coordinate of the upper left corner of the copied part of the respective input image.

  **Default Value:** -1

  **Suggested values:** Row1 \[\in\{-1, 0, 10, 20, 50, 100, 200, 300, 500\}

- **Col1** (input control) \[\ldots\] rectangle.origin.x(-array) \[\sim\ \text{VARIANT} (integer)

  Column coordinate of the upper left corner of the copied part of the respective input image.

  **Default Value:** -1

  **Suggested values:** Col1 \[\in\{-1, 0, 10, 20, 50, 100, 200, 300, 500\}

- **Row2** (input control) \[\ldots\] rectangle.corner.y(-array) \[\sim\ \text{VARIANT} (integer)

  Row coordinate of the lower right corner of the copied part of the respective input image.

  **Default Value:** -1

  **Suggested values:** Row2 \[\in\{-1, 0, 10, 20, 50, 100, 200, 300, 500\}

- **Col2** (input control) \[\ldots\] rectangle.corner.x(-array) \[\sim\ \text{VARIANT} (integer)

  Column coordinate of the lower right corner of the copied part of the respective input image.

  **Default Value:** -1

  **Suggested values:** Col2 \[\in\{-1, 0, 10, 20, 50, 100, 200, 300, 500\}

- **Width** (input control) \[\ldots\] extent.x \[\sim\ long / \text{VARIANT}

  Width of the output image.

  **Default Value:** 512

  **Suggested values:** Width \[\in\{32, 64, 128, 256, 512, 768, 1024, 2048, 4096\}

- **Height** (input control) \[\ldots\] extent.y \[\sim\ long / \text{VARIANT}

  Height of the output image.

  **Default Value:** 512

  **Suggested values:** Height \[\in\{32, 64, 128, 256, 512, 525, 1024, 2048, 4096\}

---

**Example**

```c
/* Grab 2 (multi-channel) NTSC images, crop the bottom 5 lines of each image, the right 5 columns of the first image, and the left five lines of the second image, and put the cropped images side-by-side. */
gen_empty_obj (Images)
for I := 1 to 2 by 1
  grab_image_async (ImageGrabbed, FHandle, -1)
  concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images_offset (Images, TiledImage, [0,635], [0,0], [0,0], [0,5], [474,474], [634,639])
```

---

**Result**

`TileImagesOffset` returns TRUE if all parameters are correct and no error occurs during execution. If the
input is empty the behavior can be set via SetSystem(:,‘noObjectResult’,<Result>:). If necessary, an exception handling is raised.

---

**Parallelization Information**

TileImagesOffset is reentrant and automatically parallelized (on channel level).

---

**Possible Predecessors**

AppendChannel

---

**See also**

ChangeFormat, CropPart, CropRectangle1

---

**Alternatives**

TileChannels, TileImages

---

**Module**

Image / region / XLD management

---

### 5.7 Framegrabber

```cpp
void HMiscX.CloseAllFramegrabbers ( )
void HOperatorSetX.CloseAllFramegrabbers ( )
```

Close all frame grabbers.

The operator CloseAllFramegrabbers closes all frame grabbers. It is used to cope with deadlocks resulting from damaged frame grabber handles (in that case the use of CloseFramegrabber is impossible).

---

**Attention**

Since all frame grabbers are closed by CloseAllFramegrabbers all frame grabber handles become invalid.

---

**Result**

If it is possible to close the frame grabbers the operator CloseAllFramegrabbers returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

CloseAllFramegrabbers is local and processed completely exclusively without parallelization.

---

**Possible Predecessors**

GrabImage

---

**See also**

OpenFramegrabber

---

**Module**

Image / region / XLD management

---

```cpp
void HOperatorSetX.CloseFramegrabber ([in] VARIANT FGHandle )
```

Close specified frame grabber.

The operator CloseFramegrabber closes the frame grabber specified by FGHandle. In particular possible storage space for data buffers is released and the frame grabber is made available for other processes.

---

**Attention**

---

**Parameter**

- **FGHandle** (input control)          . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . framegrabber ~ HFramegrabberX / VARIANT Handle of the frame grabber to be closed.

---

**Result**

If the frame grabber is open the operator CloseFramegrabber returns the value TRUE. Otherwise an exception handling is raised.
5.7. FRAMEGRABBER

CloseFramegrabber is local and processed completely exclusively without parallelization.

Parallelization Information

Possible Predecessors

GrabImage

See also

OpenFramegrabber

Module

Image / region / XLD management

GetFramegrabberLut queries the lut of the frame grabber specified by FGHandle. This operation is not supported for all kinds of frame grabbers.

Parameter

- **FGHandle** (input control) framegrabber \(\sim HFramegrabberX / \text{VARIANT} \) Handle of the desired frame grabber.
- **ImageRed** (output control) integer \(\sim \text{VARIANT (integer)} \) Red level of the lut entries.
- **ImageGreen** (output control) integer \(\sim \text{VARIANT (integer)} \) Green level of the lut entries.
- **ImageBlue** (output control) integer \(\sim \text{VARIANT (integer)} \) Blue level of the lut entries.

Result

The operator GetFramegrabberLut returns the value TRUE if the frame grabber is open.

GetFramegrabberLut is reentrant and processed without parallelization.

Possible Predecessors

OpenFramegrabber

Possible Successors

SetFramegrabberLut

See also

SetFramegrabberLut, OpenFramegrabber

Module

Image / region / XLD management

Get specific parameters for a frame grabber.

The operator GetFramegrabberParam returns specific parameter values for the frame grabber specified by FGHandle. The standard parameters listed below are available for all frame grabbers. Additional parameters may be supported by specific frame grabbers. A list of those can be obtained with the query 'parameter' via InfoFramegrabber.

Standard values for Param, see OpenFramegrabber:

HALCON 6.1.4
‘name’ Name of the frame grabber interface.
‘horizontal_resolution’ Horizontal resolution of the frame grabber.
‘vertical_resolution’ Vertical resolution of the frame grabber.
‘image_width’ Width of the desired image part.
‘image_height’ Height of the desired image part.
‘start_row’ Row coordinate of upper left corner of desired image part.
‘start_column’ Column coordinate of upper left corner of desired image part.
‘field’ Selected video field or full frame.
‘bits_per_channel’ Number of transferred bits per pixel and image channel.
‘color_space’ Color space of resulting image.
‘gain’ Amplification factor for video amplifier.
‘external_trigger’ External triggering (‘true’ / ‘false’).
‘camera_type’ Type of used camera (frame grabber specific).
‘device’ Device the frame grabber is linked to.
‘port’ Port of the frame grabber the video input is linked to.
‘line_in’ Camera input line of multiplexer (optional).

Attention

Parameter

▷ FGHandle (input control) ................................. framegrabber ~ HFramegrabberX / VARIANT
Handle of the frame grabber to be used.
▷ Param (input control) ........................................ string(-array) ~ VARIANT( string )
Parameter of interest.
Default Value : ‘revision’
▷ Value (output control) ................................. string(-array) ~ VARIANT( string, real, integer )
Parameter value.

Result

If the frame grabber is open and the specified parameter is supported by the frame grabber, the operator
GetFramegrabberParam returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

GetFramegrabberParam is reentrant and processed without parallelization.

Possible Predecessors

OpenFramegrabber
GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync,
CloseFramegrabber

Possible Successors

See also

InfoFramegrabber, SetFramegrabberParam

Module

Image / region / XLD management

void HImageX.GrabImage ([in] HFramegrabberX FGHandle )
[out] HImageX Image HFramegrabberX.GrabImage ()

void HOperatorSetX.GrabImage ([out] HUntypedObjectX Image,
[in] VARIANT FGHandle )

Grab an image from the specified frame grabber.

HALCON/COM Reference Manual, 2005-2-1
The operator \texttt{GrabImage} grabs an image via the frame grabber specified by \texttt{FGHandle}. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator \texttt{OpenFramegrabber}. Additional (frame grabber specific) settings might be possible via \texttt{SetFramegrabberParam}.

\begin{verbatim}
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,'default',-1,'default',-1.0,
               'default','default','default',P,L,FGHandle)
grab_image(Img,FGHandle)
close_framegrabber(FGHandle)
\end{verbatim}

If the frame grabber is open the operator \texttt{GrabImage} returns the value TRUE. Otherwise an exception handling is raised.

\begin{verbatim}
GrabImageAsync is reentrant and processed without parallelization.
\end{verbatim}

\begin{verbatim}
// Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.
void HImageX.GrabImageAsync ([in] HFramegrabberX FGHandle,
[in] double MaxDelay )
[out] HImageX Image HFramegrabberX.GrabImageAsync
(void double MaxDelay )
void HOperatorSetX.GrabImageAsync ([out] HUntypedObjectX Image,
[in] VARIANT FGHandle, [in] VARIANT MaxDelay )
\end{verbatim}

Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.

The operator \texttt{GrabImageAsync} grabs an image via the frame grabber specified by \texttt{FGHandle} and starts the asynchronous grab of the next image. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator \texttt{OpenFramegrabber}. Additional (frame grabber specific) settings might be possible via \texttt{SetFramegrabberParam}.

The grab of the next image is finished by calling \texttt{GrabImageAsync} or \texttt{GrabRegionAsync}. If more than \texttt{MaxDelay} ms have passed since the asynchronous grab was started, a new image is grabbed (the asynchronously grabbed image is considered as too old). If a negative value is assigned to \texttt{MaxDelay} this control mechanism is deactivated.

Please note that if you call the operators \texttt{GrabImage} or \texttt{GrabRegion} after \texttt{GrabImageAsync}, the asynchronous grab started by \texttt{GrabImageAsync} is aborted and a new image is grabbed (and waited for).
CHAPTER 5. IMAGE

Attention

Parameter

▷ **Image** (output iconic) ......................... image \( \sim \) HIImageX / HUntypedObjectX( byte, int2 )
   Grabbed image.

▷ **FGHandle** (input control) ...................... framegrabber \( \sim \) HFramegrabberX / VARIANT
   Handle of the frame grabber to be used.

▷ **MaxDelay** (input control) ...................... number \( \sim \) double / VARIANT
   Maximum tolerated delay between the start of the asynchronous grab and the delivery of the image [ms].

Default Value : -1.0
Suggested values : MaxDelay \( \in \) \{ -1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9 \}

Example

// Select a suitable frame grabber FgName
open_framegrabber(FgName, 1, 1, 0, 0, 0, 0, 'default', -1.0, 'default', -1.0,
   'default', 'default', 'default', -1.0, FGHandle)
// grab image + start next grab
grab_image_async(Img1, FGHandle, -1.0)
// Process Img1 ...
// Finish asynchronous grab + start next grab
grab_image_async(Img2, FGHandle, -1.0)
close_framegrabber(FGHandle)

Result

If the frame grabber is open and supports asynchronous grabbing the operator GrabImageAsync returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

GrabImageAsync is reentrant and processed without parallelization.

Possible Predecessors

OpenFramegrabber

Possible Successors

GrabImageAsync, GrabRegionAsync, CloseFramegrabber

See also

GrabImageStart, OpenFramegrabber, InfoFramegrabber, SetFramegrabberParam

Module

Image / region / XLD management

```
void HFramegrabberX.GrabImageStart ([in] double MaxDelay )
void HOperatorSetX.GrabImageStart ([in] VARIANT FGHandle,
   [in] VARIANT MaxDelay )
```

Start an asynchronous grab of an image from the specified frame grabber.

The operator GrabImageStart starts an asynchronous grab of an image via the frame grabber specified by FGHandle. The desired operational mode of the frame grabber as well as a suitable image part can be adjusted via the operator OpenFramegrabber. Additional (frame grabber specific) settings might be possible via SetFramegrabberParam.

The grab is finished via GrabImageAsync or GrabRegionAsync. If one of those operators is called more than MaxDelay ms later, a new image is grabbed (the asynchronously grabbed image is considered as to old). If a negative value is assigned to MaxDelay this control mechanism is deactivated.

Please note that the operator GrabImageStart makes sense only when used together with GrabImageAsync or GrabRegionAsync. If you call the operators GrabImage or GrabRegion instead, the asynchronous grab started by GrabImageStart is aborted and a new image is grabbed (and waited for).
5.7. FRAMEGRABBER

---

Attention

Parameter

- **FGHandle** (input control) ........................ framegrabber \(\sim\) HFramegrabberX / VARIANT
  Handle of the frame grabber to be used.
- **MaxDelay** (input control) .......................... number \(\sim\) double / VARIANT
  Maximum tolerated delay between the start of the asynchronous grab and the delivery of the image [ms].
  Default Value : -1.0
  Suggested values : MaxDelay \(\in\) \{20.0, 33.3, 40.0, 66.6, 80.0, 99.9\}

---

Example

```
// Select a suitable frame grabber FgName
open_framegrabber(FgName,1,1,0,0,0,0,'default',-1.0,
                 'default','default','default',-1,-1,FgHandle)
grab_image(Img1,FgHandle)
// Start next grab
grab_image_start(FgHandle,-1.0)
// Process Img1 ...
// Finish asynchronous grab
grab_image_async(Img2,FgHandle,-1.0)
close_framegrabber(FgHandle)
```

---

Result

If the frame grabber is open and supports asynchronous grabbing the operator **GrabImageStart** returns the value TRUE. Otherwise an exception handling is raised.

---

Parallelization Information

**GrabImageStart** is reentrant and processed without parallelization.

---

Possible Predecessors

OpenFramegrabber

---

Possible Successors

GrabImageAsync, GrabRegionAsync, CloseFramegrabber

---

See also

OpenFramegrabber, InfoFramegrabber, SetFramegrabberParam

---

Module

Image / region / XLD management

---

```c
void HRegionX.GrabRegion ([in] HFramegrabberX FGHandle )
[out] HRegionX Region HFramegrabberX.GrabRegion ( )
void HOperatorSetX.GrabRegion ([out] HUntypedObjectX Region,
[in] VARIANT FGHandle )
```

Grab and segment an image from the specified frame grabber.

The operator **GrabRegion** grabs an image via the frame grabber specified by **FGHandle** and segments it. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator **OpenFramegrabber**. Additional (frame grabber specific) settings might be possible via **SetFramegrabberParam**.

---

Attention

Parameter

- **Region** (output iconic) .......................... region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Grabbed and segmented image: Region(s).
- **FGHandle** (input control) ........................ framegrabber \(\sim\) HFramegrabberX / VARIANT
  Handle of the frame grabber to be used.
Example

// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,0,'default',-1,'default',-1.0,
    'default','default','default',P,L,FgHandle)
// grab and segment image
grab_region(Region,FgHandle)
// Process Region ...
close_framegrabber(FgHandle)

Result

If the frame grabber is open the operator GrabRegion returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

GrabRegion is reentrant and processed without parallelization.

Possible Predecessors

OpenFramegrabber, GrabImageStart

Possible Successors

GrabRegion, GrabRegionAsync, GrabImageStart, GrabImage, GrabImageAsync, CloseFramegrabber

See also

OpenFramegrabber, InfoFramegrabber, SetFramegrabberParam

Module

Image / region / XLD management

```c
void HRegionX.GrabRegionAsync ([in] HFramegrabberX FGHandle,
    [in] double MaxDelay )
    ([out] HRegionX Region HFramegrabberX.GrabRegionAsync)

void HOperatorSetX.GrabRegionAsync ([out] HUntypedObjectX Region,
    [in] VARIANT FGHandle, [in] VARIANT MaxDelay )
```

Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.

The operator GrabRegionAsync grabs an image via the frame grabber specified by FGHandle, segments it, and starts the asynchronous grab of the next image. The desired operational mode of the frame grabber as well as a suitable image area can be adjusted via the operator OpenFramegrabber. Additional (frame grabber specific) settings might be possible via SetFramegrabberParam.

The grab of the next image is finished via GrabRegionAsync or via GrabRegionAsync. If more than MaxDelay ms have passed since the asynchronous grab was started, a new image is grabbed (the asynchronously grabbed image is considered as too old). If a negative value is assigned to MaxDelay this control mechanism is deactivated.

Please note that if you call the operators GrabImage or GrabRegion after GrabRegionAsync, the asynchronous grab started by GrabRegionAsync is aborted and a new image is grabbed (and waited for).

Attention

Parameter

- Region (output iconic) ..............................................region(-array) ~ HRegionX / HUntypedObjectX
  Grabbed and segmented image: Region(s).
- FGHandle (input control) ..............................................framegrabber ~ HFramegrabberX / VARIANT
  Handle of the frame grabber to be used.
5.7. FRAMEGRABBER

- **MaxDelay** (input control) .............................................. number ∼ double / VARIANT
  - Maximum tolerated delay between the start of the asynchronous grab and the delivery of the image [ms].
  - **Default Value**: -1.0
  - **Suggested values**: MaxDelay ∈ {-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9}

---

**Example**

```c
// Select a suitable frame grabber FgName
open_framegrabber(FgName,1,1,0,0,0,'default',-1,'default',-1.0,
                 'default','default','default',-1,-1,FgHandle)
// grab image, segment it, and start next grab
grab_region_async(Region1,FgHandle,-1.0)
// Process Region1 ...
// Finish asynchronous grab, segment this image, and start next grab
grab_region_async(Region2,FgHandle,-1.0)
close_framegrabber(FgHandle)
```

---

**Result**

If the frame grabber is open the operator **GrabRegionAsync** returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

**GrabRegionAsync** is reentrant and processed without parallelization.

---

**Possible Predecessors**

OpenFramegrabber, GrabImageStart

---

**Possible Successors**

GrabRegionAsync, GrabImageAsync, CloseFramegrabber

---

**See also**

OpenFramegrabber, InfoFramegrabber, SetFramegrabberParam

---

**Module**

Image / region / XLD management

---

```c
[out] String Information HInfoX.InfoFramegrabber ([in] String Name,
[in] String Query, [out] VARIANT ValueList )

void HOperatorSetX.InfoFramegrabber ([in] VARIANT Name,
```

**Information about the specified frame grabber.**

The operator **InfoFramegrabber** returns information about the frame grabber **Name**. The desired information is specified via **Query**. A textual description according to the selected topic is returned in **Information**. If applicable, **ValueList** contains a list of supported values. Up to now, the following queries are possible:

- **'camera types'**: Description of the frame grabber specific parameter `CameraType`, see OpenFramegrabber.
- **'defaults'**: Framegrabber specific default values in **ValueList**, see OpenFramegrabber.
- **'general'**: General information (in **Information**).
- **'info boards'**: Information about currently installed boards, connected ports, etc. This data is especially useful for the auto-detect functionality of ActivVisionTools.
- **'parameters'**: List of all frame grabber specific parameters which are accessible via SetFramegrabberParam and GetFramegrabberParam.
- **'ports'**: Description of the ports (signal, connectors etc.) in **Information** and the port numbers in **ValueList**.
- **'revision'**: Version number of the frame grabber interface.

Please check also the directory `doc/html/manuals` for files describing selected frame grabbers.
CHAPTER 5. IMAGE

Parameter

- **Name** (input control) .................................................. string  \(\sim\) String / VARIANT
  Frame grabber of interest.

  **Default Value:** 'File'
  **Suggested values:** Name \(\in\) \{'Barracuda', 'Bcam1394', 'BitFlow', 'CCi4', 'DFG-BW', 'DFG-LC',
  'DirectShow', 'DqVll', 'DT315x', 'DT3162', 'File', 'Fire-i', 'FirePackage', 'FlashBus', 'FlashBusMX',
  'Ginga', 'Ginga++', 'IDS', 'Inspecta', 'MatrixVision', 'MultiCam', 'Opteon', 'p3i2', 'p3i4', 'PCEye',
  'PicPort', 'PicPortPro', 'PicProdigy', 'PX', 'PXC', 'PXD', 'PXR', 'RamsesI', 'TWAIN', 'uEye'

- **Query** (input control) .................................................. string  \(\sim\) String / VARIANT
  Name of the chosen query.

  **Default Value:** 'info_boards'
  **List of values:** Query \(\in\) \{'camera_types', 'defaults', 'general', 'info_boards', 'parameters', 'ports',
  'revision'

- **Information** (output control) ........................................ string  \(\sim\) String / VARIANT
  Textual information (according to Query).

- **ValueList** (output control) .......................... string(-array)  \(\sim\) VARIANT (string, integer, real)
  If applicable, a list of values (according to Query).

Example

```c
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connect ed to
open_framegrabber(FgName,1,1,0,0,0,0,'default','-1','default','-1.0,
  'default','default','default',P,L,FgHandle)
grab_image(Img,FgHandle)
close_framegrabber(FgHandle)
```

Result

If the parameter values are correct and the desired frame grabber is available at call time, **InfoFramegrabber**
returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

**InfoFramegrabber** is local and processed completely exclusively without parallelization.

Possible Predecessors

**OpenFramegrabber**

Possible Successors

**OpenFramegrabber**

See also

**OpenFramegrabber**

Module

Image / region / XLD management
Open and configure a frame grabber.

The operator **OpenFramegrabber** opens and configures the chosen frame grabber. During this process the connection to the frame grabber is tested, the frame grabber is (normally) locked for other processes, and if necessary memory is reserved as data buffer. The image is actually grabbed via the operators **GrabImage**, **GrabRegion**, **GrabImageAsync**, or **GrabRegionAsync**. If the frame grabber is not needed anymore, it should be closed via the operator **CloseFramegrabber**, releasing it for other processes. This automatically happens when a frame grabber is reopened. Some frame grabbers allow to open several instances of the same frame grabber class (up to 5).

For some parameters frame grabber specific default values can be chosen explicitly (see the parameter description below). Additional information for a specific frame grabber is available via **InfoFramegrabber**. Please check also the directory `doc/html/manuals` for files describing selected frame grabbers.

The meaning of the particular parameters is as follows:

- **HorizontalResolution**, **VerticalResolution** Desired resolution of the frame grabber.
- **ImageWidth**, **ImageHeight** Size of the image area to be returned by **GrabImage** etc.
- **StartRow**, **StartColumn** StartRow, StartColumn upper left corner of the desired image area.
- **Field** Desired half image (`'first'`, `'second'`, or `'next'`) or selection of a full image.
- **BitsPerChannel** Number of bits, which are transferred from the frame grabber board per pixel and image channel (typically 5, 8, 10, 12, or 16 Bits).
- **ColorSpace** Specify to grab single-channel (`'gray'`) or three-channel images (`'rgb'`, `'yuv'`, ...).
- **Gain** Amplification factor for the video amplifier (if available).
- **ExternalTrigger** Activation of external triggering (if available).
- **CameraType** Frame grabber specific parameter (string type) to specify the type of the used camera, which can be inquired via **InfoFramegrabber**.
- **Device** Device name of the frame grabber card.
- **Port** Port of the frame grabber the video signal is connected to.
- **LineIn** Camera input line of multiplexer (if available).

The operator **OpenFramegrabber** returns a handle (**FGHandle**) to the opened frame grabber.

---

**Attention**

Due to the multitude of supported frame grabbers a large number of parameters is necessary for the operator **OpenFramegrabber**. However, not all parameters are needed for a specific frame grabber.

---

- **Name** (input control) ................................. string  ~ String / VARIANT
- **HALCON frame grabber interface**, i.e. name of the corresponding DLL (Windows) or shared library (UNIX).
- **Default Value** : 'File'
CHAPTER 5. IMAGE

- **HorizontalResolution** (input control) ........................................... extent.x $\sim$ long / VARIANT
  Desired horizontal resolution of frame grabber (1: full resolution, 2: half resolution, 4: quarter resolution).
  Default Value : 1
  Suggested values: HorizontalResolution $\in \{1, 2, 4, 768, 720, 640, 384, 320, 192, 160\}$

- **VerticalResolution** (input control) ............................................. extent.y $\sim$ long / VARIANT
  Desired vertical resolution of frame grabber (1: full resolution, 2: half resolution, 4: quarter resolution).
  Default Value : 1
  Suggested values: VerticalResolution $\in \{1, 2, 4, 576, 480, 288, 240, 144, 120\}$

- **ImageWidth** (input control) ..................................................... rectangle.extent.x $\sim$ long / VARIANT
  Width of desired image part (0: HorizontalResolution - 2*StartColumn).
  Default Value : 0
  Suggested values: ImageWidth $\in \{-1, 0\}$

- **ImageHeight** (input control) .................................................... rectangle.extent.y $\sim$ long / VARIANT
  Height of desired image part (0: VerticalResolution - 2*StartRow).
  Default Value : 0
  Suggested values: ImageHeight $\in \{-1, 0\}$

- **StartRow** (input control) ......................................................... rectangle.origin.y $\sim$ long / VARIANT
  Line number of upper left corner of desired image part, for ImageHeight = 0: Height of a border.
  Default Value : 0
  Suggested values: StartRow $\in \{-1, 0\}$

- **StartColumn** (input control) .................................................... rectangle.origin.x $\sim$ long / VARIANT
  Column number of upper left corner of desired image part, for ImageWidth = 0: Width of a border.
  Default Value : 0
  Suggested values: StartColumn $\in \{-1, 0\}$

- **Field** (input control) .............................................................. string $\sim$ String / VARIANT
  Desired half image or for full image.
  Default Value: 'default'

- **BitsPerChannel** (input control) ............................................... integer(-array) $\sim$ VARIANT (integer)
  Number of transferred bits per pixel and image channel (-1: frame grabber specific default value).
  Default Value : -1
  Suggested values: BitsPerChannel $\in \{5, 8, 10, 12, 16, -1\}$

- **ColorSpace** (input control) ................................................... string(-array) $\sim$ VARIANT (string)
  Specify to grab single-channel (‘gray’) or three-channel (‘rgb’, ‘yuv’, ...) images (‘default’: frame grabber specific default value, mostly ‘rgb’).
  Default Value : ‘default’
  Suggested values: ColorSpace $\in \{\text{‘gray’, ‘rgb’, ‘yuv’, ‘default’}\}$

- **Gain** (input control) ............................................................. real $\sim$ double / VARIANT
  Amplification factor for video amplifier (-1.0: frame grabber specific default value).
  Default Value : -1.0
  Suggested values: Gain $\in \{0.25, 0.5, 0.75, 1.0, -1.0\}$

- **ExternalTrigger** (input control) ............................................. string $\sim$ String / VARIANT
  External triggering.
  Default Value: ‘default’
  List of values: ExternalTrigger $\in \{\text{‘true’, ‘false’, ‘default’}\}$

- **CameraType** (input control) .................................................. string(-array) $\sim$ VARIANT (string)
  Type of used camera (frame grabber specific) (‘default’: frame grabber specific default value).
  Default Value : ‘default’
  Suggested values: CameraType $\in \{\text{‘ntsc’, ‘pal’, ‘auto’, ‘default’}\}$

- **Device** (input control) ......................................................... string(-array) $\sim$ VARIANT (string)
  Device the frame grabber is linked to (‘default’: frame grabber specific default value).
  Default Value : ‘default’
  Suggested values: Device $\in \{\text{‘-1’, ‘0’, ‘1’, ‘2’, ‘3’, ‘default’}\}$

- **Port** (input control) ............................................................ integer(-array) $\sim$ VARIANT (integer)
  Port of the frame grabber the video signal is linked to (-1: frame grabber specific default value).
  Default Value : -1
  Suggested values: Port $\in \{0, 1, 2, 3, -1\}$
5.7. FRAMEGRABBER

- **LineIn** (input control) ........................................... integer(-array) \(\sim\) VARIANT (integer)
  Camera input line of multiplexer.
- **Default Value** : -1
- **Suggested values** : \(\text{LineIn} \in \{1, 2, 3, 4, -1\}\)

- **FGHandle** (output control) ..................................... framegrabber \(\sim\) HFramegrabberX / VARIANT
  Handle of the opened frame grabber.

---

**Example**

```c
// Select a suitable frame grabber FgName
info_framegrabber(FgName,'ports',Information,Val)
// Choose the port P and the input line L your camera is connected to
open_framegrabber(FgName,1,1,0,0,0,'default',-1,'default',-1.0,
                  'default','default','default',P,L,FgHandle)
grab_image(Img,FgHandle)
close_framegrabber(FgHandle)
```

---

**Result**

If the parameter values are correct and the desired frame grabber is available at call time, `OpenFramegrabber` returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

`OpenFramegrabber` is *local* and processed *completely exclusively* without parallelization.

---

**Possible Predecessors**

- InfoFramegrabber

---

**Possible Successors**

- GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync,
- SetFramegrabberParam

---

**See also**

- InfoFramegrabber, CloseFramegrabber, GrabImage

---

**Module**

Image / region / XLD management

---

```c
void HFramegrabberX.SetFramegrabberLut ([in] VARIANT ImageRed,
[in] VARIANT ImageGreen, [in] VARIANT ImageBlue )
void HOperatorSetX.SetFramegrabberLut ([in] VARIANT FGHandle,
```

---

**Set frame grabber lut.**

The operator `SetFramegrabberLut` sets the lut of the frame grabbers specified by `FGHandle`. This operation is not supported for all kinds of frame grabbers.

---

**Parameter**

- **FGHandle** (input control) ..................................... framegrabber \(\sim\) HFramegrabberX / VARIANT
  Handle of the desired frame grabber.
- **ImageRed** (input control) ....................................... integer \(\sim\) VARIANT (integer)
  Red level of the lut entries.
- **ImageGreen** (input control) ...................................... integer \(\sim\) VARIANT (integer)
  Green level of the lut entries.
- **ImageBlue** (input control) ....................................... integer \(\sim\) VARIANT (integer)
  Blue level of the lut entries.

---

**Result**

The operator `SetFramegrabberLut` returns the value TRUE if the transferred lut is correct and the frame grabber is open.

---

**Parallelization Information**

`SetFramegrabberLut` is *reentrant* and processed *without* parallelization.
CHAPTER 5. IMAGE

Possible Predecessors
OpenFramegrabber, GetFramegrabberLut

Possible Successors
GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync

See also
GetFramegrabberLut, OpenFramegrabber

Module
Image / region / XLD management


Set specific parameters for a frame grabber.
The operator SetFramegrabberParam sets specific parameters for the frame grabber specified by FGHandle.

Parameter

FGHandle (input control) .................. framegrabber ~ HFramegrabberX / VARIANT Handle of the frame grabber to be used.

Param (input control) .................. string(-array) ~ VARIANT ( string ) Parameter to be set.

Value (input control) .................. string(-array) ~ VARIANT ( string, real, integer ) Parameter value.

Result
If the frame grabber is open and the specified parameter / parameter value is supported by the frame grabber, the operator SetFramegrabberParam returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
SetFramegrabberParam is reentrant and processed without parallelization.

Possible Predecessors
OpenFramegrabber

Possible Successors
GrabImage, GrabRegion, GrabImageStart, GrabImageAsync, GrabRegionAsync, CloseFramegrabber

See also
OpenFramegrabber, InfoFramegrabber, GetFramegrabberParam

Module
Image / region / XLD management

5.8 Manipulation

[out] HImageX MixedImage HImageX.PaintGray ([in] HImageX ImageDestination )


Paint the gray values of an image into another image.

HALCON/COM Reference Manual, 2005-2-1
PaintGray paints the gray values of the image given in ImageSource into the image in ImageDestination and returns the resulting image in MixedImage. Only the gray values of the domain of ImageSource are copied (see ReduceDomain).

Parameter

- **ImageSource** (input iconic) ...... image \(\sim HImageX / IOHObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image containing the desired gray values.

- **ImageDestination** (input iconic) ...... image \(\sim HImageX / IOHObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Input image to be painted over.

- **MixedImage** (output iconic) ...... image \(\sim HImageX / HUntypedObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
  - Result image.

Example

/* Copy a circular part of the image ’monkey’ into a new image (New1): */

read_image(Image,’monkey’)
gen_circle(Circle,200,200,150)
reduce_domain(Image,Circle,Mask)
/* New image with black (0) background */
gen_image_proto(Image,New1,0.0)
/* Copy a part of the image ’monkey’ into New1 */
paint_gray(Mask,New1,New1).

Result

PaintGray returns TRUE if all parameters are correct. If necessary, an exception is raised.

Parallelization Information

PaintGray is reentrant and processed without parallelization.

Possible Predecessors

ReadImage, GenImageConst, GenImageProto

See also

PaintRegion

Alternatives

GetImagePointer1, SetGrayval, CopyImage

Module

Basic operators

```c
```

```c
```

Paint regions into an image.

PaintRegion paints the regions given in Region with a constant gray value into the image given in Image and returns the result in ImageResult. These gray values can either be specified for each channel once, valid for all regions, or for each region separately. To define the latter, group the channel gray values \(g\) of each region and concatenate them to a tuple according to the regions’ order, e.g., for a three channel image:

\[
g(channel1,region1), g(channel2,region1), g(channel3,region1), g(channel1,region2), \ldots
\]

The parameter Type determines whether the region should be painted filled (’fill’) or whether only its boundary should be painted (’margin’).
CHAPTER 5. IMAGE

Parameter

- **Region** (input iconic) .................................................. region(-array) \(\sim\) HRegionX / IHObjectX
  Regions to be painted into the input image.

- **Image** (input iconic) ...... image \(\sim\) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex)
  Image in which the regions are to be painted.

- **ImageResult** (output iconic) ...... image \(\sim\) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex)
  Image containing the result.

- **Grayval** (input control) ....................... number \(\sim\) VARIANT (integer, real)
  Desired gray values of the regions.
  Default Value : 255.0
  Suggested values : Grayval \(\in\) \{0.0, 1.0, 2.0, 5.0, 10.0, 16.0, 32.0, 64.0, 128.0, 253.0, 254.0, 255.0\}

- **Type** (input control) .......................................................... string \(\sim\) String / VARIANT
  Paint regions filled or as boundaries.
  Default Value : 'fill'
  List of values : Type \(\in\) \{'fill’, ‘margin’\}

Example

```c
/* Paint a rectangle into a new image (New1) */
gen_rectangle1(Rectangle, 100.0, 100.0, 300.0, 300.0)
/* generate a black image */
gen_image_const(New1, "byte", 768, 576)
/* paint a white rectangle */
paint_region(Rectangle, New1, New1, 255.0, 'fill',).
```

Result

**PaintRegion** returns TRUE if all parameters are correct. If the input is empty the behavior can be set via **SetSystem(‘noObjectResult’,<Result>):**. If necessary, an exception is raised.

Parallelization Information

**PaintRegion** is reentrant and processed without parallelization.

Possible Predecessors

ReadImage, GenImageConst, GenImageProto, ReduceDomain

See also

ReduceDomain, SetDraw, GenImageConst

Alternatives

SetGrayval, PaintGray, PaintXld

Module

Basic operators

```c

```

**Paint XLD objects into an image.**

**PaintXld** paints the XLD objects XLD of type contour or polygon with the constant gray values Grayval into each channel of the background image given in **Image** and returns the result in **ImageResult**. Open contours of XLD objects are closed and their enclosed regions are filled up. The rim of the subpixel XLD objects is painted onto the background image using anti-aliasing. Note that only objects without crossings or touching segments are painted correctly.
Grayval contains the gray values for painting the XLD objects. These gray values can either be specified for each channel once, valid for all XLD objects, or for each XLD object separately. To define the latter, group the channel gray values \( g \) of each XLD object and concatenate them to a tuple according to the order of the XLD objects, e.g., for a three channel image:

\[
[g(channel1,xld1), g(channel2,xld1), g(channel3,xld1), g(channel1,xld2), ...]
\]

Parameter

- **XLD** (input iconic) \( \ldots \) \( \text{xld-array} \) \( \sim \) \( \text{IHLDX / IHOBJECTX} \)
  XLD objects to be painted into the input image.
- **Image** (input iconic) \( \ldots \) \( \text{image} \) \( \sim \) \( \text{HIMAGEX / IHOBJECTX} \)
  Byte, direction, cyclic, int1, int2, uint2, int4, real, complex
  Image in which the xld objects are to be painted.
- **ImageResult** (output iconic) \( \ldots \) \( \text{image} \) \( \sim \) \( \text{HIMAGEX / HUNTYPEDOBJECTX} \)
  Image containing the result.
- **Grayval** (input control) \( \ldots \) \( \text{number} \) \( \sim \) \( \text{VARIANT} \) (integer, real)
  Desired gray value of the xld object.
  **Default Value:** 255.0
  **Suggested values:** \( \text{Grayval} \in \{0.0, 1.0, 2.0, 5.0, 10.0, 16.0, 32.0, 64.0, 128.0, 253.0, 254.0, 255.0\} \)

Example

/* Paint colored xld objects into a gray image */
/* read and copy image to generate a three channel image */
read_image(Image1,'green-dot')
copy_image(Image1,Image2)
copy_image(Image1,Image3)
compose3(Image1,Image2,Image3,Image)
/* extract subpixel border */
threshold_sub_pix(Image1,Border,128)
/* select the circle and the arrows */
select_obj(Border,circle,14)
select_obj(Border,arrows,16)
concat_obj(circle,arrows,green_dot)
/* paint a green circle and white arrows (to paint all * objects e.g. blue, pass \([0,0,255]\) tuple for GrayVal) */
paint_xld(green_dot,Image,ImageResult,[0,255,0,255,255,255])

Result

PaintXld returns TRUE if all parameters are correct. If the input is empty the behavior can be set via `SetSystem(::'noObjectResult',<Result>:)`. If necessary, an exception is raised.

Parallelization Information

PaintXld is reentrant and processed without parallelization.

Possible Predecessors

ReadImage, GenImageConst, GenImageProto, GenContourPolygonXld, ThresholdSubPix

See also

GenImageConst

Alternatives

SetGrayval, PaintGray, PaintRegion

Module

Basic operators
Set single gray values in an image.

SetGrayval sets the gray values of the input image \texttt{Image} at the positions (Row,Column) to the values specified by \texttt{Grayval}. The number of values in \texttt{Grayval} must match the number of points passed to the operator.

- **Image** (input iconic) \ldots \texttt{image} $\rightsquigarrow$ \texttt{HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)}
  - Image to be modified.
- **Row** (input control) \ldots \texttt{point.y(-array)} $\rightsquigarrow$ \texttt{VARIANT (integer)}
  - Row coordinates of the pixels to be modified.
  - Default Value: 0
  - Suggested values: Row $\in \{0, 10, 50, 127, 255, 511\}$
  - Restriction: (Row $\wedge$ (Image $<$ height))
- **Column** (input control) \ldots \texttt{point.x(-array)} $\rightsquigarrow$ \texttt{VARIANT (integer)}
  - Column coordinates of the pixels to be modified.
  - Default Value: 0
  - Suggested values: Column $\in \{0, 10, 50, 127, 255, 511\}$
  - Restriction: (Column $\wedge$ (Image $<$ width))
- **Grayval** (input control) \ldots \texttt{grayval(-array)} $\rightsquigarrow$ \texttt{VARIANT (integer, real)}
  - Gray values to be used.
  - Default Value: 255.0
  - Suggested values: Grayval $\in \{0.0, 1.0, 10.0, 128.0, 255.0\}$

\texttt{SetGrayval} returns TRUE if all parameters are correct. If the input is empty the behavior can be set via \texttt{SetSystem(::‘noObjectResult’,<Result>:)}. If necessary, an exception is raised.

\texttt{SetGrayval} is reentrant and processed without parallelization.

Possible Predecessors
- ReadImage, getImagePointer1, GenImageProto, GenImage1
- GetGrayval, GenImageConst, GenImage1, GenImageProto

See also
- Alternatives
- GetImagePointer1, PaintGray, PaintRegion

Module

Basic operators

---

5.9 Type-Conversion

\texttt{HImageX.ComplexToReal} (\texttt{ImageComplexToReal})

\texttt{HOperatorSetX.ComplexToReal} (\texttt{ComplexToReal})

Convert a complex image into two real images.

\texttt{ComplexToReal} converts a complex image \texttt{ImageComplex} into two real images \texttt{ImageReal} and \texttt{ImageImaginary}, which contain the real and imaginary part of the complex image.
5.9. TYPE-CONVERSION

Parameter

▷ ImageComplex (input iconic) ......................... image(-array)  \(\sim HImageX / IHObjectX\) (complex)
  Complex image.

▷ ImageReal (output iconic) .......................... image(-array)  \(\sim HImageX / HUntypedObjectX\) (real)
  Real part.

▷ ImageImaginary (output iconic) ...................... image(-array)  \(\sim HImageX / HUntypedObjectX\) (real)
  Imaginary part.

Parallelization Information

ComplexToReal is reentrant and automatically parallelized (on tuple level).

RealToComplex

See also

Image / region / XLD management

Module

\[
\begin{align*}
&\text{void} \quad \text{HOperatorSetX.ConvertImageType}(\text{[in]} \ IHObjectX \text{Image}, \\
&\text{[out]} \quad \text{HUntypedObjectX \text{ImageConverted}}, \text{[in]} \ \text{VARIANT \ NewType})
\end{align*}
\]

Convert the type of an image.

ConvertImageType converts images of an arbitrary type into an arbitrary new image type. If the conversion is done from a larger to a smaller gray value range (e.g., from ‘int4’ to ’byte’), too large or too small values are simply “clipped.” It is therefore advisable to adapt the range of gray values by calling ScaleImage before calling this operator.

Attention

If the source and destination image type are identical, no new image matrix is allocated.

Parameter

▷ Image (input iconic) ..................... (multichannel-)image(-array)  \(\sim HImageX / IHObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real)
  Image whose image type is to be changed.

▷ ImageConverted (output iconic) ........... (multichannel-)image(-array)  \(\sim HImageX / HUntypedObjectX\) (byte, direction, cyclic, int1, int2, uint2, int4, real, complex)
  Converted image.

▷ NewType (input control) .......................... string  \(\sim String / \text{VARIANT}
  Desired image type (i.e., type of the gray values).
  Default Value : ’byte’

Result

ConvertImageType returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via SetSystem(‘noObjectResult’, <Result>). If necessary, an exception is raised.

Parallelization Information

ConvertImageType is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors

ScaleImage

See also

ScaleImage, AbsImage

Image / region / XLD management

HALCON 6.1.4
Convert a displacement vector field into two int1-images.

DvfToInt1 converts the displacement vector field VectorField into two int1-images Row and Col. The output images contain the displacements in $x$- and $y$-direction, respectively.

Parameter

- **VectorField** (input iconic) ........................................ image(-array) $\rightarrow$ HImageX / IObjectIdX ( dvf )
  Displacement vector field.
- **Row** (output iconic) ................................. image(-array) $\rightarrow$ HImageX / HUntypedObjectX ( int1 )
  Displacement along $y$.
- **Col** (output iconic) ................................. image(-array) $\rightarrow$ HImageX / HUntypedObjectX ( int1 )
  Displacement along $x$.

Parallelization Information

DvfToInt1 is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

MeanImage, OpticalFlowMatch, FillDvf

See also

OpticalFlowMatch

Module

Image / region / XLD management

Convert two int1-images into a displacement vector field.

Int1ToDvf converts two int1-images Row and Col into a displacement vector field VectorField. The input images contain the displacements in $x$- and $y$-direction, respectively.

Parameter

- **Row** (input iconic) ................................. image(-array) $\rightarrow$ HImageX / IObjectIdX ( int1 )
  Displacement along $y$.
- **Col** (input iconic) ................................. image(-array) $\rightarrow$ HImageX / IObjectIdX ( int1 )
  Displacement along $x$.
- **VectorField** (output iconic) ......................... image(-array) $\rightarrow$ HImageX / HUntypedObjectX ( dvf )
  Displacement vector field.

Parallelization Information

Int1ToDvf is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

DvfToInt1

Possible Successors

OpticalFlowMatch, FillDvf

Module

Image / region / XLD management
Convert two real images into a complex image.

RealToComplex converts two real images ImageReal and ImageImaginary, which contain the real and imaginary part of a complex image, into a complex image ImageComplex.

Parameter

- **ImageReal** (input iconic) image-array \( \sim \) HImageX / IHObjectX (real) Real part.
- **ImageImaginary** (input iconic) image-array \( \sim \) HImageX / IHObjectX (real) Imaginary part.
- **ImageComplex** (output iconic) image-array \( \sim \) HImageX / HUntypedObjectX (complex) Complex image.

Parallelization Information

RealToComplex is reentrant and automatically parallelized (on tuple level).

See also

ComplexToReal Module

Image / region / XLD management
Approximate a contour by arcs and lines.

The coordinates of a curve are approximated by a row of lines and arcs. The procedure tries values from a user-definable range for certain parameters. The limits of these ranges are explicitly stated in the parameter list of the function (MinWidthCoord ... MaxWidthCoord, ThreshStart ... ThreshEnd, MinWidthSmooth ... MaxWidthSmooth, MinWidthCurve ... MaxWidthCurve). Additionally, the step width for the parameter area of the threshold value for pointed corners has to be indicated (ThreshStep). By narrowing the covered areas the runtime of the calculation can be shortened, but the result may deteriorate.

The parameters Weight1, Weight2 and Weight3 indicate whether the desired weighting is placed more on precision of the approximation, obtaining as much large segments as possible or as few small segments as possible. Thus, for (Weight1,Weight2,Weight3) (1,0,0) creates a very precise approximation and (0,1,1) an approximation with as few large segments as possible.

The result of the procedure is returned separately as arcs and lines. If one is interested in the sequence of the segments the individual resulting elements can be read successively from the resulting tuples; the sequence can be taken from the return parameter order (0: next element is next line segment, 1: next element is next arc segment).

Contours which can possibly consist of only one segment should also be examined with a threshold maximum (ThreshEnd) > 1.0, because otherwise at least one “corner point” is determined in any case.
Parameter

- **Row** (input control) ........................................ point.y ~ long / VARIANT
  Row of the contour.
  Default Value : 32

- **Column** (input control) ................................. point.x ~ long / VARIANT
  Column of the contour.
  Default Value : 32

- **MinWidthCoord** (input control) ..................... real ~ double / VARIANT
  Minimum width of Gauss operator for coordinate smoothing (> 0.4).
  Default Value : 0.5
  Suggested values : MinWidthCoord ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
  Typical range of values : 0.4 ≤ MinWidthCoord ≤ 0.4(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1

- **MaxWidthCoord** (input control) ..................... real ~ double / VARIANT
  Maximum width of Gauss operator for coordinate smoothing (> 0.4).
  Default Value : 2.4
  Suggested values : MaxWidthCoord ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
  Typical range of values : 0.4 ≤ MaxWidthCoord ≤ 0.4(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1

- **ThreshStart** (input control) .......................... real ~ double / VARIANT
  Minimum threshold value of the curvature for accepting a corner (relative to the largest curvature present).
  Default Value : 0.3
  Suggested values : ThreshStart ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
  Typical range of values : 0.1 ≤ ThreshStart ≤ 0.1(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1

- **ThreshEnd** (input control) ............................. real ~ double / VARIANT
  Maximum threshold value of the curvature for accepting a corner (relative to the largest curvature present).
  Default Value : 0.9
  Suggested values : ThreshEnd ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8}
  Typical range of values : 0.1 ≤ ThreshEnd ≤ 0.1(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1

- **ThreshStep** (input control) ........................... real ~ double / VARIANT
  Step width for threshold increase.
  Default Value : 0.2
  Suggested values : ThreshStep ∈ {0.3, 0.4, 0.5}
  Typical range of values : 0.1 ≤ ThreshStep ≤ 0.1(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1

- **MinWidthSmooth** (input control) ................... real ~ double / VARIANT
  Minimum width of Gauss operator for smoothing the curvature function (> 0.4).
  Default Value : 0.5
  Suggested values : MinWidthSmooth ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
  Typical range of values : 0.4 ≤ MinWidthSmooth ≤ 0.4(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1

- **MaxWidthSmooth** (input control) .................... real ~ double / VARIANT
  Maximum width of Gauss operator for smoothing the curvature function.
  Default Value : 2.4
  Suggested values : MaxWidthSmooth ∈ {0.5, 0.7, 1.0, 1.2, 1.5, 1.7}
  Typical range of values : 0.4 ≤ MaxWidthSmooth ≤ 0.4(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1
6.1. ACCESS

- MinWidthCurve (input control) ...................................................... integer  \(\sim long / VARIANT\)
  Minimum width of curve area for curvature determination (> 0.4).
  Default Value : 2
  Suggested values : MinWidthCurve \(\in\) \{2, 5, 7\}
  Typical range of values : \(1 \leq \text{MinWidthCurve} \leq 1(\text{lin})\)
  Minimum Increment : 1
  Recommended Increment : 2

- MaxWidthCurve (input control) ..................................................... integer  \(\sim long / VARIANT\)
  Maximum width of curve area for curvature determination.
  Default Value : 12
  Suggested values : MaxWidthCurve \(\in\) \{2, 5, 7\}
  Typical range of values : \(1 \leq \text{MaxWidthCurve} \leq 1(\text{lin})\)
  Minimum Increment : 1
  Recommended Increment : 2

- Weight1 (input control) ............................................................... real  \(\sim double / VARIANT\)
  Weighting factor for approximation precision.
  Default Value : 1.0
  Suggested values : Weight1 \(\in\) \{0.0, 0.5, 1.0\}
  Typical range of values : \(0.0 \leq \text{Weight1} \leq 0.0(\text{lin})\)
  Minimum Increment : 0.1
  Recommended Increment : 0.5

- Weight2 (input control) ............................................................... real  \(\sim double / VARIANT\)
  Weighting factor for large segments.
  Default Value : 1.0
  Suggested values : Weight2 \(\in\) \{0.0, 0.5, 1.0\}
  Typical range of values : \(0.0 \leq \text{Weight2} \leq 0.0(\text{lin})\)
  Minimum Increment : 0.1
  Recommended Increment : 0.5

- Weight3 (input control) ............................................................... real  \(\sim double / VARIANT\)
  Weighting factor for small segments.
  Default Value : 1.0
  Suggested values : Weight3 \(\in\) \{0.0, 0.5, 1.0\}
  Typical range of values : \(0.0 \leq \text{Weight3} \leq 0.0(\text{lin})\)
  Minimum Increment : 0.1
  Recommended Increment : 0.5

- ArcCenterRow (output control) ................................................... arc.center.y  \(\sim\) VARIANT(integer)
  Row of the center of an arc.

- ArcCenterCol (output control) ................................................... arc.center.x  \(\sim\) VARIANT(integer)
  Column of the center of an arc.

- ArcAngle (output control) ......................................................... arc.angle.rad  \(\sim\) VARIANT(real)
  Angle of an arc.

- ArcBeginRow (output control) .................................................... arc.begin.y  \(\sim\) VARIANT(integer)
  Row of the starting point of an arc.

- ArcBeginCol (output control) ..................................................... arc.begin.x  \(\sim\) VARIANT(integer)
  Column of the starting point of an arc.

- LineBeginRow (output control) ................................................... line.begin.y  \(\sim\) VARIANT(integer)
  Row of the starting point of a line segment.

- LineBeginCol (output control) ................................................... line.begin.x  \(\sim\) VARIANT(integer)
  Column of the starting point of a line segment.

- LineEndRow (output control) ...................................................... line.end.y  \(\sim\) VARIANT(integer)
  Row of the ending point of a line segment.

- LineEndCol (output control) ...................................................... line.end.x  \(\sim\) VARIANT(integer)
  Column of the ending point of a line segment.

- Order (output control) .............................................................. integer  \(\sim\) VARIANT(integer)
  Sequence of line (value 0) and arc segments (value 1).

The operator \texttt{ApproxChain} returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

HALCON 6.1.4
Approximate a contour by arcs and lines.

The contour of a curve is approximated by a sequence of lines and arcs. The result of the procedure is returned separately as arcs and lines. If one is interested in the sequence of the segments the individual resulting elements can be read successively from the resulting tuples. The sequence can be taken from the return parameter order (0: next element is next line segment, 1: next element is next arc segment).

The operator \texttt{ApproxChainSimple} behaves similarly as \texttt{ApproxChain} except that in the case of \texttt{ApproxChainSimple} the missing parameters are internally allocated as follows: MinWidthCoord = 1.0, MaxWidthCoord = 3.0, ThreshStart = 0.5, ThreshEnd = 0.9, ThreshStep = 0.3, MinWidthSmooth = 1.0, MaxWidthSmooth = 3.0, MinWidthCurve = 2, MaxWidthCurve = 9, Weight1 = 1.0, Weight2 = 1.0, Weight3 = 1.0.

**Parameter**

- **Row** (input control) ........................ point.y $\sim$ long / VARIANT
  Row of the contour.
  - **Default Value:** 32

- **Column** (input control) ........................ point.x $\sim$ long / VARIANT
  Column of the contour.
  - **Default Value:** 32

- **ArcCenterRow** (output control) ...................... arc.center.y $\sim$ VARIANT (integer)
  Row of the center of an arc.

- **ArcCenterCol** (output control) ...................... arc.center.x $\sim$ VARIANT (integer)
  Column of the center of an arc.

- **ArcAngle** (output control) ......................... arc.angle.rad $\sim$ VARIANT (real)
  Angle of an arc.

- **ArcBeginRow** (output control) ...................... arc.begin.y $\sim$ VARIANT (integer)
  Row of the starting point of an arc.

- **ArcBeginCol** (output control) ...................... arc.begin.x $\sim$ VARIANT (integer)
  Column of the starting point of an arc.

- **LineBeginRow** (output control) ...................... line.begin.y $\sim$ VARIANT (integer)
  Row of the starting point of a line segment.
6.2 Features

The operator \texttt{ApproxChainSimple} returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

\textbf{Parallelization Information}

\texttt{ApproxChainSimple} is \textit{reentrant} and processed without parallelization.

\textbf{Possible Predecessors}

\texttt{SobelAmp, EdgesImage, GetRegionContour, Threshold, HysteresisThreshold}

\textbf{Possible Successors}

\texttt{SetLineWidth, DispArc, DispLine}

\textbf{See also}

\texttt{GetRegionChain, SmallestCircle, DispCircle, DispLine}

\textbf{Alternatives}

\texttt{GetRegionPolygon, ApproxChain}

\textbf{Module}

\textbf{Region processing}

\section*{6.2 Features}

\begin{verbatim}

\end{verbatim}

Calculate the orientation of lines.

The operator \texttt{LineOrientation} returns the orientation ($-\pi/2 \leq \Phi \leq \pi/2$) of the given lines. If more than one line is to be treated the line and column indices can be passed as tuples. In this case \texttt{\Phi} is, of course, also a tuple and contains the corresponding orientations.

The procedure is typically applied to model lines in order to select parallel image lines, which were found, e.g., by \texttt{DetectEdgeSegments}, via the operator \texttt{SelectLines}.

\begin{verbatim}
RowBegin (input control) ......................... line.begin.y(-array) \rightsquigarrow\texttt{VARIANT} ( integer, real ) Row coordinates of the starting points of the input lines.
ColBegin (input control) .......................... line.begin.x(-array) \rightsquigarrow\texttt{VARIANT} ( integer, real ) Column coordinates of the starting points of the input lines.
RowEnd (input control) ............................ line.end.y(-array) \rightsquigarrow\texttt{VARIANT} ( integer, real ) Row coordinates of the ending points of the input lines.
ColEnd (input control) ............................. line.end.x(-array) \rightsquigarrow\texttt{VARIANT} ( integer, real ) Column coordinates of the ending points of the input lines.
Phi (output control) ............................... angle.rad(-array) \rightsquigarrow\texttt{VARIANT} ( real ) Orientation of the input lines.
\end{verbatim}

\texttt{LineOrientation} always returns the value TRUE.
Parallelization Information

LineOrientation is reentrant and processed without parallelization.

Possible Predecessors

SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines

Possible Successors

SetLineWidth, DispLine

See also

LinePosition, SelectLines, PartitionLines, DetectEdgeSegments

Alternatives

LinePosition, SelectLines, PartitionLines

Module

Region processing

Calculate the center of gravity, length, and orientation of a line.

The operator LinePosition returns the center (RowCenter, ColCenter), the (Euclidean) length (Length) and the orientation (−π/2 < Phi ≤ π/2) of the given lines. If more than one line is to be treated the line and column indices can be passed as tuples. In this case the output parameters, of course, are also tuples.

The routine is applied, for example, to model lines in order to determine search regions for the edge detection (DetectEdgeSegments).

Parameter

- **RowBegin** (input control) .................. line.begin.y(-array) ~ VARIANT (integer) Row coordinates of the starting points of the input lines.
- **ColBegin** (input control) .................. line.begin.x(-array) ~ VARIANT (integer) Column coordinates of the starting points of the input lines.
- **RowEnd** (input control) .................. line.end.y(-array) ~ VARIANT (integer) Row coordinates of the ending points of the input lines.
- **ColEnd** (input control) .................. line.end.x(-array) ~ VARIANT (integer) Column coordinates of the ending points of the input lines.
- **RowCenter** (output control) .............. point.y(-array) ~ VARIANT (real) Row coordinates of the centers of gravity of the input lines.
- **ColCenter** (output control) .............. point.x(-array) ~ VARIANT (real) Column coordinates of the centers of gravity of the input lines.
- **Length** (output control) .................. real(-array) ~ VARIANT (real) Euclidean length of the input lines.
- **Phi** (output control) ...................... angle.rad(-array) ~ VARIANT (real) Orientation of the input lines.

Result

LinePosition always returns the value TRUE.
6.2. FEATURES

Possible Successors

SetLineWidth, DispLine

See also

LineOrientation, SelectLines, PartitionLines, DetectEdgeSegments

Alternatives

LineOrientation, SelectLines, PartitionLines

Module

Region processing

Partition lines according to various criteria.

The operator PartitionLines divides lines into two sets according to various criteria. For each input line the indicated features (Feature) are calculated. If each (Operation = 'and') or at least one (Operation = 'or') of the calculated features is within the given limits (Min, Max) the line is transferred into the first set (parameters RowBeginOut to ColEndOut), otherwise into the second set (parameters FailRowBOut to FailColEOut).

Condition: \[ \text{Min}_i \leq \text{Feature}_i(\text{Line}) \leq \text{Max}_i \]

Possible values for Feature:

'length' (Euclidean) length of the line
'row' Line index of the center
'column' Column index of the center
'phi' Orientation of the line \((\pi \leq \phi \leq 2\pi)\)

Attention

If only one feature is used the value of Operation is meaningless. Several features are processed according to the sequence in which they are passed.

Parameter

- **RowBeginIn** (input control) ............................................... line.begin.y \sim \text{VARIANT} (integer)
  Row coordinates of the starting points of the input lines.
- **ColBeginIn** (input control) ............................................... line.begin.x \sim \text{VARIANT} (integer)
  Column coordinates of the starting points of the input lines.
- **RowEndIn** (input control) ............................................... line.end.y \sim \text{VARIANT} (integer)
  Row coordinates of the ending points of the input lines.
- **ColEndIn** (input control) ............................................... line.end.x \sim \text{VARIANT} (integer)
  Column coordinates of the ending points of the input lines.
- **Feature** (input control) ............................................... \text{string(-array)} \sim \text{VARIANT} (string)
  Features to be used for selection.

List of values: Feature \in \{ 'length', 'row', 'column', 'phi' \}
The operator `PartitionLines` returns the value `TRUE` if the parameter values are correct. Otherwise an exception is raised.

---

### Parallelization Information

`PartitionLines` is reentrant and processed without parallelization.

### Possible Predecessors

SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines

### Possible Successors

SetLineWidth, DispLine

### See also

SelectLines, SelectLinesLongest, DetectEdgeSegments, SelectShape

### Alternatives

LineOrientation, LinePosition, SelectLines, SelectLinesLongest

### Module

Region processing

---

```
[out] VARIANT RowBeginOut HMiscX.SelectLines ([in] VARIANT RowBeginIn,
[in] VARIANT ColBeginIn, [in] VARIANT RowEndIn, [in] VARIANT ColEndIn,
[in] VARIANT Max, [out] VARIANT ColBeginOut, [out] VARIANT RowEndOut,
[out] VARIANT ColEndOut )

void HOperatorSetX.SelectLines ([in] VARIANT RowBeginIn,
[in] VARIANT ColBeginIn, [in] VARIANT RowEndIn, [in] VARIANT ColEndIn,
[in] VARIANT Max, [out] VARIANT RowBeginOut, [out] VARIANT ColBeginOut,
[out] VARIANT RowEndOut, [out] VARIANT ColEndOut )
```

Select lines according to various criteria.

HALCON/COM Reference Manual, 2005-2-1
The operator \texttt{SelectLines} chooses lines according to various criteria. For every input line the indicated features (\texttt{Feature}) are calculated. If each (\texttt{Operation} = \texttt{’and’}) or at least one (\texttt{Operation} = \texttt{’or’}) of the calculated features is within the given limits (\texttt{Min}, \texttt{Max}) the line is transferred into the output.

\textbf{Condition}: \quad \texttt{Min}_i \leq \texttt{Feature}_i(\texttt{Line}) \leq \texttt{Max}_i

Possible values for \texttt{Feature}:

- \texttt{’length’} (Euclidean) length of the line
- \texttt{’row’} Line index of the center
- \texttt{’column’} Column index of the center
- \texttt{’phi’} Orientation of the line \((\pi \leq \phi \leq \pi)\)

\textbf{Attention}\quad If only one feature is used the value of \texttt{Operation} is meaningless. Several features are processed according to the sequence in which they are passed.

\textbf{Parameter}

- \texttt{RowBeginIn} (input control) \:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\:\\ Result

The operator \texttt{SelectLines} returns the value TRUE if the parameter values are correct. Otherwise an exception is raised.

\textbf{Parallelization Information}

\texttt{SelectLines} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Possible Predecessors}

\texttt{SobelAmp, EdgesImage, Threshold, HysteresisThreshold, SplitSkeletonRegion, SplitSkeletonLines}

HALCON 6.1.4
Select the longest input lines.

The operator `SelectLinesLongest` selects the `Num` longest input lines from the input lines described by the tuples `RowBeginIn`, `ColBeginIn`, `RowEndIn` and `ColEndIn`.

Parameter

- **RowBeginIn** (input control) ................... line.begin.y ~ VARIANT (integer)
  Row coordinates of the starting points of the input lines.
- **ColBeginIn** (input control) ................... line.begin.x ~ VARIANT (integer)
  Column coordinates of the starting points of the input lines.
- **RowEndIn** (input control) ................... line.end.y ~ VARIANT (integer)
  Row coordinates of the ending points of the input lines.
- **ColEndIn** (input control) ................... line.end.x ~ VARIANT (integer)
  Column coordinates of the ending points of the input lines.
- **Num** (input control) .................... integer ~ long / VARIANT
  (Maximum) desired number of output lines.
  Default Value: 10

Result

The operator `SelectLinesLongest` returns the value TRUE if the parameter values are correct. Otherwise an exception is raised.

Parallelization Information

`SelectLinesLongest` is reentrant and processed without parallelization.
See also

SelectLines, PartitionLines, DetectEdgeSegments, SelectShape

Alternatives

LineOrientation, LinePosition, SelectLines, PartitionLines

Module

Region processing
Chapter 7

Matching

7.1 Gray-Value-Based

```
void HImageX.AdaptTemplate ([in] HTemplateX TemplateID )
void HTemplateX.AdaptTemplate ([in] HImageX Image )
void HOperatorSetX.AdaptTemplate ([in] IOBJECT X Image, [in] VARIANT TemplateID )
```

Adapting a template to the size of an image.

The operator AdaptTemplate serves to adapt a template which has been created by CreateTemplate to the size of an image. The operator AdaptTemplate can be called before the template is used with images of another size, or if the image used to create the template had another size. If it is not called explicitly it will be called internally each time another image size is used. The contents of the image is hereby irrelevant; only the width of Image will be considered.

Parameter

- Image (input iconic) .......... image-array .......... HImageX / IOBJECT X (byte)
  Image which determines the size of the later matching.

- TemplateID (input control) ................. template .......... HTemplateX / VARIANT
  Template number.

Result

If the parameter values are correct, the operator AdaptTemplate returns the value TRUE. If necessary, an exception handling is raised.

Parallelization Information

AdaptTemplate is processed under mutual exclusion against itself and without parallelization.

Possible Predecessors

CreateTemplate, CreateTemplateRot, ReadTemplate

Possible Successors

SetReferenceTemplate, BestMatch, FastMatch, FastMatchMg, SetOffsetTemplate, BestMatchMg, BestMatchPreMg, BestMatchRotMg, BestMatchRotMg

Module

Template matching
Searching the best matching of a template and an image.

The operator `BestMatch` performs a matching of the template of `TemplateID` and `Image`. Hereby the template will be moved over the points of `Image` so that the template will lie always inside `Image`. `BestMatch` works similar to `FastMatch`, with the exception, that each time a better match is found the value of `MaxError` is internally updated to a lower value to reduce runtime.

With regard to the parameter `SubPixel`, the position will be indicated by subpixel accuracy. The matching criterion (“displaced frame difference”) is defined as follows:

\[
\text{error}[\text{row}, \text{col}] = \frac{\sum_{u,v}\mid \text{Image}[\text{row} - u, \text{col} - v] - \text{TemplateID}[u, v] \mid}{\text{area}(\text{TemplateID})}
\]

The runtime of the operator depends on the size of the domain of `Image`. Therefore it is important to restrict the domain as far as possible, i.e. to apply the operator only in a very confined “region of interest”. The parameter `MaxError` determines the maximal error which the searched position is allowed to have at most. The lower this value is, the faster the operator runs.

`Row` and `Column` return the position of the best match, whereby `Error` indicates the average difference of the grayvalues. If no position with an error below `MaxError` was found the position (0, 0) and a matching result of 255 for `Error` are returned. In this case `MaxError` has to be set larger.

The maximum error of the position (without noise) is 0.1 pixel. The average error is 0.03 pixel.

---

**Parameter**

- **Image** (input iconic) .......................... image(-array) \( \sim \) `HImageX / IObjectX (byte)`
  Input image inside of which the pattern has to be found.

- **TemplateID** (input control) .................. template \( \sim \) `HTemplateX / VARIANT`
  Template number.

- **MaxError** (input control) .................. real \( \sim \) `double / VARIANT`
  Maximum average difference of the grayvalues.
  **Default Value**: 20
  **Suggested values**: `MaxError` \( \in \) \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 0, 70\}
  **Typical range of values**: \(0 \leq \text{MaxError} \leq 0\)
  **Minimum Increment**: 1
  **Recommended Increment**: 3

- **SubPixel** (input control) .................. string \( \sim \) `String / VARIANT`
  Subpixel accuracy in case of ‘true’.
  **Default Value**: ‘false’
  **List of values**: `SubPixel` \( \in \) \{'true’, ‘false’\}

- **Row** (output control) .................. point.y(-array) \( \sim \) `VARIANT` (real)
  Row position of the best match.

- **Column** (output control) .................. point.x(-array) \( \sim \) `VARIANT` (real)
  Column position of the best match.

- **Error** (output control) .................. real(-array) \( \sim \) `VARIANT` (real)
  Average divergence of the grayvalues of the best match.

---

**Result**

If the parameter values are correct, the operator `BestMatch` returns the value `TRUE`. If the input is empty (no input images are available) the behaviour can be set via `SetSystem(‘noObjectResult’,<Result>)`. If necessary, an exception handling is raised.

HALCON/COM Reference Manual, 2005-2-1
7.1. GRAY-VALUE-BASED

--- Parallelization Information ---

**BestMatch** is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessors ---

CreateTemplate, ReadTemplate, SetOffsetTemplate, SetReferenceTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain

--- Alternatives ---

FastMatch, FastMatchMg, BestMatchMg, BestMatchPreMg, BestMatchRot, BestMatchRotMg, ExhaustiveMatch, ExhaustiveMatchMg

--- Module ---

Template matching

```
[out] double Row HImageX.BestMatchMg ([in] HTemplateX TemplateID, 
```

```
[out] double Row HTemplateX.BestMatchMg ([in] HImageX Image, 
```

```
void HOperatorSetX.BestMatchMg ([in] IObjectX Image, 
[in] VARIANT TemplateID, [in] VARIANT MaxError, [in] VARIANT SubPixel, 
[in] VARIANT NumLevels, [in] VARIANT WhichLevels, [out] VARIANT Row, 
[out] VARIANT Column, [out] VARIANT Error )
```

Searching the best grayvalue matches in a pyramid.

**BestMatchMg** applies gray value matching using an image pyramid. **BestMatchMg** works analogously to **BestMatch**, but it is faster due to the use of a pyramid. Input is an image with an optionally reduced domain. The parameter **MaxError** specifies the maximum error for template matching. Using smaller values results in a reduced runtime but it is possible that the pattern might be missed. The value of **MaxError** has to set larger compared with **BestMatch**, because the error is at higher levels of the pyramid often increased.

**SubPixel** specifies if the result is calculated with sub pixel accuracy or not. A value of 1 for **SubPixel** results in an operator similar to **BestMatch**, i.e. only the original gray values are used. For values larger than 1, the algorithm starts at the lowest resolution and searches for a position with the lowest matching error. At the next higher resolution this position is refined. This is continued up to the maximum resolution (WhichLevels = ’all’). As an alternative Method the mode **WhichLevels** with value ’original’ can be used. In this case not only the position with the lowest error but all points below **MaxError** are analysed further on in the next higher resolution. This method is slower but it is more stable and the possibility to miss the correct position is very low. In this case it is often possible to start with a lower resolution (higher level in Pyramid, i.e. larger value for **NumLevels**) which leads to a reduced runtime. Besides the values ’all’ and ’original’ for **WhichLevels** you can specify the pyramid level explicitly where to switch between a “match all” and the ”best match”. Here 0 corresponds to ’original’ and **NumLevels** - 1 is equivalent to ’all’. A value in between is in most cases a good compromise between speed and a stable detection. A larger value for **WhichLevels** results in a reduced runtime, a smaller value results in a more stable detection. The value of **NumLevels** has to equal or smaller than the value used to create the template.

The position of the found matching position is returned in **Row** and **Column**. The corresponding error is given in **Error**. If no point below **MaxError** is found a value of 255 for **Error** and 0 for **Row** and **Column** is returned. If the desired object is missed (no object found or wrong position) you have to set **MaxError** higher or **WhichLevels** lower. Check also if the illumination has changed (see **SetOffsetTemplate**).

The maximum error of the position (without noise) is 0.1 pixel. The average error is 0.03 pixel.

--- Parameter ---

- **Image** (input iconic) ......................... image(-array) ~ HImageX / IObjectX (byte )
  Input image inside of which the pattern has to be found.
- **TemplateID** (input control) ................... template ~ HTemplateX / VARIANT
  Template number.
Searching the best grayvalue matches in a pre generated pyramid.

BestMatchPreMg applies gray value matching using an image pyramid. BestMatchPreMg works analogously to BestMatchMg, but it makes use of pre calculated pyramid which has to be generated beforehand using
7.1. GRAY-VALUE-BASED

GenGaussPyramid. This reduces runtime if more than one match has to be done or the pyramid has been used otherwise. The pyramid has to be generated using the zooming factor 0.5 and the mode 'constant'.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImagePyramid</td>
<td>HImageX / HObjectX (byte)</td>
<td>Image pyramid inside of which the pattern has to be found.</td>
</tr>
<tr>
<td>TemplateID</td>
<td>HTemplateX / VARIANT</td>
<td>Template number.</td>
</tr>
<tr>
<td>MaxError</td>
<td>real / VARIANT</td>
<td>Maximal average difference of the grayvalues.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Suggested values: MaxError ( \in {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70} )</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Typical range of values: 0 ( \leq ) MaxError ( \leq ) 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximum Increment: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recommended Increment: 3</td>
<td></td>
</tr>
<tr>
<td>SubPixel</td>
<td>String / VARIANT</td>
<td>Exactness in subpixels in case of 'true'.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 'false'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>List of values: SubPixel ( \in {\text{true}', 'false'} )</td>
<td></td>
</tr>
<tr>
<td>NumLevels</td>
<td>integer / VARIANT</td>
<td>Number of the used resolution levels.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>List of values: NumLevels ( \in {1, 2, 3, 4, 5, 6} )</td>
<td></td>
</tr>
<tr>
<td>WhichLevels</td>
<td>integer / VARIANT (string, integer)</td>
<td>Resolution level up to which the method “best match” is used.</td>
</tr>
<tr>
<td></td>
<td>Default Value: 'original'</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Suggested values: WhichLevels ( \in {'all', 'original', 0, 1, 2, 3, 4, 5, 6} )</td>
<td></td>
</tr>
<tr>
<td>Row</td>
<td>point.y / VARIANT</td>
<td>Row position of the best match.</td>
</tr>
<tr>
<td>Column</td>
<td>point.x / VARIANT</td>
<td>Column position of the best match.</td>
</tr>
<tr>
<td>Error</td>
<td>real / VARIANT</td>
<td>Average divergence of the grayvalues in the best match.</td>
</tr>
</tbody>
</table>

If the parameter values are correct, the operator BestMatchPreMg returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

BestMatchPreMg is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

GenGaussPyramid, CreateTemplate, ReadTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain, SetReferenceTemplate

Alternatives

FastMatch, FastMatchMg, ExhaustiveMatch, ExhaustiveMatchMg

Template matching
Searching the best matching of a template and an image with rotation.

The operator `BestMatchRot` performs a matching of the template of `TemplateID` and `Image`. It works similar to `BestMatch` with the extension that the pattern can be rotated. The parameters `AngleStart` and `AngleExtend` define the maximum rotation of the pattern: `AngleStart` specifies the maximum counter clockwise rotation and `AngleExtend` the maximum clockwise rotation relative to this angle. Both values have to be smaller or equal to the values used for the creation of the pattern (see `CreateTemplateRot`). In addition to `BestMatch`, `BestMatchRot` returns the rotation angle of the pattern in `Angle` (radian). The accuracy of this parameter depends on the parameter `AngleStep` of `CreateTemplateRot`. In the case of `SubPixel = 'true'` the position and the angle are calculated with “sub pixel” accuracy.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image</code></td>
<td>Input image inside of which the pattern has to be found.</td>
</tr>
<tr>
<td><code>TemplateID</code></td>
<td>Template number.</td>
</tr>
<tr>
<td><code>AngleStart</code></td>
<td>Smallest Rotation auf the pattern. Default Value: 0.39, Suggested values: <code>AngleStart</code> ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}</td>
</tr>
<tr>
<td><code>AngleExtend</code></td>
<td>Maximum positive Extention of <code>AngleStart</code>. Default Value: 0.79, Suggested values: <code>AngleExtend</code> ∈ {6.28, 3.14, 1.57, 0.79, 0.39}, Restriction: <code>(AngleExtend &gt; 0)</code></td>
</tr>
<tr>
<td><code>MaxError</code></td>
<td>Maximum average difference of the grayvalues. Default Value: 30, Suggested values: <code>MaxError</code> ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}, Typical range of values: <code>0 ≤ MaxError ≤ 0</code>, Minimum Increment: 1, Recommended Increment: 3</td>
</tr>
<tr>
<td><code>SubPixel</code></td>
<td>Subpixel accuracy in case of ‘true’. Default Value: ’false’, List of values: <code>SubPixel</code> ∈ {’true’, ’false’}</td>
</tr>
<tr>
<td><code>Row</code></td>
<td>Row position of the best match.</td>
</tr>
<tr>
<td><code>Column</code></td>
<td>Column position of the best match.</td>
</tr>
<tr>
<td><code>Angle</code></td>
<td>Rotation angle of pattern.</td>
</tr>
<tr>
<td><code>Error</code></td>
<td>Average divergence of the grayvalues of the best match.</td>
</tr>
</tbody>
</table>
7.1. GRAY-VALUE-BASED

Result

If the parameter values are correct, the operator `BestMatchRot` returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via `SetSystem('noObjectResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

`BestMatchRot` is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

`CreateTemplateRot`, `ReadTemplate`, `SetOffsetTemplate`, `SetReferenceTemplate`, `AdaptTemplate`, `DrawRegion`, `DrawRectangle1`, `ReduceDomain`

See also

`BestMatch`, `BestMatchMg`

Alternatives

`BestMatchRotMg`

Module

Template matching

```halcon
```

Searching the best matching of a template and a pyramid with rotation.

The operator `BestMatchRotMg` performs a matching of the template of `TemplateID` and `Image`. It works similar to `BestMatchMg` with the extension that the pattern can be rotated analogously to `BestMatchRot`. The parameters `AngleStart` and `AngleExtend` define the maximum rotation of the pattern: `AngleStart` specifies the maximum counter clockwise rotation and `AngleExtend` the maximum clockwise rotation relative to this angle. Both values have to be smaller or equal to the values used for the creation of the pattern (see `CreateTemplateRot`). In addition to `BestMatchMg` `BestMatchRotMg` returns the rotation angle of the pattern in `Angle` (radian).

The value of `MaxError` has to be set larger compared with `BestMatchRot`, because the error is at higher levels of the pyramid often increased.

In the case of `SubPixel` = 'true' the position and the angle are calculated with “sub pixel” accuracy.

The value of `NumLevels` has to equal or smaller than the value used to create the template.

Parameter

- **Image** (input iconic) ................. image(-array) \sim HImageX / IHObjectX (byte)
  - Input image inside of which the pattern has to be found.
- **TemplateID** (input control) ........ Template \sim HTemplateX / VARIANT
  - Template number.
- **AngleStart** (input control) ........Angle.rad \sim double / VARIANT
  - Smallest Rotation auf the pattern.
  - Default Value : -0.39
  - Suggested values : `AngleStart \in \{-3.14, -1.57, -0.79, -0.39, -0.20, 0.0\}`
\textbf{AngleExtend} (input control) \(\ldots\) \textit{angle.rad} \(\sim\) \textit{double / VARIANT}

Maximum positive extension of \textit{AngleStart.}

Default Value : 0.79

Suggested values : \textit{AngleExtend} \(\in\) \{6.28, 3.14, 1.57, 0.79, 0.39\}

Restriction : \((\text{AngleExtend} > 0)\)

\textbf{MaxError} (input control) \(\ldots\) \textit{real} \(\sim\) \textit{double / VARIANT}

Maximum average difference of the grayvalues.

Default Value : 40

Suggested values : \textit{MaxError} \(\in\) \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70\}

Typical range of values : \(0 \leq \text{MaxError} \leq 0\)

Minimum Increment : 1

Recommended Increment : 1

\textbf{SubPixel} (input control) \(\ldots\) \textit{string} \(\sim\) \textit{String / VARIANT}

Subpixel accuracy in case of \textit{true}.

Default Value : \textit{false}

List of values : \textit{SubPixel} \(\in\) \{'true', 'false'\}

\textbf{NumLevels} (input control) \(\ldots\) \textit{integer} \(\sim\) \textit{long / VARIANT}

Number of the used resolution levels.

Default Value : 3

List of values : \textit{NumLevels} \(\in\) \{1, 2, 3, 4, 5, 6\}

\textbf{Row} (output control) \(\ldots\) \textit{point.y(-array)} \(\sim\) \textit{VARIANT ( real )}

Row position of the best match.

\textbf{Column} (output control) \(\ldots\) \textit{point.x(-array)} \(\sim\) \textit{VARIANT ( real )}

Column position of the best match.

\textbf{Angle} (output control) \(\ldots\) \textit{angle.rad(-array)} \(\sim\) \textit{VARIANT ( real )}

Rotation angle of pattern.

\textbf{Error} (output control) \(\ldots\) \textit{real(-array)} \(\sim\) \textit{VARIANT ( real )}

Average divergence of the grayvalues of the best match.

---

If the parameter values are correct, the operator \textit{BestMatchRotMg} returns the value \textit{TRUE}. If the input is empty (no input images are available) the behaviour can be set via \textit{SetSystem(‘noObjectResult’,<Result>).} If necessary, an exception handling is raised.

---

\textbf{Parallelization Information}

\textbf{BestMatchRotMg} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

---

\textbf{Possible Predecessors}

\textit{CreateTemplateRot, SetReferenceTemplate, SetOffsetTemplate, AdaptTemplate}, \textit{DrawRegion, DrawRectangle1, ReduceDomain}

---

\textbf{See also}

\textit{FastMatch}

---

\textbf{Alternatives}

\textit{BestMatchRot, BestMatchMg}

---

\textbf{Module}

---

\begin{verbatim}
void HMiscX.ClearAllTemplates ()
void HOperatorSetX.ClearAllTemplates ()
\end{verbatim}

\texttt{Deallocation of the memory of all templates.}

The operator \textit{ClearTemplate} deallocates the memory of all template that were created by \textit{CreateTemplate} or \textit{CreateTemplateRot}. After calling \textit{ClearAllTemplates}, no template can be used any longer.

\textbf{Result}

\textit{ClearAllTemplates} always returns \textit{TRUE}.  

---

HALCON/COM Reference Manual, 2005-2-1
7.1. GRAY-VALUE-BASED

Parallelization Information

ClearAllTemplates is processed completely exclusively without parallelization.

Possible Predecessors

CreateTemplate, CreateTemplateRot, ReadTemplate, WriteTemplate

Alternatives

ClearTemplate

Module

Template matching

void HOperatorSetX.ClearTemplate ([in] VARIANT TemplateID )

Deallocation of the memory of a template.

The operator ClearTemplate deallocates the memory of a template which has been created by CreateTemplate or CreateTemplateRot. After execution of the operator ClearTemplate the template can no longer be used. The value of TemplateID is not valid. However, the number can be used again by further calls of CreateTemplate or CreateTemplateRot.

Parameter

> TemplateID (input control) ......................................... template \( \sim \) HTemplateX / VARIANT Template number.

Result

If the number of the template is valid, the operator ClearTemplate returns the value TRUE. If necessary an exception handling will be raised.

Parallelization Information

ClearTemplate is processed completely exclusively without parallelization.

Possible Predecessors

CreateTemplate, CreateTemplateRot, ReadTemplate, WriteTemplate

See also

ClearAllTemplates

Module

Template matching

[out] HTemplateX TemplateID HImageX.CreateTemplate
(([in] long FirstError, [in] long NumLevel, [in] String Optimize,
[in] String GrayValues )

void HTemplateX.CreateTemplate ([in] HImageX Template,
[in] long FirstError, [in] long NumLevel, [in] String Optimize,
[in] String GrayValues )

void HOperatorSetX.CreateTemplate ([in] IHObjectX Template,
in] VARIANT GrayValues, [out] VARIANT TemplateID )

Preparing a pattern for template matching.

The operator CreateTemplate preprocesses a pattern (Template), which is passed as an image, for the template matching. After the transformation, a number (TemplateID) is assigned to the template for being used in the further process. The shape and the size of Template can be chosen arbitrarily. You have to be aware, that the matching is only applied to that part of an image where Template fits completely into the image.

The template has to be chosen such that it contains no pixels of the (changing) background. Here you can make use of the arbitrary shape of a template which is not restricted to a rectangle. To create a template you can use segmentation operators like Threshold. In the case of sub pixel accurate matching Template has in addition
to be one pixel smaller than the pattern (i.e. one pixel border to the changing background). This can be done e.g.
by applying the operator \texttt{ErosionCircle}.

The parameter \texttt{NumLevel} specifies the number of pyramid levels (\texttt{NumLevel} = 1 means only original gray
values) which can be used for matching. The number of levels used later for matching will be below or equal this
value. If the pattern becomes to small due to zooming, the maximum number of pyramid levels is automatically
reduced (without error message).

The parameter \texttt{GrayValues} defines, wheter the original gray values ('original', 'normalized) or the edge amplitude
('gradient', 'sobel') is used. With 'original' the sum of the differences is used as feature which is very stable
and fast if there is no change in illumination. 'normalized' is used if the illumination changes. The method is a bit
slowier and not quite as stable. If there is no change in illumination the mode 'original' should be used. The edge
amplitude is another method to be invariant to changes in illumination. The disadvantage is the increased execution
time and the higher sensitivity to changes in the shape of the pattern. The mode 'gradient' is slighy faster but more
sensitive to noise.

The maximum error for matching has typically to be chosen higher when using the edge amplitude. The mode
chosen by \texttt{GrayValues} leads automatically to calling the appropriate filter during matching — if necessary.

As an alternative to the gradient approach the operator \texttt{SetOffsetTemplate} can be used, if the change in
illumination is known.

The parameter \texttt{Optimize} specifies if the pattern has to optimized for runtime. This optimization results in a
longer time to create the template but reduces the time for matching. In addition the optimization leads to a more
stable matching, i.e., the possibility to miss good matches is reduced. The optimization process selects the most
stable and significant gray values to be tested first during the matching process. Using this technique a wrong
match can be eliminated very early.

The reference position for the template is its center of gravity. I.e. if you apply the template to the orig-
inal image the center of gravity is returned. This default reference can be adapted using the operator
\texttt{SetReferenceTemplate}.

In sub pixel mode a special position correction is calculated which is added after each matching: The template is
applied to the original image and the difference between the found position and the center of gravity is used as a
correction vector. This is important for patterns in a textured context or for asymetric pattern. For most templates
this correction vector is near null.

If the pattern is no longer used, it has to be freed by the operator \texttt{ClearTemplate} in order to deallocate the
memory.

Before the use of the template, which is stored independently of the image size, it can be adapted explicitly to the
size of a definite image size by using \texttt{AdaptTemplate}.

\begin{tabular}{p{0.05\textwidth}p{0.85\textwidth}}
\hline
\textbf{Parameter} & \textbf{Type} \\
\hline
\texttt{Template} (input iconic) & image \sim HImageX / IHObjectX ( byte ) \\
\texttt{FirstError} (input control) & integer \sim long / VARIANT \\
\texttt{NumLevel} (input control) & integer \sim long / VARIANT \\
\texttt{Optimize} (input control) & string \sim String / VARIANT \\
\texttt{GrayValues} (input control) & string \sim String / VARIANT \\
\texttt{TemplateID} (output control) & template \sim HTemplateX / VARIANT \\
\hline
\end{tabular}
If the parameters are valid, the operator \texttt{CreateTemplate} returns the value TRUE. If necessary an exception handling will be raised.

\textbf{Parallelization Information}

\texttt{CreateTemplate} is processed completely exclusively without parallelization.

\textbf{Possible Predecessors}

\texttt{DrawRegion, ReduceDomain, Threshold}

\textbf{Possible Successors}

\texttt{AdaptTemplate, SetReferenceTemplate, ClearTemplate, WriteTemplate, SetOffsetTemplate, BestMatch, BestMatchMg, FastMatch, FastMatchMg}

\textbf{Alternatives}

\texttt{CreateTemplateRot, ReadTemplate}

\textbf{Module}

Template matching

\package{HTemplateX} \function{CreateTemplateRot} ([\in] \long \texttt{NumLevel}, [\in] \double \texttt{AngleStart}, [\in] \double \texttt{AngleExtend}, [\in] \texttt{String \texttt{Optimize}}, [\in] \texttt{String \texttt{GrayValues}})

\package{HOperatorSetX} \function{CreateTemplateRot} ([\in] \texttt{HImageX Template}, [\in] \long \texttt{NumLevel}, [\in] \double \texttt{AngleStart}, [\in] \double \texttt{AngleExtend}, [\in] \double \texttt{AngleStep}, [\in] \texttt{String \texttt{Optimize}}, [\in] \texttt{String \texttt{GrayValues}})

Preparing a pattern for template matching with rotation.

The operator \texttt{CreateTemplateRot} preprocesses a pattern, which is passed as an image, for the template matching. An extension to \texttt{CreateTemplate} the matching can applied to rotated patterns. The parameters \texttt{AngleStart} and \texttt{AngleExtend} define the maximum rotation of the pattern: \texttt{AngleStart} specifies the maximum counter clockwise rotation and \texttt{AngleExtend} the maximum clockwise rotation relative to this angle. Therefore \texttt{AngleExtend} has to be smaller than $2\pi$. With the parameter \texttt{AngleStep} the maximum angle resolution (on the highest resolution level) can be specified.

You have to be aware, that all possible rotations are calculated beforehand to reduce runtime during matching. This leads to a higher execution time for \texttt{CreateTemplateRot} and high memory requirements for the template.

The amount of memory depends on the parameters \texttt{AngleExtend} and \texttt{AngleStep}. The number of pyramid levels can be neglected. If $A$ is the number of pixels of \texttt{Template}, the memory $M$ needed for the template in byte is about:

\[ M = \frac{A \times 12 \times \texttt{AngleExtend}}{\texttt{AngleStep}} \]

After the transformation, a number (\texttt{TemplateID}) is assigned to the template for being used in the further process.

A description of the other parameters can be found at the operator \texttt{CreateTemplate}.

\textbf{Attention}

You have to be aware, that depending on the resolution a large number of pre calculated patterns have to be created which might result in a large amount of memory needed.

\textbf{Parameter}

\begin{itemize}
  \item \textbf{Template} (input iconic) \hspace{1cm} \text{image $\sim$ \texttt{HImageX/IObjectX (byte)}}
  \item Input image whose domain will be processed for the pattern matching.
\end{itemize}
 CHAPTER 7. MATCHING

- **NumLevel** (input control) .................................................. integer  \(\sim long /\) VARIANT
  Maximal number of pyramid levels.
  Default Value : 4
  List of values : \(\text{NumLevel} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\)

- **AngleStart** (input control) .................................................. angle.rad  \(\sim double /\) VARIANT
  Smallest Rotation of the pattern.
  Default Value : -0.39
  Suggested values : \(\text{AngleStart} \in \{-3.14, -1.57, -0.79, -0.39, -0.20, 0.0\}\)

- **AngleExtend** (input control) .............................................. angle.rad  \(\sim double /\) VARIANT
  Maximum positive Extension of \(\text{AngleStart}\).
  Default Value : 0.79
  Suggested values : \(\text{AngleExtend} \in \{6.28, 3.14, 1.57, 0.79, 0.39\}\)
  Restriction : \((\text{AngleExtend} > 0)\)

- **AngleStep** (input control) ................................................. angle.rad  \(\sim double /\) VARIANT
  Step rate (angle precision) of matching.
  Default Value : 0.0982
  Suggested values : \(\text{AngleStep} \in \{0.3927, 0.1963, 0.0982, 0.0491, 0.0245\}\)
  Restriction : \((\text{AngleStep} > 0)\)

- **Optimize** (input control) ................................................... string  \(\sim String /\) VARIANT
  Kind of optimizing.
  Default Value : ‘sort’
  List of values : \(\text{Optimize} \in \{'none', 'sort'\}\)

- **GrayValues** (input control) .............................................. string  \(\sim String /\) VARIANT
  Kind of grayvalues.
  Default Value : ‘original’
  List of values : \(\text{GrayValues} \in \{'original', 'normalized', 'gradient', 'sobel'\}\)

- **TemplateID** (output control) ........................................... template  \(\sim HTemplateX /\) VARIANT
  Template number.

If the parameters are valid, the operator \texttt{CreateTemplateRot} returns the value TRUE. If necessary an exception handling will be raised.

**Parallelization Information**

\texttt{CreateTemplateRot} is processed completely exclusively without parallelization.

**Possible Predecessors**

\texttt{DrawRegion, ReduceDomain, Threshold}

**Possible Successors**

\texttt{BestMatchRot, BestMatchRotMg, AdaptTemplate, SetReferenceTemplate, ClearTemplate, SetOffsetTemplate, WriteTemplate}

**Alternatives**

\texttt{CreateTemplate}

**Module**

```
[out] HRegionX Matches HImageX.FastMatch ([in] HTemplateX TemplateID, [in] double MaxError )
```

Searching all good matches of a template and an image.

HALCON/COM Reference Manual, 2005-2-1
The operator **FastMatch** performs a matching of the template of **TemplateID** and **Image**. Hereby the template will be moved over the points of **Image** so that the template always lies completely inside of **Image**. The matching criterion ("displaced frame difference") is defined as follows:

$$
error[row, col] = \sum_{u,v} |Image[row - u, col - v] - TemplateID[u, v]| / area(TemplateID)
$$

The difference between **FastMatch** and **ExhaustiveMatch** is that the matching for one position is stopped if the error is too high. This leads to a reduced runtime but one might miss correct matches. The runtime of the operator depends mainly on the size of the domain of **Image**. Therefore it is important to restrict the domain as far as possible, i.e. to apply the operator only in a very confined "region of interest". The parameter **MaxError** determines the maximal error which the searched position is allowed to show. The lower this value is, the faster the operator runs.

All points which show a matching error smaller than **MaxError** will be returned in the output region **Matches**. This region can be used for further processing. For example by using a connection and **BestMatch** to find all the matching objects. If no point has a match error below **MaxError** the empty region (i.e no points) is returned.

---

**Parameter**

- **Image** (input iconic) ......................... image(-array) \sim HImageX / HObjectX (byte)
  Input image inside of which the pattern has to be found.
- **Matches** (output iconic) ....................... region(-array) \sim HRegionX / HUntypedObjectX
  All points whose error lies below a certain threshold.
- **TemplateID** (input control) .................... template \sim HTemplateX / VARIANT
  Template number.
- **MaxError** (input control) ..................... real \sim double / VARIANT
  Maximal average difference of the grayvalues.
  **Default Value**: 20
  **Suggested values**: MaxError \in \{0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30\}
  **Typical range of values** : 0 \leq MaxError \leq 0
  **Minimum Increment** : 1
  **Recommended Increment** : 1

---

**Result**

If the parameter values are correct, the operator **FastMatch** returns the value TRUE. If the input is empty (no input images are available) the behaviour can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

**FastMatch** is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

CreateTemplate, ReadTemplate, AdaptTemplate, DrawRegion, DrawRectangle1, ReduceDomain

---

**Possible Successors**

Connection, BestMatch

---

**Alternatives**

BestMatch, BestMatchMg, FastMatchMg, ExhaustiveMatch, ExhaustiveMatchMg

---

**Module**

Template matching
Searching all good grayvalue matches in a pyramid.

The operator \texttt{FastMatchMg} like the operator \texttt{FastMatch} performs a matching of the template of \texttt{TemplateID} and \texttt{Image}. In contrast to \texttt{FastMatch}, however, the search for good matches starts in scaled down images (pyramid). The number of levels of the pyramid will be determined by \texttt{NumLevel}. Hereby the value 1 indicates that no pyramid will be used. In this case the operator \texttt{FastMatchMg} is equivalent to the operator \texttt{FastMatch}. The value 2 triggers the search in the image with half the frame size. The search for all those points showing an error small enough in the scaled down image (error smaller than \texttt{MaxError}) will be refined at the corresponding positions in the original image (\texttt{Image}).

The runtime of matching depends on the parameter \texttt{MaxError}: the larger the value the longer is the processing time, because more points of the pattern have to be tested. If \texttt{MaxError} is to low the pattern will not be found.

The value has therefore to be optimized for every application. \texttt{NumLevel} indicates the number of levels of the pyramid, including the original image. Optionally a second value can be given. This value specifies the number (0..n) of the lowest level which is used the the matching. The region found up to this level will then be zoomed to the size of the original level. This can used to increase the runtime in the case that the accuracy does not have to be so high.

\begin{verbatim}


\end{verbatim}

\textbf{Parallelization Information}: \texttt{FastMatchMg} is reentrant and automatically parallelized (on tuple level).

\textbf{Possible Predecessors}: \texttt{CreateTemplate}, \texttt{ReadTemplate}, \texttt{AdaptTemplate}, \texttt{DrawRegion}, \texttt{DrawRectangle1}, \texttt{ReduceDomain}

\textbf{Alternatives}: \texttt{BestMatch}, \texttt{BestMatchMg}, \texttt{FastMatch}, \texttt{ExhaustiveMatch}, \texttt{ExhaustiveMatchMg}

\textbf{Module}: Template matching

HALCON/COM Reference Manual, 2005-2-1
### 7.1. Gray-Value-Based

#### void HTemplateX.ReadTemplate ([in] String FileName )

#### void HOperatorSetX.ReadTemplate ([in] VARIANT FileName, [out] VARIANT TemplateID )

**Reading a template from file.**

The operator **ReadTemplate** reads a matching template from file which has been written with **WriteTemplate**.

- **Parameter**
  - **FileName** (input control) ....... filename \(\sim\) String / VARIANT file name.
  - **TemplateID** (output control) ......... template \(\sim\) HTemplateX / VARIANT Template number.

If the file name is valid, the operator **ReadTemplate** returns the value TRUE. If necessary an exception handling will be raised.

**Parallelization Information**

**ReadTemplate** is processed completely exclusively without parallelization.

**Possible Successors**

AdaptTemplate, SetReferenceTemplate, SetOffsetTemplate, BestMatch, FastMatch, BestMatchRot

**Module**

Template matching

#### void HTemplateX.SetOffsetTemplate ([in] long GrayOffset )

#### void HOperatorSetX.SetOffsetTemplate ([in] VARIANT TemplateID, [in] VARIANT GrayOffset )

**Gray value offset for template.**

The operator **SetOffsetTemplate** adds a gray value offset to the template to eliminate gray value changes in the image. The parameter **GrayOffset** specifies a difference relative to the gray values of the pattern when it was created with **CreateTemplate** (not relative to the last call of **SetOffsetTemplate**). The values of **GrayOffset** has to be chosen according to the gray values of the image: A brighter image results in a positive value, a darker image results in a negative value. **SetOffsetTemplate** has to be called each time the gray values of the image changes. The gray values can be measured in a reference area using **Intensity** or **MinMaxGray**

- **Parameter**
  - **TemplateID** (input control) ......... template \(\sim\) HTemplateX / VARIANT Template number.
  - **GrayOffset** (input control) ............ number \(\sim\) long / VARIANT Offset of gray values.

  **Default Value** : 0
  **Suggested values** : GrayOffset \(\in\) \{-10, -5, -2, -1, 0, 1, 2, 5, 10\}
  **Typical range of values** : \(-255 \leq\) GrayOffset \(\leq\) -255
  **Minimum Increment** : 1
  **Recommended Increment** : 1

- **Result**

If the parameter values are correct, the operator **SetOffsetTemplate** returns the value TRUE. If necessary, an exception handling is raised.

**Parallelization Information**

**SetOffsetTemplate** is processed under mutual exclusion against itself and without parallelization.

**Possible Predecessors**

CreateTemplate, AdaptTemplate, ReadTemplate

HALCON 6.1.4
Possible Successors
BestMatch, BestMatchMg, BestMatchRot, FastMatch, FastMatchMg

Module

Template matching

void HTemplateX.SetReferenceTemplate ( [in] double Row, 
[ in] double Column )

void HOperatorSetX.SetReferenceTemplate ( [in] VARIANT TemplateID, 

Define reference position for a matching template.

SetReferenceTemplate allows to define a new reference position for a template. As default after calling CreateTemplate or CreateTemplateRot the center of gravity of the template is used. Using SetReferenceTemplate the reference position can be redefined. In the case of the center of gravity as reference the vector (0, 0) is returned after matching for a null translation of the pattern relative to the image.

Parameter

\(\bigtriangleup\) TemplateID (input control) ................................template \(\sim\) HTemplateX / VARIANT
Template number.

\(\bigtriangleup\) Row (input control) ............................................. point.y \(\sim\) double / VARIANT
Reference position of template (row).

\(\bigtriangleup\) Column (input control) ....................................... point.x \(\sim\) double / VARIANT
Reference position of template (column).

Result

If the parameter values are correct, the operator SetReferenceTemplate returns the value TRUE. If necessary, an exception handling is raised.

Parallelization Information

SetReferenceTemplate is processed under mutual exclusion against itself and without parallelization.

Possible Predecessors
CreateTemplate, CreateTemplateRot, ReadTemplate, AdaptTemplate

Possible Successors
BestMatch, BestMatchMg, BestMatchRot, FastMatch, FastMatchMg

Module

Template matching

void HTemplateX.WriteTemplate ( [in] String FileName )

void HOperatorSetX.WriteTemplate ( [in] VARIANT TemplateID, 
[ in] VARIANT FileName )

Writing a template to file.

The operator WriteTemplate writes a matching template to file which can be read again with ReadTemplate.

Parameter

\(\bigtriangleup\) TemplateID (input control) ................................template \(\sim\) HTemplateX / VARIANT
Template number.

\(\bigtriangleup\) FileName (input control) ......................................filename \(\sim\) String / VARIANT
file name.

Result

If the file name is valid (permission to write), the operator WriteTemplate returns the value TRUE. If necessary an exception handling will be raised.
7.2 SHAPE-BASED

WriteTemplate is reentrant and processed without parallelization.

Possible Predecessors
CreateTemplate, CreateTemplateRot

Template matching

7.2 Shape-Based

void HMiscX.ClearAllShapeModels ()
void HOperatorSetX.ClearAllShapeModels ()

Free the memory of all shape models.
The operator ClearAllShapeModels frees the memory of all shape models that were created by CreateShapeModel or CreateScaledShapeModel. After calling ClearAllShapeModels, no model can be used any longer.

Result
ClearAllShapeModels always returns TRUE.

Parallelization Information
ClearAllShapeModels is processed completely exclusively without parallelization.

Possible Predecessors
CreateShapeModel, CreateScaledShapeModel, ReadShapeModel, WriteShapeModel

Alternatives
ClearShapeModel

Template matching

void HOperatorSetX.ClearShapeModel ([in] VARIANT ModelID )

Free the memory of a shape model.
The operator ClearShapeModel frees the memory of a shape model that was created by CreateShapeModel or CreateScaledShapeModel. After calling ClearShapeModel, the model can no longer be used. The handle ModelID becomes invalid.

Parameter
ModelID (input control) ................. shape_model ~ HShapeModelX / VARIANT
Handle of the model.

Result
If the handle of the model is valid, the operator ClearShapeModel returns the value TRUE. If necessary an exception is raised.

Parallelization Information
ClearShapeModel is processed completely exclusively without parallelization.

Possible Predecessors
CreateShapeModel, CreateScaledShapeModel, ReadShapeModel, WriteShapeModel

See also
ClearAllShapeModels

Template matching
Prepare a shape model for scale invariant matching.

The operator CreateScaledShapeModel prepares a template, which is passed in the image Template, as a shape model used for scale invariant matching.

The model is generated using multiple image pyramid levels and multiple rotations and scales, and is stored in memory. The output parameter ModelID is a handle for this model, which is used in subsequent calls to FindScaledShapeModel.

The number of pyramid levels is determined with the parameter NumLevels. It should be chosen as large as possible because by this the time necessary to find the object is significantly reduced. On the other hand, NumLevels must be chosen such that the model is still recognizable and contains a sufficient number of points (at least four) on the highest pyramid level. This can be checked using the output of InspectShapeModel. If not enough model points are generated, the number of pyramid levels is reduced internally until enough model points are found on the highest pyramid level. If this procedure would lead to a model with no pyramid levels, i.e., if the number of model points is already too small on the lowest pyramid level, CreateScaledShapeModel returns with an error message. If NumLevels is set to 0, CreateScaledShapeModel determines the number of pyramid levels automatically. The automatically computed number of pyramid levels can be queried using GetShapeModelParams. In rare cases, it might happen that CreateScaledShapeModel determines a value for the number of pyramid levels that is too large or too small. If the number of pyramid levels is chosen too large, the model may not be recognized in the image or it may be necessary to select very low parameters for MinScore or Greediness in FindScaledShapeModel in order to find the model. If the number of pyramid levels is chosen too small, the time required to find the model in FindScaledShapeModel may increase. In these cases, the number of pyramid levels should be selected using the output of InspectShapeModel.

The parameters AngleStart and AngleExtent determine the range of possible rotations, in which the model can occur in the image. Note that the model can only be found in this range of angles by FindScaledShapeModel. The parameter AngleStep determines the step length within the selected range of angles. Hence, if subpixel accuracy is not specified in FindScaledShapeModel, this parameter specifies the accuracy that is achievable for the angles in FindScaledShapeModel. AngleStep should be chosen based on the size of the object. Smaller models do not have many different discrete rotations in the image, and hence AngleStep should be chosen larger for smaller models. If AngleExtent is not an integer multiple of AngleStep, AngleStep is modified accordingly.

The parameters ScaleMin and ScaleMax determine the range of possible scales (sizes) of the model. A scale of 1 corresponds to the original size of the model. The parameter ScaleStep determines the step length within the selected range of scales. Hence, if subpixel accuracy is not specified in FindScaledShapeModel, this parameter specifies the accuracy that is achievable for the scales in FindScaledShapeModel. Like AngleStep, ScaleStep should be chosen based on the size of the object. If the range of scales is not an integer multiple of ScaleStep, ScaleStep is modified accordingly.

The model is pre-generated for the selected angle and scale range and stored in memory. The memory required to store the model is proportional to the number of angle steps, the number of scale steps, and the number of points in the model. Hence, if AngleStep or ScaleStep are too small or AngleExtent or the range of scales are too big, it may happen that the model no longer fits into the (virtual) memory. In this case, either AngleStep or ScaleStep must be enlarged or AngleExtent or the range of scales must be reduced. In
any case, it is desirable that the model completely fits into the main memory, because this avoids paging by the operating system, and hence the time to find the object will be much smaller. Since angles can be determined with subpixel resolution by \texttt{FindScaledShapeModel}, \texttt{AngleStep} \geq 1° and \texttt{ScaleStep} \geq 0.02 can be selected for models of a diameter smaller than about 200 pixels. If \texttt{AngleStep} = 0 or \texttt{ScaleStep} = 0 is selected, \texttt{CreateScaledShapeModel} automatically determines a suitable angle or scale step length, respectively, based on the size of the model. The automatically computed angle and scale step lengths can be queried using \texttt{GetShapeModelParams}.

For particularly large models, it may be useful to reduce the number of model points by setting \texttt{Optimization} to a value different from ‘\texttt{none}’. If \texttt{Optimization} = ‘\texttt{none}’, all model points are stored. In all other cases, the number of points is reduced according to the value of \texttt{Optimization}. If the number of points is reduced, it may be necessary in \texttt{FindScaledShapeModel} to set the parameter \texttt{Greediness} to a smaller value, e.g., 0.7 or 0.8. For small models, the reduction of the number of model points does not result in a speed-up of the search because in this case usually significantly more potential instances of the model must be examined.

The parameter \texttt{Contrast} determines the contrast the model points must have. The contrast is a measure for local gray value differences between the object and the background and between different parts of the object. \texttt{Contrast} should be chosen such that only the significant features of the template are used for the model. \texttt{Contrast} can also contain a tuple with two values. In this case, the model is segmented using a method similar to the hysteresis threshold method used in \texttt{EdgesImage}. Here, the first element of the tuple determines the lower threshold, while the second element determines the upper threshold. For more information about the hysteresis threshold method, see \texttt{HysteresisThreshold}. Optionally, \texttt{Contrast} can contain a third value as the last element of the tuple. This value determines a threshold for the selection of significant model components based on the size of the components, i.e., components that have fewer points than the minimum size thus specified are suppressed. This threshold for the minimum size is divided by two for each successive pyramid level. If small model components should be suppressed, but hysteresis thresholding should not be performed, nevertheless three values must be specified in \texttt{Contrast}. In this case, the first two values can simply be set to identical values. The effect of this parameter can be checked in advance with \texttt{InspectShapeModel}.

With \texttt{MinContrast}, it can be determined which contrast the model must at least have in the recognition performed by \texttt{FindScaledShapeModel}. In other words, this parameter separates the model from the noise in the image. Therefore, a good choice is the range of gray value changes caused by the noise in the image. If, for example, the gray values fluctuate within a range of 10 gray levels, \texttt{MinContrast} should be set to 10. Obviously, \texttt{MinContrast} must be smaller than \texttt{Contrast}. If the model should be recognized in very low contrast images, \texttt{MinContrast} must be set to a correspondingly small value. If the model should be recognized even if it is severely occluded, \texttt{MinContrast} should be slightly larger than the range of gray value fluctuations created by noise in order to ensure that the position and rotation of the model are extracted robustly and accurately by \texttt{FindScaledShapeModel}.

The parameter \texttt{Metric} determines the conditions under which the model is recognized in the image. If \texttt{Metric} = ‘\texttt{use\_polarity}’, the object in the image and the model must have the same contrast. If, for example, the model is a bright object on a dark background, the object is found only if it is also brighter than the background. If \texttt{Metric} = ‘\texttt{ignore\_global\_polarity}’, the object is found in the image also if the contrast reverses globally. In the above example, the object hence is also found if it is darker than the background. The runtime of \texttt{FindScaledShapeModel} will increase slightly in this case. If \texttt{Metric} = ‘\texttt{ignore\_local\_polarity}’, the model is found even if the contrast changes locally. This mode can, for example, be useful if the object consists of a part with medium gray value, within which either darker or brighter sub-objects lie. Since in this case the runtime of \texttt{FindScaledShapeModel} increases significantly, it is usually better to create several models that reflect the possible contrast variations of the object with \texttt{CreateScaledShapeModel}, and to match them simultaneously with \texttt{FindScaledShapeModels}.

The center of gravity of the domain (region) of the model image \texttt{Template} is used as the origin (reference point) of the model. \texttt{FindScaledShapeModel} returns the coordinates of this point in the search image. A different origin can be set with \texttt{SetShapeModelOrigin}.

\begin{itemize}
  \item \textbf{Template} (input iconic) \texttt{image} ~\texttt{HImageX / HObjectX (byte)}
    Input image whose domain will be used to create the model.
  \item \textbf{NumLevels} (input control) \texttt{integer} ~\texttt{long / VARIANT}
    Maximum number of pyramid levels.
    \textbf{Default Value : 0}
    \textbf{List of values:} \texttt{NumLevels} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}
\end{itemize}

HALCON 6.1.4
CHAPTER 7. MATCHING

- **AngleStart** (input control) \(\text{angle.rad} \sim \text{double} / \text{VARIANT}\)
  - Smallest rotation of the pattern.
  - Default Value: -0.39
  - Suggested values: \(\text{AngleStart} \in \{-3.14, -1.57, -0.79, -0.39, -0.20, 0.0\}\)

- **AngleExtent** (input control) \(\text{angle.rad} \sim \text{double} / \text{VARIANT}\)
  - Extent of the rotation angles.
  - Default Value: 0.79
  - Suggested values: \(\text{AngleExtent} \in \{6.28, 3.14, 1.57, 0.79, 0.39\}\)

- **AngleStep** (input control) \(\text{angle.rad} \sim \text{double} / \text{VARIANT}\)
  - Step length of the angles (resolution).
  - Default Value: 0
  - Suggested values: \(\text{AngleStep} \in \{0.0175, 0.0349, 0.0524, 0.0698, 0.0873\}\)

- **ScaleMin** (input control) \(\text{number} \sim \text{double} / \text{VARIANT}\)
  - Minimum scale of the pattern.
  - Default Value: 0.9
  - Suggested values: \(\text{ScaleMin} \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}\)

- **ScaleMax** (input control) \(\text{number} \sim \text{double} / \text{VARIANT}\)
  - Maximum scale of the pattern.
  - Default Value: 1.1
  - Suggested values: \(\text{ScaleMax} \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}\)

- **ScaleStep** (input control) \(\text{number} \sim \text{double} / \text{VARIANT}\)
  - Scale step length (resolution).
  - Default Value: 0
  - Suggested values: \(\text{ScaleStep} \in \{0.0, 0.01, 0.02, 0.05, 0.1, 0.15, 0.2\}\)

- **Optimization** (input control) \(\text{string} \sim \text{String} / \text{VARIANT}\)
  - Kind of optimization.
  - Default Value: 'none'
  - List of values: \(\text{Optimization} \in \{\text{‘none’, ‘point_reduction_low’, ‘point_reduction_medium’, ‘point_reduction_high’}\}\)

- **Metric** (input control) \(\text{string} \sim \text{String} / \text{VARIANT}\)
  - Match metric.
  - Default Value: 'use_polarity'
  - List of values: \(\text{Metric} \in \{\text{‘use_polarity’, ‘ignore_global_polarity’, ‘ignore_local_polarity’}\}\)

- **Contrast** (input control) \(\text{number} \sim \text{VARIANT} / \text{integer}\)
  - Threshold or hysteresis thresholds for the contrast of the object in the template image and optionally minimum size of the object parts.
  - Default Value: 30
  - Suggested values: \(\text{Contrast} \in \{10, 20, 30, 40, 60, 80, 100, 120, 140, 160\}\)

- **MinContrast** (input control) \(\text{number} \sim \text{long} / \text{VARIANT}\)
  - Minimum contrast of the objects in the search images.
  - Default Value: 5
  - Suggested values: \(\text{MinContrast} \in \{1, 2, 3, 5, 7, 10, 20, 40\}\)

**ModelID** (output control) \(\text{shape_model} \sim \text{HShapeModelX} / \text{VARIANT}\)
- Handle of the model.

If the parameters are valid, the operator **CreateScaledShapeModel** returns the value TRUE. If necessary an exception is raised. If the parameters **NumLevels** and **Contrast** are chosen such that the model contains too few points, the error 8510 is raised.

**Parallelization Information**
**CreateScaledShapeModel** is processed completely exclusively without parallelization.

**Possible Predecessors**
DrawRegion, ReduceDomain, Threshold

HALCON/COM Reference Manual, 2005-2-1
Possible Successors
FindScaledShapeModel, FindScaledShapeModels, GetShapeModelParams,
ClearShapeModel, WriteShapeModel, SetShapeModelOrigin

Alternatives
CreateShapeModel, CreateTemplateRot

Template matching

```cpp
[out] HShapeModelX ModelID HImageX.CreateShapeModel
in VARIANT Contrast, [in] long MinContrast )
```

```cpp
void HShapeModelX.CreateShapeModel ([in] HImageX Template,
in VARIANT Contrast, [in] long MinContrast )
```

```cpp
void HOperatorSetX.CreateShapeModel ([in] IObjectX Template,
in VARIANT Contrast, [in] VARIANT MinContrast, [out] VARIANT ModelID )
```

Prepare a shape model for matching.
The operator CreateShapeModel prepares a template, which is passed in the image Template, as a shape model used for matching.

The model is generated using multiple image pyramid levels and multiple rotations and is stored in memory. The output parameter ModelID is a handle for this model, which is used in subsequent calls to FindShapeModel.

The number of pyramid levels is determined with the parameter NumLevels. It should be chosen as large as possible because by this the time necessary to find the object is significantly reduced. On the other hand, NumLevels must be chosen such that the model is still recognizable and contains a sufficient number of points (at least four) on the highest pyramid level. This can be checked using the output of InspectShapeModel. If not enough model points are generated, the number of pyramid levels is reduced internally until enough model points are found on the highest pyramid level. If this procedure would lead to a model with no pyramid levels, i.e., if the number of model points is already too small on the lowest pyramid level, CreateShapeModel returns with an error message. If NumLevels is set to 0, CreateShapeModel determines the number of pyramid levels automatically. The automatically computed number of pyramid levels can be queried using GetShapeModelParams. In rare cases, it might happen that CreateShapeModel determines a value for the number of pyramid levels that is too large or too small. If the number of pyramid levels is chosen too large, the model may not be recognized in the image or it may be necessary to select very low parameters for MinScore or Greediness in FindShapeModel in order to find the model. If the number of pyramid levels is chosen too small, the time required to find the model in FindShapeModel may increase. In these cases, the number of pyramid levels should be selected using the output of InspectShapeModel.

The parameters AngleStart and AngleExtent determine the range of possible rotations, in which the model can occur in the image. Note that the model can only be found in this range of angles by FindShapeModel. The parameter AngleStep determines the step length within the selected range of angles. Hence, if subpixel accuracy is not specified in FindShapeModel, this parameter specifies the accuracy that is achievable for the angles in FindShapeModel. AngleStep should be chosen based on the size of the object. Smaller models do not possess many different discrete rotations in the image, and hence AngleStep should be chosen larger for smaller models. If AngleExtent is not an integer multiple of AngleStep, AngleStep is modified accordingly. The model is pre-generated for the selected angle range and stored in memory. The memory required to store the model is proportional to the number of angle steps and the number of points in the model. Hence, if AngleStep is too small or AngleExtent too big, it may happen that the model no longer fits into the (virtual) memory. In this case, either AngleStep must be enlarged or AngleExtent must be reduced. In any case, it is desirable that the model completely fits into the main memory, because this avoids paging by the operating system, and hence the time to find the object will be much smaller. Since angles can be determined with subpixel
resolution by \texttt{FindShapeModel.AngleStep} \geq 1 can be selected for models of a diameter smaller than about 200 pixels. If \texttt{AngleStep} = 0 is selected, \texttt{CreateShapeModel} automatically determines a suitable angle step length based on the size of the model. The automatically computed angle step length can be queried using \texttt{GetShapeModelParams}.

For particularly large models, it may be useful to reduce the number of model points by setting \texttt{Optimization} to a value different from \textquote{none}. If \texttt{Optimization} = \textquote{none}, all model points are stored. In all other cases, the number of points is reduced according to the value of \texttt{Optimization}. If the number of points is reduced, it may be necessary in \texttt{FindShapeModel} to set the parameter \texttt{Greediness} to a smaller value, e.g., 0.7 or 0.8. For small models, the reduction of the number of model points does not result in a speed-up of the search because in this case usually significantly more potential instances of the model must be examined.

The parameter \texttt{Contrast} determines the contrast the model points must have. The contrast is a measure for local gray value differences between the object and the background and between different parts of the object. \texttt{Contrast} should be chosen such that only the significant features of the template are used for the model. \texttt{Contrast} can also contain a tuple with two values. In this case, the model is segmented using a method similar to the hysteresis threshold method used in \texttt{EdgesImage}. Here, the first element of the tuple determines the lower threshold, while the second element determines the upper threshold. For more information about the hysteresis threshold method, see \texttt{HysteresisThreshold}. Optionally, \texttt{Contrast} can contain a third value as the last element of the tuple. This value determines a threshold for the selection of significant model components based on the size of the components, i.e., components that have fewer points than the minimum size thus specified are suppressed. This threshold for the minimum size is divided by two for each successive pyramid level. If small model components should be suppressed, but hysteresis thresholding should not be performed, nevertheless three values must be specified in \texttt{Contrast}. In this case, the first two values can simply be set to identical values. The effect of this parameter can be checked in advance with \texttt{InspectShapeModel}.

With \texttt{MinContrast}, it can be determined which contrast the model must at least have in the recognition performed by \texttt{FindShapeModel}. In other words, this parameter separates the model from the noise in the image. Therefore, a good choice is the range of gray value changes caused by the noise in the image. If, for example, the gray values fluctuate within a range of 10 gray levels, \texttt{MinContrast} should be set to 10. Obviously, \texttt{MinContrast} must be smaller than \texttt{Contrast}. If the model should be recognized in very low contrast images, \texttt{MinContrast} must be set to a correspondingly small value. If the model should be recognized even if it is severely occluded, \texttt{MinContrast} should be slightly larger than the range of gray value fluctuations created by noise in order to ensure that the position and rotation of the model are extracted robustly and accurately by \texttt{FindShapeModel}.

The parameter \texttt{Metric} determines the conditions under which the model is recognized in the image. If \texttt{Metric} = \textquote{use\_polarity}, the object in the image and the model must have the same contrast. If, for example, the model is a bright object on a dark background, the object is found only if it is also brighter than the background. If \texttt{Metric} = \textquote{ignore\_global\_polarity}, the object is found in the image also if the contrast reverses globally. In the above example, the object hence is also found if it is darker than the background. The runtime of \texttt{FindShapeModel} will increase slightly in this case. If \texttt{Metric} = \textquote{ignore\_local\_polarity}, the object is found even if the contrast changes locally. This mode can, for example, be useful if the object consists of a part with medium gray value, within which either darker or brighter sub-objects lie. Since in this case the runtime of \texttt{FindShapeModel} increases significantly, it is usually better to create several models that reflect the possible contrast variations of the object with \texttt{CreateShapeModel}, and to match them simultaneously with \texttt{FindShapeModels}.

The center of gravity of the domain (region) of the model image \texttt{Template} is used as the origin (reference point) of the model. \texttt{FindShapeModel} returns the coordinates of this point in the search image. A different origin can be set with \texttt{SetShapeModelOrigin}.

---

**Parameter**

- **Template** (input iconic) \(\text{image} \sim HImageX / IObjectX \text{ (byte)}\)
  Input image whose domain will be used to create the model.

- **NumLevels** (input control) \(\text{integer} \sim \text{long} / \text{VARIANT}\)
  Maximum number of pyramid levels.
  \textbf{Default Value} : 0
  \textbf{List of values} : \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}

- **AngleStart** (input control) \(\text{angle.rad} \sim \text{double} / \text{VARIANT}\)
  Smallest rotation of the pattern.
  \textbf{Default Value} : -0.39
  \textbf{Suggested values} : \{\(-3.14, -1.57, -0.79, -0.39, -0.20, 0.0\)\}

---

HALCON/COM Reference Manual, 2005-2-1
7.2. SHAPE-BASED

- **AngleExtent** (input control) .......................................................... angle.rad ~ double / VARIANT
  Extent of the rotation angles.
  Default Value: 0.79
  Suggested values: AngleExtent ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
  Restriction: \( \text{AngleExtent} \geq 0 \)

- **AngleStep** (input control) .......................................................... angle.rad ~ double / VARIANT
  Step length of the angles (resolution).
  Default Value: 0
  Suggested values: AngleStep ∈ {0, 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
  Restriction: \( \text{AngleStep} \geq 0 \)

- **Optimization** (input control) ....................................................... string ~ String / VARIANT
  Kind of optimization.
  Default Value: 'none'
  List of values: Optimization ∈ {'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}

- **Metric** (input control) .............................................................. string ~ String / VARIANT
  Match metric.
  Default Value: 'use_polarity'
  List of values: Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity'}

- **Contrast** (input control) .......................................................... number(-array) ~ VARIANT (integer)
  Threshold or hysteresis thresholds for the contrast of the object in the template image and optionally minimum size of the object parts.
  Default Value: 30
  Suggested values: Contrast ∈ {10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

- **MinContrast** (input control) ...................................................... number ~ long / VARIANT
  Minimum contrast of the objects in the search images.
  Default Value: 5
  Suggested values: MinContrast ∈ {1, 2, 3, 5, 7, 10, 20, 40}
  Restriction: \( \text{MinContrast} < \text{Contrast} \)

- **ModelID** (output control) .......................................................... shape_model ~ HShapeModelIX / VARIANT
  Handle of the model.

---

If the parameters are valid, the operator **CreateShapeModel** returns the value TRUE. If necessary an exception is raised. If the parameters NumLevels and Contrast are chosen such that the model contains too few points, the error 8510 is raised.

---

**Parallelization Information**

**CreateShapeModel** is processed completely exclusively without parallelization.

---

**Possible Predecessors**

DrawRegion, ReduceDomain, Threshold

---

**Possible Successors**

FindShapeModel, FindShapeModels, GetShapeModelParams, ClearShapeModel, WriteShapeModel, SetShapeModelOrigin

---

**Alternatives**

CreateScaledShapeModel, CreateTemplateRot

---

Template matching
Find the best matches of a scale invariant shape model in an image.

The operator `FindScaledShapeModel` finds the best `NumMatches` instances of the scale invariant shape model `ModelID` in the input image `Image`. The model must have been created previously by calling `CreateScaledShapeModel` or `ReadShapeModel`.

The position, rotation, and scale of the found instances of the model are returned in `Row`, `Column`, `Angle`, and `Scale`. The coordinates `Row` and `Column` are the coordinates of the origin of the shape model in the search image. By default, the origin is the center of gravity of the domain (region) of the image that was used to create the shape model with `CreateScaledShapeModel`. A different origin can be set with `SetShapeModelOrigin`. Additionally, the score of each found instance is returned in `Score`. The score is a number between 0 and 1, which is an approximate measure of how much of the model is visible in the image. If, for example, half of the model is occluded, the score cannot exceed 0.5.

The domain of the image `Image` determines the search space for the reference point of the model, i.e., for the center of gravity of the domain (region) of the image that was used to create the shape model with `CreateScaledShapeModel`. A different origin set with `SetShapeModelOrigin` is not taken into account. The model is searched within those points of the domain of the image, in which the model lies completely within the image. This means that the model will not be found if it extends beyond the borders of the image, even if it would achieve a score greater than `MinScore` (see below). The parameters `AngleStart` and `AngleExtent` determine the range of rotations for which the model is searched. The parameters `ScaleMin` and `ScaleMax` determine the range of scales for which the model is searched. If necessary, both ranges are clipped to the range given when the model was created with `CreateScaledShapeModel`. In particular, this means that the angle ranges of the model and the search must truly overlap. The angle range in the search is not adapted modulo $2\pi$. To simplify the presentation, all angles in the remainder of the paragraph are given in degrees, whereas they have to be specified in radians in `FindScaledShapeModel`. Hence, if the model, for example, was created with $\text{AngleStart} = -20^\circ$ and $\text{AngleExtent} = 40^\circ$ and the angle search space in `FindScaledShapeModel` is, for example, set to $\text{AngleStart} = 350^\circ$ and $\text{AngleExtent} = 20^\circ$, the model will not be found, even though the angle ranges would overlap if they were regarded modulo $360^\circ$. To find the model, in this example it would be necessary to select $\text{AngleStart} = -10^\circ$.

The parameter `MinScore` determines what score a potential match must at least have to be regarded as an instance of the model in the image. The larger `MinScore` is chosen, the faster the search is. If the model can be expected never to be occluded in the images, `MinScore` may be set as high as 0.8 or even 0.9.

The maximum number of instances to be found can be determined with `NumMatches`. If more than `NumMatches` instances with a score greater than `MinScore` are found in the image, only the best `NumMatches` instances are returned. If fewer than `NumMatches` are found, only that number is returned, i.e., the parameter `MinScore` takes precedence over `NumMatches`.

If the model exhibits symmetries it may happen that multiple instances with similar positions but different rotations are found in the image. The parameter `MaxOverlap` determines by what fraction (i.e., a number between
0 and 1) two instances may at most overlap in order to consider them as different instances, and hence to be returned separately. If two instances overlap each other by more than MaxOverlap only the best instance is returned. The calculation of the overlap is based on the smallest enclosing rectangle of arbitrary orientation (see SmallestRectangle2) of the found instances. If MaxOverlap = 0, the found instances may not overlap at all, while for MaxOverlap = 1 all instances are returned.

The parameter SubPixel determines whether the instances should be extracted with subpixel accuracy. If SubPixel is set to 'none' (or 'false' for backwards compatibility) the model's pose is only determined with pixel accuracy and the angle and scale resolution that was specified with CreateScaledShapeModel. If SubPixel is set to 'interpolation' (or 'true') the position as well as the rotation and scale are determined with subpixel accuracy. In this mode, the model's pose is interpolated from the score function. This mode costs almost no computation time and achieves an accuracy that is high enough for most applications. In some applications, however, the accuracy requirements are extremely high. In these cases, the model's pose can be determined through a least-squares adjustment, i.e., by minimizing the distances of the model points to their corresponding image points. In contrast to 'interpolation', this mode requires additional computation time. The different modes for least-squares adjustment ('least_squares', 'least_squares_high', and 'least_squares_very_high') can be used to determine the accuracy with which the minimum distance is being searched. The higher the accuracy is chosen, the longer the subpixel extraction will take, however. Usually, SubPixel should be set to 'interpolation'. If least-squares adjustment is desired, 'least_squares' should be chosen because this results in the best tradeoff between run time and accuracy.

The number of pyramid levels used during the search is determined with NumLevels. If necessary, the number of levels is clipped to the range given when the shape model was created with CreateScaledShapeModel. If NumLevels is set to 0, the number of pyramid levels specified in CreateScaledShapeModel is used. If NumLevels is set to 0, the number of pyramid levels specified in CreateShapeModel is used. Optionally, NumLevels can contain a second value that determines the lowest pyramid level to which the found matches are tracked. Hence, a value of [4, 2] for NumLevels means that the matching starts at the fourth pyramid level and tracks the matches to the second lowest pyramid level (the lowest pyramid level is denoted by a value of 1). This mechanism can be used to decrease the runtime of the matching. It should be noted, however, that in general the accuracy of the extracted pose parameters is lower in this mode than in the normal mode, in which the matches are tracked to the lowest pyramid level. Hence, if a high accuracy is desired, SubPixel should be set to at least 'least_squares'. If the lowest pyramid level to use is chosen too large, it may happen that the desired accuracy cannot be achieved, or that wrong instances of the model are found because the model is not specific enough on the higher pyramid levels to facilitate a reliable selection of the correct instance of the model. In this case, the lowest pyramid level to use must be set to a smaller value.

The parameter Greediness determines how "greedily" the search should be carried out. If Greediness = 0, a safe search heuristic is used, which always finds the model if it is visible in the image. However, the search will be relatively time consuming in this case. If Greediness = 1, an unsafe search heuristic is used, which may cause the model not to be found in rare cases, even though it is visible in the image. For Greediness = 1, the maximum search speed is achieved. In almost all cases, the shape model will always be found for Greediness = 0.9.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Suggested values</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Input image in which the model should be found.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ModelID</td>
<td>Handle of the model.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AngleStart</td>
<td>Smallest rotation of the model.</td>
<td>-0.39</td>
<td>{-3.14, -1.57, -0.78, -0.39, -0.20, 0.0}</td>
<td></td>
</tr>
<tr>
<td>AngleExtent</td>
<td>Extent of the rotation angles.</td>
<td>0.78</td>
<td>{6.28, 3.14, 1.57, 0.78, 0.39, 0.0}</td>
<td>(AngleExtent \geq 0)</td>
</tr>
</tbody>
</table>
CHAPTER 7. MATCHING

- **ScaleMin** (input control) ...................................................... number \( \sim double / VARIANT \)  
  Minimum scale of the model.  
  Default Value : 0.9  
  Suggested values : ScaleMin \( \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \)  
  Restriction : \( (ScaleMin > 0) \)

- **ScaleMax** (input control) ...................................................... number \( \sim double / VARIANT \)  
  Maximum scale of the model.  
  Default Value : 1.1  
  Suggested values : ScaleMax \( \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\} \)  
  Restriction : \( (ScaleMax \geq ScaleMin) \)

- **MinScore** (input control) ..................................................... real \( \sim double / VARIANT \)  
  Minimum score of the instances of the model to be found.  
  Default Value : 0.5  
  Suggested values : MinScore \( \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \)  
  Typical range of values : \( 0 \leq MinScore \leq 0 \)  
  Minimum Increment : 0.01  
  Recommended Increment : 0.05

- **NumMatches** (input control) ................................................ integer \( \sim long / VARIANT \)  
  Number of instances of the model to be found.  
  Default Value : 1  
  Suggested values : NumMatches \( \in \{0, 1, 2, 3, 4, 5, 10\} \)

- **MaxOverlap** (input control) ................................................. real \( \sim double / VARIANT \)  
  Maximum overlap of the instances of the model to be found.  
  Default Value : 0.5  
  Suggested values : MaxOverlap \( \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \)  
  Typical range of values : \( 0 \leq MaxOverlap \leq 0 \)  
  Minimum Increment : 0.01  
  Recommended Increment : 0.05

- **SubPixel** (input control) ..................................................... string \( \sim String / VARIANT \)  
  Subpixel accuracy if not equal to ‘none’.  
  Default Value : ‘interpolation’  
  List of values : SubPixel \( \in \{’none’, ’interpolation’, ’least_squares’, ’least_squares_high’, ’least_squares_very_high’\} \)

- **NumLevels** (input control) ................................................ integer(-array) \( \sim VARIANT( integer ) \)  
  Number of pyramid levels used in the matching (and lowest pyramid level to use if \( |NumLevels| = 2 \)).  
  Default Value : 0  
  List of values : NumLevels \( \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \)

- **Greediness** (input control) ................................................ real \( \sim double / VARIANT \)  
  “Greediness” of the search heuristic (0: safe but slow; 1: fast but matches may be missed).  
  Default Value : 0.9  
  Suggested values : Greediness \( \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\} \)  
  Typical range of values : \( 0 \leq Greediness \leq 0 \)  
  Minimum Increment : 0.01  
  Recommended Increment : 0.05

- **Row** (output control) ......................................................... point.y \( \sim VARIANT( real ) \)  
  Row coordinate of the found instances of the model.

- **Column** (output control) .................................................... point.x \( \sim VARIANT( real ) \)  
  Column coordinate of the found instances of the model.

- **Angle** (output control) ...................................................... angle.rad \( \sim VARIANT( real ) \)  
  Rotation angle of the found instances of the model.

- **Scale** (output control) ...................................................... number \( \sim VARIANT( real ) \)  
  Scale of the found instances of the model.

- **Score** (output control) ..................................................... real \( \sim VARIANT( real ) \)  
  Score of the found instances of the model.

**Result**  
If the parameter values are correct, the operator **FindScaledShapeModel** returns the value TRUE.  
If the input is empty (no input images are available) the behavior can be set via **SetSystem** ('noObjectResult',<Result>). If necessary, an exception is raised.

HALCON/COM Reference Manual, 2005-2-1
Find the best matches of multiple scale invariant shape models.

The operator `FindScaledShapeModels` finds the best `NumMatches` instances of the scale invariant shape models that are passed in `ModelIDs` in the input image `Image`. The models must have been created previously by calling `CreateScaledShapeModel` or `ReadShapeModel`.

Hence, in contrast to `FindScaledShapeModel`, multiple models can be searched in the same image in one call. This changes the semantics of all input parameters to some extent. All input parameters must either contain one element, in which case the parameter is used for all models, or must contain the same number of elements as `ModelIDs`, in which case each parameter element refers to the corresponding element in `ModelIDs`. `NumLevels` may also contain either two or twice the number of elements as `ModelIDs`; see below.) As usual, the domain of the input image `Image` is used to restrict the search space for the reference point of the models `ModelIDs`. Consistent with the above semantics, the input image `Image` can therefore contain a single image object or an image object tuple containing multiple image objects. If `Image` contains a single image object, its domain is used as the region of interest for all models in `ModelIDs`. If `Image` contains multiple image objects, each domain is used as the region of interest for the corresponding model in `ModelIDs`. In this case, the image matrix of all image objects in the tuple must be identical, i.e. `Image` cannot be constructed in an arbitrary manner using `ConcatObj`, but must be created from the same image using `AddChannels` or equivalent calls. If this is not the case, an error message is returned. The above semantics also hold for the input control parameters. Hence, for example, `MinScore` can contain a single value or the same number of values as `ModelIDs`. In the first case, the value of `MinScore` is used for all models in `ModelIDs`, while in the second case the respective value of the elements in `MinScore` is used for the corresponding model in `ModelIDs`. An extension to these semantics holds for `NumMatches` and `MaxOverlap`. If `NumMatches` contains one element, `FindScaledShapeModels` returns the best `NumMatches` instances of the model irrespective of the type of the model. If, for example, two models are passed in `ModelIDs` and `NumMatches = 2` is selected, it can happen that two instances of the first model and no instances of the second model, one instance of the first model and one instance of the second
model, or no instances of the first model and two instances of the second model are returned. If, on the other hand, `NumMatches` contains multiple values, the number of instances returned of the different models corresponds to the number specified in the respective entry in `NumMatches`. If, for example, `NumMatches = (1, 1)` is selected, one instance of the first model and one instance of the second model is returned. For a detailed description of the semantics of `NumMatches`, see below. A similar extension of the semantics holds for `MaxOverlap`. If a single value is passed for `MaxOverlap`, the overlap is computed for all found instances of the different models, irrespective of the model type, i.e., instances of the same or of different models that overlap too much are eliminated. If, on the other hand, multiple values are passed in `MaxOverlap`, the overlap is only computed for found instances of the model that have the same model type, i.e., only instances of the same model that overlap too much are eliminated. In this mode, models of different types may overlap completely. For a detailed description of the semantics of `MaxOverlap`, see below. Hence, a call to `FindScaledShapeModels` with multiple values for `ModelIDs`, `NumMatches` and `MaxOverlap` has the same effect as multiple independent calls to `FindScaledShapeModel` with the respective parameters. However, a single call to `FindShapeModels` is considerably more efficient.

The type of the found instances of the models is returned in `Model`. The elements of `Model` are indices into the tuple `ModelIDs`, i.e., they can contain values from 0 to `|ModelIDs|` − 1. Hence, a value of 0 in an element of `Model` corresponds to an instance of the first model in `ModelIDs`.

The position, rotation, and scale of the found instances of the model are returned in `Row`, `Column`, `Angle`, and `Scale`. The coordinates `Row` and `Column` are the coordinates of the origin of the shape model in the search image. By default, the origin is the center of gravity of the domain (region) of the image that was used to create the shape model with `CreateScaledShapeModel`. A different origin can be set with `SetShapeModelOrigin`. Additionally, the score of each found instance is returned in `Score`. The score is a number between 0 and 1, which is an approximate measure of how much of the model is visible in the image. If, for example, half of the model is occluded, the score cannot exceed 0.5.

The domain of the image `Image` determines the search space for the reference point of the model, i.e., for the center of gravity of the domain (region) of the image that was used to create the shape model with `CreateScaledShapeModel`. A different origin set with `SetShapeModelOrigin` is not taken into account. The model is searched within those points of the domain of the image, in which the model lies completely within the image. This means that the model will not be found if it extends beyond the borders of the image, even if it would achieve a score greater than `MinScore` (see below). The parameters `AngleStart` and `AngleExtent` determine the range of rotations for which the model is searched. The parameters `ScaleMin` and `ScaleMax` determine the range of scales for which the model is searched. If necessary, both ranges are clipped to the range given when the model was created with `CreateScaledShapeModel`. In particular, this means that the angle ranges of the model and the search must truly overlap. The angle range in the search is not adapted modulo 2π. To simplify the presentation, all angles in the remainder of the paragraph are given in degrees, whereas they have to be specified in radians in `FindScaledShapeModels`. Hence, if the model, for example, was created with `AngleStart = −20°` and `AngleExtent = 40°` and the angle search space in `FindScaledShapeModels` is, for example, set to `AngleStart = 350°` and `AngleExtent = 20°`, the model will not be found, even though the angle ranges would overlap if they were regarded modulo 360°. To find the model, in this example it would be necessary to select `AngleStart = −10°`.

The parameter `MinScore` determines what score a potential match must at least have to be regarded as an instance of the model in the image. The larger `MinScore` is chosen, the faster the search is. If the model can be expected never to be occluded in the images, `MinScore` may be set as high as 0.8 or even 0.9.

The maximum number of instances to be found can be determined with `NumMatches`. If more than `NumMatches` instances with a score greater than `MinScore` are found in the image, only the best `NumMatches` instances are returned. If fewer than `NumMatches` are found, only that number is returned, i.e., the parameter `MinScore` takes precedence over `NumMatches`.

If the model exhibits symmetries it may happen that multiple instances with similar positions but different rotations are found in the image. The parameter `MaxOverlap` determines by what fraction (i.e., a number between 0 and 1) two instances may at most overlap in order to consider them as different instances, and hence to be returned separately. If two instances overlap each other by more than `MaxOverlap` only the best instance is returned. The calculation of the overlap is based on the smallest enclosing rectangle of arbitrary orientation (see `SmallestRectangle2`) of the found instances. If `MaxOverlap = 0`, the found instances may not overlap at all, while for `MaxOverlap = 1` all instances are returned.

The parameter `SubPixel` determines whether the instances should be extracted with subpixel accuracy. If `SubPixel` is set to 'none' (or 'false' for backwards compatibility) the model’s pose is only determined with pixel accuracy and the angle and scale resolution that was specified with `CreateScaledShapeModel`. If
SubPixel is set to ‘interpolation’ (or ‘true’) the position as well as the rotation and scale are determined with subpixel accuracy. In this mode, the model’s pose is interpolated from the score function. This mode costs almost no computation time and achieves an accuracy that is high enough for most applications. In some applications, however, the accuracy requirements are extremely high. In these cases, the model’s pose can be determined through a least-squares adjustment, i.e., by minimizing the distances of the model points to their corresponding image points. In contrast to ‘interpolation’, this mode requires additional computation time. The different modes for least-squares adjustment (‘least_squares’, ‘least_squares_high’, and ‘least_squares_extra_high’) can be used to determine the accuracy with which the minimum distance is being searched. The higher the accuracy is chosen, the longer the subpixel extraction will take, however. Usually, SubPixel should be set to ‘interpolation’. If least-squares adjustment is desired, ‘least_squares’ should be chosen because this results in the best tradeoff between run time and accuracy.

The number of pyramid levels used during the search is determined with NumLevels. If necessary, the number of levels is clipped to the range given when the shape model was created with CreateScaledShapeModel. If NumLevels is set to 0, the number of pyramid levels specified in CreateScaledShapeModel is used. Optionally, NumLevels can contain a second value that determines the lowest pyramid level to which the found matches are tracked. Hence, a value of \([4,2]\) for NumLevels means that the matching starts at the fourth pyramid level and tracks the matches to the second lowest pyramid level (the lowest pyramid level is denoted by a value of 1). This mechanism can be used to decrease the runtime of the matching. It should be noted, however, that in general the accuracy of the extracted pose parameters is lower in this mode than in the normal mode, in which the matches are tracked to the lowest pyramid level. Hence, if a high accuracy is desired, SubPixel should be set to at least ‘least_squares’. If the lowest pyramid level to use is chosen too large, it may happen that the desired accuracy cannot be achieved, or that wrong instances of the model are found because the model is not specific enough on the higher pyramid levels to facilitate a reliable selection of the correct instance of the model. In this case, the lowest pyramid level to use must be set to a smaller value. If the lowest pyramid level is specified separately for each model, NumLevels must contain twice the number of elements as ModelIDs. In this case, the number of pyramid levels and the lowest pyramid level must be specified interleaved in NumLevels. If, for example, two models are specified in ModelIDs, the number of pyramid levels is 5 for the first model and 4 for the second model, and the lowest pyramid level is 2 for the first model and 1 for the second model, NumLevels = [5, 2, 4, 1] must be selected. If exactly two models are specified in ModelIDs, a special case occurs. If in this case the lowest pyramid level is to be specified, the number of pyramid levels and the lowest pyramid level must be specified explicitly for both models, even if they are identical, because specifying two values in NumLevels is interpreted as the explicit specification of the number of pyramid levels for the two models.

The parameter Greediness determines how “greedily” the search should be carried out. If Greediness = 0, a safe search heuristic is used, which always finds the model if it is visible in the image. However, the search will be relatively time consuming in this case. If Greediness = 1, an unsafe search heuristic is used, which may cause the model not to be found in rare cases, even though it is visible in the image. For Greediness = 1, the maximum search speed is achieved. In almost all cases, the shape model will always be found for Greediness = 0.9.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Suggested Values</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>(input iconic) image(-array) (\sim HImageX / HObjectX) (byte)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ModelIDs</td>
<td>(input control) shape_model(-array) (\sim HShapeModelX / VARIANT) (integer)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>AngleStart</td>
<td>(input control) angle.rad(-array) (\sim VARIANT) (real)</td>
<td>-0.39</td>
<td>([-3.14, -1.57, -0.78, -0.39, -0.20, 0.0])</td>
<td>-</td>
</tr>
<tr>
<td>AngleExtent</td>
<td>(input control) angle.rad(-array) (\sim VARIANT) (real)</td>
<td>0.78</td>
<td>([6.28, 3.14, 1.57, 0.78, 0.39, 0.0])</td>
<td>((\text{AngleExtent} \geq 0))</td>
</tr>
<tr>
<td>ScaleMin</td>
<td>(input control) number(-array) (\sim VARIANT) (real)</td>
<td>0.9</td>
<td>([0.5, 0.6, 0.7, 0.8, 0.9, 1.0])</td>
<td>((\text{ScaleMin} &gt; 0))</td>
</tr>
</tbody>
</table>
CHAPTER 7. MATCHING

- ScaleMax (input control) .................................................. number(-array) \rightarrow \text{VARIANT (real)}
  Maximum scale of the models.
  Default Value: 1.1
  Suggested values: ScaleMax \in \{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}
  Restriction: (ScaleMax \geq ScaleMin)

- MinScore (input control) .................................................. real(-array) \rightarrow \text{VARIANT (real)}
  Minimum score of the instances of the models to be found.
  Default Value: 0.5
  Suggested values: MinScore \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}
  Typical range of values: 0 \leq \text{MinScore} \leq 0
  Minimum Increment: 0.01
  Recommended Increment: 0.05

- NumMatches (input control) .............................................. integer(-array) \rightarrow \text{VARIANT (integer)}
  Number of instances of the models to be found.
  Default Value: 1
  Suggested values: NumMatches \in \{0, 1, 2, 3, 4, 5, 10, 20\}

- MaxOverlap (input control) ............................................. real(-array) \rightarrow \text{VARIANT (real)}
  Maximum overlap of the instances of the models to be found.
  Default Value: 0.5
  Suggested values: MaxOverlap \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}
  Typical range of values: 0 \leq \text{MaxOverlap} \leq 0
  Minimum Increment: 0.01
  Recommended Increment: 0.05

- SubPixel (input control) ................................................ string(-array) \rightarrow \text{VARIANT (string)}
  Subpixel accuracy if not equal to 'none'.
  Default Value: 'interpolation'
  List of values: SubPixel \in \{'none', 'interpolation', 'least_squares', 'least_squares\_high',
  'least_squares\_very\_high'\}

- NumLevels (input control) ............................................. integer(-array) \rightarrow \text{VARIANT (integer)}
  Number of pyramid levels used in the matching (and lowest pyramid level to use if |\text{NumLevels}| = 2).
  Default Value: 0
  List of values: NumLevels \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}

- Greediness (input control) .......................................... real(-array) \rightarrow \text{VARIANT (real)}
  “Greediness” of the search heuristic (0: safe but slow; 1: fast but matches may be missed).
  Default Value: 0.9
  Suggested values: Greediness \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}
  Typical range of values: 0 \leq \text{Greediness} \leq 0
  Minimum Increment: 0.01
  Recommended Increment: 0.05

- Row (output control) ..................................................... point.y \rightarrow \text{VARIANT (real)}
  Row coordinate of the found instances of the models.

- Column (output control) .............................................. point.x \rightarrow \text{VARIANT (real)}
  Column coordinate of the found instances of the models.

- Angle (output control) ................................................ angle.rad \rightarrow \text{VARIANT (real)}
  Rotation angle of the found instances of the models.

- Scale (output control) ................................................ number \rightarrow \text{VARIANT (real)}
  Scale of the found instances of the models.

- Score (output control) ................................................ real \rightarrow \text{VARIANT (real)}
  Score of the found instances of the models.

- Model (output control) .............................................. integer \rightarrow \text{VARIANT (integer)}
  Index of the found instances of the models.

\begin{align*}
\text{Result} & \quad \text{if the parameter values are correct, the operator \text{FindScaledShapeModels} returns the value TRUE.} \\
\text{SetSystem} & \quad \text{if the input is empty (no input images are available) the behavior can be set via \text{SetSystem ('noObjectResult',<Result>).} If necessary, an exception is raised.} \\
\text{Parallelization Information} & \quad \text{\text{FindScaledShapeModels} is reentrant and processed without parallelization.} \\
\end{align*}

HALCON/COM Reference Manual, 2005-2-1
Find the best matches of a shape model in an image.

The operator `FindShapeModel` finds the best `NumMatches` instances of the shape model `ModelID` in the input image `Image`. The model must have been created previously by calling `CreateShapeModel` or `ReadShapeModel`.

The position and rotation of the found instances of the model is returned in `Row`, `Column` and `Angle`. The coordinates `Row` and `Column` are the coordinates of the origin of the shape model in the search image. By default, the origin is the center of gravity of the domain (region) of the image that was used to create the shape model with `CreateShapeModel`. A different origin can be set with `SetShapeModelOrigin`. Additionally, the score of each found instance is returned in `Score`. The score is a number between 0 and 1, which is an approximate measure of how much of the model is visible in the image. If, for example, half of the model is occluded, the score cannot exceed 0.5.

The domain of the image `Image` determines the search space for the reference point of the model, i.e., for the center of gravity of the domain (region) of the image that was used to create the shape model with `CreateShapeModel`. A different origin set with `SetShapeModelOrigin` is not taken into account. The model is searched within those points of the domain of the image, in which the model lies completely within the image. This means that the model will not be found if it extends beyond the borders of the image, even if it would achieve a score greater than `MinScore` (see below). The parameters `AngleStart` and `AngleExtent` determine the range of rotations for which the model is searched. If necessary, the range of rotations is clipped to the range given when the model was created with `CreateShapeModel`. In particular, this means that the angle ranges of the model and the search must truly overlap. The angle range in the search is not adapted modulo $2\pi$. To simplify the presentation, all angles in the remainder of the paragraph are given in degrees, whereas they have to be specified in radians in `FindShapeModel`. Hence, if the model, for example, was created with `AngleStart = -20^\circ` and `AngleExtent = 40^\circ` and the angle search space in `FindShapeModel` is, for example, set to `AngleStart = 350^\circ` and `AngleExtent = 20^\circ`, the model will not be found, even though the angle ranges would overlap if they were regarded modulo $360^\circ$. To find the model, in this example it would be necessary to select `AngleStart = -10^\circ`.

The parameter `MinScore` determines what score a potential match must at least have to be regarded as an instance of the model in the image. The larger `MinScore` is chosen, the faster the search is. If the model can be expected never to be occluded in the images, `MinScore` may be set as high as 0.8 or even 0.9.

The maximum number of instances to be found can be determined with `NumMatches`. If more than
NumMatches instances with a score greater than MinScore are found in the image, only the best NumMatches instances are returned. If fewer than NumMatches are found, only that number is returned, i.e., the parameter MinScore takes precedence over NumMatches.

If the model exhibits symmetries it may happen that multiple instances with similar positions but different rotations are found in the image. The parameter MaxOverlap determines by what fraction (i.e., a number between 0 and 1) two instances may at most overlap in order to consider them as different instances, and hence to be returned separately. If two instances overlap each other by more than MaxOverlap only the best instance is returned. The calculation of the overlap is based on the smallest enclosing rectangle of arbitrary orientation (see SmallestRectangle2) of the found instances. If MaxOverlap = 0, the found instances may not overlap at all, while for MaxOverlap = 1 all instances are returned.

The parameter SubPixel determines whether the instances should be extracted with subpixel accuracy. If SubPixel is set to 'none' (or 'false' for backwards compatibility) the model’s pose is only determined with pixel accuracy and the angle resolution that was specified with CreateShapeModel. If SubPixel is set to 'interpolation' (or 'true') the position as well as the rotation are determined with subpixel accuracy. In this mode, the model’s pose is interpolated from the score function. This mode costs almost no computation time and achieves an accuracy that is high enough for most applications. In some applications, however, the accuracy requirements are extremely high. In these cases, the model’s pose can be determined through a least-squares adjustment, i.e., by minimizing the distances of the model points to their corresponding image points. In contrast to 'interpolation', this mode requires additional computation time. The different modes for least-squares adjustment ('least_squares', 'least_squares_high', and 'least_squares_very_high') can be used to determine the accuracy with which the minimum distance is being searched. The higher the accuracy is chosen, the longer the subpixel extraction will take, however. Usually, SubPixel should be set to 'interpolation'. If least-squares adjustment is desired, 'least_squares' should be chosen because this results in the best tradeoff between run time and accuracy.

The number of pyramid levels used during the search is determined with NumLevels. If necessary, the number of levels is clipped to the range given when the shape model was created with CreateShapeModel. If NumLevels is set to 0, the number of pyramid levels specified in CreateShapeModel is used. Optionally, NumLevels can contain a second value that determines the lowest pyramid level to which the found matches are tracked. Hence, a value of [4,2] for NumLevels means that the matching starts at the fourth pyramid level and tracks the matches to the second lowest pyramid level (the lowest pyramid level is denoted by a value of 1). This mechanism can be used to decrease the runtime of the matching. It should be noted, however, that in general the accuracy of the extracted pose parameters is lower in this mode than in the normal mode, in which the matches are tracked to the lowest pyramid level. Hence, if a high accuracy is desired, SubPixel should be set to at least 'least_squares'. If the lowest pyramid level to use is chosen too large, it may happen that the desired accuracy cannot be achieved, or that wrong instances of the model are found because the model is not specific enough on the higher pyramid levels to facilitate a reliable selection of the correct instance of the model. In this case, the lowest pyramid level to use must be set to a smaller value.

The parameter Greediness determines how “greedily” the search should be carried out. If Greediness = 0, a safe search heuristic is used, which always finds the model if it is visible in the image. However, the search will be relatively time consuming in this case. If Greediness = 1, an unsafe search heuristic is used, which may cause the model not to be found in rare cases, even though it is visible in the image. For Greediness = 1, the maximum search speed is achieved. In almost all cases, the shape model will always be found for Greediness = 0.9.

Parameter

- **Image** (input iconic) .......................... image ~ HImageX / IHObjectX (byte)
  Input image in which the model should be found.

- **ModelID** (input control) .......................... shape_model ~ HShapeModelX / VARIANT
  Handle of the model.

- **AngleStart** (input control) .......................... angle.rad ~ double / VARIANT
  Smallest rotation of the model.
  Default Value : -0.39
  Suggested values : AngleStart ∈ [-3.14, -1.57, -0.78, -0.39, -0.20, 0.0]

- **AngleExtent** (input control) .......................... angle.rad ~ double / VARIANT
  Extent of the rotation angles.
  Default Value : 0.78
  Suggested values : AngleExtent ∈ {6.28, 3.14, 1.57, 0.78, 0.39, 0.0}
  Restriction : (AngleExtent ≥ 0)
7.2. SHAPE-BASED

- **MinScore** (input control) .......................... real  \(\sim\) double / VARIANT
  
  Minimum score of the instances of the model to be found.

  - **Default Value**: 0.5
  - **Suggested values**: \(\text{MinScore} \in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}\)
  - **Typical range of values**: \(0 \leq \text{MinScore} \leq 0\)
  - **Minimum Increment**: 0.01
  - **Recommended Increment**: 0.05

- **NumMatches** (input control) ........................ integer  \(\sim\) long / VARIANT
  
  Number of instances of the model to be found.

  - **Default Value**: 1
  - **Suggested values**: \(\text{NumMatches} \in \{0, 1, 2, 3, 4, 5, 10, 20\}\)

- **MaxOverlap** (input control) .......................... real  \(\sim\) double / VARIANT
  
  Maximum overlap of the instances of the model to be found.

  - **Default Value**: 0.5
  - **Suggested values**: \(\text{MaxOverlap} \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}\)
  - **Typical range of values**: \(0 \leq \text{MaxOverlap} \leq 0\)
  - **Minimum Increment**: 0.01
  - **Recommended Increment**: 0.05

- **SubPixel** (input control) .............................. string  \(\sim\) String / VARIANT
  
  Subpixel accuracy if not equal to ‘none’.

  - **Default Value**: ‘interpolation’
  - **List of values**: \(\text{SubPixel} \in \{\text{‘none’}, \text{‘interpolation’}, \text{‘least_squares’}, \text{‘least_squares_high’}, \text{‘least_squares_very_high’}\}\)

- **NumLevels** (input control) ............................ integer(-array)  \(\sim\) VARIANT ( integer )
  
  Number of pyramid levels used in the matching (and lowest pyramid level to use if \(|\text{NumLevels}| = 2\)).

  - **Default Value**: 0
  - **List of values**: \(\text{NumLevels} \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}\)

- **Greediness** (input control) .......................... real  \(\sim\) double / VARIANT
  
  “Greediness” of the search heuristic (0: safe but slow; 1: fast but matches may be missed).

  - **Default Value**: 0.9
  - **Suggested values**: \(\text{Greediness} \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}\)
  - **Typical range of values**: \(0 \leq \text{Greediness} \leq 0\)
  - **Minimum Increment**: 0.01
  - **Recommended Increment**: 0.05

- **Row** (output control) ................................. point.y  \(\sim\) VARIANT ( real )
  
  Row coordinate of the found instances of the model.

- **Column** (output control) .............................. point.x  \(\sim\) VARIANT ( real )
  
  Column coordinate of the found instances of the model.

- **Angle** (output control) .............................. angle.rad  \(\sim\) VARIANT ( real )
  
  Rotation angle of the found instances of the model.

- **Score** (output control) .............................. real  \(\sim\) VARIANT ( real )
  
  Score of the found instances of the model.

---

**Result**

If the parameter values are correct, the operator **FindShapeModel** returns the value TRUE. If the input is empty (no input images are available) the behavior can be set via **SetSystem(‘noObjectResult’,<Result>).** If necessary, an exception is raised.

---

**Parallelization Information**

**FindShapeModel** is reentrant and processed without parallelization.

---

**Possible Predecessors**

CreateShapeModel, ReadShapeModel, SetShapeModelOrigin

---

**Alternatives**

FindScaledShapeModel, FindScaledShapeModels, FindShapeModels, BestMatchRotMg

---

**Module**

Template matching
Find the best matches of multiple shape models.

The operator \texttt{FindShapeModels} finds the best \texttt{NumMatches} instances of the shape models that are passed in the tuple \texttt{ModelIDs} in the input image \texttt{Image}. The models must have been created previously by calling \texttt{CreateShapeModel} or \texttt{ReadShapeModel}.

Hence, in contrast to \texttt{FindShapeModel}, multiple models can be searched in the same image in one call. This changes the semantics of all input parameters to some extent. All input parameters must either contain one element, in which case the parameter is used for all models, or must contain the same number of elements as \texttt{ModelIDs}, in which case each parameter element refers to the corresponding element in \texttt{ModelIDs}. (\texttt{NumLevels} may also contain either two or twice the number of elements as \texttt{ModelIDs}; see below.) As usual, the domain of the input image \texttt{Image} is used to restrict the search space for the reference point of the models \texttt{ModelIDs}. Consistent with the above semantics, the input image \texttt{Image} can therefore contain a single image object or an image object tuple containing multiple image objects. If \texttt{Image} contains a single image object, its domain is used as the region of interest for all models in \texttt{ModelIDs}. If \texttt{Image} contains multiple image objects, each domain is used as the region of interest for the corresponding model in \texttt{ModelIDs}. In this case, the image matrix of all image objects in the tuple must be identical, i.e., \texttt{Image} cannot be constructed in an arbitrary manner using \texttt{ConcatObj}, but must be created from the same image using \texttt{AddChannels} or equivalent calls. If this is not the case, an error message is returned. The above semantics also hold for the input control parameters. Hence, for example, \texttt{MinScore} can contain a single value or the same number of values as \texttt{ModelIDs}. In the first case, the value of \texttt{MinScore} is used for all models in \texttt{ModelIDs}, while in the second case the respective value of the elements in \texttt{MinScore} is used for the corresponding model in \texttt{ModelIDs}. An extension to these semantics holds for \texttt{NumMatches} and \texttt{MaxOverlap}. If \texttt{NumMatches} contains one element, \texttt{FindShapeModels} returns the best \texttt{NumMatches} instances of the model irrespective of the type of the model. If, for example, two models are passed in \texttt{ModelIDs} and \texttt{NumMatches} = 2 is selected, it can happen that two instances of the first model and no instances of the second model, one instance of the first model and one instance of the second model, or no instances of the first model and two instances of the second model are returned. If, on the other hand, \texttt{NumMatches} contains multiple values, the number of instances returned of the different models corresponds to the number specified in the respective entry in \texttt{NumMatches}. If, for example, \texttt{NumMatches} = [1, 1] is selected, one instance of the first model and one instance of the second model is returned. For a detailed description of the semantics of \texttt{NumMatches}, see below. A similar extension of the semantics holds for \texttt{MaxOverlap}. If a single value is passed for \texttt{MaxOverlap}, the overlap is computed for all found instances of the different models, irrespective of the model type, i.e., instances of the same or of different models that overlap too much are eliminated. If, on the other hand, multiple values are passed in \texttt{MaxOverlap}, the overlap is only computed for found instances of the model that have the same model type, i.e., only instances of the same model that overlap too much are eliminated. In this mode, models of different types may overlap completely. For a detailed description of the semantics of \texttt{MaxOverlap}, see below. Hence, a call to \texttt{FindShapeModels} with multiple values for \texttt{ModelIDs}, \texttt{NumMatches} and \texttt{MaxOverlap} has the same effect as multiple independent calls to \texttt{FindShapeModel} with the respective parameters. However, a single call to \texttt{FindShapeModels} is considerably more efficient.

The type of the found instances of the models is returned in \texttt{Model}. The elements of \texttt{Model} are indices into the tuple \texttt{ModelIDs}, i.e., they can contain values from 0 to \texttt{|ModelIDs| − 1}. Hence, a value of 0 in an element of \texttt{Model} corresponds to an instance of the first model in \texttt{ModelIDs}.
The position and rotation of the found instances of the model is returned in \texttt{Row}, \texttt{Column} and \texttt{Angle}. The coordinates \texttt{Row} and \texttt{Column} are the coordinates of the origin of the shape model in the search image. By default, the origin is the center of gravity of the domain (region) of the image that was used to create the shape model with \texttt{CreateShapeModel}. A different origin can be set with \texttt{SetShapeModelOrigin}. Additionally, the score of each found instance is returned in \texttt{Score}. The score is a number between 0 and 1, which is an approximate measure of how much of the model is visible in the image. If, for example, half of the model is occluded, the score cannot exceed 0.5.

The domain of the image \texttt{Image} determines the search space for the reference point of the model, i.e., for the center of gravity of the domain (region) of the image that was used to create the shape model with \texttt{CreateShapeModel}. A different origin set with \texttt{SetShapeModelOrigin} is not taken into account. The model is searched within those points of the domain of the image, in which the model lies completely within the image. This means that the model will not be found if it extends beyond the borders of the image, even if it would achieve a score greater than \texttt{MinScore} (see below). The parameters \texttt{AngleStart} and \texttt{AngleExtent} determine the range of rotations for which the model is searched. If necessary, the range of rotations is clipped to the range given when the model was created with \texttt{CreateShapeModel}. In particular, this means that the angle ranges of the model and the search must truly overlap. The angle range in the search is not adapted modulo $2\pi$. To simplify the presentation, all angles in the remainder of the paragraph are given in degrees, whereas they have to be specified in radians in \texttt{FindShapeModels}. Hence, if the model, for example, was created with \texttt{AngleStart} = $-20^\circ$ and \texttt{AngleExtent} = $40^\circ$ and the angle search space in \texttt{FindShapeModels} is, for example, set to \texttt{AngleStart} = $350^\circ$ and \texttt{AngleExtent} = $20^\circ$, the model will not be found, even though the angle ranges would overlap if they were regarded modulo $360^\circ$. To find the model, in this example it would be necessary to select \texttt{AngleStart} = $-10^\circ$.

The parameter \texttt{MinScore} determines what score a potential match must at least have to be regarded as an instance of the model in the image. The larger \texttt{MinScore} is chosen, the faster the search is. If the model can be expected never to be occluded in the images, \texttt{MinScore} may be set as high as 0.8 or even 0.9.

The maximum number of instances to be found can be determined with \texttt{NumMatches}. If more than \texttt{NumMatches} instances with a score greater than \texttt{MinScore} are found in the image, only the best \texttt{NumMatches} instances are returned. If fewer than \texttt{NumMatches} are found, only that number is returned, i.e., the parameter \texttt{MinScore} takes precedence over \texttt{NumMatches}.

If the model exhibits symmetries it may happen that multiple instances with similar positions but different rotations are found in the image. The parameter \texttt{MaxOverlap} determines by what fraction (i.e., a number between 0 and 1) two instances may at most overlap in order to consider them as different instances, and hence to be returned separately. If two instances overlap each other by more than \texttt{MaxOverlap} only the best instance is returned. The calculation of the overlap is based on the smallest enclosing rectangle of arbitrary orientation (see \texttt{SmallestRectangle2}) of the found instances. If \texttt{MaxOverlap} = 0, the found instances may not overlap at all, while for \texttt{MaxOverlap} = 1 all instances are returned.

The parameter \texttt{SubPixel} determines whether the instances should be extracted with subpixel accuracy. If \texttt{SubPixel} is set to 'none' (or 'false' for backwards compatibility) the model's pose is only determined with pixel accuracy and the angle resolution that was specified with \texttt{CreateShapeModel}. If \texttt{SubPixel} is set to 'interpolation' (or 'true') the position as well as the rotation are determined with subpixel accuracy. In this mode, the model's pose is interpolated from the score function. This mode costs almost no computation time and achieves an accuracy that is high enough for most applications. In some applications, however, the accuracy requirements are extremely high. In these cases, the model's pose can be determined through a least-squares adjustment, i.e., by minimizing the distances of the model points to their corresponding image points. In contrast to 'interpolation', this mode requires additional computation time. The different modes for least-squares adjustment ('least_squares', 'least_squares_high', and 'least_squares_very_high') can be used to determine the accuracy with which the minimum distance is being searched. The higher the accuracy is chosen, the longer the subpixel extraction will take, however. Usually, \texttt{SubPixel} should be set to 'interpolation'. If least-squares adjustment is desired, 'least_squares' should be chosen because this results in the best tradeoff between run time and accuracy.

The number of pyramid levels used during the search is determined with \texttt{NumLevels}. If necessary, the number of levels is clipped to the range given when the shape model was created with \texttt{CreateShapeModel}. If \texttt{NumLevels} is set to 0, the number of pyramid levels specified in \texttt{CreateShapeModel} is used. Optionally, \texttt{NumLevels} can contain a second value that determines the lowest pyramid level to which the found matches are tracked. Hence, a value of $[4,2]$ for \texttt{NumLevels} means that the matching starts at the fourth pyramid level and tracks the matches to the second lowest pyramid level (the lowest pyramid level is denoted by a value of 1). This mechanism can be used to decrease the runtime of the matching. It should be noted, however, that in general the accuracy of the extracted pose parameters is lower in this mode than in the normal mode, in which the matches
are tracked to the lowest pyramid level. Hence, if a high accuracy is desired, SubPixel should be set to at least ‘least_squares’. If the lowest pyramid level to use is chosen too large, it may happen that the desired accuracy cannot be achieved, or that wrong instances of the model are found because the model is not specific enough on the higher pyramid levels to facilitate a reliable selection of the correct instance of the model. In this case, the lowest pyramid level to use must be set to a smaller value. If the lowest pyramid level is specified separately for each model, NumLevels must contain twice the number of elements as ModelIDs. In this case, the number of pyramid levels and the lowest pyramid level must be specified interleaved in NumLevels. If, for example, two models are specified in ModelIDs, the number of pyramid levels is 5 for the first model and 4 for the second model, and the lowest pyramid level is 2 for the first model and 1 for the second model, NumLevels = [5, 2, 4, 1] must be selected. If exactly two models are specified in ModelIDs, a special case occurs. If in this case the lowest pyramid level is to be specified, the number of pyramid levels and the lowest pyramid level must be specified explicitly for both models, even if they are identical, because specifying two values in NumLevels is interpreted as the explicit specification of the number of pyramid levels for the two models.

The parameter Greediness determines how “greedily” the search should be carried out. If Greediness = 0, a safe search heuristic is used, which always finds the model if it is visible in the image. However, the search will be relatively time consuming in this case. If Greediness = 1, an unsafe search heuristic is used, which may cause the model not to be found in rare cases, even though it is visible in the image. For Greediness = 1, the maximum search speed is achieved. In almost all cases, the shape model will always be found for Greediness = 0.9.

- **Image** (input iconic) ........................................... image(-array)  ~  HImageX / IObjectX (byte)
  Input image in which the models should be found.
- **ModelIDs** (input control) ...................... shape_model(-array)  ~  HShapeModelX / VARIANT (integer)
  Handle of the models.
- **AngleStart** (input control) ...................... angle.rad(-array)  ~  VARIANT (real)
  Smallest rotation of the models.
  Default Value : -0.39
  Suggested values : AngleStart ∈ {-3.14, -1.57, -0.78, -0.39, -0.20, 0.0}
- **AngleExtent** (input control) ...................... angle.rad(-array)  ~  VARIANT (real)
  Extent of the rotation angles.
  Default Value : 0.78
  Suggested values : AngleExtent ∈ {6.28, 3.14, 1.57, 0.78, 0.39, 0.0}
  Restriction : (AngleExtent ≥ 0)
- **MinScore** (input control) .................................. real(-array)  ~  VARIANT (real)
  Minimum score of the instances of the models to be found.
  Default Value : 0.5
  Suggested values : MinScore ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typical range of values : 0 ≤ MinScore ≤ 0
  Minimum Increment : 0.01
  Recommended Increment : 0.05
- **NumMatches** (input control) ....................... integer(-array)  ~  VARIANT (integer)
  Number of instances of the models to be found.
  Default Value : 1
  Suggested values : NumMatches ∈ {0, 1, 2, 3, 4, 5, 10, 20}
- **MaxOverlap** (input control) ............................. real(-array)  ~  VARIANT (real)
  Maximum overlap of the instances of the models to be found.
  Default Value : 0.5
  Suggested values : MaxOverlap ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
  Typical range of values : 0 ≤ MaxOverlap ≤ 0
  Minimum Increment : 0.01
  Recommended Increment : 0.05
- **SubPixel** (input control) ................................. string(-array)  ~  VARIANT (string)
  Subpixel accuracy if not equal to 'none'.
  Default Value : 'interpolation'
  List of values : SubPixel ∈ {'none', 'interpolation', 'least_squares', 'least_squares_high', 'least_squares_very_high'}
7.2. SHAPE-BASED

**NumLevels** (input control) .................................................. integer(-array) \( \sim \) VARIANT (integer)
Number of pyramid levels used in the matching (and lowest pyramid level to use if \(|\text{NumLevels}| = 2\)).
Default Value: 0
List of values: NumLevels \( \in \{0, 1, 2, 4, 5, 6, 7, 8, 9, 10\}\)

**Greediness** (input control) ................................................. real(-array) \( \sim \) VARIANT (real)
“Greediness” of the search heuristic (0: safe but slow; 1: fast but matches may be missed).
Default Value: 0.9
Suggested values: Greediness \( \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}\)
Typical range of values: \(0 \leq \text{Greediness} \leq 0\)
Minimum Increment: 0.01
Recommended Increment: 0.05

**Row** (output control) ....................................................... point.y \( \sim \) VARIANT (real)
Row coordinate of the found instances of the models.

**Column** (output control) .................................................... point.x \( \sim \) VARIANT (real)
Column coordinate of the found instances of the models.

**Angle** (output control) ...................................................... angle.rad \( \sim \) VARIANT (real)
Rotation angle of the found instances of the models.

**Score** (output control) ..................................................... real \( \sim \) VARIANT (real)
Score of the found instances of the models.

**Model** (output control) .................................................... integer \( \sim \) VARIANT (integer)
Index of the found instances of the models.

---

If the parameter values are correct, the operator **FindShapeModels** returns the value TRUE.
If the input is empty (no input images are available) the behavior can be set via **SetSystem**
('noObjectResult', <Result>). If necessary, an exception is raised.

---

**Parallelization Information**

FindShapeModels is reentrant and processed without parallelization.

---

**Possible Predecessors**

AddChannels, CreateShapeModel, ReadShapeModel, SetShapeModelOrigin

---

**Alternatives**

FindScaledShapeModels, FindShapeModel, FindScaledShapeModel, BestMatchRotMg

---

**Module**

Template matching

---

[out] double Row **HShapeModelX.GetShapeModelOrigin**
([out] double Column )

void **HOperatorSetX.GetShapeModelOrigin** ([in] VARIANT ModelID,
[out] VARIANT Row, [out] VARIANT Column )

Return the origin (reference point) of a shape model.
The operator **GetShapeModelOrigin** returns the origin (reference point) of the shape model ModelID. The origin is specified relative to the center of gravity of the domain (region) of the image that was used to create the shape model with CreateShapeModel or CreateScaledShapeModel. Hence, an origin of (0,0) means that the center of gravity of the domain of the model image is used as the origin. An origin of (-20,-40) means that the origin lies to the upper left of the center of gravity.

---

**Parameter**

**ModelID** (input control) .................................................. shape_model \( \sim \) HShapeModelX / VARIANT
Handle of the model.

**Row** (output control) ....................................................... point.y \( \sim \) double / VARIANT
Row coordinate of the origin of the shape model.

**Column** (output control) .................................................. point.x \( \sim \) double / VARIANT
Column coordinate of the origin of the shape model.
CHAPTER 7. MATCHING

If the handle of the model is valid, the operator `GetShapeModelOrigin` returns the value TRUE. If necessary an exception is raised.

---

**Parallelization Information**

`GetShapeModelOrigin` is processed completely exclusively without parallelization.

---

**Possible Predecessors**

CreateShapeModel, CreateScaledShapeModel, ReadShapeModel, SetShapeModelOrigin

---

**Possible Successors**

FindShapeModel, FindScaledShapeModel, FindShapeModels, FindScaledShapeModels

---

See also

AreaCenter

---

**Module**

Template matching

---

Return the parameters of a shape model.

The operator `GetShapeModelParams` returns the parameters of the shape model `ModelID` that were used to create it using CreateShapeModel or CreateScaledShapeModel. In particular, this output can be used to check the parameters NumLevels, AngleStep and ScaleStep if they were determined automatically during the model creation with CreateShapeModel or CreateScaledShapeModel.

---

**Parameter**

- **ModelID** (input control) shape model ~ `HShapeModelIX` VARIANT
  - Handle of the model.

- **NumLevels** (output control) integer ~ `long` VARIANT
  - Number of pyramid levels.

- **AngleStart** (output control) angle.rad ~ `double` VARIANT
  - Smallest rotation of the pattern.

- **AngleExtent** (output control) angle.rad ~ `double` VARIANT
  - Extent of the rotation angles.
  - **Restriction**: \((\text{AngleExtent} \geq 0)\)

- **AngleStep** (output control) angle.rad ~ `double` VARIANT
  - Step length of the angles (resolution).
  - **Restriction**: \((\text{AngleStep} \geq 0)\)

- **ScaleMin** (output control) number ~ `double` VARIANT
  - Minimum scale of the pattern.
  - **Restriction**: \((\text{ScaleMin} > 0)\)

- **ScaleMax** (output control) number ~ `double` VARIANT
  - Maximum scale of the pattern.
  - **Restriction**: \((\text{ScaleMax} \geq \text{ScaleMin})\)

- **ScaleStep** (output control) number ~ `double` VARIANT
  - Scale step length (resolution).
  - **Restriction**: \((\text{ScaleStep} \geq 0)\)

- **Metric** (output control) string ~ `String` VARIANT
  - Match metric.
MinContrast (output control) ............................................. number \rightarrow \text{long / VARIANT}

Minimum contrast of the objects in the search images.

Result

If the handle of the model is valid, the operator \text{GetShapeModelParams} returns the value TRUE. If necessary an exception is raised.

Parallelization Information

\text{GetShapeModelParams} is processed \text{completely exclusively} without parallelization.

Possible Predecessors

\text{CreateShapeModel, CreateScaledShapeModel, ReadShapeModel}

See also

\text{FindShapeModel, FindScaledShapeModel, FindShapeModels, FindScaledShapeModels}

Module

Template matching

Create the representation of a shape model.

\text{InspectShapeModel} creates a representation of a shape model. The operator is particularly useful in order to determine the parameters \text{NumLevels} and \text{Contrast}, which are used in \text{CreateShapeModel}, quickly and conveniently. The representation of the model is created on multiple image pyramid levels, where the number of levels is determined by \text{NumLevels}. In contrast to \text{CreateShapeModel}, the model is only created for the rotation of the object in the input image, i.e., $0^\circ$. As output, \text{InspectShapeModel} creates an image object \text{ModelImages} containing the images of the individual levels of the image pyramid as well as a region in \text{ModelRegions} for each pyramid level that represents the model at the respective pyramid level. The individual objects can be accessed with \text{SelectObj}. As described for \text{CreateShapeModel}, the number of pyramid levels should be chosen as large as possible, while taking into account that the model must be recognizable on the highest pyramid level and must have enough model points. The parameter \text{Contrast} should be chosen such that only the significant features of the template object are used for the model. \text{Contrast} can also contain a tuple with two values. In this case, the model is segmented using a method similar to the hysteresis threshold method used in \text{EdgesImage}. Here, the first element of the tuple determines the lower threshold, while the second element determines the upper threshold. For more information about the hysteresis threshold method, see \text{HysteresisThreshold}. Optionally, \text{Contrast} can contain a third value as the last element of the tuple. This value determines a threshold for the selection of significant model components based on the size of the components, i.e., components that have fewer points than the minimum size thus specified are suppressed. This threshold for the minimum size is divided by two for each successive pyramid level. If small model components should be suppressed, but hysteresis thresholding should not be performed, nevertheless three values must be specified in \text{Contrast}. In this case, the first two values can simply be set to identical values. In its typical use, \text{InspectShapeModel} is called interactively multiple times with different parameters for \text{NumLevels} and \text{Contrast}, until a satisfactory model is obtained. After this, \text{CreateShapeModel} is called with the parameters thus obtained.

Parameter

\begin{itemize}
  \item \text{Image} (input iconic) ............................................. image \rightarrow \text{HImageX / IObjectX (byte)}
  \end{itemize}

\begin{itemize}
  \item \text{ModelImages} (output iconic) .............................. image \rightarrow \text{HImageX / HUntypedObjectX (byte)}
  \end{itemize}

\begin{itemize}
  \item \text{ModelRegions} (output iconic) .............................. region \rightarrow \text{HRegionX / HUntypedObjectX}
  \end{itemize}
CHAPTER 7. MATCHING

- **NumLevels** (input control) .......................... integer  \(\sim\) long / VARIANT
  Number of pyramid levels.
  
  Default Value : 4
  List of values : NumLevels \(\in\) \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}

- **Contrast** (input control) ............................ number(-array)  \(\sim\) VARIANT( integer )
  Threshold or hysteresis thresholds for the contrast of the object in the image and optionally minimum size of
  the object parts.
  
  Default Value : 30
  Suggested values : Contrast \(\in\) \{10, 20, 30, 40, 60, 80, 100, 120, 140, 160\}

If the parameters are valid, the operator **InspectShapeModel** returns the value TRUE. If necessary an exception is raised.

**Parallelization Information**

**InspectShapeModel** is reentrant and processed without parallelization.

**Possible Predecessors**

ReduceDomain

**Possible Successors**

SelectObj

See also **CreateShapeModel**

**Module**

Template matching

```c
void HShapeModelX.ReadShapeModel ([in] String FileName )

void HOperatorSetX.ReadShapeModel ([in] VARIANT FileName, [out] VARIANT ModelID )
```

**Read a shape model from a file.**

The operator **ReadShapeModel** reads a shape model, which has been written with **WriteShapeModel**, from the file **FileName**.

**Parameter**

- **FileName** (input control) .......................... filename  \(\sim\) String / VARIANT
  File name.

- **ModelID** (output control) .......................... shape_model  \(\sim\) HShapeModelX / VARIANT
  Handle of the model.

If the file name is valid, the operator **ReadShapeModel** returns the value TRUE. If necessary an exception is raised.

**Parallelization Information**

**ReadShapeModel** is processed completely exclusively without parallelization.

**Possible Successors**

FindShapeModel, FindScaledShapeModel, FindShapeModels, FindScaledShapeModels

See also **CreateShapeModel, ClearShapeModel**

**Module**

Template matching
7.2. SHAPE-BASED

void HShapeModelX.SetShapeModelOrigin ([in] double Row, [in] double Column )


Set the origin (reference point) of a shape model.

The operator SetShapeModelOrigin sets the origin (reference point) of the shape model ModelID to a new value. The origin is specified relative to the center of gravity of the domain (region) of the image that was used to create the shape model with CreateShapeModel or CreateScaledShapeModel. Hence, an origin of (0,0) means that the center of gravity of the domain of the model image is used as the origin. An origin of (-20,-40) means that the origin lies to the upper left of the center of gravity.

Parameter

- ModelID (input control) shape_model ~ HShapeModelX / VARIANT
  Handle of the model.
- Row (input control) point.y ~ double / VARIANT
  Row coordinate of the origin of the shape model.
- Column (input control) point.x ~ double / VARIANT
  Column coordinate of the origin of the shape model.

Result

If the handle of the model is valid, the operator SetShapeModelOrigin returns the value TRUE. If necessary an exception is raised.

Parallelization Information

SetShapeModelOrigin is processed completely exclusively without parallelization.

Possible Predecessors

CreateShapeModel, CreateScaledShapeModel, ReadShapeModel

Possible Successors

FindShapeModel, FindScaledShapeModel, FindShapeModels, FindScaledShapeModels, GetShapeModelOrigin

See also

AreaCenter

Template matching

Write a shape model to a file.

The operator WriteShapeModel writes a shape model to the file FileName. The model can be read again with ReadShapeModel.

Parameter

- ModelID (input control) shape_model ~ HShapeModelX / VARIANT
  Handle of the model.
- FileName (input control) filename ~ String / VARIANT
  File name.

Result

If the file name is valid (write permission), the operator WriteShapeModel returns the value TRUE. If necessary an exception is raised.

Parallelization Information

WriteShapeModel is reentrant and processed without parallelization.

HALCON 6.1.4
### Possible Predecessors

<table>
<thead>
<tr>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateShapeModel</td>
</tr>
</tbody>
</table>

Template matching
Chapter 8

Morphology

8.1 Gray-Values

Opening, Median and Closing with circle or rectangle mask.

The operator DualRank carries out a non-linear transformation of the gray values of all input images (Image). Circles or squares can be used as structuring elements. The operator DualRank effects two consecutive calls of RankImage. At the first call the range gray value is calculated with the indicated range (ModePercent). The result of this calculation is the input of a further call of RankImage, this time using the range value 100–ModePercent.

When filtering different parameters for border treatment (Margin) can be chosen:

- **gray value** Pixels outside of the image edges are assumed to be constant (with the indicated gray value).
- **'continued'** Continuation of edge pixels.
- **'cyclic'** Cyclic continuation of image edges.
- **'mirrored'** Reflection of pixels at the image edges.

A range filtering is calculated according to the following scheme: The indicated mask is put over the image to be filtered in such a way that the center of the mask touches all pixels once. For each of these pixels all neighboring pixels covered by the mask are sorted in an ascending sequence corresponding to their gray values. Each sorted sequence of gray values contains the same number of gray values like the mask has image points. The n-th highest element, (= ModePercent, rank values between 0...100 in percent) is selected and set as result gray value in the corresponding result image.

If ModePercent is 0, then the operator equals to the gray value opening (GrayOpening). If ModePercent is 50, the operator results in the median filter, which is applied twice (MedianImage). The ModePercent 100 in DualRank means that it calculates the gray value closing (GrayClosing). Choosing parameter values inside this range results in a smooth transformation of these operators.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong></td>
<td>(input iconic) .... (multichannel-)image(-array) ~ HImageX / IObjectX (byte, int2, uint2, int4, real)</td>
</tr>
<tr>
<td><strong>ImageRank</strong></td>
<td>(output iconic) .... multichannel-image-array ~ HImageX / HUntypedObjectX (byte, int2, uint2, int4, real)</td>
</tr>
</tbody>
</table>

427
MaskType (input control)  string  \( \sim String / \text{VARIANT} \)
Shape of the mask.

Default Value : ‘circle’
List of values : \( \text{MaskType} \in \{‘circle’, ‘rectangle’\} \)

Radius (input control)  integer  \( \sim long / \text{VARIANT} \)
Radius of the filter mask.

Default Value : 1
Suggested values : \( \text{Radius} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59\} \)
Typical range of values : \( 1 \leq \text{Radius} \leq 1 \)
Minimum Increment : 1
Recommended Increment : 2

ModePercent (input control)  integer  \( \sim long / \text{VARIANT} \)

Filter Mode: 0 corresponds to a gray value opening, 50 corresponds to a median and 100 to a gray values closing.

Default Value : 10
Suggested values : \( \text{ModePercent} \in \{0, 2, 5, 10, 15, 20, 40, 50, 60, 80, 85, 90, 95, 98, 100\} \)
Typical range of values : \( 0 \leq \text{ModePercent} \leq 0 \)
Minimum Increment : 1
Recommended Increment : 2

Margin (input control)  string  \( \sim \text{VARIANT}(\text{integer, real, string}) \)

Border treatment.

Default Value : ‘mirrored’
Suggested values : \( \text{Margin} \in \{‘mirrored’, ‘cyclic’, ‘continued’, 0, 30, 60, 90, 120, 150, 180, 210, 240, 255\} \)

Example

```plaintext
read_image(Image,’fabrik’)
dual_rank(Image,ImageOpening,’circle’,10,10,’mirrored’)
disp_image(ImageOpening,WindowHandle).
```

Complexity
For each pixel: \( O(\sqrt{F} \times 10) \) with \( F \) = area of the structuring element.

Result
If the parameter values are correct the operator \texttt{DualRank} returns the value TRUE. The behavior in case of empty input (no input images available) is set via the operator \texttt{SetSystem(’noObjectResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information
\texttt{DualRank} is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Possible Predecessors
ReadImage

Possible Successors
Threshold, DynThreshold, SubImage, Regiongrowing

See also
GenCircle, GenRectangle1, GrayErosionRect, GrayDilationRect, SigmaImage

Alternatives
RankImage, GrayClosing, GrayOpening, MedianImage

References

Module
Image filters

HALCON/COM Reference Manual, 2005-2-1
void HImageX.GenDiscSe ([in] long Width, [in] long Height, [in] long Smax )


Generate ellipsoidal structuring elements for gray morphology.

GenDiscSe generates an ellipsoidal structuring element (SE) for gray morphology of images. The parameters Width and Height determine the length of the two major axes of the ellipse. The value of Smax determines the maximum gray value of the structuring element. For the generation of arbitrary structuring elements, see ReadGraySe.

Parameter

SE (output iconic) .......................................................... image ~ HImageX / HUntypedObjectX (byte)
Generated structuring element.

Width (input control) ...................................................... integer ~ long / VARIANT
Width of the structuring element.

Default Value : 5
Suggested values : Width ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}
Typical range of values : 0 ≤ Width ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 1

Height (input control) ...................................................... integer ~ long / VARIANT
Height of the structuring element.

Default Value : 5
Suggested values : Height ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}
Typical range of values : 0 ≤ Height ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 1

Smax (input control) ...................................................... integer ~ long / VARIANT
Maximum gray value of the structuring element.

Default Value : 0
Suggested values : Smax ∈ {0, 1, 2, 5, 10, 20, 30, 40}
Typical range of values : 0 ≤ Smax ≤ 0(lin)
Minimum Increment : 1
Recommended Increment : 1

Result

GenDiscSe returns TRUE if all parameters are correct. If necessary, an exception is raised.

Parallelization Information

GenDiscSe is reentrant and processed without parallelization.

Possible Successors

GrayErosion, GrayDilation, GrayOpening, GrayClosing, GrayTophat, GrayBothat

See also

ReadImage, PaintRegion, PaintGray, CropPart

Alternatives

ReadGraySe

Module

Image filters

[out] HImageX ImageBotHat HImageX.GrayBothat ([in] HImageX SE )


Perform a gray value bottom hat transformation on an image.
GrayBothat applies a gray value bottom hat transformation to the input image Image with the structuring element SE. The gray value bottom hat transformation of an image i with a structuring element s is defined as

\[ \text{bothat}(i, s) = (i \ast s) - i, \]

i.e., the difference of the closing of the image with s and the image (see GrayClosing). For the generation of structuring elements, see ReadGraySe.

\[ \text{GrayBothat}(\text{input iconic}) \rightarrow \text{HImageX / IHObjectX (byte, real)} \]

\[ \text{ImageBotHat}(\text{output iconic}) \rightarrow \text{HImageX / HUntypedObjectX (byte, real)} \]

\[ \text{Result} \]

GrayBothat returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

Parallelization Information

GrayBothat is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

ReadGraySe, GenDiscSe

Possible Successors

Threshold

See also

GrayTophat, TopHat, GrayErosionRect, SubImage

Alternatives

GrayClosing

Module

Image filters


\[
\text{HImageX ImageClosing HImageX.GrayClosing ([in] HImageX SE )}
\]

\[
\]

Perform a gray value closing on an image.

GrayClosing applies a gray value closing to the input image Image with the structuring element SE. The gray value closing of an image i with a structuring element s is defined as

\[ i \ast s = (i \oplus s) \ominus s, \]

i.e., a dilation of the image with s followed by an erosion with s (see GrayDilation and GrayErosion). For the generation of structuring elements, see ReadGraySe.

\[ \text{GrayClosing}(\text{input iconic}) \rightarrow \text{HImageX / IHObjectX (byte, real)} \]

\[ \text{ImageClosing}(\text{output iconic}) \rightarrow \text{HImageX / HUntypedObjectX (byte, real)} \]

\[ \text{Result} \]

GrayClosing returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.
GrayClosing is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

ReadGraySe  See also

Closing, GrayDilation, GrayErosion

Alternatives

DualRank

Module

Image filters

**Parallelization Information**

GrayClosing is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

ReadGraySe  See also

Closing, GrayDilation, GrayErosion

Alternatives

DualRank

Module

Image filters

**Perform a grayvalue closing with a selected mask.**

GrayClosingShape applies a gray value closing to the input image Image with the structuring element of shape MaskShape. The mask’s offset values are 0 and its horizontal and vertical size is defined by MaskHeight and MaskWidth. The resulting image is returned in ImageClosing. If the parameters MaskHeight or MaskWidth are even, they are changed to the next larger odd value. In case of the values ‘rhombus’ and ‘octagon’ for the MaskShape control parameter, MaskHeight and MaskWidth must be equal. The parameter value ‘octagon’ for MaskShape denotes an equilateral octagonal mask which is a suitable approximation for a circular structure. At the border of the image the gray values are mirrored.

The gray value closing of an image $i$ with a structuring element $s$ is defined as

$$ i \ast s = (i \oplus s) \ominus s , $$

i.e., a dilation of the image with $s$ followed by an erosion with $s$ (see GrayDilationShape and GrayErosionShape).

**Parameter**

- **Image** (input iconic) ............................... image(-array) $\sim$ HImageX / IObjectX (byte)
  Image for which the minimum gray values are to be calculated.

- **ImageClosing** (output iconic) ....................... image(-array) $\sim$ HImageX / HUntypedObjectX (byte)
  Image containing the minimum gray values.

- **MaskHeight** (input control) .............................. extent.y $\sim$ long / VARIANT
  Height of the filter mask.
  Default Value : 11
  Suggested values : MaskHeight $\in \{3, 5, 7, 9, 11, 13, 15\}$
  Typical range of values : $3 \leq$ MaskHeight $\leq 3$
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : odd

- **MaskWidth** (input control) .............................. extent.x $\sim$ long / VARIANT
  Width of the filter mask.
  Default Value : 11
  Suggested values : MaskWidth $\in \{3, 5, 7, 9, 11, 13, 15\}$
  Typical range of values : $3 \leq$ MaskWidth $\leq 3$
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : odd
CHAPTER 8. MORPHOLOGY

- **MaskShape** (input control) ................................................. string ~ String / VARIANT
  Shape of the mask.
  Default Value : ‘octagon’
  List of values : MaskShape ∈ {‘rectangle’, ‘rhombus’, ‘octagon’}

GrayClosingShape returns TRUE if all parameters are correct.

GrayClosingShape is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also
GrayDilationShape, GrayErosionShape, Closing

Alternatives
GrayClosing

Module

Image filters

```haskell
[out] HImageX ImageDilation HImageX.GrayDilation ([in] HImageX SE )
```

Perform a gray value dilation on an image.

GrayDilation applies a gray value dilation to the input image Image with the structuring element SE. The gray value dilation of an image \( i \) with a structuring element \( s \) at the pixel position \( x \) is defined as:

\[
(i \oplus s)(x) = \max\{ f(x - z) + s(z) | z \in S \}
\]

Here, \( S \) is the domain of the structuring element \( s \), i.e., the pixels \( z \) where \( s(z) > 0 \) (see ReadGraySe).

GrayDilation returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

GrayDilation is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
ReadGraySe

Possible Successors
SubImage, GrayErosion

See also
GrayOpening, GrayClosing, Dilation1, GraySkeleton

Alternatives
GrayDilationRect

Module

Image filters
### 8.1. GRAY-VALUES

Determine the maximum gray value within a rectangle.

**GrayDilationRect** calculates the maximum gray value of the input image *Image* within a rectangular mask of size (*MaskHeight*, *MaskWidth*) for each image point. The resulting image is returned in *ImageMax*. If the parameters *MaskHeight* or *MaskWidth* are even, they are changed to the next larger odd value. At the border of the image the gray values are mirrored.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic) ............................................. extent.y (\sim) long / VARIANT</td>
</tr>
<tr>
<td>Image for which the maximum gray values are to be calculated.</td>
</tr>
<tr>
<td><strong>ImageMax</strong> (output iconic) ...................................... extent.y (\sim) HImageX / HUntypedObjectX</td>
</tr>
<tr>
<td>Image containing the maximum gray values.</td>
</tr>
<tr>
<td><strong>MaskHeight</strong> (input control) ...................................... extent.x (\sim) long / VARIANT</td>
</tr>
<tr>
<td>Height of the filter mask.</td>
</tr>
<tr>
<td>Default Value: 11</td>
</tr>
<tr>
<td>Suggested values: <em>MaskHeight</em> (\in) {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>Typical range of values: (3 \leq \text{MaskHeight} \leq 3)</td>
</tr>
<tr>
<td>Minimum Increment: 2</td>
</tr>
<tr>
<td>Recommended Increment: 2</td>
</tr>
<tr>
<td>Restriction: odd</td>
</tr>
<tr>
<td><strong>MaskWidth</strong> (input control) ...................................... extent.x (\sim) long / VARIANT</td>
</tr>
<tr>
<td>Width of the filter mask.</td>
</tr>
<tr>
<td>Default Value: 11</td>
</tr>
<tr>
<td>Suggested values: <em>MaskWidth</em> (\in) {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>Typical range of values: (3 \leq \text{MaskWidth} \leq 3)</td>
</tr>
<tr>
<td>Minimum Increment: 2</td>
</tr>
<tr>
<td>Recommended Increment: 2</td>
</tr>
<tr>
<td>Restriction: odd</td>
</tr>
</tbody>
</table>

**Result**

**GrayDilationRect** returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult', <Result>)`. If necessary, an exception is raised.

**Parallelization Information**

**GrayDilationRect** is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

**Image filters**

Determine the maximum gray value within a selected mask.

**GrayDilationShape** calculates the maximum gray value of the input image *Image* within a mask of shape *MaskShape*, vertical size *MaskHeight* and horizontal size *MaskWidth* for each image point. The resulting image is returned in *ImageMax*. If the parameters *MaskHeight* or *MaskWidth* are even, they are changed to the next larger odd value. At the border of the image the gray values are mirrored.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic) ............................................. extent.y (\sim) HImageX / IObjectX (byte, direction, cyclic, int2, int4, real)</td>
</tr>
<tr>
<td>Image for which the maximum gray values are to be calculated.</td>
</tr>
<tr>
<td><strong>ImageMax</strong> (output iconic) ...................................... HImageX / HUntypedObjectX (byte, direction, cyclic, int2, int4, real)</td>
</tr>
<tr>
<td>Image containing the maximum gray values.</td>
</tr>
<tr>
<td><strong>MaskHeight</strong> (input control) ...................................... extent.x (\sim) long / VARIANT</td>
</tr>
<tr>
<td>Height of the filter mask.</td>
</tr>
<tr>
<td>Default Value: 11</td>
</tr>
<tr>
<td>Suggested values: <em>MaskHeight</em> (\in) {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>Typical range of values: (3 \leq \text{MaskHeight} \leq 3)</td>
</tr>
<tr>
<td>Minimum Increment: 2</td>
</tr>
<tr>
<td>Recommended Increment: 2</td>
</tr>
<tr>
<td>Restriction: odd</td>
</tr>
<tr>
<td><strong>MaskWidth</strong> (input control) ...................................... extent.x (\sim) long / VARIANT</td>
</tr>
<tr>
<td>Width of the filter mask.</td>
</tr>
<tr>
<td>Default Value: 11</td>
</tr>
<tr>
<td>Suggested values: <em>MaskWidth</em> (\in) {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>Typical range of values: (3 \leq \text{MaskWidth} \leq 3)</td>
</tr>
<tr>
<td>Minimum Increment: 2</td>
</tr>
<tr>
<td>Recommended Increment: 2</td>
</tr>
<tr>
<td>Restriction: odd</td>
</tr>
</tbody>
</table>

**Result**

**GrayDilationShape** returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult', <Result>)`. If necessary, an exception is raised.

**Parallelization Information**

**GrayDilationShape** is reentrant and automatically parallelized (on tuple level, channel level, domain level).
the next larger odd value. In case of the values 'rhombus' und 'octagon' for the MaskShape control parameter, MaskHeight and MaskWidth must be equal. The parameter value 'octagon' for MaskShape denotes an equilateral octagonal mask which is a suitable approximation for a circular structure. At the border of the image the gray values are mirrored.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic) .............................. image(-array) → HImageX / IObjectX (byte)</td>
</tr>
<tr>
<td><strong>ImageMax</strong> (output iconic) ......................... image(-array) → HImageX / HUntypedObjectX (byte)</td>
</tr>
<tr>
<td><strong>MaskHeight</strong> (input control) ................................ extent.y → long / VARIANT</td>
</tr>
<tr>
<td>Default Value : 11</td>
</tr>
<tr>
<td>Suggested values : MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>Minimum Increment : 2</td>
</tr>
<tr>
<td>Recommended Increment : 2</td>
</tr>
<tr>
<td>Restriction : odd</td>
</tr>
<tr>
<td><strong>MaskWidth</strong> (input control) ................................ extent.x → long / VARIANT</td>
</tr>
<tr>
<td>Default Value : 11</td>
</tr>
<tr>
<td>Suggested values : MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}</td>
</tr>
<tr>
<td>Minimum Increment : 2</td>
</tr>
<tr>
<td>Recommended Increment : 2</td>
</tr>
<tr>
<td>Restriction : odd</td>
</tr>
<tr>
<td><strong>MaskShape</strong> (input control) ................................ string → String / VARIANT</td>
</tr>
<tr>
<td>Default Value : 'octagon'</td>
</tr>
<tr>
<td>List of values : MaskShape ∈ {'rectangle', 'rhombus', 'octagon'}</td>
</tr>
</tbody>
</table>

GrayDilationShape returns TRUE if all parameters are correct. GrayDilationShape is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

GrayOpeningShape, GrayClosingShape, GraySkeleton

Alternatives

GrayDilation, GrayDilationRect

Module

Image filters

```
void HOperatorSetX.GrayErosion ([in] IObjectX Image,
```

Perform a gray value erosion on an image.

GrayErosion applies a gray value erosion to the input image Image with the structuring element SE. The gray value erosion of an image i with a structuring element s at the pixel position x is defined as:

\[
(i \ominus s)(x) = \min\{f(x + z) - s(z) | z \in S\}
\]

Here, S is the domain of the structuring element s, i.e., the pixels z where s(z) > 0 (see ReadGraySe).
8.1. GRAY-VALUES

Parameter

- **Image** (input iconic) ................. image(-array) \( \rightarrow \) HimageX / HObjectX (byte, real)
  Input image.
- **SE** (input iconic) ............................. image \( \rightarrow \) HimageX / HObjectX (byte)
  Structuring element.
- **ImageErosion** (output iconic) ............ image(-array) \( \rightarrow \) HimageX / HUntypedObjectX (byte, real)
  Gray-eroded image.

Result

GrayErosion returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

Parallelization Information

GrayErosion is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

ReadGraySe
GrayDilation, SubImage

Possible Successors

See also
GrayOpening, GrayClosing, Erosion1, GraySkeleton

Alternatives

GrayErosionRect

Module

Image filters

```plaintext

```

Determine the minimum gray value within a rectangle.

GrayErosionRect calculates the minimum gray value of the input image Image within a rectangular mask of size (MaskHeight, MaskWidth) for each image point. The resulting image is returned in ImageMin. If the parameters MaskHeight or MaskWidth are even, they are changed to the next larger odd value. At the border of the image the gray values are mirrored.

Parameter

- **Image** (input iconic) ................. image(-array) \( \rightarrow \) HimageX / HObjectX (byte, direction, cyclic, int2, int4, real)
  Image for which the minimum gray values are to be calculated.
- **ImageMin** (output iconic) .............. image(-array) \( \rightarrow \) HimageX / HUntypedObjectX (byte, direction, cyclic, int2, int4, real)
  Image containing the minimum gray values.
- **MaskHeight** (input control) ............... extent.y \( \rightarrow \) long / VARIANT
  Height of the filter mask.
  Default Value : 11
  Suggested values : MaskHeight \( \in \) \{3, 5, 7, 9, 11, 13, 15\}
  Typical range of values : 3 \( \leq \) MaskHeight \( \leq \) 3(lin)
  Minimum Increment : 2
  Recommended Increment : 2
  Restriction : odd
```
CHAPTER 8. MORPHOLOGY

**MaskWidth** (input control) .................................................. extent.x  \(\rightarrow\) long / VARIANT

Width of the filter mask.

**Default Value:** 11

**Suggested values:** \(\text{MaskWidth} \in \{3, 5, 7, 9, 11, 13, 15\}\)

**Typical range of values:** \(3 \leq \text{MaskWidth} \leq 3\) (lin)

**Minimum Increment:** 2

**Recommended Increment:** 2

**Restriction:** odd

---

**Result**

**Parallelization Information**

GrayErosionRect returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception is raised.

GrayErosionRect is reentrant and automatically parallelized (on tuple level, channel level, domain level).

See also

GrayDilationRect

---

Image filters

---

<table>
<thead>
<tr>
<th>out</th>
<th>HImageX ImageMin</th>
<th>HImageX.GrayErosionShape ([in] long MaskHeight, [in] long MaskWidth, [in] String MaskShape)</th>
</tr>
</thead>
</table>

Determine the minimum gray value within a selected mask.

GrayDilationShape calculates the minimum gray value of the input image `Image` within a mask of shape `MaskShape`, vertical size `MaskHeight` and horizontal size `MaskWidth` for each image point. The resulting image is returned in `ImageMin`. If the parameters `MaskHeight` or `MaskWidth` are even, they are changed to the next larger odd value. In case of the values ’rhombus’ and ’octagon’ for the `MaskShape` control parameter, `MaskHeight` and `MaskWidth` must be equal. The parameter value ’octagon’ for `MaskShape` denotes an equilateral octagonal mask which is a suitable approximation for a circular structure. At the border of the image the gray values are mirrored.

---

**Parameter**

| Image (input iconic) .................................................. image(-array)  \(\rightarrow\) HImageX / IObjectX (byte) |
|-----------------|--------------------------------------------------------------------------------------------------|

Image for which the minimum gray values are to be calculated.

| ImageMin (output iconic) .................................................. image(-array)  \(\rightarrow\) HImageX / HUntypedObjectX (byte) |
|-----------------|--------------------------------------------------------------------------------------------------|

Image containing the minimum gray values.

| MaskHeight (input control) .................................................. extent.y  \(\rightarrow\) long / VARIANT |
|-----------------|--------------------------------------------------------------------------------------------------|

Height of the filter mask.

**Default Value:** 11

**Suggested values:** \(\text{MaskHeight} \in \{3, 5, 7, 9, 11, 13, 15\}\)

**Minimum Increment:** 2

**Recommended Increment:** 2

**Restriction:** odd

| MaskWidth (input control) .................................................. extent.x  \(\rightarrow\) long / VARIANT |
|-----------------|--------------------------------------------------------------------------------------------------|

Width of the filter mask.

**Default Value:** 11

**Suggested values:** \(\text{MaskWidth} \in \{3, 5, 7, 9, 11, 13, 15\}\)

**Minimum Increment:** 2

**Recommended Increment:** 2

**Restriction:** odd

HALCON/COM Reference Manual, 2005-2-1
8.1. GRAY-VALUES

**MaskShape** (input control) ................................. string  \( \sim \) String / VARIANT
Shape of the mask.

Default Value: ‘octagon’
List of values: \( \text{MaskShape} \in \{ \text{rectangle'}, \text{rhombus'}, \text{octagon}' \}

\[ \text{GrayDilationShape} \text{ returns TRUE if all parameters are correct.} \]

\[ \text{Parallelization Information} \]
\[ \text{GrayErosionShape} \text{ is reentrant and automatically parallelized (on tuple level, channel level, domain level).} \]

\[ \text{See also} \]
GrayOpeningShape, GrayClosingShape, GraySkeleton

\[ \text{Alternatives} \]
GrayErosion, GrayErosionRect

\[ \text{Module} \]

Image filters

```
[out] HImageX ImageOpening  HImageX.GrayOpening ([in] HImageX SE )
void HOperatorSetX.GrayOpening ([in] IObjectX Image,
[in] IObjectX SE, [out] HUntypedObjectX ImageOpening )
```

Perform a gray value opening on an image.

GrayOpening applies a gray value opening to the input image Image with the structuring element SE. The gray value opening of an image \( i \) with a structuring element \( s \) is defined as

\[ i \circ s = (i \ominus s) \oplus s , \]

i.e., an erosion of the image with \( s \) followed by a dilation with \( s \) (see GrayErosion and GrayDilation). For the generation of structuring elements, see ReadGraySe.

\[ \text{Result} \]

GrayOpening returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

\[ \text{Parallelization Information} \]
GrayOpening is reentrant and automatically parallelized (on tuple level).

\[ \text{Possible Predecessors} \]
ReadGraySe

\[ \text{See also} \]
Opening, GrayDilation, GrayErosion

\[ \text{Alternatives} \]
DualRank

\[ \text{Module} \]

Image filters
Perform a gray value opening with a selected mask.

GrayOpeningShape applies a gray value opening to the input image Image with the structuring element of shape MaskShape. The mask’s offset values are 0 and its horizontal and vertical size is defined by MaskHeight and MaskWidth. The resulting image is returned in ImageOpening. If the parameters MaskHeight or MaskWidth are even, they are changed to the next larger odd value. In case of the values ‘rhombus’ and ‘octagon’ for the MaskShape control parameter, MaskHeight and MaskWidth must be equal. The parameter value ‘octagon’ for MaskShape denotes an equilateral octagonal mask which is a suitable approximation for a circular structure. At the border of the image the gray values are mirrored.

The gray value opening of an image \( i \) with a structuring element \( s \) is defined as

\[
i \circ s = (i \ominus s) \oplus s,
\]

i.e., an erosion of the image with \( s \) followed by a dilation with \( s \) (see GrayErosionShape and GrayDilationShape).

### Parameter

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Image} ) (input iconic)</td>
<td>Image(-array) ( \sim ) HImageX / IHObjectX (byte) Image for which the minimum gray values are to be calculated.</td>
</tr>
<tr>
<td>( \text{ImageOpening} ) (output iconic)</td>
<td>Image(-array) ( \sim ) HImageX / HUntypedObjectX (byte) Image containing the minimum gray values.</td>
</tr>
<tr>
<td>( \text{MaskHeight} ) (input control)</td>
<td>extent.y ( \sim ) long / VARIANT Height of the filter mask.</td>
</tr>
<tr>
<td>Default Value</td>
<td>11</td>
</tr>
<tr>
<td>Suggested values</td>
<td>( \text{MaskHeight} \in {3, 5, 7, 9, 11, 13, 15} )</td>
</tr>
<tr>
<td>Typical range of values</td>
<td>( 3 \leq \text{MaskHeight} \leq 3 )</td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>2</td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>2</td>
</tr>
<tr>
<td>Restriction</td>
<td>odd</td>
</tr>
<tr>
<td>( \text{MaskWidth} ) (input control)</td>
<td>extent.x ( \sim ) long / VARIANT Width of the filter mask.</td>
</tr>
<tr>
<td>Default Value</td>
<td>11</td>
</tr>
<tr>
<td>Suggested values</td>
<td>( \text{MaskWidth} \in {3, 5, 7, 9, 11, 13, 15} )</td>
</tr>
<tr>
<td>Typical range of values</td>
<td>( 3 \leq \text{MaskWidth} \leq 3 )</td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>2</td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>2</td>
</tr>
<tr>
<td>Restriction</td>
<td>odd</td>
</tr>
<tr>
<td>( \text{MaskShape} ) (input control)</td>
<td>string ( \sim ) String / VARIANT Shape of the mask.</td>
</tr>
<tr>
<td>Default Value</td>
<td>‘octagon’</td>
</tr>
<tr>
<td>List of values</td>
<td>( \text{MaskShape} \in {\text{‘rectangle’}, \text{‘rhombus’}, \text{‘octagon’}} )</td>
</tr>
</tbody>
</table>

### Result

GrayOpeningShape returns TRUE if all parameters are correct.

### Parallelization Information

GrayOpeningShape is reentrant and automatically parallelized (on tuple level, channel level, domain level).

### See also

GrayDilationShape, GrayErosionShape, Opening

### Alternatives

GrayOpening

Image filters

HALCON/COM Reference Manual, 2005-2-1
Determine the gray value range within a rectangle.

`GrayRangeRect` calculates the gray value range, i.e., the difference $(\text{max} - \text{min})$ of the maximum and minimum gray values, of the input image `Image` within a rectangular mask of size $(\text{MaskHeight}, \text{MaskWidth})$ for each image point. The resulting image is returned in `ImageResult`. If the parameters `MaskHeight` or `MaskWidth` are even, they are changed to the next larger odd value. At the border of the image the gray values are mirrored.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image</code></td>
<td>(input iconic) Image(-array)</td>
</tr>
<tr>
<td><code>ImageResult</code></td>
<td>(output iconic) Image(-array)</td>
</tr>
<tr>
<td><code>MaskHeight</code></td>
<td>(input control)</td>
</tr>
<tr>
<td>Default Value:</td>
<td>11</td>
</tr>
<tr>
<td>Suggested values:</td>
<td>${3, 5, 7, 9, 11, 13, 15}$</td>
</tr>
<tr>
<td>Typical range of values:</td>
<td>$3 \leq \text{MaskHeight} \leq 3($lin$)$</td>
</tr>
<tr>
<td>Minimum Increment:</td>
<td>2</td>
</tr>
<tr>
<td>Recommended Increment:</td>
<td>2</td>
</tr>
<tr>
<td>Restriction:</td>
<td><code>odd</code></td>
</tr>
<tr>
<td><code>MaskWidth</code></td>
<td>(input control)</td>
</tr>
<tr>
<td>Default Value:</td>
<td>11</td>
</tr>
<tr>
<td>Suggested values:</td>
<td>${3, 5, 7, 9, 11, 13, 15}$</td>
</tr>
<tr>
<td>Typical range of values:</td>
<td>$3 \leq \text{MaskWidth} \leq 3($lin$)$</td>
</tr>
<tr>
<td>Minimum Increment:</td>
<td>2</td>
</tr>
<tr>
<td>Recommended Increment:</td>
<td>2</td>
</tr>
<tr>
<td>Restriction:</td>
<td><code>odd</code></td>
</tr>
</tbody>
</table>

`GrayRangeRect` returns TRUE if all parameters are correct. If the input is empty the behaviour can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception is raised.

Parallelization Information

`GrayRangeRect` is reentrant and automatically parallelized (on tuple level, channel level, domain level).

Alternatives

`GrayDilationRect`, `GrayErosionRect`, `SubImage`

Module

Image filters

Perform a gray value top hat transformation on an image.

`GrayTopHat` applies a gray value top hat transformation to the input image `Image` with the structuring element `SE`. The gray value top hat transformation of an image $i$ with a structuring element $s$ is defined as

$$\text{tophat}(i, s) = i - (i \circ s),$$
i.e., the difference of the image and its opening with $s$ (see \texttt{GrayOpening}). For the generation of structuring elements, see \texttt{ReadGraySe}.

\begin{tabular}{ll}
\hline
\textbf{Image} (input iconic) & image(-array) ~ $HImageX / HObjectX$ (byte, real) \\
\textbf{SE} (input iconic) & image ~ $HImageX / HObjectX$ (byte) \\
\textbf{ImageTopHat} (output iconic) & image(-array) ~ $HImageX / HUntypedObjectX$ (byte, real)
\end{tabular}

\texttt{GrayTopHat} returns TRUE if the structuring element is not the empty region. Otherwise, an exception is raised.

\textbf{Parallelization Information} \texttt{GrayTopHat} is reentrant and automatically parallelized (on tuple level).

\begin{tabular}{ll}
\hline
\textbf{Possible Predecessors} & \texttt{ReadGraySe, GenDiscSe} \\
\textbf{Possible Successors} & \texttt{Threshold} \\
\textbf{See also} & \texttt{GrayBothat, TopHat, GrayErosionRect, SubImage} \\
\textbf{Alternatives} & \texttt{GrayOpening} \\
\textbf{Module} & \texttt{Image filters}
\end{tabular}

\textbf{Load a structuring element for gray morphology.} \texttt{ReadGraySe} loads a structuring element for gray morphology from a file. The file names of these structuring elements must end in `.gse' (for gray-scale structuring element). This suffix is automatically appended by \texttt{ReadGraySe} to the passed file name, and thus must not be passed. The structuring element’s data must be contained in the file in the following format: The first two numbers in the file determine the width and height of the structuring element, and determine a rectangle enclosing the structuring element. Both values must be greater than 0. Then, Width*Height integer numbers follow, with the following interpretation: Values smaller than 0 are regarded as not belonging to the region of the structuring element, i.e., they are not considered in morphological operations. This allows the creation of irregularly shaped, not connected structuring elements. All other values are regarded as the corresponding values for gray morphology. Structuring elements are stored internally as byte-images, with negative values being mapped to 0, and all other values increased by 1. Thus, normal byte-images can also be used as structuring elements. However, care should be taken not to use too large images, since the runtime is proportional to the area of the image times the area of the structuring element.

\begin{tabular}{ll}
\hline
\textbf{Parameter} & \\
\hline
\textbf{SE} (output iconic) & image ~ $HImageX / HUntypedObjectX$ (byte) \\
\textbf{FileName} (input control) & filename ~ String / VARIANT
\end{tabular}

\texttt{ReadGraySe} returns TRUE if all parameters are correct. If the file cannot be opened, FAIL is returned. Otherwise, an exception is raised.

\textbf{Parallelization Information} \texttt{ReadGraySe} is reentrant and processed without parallelization.
8.2 Region

Compute the bottom hat of regions.

**BottomHat** computes the **Closing** of **Region** with **StructElement**. The difference between the result of the closing and the original region is called the bottom hat. In contrast to **Closing**, which merges regions under certain circumstances, **BottomHat** computes the regions generated by such a merge.

The position of **StructElement** is meaningless, since a closing operation is invariant with respect to the choice of the reference point.

Structuring elements (**StructElement**) can be generated with operators such as **GenCircle**, **GenRectangle1**, **GenRectangle2**, **GenEllipse**, **DrawRegion**, **GenRegionPolygon**, **GenRegionPoints**, etc.

**Parameter**

- **Region** (input iconic) . . . . . . . . . . . . . . . . . . . . region(-array) \(\rightarrow\) **HRegionX / IHObjectX**
  Regions to be processed.
- **StructElement** (input iconic) . . . . . . . . . . . . . . region \(\rightarrow\) **HRegionX / IHObjectX**
  Structuring element (position independent).
- **RegionBottomHat** (output iconic) . . . . . . . . . . . . . . . . . . region(-array) \(\rightarrow\) **HRegionX / HUntypedObjectX**
  Result of the bottom hat operator.

**Example**

```halcon
read_image (Image,’/bilder/name.ext’)
threshold (Image,Regions,128,255)
gen_circle (Circle,0,0,16)
bottom_hat (Regions,Circle,RegionBottomHat).
```

**Result**

**BottomHat** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: **SetSystem(’noObjectResult’,<RegionResult>)**
- empty region: **SetSystem(’emptyRegionResult’,<RegionResult>)**

Otherwise, an exception is raised.

**Parallelization Information**

**BottomHat** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle,
Reduction of a region to its boundary.

**Boundary** computes the boundary of a region by using morphological operations. The parameter **BoundaryType** determines the type of boundary to compute:

- 'inner', 'inner filled' and 'outer'.

**Boundary** computes the contour of each input region. The resulting regions consist only of the minimal border of the input regions. If **BoundaryType** is set to 'inner', the contour lies within the original region, if it is set to 'outer', it is one pixel outside of the original region. If **BoundaryType** is set to 'inner filled', holes in the interior of the input region are suppressed.

**Parameter**

- **Region** (input iconic) region(-array) \( \rightarrow \) HRegionX / IHObjectX
  
  Regions for which the boundary is to be computed.

- **RegionBorder** (output iconic) region(-array) \( \rightarrow \) HRegionX / HUntypedObjectX
  
  Resulting boundaries.

- **BoundaryType** (input control) string \( \rightarrow \) String / VARIANT
  
  Boundary type.

  **Default Value**: 'inner'

  **List of values**: BoundaryType \( \in \) {'inner', 'outer', 'inner_filled'}

**Complexity**

Let \( F \) be the area of the input region. Then the runtime complexity for one region is

\[
O(3\sqrt{F}) \,.
\]

**Result**

**Boundary** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: \texttt{SetSystem(’noObjectResult’,<RegionResult>)}
- empty region: \texttt{SetSystem(’emptyRegionResult’,<RegionResult>)}

Otherwise, an exception is raised.

**Parallelization Information**

**Boundary** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection
8.2. REGION

See also

FillUp

Alternatives

DilationCircle, ErosionCircle, Difference

Module

Morphology

[out] HRegionX RegionClosing HRegionX.Closing
([in] HRegionX StructElement)

void HOperatorSetX.Closing ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionClosing)

Close a region.

A **Closing** operation is defined as a dilation followed by a Minkowski subtraction. By applying **Closing**
to a region, larger structures remain mostly intact, while small gaps between adjacent regions and holes smaller
than **StructElement** are closed, and the regions’ boundaries are smoothed. All **Closing** variants share the
property that separate regions are not merged, but remain separate objects. The position of **StructElement** is
meaningless, since a closing operation is invariant with respect to the choice of the reference point.

Structuring elements (**StructElement**) can be generated with operators such as **GenCircle**, **GenRectangle1**, **GenRectangle2**, **GenEllipse**, **DrawRegion**, **GenRegionPolygon**, **GenRegionPoints**, etc.

**Attention**

**Closing** is applied to each input region separately. If gaps between different regions are to be closed, **Union1**
or **Union2** has to be called first.

**Parameter**

- **Region** (input iconic) .............................. region(-array) \(\sim\) HRegionX / IHObjectX
  Regions to be closed.
- **StructElement** (input iconic) ..................... region \(\sim\) HRegionX / IHObjectX
  Structuring element (position-invariant).
- **RegionClosing** (output iconic) ..................... region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Closed regions.

**Complexity**

Let \(F_1\) be the area of the input region, and \(F_2\) be the area of the structuring element. Then the runtime complexity
for one region is:

\[O(2 \cdot \sqrt{F_1} \cdot \sqrt{F_2})\]

**Result**

**Closing** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set
via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

**Closing** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection
Close a region with a circular structuring element.

ClosingCircle behaves analogously to Closing, i.e., the regions’ boundaries are smoothed and holes within a region which are smaller than the circular structuring element of radius \( \text{Radius} \) are closed. The ClosingCircle operation is defined as a dilation followed by a Minkowski subtraction, both with the same circular structuring element.

**Attention**

ClosingCircle is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.

**Parameter**

- **Region** (input iconic) region(-array) \( \sim \) HRegionX / IHObjectX
  - Regions to be closed.
- **RegionClosing** (output iconic) region(-array) \( \sim \) HRegionX / HUntypedObjectX
  - Closed regions.
- **Radius** (input control) real \( \sim \) VARIANT( real, integer )
  - Radius of the circular structuring element.
  - Default Value: 3.5
  - Suggested values: \( \text{Radius} \in \{1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5\} \)
  - Typical range of values: \( 0.5 \leq \text{Radius} \leq 0.5 \) (lin)
  - Minimum Increment: 1.0
  - Recommended Increment: 1.0

**Complexity**

Let \( F_1 \) be the area of the input region. Then the runtime complexity for one region is:

\[
O(4 \cdot \sqrt{F_1} \cdot \text{Radius})
\]

**Result**

ClosingCircle returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

ClosingCircle is reentrant and automatically parallelized (on tuple level).
8.2. REGION

Alternatives

RankRegion, FillUp, Closing, ClosingCircle, ClosingGolay

Module

Morphology

[out] HRegionX RegionClosing HRegionX.ClosingGolay

([in] String GolayElement, [in] long Rotation)

void HOperatorSetX.ClosingGolay ([in] IHObjectX Region,

[out] HUntypedObjectX RegionClosing, [in] VARIANT GolayElement,

[in] VARIANT Rotation)

Close a region with an element from the Golay alphabet.

ClosingGolay is defined as a Minkowski addition followed by a Minkowski subtraction. First the Minkowski addition of the input region (Region) with the structuring element from the Golay alphabet defined by GolayElement and Rotation is computed. Then the Minkowski subtraction of the result and the structuring element rotated by 180° is performed.

The following structuring elements are available:

'1', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

ClosingGolay serves to close holes smaller than the structuring element, and to smooth regions’ boundaries.

Attention

Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

Parameter

– Region (input iconic) ........................................................... region(-array)  HRegionX / IHObjectX

Regions to be closed.

– RegionClosing (output iconic) ................................. region(-array)  HRegionX / HUntypedObjectX

Closed regions.

– GolayElement (input control) ............................................ string  String / VARIANT

Structuring element from the Golay alphabet.

Default Value: 'h'

List of values: GolayElement ∈ {'1', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

– Rotation (input control) ................................................... integer  long / VARIANT

Rotation of the Golay element. Depending on the element, not all rotations are valid.

Default Value: 0

List of values: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Complexity

Let F be the area of an input region. Then the runtime complexity for one region is:

\[ O(6 \cdot \sqrt{F}) \].

Result

ClosingGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

• no region: SetSystem(‘noObjectResult’,<RegionResult>)

• empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

ClosingGolay is reentrant and automatically parallelized (on tuple level).

HALCON 6.1.4
Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

See also

ErosionGolay, DilationGolay, OpeningGolay, HitOrMissGolay, ThinningGolay, ThickeningGolay, GolayElements

Alternatives

Closing

Module

Morphology

```c
[out] HRegionX RegionClosing HRegionX.ClosingRectangle1
([in] VARIANT Width, [in] VARIANT Height )

void HOperatorSetX.ClosingRectangle1 ([in] IHObjectX Region,
[out] HUntypedObjectX RegionClosing, [in] VARIANT Width,
[in] VARIANT Height )
```

Close a region with a rectangular structuring element.

ClosingRectangle1 performs a DilationRectangle1 followed by an ErosionRectangle1 on the input region Region. The size of the rectangular structuring element is determined by the parameters Width and Height. As is the case for all Closing variants, regions’ boundaries are smoothed and holes within a region which are smaller than the rectangular structuring element are closed.

**Attention**

ClosingRectangle1 is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.

**Parameter**

- **Region** (input iconic) region(-array) ➞ HRegionX / IHObjectX
  Regions to be closed.
- **RegionClosing** (output iconic) region(-array) ➞ HRegionX / HUntypedObjectX
  Closed regions.
- **Width** (input control) extent.x ➞ VARIANT ( integer, real )
  Width of the structuring rectangle.
  Default Value : 10
  Suggested values : Width ∈ { 1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200 }
  Typical range of values : 1 ≤ Width ≤ 1(lin)
  Minimum Increment : 1
  Recommended Increment : 1
- **Height** (input control) extent.y ➞ VARIANT ( integer, real )
  Height of the structuring rectangle.
  Default Value : 10
  Suggested values : Height ∈ { 1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200 }
  Typical range of values : 1 ≤ Height ≤ 1(lin)
  Minimum Increment : 1
  Recommended Increment : 1

**Complexity**

Let $F$ be the area of an input region and $H$ be the height of the rectangle. Then the runtime complexity for one region is:

$$O(2 \cdot \sqrt{F} \cdot \log_2(H))$$

**Result**

ClosingRectangle1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:
8.2. REGION

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`

Otherwise, an exception is raised.

---

**Parallelization Information**

`ClosingRectangle1` is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

---

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

---

**See also**

DilationRectangle1, ErosionRectangle1, OpeningRectangle1, GenRectangle1

---

**Alternatives**

Closing

---

**Module**

Morphology

---

```c
[out] HRegionX RegionDilation HRegionX.Dilation1
([in] HRegionX StructElement, [in] long Iterations )

void HOperatorSetX.Dilation1 ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionDilation,
[in] VARIANT Iterations )
```

---

**Dilate a region.**

`Dilation1` dilates the input regions with a structuring element. By applying `Dilation1` to a region, its boundary gets smoothed. In the process, the area of the region is enlarged. Furthermore, disconnected regions may be merged. Such regions, however, remain logically distinct region. The dilation is a set-theoretic region operation. It uses the union operation.

Let \( M (\text{StructElement}) \) and \( R (\text{Region}) \) be two regions, where \( M \) is the structuring element and \( R \) is the region to be processed. Furthermore, let \( m \) be a point in \( M \). Then the displacement vector \( \vec{v}_m = (dx, dy) \) is defined as the difference of the center of gravity of \( M \) and the vector \( \vec{m} \). Let \( t_{\vec{v}_m}(R) \) denote the translation of a region \( R \) by a vector \( \vec{v} \). Then

\[
Dilation1(R, M) := \bigcup_{m \in M} t_{\vec{v}_m}(R)
\]

For each point \( m \) in \( M \) a translation of the region \( R \) is performed. The union of all these translations is the dilation of \( R \) with \( M \). `Dilation1` is similar to the operator `MinkowskiAdd1`, the difference is that in `Dilation1` the structuring element is mirrored at the origin. The position of `StructElement` is meaningless, since the displacement vectors are determined with respect to the center of gravity of \( M \).

The parameter `Iterations` determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \). From the above definition it follows that an empty region is generated in case of an empty structuring element.

Structuring elements (`StructElement`) can be generated with operators such as `GenCircle`, `GenRectangle1`, `GenRectangle2`, `GenEllipse`, `DrawRegion`, `GenRegionPolygon`, `GenRegionPoints`, etc.

---

**Attention**

A dilation always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator `Union1` has to be called first.

HALCON 6.1.4
Parameter

- **Region** (input iconic)  
  Regions to be dilated.
- **StructElement** (input iconic)  
  Structuring element.
- **RegionDilation** (output iconic)  
  Dilated regions.
- **Iterations** (input control)  
  Number of iterations.
  Default Value: 1
  Suggested values: Iterations \( \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\} \)

Complexity

Let \( F_1 \) be the area of the input region, and \( F_2 \) be the area of the structuring element. Then the runtime complexity for one region is:

\[
O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations})
\]

Result

\( \text{Dilation1} \) returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: \( \text{SetSystem('noObjectResult',<RegionResult>)} \)
- empty region: \( \text{SetSystem('emptyRegionResult',<RegionResult>)} \)

Otherwise, an exception is raised.

Parallelization Information

\( \text{Dilation1} \) is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successors

- ReduceDomain, AddChannels, SelectShape, AreaCenter, Connection

See also

- Erosion1, Erosion2, Opening, Closing

Alternatives

- MinkowskiAdd1, MinkowskiAdd2, Dilation2, DilationGolay, DilationSeq

Module

\[
\text{[out]} \ HRegionX \ \text{RegionDilation} \ HRegionX.Dilation2 \\
\text{([in]} \ HRegionX \ \text{StructElement}, \ \text{[in]} \ \text{long Row}, \ \text{[in]} \ \text{long Column}, \ \text{[in]} \ \text{long Iterations})
\]

\[
\text{void} \ \text{HOperatorSetX.Dilation2} \ (\text{[in]} \ IObjectX \ \text{Region}, \ \text{[in]} \ IObjectX \ \text{StructElement}, \ \text{[out]} \ \text{HUntypedObjectX RegionDilation}, \ \text{[in]} \ \text{VARIANT Row}, \ \text{[in]} \ \text{VARIANT Column}, \ \text{[in]} \ \text{VARIANT Iterations})
\]

\( \text{Dilate a region (using a reference point)} \)

\( \text{Dilation2} \) dilates the input regions with a structuring element (\( \text{StructElement} \)) having the reference point \( (\text{Row,Column}) \). \( \text{Dilation2} \) has a similar effect as \( \text{Dilation1} \), the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter \( \text{Iterations} \) determines the number of iterations.
which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

An empty region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Attention

A dilation always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator Union1 has to be called first.

Parameter

\[农资 \text{Region} \text{(input iconic)} \rightarrow \text{Region(-array)} \]
Regions to be dilated.

\[农资 \text{StructElement} \text{(input iconic)} \rightarrow \text{region} \]
Structuring element.

\[农资 \text{RegionDilation} \text{(output iconic)} \rightarrow \text{region(-array)} \]
Dilated regions.

\[农资 \text{Row} \text{(input control)} \rightarrow \text{point.y} \rightarrow \text{long / VARIANT} \]
Row coordinate of the reference point.

\[农资 \text{Column} \text{(input control)} \rightarrow \text{point.x} \rightarrow \text{long / VARIANT} \]
Column coordinate of the reference point.

\[农资 \text{Iterations} \text{(input control)} \rightarrow \text{integer} \rightarrow \text{long / VARIANT} \]
Number of iterations.

\[农资 \text{Default Value : 0} \]

\[农资 \text{Suggested values : Iterations} \in \{1, 2, 3, 4, 5, 7, 11, 17, 25, 32, 64, 128\} \]

\[农资 \text{Recommended Increment : 1} \]

Complexity

Let \( F_1 \) be the area of the input region, and \( F_2 \) be the area of the structuring element. Then the runtime complexity for one region is:

\[ O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations}) . \]

Result

\[农资 \text{Dilation2} \text{returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:} \]

- no region: \text{SetSystem(‘noObjectResult’,<RegionResult>)}
- empty region: \text{SetSystem(‘emptyRegionResult’,<RegionResult>)}

Otherwise, an exception is raised.

Parallelization Information

\[农资 \text{Dilation2 is reentrant and automatically parallelized (on tuple level).} \]

Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successors

ReduceDomain, AddChannels, SelectShape, AreaCenter, Connection

See also

Erosion1, Erosion2, Opening, Closing

Alternatives

MinkowskiAdd1, MinkowskiAdd2, Dilation1, DilationGolay, DilationSeq

HALCON 6.1.4
Dilate a region with a circular structuring element.

\texttt{DilationCircle} applies a Minkowski addition with a circular structuring element to the input regions \texttt{Region}. Because the circular mask is symmetrical, this is identical to a dilation. The size of the circle used as structuring element is determined by \texttt{Radius}.

The operator results in enlarged regions, smoothed region boundaries, and the holes smaller than the circular mask in the interior of the region are closed. It is useful to select only values like 3.5, 5.5, etc. for \texttt{Radius} in order to avoid a translation of a region, because integer radii result in the circle having a non-integer center of gravity which is rounded to the next integer.

\textbf{Attention}

\texttt{DilationCircle} is applied to each input region separately. If gaps between different regions are to be closed, \texttt{Union1} or \texttt{Union2} has to be called first.

\textbf{Parameter}

\begin{itemize}
  \item \texttt{Region} (input iconic) \hspace{1em} region(-array) $\sim$ \texttt{HRegionX} / \texttt{IHObjectX}
    \hspace{1em} Regions to be dilated.
  \item \texttt{RegionDilation} (output iconic) \hspace{1em} region(-array) $\sim$ \texttt{HRegionX} / \texttt{HUntypedObjectX}
    \hspace{1em} Dilated regions.
  \item \texttt{Radius} (input control) \hspace{1em} real $\sim$ \texttt{VARIANT} ( real, integer )
    \hspace{1em} Radius of the circular structuring element.
\end{itemize}

\textbf{Default Value} : 3.5

\textbf{Suggested values} : \texttt{Radius} $\in \{ 1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5 \}$

\textbf{Typical range of values} : 0.5 $\leq$ \texttt{Radius} $\leq$ 0.5

\textbf{Minimum Increment} : 1.0

\textbf{Recommended Increment} : 1.0

\textbf{Complexity}

Let $F_1$ be the area of an input region. Then the runtime complexity for one region is:

$$O(2 \cdot \text{Radius} \cdot \sqrt{F_1})$$

\textbf{Result}

\texttt{DilationCircle} returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: \texttt{SetSystem('noObjectResult',<RegionResult>)}
- empty region: \texttt{SetSystem('emptyRegionResult',<RegionResult>)}

Otherwise, an exception is raised.

\textbf{Parallelization Information}

\texttt{DilationCircle} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Predecessors}

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

\textbf{Possible Successors}

ReduceDomain, SelectShape, AreaCenter, Connection

\textbf{See also}

GenCircle, ErosionCircle, ClosingCircle, OpeningCircle
Alternatives
MinkowskiAdd1, MinkowskiAdd2, ExpandRegion, Dilation1, Dilation2, DilationRectangle1

Module

Morphology

[out] HRegionX RegionDilation HRegionX.DilationGolay

void HOperatorSetX.DilationGolay ([in] IHObjectX Region,
[out] HUntypedObjectX RegionDilation, [in] VARIANT GolayElement,
in] VARIANT Iterations, [in] VARIANT Rotation)

Dilate a region with an element from the Golay alphabet.

DilationGolay dilates a region with the selected element GolayElement from the Golay alphabet. The following structuring elements are available:
't', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements. The operator works by shifting the structuring element over the region to be processed (Region). For all positions of the structuring element that intersect the region, the corresponding reference point (relative to the structuring element) is added to the output region. This means that the union of all translations of the structuring element within the region is computed.

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

Attention
Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

Parameter

- Region (input iconic) ........................................ region(-array) \( \sim \) HRegionX / IHObjectX
  Regions to be dilated.
- RegionDilation (output iconic) ................................ region(-array) \( \sim \) HRegionX / HUntypedObjectX
  Dilated regions.
- GolayElement (input control) ........................................ string \( \sim \) String / VARIANT
  Structuring element from the Golay alphabet.
  Default Value: 'h'
  List of values: GolayElement \( \in \{ 't', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k' \} \)
- Iterations (input control) ........................................ integer \( \sim \) long / VARIANT
  Number of iterations.
  Default Value: 1
  Suggested values: Iterations \( \in \{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50 \} \)
  (lin) Minimum Increment: 1
  Recommended Increment: 1
- Rotation (input control) ........................................ integer \( \sim \) long / VARIANT
  Rotation of the Golay element. Depending on the element, not all rotations are valid.
  Default Value: 0
  List of values: Rotation \( \in \{ 0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15 \} \)

Complexity
Let \( F \) be the area of an input region. Then the runtime complexity for one region is:

\[
O(3 \cdot \sqrt{F})
\]

Result
DilationGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:
• no region: SetSystem('noObjectResult',<RegionResult>)
• empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

--- Parallelization Information ---
DilationGolay is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessors ---
Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

--- Possible Successors ---
ReduceDomain, SelectShape, AreaCenter, Connection

See also
ErosionGolay, OpeningGolay, ClosingGolay, HitOrMissGolay, ThinningGolay, ThickeningGolay, GolayElements

--- Alternatives ---
Dilation1, Dilation2, DilationSeq

--- Module ---
Morphology

```c
#include "HRegionX.H"

HRegionX RegionDilation = HRegionX.DilationRectangle1([in] long Width, [in] long Height )

void HOperatorSetX.DilationRectangle1 ([in] IObjectX Region,
[in] HUntypedObjectX RegionDilation, [in] VARIANT Width,
[in] VARIANT Height )
```

Dilate a region with a rectangular structuring element.

DilationRectangle1 applies a dilation with a rectangular structuring element to the input regions Region. The size of the structuring rectangle is Width × Height. The operator results in enlarged regions, and the holes smaller than the rectangular mask in the interior of the regions are closed.

DilationRectangle1 is a very fast operation because the height of the rectangle enters only logarithmically into the runtime complexity, while the width does not enter at all. This leads to excellent runtime efficiency, even in the case of very large rectangles (edge length > 100).

--- Attention ---
DilationRectangle1 is applied to each input region separately. If gaps between different regions are to be closed, Union1 or Union2 has to be called first.

--- Parameter ---

- **Region** (input iconic) ..............................................region(-array) ↦ HRegionX / IObjectX
  
  Regions to be dilated.

- **RegionDilation** (output iconic) .............................region(-array) ↦ HRegionX / HUntypedObjectX
  
  Dilated regions.

- **Width** (input control) ...........................................extent.x ↦ long / VARIANT
  
  Width of the structuring rectangle.

  Default Value : 10
  Suggested values : Width ∈ {1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100, 150, 200}
  Typical range of values : 1 ≤ Width ≤ 1(lin)
  Minimum Increment : 1
  Recommended Increment : 10

- **Height** (input control) .........................................extent.y ↦ long / VARIANT
  
  Height of the structuring rectangle.

  Default Value : 10
  Suggested values : Height ∈ {1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100, 150, 200}
  Typical range of values : 1 ≤ Height ≤ 1(lin)
  Minimum Increment : 1
  Recommended Increment : 10
Let $F$ be the area of an input region and $H$ be the height of the rectangle. Then the runtime complexity for one region is:

$$O(\sqrt{F \cdot \log(H)})$$.

**Result**

*DilationRectangle1* returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`

Otherwise, an exception is raised.

**Parallelization Information**

*DilationRectangle1* is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

See also

GenRectangle1, GenRegionPolygonFilled

**Alternatives**

MinkowskiAdd1, MinkowskiAdd2, ExpandRegion, Dilation1, Dilation2, DilationCircle

**Module**

Morphology

Dilate a region sequentially.

*DilationSeq* computes the sequential dilation of the input region *Region* with the selected structuring element *GolayElement* from the Golay alphabet. This is done by executing the operator *DilationGolay* with all rotations of the structuring element *Iterations* times. The following structuring elements can be selected: 'l', 'd', 'c', 'f', 'h', 'k'.

In order to compute the skeleton of a region, usually the elements 'l' and 'm' are used. Only the “foreground elements” (even rotation numbers) are used. The elements 'i' and 'e' result in unchanged output regions. The elements 'l', 'm' and 'f2' are identical for the foreground. The Golay elements, together with all possible rotations, are described with the operator *GolayElements*.

**Parameter**

- **Region** (input iconic) region(-array) \(\rightarrow\) HRegionX / IObjectX
  - Regions to be dilated.
- **RegionDilation** (output iconic) region(-array) \(\rightarrow\) HRegionX / HUntypedObjectX
  - Dilated regions.
- **GolayElement** (input control) string \(\rightarrow\) String / VARIANT
  - Structuring element from the Golay alphabet.

  **Default Value**: 'h'

  **List of values**: GolayElement \(\in\) \{'l', 'd', 'c', 'f', 'h', 'k'\}
Iterations (input control) ......................... integer \rightarrow long / VARIANT
Number of iterations.
Default Value : 1
Suggested values: \( \text{Iterations} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\} \)
Minimum Increment : 1
Recommended Increment : 1

Complexity
Let \( F \) be the area of an input region. Then the runtime complexity for one region is:
\[
O(\text{Iterations} \cdot 20 \cdot \sqrt{F}).
\]

Result
DilationSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
DilationSeq is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

Possible Successors
ReduceDomain, SelectShape, AreaCenter, Connection

See also
ErosionSeq, HitOrMissSeq, ThinningSeq

Alternatives
Dilation1, Dilation2, DilationGolay

Module

**Erosion1** erodes the input regions with a structuring element. By applying **Erosion1** to a region, its boundary gets smoothed. In the process, the area of the region is reduced. Furthermore, connected regions may be split. Such regions, however, remain logically one region. The erosion is a set-theoretic region operation. It uses the intersection operation.

Let \( M \) (StructElement) and \( R \) (Region) be two regions, where \( M \) is the structuring element and \( R \) is the region to be processed. Furthermore, let \( m \) be a point in \( M \). Then the displacement vector \( \vec{v}_m = (dx, dy) \) is defined as the difference of the center of gravity of \( M \) and the vector \( \vec{m} \). Let \( t_{-\vec{v}_m}(R) \) denote the translation of a region \( R \) by a vector \( \vec{v}_m \). Then
\[
\text{Erosion1}(R, M) := \bigcap_{m \in M} t_{-\vec{v}_m}(R).
\]

For each point \( m \) in \( M \) a translation of the region \( R \) is performed. The intersection of all these translations is the erosion of \( R \) with \( M \). **Erosion1** is similar to the operator **MinkowskiSub1**, the difference is that in
Erosion1 the structuring element is mirrored at the origin. The position of StructElement is meaningless, since the displacement vectors are determined with respect to the center of gravity of $M$.

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration $n - 1$ is used as input for iteration $n$. From the above definition it follows that the maximum region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Parameter

**Region** (input iconic) ................................. region(-array) $\rightarrow$ HRegionX / HObjectX
Regions to be eroded.

**StructElement** (input iconic) ......................... region $\rightarrow$ HRegionX / HObjectX
Structuring element.

**RegionErosion** (output iconic) ....................... region(-array) $\rightarrow$ HRegionX / HUntypedObjectX
Eroded regions.

**Iterations** (input control) ............................ integer $\rightarrow$ long / VARIANT
Number of iterations.

Default Value : 1
Suggested values: Iterations $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}$
Minimum Increment : 1
Recommended Increment : 1

Complexity
Let $F_1$ be the area of the input region, and $F_2$ be the area of the structuring element. Then the runtime complexity for one region is:

$$O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations}).$$

Result
Erosion1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
Erosion1 is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Threshold, Regiongrowing, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successors
Connection, ReduceDomain, SelectShape, AreaCenter
See also
TransposeRegion

Alternatives
MinkowskiSub1, MinkowskiSub2, Erosion2, ErosionGolay, ErosionSeq

Module
Morphology
**CHAPTER 8. MORPHOLOGY**

**Erosion2** erodes the input regions with a structuring element (*StructElement*) having the reference point (*Row, Column*). **Erosion2** has a similar effect as **Erosion1**, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter **Iterations** determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

A maximum region is generated in case of an empty structuring element.

Structuring elements (*StructElement*) can be generated with operators such as **GenCircle**, **GenRectangle1**, **GenRectangle2**, **GenEllipse**, **DrawRegion**, **GenRegionPolygon**, **GenRegionPoints**, etc.

---

**Parameter**

- **Region** (input iconic) .......................... region(-array) \( \rightarrow \) HRegionX / IHObjectX
  Regions to be eroded.

- **StructElement** (input iconic) ....................... region \( \rightarrow \) HRegionX / IHObjectX
  Structuring element.

- **RegionErosion** (output iconic) ................. region(-array) \( \rightarrow \) HRegionX / HUntypedObjectX
  Eroded regions.

- **Row** (input control) .............................. point.y \( \rightarrow \) long / VARIANT
  Row coordinate of the reference point.
  **Default Value**: 0
  **Typical range of values**: \( 0 \leq \text{Row} \leq 0 \) (lin)
  **Minimum Increment**: 1
  **Recommended Increment**: 1

- **Column** (input control) ............................ point.x \( \rightarrow \) long / VARIANT
  Column coordinate of the reference point.
  **Default Value**: 0
  **Typical range of values**: \( 0 \leq \text{Column} \leq 0 \) (lin)
  **Minimum Increment**: 1
  **Recommended Increment**: 1

- **Iterations** (input control) ......................... integer \( \rightarrow \) long / VARIANT
  Number of iterations.
  **Default Value**: 1
  **Suggested values**: \( \text{Iterations} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\} \) (lin)
  **Minimum Increment**: 1
  **Recommended Increment**: 1

---

**Complexity**

Let \( F_1 \) be the area of the input region, and \( F_2 \) be the area of the structuring element. Then the runtime complexity for one region is:

\[
O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations}).
\]

**Result**

**Erosion2** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`
Otherwise, an exception is raised.

**Parallelization Information**

Erosion2 is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, Regiongrowing, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

See also

TransposeRegion, GenCircle, GenRectangle2, GenRegionPolygon

**Alternatives**

MinkowskiSub2, MinkowskiSub1, Erosion1, ErosionGolay, ErosionSeq

**Module**

Morphology

---

```c
[out] HRegionX RegionErosion HRegionX.ErosionCircle
([in] VARIANT Radius )
```

```c
void HOperatorSetX.ErosionCircle ([in] IObjectX Region,
[out] HUntypedObjectX RegionErosion, [in] VARIANT Radius )
```

Erode a region with a circular structuring element.

**ErosionCircle** applies a Minkowski subtraction with a circular structuring element to the input regions Region. Because the circular mask is symmetrical, this is identical to an erosion. The size of the circle used as structuring element is determined by Radius.

The operator results in reduced regions, smoothed region boundaries, and the regions smaller than the circular mask are eliminated. It is useful to select only values like 3.5, 5.5, etc. for Radius in order to avoid a translation of a region, because integer radii result in a circle having a non-integer center of gravity which is rounded to the next integer.

**Parameter**

- **Region** (input iconic) ..................................................... region(-array) \(\sim\) HRegionX / IObjectX
  Regions to be eroded.
- **RegionErosion** (output iconic) ................................. region(-array) \(\sim\) HRegionX / HUntypedObjectX
  Eroded regions.
- **Radius** (input control) .................................................. real \(\sim\) VARIANT ( real, integer )
  Radius of the circular structuring element.
  - **Default Value** : 3.5
  - **Suggested values** : Radius \(\in\) \{1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5\}
  - **Typical range of values** : \(0.5 \leq \text{Radius} \leq 0.5\text{(lin)}\)
  - **Minimum Increment** : 1.0
  - **Recommended Increment** : 1.0

**Complexity**

Let \(F_1\) be the area of an input region. Then the runtime complexity for one region is:

\[
O(2 \cdot \text{Radius} \cdot \sqrt{F_1})
\]

**Result**

ErosionCircle returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

HALCON 6.1.4
Otherwise, an exception is raised.

---

**Parallelization Information**

ErosionCircle is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, RegionGrowing, Watersheds, ClassNdimNorm

---

**Possible Successors**

Connection, ReduceDomain, SelectShape, AreaCenter

---

See also

GenCircle, DilationCircle, ClosingCircle, OpeningCircle

---

**Alternatives**

MinkowskiSub1

---

**Module**

Morphology

---

```c
[out] HRegionX RegionErosion : HRegionX.erosionGolay
```

**Erode a region with an element from the Golay alphabet.**

ErosionGolay erodes a region with the selected element GolayElement from the Golay alphabet. The following structuring elements are available:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements. The operator works by shifting the structuring element over the region to be processed (Region). For all positions of the structuring element fully contained in the region, the corresponding reference point (relative to the structuring element) is added to the output region. This means that the intersection of all translations of the structuring element within the region is computed.

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration $n - 1$ is used as input for iteration $n$.

---

**Attention**

Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

---

**Parameter**

- **Region** (input iconic)  
  region(-array) $\rightarrow$ HRegionX / IHOBJECTX  
  Regions to be eroded.

- **RegionErosion** (output iconic)  
  region(-array) $\rightarrow$ HRegionX / HUNTYDEDOBJECTX  
  Eroded regions.

- **GolayElement** (input control)  
  string $\rightarrow$ String / VARIANT  
  Structuring element from the Golay alphabet.  
  Default Value : 'h'  
  List of values : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

- **Iterations** (input control)  
  integer $\rightarrow$ long / VARIANT  
  Number of iterations.  
  Default Value : 1  
  Suggested values : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}  
  (lin)Minimum Increment : 1  
  Recommended Increment : 1
8.2. REGION

Rotation (input control) .............................................. integer \sim long / VARIANT
Rotation of the Golay element. Depending on the element, not all rotations are valid.

Default Value : 0
List of values : Rotation \in \{0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15\}

Complexity
Let \( F \) be the area of an input region. Then the runtime complexity for one region is:

\[ O(3 \cdot \sqrt{F}) \]

Result
ErosionGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information
ErosionGolay is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Threshold, RegionGrowing, Watersheds, ClassNdimNorm

Possible Successors
ReduceDomain, SelectShape, AreaCenter, Connection

See also
DilationGolay, OpeningGolay, ClosingGolay, HitOrMissGolay, ThinningGolay,
ThickeningGolay, GolayElements

Alternatives
ErosionSeq, Erosion1, Erosion2

Module
Morphology

```
[out] HRegionX RegionErosion HRegionX.ErosionRectangle1
([in] long Width, [in] long Height )

void HOperatorSetX.ErosionRectangle1 ([in] IObjectX Region,
[out] HUntypedObjectX RegionErosion, [in] VARIANT Width,
[in] VARIANT Height )
```

Erod a region with a rectangular structuring element.

ErosionRectangle1 applies an erosion with a rectangular structuring element to the input regions Region.
The size of the structuring rectangle is Width \times Height. The operator results in reduced regions, and the areas smaller than the rectangular mask are eliminated.

ErosionRectangle1 is a very fast operation because the height of the rectangle enters only logarithmically into the runtime complexity, while the width does not enter at all. This leads to excellent runtime efficiency, even in the case of very large rectangles (edge length \( > 100 \)).

Regions containing small connecting strips between large areas are separated only seemingly. They remain logically one region.

Parameter

- Region (input iconic) .............................................. region(-array) \sim HRegionX / IObjectX
  Regions to be eroded.
- RegionErosion (output iconic) ......................... region(-array) \sim HRegionX / HUntypedObjectX
  Eroded regions.
CHAPTER 8. MORPHOLOGY

- **Width** (input control) ..........................  extent.x  \( \sim \) long / VARIANT
  Width of the structuring rectangle.

  Default Value : 10
  Suggested values : Width \( \in \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100\}\)
  Typical range of values : \(1 \leq Width \leq 1(\text{lin})\)
  Minimum Increment : 1
  Recommended Increment : 1

- **Height** (input control) ..........................  extent.y  \( \sim \) long / VARIANT
  Height of the structuring rectangle.

  Default Value : 10
  Suggested values : Height \( \in \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100\}\)
  Typical range of values : \(1 \leq Height \leq 1(\text{lin})\)
  Minimum Increment : 1
  Recommended Increment : 1

**Complexity**

Let \(F1\) be the area of an input region and \(H\) be the height of the rectangle. Then the runtime complexity for one
region is:

\[O\left(\sqrt{F1 \cdot \text{ld}(H)}\right)\]  

**Result**

`ErosionRectangle1` returns TRUE if all parameters are correct. The behavior in case of empty or no input
region can be set via:

- no region: `SetSystem(‘noObjectResult’,<RegionResult>)`
- empty region: `SetSystem(‘emptyRegionResult’,<RegionResult>)`

Otherwise, an exception is raised.

**Parallelization Information**

`ErosionRectangle1` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

`Threshold, RegionGrowing, Watersheds, ClassNdimNorm`

**Possible Successors**

`ReduceDomain, SelectShape, AreaCenter, Connection`

See also

`GenRectangle1`

**Alternatives**

`Erosion1, MinkowskiSub1`

**Module**

`Morphology`

```c
[out] HRegionX RegionErosion HRegionX.ErosionSeq
([in] String GolayElement, [in] long Iterations )
```

```c
void HOperatorSetX.ErosionSeq ([in] IObjectX Region,
[out] HUntypedObjectX RegionErosion, [in] VARIANT GolayElement,
[in] VARIANT Iterations )
```

Erode a region sequentially.

`ErosionSeq` computes the sequential erosion of the input region `Region` with the selected structuring element
`GolayElement` from the Golay alphabet. This is done by executing the operator `ErosionGolay` with all
rotations of the structuring element `Iterations` times. The following structuring elements can be selected:

8.2. REGION

Only the “foreground elements” (even rotation numbers) are used. The elements ‘i’ and ‘e’ result in unchanged output regions. The elements ‘l’, ‘m’ and ‘f2’ are identical for the foreground. The Golay elements, together with all possible rotations, are described with the operator `GolayElements`.

- **Region** (input iconic) region(-array) \( \sim \) `HRegionX / IHObjectX` Regions to be eroded.
- **RegionErosion** (output iconic) region(-array) \( \sim \) `HRegionX / HUntypedObjectX` Eroded regions.
- **GolayElement** (input control) string \( \sim \) `String / VARIANT` Structuring element from the Golay alphabet.
  - Default Value: ‘h’
- **Iterations** (input control) integer \( \sim \) `long / VARIANT` Number of iterations.
  - Default Value: 1
  - Suggested values: `Iterations \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}`
  - (lin)Minimum Increment: 1
  - Recommended Increment: 1

Let \( F \) be the area of an input region. Then the runtime complexity for one region is:

\[
O(\text{Iterations} \cdot 20 \cdot \sqrt{F})
\]

**ErosionSeq** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem(‘noObjectResult’,<RegionResult>)`
- empty region: `SetSystem(‘emptyRegionResult’,<RegionResult>)`

Otherwise, an exception is raised.

**Parallelization Information**

- **ErosionSeq** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**
- Threshold, Regiongrowing, Watersheds, ClassNdimNorm

**Possible Successors**
- Connection, ReduceDomain, SelectShape, AreaCenter
- DilationSeq, HitOrMissSeq, ThinningSeq

**See also**
- ErosionGolay, Erosion1, Erosion2

**Alternatives**
- Morphology

```java
[out] HRegionX RegionFitted HRegionX.Fitting
([in] HRegionX StructElements )

void HOperatorSetX.Fitting ([in] IHObjectX Region,
[in] IHObjectX StructElements, [out] HUntypedObjectX RegionFitted )
```

Perform a closing after an opening with multiple structuring elements.

`Fitting` performs an **Opening** and a **Closing** successively on the input regions. The eight structuring elements normally used for this operation can be generated with the operator `GenStructElements`. However, other user-defined structuring elements can also be used. Let \( R \) be the input region(s) and let \( M_i \) denote the...
structuring elements. Furthermore, let $P$ be the result of the opening and $Q$ be the final result. Then the operator can be formalized as follows:

$$
P = \bigcup_{i=1}^{n} (R \circ M_i)$$

$$
Q = \bigcap_{i=1}^{n} (P \bullet M_i)$$

Regions larger than the structuring elements are preserved, while small gaps are closed.

---

**Parameter**

- **Region** (input iconic) .............................................. region(-array) $\rightarrow$ HRegionX / IHObjectX
  
  Regions to be processed.

- **StructElements** (input iconic) ................................. region(-array) $\rightarrow$ HRegionX / IHObjectX
  
  Structuring elements.

- **RegionFitted** (output iconic) ................................. region(-array) $\rightarrow$ HRegionX / HUntypedObjectX
  
  Fitted regions.

---

**Result**

Fitting returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

Fitting is reentrant and processed without parallelization.

---

**Possible Predecessors**

- GenStructElements, GenRegionPoints

---

**Possible Successors**

- ReduceDomain, SelectShape, AreaCenter, Connection

---

**Alternatives**

- Opening, Closing, Connection, SelectShape

---

**Module**

---

```c
void HRegionX.GenStructElements ([in] String Type, [in] long Row,
[in] long Column )

void HOperatorSetX.GenStructElements ([out] HUntypedObjectX StructElements, [in] VARIANT Type, [in] VARIANT Row,
[in] VARIANT Column )
```

Generate standard structuring elements.

GenStructElements serves to generate eight structuring elements normally used in the operator Fitting. The default value ‘noise’ of the parameter Type generates elements especially suited for the elimination of noise.
> **StructElements** (output iconic) 
>                region(-array) \(\sim\) \(H\)RegionX / \(H\)UntypedObjectX 
> Generated structuring elements.

> **Type** (input control) 
>                string \(\sim\) String / VARIANT 
> Type of structuring element to generate. 
>  
> **Default Value**: 'noise' 
>  
> **List of values**: Type \(\in\) {'noise'} 

> **Row** (input control) 
>                point.y \(\sim\) long / VARIANT 
> Row coordinate of the reference point. 
>  
> **Default Value**: 1 
>  
> **Suggested values**: Row \(\in\) \{0, 1, 10, 50, 100, 200, 300, 400\} (lin) 

> **Column** (input control) 
>                point.x \(\sim\) long / VARIANT 
> Column coordinate of the reference point. 
>  
> **Default Value**: 1 
>  
> **Suggested values**: Column \(\in\) \{0, 1, 10, 50, 100, 200, 300, 400\} (lin) 

---

**Result** 

GenStructElements returns TRUE if all parameters are correct. Otherwise, an exception is raised.

---

**Parallelization Information** 

GenStructElements is reentrant and processed without parallelization.

---

**Possible Successors** 

Fitting, HitOrMiss, Opening, Closing, Erosion2, Dilation2 

---

**See also** 

GolayElements 

---

**Module** 

Morphology 

---

```c
void HRegionX.GolayElements ([out] HRegionX StructElement2, 
[in] String GolayElement, [in] long Rotation, [in] long Row, 
[in] long Column )
```

```c
void HOperatorSetX.GolayElements 
([out] HUntypedObjectX StructElement1, [out] HUntypedObjectX StructElement2, 
[in] VARIANT Column )
```

---

Generate the structuring elements of the Golay alphabet.

GolayElements generates the structuring elements from the Golay alphabet. The parameter GolayElement determines the name of the structuring element, while Rotation determines its rotation. The structuring elements are intended for use in HitOrMiss: In StructElement1 the structuring element for the foreground is returned, while in StructElement2 the structuring element for the background is returned. Row and Column determine the reference point of the structuring element.

The rotations are numbered from 0 to 15. This does not mean, however, that there are 16 different rotations: Even values denote rotations of the foreground elements, while odd values denote rotations of the background elements.

For GolayElements only even values are accepted, and determine the Golay element for StructElement1. The next larger odd value is used for StructElement2. There are no rotations for the Golay elements ‘h’ and ‘i’.
Therefore, only the values 0 and 1 are possible as “rotations” (and hence only 0 for *GolayElements*). The element 'e' has only four possible rotations, and hence the rotation must be between 0 and 7 (for *GolayElements* the values 0, 2, 4, or 6 must be used).

The tables below show the elements of the Golay alphabet with all possible rotations. The characters used have the following meaning:

- Foreground pixel
- Background pixel
- Don’t care pixel

The names of the elements and their rotation numbers are displayed below the respective element. The elements with even number contain the foreground pixels, while the elements with odd numbers contain the background pixels.

<table>
<thead>
<tr>
<th>Element</th>
<th>Rotation 1</th>
<th>Rotation 2</th>
<th>Rotation 3</th>
<th>Rotation 4</th>
<th>Rotation 5</th>
<th>Rotation 6</th>
<th>Rotation 7</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(0,1)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>i(0,1)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>e(0,1)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>e(2,3)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>e(4,5)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>e(6,7)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(0,1)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(2,3)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(4,5)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(6,7)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(8,9)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(10,11)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>l(12,13)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(0,1)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(2,3)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(4,5)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(6,7)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(8,9)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(10,11)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>m(12,13)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>d(0,1)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>d(2,3)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>d(4,5)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
<tr>
<td>d(6,7)</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
<td>. . . .</td>
</tr>
</tbody>
</table>
CHAPTER 8. MORPHOLOGY

Parameter

- **StructElement1** (output iconic) ....................... region $\rightsquigarrow HRegionX / HUntypedObjectX
  Structuring element for the foreground.

- **StructElement2** (output iconic) ....................... region $\rightsquigarrow HRegionX / HUntypedObjectX
  Structuring element for the background.

- **GolayElement** (input control) ....................... string $\rightsquigarrow String / VARIANT
  Name of the structuring element.
  Default Value : 'l'
  List of values : GolayElement $\in \{ 'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k' \}$

- **Rotation** (input control) .......................... integer $\rightsquigarrow long / VARIANT
  Rotation of the Golay element. Depending on the element, not all rotations are valid.
  Default Value : 0
  List of values : Rotation $\in \{ 0, 2, 4, 6, 8, 10, 12, 14 \}$

- **Row** (input control) ................................. point.y $\rightsquigarrow long / VARIANT
  Row coordinate of the reference point.
  Default Value : 16
  Suggested values : Row $\in \{ 0, 16, 32, 128, 256 \}$
  Typical range of values : $0 \leq \text{Row} \leq 0(\text{lin})$
  Minimum Increment : 1
  Recommended Increment : 1

- **Column** (input control) ................................. point.x $\rightsquigarrow long / VARIANT
  Column coordinate of the reference point.
  Default Value : 16
  Suggested values : Column $\in \{ 0, 16, 32, 128, 256 \}$
  Typical range of values : $0 \leq \text{Column} \leq 0(\text{lin})$
  Minimum Increment : 1
  Recommended Increment : 1

Result

GolayElements returns TRUE if all parameters are correct. Otherwise, an exception is raised.

Parallelization Information

GolayElements is reentrant and processed without parallelization.

Possible Successors

HitOrMiss

See also

DilationGolay, ErosionGolay, OpeningGolay, ClosingGolay, HitOrMissGolay, ThickeningGolay

Alternatives

GenRegionPoints, GenStructElements, GenRegionPolygonFilled

References


Module

Hit-or-miss operation for regions.

HitOrMiss performs the hit-or-miss-transformation. First, an erosion with the structuring element StructElement1 is done on the input region Region. Then an erosion with the structuring element StructElement2...
8.2. REGION

StructElement2 is performed on the complement of the input region. The intersection of the two resulting regions is the result RegionHitMiss of HitOrMiss.

The hit-or-miss-transformation selects precisely the points for which the conditions given by the structuring elements StructElement1 and StructElement2 are fulfilled. StructElement1 determines the condition for the foreground pixels, while StructElement2 determines the condition for the background pixels. In order to obtain sensible results, StructElement1 and StructElement2 must fit like key and lock. In any case, StructElement1 and StructElement2 must be disjunct. Row and Column determine the reference point of the structuring elements.

Structuring elements (StructElement1, StructElement2) can be generated by calling operators like GenStructElements, GenRegionPoints, etc.

Parameter

- Region (input iconic) ......................................... region(-array) $\rightarrow$ HRegionX / IHObjectX
  Regions to be processed.
- StructElement1 (input iconic) ................................. region $\rightarrow$ HRegionX / IHObjectX
  Erosion mask for the input regions.
- StructElement2 (input iconic) ................................. region $\rightarrow$ HRegionX / IHObjectX
  Erosion mask for the complements of the input regions.
- RegionHitMiss (output iconic) ............................... region(-array) $\rightarrow$ HRegionX / HUntypedObjectX
  Result of the hit-or-miss operation.
- Row (input control) .......................................... point.y $\rightarrow$ long / VARIANT
  Row coordinate of the reference point.
  Default Value : 16
  Suggested values : Row \in \{0, 16, 32, 128, 256\}
  Typical range of values : $0 \leq$ Row \leq 0(lin)
  Minimum Increment : 1
  Recommended Increment : 1
- Column (input control) ....................................... point.x $\rightarrow$ long / VARIANT
  Column coordinate of the reference point.
  Default Value : 16
  Suggested values : Column \in \{0, 16, 32, 128, 256\}
  Typical range of values : $0 \leq$ Column \leq 0(lin)
  Minimum Increment : 1
  Recommended Increment : 1

Complexity

Let $F$ be the area of an input region, $F_1$ the area of the structuring element 1, and $F_2$ the area of the structuring element 2. Then the runtime complexity for one object is:

$$O \left( \sqrt{F} \cdot \left( \sqrt{F_1} + \sqrt{F_2} \right) \right) .$$

Result

HitOrMiss returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

HitOrMiss is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

GolayElements, GenStructElements, Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successors

Difference, ReduceDomain, SelectShape, AreaCenter, Connection
See also
Thinning, Thickening, GenRegionPoints, GenRegionPolygonFilled

Alternatives
HitOrMissGolay, HitOrMissSeq, Erosion2, Dilation2

Module

Hit-or-miss operation for regions using the Golay alphabet.

HitOrMissGolay performs the hit-or-miss-transformation for the input regions Region (using structuring elements from the Golay alphabet). First, an erosion with the foreground of the structuring element GolayElement is done on the input region Region. Then an erosion with the background of the structuring element GolayElement is performed on the complement of the input region. The intersection of the two resulting regions is the result RegionHitMiss of HitOrMissGolay. The following structuring elements are available:

' l', ' m', ' d', ' c', ' e', ' i', ' f', ' f2', ' h', ' k'.

The rotation number Rotation determines which rotation of the element should be used. The hit-or-miss-transformation selects precisely the points for which the conditions given by the selected Golay element are fulfilled.

Attention
Not all values of Rotation are valid for any Golay element.

Parameter

- Region (input iconic) ......................................... region(-array) \(\rightarrow\) HRegionX / IOBJECTX
  Regions to be processed.
- RegionHitMiss (output iconic) .............................. region(-array) \(\rightarrow\) HRegionX / HUntypedObjectX
  Result of the hit-or-miss operation.
- GolayElement (input control) .............................. string \(\rightarrow\) String / VARIANT
  Structuring element from the Golay alphabet.
  Default Value: ' h'
  List of values: GolayElement \(\in\) {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- Rotation (input control) .................................... integer \(\rightarrow\) long / VARIANT
  Rotation of the Golay element. Depending on the element, not all rotations are valid.
  Default Value: 0
  List of values: Rotation \(\in\) {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Complexity

Let \(F\) be the area of an input region. Then the runtime complexity for one region is:

\[
O(6 \cdot \sqrt{F}) .
\]

Result

HitOrMissGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult', <RegionResult>)
- empty region: SetSystem('emptyRegionResult', <RegionResult>)

Otherwise, an exception is raised.
HitOrMissGolay is reentrant and automatically parallelized (on tuple level).

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successors
ReduceDomain, SelectShape, AreaCenter, Connection

See also
ErosionGolay, DilationGolay, OpeningGolay, ThickeningGolay, ClassNdimNorm

Parallelization Information
HitOrMissGolay is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successors
ReduceDomain, SelectShape, AreaCenter, Connection

See also
ErosionGolay, DilationGolay, OpeningGolay, ThickeningGolay, ClassNdimNorm

HitOrMissSeq, HitOrMiss

Alternatives
Morphology

Module

Hit-or-miss operation for regions using the Golay alphabet (sequential).

HitOrMissGolay performs the hit-or-miss-transformation for the input regions Region using all rotations of a structuring element from the Golay alphabet. The result of the operator is the union of all intermediate results of the respective rotations. The following structuring elements are available:
'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Parameter

Region (input iconic) region-array
RegionHitMiss (output iconic) region-array
GolayElement (input control) string

Default Value: 'h'
List of values: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

Complexity
Let F be the area of an input region, and R be the number of rotations. Then the runtime complexity for one region is:

\[ O(R \cdot 6 \cdot \sqrt{F}) . \]

Result
HitOrMissSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.
Possible Successors
ReduceDomain, SelectShape, AreaCenter, Connection

See also
ThinningSeq, ThickeningSeq

Alternatives
HitOrMissGolay, HitOrMiss

Module
Morphology

```plaintext
HRegionX RegionMinkAdd HRegionX.MinkowskiAdd1
([in] HRegionX StructElement, [in] long Iterations )

void HOperatorSetX.MinkowskiAdd1 ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionMinkAdd,
[in] VARIANT Iterations )
```

Perform a Minkowski addition on a region.

MinkowskiAdd1 dilates the input regions with a structuring element. By applying MinkowskiAdd1 to a region, its boundary gets smoothed. In the process, the area of the region is enlarged. Furthermore, disconnected regions may be merged. Such regions, however, remain logically distinct region. The Minkowski addition is a set-theoretic region operation. It is based on translations and union operations.

Let $M$ (StructElement) and $R$ (Region) be two regions, where $M$ is the structuring element and $R$ is the region to be processed. Furthermore, let $m$ be a point in $M$. Then the displacement vector $\vec{v}_m = (dx, dy)$ is defined as the difference of the center of gravity of $M$ and the vector $\vec{m}$. Let $t_{\vec{v}_m}(R)$ denote the translation of a region $R$ by a vector $\vec{v}$. Then

$$MinkowskiAdd1(R, M) := \bigcup_{m \in M} t_{\vec{v}_m}(R)$$

For each point $m$ in $M$ a translation of the region $R$ is performed. The union of all these translations is the Minkowski addition of $R$ with $M$. MinkowskiAdd1 is similar to the operator Dilation1, the difference is that in Dilation1 the structuring element is mirrored at the origin. The position of StructElement is meaningless, since the displacement vectors are determined with respect to the center of gravity of $M$.

The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration $n - 1$ is used as input for iteration $n$. From the above definition it follows that an empty region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Attention
A Minkowski addition always results in enlarged regions. Closely spaced regions which may touch or overlap as a result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator Union1 has to be called first.

Parameter
- **Region** (input iconic) ............................................ region(-array) ⇒ HRegionX / IHObjectX
  Regions to be dilated.
- **StructElement** (input iconic) ................................. region ⇒ HRegionX / IHObjectX
  Structuring element.
- **RegionMinkAdd** (output iconic) ............................... region(-array) ⇒ HRegionX / HUntypedObjectX
  Dilated regions.
8.2. REGION

- **Iterations** (input control) .................................................. integer  \(\rightarrow\) long / VARIANT
  Number of iterations.
  
  **Default Value**: 1
  
  **Suggested values**: Iterations \(\in\) \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}
  
  (lin) **Minimum Increment**: 1
  
  **Recommended Increment**: 1

---

**Complexity**

Let \(F_1\) be the area of the input region, and \(F_2\) be the area of the structuring element. Then the runtime complexity for one region is:

\[
O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations}).
\]

---

**Result**

*MinkowskiAdd1* returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: *SetSystem* (‘noObjectResult’,<RegionResult>)
- empty region: *SetSystem* (‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

*MinkowskiAdd1* is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union, Watersheds, ClassNDimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

---

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

---

**See also**

TransposeRegion, MinkowskiSub1

---

**Alternatives**

MinkowskiAdd2, Dilation1

---

**Module**

**Morphology**

---

```c
[out] HRegionX RegionMinkAdd HRegionX.MinkowskiAdd2
```

---

**Dilate a region (using a reference point).**

*MinkowskiAdd2* computes the Minkowski addition of the input regions with a structuring element (StructElement) having the reference point (Row,Column). *MinkowskiAdd2* has a similar effect as *MinkowskiAdd1*, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter **Iterations** determines the number of iterations which are to be performed with the structuring element. The result of iteration \(n - 1\) is used as input for iteration \(n\).

An empty region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as *GenCircle*, *GenRectangle1*, *GenRectangle2*, *GenEllipse*, *DrawRegion*, *GenRegionPolygon*, *GenRegionPoints*, etc.

---

**Attention**

A Minkowski addition always results in enlarged regions. Closely spaced regions which may touch or overlap as
A result of the dilation are still treated as two separate regions. If the desired behavior is to merge them into one region, the operator \texttt{Union1} has to be called first.

---

**Parameter**

- **\texttt{Region}** (input iconic) \texttt{region(-array)} \rightarrow \texttt{HRegionX / IHOBJECTX}
  
  Regions to be dilated.

- **\texttt{StructElement}** (input iconic) \texttt{region} \rightarrow \texttt{HRegionX / IHOBJECTX}
  
  Structuring element.

- **\texttt{RegionMinkAdd}** (output iconic) \texttt{region(-array)} \rightarrow \texttt{HRegionX / HUntypedObjectX}
  
  Dilated regions.

- **\texttt{Row}** (input control) \texttt{point.y} \rightarrow \texttt{long / VARIANT}
  
  Row coordinate of the reference point.

  **Typical range of values**: \(1 \leq \text{Row} \leq 1\text{(lin)}\)

  **Minimum Increment**: 1

  **Recommended Increment**: 1

- **\texttt{Column}** (input control) \texttt{point.x} \rightarrow \texttt{long / VARIANT}
  
  Column coordinate of the reference point.

  **Typical range of values**: \(1 \leq \text{Column} \leq 1\text{(lin)}\)

  **Minimum Increment**: 1

  **Recommended Increment**: 1

- **\texttt{Iterations}** (input control) \texttt{integer} \rightarrow \texttt{long / VARIANT}
  
  Number of iterations.

  **Default Value**: 1

  **Suggested values**: \(\text{Iterations} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}\)

  **Minimum Increment**: 1

  **Recommended Increment**: 1

---

**Complexity**

Let \(F_1\) be the area of the input region, and \(F_2\) be the area of the structuring element. Then the runtime complexity for one region is:

\[
O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations})
\]

---

**Result**

\texttt{MinkowskiAdd2} returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: \texttt{SetSystem(‘noObjectResult’,<RegionResult>)}
- empty region: \texttt{SetSystem(‘emptyRegionResult’,<RegionResult>)}

Otherwise, an exception is raised.

---

**Parallelization Information**

\texttt{MinkowskiAdd2} is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

\texttt{Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled}

---

**Possible Successors**

\texttt{ReduceDomain, SelectShape, AreaCenter, Connection}

---

See also

\texttt{TransposeRegion}

---

Alternatives

\texttt{MinkowskiAdd1, Dilation1}

---

Module

Morphology
Erode a region.

MinkowskiSub1 computes the Minkowski subtraction of the input regions with a structuring element. By applying MinkowskiSub1 to a region, its boundary gets smoothed. In the process, the area of the region is reduced. Furthermore, connected regions may be split. Such regions, however, remain logically one region. The Minkowski subtraction is a set-theoretic region operation. It uses the intersection operation.

Let \( M \) (StructElement) and \( R \) (Region) be two regions, where \( M \) is the structuring element and \( R \) is the region to be processed. Furthermore, let \( m \) be a point in \( M \). Then the displacement vector \( \vec{v}_m = (dx,dy) \) is defined as the difference of the center of gravity of \( M \) and the vector \( m \). Let \( t_{\vec{v}_m}(R) \) denote the translation of a region \( R \) by a vector \( \vec{v} \). Then

\[
\text{MinkowskiSub1}(R,M) := \bigcap_{m \in M} t_{\vec{v}_m}(R)
\]

For each point \( m \) in \( M \) a translation of the region \( R \) is performed. The intersection of all these translations is the Minkowski subtraction of \( R \) with \( M \). MinkowskiSub1 is similar to the operator Erosion1, the difference is that in Erosion1 the structuring element is mirrored at the origin. The position of StructElement is meaningless, since the displacement vectors are determined with respect to the center of gravity of \( M \).

The parameter \( \text{Iterations} \) determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n-1 \) is used as input for iteration \( n \). From the above definition it follows that the maximum region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{Region}</td>
<td>(input iconic) region(-array) ( \sim ) HRegionX / IHOBJECTX</td>
</tr>
<tr>
<td>\textbf{StructElement}</td>
<td>(input iconic) region ( \sim ) HRegionX / IHOBJECTX</td>
</tr>
<tr>
<td>\textbf{RegionMinkSub}</td>
<td>(output iconic) region(-array) ( \sim ) HRegionX / HUntypedObjectX</td>
</tr>
<tr>
<td>\textbf{Iterations}</td>
<td>(input control) integer ( \sim ) long / VARIANT</td>
</tr>
</tbody>
</table>

Default Value : 1
Suggested values : Iterations \( \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\} \)
(Lin)Minimum Increment : 1
Recommended Increment : 1

Complexity

\[
O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations})
\]

Result

MinkowskiSub1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

HALCON 6.1.4
Otherwise, an exception is raised.

---

**Parallelization Information**

MinkowskiSub1 is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, Regiongrow, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

---

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

See also

---

**Alternatives**

MinkowskiSub2, Erosion1

---

**Module**

Morphology

---

```c
[out] HRegionX RegionMinkSub HRegionX.MinkowskiSub2
```

```
```

Erode a region (using a reference point).

MinkowskiSub2 computes the Minkowski subtraction of the input regions with a structuring element (StructElement) having the reference point (Row, Column). MinkowskiSub2 has a similar effect as MinkowskiSub1, the difference is that the reference point of the structuring element can be chosen arbitrarily. The parameter Iterations determines the number of iterations which are to be performed with the structuring element. The result of iteration \( n - 1 \) is used as input for iteration \( n \).

A maximum region is generated in case of an empty structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

---

**Parameter**

- **Region** (input iconic) ........................ region(-array) \( \sim \) HRegionX / IObjectX
  
  Regions to be eroded.

- **StructElement** (input iconic) ..................... region \( \sim \) HRegionX / IObjectX
  
  Structuring element.

- **RegionMinkSub** (output iconic) .................. region(-array) \( \sim \) HRegionX / HUntypedObjectX
  
  Eroded regions.

- **Row** (input control) ............................ point.y \( \sim \) long / VARIANT
  
  Row coordinate of the reference point.

**Default Value:** 0

**Suggested values:** \( \text{Row} \in \{0, 10, 16, 32, 64, 100, 128\} \)

**Typical range of values:** \( 0 \leq \text{Row} \leq 0(\text{lin}) \)

**Minimum Increment:** 1

**Recommended Increment:** 1
8.2. REGION

- **Column** (input control) .......................... point.x  \(\sim long /\text{VARIANT}

  Column coordinate of the reference point.

  Default Value: 0
  Suggested values: Column ∈ \{0, 10, 16, 32, 64, 100, 128\}
  Typical range of values: \(0 \leq \text{Column} \leq 0\) (lin)
  Minimum Increment: 1
  Recommended Increment: 1

- **Iterations** (input control) .......................... integer  \(\sim long /\text{VARIANT}

  Number of iterations.

  Default Value: 1
  Suggested values: Iterations ∈ \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}
  (lin) Minimum Increment: 1
  Recommended Increment: 1

### Complexity

Let \(F_1\) be the area of the input region, and \(F_2\) be the area of the structuring element. Then the runtime complexity for one region is:

\[
O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \text{Iterations}) .
\]

### Result

MinkowskiSub2 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(’noObjectResult’,<RegionResult>)
- empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

MinkowskiSub2 is reentrant and automatically parallelized (on tuple level).

### Possible Predecessors

Threshold, RegionGrowing, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

### Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

### See also

GenCircle, GenRectangle2, GenRegionPolygon

### Alternatives

MinkowskiSub1, Erosion1, Erosion2, ErosionGolay, ErosionSeq

### Module

Morphology

```halcon
[out] HRegionX RegionMorphHat HRegionX.MorphHat
([in] HRegionX StructElement )

void HOperatorSetX.MorphHat ([in] IHObjectX Region,
[in] IHObjectX StructElement, [out] HUntypedObjectX RegionMorphHat )
```

Compute the union of BottomHat and TopHat.

MorphHat computes the union of the regions that are removed by an Opening operation with the regions that are added by a Closing operation. Hence this is the union of the results of TopHat and BottomHat. The position of StructElement does not influence the result.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

HALCON 6.1.4
Attention

The individual regions are processed separately.

Parameter

▷ Region (input iconic) .............................. region(-array) \rightarrow HRegionX / IObjectX
   Regions to be processed.

▷ StructElement (input iconic) ...................... region \rightarrow HRegionX / IObjectX
   Structuring element (position-invariant).

▷ RegionMorphHat (output iconic) .................. region(-array) \rightarrow HRegionX / HUntypedObjectX
   Union of top hat and bottom hat.

Result

MorphHat returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

MorphHat is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

See also

Opening, Closing

Alternatives

TopHat, BottomHat, Union2

Module

Morphology

[out] HRegionX RegionSkeleton HRegionX.MorphSkeleton ( )

void HOperatorSetX.MorphSkeleton ([in] IObjectX Region,
[out] HUntypedObjectX RegionSkeleton )

Compute the morphological skeleton of a region.

MorphSkeleton computes the skeleton of the input regions (Region) using morphological transformations. The computation yields a disconnected skeleton (gaps in the diagonals) having a width of one or two pixels. The calculation uses the Golay element ‘h’, i.e., an 8-neighborhood. This is equivalent to the maximum-norm.

Parameter

▷ Region (input iconic) .............................. region(-array) \rightarrow HRegionX / IObjectX
   Regions to be processed.

▷ RegionSkeleton (output iconic) ..................... region(-array) \rightarrow HRegionX / HUntypedObjectX
   Resulting morphological skeleton.

Result

MorphSkeleton returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult',<RegionResult>)
- empty region: SetSystem('emptyRegionResult',<RegionResult>)
Otherwise, an exception is raised.

---

**Parallelization Information**

MorphSkeleton is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

---

**Possible Successors**

Skeleton, ReduceDomain, SelectShape, AreaCenter, Connection

---

See also

ThinningSeq, MorphSkiz

---

Alternatives

Skeleton, Thinning

---

Module

Morphology

---

```plaintext
[out] HRegionX RegionSkiz HRegionX.MorphSkiz ([in] VARIANT Iterations1, [in] VARIANT Iterations2 )
```

---

**Thinning of a region.**

MorphSkiz first performs a sequential thinning (ThinningSeq) of the input region with the element ‘l’ of the Golay alphabet. The number of iterations is determined by the parameter Iterations1. Then a sequential thinning of the resulting region with the element ‘e’ of the Golay alphabet is carried out. The number of iterations for this step is determined by the parameter Iterations2. The skiz operation serves to compute a kind of skeleton of the input regions, and to prune the branches of the resulting skeleton. If the skiz operation is applied to the complement of the region, the region and the resulting skeleton are separated.

If very large values or ‘maximal’ are passed for Iterations1 or Iterations2, the processing stops if no more changes occur.

---

**Parameter**

- **Region** (input iconic) ..........................region(-array)  θ  HRegionX / IHObjectX  Regions to be thinned.
- **RegionSkiz** (output iconic) .....................region(-array)  θ  HRegionX / HUntypedObjectX  Result of the skiz operator.
- **Iterations1** (input control) ......................integer  θ  VARIANT (integer, string )  Number of iterations for the sequential thinning with the element ‘l’ of the Golay alphabet.
  - Default Value : 100
  - Suggested values: Iterations1 ∈ {'maximal', 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200, 300, 400}
  - (lin)Minimum Increment : 1
  - Recommended Increment : 1
- **Iterations2** (input control) ......................integer  θ  VARIANT (integer, string )  Number of iterations for the sequential thinning with the element ‘e’ of the Golay alphabet.
  - Default Value : 1
  - Suggested values: Iterations2 ∈ {'maximal', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  - (lin)Minimum Increment : 1
  - Recommended Increment : 1

---

**Complexity**

Let \( F \) be the area of the input region. Then the runtime complexity for one region is

\[
O((\text{Iterations1} + \text{Iterations2}) \cdot 3 \cdot \sqrt{F}).
\]
Result

MorphSkiz returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

MorphSkiz is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm

Possible Successors

Pruning, ReduceDomain, SelectShape, AreaCenter, Connection, BackgroundSeg, Complement

See also

Thinning, HitOrMissSeq, Difference

Alternatives

Skeleton, ThinningSeq, MorphSkeleton, Interjacent

Module

Morphology

[out] HRegionX RegionOpening  
HRegionX.Opening

{[in] HRegionX StructElement  }

void HOperatorSetX.Opening ( [in] IObjectX Region,  
[in] IObjectX StructElement, [out] HUntypedObjectX RegionOpening )

Open a region.

An Opening operation is defined as an erosion followed by a Minkowski addition. By applying Opening to a region, larger structures remain mostly intact, while small structures like lines or points are eliminated. In contrast, a Closing operation results in small gaps being retained or filled up (see Closing).

Opening serves to eliminate small regions (smaller than StructElement) and to smooth the boundaries of a region. The position of StructElement is meaningless, since an opening operation is invariant with respect to the choice of the reference point.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

Parameter

- Region (input iconic) ............................................. region(-array) ~ HRegionX/IObjectX  
  Regions to be opened.
- StructElement (input iconic) ................................. region ~ HRegionX/IObjectX  
  Structuring element (position-invariant).
- RegionOpening (output iconic) .............................. region(-array) ~ HRegionX/HUntypedObjectX  
  Opened regions.

Example

/* Large regions in an aerial picture (beech trees or meadows): */
read_image(Image,’wald1’)  
threshold(Image,Light,80,255)  
gen_circle(StructElement1,100,100,2)  
gen_circle(StructElement2,100,100,20)  
/* close the small gap */
closing(Light,StructElement1,H)
/* selecting the large regions */
opening(H,StructElement2,Large).

/* Selecting of edges with certain orientation: */
read_image(Image,'fabrik')
sobel_amp(Image,Sobel,'sum_abs',3)
threshold(Sobel,Edges,10,255)
gen_rectangle2(StructElement,100,100,3.07819,20,1)
opening(Edges,StructElement,Direction).

**Complexity**

Let \( F_1 \) be the area of the input region, and \( F_2 \) be the area of the structuring element. Then the runtime complexity for one region is:

\[
O(2 \cdot \sqrt{F_1} \cdot \sqrt{F_2})
\]

**Result**

*Opening* returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

**Parallelization Information**

*Opening* is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

See also

GenCircle, GenRectangle2, GenRegionPolygon

**Alternatives**

MinkowskiAdd1, Erosion1, OpeningCircle

**Module**

Morphology

```plaintext
[out] HRegionX RegionOpening HRegionX.OpeningCircle
([in] VARIANT Radius )

void HOperatorSetX.OpeningCircle ([in] IObjectX Region,
[out] HUntypedObjectX RegionOpening, [in] VARIANT Radius )
```

Open a region with a circular structuring element.

*OpeningCircle* is defined as an erosion followed by a Minkowski addition with a circular structuring element (see example). *Opening* serves to eliminate small regions (smaller than the circular structuring element) and to smooth the boundaries of a region.
Parameter

- **Region** (input iconic) must be a **region(-array)** \(\sim HRegionX / IHObjectX\)
  - Regions to be opened.

- **RegionOpening** (output iconic) must be a **region(-array)** \(\sim HRegionX / HUntypedObjectX\)
  - Opened regions.

- **Radius** (input control) must be a **real** \(\sim\) VARIANT (real, integer)
  - Radius of the circular structuring element.

**Default Value:** 3.5

**Suggested values:** Radius \(\in\) \{1.5, 2.5, 3.5, 4.5, 5.5, 7.5, 9.5, 12.5, 15.5, 19.5, 25.5, 33.5, 45.5, 60.5, 110.5\}

**Typical range of values:** \(0.5 \leq \text{Radius} \leq 0.5\) (lin)

**Minimum Increment:** 1.0

**Recommended Increment:** 1.0

Example

/* Large regions in an aerial picture (beech trees or meadows): */
read_image(Image, ’wald1’)
threshold(Image, Light, 80, 255)
/* close the small gap */
closing_circle(LightH, 2)
/* selecting the large regions */
opening_circle(H, Large, 20).

Complexity

Let \(F\) be the area of the input region. Then the runtime complexity for one region is:

\[
O(4 \cdot \sqrt{F} \cdot \text{Radius})
\]

Result

**OpeningCircle** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem(’noObjectResult’,<RegionResult>)`
- empty region: `SetSystem(’emptyRegionResult’,<RegionResult>)`

Otherwise, an exception is raised.

Parallelization Information

**OpeningCircle** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

See also

TransposeRegion

Alternatives

Opening, Dilation1, MinkowskiAdd1, GenCircle

Module

```
[out] HRegionX RegionOpening \(\sim\) HRegionX.OpeningGolay
([in] String GolayElement, [in] long Rotation )

void HOperatorSetX.OpeningGolay ([in] IHObjectX Region,
[out] HUntypedObjectX RegionOpening, [in] VARIANT GolayElement,
[in] VARIANT Rotation )
```

Open a region with an element from the Golay alphabet.
OpeningGolay is defined as a Minkowski subtraction followed by a Minkowski addition. First the Minkowski subtraction of the input region (Region) with the structuring element from the Golay alphabet defined by GolayElement and Rotation is computed. Then the Minkowski addition of the result and the structuring element rotated by 180° is performed.


The rotation number Rotation determines which rotation of the element should be used, and whether the foreground (even) or background version (odd) of the selected element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

OpeningGolay serves to eliminate regions smaller than the structuring element, and to smooth regions’ boundaries.

Attention
Not all values of Rotation are valid for any Golay element. For some of the values of Rotation, the resulting regions are identical to the input regions.

### Parameter

<table>
<thead>
<tr>
<th>Region (input iconic)</th>
<th>region(-array)</th>
<th>HRegionX / IHObjectX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegionOpening (output iconic)</td>
<td>region(-array)</td>
<td>HRegionX / HUntypedObjectX</td>
</tr>
<tr>
<td>GolayElement (input control)</td>
<td>string</td>
<td>String / VARIANT</td>
</tr>
<tr>
<td>Rotation (input control)</td>
<td>integer</td>
<td>long / VARIANT</td>
</tr>
</tbody>
</table>

#### Default Value
- ’h’
- 0

#### List of values
- Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

### Complexity

Let $F$ be the area of an input region. Then the runtime complexity for one region is:

$$O(6 \cdot \sqrt{F})$$

### Result

OpeningGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

OpeningGolay is reentrant and automatically parallelized (on tuple level).

### Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNDimNorm

### Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

### See also

ErosionGolay, DilationGolay, ClosingGolay, HitOrMissGolay, ThinningGolay, ThickeningGolay, GolayElements

### Alternatives

OpeningSeg, Opening

### Module

Morphology
Open a region with a rectangular structuring element.

OpeningRectangle1 performs an ErosionRectangle1 followed by a DilationRectangle1 on the input region Region. The size of the rectangular structuring element is determined by the parameters Width and Height. As is the case for all Opening variants, larger structures are preserved, while small regions like lines or points are eliminated.

### Parameter

- **Region** (input iconic)  
  - Description: Regions to be opened.
  - Type: region(-array)  
  - Variant: HRegionX / IHObjectX

- **RegionOpening** (output iconic)  
  - Description:Opened regions.
  - Type: region(-array)  
  - Variant: HRegionX / HUntypedObjectX

- **Width** (input control)  
  - Description: Width of the structuring rectangle.
  - Default Value: 10
  - Suggested values: Width ∈ \{1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200\}
  - Typical range of values: 1 ≤ Width ≤ 1(lin)
  - Minimum Increment: 1
  - Recommended Increment: 1

- **Height** (input control)  
  - Description: Height of the structuring rectangle.
  - Default Value: 10
  - Suggested values: Height ∈ \{1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200\}
  - Typical range of values: 1 ≤ Height ≤ 1(lin)
  - Minimum Increment: 1
  - Recommended Increment: 1

### Complexity

Let \( F1 \) be the area of an input region and \( H \) be the height of the rectangle. Then the runtime complexity for one region is:

\[
O(2 \cdot \sqrt{F1} \cdot \log(H))
\]

### Result

OpeningRectangle1 returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem('noObjectResult,<RegionResult>)
- empty region: SetSystem('emptyRegionResult,<RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

OpeningRectangle1 is reentrant and automatically parallelized (on tuple level).

### Possible Predecessors

Threshold, RegionGrowing, Watersheds, ClassNDimNorm

### Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

### See also

OpeningSeg, OpeningGolay

### Alternatives

Opening, GenRectangle1, DilationRectangle1, ErosionRectangle1

### Module

Morphology
Separate overlapping regions.

The OpeningSeg operation is defined as a sequence of the following operators: Erosion\textsuperscript{1}, Connection and Dilation\textsuperscript{1} (see example). Only one iteration is done in Erosion\textsuperscript{1} and Dilation\textsuperscript{1}.

OpeningSeg serves to separate overlapping regions whose area of overlap is smaller than StructElement. It should be noted that the resulting regions can overlap without actually merging (see ExpandRegion). OpeningSeg uses the center of gravity as the reference point of the structuring element.

Structuring elements (StructElement) can be generated with operators such as GenCircle, GenRectangle\textsubscript{1}, GenRectangle\textsubscript{2}, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

### Parameter

- **Region** (input iconic) \(\rightarrow\) region(-array) \(\sim\) HRegionX / IHObjectX
  - Regions to be opened.
- **StructElement** (input iconic) \(\rightarrow\) region \(\sim\) HRegionX / IHObjectX
  - Structuring element (position-invariant).
- **RegionOpening** (output iconic) \(\rightarrow\) region \(\sim\) HRegionX / HUntypedObjectX
  - Opened regions.

### Example

```c
/* Simulation of opening_seg */
opening_seg(Region,StructElement,RegionOpening):
  erosion1(Region,StructElement,H1,1) >
  connection(H1,H2)
  dilation1(H2,StructElement,RegionOpening,1)
  clear_obj([H1,H2]).
```

### Complexity

Let \(F_1\) be the area of the input region, and \(F_2\) be the area of the structuring element. Then the runtime complexity for one region is:

\[ O(\sqrt{F_1} \cdot \sqrt{F_2} \cdot \sqrt{F_1}) \]

### Result

OpeningSeg returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

### Parallelization Information

OpeningSeg is reentrant and automatically parallelized (on tuple level).

### Possible Predecessors

- Threshold, RegionGrowing, Connection, Union\textsuperscript{1}, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle\textsubscript{1}, GenRectangle\textsubscript{2}, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

### Possible Successors

- ExpandRegion, ReduceDomain, SelectShape, AreaCenter, Connection
Chapter 8. Morphology

Alternatives

Erosion1, Connection, Dilation1

Morphology Module

[out] HRegionX RegionPrune HRegionX.Pruning ([in] long Length )

void HOperatorSetX.Pruning ([in] IHObjectX Region,
[out] HUntypedObjectX RegionPrune, [in] VARIANT Length )

Prune the branches of a region.

Pruning removes branches from a skeleton (Region) having a length less than Length. All other branches are preserved.

Parameter

▷ Region (input iconic) .................................. region(-array) ~ HRegionX / IHObjectX
Regions to be processed.

▷ RegionPrune (output iconic) ............................ region(-array) ~ HRegionX / HUntypedObjectX
Result of the pruning operation.

▷ Length (input control) .................................. integer ~ long / VARIANT
Length of the branches to be removed.

Default Value: 2

Suggested values: Length ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}

Typical range of values: 1 ≤ Length ≤ 1(lin)

Minimum Increment: 1

Recommended Increment: 1

Complexity

Let F be the area of the input region. Then the runtime complexity for one region is

\[ O(\text{Length} \cdot 3 \cdot \sqrt{F}) \].

Result

Pruning returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

Pruning is reentrant and automatically parallelized (on tuple level).

Possible Predecessors: MorphSkiz, Skeleton, ThinningSeq

Possible Successors: ReduceDomain, SelectShape, AreaCenter, Connection

See also: MorphSkeleton, JunctionsSkeleton

Module: Morphology

HALCON/COM Reference Manual, 2005-2-1
Add the result of a hit-or-miss operation to a region.

Thickening performs a thickening of the input regions using morphological operations. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMiss), and then adds the detected points to the input region. The parameter Iterations determines the number of iterations performed.

For the choice of the structuring elements StructElement1 and StructElement2, as well as for Row and Column, the same restrictions described under HitOrMiss apply.

The structuring elements (StructElement1 and StructElement2) can be generated by calling GolayElements, for example.

Attention
If the reference point is contained in StructElement1 the input region remains unchanged.

Parameter

- Region (input iconic) .................. region(-array) ~ > HRegionX / IHOBJECTX
  Regions to be processed.
- StructElement1 (input iconic) ........... region ~ > HRegionX / IHOBJECTX
  Structuring element for the foreground.
- StructElement2 (input iconic) ........... region ~ > HRegionX / IHOBJECTX
  Structuring element for the background.
- RegionThick (output iconic) ............. region(-array) ~ > HREGIONX / HUNTYPEDOBJECTX
  Result of the thickening operator.
- Row (input control) ..................... point.y ~ > long / VARIANT
  Row coordinate of the reference point.
  Default Value : 16
  Suggested values : Row ∈ {0, 2, 4, 8, 16, 32, 128}
  Typical range of values : 0 ≤ Row ≤ 0(lin)
  Minimum Increment : 1
  Recommended Increment : 1
- Column (input control) .................. point.x ~ > long / VARIANT
  Column coordinate of the reference point.
  Default Value : 16
  Suggested values : Column ∈ {0, 2, 4, 8, 16, 32, 128}
  Typical range of values : 0 ≤ Column ≤ 0(lin)
  Minimum Increment : 1
  Recommended Increment : 1
- Iterations (input control) ............... integer ~ > long / VARIANT
  Number of iterations.
  Default Value : 1
  Suggested values : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200, 400} (lin)
  Minimum Increment : 1
  Recommended Increment : 1

Complexity

Let \( F \) be the area of an input region, \( F_1 \) the area of the structuring element 1, and \( F_2 \) the area of the structuring element 2. Then the runtime complexity for one object is:

\[
O\left(\text{Iterations} \cdot \sqrt{F} \cdot \left(\sqrt{F_1} + \sqrt{F_2}\right)\right)
\]

Result

Thickening returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:
• no region: SetSystem('noObjectResult',<RegionResult>)
• empty region: SetSystem('emptyRegionResult',<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

Thickening is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

GolayElements, Threshold, Regiongrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

See also

HitOrMiss

Alternatives

ThickeningGolay, ThickeningSeq

---

**Module**

Morphology

---

[out] HRegionX RegionThick

HRegionX.ThickeningGolay

([in] String GolayElement, [in] long Rotation )

void HOperatorSetX.ThickeningGolay ([in] IHObjectX Region,

[out] HUntypedObjectX RegionThick, [in] VARIANT GolayElement,

[in] VARIANT Rotation )

---

Add the result of a hit-or-miss operation to a region (using a Golay structuring element).

ThickeningGolay performs a thickening of the input regions using morphological operations and structuring elements from the Golay alphabet. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMissGolay), and then adds the detected points to the input region. The following structuring elements are available:

'1', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

The rotation number Rotation determines which rotation of the element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

---

Attention

Not all values of Rotation are valid for any Golay element.

---

**Parameter**

▷ Region (input iconic) .......................... region(-array)  HRegionX / IHObjectX

Regions to be processed.

▷ RegionThick (output iconic) .................. region(-array)  HRegionX / HUntypedObjectX

Result of the thickening operator.

▷ GolayElement (input control) ............... string  String / VARIANT

Structuring element from the Golay alphabet.

Default Value : 'h'


▷ Rotation (input control) ........................ integer  long / VARIANT

Rotation of the Golay element. Depending on the element, not all rotations are valid.

Default Value : 0

List of values: Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

---

**Complexity**

Let F be the area of an input region. Then the runtime complexity for one region is:

\[ O(6 \cdot \sqrt{F}) \]
ThickeningGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`

Otherwise, an exception is raised.

**Parallelization Information**

ThickeningGolay is reentrant and automatically parallelized (on tuple level).

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

**See also**

ErosionGolay, HitOrMissGolay

**Alternatives**

Thickening, ThickeningSeq

**Module**

Morphology

---

**Add the result of a hit-or-miss operation to a region (sequential).**

ThickeningSeq calculates the sequential thickening of the input regions with a structuring element from the Golay alphabet (`GolayElement`). To do so, ThickeningSeq calls the operator ThickeningGolay with all possible rotations of the structuring element `Iterations` times. The following structuring elements are available:

' I', ' m', ' d', ' c', ' e', ' i', ' f', ' f2', ' h', ' k'.

The Golay elements, together with all possible rotations, are described with the operator `GolayElements`. For all elements of the Golay alphabet, except for 'c', the foreground and background masks are exchanged in order to have an effect for them on the outer boundary of the region. The element 'c' can be used to generate the convex hull of the input region if enough iterations are performed.

**Parameter**

- **Region** (input iconic) ................................... region(-array) ~ `HRegionX / IObjectX`
  Regions to be processed.
- **RegionThick** (output iconic) ......................... region(-array) ~ `HRegionX / HUntypedObjectX`
  Result of the thickening operator.
- **GolayElement** (input control) .......................... string ~ `String / VARIANT`
  Structuring element from the Golay alphabet.
  **Default Value:** 'h'
- **Iterations** (input control) ............................. integer ~ `long / VARIANT`
  Number of iterations.
  **Default Value:** 1
  **Suggested values:** `Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200}`
  *(lin)* `Minimum Increment : 1`
  *(lin)* `Recommended Increment : 1`

**Complexity**

Let $F$ be the area of an input region. Then the runtime complexity for one region is:

HALCON 6.1.4
**CHAPTER 8. MORPHOLOGY**

\[ O(\text{Iterations} \cdot 6 \cdot \sqrt{F}) \]

---

**Result**

ThickeningSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- **no region:** SetSystem(‘noObjectResult’,<RegionResult>)
- **empty region:** SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

---

**Parallelization Information**

ThickeningSeq is reentrant and automatically parallelized (on tuple level).

---

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

---

**See also**

ErosionGolay, ThinningSeq

---

**Alternatives**

ThickeningGolay, Thickening

---

**Module**

Morphology

---

**Remove the result of a hit-or-miss operation from a region.**

**Thinning** performs a thinning of the input regions using morphological operations. The operator first applies a hit-or-miss-transformation to Region (cf. HitOrMiss), and then removes the detected points from the input region. The parameter Iterations determines the number of iterations performed.

For the choice of the structuring elements StructElement1 and StructElement2, as well as for Row and Column, the same restrictions described under HitOrMiss apply.

Structuring elements (StructElement1, StructElement2) can be generated with operators such as GenCircle, GenRectangle1, GenRectangle2, GenEllipse, DrawRegion, GenRegionPolygon, GenRegionPoints, etc.

---

**Parameter**

- **Region** (input iconic) .......................... region(-array) \( \sim \) HRegionX / IHObjectX
  - Regions to be processed.
- **StructElement1** (input iconic) ...................... region \( \sim \) HRegionX / IHObjectX
  - Structuring element for the foreground.
- **StructElement2** (input iconic) ...................... region \( \sim \) HRegionX / IHObjectX
  - Structuring element for the background.
- **RegionThin** (output iconic) ....................... region(-array) \( \sim \) HRegionX / HUntypedObjectX
  - Result of the thinning operator.
- **Row** (input control) .............................. point.y \( \sim \) long / VARIANT
  - Row coordinate of the reference point.
  - **Default Value:** 0
  - **Typical range of values:** \( 0 \leq \text{Row} \leq 0(\text{lin}) \)
  - **Minimum Increment:** 1
  - **Recommended Increment:** 1

HALCON/COM Reference Manual, 2005-2-1
8.2. REGION

- **Column** (input control) ........................................ point.x $\rightarrow$ long / VARIANT
  - Column coordinate of the reference point.
  - Default Value: 0
  - Typical range of values: $0 \leq \text{Column} \leq 0$(lin)
  - Minimum Increment: 1
  - Recommended Increment: 1

- **Iterations** (input control) ...................................... integer $\rightarrow$ long / VARIANT
  - Number of iterations.
  - Default Value: 1
  - Suggested values: $\text{Iterations} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}$
  - Minimum Increment: 1
  - Recommended Increment: 1

--- Complexity

Let $F$ be the area of an input region, $F1$ the area of the structuring element 1, and $F2$ the area of the structuring element 2. Then the runtime complexity for one object is:

$$O\left(\text{Iterations} \cdot \sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right).$$

--- Result

**Thinning** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: `SetSystem('noObjectResult',<RegionResult>)`
- empty region: `SetSystem('emptyRegionResult',<RegionResult>)`

Otherwise, an exception is raised.

--- Parallelization Information

**Thinning** is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

--- Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

--- See also

HitOrMiss

--- Alternatives

ThinGolay, ThinSeq

--- Module

Morphology

--- Examples

```plaintext
[out] HRegionX RegionThin HRegionX.ThinningGolay
([in] String GolayElement, [in] long Rotation )
```

```plaintext
```

**Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).**

**ThinGolay** performs a thinning of the input regions using morphological operations and structuring elements from the Golay alphabet. The operator first applies a hit-or-miss-transformation to **Region** (cf. **HitOrMissGolay**), and then removes the detected points from the input region. The following structuring elements are available:

'$l'$, '$m'$, '$d'$, '$c$', '$e$', '$i$', '$f$', '$f2$', '$h$', '$k$'.

HALCON 6.1.4
The rotation number Rotation determines which rotation of the element should be used. The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Attention

Not all values of Rotation are valid for any Golay element.

Parameter

▷ Region (input iconic) ........................................ region(-array) ⇝ HRegionX / IHObjectX
  Regions to be processed.
▷ RegionThin (output iconic) .............................. region(-array) ⇝ HRegionX / HUntypedObjectX
  Result of the thinning operator.
▷ GolayElement (input control) .............................. string ⇝ String / VARIANT
  Structuring element from the Golay alphabet.
  Default Value : ’h’
  List of values : GolayElement ∈ {’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’}
▷ Rotation (input control) ........................................... integer ⇝ long / VARIANT
  Rotation of the Golay element. Depending on the element, not all rotations are valid.
  Default Value : 0
  List of values : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Complexity

Let \( F \) be the area of an input region. Then the runtime complexity for one region is:

\[
O(6 \cdot \sqrt{F})
\]

Result

ThinningGolay returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

• no region: SetSystem(’noObjectResult’,<RegionResult>)
• empty region: SetSystem(’emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

ThinningGolay is reentrant and automatically parallelized (on tuple level).

Possible Successors

ReduceDomain, SelectShape, AreaCenter, Connection

See also

ErosionGolay, HitOrMissGolay

Alternatives

ThinningSeq, Thinning

Module

Morphology

```c
[out] HRegionX RegionThin HRegionX.ThinningSeq
([in] String GolayElement, [in] VARIANT Iterations )

void HOperatorSetX.ThinningSeq ([in] IHObjectX Region,
[out] HUntypedObjectX RegionThin, [in] VARIANT GolayElement,
[in] VARIANT Iterations )
```

Remove the result of a hit-or-miss operation from a region (sequential).

ThinningSeq calculates the sequential thinning of the input regions with a structuring element from the Golay alphabet (GolayElement). To do so, ThinningSeq calls the operator ThinningGolay with all possible rotations of the structuring element Iterations times. If Iterations is chosen large enough, the operator calculates the skeleton of a region if the structuring elements ’l’ or ’m’ are used. For the element ’c’ the background and foreground are exchanged in order to have an effect on the interior boundary of a region. If a very large value or ’maximal’ is passed for Iterations the iteration stops if no more changes occur. The following structuring elements are available:
8.2. REGION

I'  Skeleton, similar to Skeleton. This structuring element is also used in MorphSkiz.

'm'  A skeleton with many “hairs” and multiple (parallel) branches.

d'  A skeleton without multiple branches, but with many gaps, similar to MorphSkeleton.

c'  Uniform erosion of the region.

e'  One pixel wide lines are shortened. This structuring element is also used in MorphSkiz.

'i'  Isolated points are removed. (Only Iterations = 1 is useful.)

'f'  Y-junctions are eliminated. (Only Iterations = 1 is useful.)

'f2'  One pixel long branches and corners are removed. (Only Iterations = 1 is useful.)

'h'  A kind of inner boundary, which, however, is thicker than the result of Boundary, is generated. (Only Iterations = 1 is useful.)

'k'  Junction points are eliminated, but also new ones are generated.

The Golay elements, together with all possible rotations, are described with the operator GolayElements.

Parameter

<table>
<thead>
<tr>
<th>Region</th>
<th>RegionThin</th>
<th>GolayElement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(input iconic)</td>
<td>(output iconic)</td>
<td>(input control)</td>
</tr>
<tr>
<td>region(-array)</td>
<td>region(-array)</td>
<td>string</td>
</tr>
<tr>
<td>HRegionX / IHObjectX</td>
<td>HRegionX / HUntypedObjectX</td>
<td>String / VARIANT</td>
</tr>
</tbody>
</table>

Default Value: 'l'
List of values: GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

Iterations (input control)  Number of iterations. For 'f', 'f2', 'h' and 'i' the only useful value is 1.

Default Value: 20
Suggested values: Iterations ∈ {'maximal', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200}

Minimum Increment: 1
Recommended Increment: 1

Complexity

Let $F$ be the area of an input region. Then the runtime complexity for one region is:

$$O(\text{Iterations} \cdot 6 \cdot \sqrt{F})$$

Result

ThinningSeq returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

Parallelization Information

ThinningSeq is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints,
GenStructElements, GenRegionPolygonFilled

Possible Successors

Pruning, ReduceDomain, SelectShape, AreaCenter, Connection, Complement

See also

HitOrMissSeq, ErosionGolay, Difference, ThinningGolay, ThickeningSeq

Alternatives

Skeleton, MorphSkiz, ExpandRegion

Module

Morphology

HALCON 6.1.4
CHAPTER 8. MORPHOLOGY

Compute the top hat of regions.

**TopHat** computes the **Opening** of **Region** with **StructElement**. The difference between the original region and the result of the opening is called the top hat. In contrast to **Opening**, which splits regions under certain circumstances, **TopHat** computes the regions removed by such a splitting.

The position of **StructElement** is meaningless, since an opening operation is invariant with respect to the choice of the reference point.

Structuring elements (**StructElement**) can be generated with operators such as **GenCircle**, **GenRectangle1**, **GenRectangle2**, **GenEllipse**, **DrawRegion**, **GenRegionPolygon**, **GenRegionPoints**, etc.

---

**Parameter**

- **Region** (input iconic) . . . . . . . . . . . . . region(-array) ~ HRegionX / IHObjectX Regions to be processed.
- **StructElement** (input iconic) . . . . . . . . . . . . . region ~ HRegionX / IHObjectX Structuring element (position independent).
- **RegionTopHat** (output iconic) . . . . . . . . . . . . . region(-array) ~ HRegionX / HUntypedObjectX Result of the top hat operator.

---

**Result**

**TopHat** returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: **SetSystem('noObjectResult',<RegionResult>)**
- empty region: **SetSystem('emptyRegionResult',<RegionResult>)**

Otherwise, an exception is raised.

---

**Parallelization Information**

**TopHat** is reentrant and automatically **parallelized** (on tuple level).

---

**Possible Predecessors**

Threshold, RegionGrowing, Connection, Union1, Watersheds, ClassNdimNorm, GenCircle, GenEllipse, GenRectangle1, GenRectangle2, DrawRegion, GenRegionPoints, GenStructElements, GenRegionPolygonFilled

---

**Possible Successors**

ReduceDomain, SelectShape, AreaCenter, Connection

---

**See also**

BottomHat, MorphHat, GrayTophat, Opening

---

**Alternatives**

Opening, Difference

---

**Module**

Morphology

HALCON/COM Reference Manual, 2005-2-1
Chapter 9

Object

9.1 Information

| [out] long Number HObjectX.CountObj ( ) |
| void HOperatorSetX.CountObj ([in] IObjectX Objects, |
| [out] VARIANT Number ) |

Number of objects in a tuple.

The operator CountObj determines for the object parameter Objects the number of objects it contains. In this connection it should be noted that object is not the same as connection component (see Connection). For example, the number of objects of a region not consisting of three connected parts is 1.

Attention

In Prolog and Lisp the length of the list is not necessarily identical with the number of objects. This is the case when object keys are contained which were created in the compact mode (keys from compact and normal mode can be used as a mixture). See in this connection SetSystem(::’compactObject’,<true/false>:).

Parameter

- Objects (input iconic) ........................................... object ~ IObjectX Objects to be examined.
- Number (output control) ........................................ integer ~ long / VARIANT Number of objects in the tuple Objects.

Complexity

Runtime complexity: \( O(|\text{Objects}|) \).

Result

If the surrogates are correct, i.e. all objects are present in the HALCON operator data base, the operator CountObj returns the value TRUE. The behavior in case of empty input (no input objects available) is set via the operator SetSystem(::’noObjectResult’,<Result>:).

Parallelization Information

CountObj is reentrant and processed without parallelization.

See also

CopyObj, ObjToInteger, Connection, SetSystem

Module

Basic operators
Informations about the components of an image object.

The operator `GetChannelInfo` gives information about the components of an image object. The following requests (`Request`) are currently possible:

- `creator` Output of the names of the procedures which initially created the image components (not the object).
- `type` Output of the type of image component (`byte`, `int1`, `int2`, `uint2`, `int4`, `real`, `direction`, `cyclic`, `complex`, `dvf`, `lut`). The component 0 is of type `region` or `xld`.

In the tuple `Channel` the numbers of the components about which information is required are stated. After carrying out `GetChannelInfo`, `Information` contains a tuple of strings (one string per entry in `Channel`) with the required information.

### Parameter

- **Object** (input iconic) .......................................................... `object` $\sim$ `IHObjectX`
  Image object to be examined.
- **Request** (input control) ......................................................... `string` $\sim$ `String / VARIANT`
  Required information about object components.
  - **Default Value:** `creator`
  - **List of values:** `Request` $\in \{ `creator`, `type` \}`
- **Channel** (input control) ....................................................... `channel(-array)` $\sim$ `VARIANT (integer)`
  Components to be examined (0 for region/XLD).
  - **Default Value:** 0
  - **Suggested values:** `Channel` $\in \{ 0, 1, 2, 3, 4, 5, 6, 7, 8 \}`
- **Information** (output control) .................................................. `string(-array)` $\sim$ `VARIANT (string)`
  Requested information.

### Result

If the parameters are correct the operator `GetChannelInfo` returns the value TRUE. Otherwise an exception is raised.

### Parallelization Information

`GetChannelInfo` is reentrant and processed without parallelization.

### Possible Predecessors

- **ReadImage**
  See also

### Module

Basic operators

```
[out] VARIANT Information HObjectX.GetChannelInfo ([in] String Request,
[in] VARIANT Channel )

void HOperatorSetX.GetChannelInfo ([in] IHObjectX Object,
```

**Name of the class of an image object.**

`GetObjectClass` returns the name of the corresponding class to each object. The following classes are possible:

- `image` Object with region (definition domain) and at least one channel.
- `region` Object with a region without gray values.
- `xld_cont` XLD object as contour
- `xld_poly` XLD object as polygon

HALCON/COM Reference Manual, 2005-2-1
'xld_parallel' XLD object with parallel polygons

Parameter

- **Object** (input iconic) 
  Image objects to be examined.
- **Class** (output control) 
  Name of class.

Result

If the parameter values are correct the operator `GetObjClass` returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

`GetObjClass` is *reentrant* and automatically parallelized (on tuple level).

Possible Successors

- DispImage, DispRegion, DispXld
- GetChannelInfo, CountRelation

Basic operators

```c
void HObjectX.TestEqualObj ([in] IHObjectX Objects2 )
void HOperatorSetX.TestEqualObj ([in] IHObjectX Objects1, [in] IHObjectX Objects2 )
```

Compare image objects regarding equality.

The operator `TestEqualObj` compares the regions and gray value components of all objects of the two input parameters. The n-th object in `Objects1` is compared to the n-th object in `Objects2` (for all n). If all corresponding regions are equal and the number of regions is also identical the operator `TestEqualObj` returns the value TRUE, otherwise FALSE.

Attention

Image matrices are not compared regarding their contents. Thus, two images are “equal” if they are at the same place in the store. If the input parameters are empty and the behavior was set via the operator `SetSystem (::'noObjectResult','true');`, the operator `TestEqualObj` returns TRUE, since all input (= empty set) is equal.

Parameter

- **Objects1** (input iconic) 
  Test objects.
- **Objects2** (input iconic) 
  Comparative objects.

Complexity

If $F$ is the area of a region the runtime complexity is $O(1)$ or $O(\sqrt{F})$ if the result is TRUE and $O(\sqrt{F})$ if the result is FALSE.

Result

The operator `TestEqualObj` returns the value TRUE if both object tuples are identical. If the tuples differ in at least one place `TestEqualObj` returns FALSE. The behavior in case of empty input (no input objects available) is set via the operator `SetSystem (::'noObjectResult','<Result>');?>. If the number of objects differs an exception is raised.

Parallelization Information

`TestEqualObj` is *reentrant* and processed *without* parallelization.

See also

- `TestEqualRegion`

Basic operators
Test whether the regions of two objects are identical.

The operator `TestEqualRegion` compares the regions of the two input parameters. The n-th element in `Regions1` is compared to the n-th object in `Regions2` (for all n). If all regions are equal and the number of regions is identical the operator `TestEqualRegion` returns the value TRUE, otherwise FALSE.

```c
void HRegionX.TestEqualRegion ([in] HRegionX Regions2 )
void HOperatorSetX.TestEqualRegion ([in] IHOBJECTX Regions1,
[in] IHOBJECTX Regions2 )
```

Test whether the regions of two objects are identical.

The operator `TestEqualRegion` compares the regions of the two input parameters. The n-th element in `Regions1` is compared to the n-th object in `Regions2` (for all n). If all regions are equal and the number of regions is identical the operator `TestEqualRegion` returns the value TRUE, otherwise FALSE.

### Attention

- **Parameter**
  - `Regions1` (input iconic) .......................... region(-array)  \(\leadsto\) HRegionX / IHOBJECTX
    - Test regions.
  - `Regions2` (input iconic) .......................... region(-array)  \(\leadsto\) HRegionX / IHOBJECTX
    - Comparative regions.

### Number of elements

\(\text{Regions}1 = \text{Regions}2\)

### Complexity

If \(F\) is the area of a region the runtime complexity is \(O(1)\) or \(O(\sqrt{F})\) if the result is TRUE, \(O(\sqrt{F})\) if the result is FALSE.

### Result

The operator `TestEqualRegion` returns the value TRUE if both object tuples are identical. If the tuples differ in at least one place `TestEqualRegion` returns FALSE. The behavior in case of empty input (no input objects available) is set via the operator `SetSystem(::'noObjectResult',<Result>;)`. If the number of objects differs an exception is raised.

**Parallelization Information**

`TestEqualRegion` is reentrant and processed without parallelization.

**See also**

**Alternatives**

Intersection, Complement, AreaCenter

**Module**

Basic operators

---

Test whether an object is already deleted.

The operator `TestObjDef` checks whether the object still exists in the HALCON operator data base (i.e. whether the surrogate is still valid). This check especially makes sense before deleting an object if it is not sure that the object has already been deleted by a prior deleting operator (`ClearObj`).

### Attention

The operator `TestObjDef` can return TRUE even if the object was already deleted because the surrogates of deleted objects are re-used for new objects. In this context see the example.

### Parameter

- `Object` (input iconic) .......................... object  \(\leadsto\) IHOBJECTX
  - Object to be checked.

### Complexity

The runtime complexity is \(O(1)\).

### Result

The operator `TestObjDef` returns the value TRUE if an object with this surrogate is present in the HALCON...
9.2 Manipulation

void HOperatorSetX.ClearObj ([in] IHObjectX Objects )

Delete an iconic object from the HALCON database. ClearObj deletes iconic objects, which are no longer needed, from the HALCON database. It should be noted that ClearObj is the only way to delete objects from the database, and hence to reclaim their memory, in all host languages except Smalltalk and C++.

Images and regions are normally used by several iconic objects at the same time (uses less memory!). This has the consequence that a region or an image is only deleted if all objects using it have been deleted. The operator ResetObjDb can be used to reset the system and clear all remaining iconic objects.

Attention

Regarding the use of local variables: Because only local variables are deleted on exit of a subroutine (or a rule in Prolog), while the HALCON database is not updated, it is necessary to clear local objects before exiting the subroutine. Special care has to be taken if backtracking is possible in Prolog (before reaching the ClearObj statements).

Parameter

Objects (input iconic) ...................... object(-array) \sim IHObjectX

Objects to be deleted.

Result

ClearObj returns TRUE if all objects are contained in the HALCON database. If not all objects are valid (e.g., already cleared), an exception is raised, which also clears all valid objects. The operator SetCheck (::‘clear’:) can be used to suppress the raising of this exception. If the input is empty the behavior can be set via SetSystem::‘noObjectResult’,<Result>:). If necessary, an exception is raised.

Parallelization Information

ClearObj is reentrant and processed without parallelization.

Possible Predecessors

TestObjDef, SetCheck

See also

TestObjDef, SetCheck

Alternatives

ResetObjDb

Module

Basic operators
Concatenate two iconic object tuples.

ConcataObj concatenates the two tuples of iconic objects Objects1 and Objects2 into a new object tuple ObjectsConcat. Hence, this tuple contains all the objects of the two input tuples:

\[
\text{ObjectsConcat} = \{\text{Objects1}, \text{Objects2}\}
\]

In ObjectsConcat the objects of Objects1 are stored first, followed by the objects of Objects2. The order of the objects is preserved. As usual, only the objects are copied, and not the corresponding images and regions, i.e., no new memory is allocated. ConcataObj is designed especially for HALCON/C. In languages like Smalltalk, Prolog, and C++ it is not needed.

ConcataObj should not be confused with Union1 or Union2, in which regions are merged, i.e., in which the number of objects is modified.

ConcataObj can be used to concatenate objects of different image object types (e.g., images and XLD contours) into a single object. This is only recommended if it is necessary to accumulate in a single object variable, for example, the results of an image processing sequence. It should be noted that the only operators that can handle such object tuples of mixed type are ConcataObj, COPYObj, SELECTObj, and DispObj. For technical reasons, object tuples of mixed type must not be created in HDDevelop.

---

**Parameter**

- **Objects1** (input iconic) \(\rightarrow\) IHObjectX Object tuple 1.
- **Objects2** (input iconic) \(\rightarrow\) IHObjectX Object tuple 2.
- **ObjectsConcat** (output iconic) \(\rightarrow\) HObjectX / HUntypedObjectX Concatenated objects.

---

**Complexity**

Runtime complexity: \(O(|\text{Objects1}| + |\text{Objects2}|)\);
Memory complexity of the result objects: \(O(|\text{Objects1}| + |\text{Objects2}|)\)

---

**Result**

ConcataObj returns TRUE if all objects are contained in the HALCON database. If the input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception is raised.

---

**Parallelization Information**

ConcataObj is reentrant and processed without parallelization.

---

**See also**

CountObj, CopyObj, SelectObj, DispObj

---

**Module**

Basic operators

---

Copy an iconic object in the HALCON database.

COPYObj copies NumObj iconic objects beginning with index Index (starting with 1) from the iconic input object tuple Objects to the output object ObjectsSelected. If -1 is passed for NumObj all objects beginning with Index are copied. No new storage is allocated for the regions and images. Instead, new objects containing
9.2. MANIPULATION

9.2.1. References to Objects

References to the existing objects are created. The number of objects in an object tuple can be queried with the operator `CountObj`.

Objective: To manipulate the existing objects and perform operations on them.

Parameter

- **Objects** (input iconic) ........................... `IHObjectX` Objects to be copied.
- **ObjectsSelected** (output iconic) .... `HObjectX` Copied objects.
- **Index** (input control) ............................ integer  
  **Default Value**: 1
  **Suggested values**: `Index ∈ {1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000}`
  **Restriction**: `Objects ≤ number`

- **NumObj** (input control) .......................... integer  
  **Default Value**: 1
  **Suggested values**: `NumObj ∈ {-1, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000}`
  **Restriction**:  
  
  \[ ([\text{Objects} ≤ \text{number}] ∧ (\text{NumObj} ≠ 0)) \]

Example

```c
/* Access all regions */
count_obj(Regions, Num)
for(1, Num, i)
    copy_obj(Regions, Single, i, 1)
    get_region_polygon(Single, 5.0, Line, Column)
    disp_polygon(WindowHandle, Line, Column)
clear_obj(Single)
loop().
```

Complexity

- **Runtime complexity**: \( O(|\text{Objects}| + \text{NumObj}) \)
- **Memory complexity of the result object**: \( O(\text{NumObj}) \)

Result

`CopyObj` returns `TRUE` if all objects are contained in the HALCON database and all parameters are correct. If the input is empty the behavior can be set via `SetSystem(::’noObjectResult’,<Result>:)`. If necessary, an exception is raised.

Parallelization Information

`CopyObj` is **reentrant** and processed **without** parallelization.

Possible Predecessors

**CountObj**

See also

**ConcatObj, ObjToInteger, CopyImage**

Alternatives

**SelectObj**

Module

Basic operators

```c
void HObjectX.GenEmptyObj ( )
void HOperatorSetX.GenEmptyObj ([out] HUntypedObjectX EmptyObject )
```

Create an empty object tuple.

The operator `GenEmptyObj` creates an empty tuple. This means that the output parameter does not contain any objects. Thus, the operator `CountObj` returns 0. However, `ClearObj` can be called for the output. It should
be noted that no objects must not be confused with an empty region. In case of an empty region, i.e. a region with 0 pixels, \texttt{CountObj} returns the value 1.

Parameter

\begin{itemize}
\item \texttt{EmptyObject} (output iconic) \texttt{... object \sim HObjectX / HUntypedObjectX}
\end{itemize}

No objects.

\begin{itemize}
\item \texttt{HObjectX / HUntypedObjectX}
\end{itemize}

Parallelization Information

\texttt{GenEmptyObj} is \textit{reentrant} and processed \textit{without} parallelization.

Module

\begin{itemize}
\item \texttt{Image / region / XLD management}
\end{itemize}

\begin{verbatim}
void HObjectX.IntegerToObj ([in] VARIANT SurrogateTuple )
void HOperatorSetX.IntegerToObj ([out] HUntypedObjectX Objects,
[in] VARIANT SurrogateTuple )
\end{verbatim}

Convert an “integer number” into an iconic object.

\texttt{IntegerToObj} is the inverse operator to \texttt{ObjToInteger}. All surrogates of objects passed in \texttt{SurrogateTuple} are stored as objects. In contrast to \texttt{ObjToInteger}, the objects are duplicated. \texttt{IntegerToObj} is intended especially for use in HALCON/C, because iconic objects and control parameters are treated differently in C.

The objects are duplicated in the database.

Parameter

\begin{itemize}
\item \texttt{Objects} (output iconic) \texttt{... object(-array) \sim HObjectX / HUntypedObjectX}
\end{itemize}

Created objects.

\begin{itemize}
\item \texttt{SurrogateTuple} (input control) \texttt{... integer(-array) \sim VARIANT ( integer )}
\end{itemize}

Tuple of object surrogates.

Result

\texttt{IntegerToObj} returns \texttt{TRUE} if all parameters are correct, i.e., if they are valid object keys. If the input is empty the behavior can be set via \texttt{SetSystem(::’noObjectResult’,<Result>:)}. If necessary, an exception is raised.

\texttt{IntegerToObj} is \textit{reentrant} and processed \textit{without} parallelization.

\begin{itemize}
\item \texttt{ObjToInteger}
\end{itemize}

See also

Module

\begin{itemize}
\item \texttt{Basic operators}
\end{itemize}

\begin{verbatim}
void HObjectX.ObjToInteger ([in] long Index,
[in] long Number )
void HOperatorSetX.ObjToInteger ([in] IHObjectX Objects,
\end{verbatim}

Convert an iconic object into an “integer number.”

\texttt{ObjToInteger} stores \texttt{Number}, starting at index \texttt{Index}, of the database keys of the input object \texttt{Objects} as integer numbers in the output parameter \texttt{SurrogateTuple}. If -1 is passed for \texttt{Number} all objects beginning with \texttt{Index} are copied. This facilitates a direct access to an arbitrary element of \texttt{Objects}. In conjunction with \texttt{CountObj} (returns the number of objects in \texttt{Objects}) the elements of \texttt{Objects} can be processed successively.

The objects are not duplicated by \texttt{ObjToInteger} and thus must not be cleared by \texttt{ClearObj}.

Attention

The objects’ data is not duplicated.
9.2. MANIPULATION

Parameter ⊲ Objects (input iconic) ...................... object(-array) \sim IHObjectX
Objects for which the surrogates are to be returned.

钐 Index (input control) ........................................ integer \sim long / VARIANT
Starting index of the surrogates to be returned.

钐 Default Value : 1

钐 Number (input control) .................................... integer \sim long / VARIANT
Number of surrogates to be returned.

钐 Default Value : -1
钐 Restriction : ((Number + Index) \lor (Objects \leq number))

钐 SurrogateTuple (output control) ...................... integer(-array) \sim VARIANT ( integer )
Tuple containing the surrogates.

Example

/* Access the i-th element: */
obj_to_integer(Objects,i,1,Surrogat).

Complexity

Runtime complexity: O(|Objects| + Number)

Result

ObjToInteger returns TRUE if all parameters are correct. If the input is empty the behavior can be set via
SetSystem::’noObjectResult’,<Result>::). If necessary, an exception is raised.

Parallelization Information

ObjToInteger is reentrant and processed without parallelization.

Possible Predecessors

TestObjDef
IntegerToObj, CountObj

Alternatives

CopyObj, SelectObj, CopyImage, GenImageProto

Basic operators

|out| HObjectX ObjectSelected HObjectX.SelectObj ([in] long Index )
void HOperatorSetX.SelectObj ([in] IHObjectX Objects, 
|out| HUntypedObjectX ObjectSelected, [in] VARIANT Index )

Select an object from an object tuple.

SelectObj copies the iconic object with index Index (starting with 1) from the iconic input object tuple
Objects to the output object ObjectSelected. No new storage is allocated for the regions and images.
Instead, new objects containing references to the existing objects are created. The number of objects in an object
tuple can be queried with the operator CountObj.

Parameter

钐 Objects (input iconic) ...................... object(-array) \sim IHObjectX
Objects, of which one is to be selected.

钐 ObjectSelected (output iconic) ..................... object \sim HObjectX / HUntypedObjectX
Selected object.

钐 Index (input control) .................................... integer \sim long / VARIANT
Index of the object to be selected.

钐 Default Value : 1
钐 Suggested values : Index \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 50, 100, 200, 500, 1000, 2000, 5000\}

HALCON 6.1.4
CHAPTER 9. OBJECT

Runtime complexity: $O(|\text{Objects}|)$

Result

SelectObj returns TRUE if all objects are contained in the HALCON database and all parameters are correct. If the input is empty the behavior can be set via SetSystem(::'noObjectResult',<Result>:). If necessary, an exception is raised.

Parallelization Information

SelectObj is reentrant and processed without parallelization.

Possible Predecessors

CountObj

See also

CountObj, ConcatObj, ObjToInteger

Alternatives

CopyObj

Module

Basic operators
Chapter 10

Regions

10.1 Access

```csharp
[out] long Row HRegionX.GetRegionChain ([out] long Column,
[out] VARIANT Chain )

void HOperatorSetX.GetRegionChain ([in] IHObjectX Region,
[out] VARIANT Row, [out] VARIANT Column, [out] VARIANT Chain )
```

Contour of an object as chain code.

The operator `GetRegionChain` returns the contour of a region. A contour is a series of pixels describing the outline of the region. The contour "lies on" the region. It starts at the smallest line number; in that line at the pixel with the largest column index. The rotation occurs clockwise. Holes of the region are ignored. The direction code (chain code) is defined as follows:

```
3 2 1
4 * 0
5 6 7
```

The operator `GetRegionChain` returns the code in the form of a tuple. In case of an empty region the parameters `Row` and `Column` are zero and `Chain` is the empty tuple.

Attention

Holes of the region are ignored. Only one region may be passed, and it must have exactly one connection component.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Region</td>
</tr>
<tr>
<td>Row</td>
</tr>
<tr>
<td>Column</td>
</tr>
<tr>
<td>Chain</td>
</tr>
</tbody>
</table>

Typical range of values: \(0 \leq \text{Chain} \leq 0\)

Result

The operator `GetRegionChain` normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem(‘noObjectResult’,<Result>).` The behavior in case of empty region (the region is the empty set) is set via the operator `SetSystem(‘emptyRegionResult’,<Result>).` If necessary an exception handling is raised.
Parallelization Information

GetRegionChain is reentrant and processed without parallelization.

Possible Predecessors

SobelAmp, Threshold, Skeleton, EdgesImage, GenRectangle1, GenCircle

Possible Successors

ApproxChain, ApproxChainSimple

See also

CopyObj, GetRegionContour, GetRegionPolygon

Module

Region processing

[out] VARIANT Rows HRegionX.GetRegionContour ([out] VARIANT Columns )

void HOperatorSetX.GetRegionContour ([in] IHObjectX Region, [out] VARIANT Rows, [out] VARIANT Columns )

Access the contour of an object.

The operator GetRegionContour returns the contour of a region. A contour is a result of line (Rows) and column coordinates (Columns), describing the boundary of the region. The contour lies on the region. It starts at the smallest line number. In that line at the pixel with the largest column index. The rotation direction is clockwise. The first pixel of the contour is identical with the last. Holes of the region are ignored. The operator GetRegionContour returns the coordinates in the form of tuples. An empty region is passed as empty tuple.

Attention

Holes of the region are ignored. Only one region may be passed, and this region must have exactly one connection component.

Parameter

▷ Region (input iconic) ........................................ region \( \rightarrow \) HRegionX / IHObjectX
Output region.

▷ Rows (output control) ........................................ \( \rightarrow \) VARIANT ( integer )
Line numbers of the contour pixels.

▷ Columns (output control) ................................. \( \rightarrow \) VARIANT ( integer )
Column numbers of the contour pixels.

Number of elements: \( (\text{Columns} = \text{Rows}) \)

Result

The operator GetRegionContour normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator SetSystem\(
\text{"noObjectResult","<Result>".}

Parallelization Information

GetRegionContour is reentrant and processed without parallelization.

Possible Predecessors

SobelAmp, Threshold, Skeleton, EdgesImage, GenRectangle1, GenCircle

See also

CopyObj, GetRegionChain, GetRegionPolygon

Module

Region processing

[out] VARIANT Rows HRegionX.GetRegionConvex ([out] VARIANT Columns )

void HOperatorSetX.GetRegionConvex ([in] IHObjectX Region, [out] VARIANT Rows, [out] VARIANT Columns )

Access convex hull as contour.
The operator `GetRegionConvex` returns the convex hull of a region as polygon. The polygon is the minimum result of line (\texttt{Rows}) and column coordinates (\texttt{Columns}) describing the hull of the region. The polygon pixels lie on the region. The polygon starts at the smallest line number; in this line at the pixel with the largest column index. The rotation direction is clockwise. The first pixel of the polygon is identical with the last. The operator `GetRegionConvex` returns the coordinates in the form of tuples. An empty region is passed as empty tuple.

```plaintext
Parameter

\begin{itemize}
\item \texttt{Region} (input iconic) \hspace{1cm} region \to HRegionX / IHObjectX
\end{itemize}

\begin{itemize}
\item \texttt{Rows} (output control) \hspace{1cm} \textcolor{red}{\texttt{contour.y}} \to \text{VARIANT} (\text{integer})
\end{itemize}

\begin{itemize}
\item \texttt{Columns} (output control) \hspace{1cm} \textcolor{red}{\texttt{contour.x}} \to \text{VARIANT} (\text{integer})
\end{itemize}

\textbf{Number of elements}: (\texttt{Columns} = \texttt{Rows})

Result

The operator `GetRegionConvex` returns the value TRUE.

Parallelization Information

`GetRegionConvex` is \textit{reentrant} and processed without parallelization.

Possible Predecessors

\texttt{Threshold}, \texttt{Skeleton}, \texttt{DynThreshold}

Possible Successors

\texttt{DispPolygon}

See also

\texttt{SelectObj}, \texttt{GetRegionContour}

Alternatives

\texttt{ShapeTrans}

Module

\texttt{Region processing}

```
void \texttt{HOperatorSetX.GetRegionPoints} ([in] IHObjectX Region, [out] VARIANT Rows, [out] VARIANT Columns)
```

Access the pixels of a region.

The operator `GetRegionPoints` returns the region data in the form of coordinate lists. The coordinates are sorted in the following order:

\[ (r_1, c_1) \leq (r_2, c_2) := (r_1 < r_2) \lor (r_1 = r_2) \land (c_1 \leq c_2) \]

`GetRegionPoints` returns the coordinates in the form of tuples. An empty region is passed as empty tuple.

Attention

Only one region may be passed.

Parameter

\begin{itemize}
\item \texttt{Region} (input iconic) \hspace{1cm} region \to HRegionX / IHObjectX
\end{itemize}

\begin{itemize}
\item \texttt{Rows} \hspace{1cm} \textcolor{red}{\texttt{coordinates.y}} \to \text{VARIANT} (\text{integer})
\end{itemize}

\begin{itemize}
\item \texttt{Columns} \hspace{1cm} \textcolor{red}{\texttt{coordinates.x}} \to \text{VARIANT} (\text{integer})
\end{itemize}

\textbf{Number of elements}: (\texttt{Columns} = \texttt{Rows})

HALCON 6.1.4
The operator **GetRegionPoints** normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem(‘noObjectResult’,<Result>).**

**Parallelization Information**

GetRegionPoints is reentrant and processed without parallelization.

**Possible Predecessors**

SobelAmp, Threshold, Connection

See also

CopyObj, GenRegionPoints

Alternatives

GetRegionRuns

Module

Region processing

```plaintext
[out] VARIANT Rows HRegionX.GetRegionPolygon ([in] VARIANT Tolerance, [out] VARIANT Columns )
```

Polygon approximation of a region.

The operator **GetRegionPolygon** calculates a polygon to approximate the edge of a region. A polygon is a sequence of line (Rows) and column coordinates (Columns). It describes the contour of the region. Only the base points of the polygon are returned. The parameter Tolerance indicates how large the maximum distance between the polygon and the edge of the region may be. Holes of the region are ignored. The operator GetRegionPolygon returns the coordinates in the form of tuples.

**Attention**

Holes of the region are ignored. Only one region may be passed, and this region must have exactly one connection component.

**Parameter**

- **Region** (input iconic) .......................................................... region \( \sim \) HRegionX / IObjectX
  Region to be approximated.
- **Tolerance** (input control) .................................................. number \( \sim \) VARIANT (integer, real)
  Maximum distance between the polygon and the edge of the region.
  - Default Value : 5.0
  - Suggested values : Tolerance \( \in \{0.0, 2.0, 5.0, 10.0\}\)
  - Minimum Increment : 0.01
  - Recommended Increment : 1.0
- **Rows** (output control) ...................................................... polygon.y \( \sim \) VARIANT (integer)
  Line numbers of the base points of the contour.
- **Columns** (output control) .................................................. polygon.x \( \sim \) VARIANT (integer)
  Column numbers of the base points of the contour.
  - Number of elements : (Columns = Rows)

The operator **GetRegionPolygon** normally returns the value TRUE. If more than one connection component is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem(‘noObjectResult’,<Result>).**

**Parallelization Information**

GetRegionPolygon is reentrant and processed without parallelization.

**Possible Predecessors**

SobelAmp, Threshold, Skeleton, EdgesImage
Access the runlength coding of a region.

The operator `GetRegionRuns` returns the region data in the form of chord tuples. The chord representation is caused by examining a region line by line with ascending line number (= from “top” to “bottom”). Every line is passed from left to right (ascending column number); storing all starting and ending points of region segments (= chords). Thus a region can be described by a sequence of chords, a chord being defined by line number, starting and ending points (column number). The operator `GetRegionRuns` returns the three components of the chords in the form of tuples. In case of an empty region three empty tuples are returned.

Only one region may be passed.

Parameter

- `Region` (input iconic) ................................. region \( \sim HRegionX / IObjectX \)
  Output region.
- `Row` (output control) ............................... chord.y \( \sim \) VARIANT (integer)
  Line numbers of the chords.
- `ColumnBegin` (output control) ...................... chord.x1 \( \sim \) VARIANT (integer)
  Column numbers of the starting points of the chords.
  **Number of elements:** \( (\text{ColumnBegin} = \text{Row}) \)
- `ColumnEnd` (output control) ......................... chord.x2 \( \sim \) VARIANT (integer)
  Column numbers of the ending points of the chords.
  **Number of elements:** \( (\text{ColumnEnd} = \text{Row}) \)

Result

The operator `GetRegionRuns` normally returns the value TRUE. If more than one region is passed an exception handling is caused. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem(‘noObjectResult’,<Result>).`

Parallelization Information

`GetRegionRuns` is reentrant and processed without parallelization.

Possible Predecessors

Threshold, Connection

See also

CopyObj, GenRegionRuns

Alternatives

GetRegionPoints

Module

Region processing

HALCON 6.1.4
10.2 Affine-Transformations

Apply an arbitrary affine 2D transformation to regions.

AffineTransRegion applies an arbitrary affine 2D transformation, i.e., scaling, rotation, translation, and slant (skewing), to the regions given in Region and returns the transformed regions in RegionAffineTrans. The affine transformation is described by the homogeneous transformation matrix given in HomMat2D, which can be created using the operators HomMat2dIdentity, HomMat2dScale, HomMat2dRotate, HomMat2dTranslate, etc., or be the result of operators like VectorAngleToRigid.

The components of the homogeneous transformation matrix are interpreted as follows: The row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system for the image. In particular, this assures that rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

The parameter Interpolate determines whether the transformation is to be done by using interpolation internally. This can lead to smoother region boundaries, especially if regions are enlarged. However, the runtime increases drastically.

AffineTransRegion in general is not reversible (clipping and discretization during rotation and scaling).

Result

AffineTransRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

Parallelization Information

AffineTransRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

HomMat2dIdentity, HomMat2dScale, HomMat2dTranslate, HomMat2dInvert, HomMat2dRotate

Possible Successors

SelectShape

See also

AffineTransImage

HALCON/COM Reference Manual, 2005-2-1
Alternatives

MoveRegion, MirrorRegion, ZoomRegion

Module

Region processing

```plaintext
[out] HRegionX RegionMirror
    HRegionX.MirrorRegion
    ([in] String RowColumn, [in] long WidthHeight )

void HOperatorSetX.MirrorRegion
    ([in] IHObjectX Region,
     [out] HUntypedObjectX RegionMirror, [in] VARIANT RowColumn,
     [in] VARIANT WidthHeight )
```

Reflect a region about the x- or y-axis.

**MirrorRegion** reflects a region about the x- or y-axis (parameter **RowColumn**). The parameter **WidthHeight** specifies the corresponding image width or height respectively, i.e., it is two times the coordinate of the axis of symmetry.

- **Parameter**
  - **Region** (input iconic)  region(-array)  ~ HRegionX / IHObjectX Region(s) to be reflected.
  - **RegionMirror** (output iconic)  region(-array)  ~ HRegionX / HUntypedObjectX Reflected region(s).
    - **Number of elements** : (RegionMirror = Region)
  - **RowColumn** (input control)  string  ~ String / VARIANT Coordinate of the axis of symmetry.
    - **Default Value** : 'row'
    - **List of values** : RowColumn ∈ {'column', 'row'}
  - **WidthHeight** (input control)  integer  ~ long / VARIANT Height or width of the corresponding image.
    - **Default Value** : 512
    - **Suggested values** : WidthHeight ∈ {128, 256, 512, 525, 768, 1024}
    - **Typical range of values** : 1 ≤ WidthHeight ≤ 1(lin)
    - **Minimum Increment** : 1
    - **Recommended Increment** : 1
    - **Restriction** : (WidthHeight > 0)

Parallelization Information

**MirrorRegion** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring

Possible Successors

SelectShape, DispRegion

See also

**ZoomRegion**

Alternatives

**AffineTransRegion**

Module

Region processing

```plaintext
[out] HRegionX RegionMoved
    HRegionX.MoveRegion
    ([in] long Row,
     [in] long Column )

void HOperatorSetX.MoveRegion
    ([in] IHObjectX Region,
```

Translate a region.
MoveRegion translates the input regions by the vector given by (Row, Column). If necessary, the resulting regions are clipped with the current image format.

Parameter

- **Region** (input iconic) .......................... region(-array) \(\rightarrow\) HRegionX / IHObjectX
  Region(s) to be moved.
- **RegionMoved** (output iconic) .................. region(-array) \(\rightarrow\) HRegionX / HUntypedObjectX
  Translated region(s).

**Number of elements** \(=\) (RegionMoved = Region)

- **Row** (input control) ............................ point.y \(\sim\) long / VARIANT
  Row coordinate of the vector by which the region is to be moved.
  Default Value : 30
  Suggested values : Row \(\in\) \{-128, -64, -32, -16, -10, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128\}
  Typical range of values : \(-512 \leq\) Row \(\leq\) -512 (lin)
  Minimum Increment : 1
  Recommended Increment : 10

- **Column** (input control) ........................ point.x \(\sim\) long / VARIANT
  Row coordinate of the vector by which the region is to be moved.
  Default Value : 30
  Suggested values : Column \(\in\) \{-128, -64, -32, -16, -10, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128\}
  Typical range of values : \(-512 \leq\) Column \(\leq\) -512 (lin)
  Minimum Increment : 1
  Recommended Increment : 10

**Complexity**

Let \(F\) be the area of the input region. Then the runtime complexity is \(O(F)\).

Result

MoveRegion always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via `SetSystem(’noObjectResult’,<Result>)`, the behavior in case of an empty input region via `SetSystem(’emptyRegionResult’,<Result>)`, and the behavior in case of an empty result region via `SetSystem(’storeEmptyRegion’,<true/false>)`. If necessary, an exception handling is raised.

**Parallelization Information**

MoveRegion is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successors**

SelectShape, DispRegion

See also

AffineTransImage, MirrorRegion, ZoomRegion

Module

Region processing

```c
[out] HRegionX Transposed HRegionX.TransposeRegion ([in] long Row,
[in] long Column )

void HOperatorSetX.TransposeRegion ([in] IHObjectX Region,
```

**Reflect a region about a point.**

TransposeRegion reflects a region about a point. The fixed point is given by Column and Row. The image \(P’\) of a point \(P\) is determined by the following requirement:

If \(P = S\), then \(P’ = S\), i.e., the point \(S\) is the fixed point of the mapping. If \(P \neq S\), \(S\) is the center point of a line segment connecting \(P\) and \(P’\). Therefore, the following equations result:

\[
\text{Column} = \frac{x + x’}{2}
\]
\[
\text{Row} = \frac{y + y'}{2}.
\]

If Row and Column are set to the origin, the in morphology often used transposition results. Hence TransposeRegion is often used to reflect (transpose) a structuring element.

\begin{itemize}
\item Region (input iconic) \( \Rightarrow \) HRegionX / IObjectX Region to be reflected.
\item Transposed (output iconic) \( \Rightarrow \) HRegionX / HUntypedObjectX Transposed region.
\item Row (input control) \( \Rightarrow \) long / VARIANT Row coordinate of the reference point.
\begin{itemize}
\item Default Value: 0
\item Suggested values: Row \( \in \) \{0, 64, 128, 256, 512\}
\item Typical range of values: \( 0 \leq \text{Row} \leq 0(\text{lin})\)
\item Minimum Increment: 1
\item Recommended Increment: 1
\end{itemize}
\end{itemize}

\begin{itemize}
\item Column (input control) \( \Rightarrow \) long / VARIANT Column coordinate of the reference point.
\begin{itemize}
\item Default Value: 0
\item Suggested values: Column \( \in \) \{0, 64, 128, 256, 512\}
\item Typical range of values: \( 0 \leq \text{Column} \leq 0(\text{lin})\)
\item Minimum Increment: 1
\item Recommended Increment: 1
\end{itemize}
\end{itemize}

\begin{itemize}
\item Complexity
\end{itemize}

Let \( F \) be the area of the input region. Then the runtime complexity for one region is

\[
O(\sqrt{F}).
\]

\begin{itemize}
\item Result
\end{itemize}

TransposeRegion returns TRUE if all parameters are correct. The behavior in case of empty or no input region can be set via:

- no region: SetSystem(‘noObjectResult’,<RegionResult>)
- empty region: SetSystem(‘emptyRegionResult’,<RegionResult>)

Otherwise, an exception is raised.

\begin{itemize}
\item Parallelization Information
\end{itemize}

TransposeRegion is reentrant and automatically parallelized (on tuple level).

\begin{itemize}
\item Possible Successors
\end{itemize}

ReduceDomain, SelectShape, AreaCenter, Connection

Dilation1, Opening, Closing

Morphology

\begin{verbatim}

Zoom a region.
\end{verbatim}
ZoomRegion enlarges or reduces the regions given in Region in the x- and y-direction by the given scale factors ScaleWidth and ScaleHeight.

Parameter

- **Region** (input iconic)  
  Region(s) to be zoomed.
- **RegionZoom** (output iconic)  
  Zoomed region(s).
  
  **Number of elements:** \( \text{RegionZoom} = \text{Region} \)
- **ScaleWidth** (input control)  
  Scale factor in x-direction.
  **Default Value:** 2.0  
  **Suggested values:** ScaleWidth \( \in \{0.25, 0.5, 1.0, 2.0, 3.0\} \)  
  **Typical range of values:** 0.0 \( \leq \) ScaleWidth \( \leq 0.0 \) (lin)  
  **Minimum Increment:** 0.01  
  **Recommended Increment:** 0.5
- **ScaleHeight** (input control)  
  Scale factor in y-direction.
  **Default Value:** 2.0  
  **Suggested values:** ScaleHeight \( \in \{0.25, 0.5, 1.0, 2.0, 3.0\} \)  
  **Typical range of values:** 0.0 \( \leq \) ScaleHeight \( \leq 0.0 \) (lin)  
  **Minimum Increment:** 0.01  
  **Recommended Increment:** 0.5

Parallelization Information

ZoomRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring

Possible Successors

SelectShape, DispRegion

See also

ZoomImageSize, ZoomImageFactor

Module

Region processing

10.3 Creation

```c
void HRegionX.GenCheckerRegion ( [in] long WidthRegion,
```

```c
void HOperatorSetX.GenCheckerRegion
( [out] HUntypedObjectX RegionChecker, [ in] VARIANT WidthRegion,
[ in] VARIANT HeightRegion, [ in] VARIANT WidthPattern,
[ in] VARIANT HeightPattern )
```

Create a checkered region.

The operator GenCheckerRegion returns a checkered region. Every black field of the checkerboard belongs to the region. The horizontal and vertical expansion of the region is limited by WidthRegion, HeightRegion respectively, the size of the fields of the checkerboard by WidthPattern \( \times \) HeightPattern.

Attention

If a very small pattern is chosen (WidthPattern < 4) the created region requires much storage.

Parameter

- **RegionChecker** (output iconic)  
  Created checkerboard region.
10.3. Creation

- **WidthRegion** (input control) .......................... extent.x $\rightarrow$ long / VARIANT
  Largest occurring $x$ value of the region.
  Default Value : 511
  Suggested values : $\text{WidthRegion} \in \{10, 20, 31, 63, 127, 255, 300, 400, 511\}$
  Typical range of values : $1 \leq \text{WidthRegion} \leq \text{lin}$
  Minimum Increment : 1
  Recommended Increment : 10
  Restriction : $(\text{WidthRegion} \geq 1)$

- **HeightRegion** (input control) .......................... extent.y $\rightarrow$ long / VARIANT
  Largest occurring $y$ value of the region.
  Default Value : 511
  Suggested values : $\text{HeightRegion} \in \{10, 20, 31, 63, 127, 255, 300, 400, 511\}$
  Typical range of values : $1 \leq \text{HeightRegion} \leq \text{lin}$
  Minimum Increment : 1
  Recommended Increment : 10
  Restriction : $(\text{HeightRegion} \geq 1)$

- **WidthPattern** (input control) .......................... extent.y $\rightarrow$ long / VARIANT
  Width of a field of the checkerboard.
  Default Value : 64
  Suggested values : $\text{WidthPattern} \in \{1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500\}$
  Typical range of values : $1 \leq \text{WidthPattern} \leq \text{lin}$
  Minimum Increment : 1
  Recommended Increment : 10
  Restriction : $(\text{WidthPattern} > 0 \land (\text{WidthPattern} < \text{WidthRegion}))$

- **HeightPattern** (input control) .......................... extent.y $\rightarrow$ long / VARIANT
  Height of a field of the checkerboard.
  Default Value : 64
  Suggested values : $\text{HeightPattern} \in \{1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500\}$
  Typical range of values : $1 \leq \text{HeightPattern} \leq \text{lin}$
  Minimum Increment : 1
  Recommended Increment : 10
  Restriction : $(\text{HeightPattern} > 0 \land (\text{HeightPattern} < \text{HeightRegion}))$

---

**Example**

```
gen_checker_region(Checker, 512, 512, 32, 64):  
set_draw(WindowHandle, 'fill')  
set_part(WindowHandle, 0, 0, 511, 511)  
disp_region(Checker, WindowHandle)
```

---

**Complexity**

The required storage (in bytes) for the region is:

$O((\text{WidthRegion} \times \text{HeightRegion})/\text{WidthPattern})$

---

**Result**

The operator `GenCheckerRegion` returns the value TRUE if the parameter values are correct. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator `SetSystem('clipRegion', <$\text{true'/'false'}$>)`.

---

**Parallelization Information**

`GenCheckerRegion` is reentrant and processed without parallelization.

---

**Possible Successors**

- PaintRegion

---

**See also**

- HammingChangeRegion, ReduceDomain

---

**Alternatives**

- GenGridRegion, GenRegionPolygonFilled, GenRegionPoints, GenRegionRuns, GenRectangle1, ConcatObj, GenRandomRegion, GenRandomRegions

HALCON 6.1.4
Create a circle.

The operator `HRegionX.GenCircle` generates one or more circles described by the center and `Radius`. If several circles shall be generated the coordinates must be passed in the form of tuples.

The coordinate system runs from (0,0) (upper left corner) to (Width-1,Height-1). See `GetSystem` and `ResetObjDb` in this context.

If an integer value (1,2,3...) is given for the radius the result is an even-numbered diameter and thus an asymmetrical circle. In case of an odd-numbered diameter (radius = 1.5,2.5,3.5...) a symmetrical circle is obtained.

If the circle extends beyond the image edge it is clipped to the current image format according to the value of the system flag `‘clip region’ (SetSystem)`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Suggested values</th>
<th>Typical range of values</th>
<th>Minimum Increment</th>
<th>Recommended Increment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Circle</code></td>
<td>Generated circle.</td>
<td>200.0</td>
<td><code>Row</code> ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}</td>
<td><code>1.0 ≤ Row ≤ 1.0(lin)</code></td>
<td>1.0</td>
<td>10.0</td>
</tr>
<tr>
<td><code>Row</code></td>
<td>Line index of center.</td>
<td>200.0</td>
<td><code>Column</code> ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}</td>
<td><code>1.0 ≤ Column ≤ 1.0(lin)</code></td>
<td>1.0</td>
<td>10.0</td>
</tr>
<tr>
<td><code>Column</code></td>
<td>Column index of center.</td>
<td>200.0</td>
<td><code>Radius</code> ∈ {1.0, 1.5, 2.0, 2.5, 3, 3.5, 4, 4.5, 5.5, 6.5, 7.5, 9.5, 11.5, 15.5, 20.5, 25.5, 31.5, 50.5}</td>
<td><code>1.0 ≤ Radius ≤ 1.0(lin)</code></td>
<td>1.0</td>
<td>10.0</td>
</tr>
<tr>
<td><code>Radius</code></td>
<td>Radius of circle.</td>
<td>100.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```plaintext
open_window(0,0,-1,-1,'root','visible','','',WindowHandle)
read_image(Image,’meer’)
gen_circle(Circle,300.0,200.0,150.5)
reduce_domain(Image,Circle,Mask)
disp_color(Mask,WindowHandle).
```

**Complexity**

- Runtime complexity: \(O(\text{Radius} \times 2)\)
- Storage complexity (byte): \(O(\text{Radius} \times 8)\)
Result
If the parameter values are correct, the operator GenCircle returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem ("clipRegion",<true/false>). If an empty region is created by clipping (the circle is completely outside of the image format) the operator SetSystem("storeEmptyRegion",<true/false>) determines whether the empty region is put out.

Parallelization Information

GenCircle is reentrant and processed without parallelization.

Possible Successors

PaintRegion, ReduceDomain

See also

DispCircle, SetShape, SmallestCircle, ReduceDomain

Alternatives

GenEllipse, GenRegionPolygonFilled, GenRegionPoints, GenRegionRuns, DrawCircle

Module

Region processing

Create an ellipse.

The operator GenEllipse generates one or more ellipses with the center (Row, Column), the orientation Phi and the half-radii Radius1 and Radius2. The angle is indicated in arc measure according to the x axis in mathematically positive direction. More than one region can be created by passing tuples of parameter values.

The center must be located within the image coordinates. The coordinate system runs from (0,0) (upper left corner) to (Width-1,Height-1). See GetSystem and ResetObjDb in this context. If the ellipse reaches beyond the edge of the image it is clipped to the current image format according to the value of the system flag 'clip region' (SetSystem).

Parameter

- **Ellipse** (output iconic) ........................ region(-array) טיבי HRegionX / HUntypedObjectX
  Created ellipse(s).

- **Row** (input control) ................................. ellipse.center.y(-array) 试管婴儿 VARIANT ( integer, real )
  Line index of center.
  **Default Value**: 200.0
  **Suggested values**: Row ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
  **Typical range of values**: 1.0 ≤ Row ≤ 1.0(lin)
  **Minimum Increment**: 1.0
  **Recommended Increment**: 10.0

- **Column** (input control) .............................. ellipse.center.x(-array) 试管婴儿 VARIANT ( integer, real )
  Column index of center.
  **Default Value**: 200.0
  **Suggested values**: Column ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
  **Typical range of values**: 1.0 ≤ Column ≤ 1.0(lin)
  **Minimum Increment**: 1.0
  **Recommended Increment**: 10.0
The orientation of the longer radius (Radius1) can be specified.

**Orientation of the longer radius (Radius1).**

**Default Value:** 0.0

**Suggested values:** \( \Phi \in \{-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097\} \)

**Typical range of values:** \(-1.178097 \leq \Phi \leq -0.392699 (\text{lin})\)

**Minimum Increment:** 0.01

**Recommended Increment:** 0.1

**Longer radius.**

**Default Value:** 100.0

**Suggested values:** \( \text{Radius1} \in \{2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0\} \)

**Typical range of values:** \(1.0 \leq \text{Radius1} \leq 1.0 (\text{lin})\)

**Minimum Increment:** 1.0

**Recommended Increment:** 10.0

**Shorter radius.**

**Default Value:** 60.0

**Suggested values:** \( \text{Radius2} \in \{1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0\} \)

**Typical range of values:** \(1.0 \leq \text{Radius2} \leq 1.0 (\text{lin})\)

**Minimum Increment:** 1.0

**Recommended Increment:** 10.0

**Restriction:** \((\text{Radius2} > 0) \land (\text{Radius2} \leq \text{Radius1})\)

---

**Example**

```
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
set_insert(WindowHandle,'xor')
repeat() 
    get_mbutton(WindowHandle,Row,Column,Button)
    gen_ellipse(Ellipse,Row,Column,Column / 300.0, 
                (Row mod 100)+1,(Column mod 50) + 1)
    disp_region(Ellipse,WindowHandle)
    clear_obj(Ellipse)
until(Button = 1)
```

---

**Complexity**

**Runtime complexity:** \(O(\text{Radius1} \times 2)\)

**Storage complexity (byte):** \(O(\text{Radius1} \times 8)\)

---

**Result**

If the parameter values are correct, the operator \texttt{GenEllipse} returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator \texttt{SetSystem (‘clipRegion’,<‘true’/‘false’>).}

---

**Parallelization Information**

\texttt{GenEllipse} is \textit{reentrant} and processed \textit{without} parallelization.

---

**Possible Successors**

\texttt{PaintRegion, ReduceDomain}

---

**See also**

\texttt{DispEllipse, SetShape, SmallestCircle, ReduceDomain}

---

**Alternatives**

\texttt{GenCircle, GenRegionPolygonFilled, DrawEllipse}

---

**Module**

Region processing
### 10.3. CREATION

```c
void HRegionX.GenEmptyRegion ( )
void HOperatorSetX.GenEmptyRegion ([out] HUntypedObjectX EmptyRegion )
```

Create an empty region.

The operator `GenEmptyRegion` creates an empty region. This means that the output parameter contains an object. Thus, `CountObj` returns 1. The area of the region is 0. Most of the shape features are undefined (0). It should be noted that an empty region must not be confused with the empty tuple.

```c
```

Create a region from lines or pixels.

The operator `GenGridRegion` creates a grid constructed of lines (Type = 'lines') or pixels (Type = 'points'). In case of 'lines' continuous lines are returned, in case of 'points' only the intersections of the lines. Starting from the pixel (0,0) to the pixel (Height-1,Width-1) the grid is built up at stepping width `RowSteps` in line direction and `ColumnSteps` in column direction. In the 'lines' mode `RowSteps`, `ColumnSteps` respectively, can be set to zero. In this case only columns, lines respectively, are created.

**Attention**

If a very small pattern is chosen (RowSteps < 4 oder ColumnSteps < 4) the created region requires much storage.

In the 'points' mode `RowSteps` and `ColumnSteps` must not be set to zero.

```c
void HRegionX.GenGridRegion ( )
void HOperatorSetX.GenGridRegion ([out] HUntypedObjectX EmptyRegion )
```

Create an empty region.

The operator `GenEmptyRegion` creates an empty region. This means that the output parameter contains an object. Thus, `CountObj` returns 1. The area of the region is 0. Most of the shape features are undefined (0). It should be noted that an empty region must not be confused with the empty tuple.

```c
```

Create a region from lines or pixels.

The operator `GenGridRegion` creates a grid constructed of lines (Type = 'lines') or pixels (Type = 'points'). In case of 'lines' continuous lines are returned, in case of 'points' only the intersections of the lines. Starting from the pixel (0,0) to the pixel (Height-1,Width-1) the grid is built up at stepping width `RowSteps` in line direction and `ColumnSteps` in column direction. In the 'lines' mode `RowSteps`, `ColumnSteps` respectively, can be set to zero. In this case only columns, lines respectively, are created.

**Attention**

If a very small pattern is chosen (RowSteps < 4 oder ColumnSteps < 4) the created region requires much storage.

In the 'points' mode `RowSteps` and `ColumnSteps` must not be set to zero.
CHAPTER 10. REGIONS

- **Type** (input control) ................................................................. string  \( \sim \) String / VARIANT
  
  Type of created pattern.
  
  Default Value : 'lines'
  
  List of values : Type \( \in \{ \text{'lines'}, \text{'points'} \} \)

- **Width** (input control) ............................................................... extent.x  \( \sim \) long / VARIANT
  
  Maximum width of pattern.
  
  Default Value : 512
  
  Suggested values : Width \( \in \{ 128, 256, 512, 1024 \} \)
  
  Typical range of values : \( 1 \leq \text{Width} \leq 1\text{(lin)} \)
  
  Minimum Increment : 1
  
  Recommended Increment : 10
  
  Restriction : \( (\text{Width} \geq 1) \)

- **Height** (input control) .............................................................. extent.y  \( \sim \) long / VARIANT
  
  Maximum height of pattern.
  
  Default Value : 512
  
  Suggested values : Height \( \in \{ 128, 256, 512, 1024 \} \)
  
  Typical range of values : \( 1 \leq \text{Height} \leq 1\text{(lin)} \)
  
  Minimum Increment : 1
  
  Recommended Increment : 10
  
  Restriction : \( (\text{Height} \geq 1) \)

---

**Example**

```plaintext
read_image(Image,'fabrik')
gen_grid_region(Raster,10,10,'lines',512,512)
reduce_domain(Image,Raster,Mask)
sobel_amp(Mask,GridSobel,'sum_abs',3)
disp_image(GridSobel,WindowHandle).
```

---

**Complexity**

The necessary storage (in bytes) for the region is:
\[ O((\text{ImageWidth}/\text{ColumnSteps}) \times (\text{ImageHeight}/\text{RowSteps})) \]

---

**Result**

If the parameter values are correct the operator **GenGridRegion** returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator **SetSystem** (’clipRegion’, <’true’/’false’>).

---

**Parallelization Information**

**GenGridRegion** is **reentrant** and processed **without** parallelization.

---

**Possible Successors**

ReduceDomain, PaintRegion

---

See also

GenCheckerRegion, ReduceDomain

---

**Alternatives**

GenRegionLine, GenRegionPolygon, GenRegionPoints, GenRegionRuns

---

**Module**

Region processing

---

```plaintext
void HRegionX.GenRandomRegion ([in] long Width, [in] long Height )

void HOperatorSetX.GenRandomRegion
([out] HUntypedObjectX RegionRandom, [in] VARIANT Width,
[in] VARIANT Height )
```

Create a random region.

The operator **GenRandomRegion** returns a random region. During this process every pixel in the image area \([0 \ldots \text{Width} - 1][0 \ldots \text{Height} - 1]\) is adapted into the region with the probability 0.5. The created region can be imagined as the threshold formation in an image with noise.
This procedure is particularly important for the creation of uncorrelated binary patterns. The random pattern is created by the C function “nrand48()”.

If \( \text{Width} \) and \( \text{Height} \) are chosen large (> 100) the created region may require much storage space due to the internally used runlength coding. The gray values of the output region are undefined.

**Parameter**

- **RegionRandom** (output iconic)

  Created random region with expansion \( \text{Width} \times \text{Height} \).

- **Width** (input control)

  Maximum horizontal expansion of random region.

  **Default Value**: 128

  **Suggested values**: \( \text{Width} \in \{16, 32, 50, 64, 100, 128, 256, 300, 400, 512\} \)

  **Typical range of values**: \( 1 \leq \text{Width} \leq 1 \text{ (lin)} \)

  **Minimum Increment**: 1

  **Recommended Increment**: 10

  **Restriction**: \((\text{Width} > 0)\)

- **Height** (input control)

  Maximum vertical expansion of random region.

  **Default Value**: 128

  **Suggested values**: \( \text{Height} \in \{16, 32, 50, 64, 100, 128, 256, 300, 400, 512\} \)

  **Typical range of values**: \( 1 \leq \text{Height} \leq 1 \text{ (lin)} \)

  **Minimum Increment**: 1

  **Recommended Increment**: 10

  **Restriction**: \((\text{Height} > 0)\)

**Complexity**

The worst case for the storage complexity for the created region (in byte) is: \( O(\text{Width} \times \text{Height} \times 2) \).

**Result**

If the parameter values are correct, the operator \texttt{GenRandomRegion} returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator \texttt{SetSystem ("clipRegion","true’/’false’}).

**Parallelization Information**

\texttt{GenRandomRegion} is reentrant and processed without parallelization.

**Possible Successors**

- \texttt{PaintRegion, ReduceDomain}

**See also**

- \texttt{GenCheckerRegion, HammingChangeRegion, AddNoiseDistribution, AddNoiseWhite, ReduceDomain}

**Module**

Region processing

```c
void HRegionX.GenRandomRegions ([in] String Type,
[in] VARIANT WidthMin, [in] VARIANT WidthMax, [in] VARIANT HeightMin,
[in] VARIANT HeightMax, [in] VARIANT PhiMin, [in] VARIANT PhiMax,
in long NumRegions, [in] long Width, [in] long Height )

void HOperatorSetX.GenRandomRegions ([out] HUntypedObjectX Regions,
in] TYPE Type, [in] VARIANT WidthMin, [in] VARIANT WidthMax,
in] VARIANT HeightMin, [in] VARIANT HeightMax, [in] VARIANT PhiMin,
in] VARIANT PhiMax, [in] VARIANT NumRegions, [in] VARIANT Width,
in] VARIANT Height )
```

Create random regions like circles, rectangles and ellipses.

The operator \texttt{GenRandomRegion} generates circles, rectangles and ellipses whose parameters are determined at random. In each case only one lower, upper limit respectively, is given. The position is always random and cannot be determined by parameters. The parameter \texttt{NumRegions} indicates how many regions shall be created.
Parameter

- **Regions** (output iconic) .......................... region \( \sim HRegionX / HUntypedObjectX \)
  Created regions.

- **Type** (input control) .......................... string \( \sim \) String / VARIANT
  Type of regions to be created.
  Default Value: 'circle'
  List of values: Type \( \in \{ 'circle', 'ring', 'ellipse', 'rectangle1', 'rectangle2' \} \)

- **WidthMin** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  Minimum width of the region.
  Default Value: 10.0
  Suggested values: WidthMin \( \in \{ 1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0 \} \)
  Typical range of values: \( 1.0 \leq \text{WidthMin} \leq 1.0 \)(lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0
  Restriction: (WidthMin > 0)

- **WidthMax** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  Maximum width of the region.
  Default Value: 20.0
  Suggested values: WidthMax \( \in \{ 1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0 \} \)
  Typical range of values: \( 1.0 \leq \text{WidthMax} \leq 1.0 \)(lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0
  Restriction: (WidthMax > 0)

- **HeightMin** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  Minimum height of the region.
  Default Value: 10.0
  Suggested values: HeightMin \( \in \{ 1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0 \} \)
  Typical range of values: \( 1.0 \leq \text{HeightMin} \leq 1.0 \)(lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0
  Restriction: (HeightMin > 0)

- **HeightMax** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  Maximum height of the region.
  Default Value: 30.0
  Suggested values: HeightMax \( \in \{ 1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0 \} \)
  Typical range of values: \( 1.0 \leq \text{HeightMax} \leq 1.0 \)(lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0
  Restriction: (HeightMax > 0)

- **PhiMin** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  Minimum rotation angle of the region.
  Default Value: -0.7854
  Suggested values: PhiMin \( \in \{ 0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5 \} \)
  Typical range of values: \( 0.0 \leq \text{PhiMin} \leq 0.0 \)(lin)
  Minimum Increment: 0.001
  Recommended Increment: 0.10
  Restriction: (PhiMin > 0)

- **PhiMax** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  Maximum rotation angle of the region.
  Default Value: 0.7854
  Suggested values: PhiMax \( \in \{ 0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5 \} \)
  Typical range of values: \( 0.0 \leq \text{PhiMax} \leq 0.0 \)(lin)
  Minimum Increment: 0.001
  Recommended Increment: 0.10
  Restriction: (PhiMax > 0)
NumRegions (input control) .................................................. integer  \( \sim \) long / VARIANT
Number of regions.
Default Value : 100
Suggested values : \( \text{NumRegions} \in \{1, 5, 20, 100, 200, 500, 1000, 2000\} \)
Typical range of values : \( 1 \leq \text{NumRegions} \leq 1\text{(lin)} \)
Minimum Increment : 1
Recommended Increment : 10
Restriction : (\( \text{NumRegions} > 0 \))

Width (input control) .................................................. integer  \( \sim \) long / VARIANT
Maximum horizontal expansion.
Default Value : 512
Suggested values : \( \text{Width} \in \{128, 256, 512, 1024\} \)
Typical range of values : \( 1 \leq \text{Width} \leq 1\text{(lin)} \)
Minimum Increment : 1
Recommended Increment : 10
Restriction : (\( \text{Width} > 0 \))

Height (input control) .................................................. integer  \( \sim \) long / VARIANT
Maximum vertical expansion.
Default Value : 512
Suggested values : \( \text{Height} \in \{128, 256, 512, 1024\} \)
Typical range of values : \( 1 \leq \text{Height} \leq 1\text{(lin)} \)
Minimum Increment : 1
Recommended Increment : 10
Restriction : (\( \text{Height} > 0 \))

Result
If the parameter values are correct \( \text{GenRandomRegions} \) returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is determined by the operator \( \text{SetSystem ('clipRegion', 'true'/false')} \).

Parallelization Information
\( \text{GenRandomRegions} \) is reentrant and processed without parallelization.

Possible Successors
PaintRegion

Module

Region processing


Create a rectangle parallel to the coordinate axes.
The operator \( \text{GenRectangle1} \) generates one or more rectangles parallel to the coordinate axes which are described by the upper left corner (\( \text{Row1, Column1} \)) and the lower right corner (\( \text{Row2, Column2} \)). More than one region can be created by passing a tuple of corner points. The coordinate system runs from (0,0) (upper left corner) to (Width-1,Height-1). See \( \text{GetSystem} \) and \( \text{ResetObjDb} \) in this context.

Parameter

| \( \triangleright \) **Rectangle** (output iconic) .................................................. region(-array)  \( \sim \) HRegionX / HUntypedObjectX
Created rectangle. |
| \( \triangleright \) **Row1** (input control) .................................................. rectangle.origin.y(-array)  \( \sim \) VARIANT ( integer, real )
Line of upper left corner point. |
Default Value : 30.0
Suggested values : \( \text{Row1} \in \{0.0, 10.0, 20.0, 50.0, 100.0, 200.0\} \)
Minimum Increment : 1.0
Recommended Increment : 10.0 |
CHAPTER 10. REGIONS

- Column1 (input control) \[\text{rectangle}.\text{origin}.x(-array) \sim \text{VARIANT} (\text{integer, real})\]
  
  Column of upper left corner point.
  
  Default Value: 20.0
  
  Suggested values: Column1 $\in$ \{0.0, 10.0, 20.0, 50.0, 100.0, 200.0\}
  
  (lin) Minimum Increment: 1.0
  
  Recommended Increment: 10.0

- Row2 (input control) \[\text{rectangle}.\text{corner}.y(-array) \sim \text{VARIANT} (\text{integer, real})\]
  
  Line of lower right corner point.
  
  Default Value: 100.0
  
  Suggested values: Row2 $\in$ \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0\}
  
  (lin) Minimum Increment: 1.0
  
  Recommended Increment: 10.0
  
  Restriction: \((\text{Row2} \geq \text{Row1})\)

- Column2 (input control) \[\text{rectangle}.\text{corner}.x(-array) \sim \text{VARIANT} (\text{integer, real})\]
  
  Column of lower right corner point.
  
  Default Value: 200.0
  
  Suggested values: Column2 $\in$ \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0\}
  
  (lin) Minimum Increment: 1.0
  
  Recommended Increment: 10.0
  
  Restriction: \((\text{Column2} \geq \text{Column1})\)

---

Example

```c
/* Contrast improvement in a rectangular region of interest */

read_image(Image,’fabrik’);
open_window(0,0,-1,-1,’root’,’visible’,’’,WindowHandle);
disp_image(Image,WindowHandle);
draw_rectangle1(WindowHandle,Row1,Column1,Row2,Column2);
gen_rectangle1(Rectangle,Row1,Column1,Row2,Column2);
reduce_domain(Image,Rectangle,Mask);
emphasize(Mask,Emphasize,9,9,1.0);
disp_image(Emphasize,WindowHandle).
```

---

Result

If the parameter values are correct, the operator GenRectangle1 returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem (‘clipRegion’,‘true’/’false’).

---

Parallelization Information

GenRectangle1 is reentrant and processed without parallelization.

---

Possible Successors

PaintRegion, ReduceDomain

---

See also

DrawRectangle1, ReduceDomain, SmallestRectangle1

---

Alternatives

GenRectangle2, GenRegionPolygon, FillUp, GenRegionRuns, GenRegionPoints, GenRegionLine

---

Module

Region processing

---

```c
void HRegionX.GenRectangle2 (\[in\] VARIANT Row, \[in\] VARIANT Column, \[in\] VARIANT Phi, \[in\] VARIANT Length1, \[in\] VARIANT Length2 )

void HOperatorSetX.GenRectangle2 (\[out\] HUntypedObjectX Rectangle, \[in\] VARIANT Row, \[in\] VARIANT Column, \[in\] VARIANT Phi, \[in\] VARIANT Length1, \[in\] VARIANT Length2 )
```

Create a rectangle of any orientation.
The operator `GenRectangle2` generates one or more rectangles with the center \((Row, Column)\), the orientation \(\Phi\) and the half edge lengths \(Length1\) and \(Length2\). The orientation is given in arc measure and indicates the angle between the horizontal axis and \(Length1\) (mathematically positive). The coordinate system runs from \((0,0)\) (upper left corner) to \((Width-1,Height-1)\). See `GetSystem` and `ResetObjDb` in this context. More than one region can be created by passing one tuple of corner points.

Attention

The gray values of the output objects are undefined.

Parameter

\- `Rectangle` (output iconic) \(\rightarrow\) \(HRegionX / HUnTypedObjectX\)

- Created rectangle.

\- `Row` (input control) \(\rightarrow\) \(VARIANT\) (integer, real)

- Line index of the center.
  - Default Value : 50.0
  - Suggested values : \(Row \in \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}\)
  - Minimum Increment : 1.0
  - Recommended Increment : 10.0

\- `Column` (input control) \(\rightarrow\) \(VARIANT\) (integer, real)

- Column index of the center.
  - Default Value : 100.0
  - Suggested values : \(Column \in \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}\)
  - Minimum Increment : 1.0
  - Recommended Increment : 10.0

\- `Phi` (input control) \(\rightarrow\) \(VARIANT\) (integer, real)

- Angle of longitudinal axis to the horizontal (in radians).
  - Default Value : 0.0
  - Suggested values : \(Phi \in \{-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097\}\)
  - Minimum Increment : 0.001
  - Recommended Increment : 0.1
  - Restriction : \(((\neg (\pi / 2) < Phi) \land (Phi \leq (\pi / 2)))\)

\- `Length1` (input control) \(\rightarrow\) \(VARIANT\) (integer, real)

- Half width.
  - Default Value : 200.0
  - Suggested values : \(Length1 \in \{3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0\}\)
  - Minimum Increment : 1.0
  - Recommended Increment : 10.0

\- `Length2` (input control) \(\rightarrow\) \(VARIANT\) (integer, real)

- Half height.
  - Default Value : 100.0
  - Suggested values : \(Length2 \in \{1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0\}\)
  - Minimum Increment : 1.0
  - Recommended Increment : 10.0
  - Restriction : \((Length2 \leq Length1)\)

Result

- If the parameter values are correct the operator `GenRectangle2` returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator `SetSystem` (‘clipRegion’,<’true’/’false’>).

Parallelization Information

- `GenRectangle2` is reentrant and processed without parallelization.

Possible Successors

- `PaintRegion`, `ReduceDomain`

See also

- `DrawRectangle2`, `ReduceDomain`, `SmallestRectangle2`, `GenEllipse`

Alternatives

- `GenRectangle1`, `GenRegionPolygonFilled`, `GenRegionPolygon`, `GenRegionPoints`, `FillUp`
Module Region processing

```
[out] HRegionX Region  HXLDContX.GenRegionContourXld ([in] String Mode )
void HOperatorSetX.GenRegionContourXld ([in] IObjectX Contour, 
[out] HUntypedObjectX Region, [in] VARIANT Mode )
```

Create a region from an XLD contour.

GenRegionContourXld creates a region Region from a subpixel XLD contour Contour. The contour is sampled according to the Bresenham algorithm and influenced by the parameter neighborhood of the operator SetSystem. Open contours are closed before converting them to regions. Finally, the parameter Mode defines whether the region is filled up (filled) or returned by its contour (margin).

Parameter

▶ Contour (input iconic) ................................. xld_cont  ~/ HXLDContX / IObjectX
Input contour.
▶ Region (output iconic) ...................................region  ~/ HRegionX / HUntypedObjectX
Created region.
▶ Mode (input control) ....................................string  ~/ String / VARIANT
Fill mode of the region.
**Default Value:** 'filled'
**Suggested values:** Mode ∈ {‘filled’, ‘margin’}

Parallelization Information

GenRegionContourXld is reentrant and processed without parallelization.

Possible Predecessors

GenContourPolygonXld, GenContourPolygonRoundedXld

See also

SetSystem

Alternatives

GenRegionPolygon, GenRegionPolygonXld

Sub-pixel operators

```
void HRegionX.GenRegionHisto ([in] VARIANT Histogram, [in] long Row, 
[in] long Column, [in] long Scale )

void HOperatorSetX.GenRegionHisto ([out] HUntypedObjectX Region, 
[in] VARIANT Scale )
```

Convert a histogram into a region.

GenRegionHisto converts a histogram created with GrayHisto into a region. The effect of the three control parameters is the same as in DispImage and SetPaint.

Parameter

▶ Region (output iconic) ................................ region  ~/ HRegionX / HUntypedObjectX
Region containing the histogram.
▶ Histogram (input control) ..............................histogram  ~/ VARIANT ( integer )
Input histogram.
10.3. CREATION

- **Row** (input control) \[\text{point.y} \sim \text{long / VARIANT}\]
  - Row coordinate of the center of the histogram.
  - Default Value: 255
  - Suggested values: \(\text{Row} \in \{100, 200, 255, 300, 400\}\)
  - Typical range of values: \(0 \leq \text{Row} \leq 0\)

- **Column** (input control) \[\text{point.x} \sim \text{long / VARIANT}\]
  - Column coordinate of the center of the histogram.
  - Default Value: 255
  - Suggested values: \(\text{Column} \in \{100, 200, 255, 300, 400\}\)
  - Typical range of values: \(0 \leq \text{Column} \leq 0\)

- **Scale** (input control) \[\text{integer} \sim \text{long / VARIANT}\]
  - Scale factor for the histogram.
  - Default Value: 1
  - List of values: \(\text{Scale} \in \{1, 2, 3, 4, 5, 6, 7\}\)
  - Typical range of values: \(1 \leq \text{Scale} \leq 1\) (lin)
  - Minimum Increment: 1
  - Recommended Increment: 1

\[\text{Result} \]

GenRegionHisto returns TRUE if all parameters are correct. If necessary, an exception handling is raised.

\[\text{Parallelization Information} \]

GenRegionHisto is reentrant and processed without parallelization.

\[\text{Possible Predecessors} \]

GenRegionHisto

\[\text{See also} \]

DispChannel, SetPaint

\[\text{Module} \]

Region processing

\begin{verbatim}
void HRegionX.GenRegionHline ([in] VARIANT Orientation, [in] VARIANT Distance )
\end{verbatim}

Store input lines described in Hesse normal shape as regions.

The operator GenRegionHline stores the lines described in Hesse normal shape as regions. A line is determined by the distance from the line to the origin (Distance, corresponds to the length of the normal vector) and the direction of the normal vector (Orientation, corresponds to the orientation of the line \(\pm \pi/2\)). The directions were defined in such a way that at Orientation = 0 the normal vector lies in the direction of the X axis, which corresponds to a vertical line. At Orientation = \(\pi/2\) the normal vector points in the direction of the Y axis, i.e. a horizontal line is described.

\[\text{Attention} \]

The lines are clipped to the current maximum image format.

\[\text{Parameter} \]

- **Regions** (output iconic) \[\text{region(-array)} \sim \text{HRegionX / HUntypedObjectX}\]
  - Created regions (one for every line), clipped to maximum image format.
  - Number of elements: \((\text{Regions} = \text{Distance})\)

- **Orientation** (input control) \[\text{hesseline.angle.rad(-array)} \sim \text{VARIANT (integer, real )}\]
  - Orientation of the normal vector in radians.
  - Default Value: 0.0
  - Suggested values: \(\text{Orientation} \in \{-0.78, 0.0, 0.78, 1.57\}\)
  - (lin)Recommended Increment: 0.02
  - Number of elements: \((\text{Orientation} = \text{Distance})\)
Distance (input control) .................. hessline.distance(-array)  \(\sim\) VARIANT (integer, real)
Distance from the line to the coordinate origin (0.0).
Default Value : 200
Suggested values : Distance \(\in\) \{10, 50, 100, 200, 300, 400\}
\(\text{(lin)}\) Recommended Increment : 1

Result
The operator GenRegionHline always returns the value TRUE.

Parallelization Information
GenRegionHline is reentrant and processed without parallelization.

See also
HoughLines
Alternatives
GenRegionLine
Module
Region processing

\begin{verbatim}

\end{verbatim}

Store input lines as regions.
The operator GenRegionLine stores the given lines (with starting point \([\text{BeginRow}, \text{BeginCol}]\) and ending point \([\text{EndRow}, \text{EndCol}]\)) as region.

Attention

Parameter

\begin{itemize}
\item RegionLines (output iconic) ......................... region(-array)  \(\sim\) HRegionX/ HUntypedObjectX
Created regions.
\item BeginRow (input control) .......................... line.begin.y(-array)  \(\sim\) VARIANT (integer)
Line coordinates of the starting points of the input lines.
Default Value : 100
Suggested values : BeginRow \(\in\) \{10, 50, 100, 200, 300, 400\}
\(\text{(lin)}\) Minimum Increment : 1
Recommended Increment : 1
\item BeginCol (input control) .......................... line.begin.x(-array)  \(\sim\) VARIANT (integer)
Column coordinates of the starting points of the input lines.
Default Value : 50
Suggested values : BeginCol \(\in\) \{10, 50, 100, 200, 300, 400\}
\(\text{(lin)}\) Minimum Increment : 1
Recommended Increment : 1
\item EndRow (input control) ........................... line.end.y(-array)  \(\sim\) VARIANT (integer)
Line coordinates of the ending points of the input lines.
Default Value : 150
Suggested values : EndRow \(\in\) \{50, 100, 200, 300, 400, 500\}
\(\text{(lin)}\) Minimum Increment : 1
Recommended Increment : 1
\item EndCol (input control) ........................... line.end.x(-array)  \(\sim\) VARIANT (integer)
Column coordinates of the ending points of the input lines.
Default Value : 250
Suggested values : EndCol \(\in\) \{50, 100, 200, 300, 400, 500\}
\(\text{(lin)}\) Minimum Increment : 1
Recommended Increment : 1
\end{itemize}
The operator `GenRegionLine` always returns the value TRUE. The clipping according to the current image format is determined by the operator `SetSystem('clipRegion',<true/false>).` `GenRegionLine` is reentrant and processed without parallelization.

### Parallelization Information
SplitSkeletonLines

### Possible Predecessors
GenRegionHline

### Module
Region processing

```c
void HRegionX.GenRegionPoints ([in] VARIANT Rows,
[in] VARIANT Columns )
```

```c
void HOperatorSetX.GenRegionPoints ([out] HUntypedObjectX Region,
```

Store individual pixels as image region.

The operator `GenRegionPoints` creates a region described by a number of pixels. The pixels do not have to be stored in a fixed order, but the best runtime behavior is obtained when the pixels are stored in ascending order. The order is as follows:

\[(l_1, c_1) \leq (l_2, c_2) \ := \ (l_1 < l_2) \lor (l_1 = l_2) \land (c_1 \leq c_2)\]

The indicated coordinates stand for two consecutive pixels in the tuple.

#### Attention

- **Region** (output iconic) \( \rightsquigarrow \) HRegionX / HUntypedObjectX
  Created region.
- **Rows** (input control) \( \rightsquigarrow \) VARIANT ( integer )
  Lines of the pixels in the region.
  - Default Value : 100
  - Suggested values : Rows \( \in \) \{0, 10, 30, 50, 100, 200, 300, 500\}
  - (lin)Minimum Increment : 1
  - Recommended Increment : 1
- **Columns** (input control) \( \rightsquigarrow \) VARIANT ( integer )
  Columns of the pixels in the region.
  - Default Value : 100
  - Suggested values : Columns \( \in \) \{0, 10, 30, 50, 100, 200, 300, 500\}
  - (lin)Minimum Increment : 1
  - Recommended Increment : 1
  - Number of elements : (Columns = Rows)

#### Complexity

\( F \) shall be the number of pixels. If the pixels are sorted in ascending order the runtime complexity is: \( O(F) \), otherwise \( O(\log(F) \cdot F) \).

#### Result

The operator `GenRegionPoints` returns the value TRUE if the pixels are located within the image format. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator `SetSystem('clipRegion',<true/false>).` If an empty region is created (by the clipping or by an empty input) the operator `SetSystem('storeEmptyRegion',<true/false>)` determines whether the region is returned or an empty object tuple.
Parallelization Information

GenRegionPoints is reentrant, local, and processed without parallelization.

Possible Predecessors

GetRegionPoints

Possible Successors

PaintRegion, ReduceDomain

See also

ReduceDomain

Alternatives

GenRegionPolygon, GenRegionRuns, GenRegionLine

Module

Region processing

void HRegionX.GenRegionPolygon ([in] VARIANT Rows,
[in] VARIANT Columns )

void HOperatorSetX.GenRegionPolygon ([out] HUntypedObjectX Region,

Store a polygon as an image object.

The operator GenRegionPolygon creates a region from a polygon row described by a series of line and column coordinates. The created region consists of the pixels of the routes defined thereby, wherein it is linearly interpolated between the base points.

Attention

The region is not automatically closed and not filled. The gray values of the output regions are undefined.

Parameter

▷ Region (output iconic) .................................................... region \sim HRegionX / HUntypedObjectX
Created region.

▷ Rows (input control) ...................................................... polygon.y \sim VARIANT( integer )
Line indices of the base points of the region contour.
  Default Value : 100
  Suggested values : Rows \in \{0, 10, 30, 50, 100, 200, 300, 500\}
  (lin)Minimum Increment : 1
  Recommended Increment : 1

▷ Columns (input control) .............................................. polygon.x \sim VARIANT( integer )
Column indices of the base points of the region contour.
  Default Value : 100
  Suggested values : Columns \in \{0, 10, 30, 50, 100, 200, 300, 500\}
  (lin)Minimum Increment : 1
  Recommended Increment : 1
  Number of elements : (Columns = Rows)

Example

/* Polygon-approximation*/
get_region_polygon(Region, 7, Row, Column)
/* store it as a region */
gen_region_polygon(Pol, Row, Column)
/* fill up the hole */
fill_up(Pol, Filled).

Result

If the base points are correct the operator GenRegionPolygon returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem ("clipRegion", <‘true’/’false’>). If an empty region is created (by the clipping or by an empty
10.3. CREATION

input) the operator SetSystem(‘storeEmptyRegion’,<true/false>) determines whether the region is returned or an empty object tuple.

Parallelization Information

GenRegionPolygon is reentrant, local, and processed without parallelization.

Possible Predecessors

GetRegionPolygon, DrawPolygon

See also

FillUp, ReduceDomain, GetRegionPolygon, DrawPolygon

Alternatives

GenRegionPolygonFilled, GenRegionPoints, GenRegionRuns

Module

Region processing

.Store a polygon as a “filled” region.

The operator GenRegionPolygonFilled creates a region from a polygon containing the corner points of the region (line and column coordinates) either clockwise or anti-clockwise. Contrary to GenRegionPolygon a “filled” region is returned here.

Attention

Parameter

▷ Region (output iconic) .................................................. region ~ HRegionX / HUntypedObjectX

Created region.

▷ Rows (input control) ...................................................... polygon.y ~ VARIANT( integer )

Line indices of the base points of the region contour.

Default Value : 100

Suggested values : Rows ∈ {0, 10, 30, 50, 100, 200, 300, 500}

Minimum Increment : 1

Recommended Increment : 1

▷ Columns (input control) ............................................... polygon.x ~ VARIANT( integer )

Column indices of the base points of the region contour.

Default Value : 100

Suggested values : Columns ∈ {0, 10, 30, 50, 100, 200, 300, 500}

Minimum Increment : 1

Recommended Increment : 1

Number of elements : (Columns = Rows)

Result

If the base points are correct the operator GenRegionPolygonFilled returns the value TRUE. Otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem(‘clipRegion’,<true’/’false’)). If an empty region is created (by the clipping or by an empty input) the operator SetSystem(‘storeEmptyRegion’,<true/false>) determines whether the region is returned or an empty object tuple.

Parallelization Information

GenRegionPolygonFilled is reentrant, local, and processed without parallelization.

Possible Predecessors

GetRegionPolygon, DrawPolygon

See also

GenRegionPolygon, ReduceDomain, GetRegionPolygon, GenRegionRuns

void HRegionX.GenRegionPolygonFilled ([in] VARIANT Rows, [in] VARIANT Columns )

void HOperatorSetX.GenRegionPolygonFilled ([out] HUntypedObjectX Region, [in] VARIANT Rows, [in] VARIANT Columns )

HALCON 6.1.4
Create a region from an XLD polygon.

GenRegionPolygonXld creates a region Region from a subpixel XLD polygon Polygon. The polygon is sampled according to the Bresenham algorithm and influenced by the parameter neighborhood of the operator SetSystem. Open polygons are closed before converting them to regions. Finally, the parameter Mode defines whether the region is filled up (filled) or returned by its contour (margin).

### Parameter

- **Polygon** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . xld_poly \( \rightarrow \) HXLDPolyX / IHObjectX
  - Input polygon.
- **Region** (output iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . region \( \rightarrow \) HRegionX / HUntypedObjectX
  - Created region.
- **Mode** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . string \( \sim \) String / VARIANT
  - Fill mode of the region.
  - Default Value: 'filled'
  - Suggested values: Mode \( \in \{ \text{`filled'}, \text{`margin'} \} \)

### Parallelization Information

GenRegionPolygonXld is reentrant and processed without parallelization.

### Possible Predecessors

GenContourPolygonXld, GenContourPolygonRoundedXld

### See also

SetSystem

### Alternatives

GenRegionPolygon, GenRegionContourXld

### Module

Sub-pixel operators

---

Create an image region from a runlength coding.

The operator GenRegionRuns creates a region described by the input runlength structure. The runlength representation is created by examining a region line by line with ascending line number (= from “top” to “bottom”). Every line runs through from left to right (ascending column number). All starting and ending points being stored by region segments (=runs). Thus a region can be described by a sequence of runs, a run being defined by line number as well as starting and ending points (column number).

The storing is fastest when the runs are sorted. The order is as follows:

\[
(l_1, b_1, e_1) \leq (l_2, b_2, e_2) := (l_1 < l_2) \vee (l_1 = l_2) \wedge (b_1 \leq b_2)
\]
10.3. CREATION

Attention

Parameter

◮ Region (output iconic) \dots \sim HRegionX / HUntypedObjectX
Created region.

◮ Row (input control) \dots \sim \text{VARIANT}(\text{integer})
Lines of the runs.

\hspace{1em} Default Value : 100
\hspace{1em} Suggested values : Row \in \{0, 50, 100, 200, 300, 500\}
\hspace{1em} Minimum Increment : 1
\hspace{1em} Recommended Increment : 10

◮ ColumnBegin (input control) \dots \sim \text{VARIANT}(\text{integer})
Columns of the starting points of the runs.

\hspace{1em} Default Value : 50
\hspace{1em} Suggested values : ColumnBegin \in \{0, 50, 100, 200, 300, 500\}
\hspace{1em} Minimum Increment : 1
\hspace{1em} Recommended Increment : 10
\hspace{1em} Number of elements : (ColumnBegin = Row)

◮ ColumnEnd (input control) \dots \sim \text{VARIANT}(\text{integer})
Columns of the ending points of the runs.

\hspace{1em} Default Value : 200
\hspace{1em} Suggested values : ColumnEnd \in \{50, 100, 200, 300, 500\}
\hspace{1em} Minimum Increment : 1
\hspace{1em} Recommended Increment : 10
\hspace{1em} Restriction : (ColumnEnd \geq ColumnBegin)
\hspace{1em} Number of elements : (ColumnEnd = Row)

Complexity

\( F \) shall be the number of pixels. If the pixels are sorted in ascending order the runtime complexity is: \( O(F) \), otherwise it is \( O(\log(F) \ast F) \).

Result

If the data is correct the operator GenRegionRuns returns the value TRUE, otherwise an exception handling is raised. The clipping according to the current image format is set via the operator SetSystem (‘clipRegion’,<true/’false’>). If an empty region is created (by the clipping or by an empty input) the operator SetSystem(‘storeEmptyRegion’,<true/false>) determines whether the region is returned or an empty object tuple.

Parallelization Information

GenRegionRuns is reentrant, local, and processed without parallelization.

Possible Predecessors

GetRegionRuns

See also

ReduceDomain

Alternatives

GenRegionPoints, GenRegionPolygon, GenRegionLine, GenRegionPolygonFilled

Module

Region processing

[\text{out}] \text{HRegionX} \text{ Regions} \text{ HImageX.LabelToRegion}() \\
\text{void HOperatorSetX.LabelToRegion}([\text{in}] \text{IObjectX LabelImage}, [\text{out}] \text{HUntypedObjectX Regions})

Extract regions with equal gray values from an image.

LabelToRegion segments an image into regions of equal gray value. One output region is generated for each gray value occurring in the image. This is similar to calling Threshold multiple times, and accumulating
the results with \texttt{ConcatObj}. Another related operator is \texttt{Regiongrowing}. However, \texttt{LabelToRegion} does not perform a \texttt{Connection} operation on the resulting regions, i.e., they may be disconnected. A typical application of \texttt{LabelToRegion} is the segmentation of label images, hence its name.

The number of output regions is limited by the system parameter ‘max\textunderscore out\textunderscore obj\textunderscore par’, which can be read via \texttt{GetSystem(‘max\textunderscore out\textunderscore obj\textunderscore par’:<Anzahl>).}

\begin{itemize}
\item \texttt{LabelToRegion} is not implemented for images of type ‘real’. The input images must not contain negative gray values.
\end{itemize}

\begin{itemize}
\item \texttt{LabelImage} (input iconic) \begin{itemize}
\item image(-array) \sim \texttt{HImageX / IImageX (byte, int2, int4)}
\end{itemize} Label image.
\item \texttt{Regions} (output iconic) \begin{itemize}
\item region \sim \texttt{HRegionX / HUntypedObjectX}
\end{itemize} Regions having a constant gray value.
\end{itemize}

\begin{itemize}
\item Complexity \begin{itemize}
\item Let \(x1\) be the minimum x-coordinate, \(x2\) the maximum x-coordinate, \(y1\) be the minimum y-coordinate, and \(y2\) the maximum y-coordinate of a particular gray value. Furthermore, let \(N\) be the number of different gray values in the image. Then the runtime complexity is \(O(N \cdot (x2 - x1 + 1) \cdot (y2 - y1 + 1))\)
\end{itemize}
\end{itemize}

\begin{itemize}
\item Result \begin{itemize}
\item \texttt{LabelToRegion} returns TRUE if the gray values lie within a correct range. The behavior with respect to the input images and output regions can be determined by setting the values of the flags ‘\texttt{no\textunderscore object\textunderscore result}’, ‘\texttt{empty\textunderscore region\textunderscore result}’, and ‘\texttt{store\textunderscore empty\textunderscore region}’ with \texttt{SetSystem}. If necessary, an exception is raised.
\end{itemize}
\end{itemize}

\begin{itemize}
\item Parallelization Information \begin{itemize}
\item \texttt{LabelToRegion} is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level).
\end{itemize}
\end{itemize}

\begin{itemize}
\item Possible Predecessors \begin{itemize}
\item \texttt{MinMaxGray, SobelAmp, GaussImage, ReduceDomain, DiffOfGauss}
\end{itemize}
\end{itemize}

\begin{itemize}
\item Possible Successors \begin{itemize}
\item \texttt{Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton}
\end{itemize}
\end{itemize}

\begin{itemize}
\item \texttt{See also} \begin{itemize}
\item \texttt{Threshold, ConcatObj, Regiongrowing, RegionToLabel}
\end{itemize}
\end{itemize}

\begin{itemize}
\item \texttt{Module} \begin{itemize}
\item \texttt{Region processing}
\end{itemize}
\end{itemize}

\section{10.4 Features}

\begin{verbatim}
[out] VARIANT Area HRegionX.AreaCenter ([out] VARIANT Row,
[out] VARIANT Column )
void HOperatorSetX.AreaCenter ([in] IObjectX Regions,
[out] VARIANT Area, [out] VARIANT Row, [out] VARIANT Column )
\end{verbatim}

\begin{itemize}
\item \texttt{Area and center of regions.}
\end{itemize}

The operator \texttt{AreaCenter} calculates the area and the center of the input regions. The area is defined as the number of pixels of a region. The center is calculated as the mean value of the line or column coordinates, respectively, of all pixels.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of the input region. In case of empty region all parameters have the value 0.0 if no other behavior was set (see \texttt{SetSystem}).
10.4. FEATURES

Parameter

▷ Regions (input iconic) ........................................... region(-array) ↦ HRegionX / IHObjectX
Region(s) to be examined.

▷ Area (output control) ........................................... integer(-array) ↦ VARIANT (integer)
Area of the region.

▷ Row (output control) .......................................... point.y(-array) ↦ VARIANT (real)
Line index of the center.

▷ Column (output control) .................................... point.x(-array) ↦ VARIANT (real)
Column index of the center.

Complexity

If $F$ is the area of a region the mean runtime complexity is $O(\sqrt{F})$.

Result

The operator AreaCenter returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

AreaCenter is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

SelectShape

Module

Region processing

[out] VARIANT Circularity HRegionX.Circularity ( )

void HOperatorSetX.Circularity ([in] IHObjectX Regions,
[out] VARIANT Circularity )

Shape factor for the circularity (similarity to a circle) of a region.

The operator Circularity calculates the similarity of the input region with a circle.

Calculation: If $F$ is the area of the region and max is the maximum distance from the center to all contour pixels, the shape factor $C$ is defined as:

$$ C = \frac{F}{(\text{max}^2 \cdot \pi)} $$

The shape factor $C$ of a circle is 1. If the region is long or has holes $C$ is smaller than 1. The operator Circularity especially responds to large bulges, holes and unconnected regions.

In case of an empty region the operator Circularity returns the value 0 (if no other behavior was set (see SetSystem)). If more than one region is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.

Parameter

▷ Regions (input iconic) ........................................... region(-array) ↦ HRegionX / IHObjectX
Region(s) to be examined.

▷ Circularity (output control) ................................ real(-array) ↦ VARIANT (real)
Roundness of the input region(s).

Restriction: $((0 \leq \text{Circularity}) \land (\text{Circularity} \leq 1.0))$

Example

/* Comparison between shape factors of rectangle, circle and ellipse: */
gen_rectangle1(R1,10,10,20,20)
CHAPTER 10. REGIONS

Result

The operator `Circularity` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

Parallelization Information

`Circularity` is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

AreaCenter, SelectShape

Alternatives

Roundness, Compactness, Convexity, Eccentricity

Module

Region processing

```c
[0x00] VARIANT Compactness HRegionX.Compactness ()

void HOperatorSetX.Compactness ([in] IHObjectX Regions,
[0x00] [out] VARIANT Compactness )
```

Shape factor for the compactness of a region.

The operator `Compactness` calculates the compactness of the input regions.

Calculation: If \( L \) is the length of the contour (see `Contlength`) and \( F \) the area of the region the shape factor \( C \) is defined as:

\[
C = \frac{L^2}{4F\pi}
\]

The shape factor \( C \) of a circle is 1. If the region is long or has holes \( C \) is larger than 1. The operator `Compactness` responds to the course of the contour (roughness) and to holes. In case of an empty region the operator `Compactness` returns the value 0 if no other behavior was set (see `SetSystem`). If more than one region is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.

Attention

Parameter

- **Regions** (input iconic) .......................... region(-array) \( \rightarrow \) HRegionX / IHObjectX Region(s) to be examined.
- **Compactness** (output control) .................... real(-array) \( \rightarrow \) VARIANT (real ) Compactness of the input region(s).

Restriction: \((\text{Compactness} \geq 1.0) \lor (\text{Compactness} = 0)\)
The operator `Compactness` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>).` The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>).` If necessary an exception handling is raised.

**Parallelization Information**

`Compactness` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, Regiongrowing, Connection

Contlength, AreaCenter, SelectShape

**Alternatives**

Compactness, Convexity, Eccentricity

**Module**

Region processing

---

**Number of connection components and holes**

The operator `ConnectAndHoles` calculates the number of connection components and the number of holes of each region of `Regions`.

If more than one region is passed the numerical values of the output control parameters `NumConnected` and `NumHoles` are each stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.

**Attention**

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ <code>Regions</code> (input iconic) ……………………………..region(-array) ↼ <code>HRegionX / IHObjectX</code> Region(s) to be examined.</td>
</tr>
<tr>
<td>▶ <code>NumConnected</code> (output control) ………………………… integer(-array) ↼ <code>VARIANT</code> ( integer ) Number of connection components of a region.</td>
</tr>
<tr>
<td>▶ <code>NumHoles</code> (output control) ……………………………..integer(-array) ↼ <code>VARIANT</code> ( integer ) Number of holes of a region.</td>
</tr>
</tbody>
</table>

The operator `ConnectAndHoles` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>).` The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>).` If necessary an exception handling is raised.

**Parallelization Information**

`ConnectAndHoles` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, Regiongrowing, Connection

Connection, FillUp, FillUpShape, Union1

**Alternatives**

EulerNumber

**Module**

Region processing

---

HALCON 6.1.4
Contour length of a region.

The operator **Contlength** calculates the total length of the contour (sum of all connection components of the region) for each region of **Regions**. The distance between two neighboring contour points parallel to the coordinate axes is rated 1, the distance in the diagonal is rated $\sqrt{2}$. If more than one region is passed the numerical values of the contour length are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple. In case of an empty region the operator **Contlength** returns the value 0.

The contour of holes is not calculated.

**Parameter**

- **Regions** (input iconic) ~ HRegionX / IHObjectX Region(s) to be examined.
- **ContLength** (output control) ~ VARIANT (real) Contour length of the input region(s).

**Restriction** : (ContLength $\geq 0$)

**Result**

The operator **Contlength** returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem**('noObjectResult',<Result>). If necessary an exception handling is raised.

**Parallelization Information**

**Contlength** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection

**Possible Successors**

GetRegionContour

**See also**

AreaCenter, GetRegionContour

**Alternatives**

Compactness

**Module**

Region processing

Shape factor for the convexity of a region.

The operator **Convexity** calculates the convexity of each input region of **Regions**.

**Calculation**: If $F_c$ is the area of the convex hull and $F_o$ the original area of the region the shape factor $C$ is defined as:

$$C = \frac{F_o}{F_c}$$

The shape factor $C$ is 1 if the region is convex (e.g., rectangle, circle etc.). If there are indentations or holes $C$ is smaller than 1.

In case of an empty region the operator **Convexity** returns the value 0 (if no other behavior was set (see **SetSystem**)). If more than one region is passed the numerical values of the contour length are stored in a tuple, the position of a value in the tuple corresponding to the position of the region in the input tuple.
10.4. FEATURES

---

### Parameter

- **Regions** (input iconic) region(-array) \( \sim \) HRegionX / IOBJECTX
  - Region(s) to be examined.
- **Convexity** (output control) real(-array) \( \sim \) VARIANT( real )
  - Convexity of the input region(s).

**Restriction:** \( \text{Convexity} \leq 1 \)

---

### Result

The operator **Convexity** returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>).` The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>).` If necessary an exception handling is raised.

---

### Parallelization Information

**Convexity** is reentrant and automatically parallelized (on tuple level).

---

### Possible Predecessors

Threshold, RegionGrowing, Connection

---

### See also

SelectShape, AreaCenter, ShapeTrans

---

### Module

Region processing

---

### Code Snippet

```c
[out] VARIANT Row1 HRegionX.DiameterRegion ([out] VARIANT Column1,
[out] VARIANT Row2, [out] VARIANT Column2, [out] VARIANT Diameter )

void HoperatorSetX.DiameterRegion ([in] IOBJECTX Regions,
[out] VARIANT Row1, [out] VARIANT Column1, [out] VARIANT Row2,
[out] VARIANT Column2, [out] VARIANT Diameter )
```

---

### Attention

If the region is empty, the results of **Row1, Column1, Row2** and **Column2** (all of them = 0) may lead to confusion.

---

### Parameter

- **Regions** (input iconic) region(-array) \( \sim \) HRegionX / IOBJECTX
  - Regions to be examined.
- **Row1** (output control) line.begin.y(-array) \( \sim \) VARIANT( integer )
  - Row index of the first extreme point.
- **Column1** (output control) line.begin.x(-array) \( \sim \) VARIANT( integer )
  - Column index of the first extreme point.
- **Row2** (output control) line.end.y(-array) \( \sim \) VARIANT( integer )
  - Row index of the second extreme point.
- **Column2** (output control) line.end.x(-array) \( \sim \) VARIANT( integer )
  - Column index of the second extreme point.
- **Diameter** (output control) number(-array) \( \sim \) VARIANT( real )
  - Distance of the two extreme points.

---

### Complexity

If \( F \) is the area of a region, the runtime complexity amounts to \( O(\sqrt{F}) \) on average.

---

### Result

The operator **DiameterRegion** returns the value TRUE, if the input is not empty. The reaction to
empty input (no input regions are available) may be determined with the help of the operator SetSystem('noObjectResult',<Result>). The reaction concerning an empty region (region is the empty set) will be determined by the operator SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

DiameterRegion is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, Region growing, Connection, RunlengthFeatures

---

**Possible Successors**

DispLine

---

**Alternatives**

SmallestRectangle2

---

**Module**

Region processing

---

```c
[out] VARIANT Anisometry HRegionX.Eccentricity ([out] VARIANT Bulkiness,
[out] VARIANT StructureFactor )

void HOperatorSetX.Eccentricity ([in] IHOBJECTX Regions,
[out] VARIANT Anisometry, [out] VARIANT Bulkiness,
[out] VARIANT StructureFactor )
```

Shape features derived from the ellipse parameters.

The operator Eccentricity calculates three shape features derived from the geometric moments.

**Definition:** If the parameters $Ra$, $Rb$ and the area $A$ of the region are given (see EllipticAxis), the following applies:

\[ \text{Anisometry} = \frac{Ra}{Rb} \]

\[ \text{Bulkiness} = \pi \cdot \frac{Ra \cdot Rb}{A} \]

\[ \text{StructureFactor} = \text{Anisometry} \cdot \text{Bulkiness} - 1 \]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

---

**Attention**

---

**Parameter**

- **Regions** (input iconic) .....................region(-array) ↷ HRegionX / IHOBJECTX Region(s) to be examined.
- **Anisometry** (output control) ..................real(-array) ↷ VARIANT (real )
  Shape feature (in case of a circle = 1.0).
  **Restriction:** $(\text{Anisometry} \geq 1.0)$
- **Bulkiness** (output control) .....................real(-array) ↷ VARIANT (real )
  Calculated shape feature.
- **StructureFactor** (output control) ................real(-array) ↷ VARIANT (real )
  Calculated shape feature.

---

**Complexity**

If $F$ is the area of the region the mean runtime complexity is $O(\sqrt{F})$.

---

**Result**

The operator Eccentricity returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>).

HALCON/COM Reference Manual, 2005-2-1
The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

Eccentricity is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, RegionGrowing, Connection

---

See also

EllipticAxis, MomentsRegion2Nd, SelectShape, AreaCenter

---

**Module**

Region processing

---

The operator EllipticAxis calculates the radii and the orientation of the ellipse having the "same orientation" and the "same side relation" as the input region. Several input regions can be passed in `Regions` as tuples. The length of the main radius `Ra` and the secondary radius `Rb` as well as the orientation of the main axis with regard to the horizontal (`Phi`) are determined. The angle is indicated in arc measure.

Calculation:

If the moments \( M_{20}, M_{02} \) and \( M_{11} \) are normalized and passed to the area (see `MomentsRegion2Nd`), the radii `Ra` and `Rb` are calculated as:

\[
\begin{align*}
Ra &= \sqrt{\frac{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}{2}} \\
Rb &= \sqrt{\frac{8(M_{20} + M_{02} - \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}{2}}
\end{align*}
\]

The orientation `Phi` is defined by:

\[
Phi = -0.5atan2(2M_{11}, M_{02} - M_{20})
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem('noObjectResult',<Result>).

---

**Attention**

---

**Parameters**

- **Regions** (input iconic)
- **Ra** (output control)
- **Rb** (output control)
- **Phi** (output control)

---

**Restrictions**

- `(Ra \geq 0.0)`
- `(Rb \geq 0.0) \land (Rb \leq Ra)`
- `((-pi)/2 < Phi) \land (Phi \leq (pi/2))`
Example

read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible',''',WindowHandle)
regiongrowing(Image,Seg,5,5,6,100)
elliptic_axis(Seg,Ra,Rb,Phi)
area_center(Seg,_,Row,Column)
gen_ellipse(Ellipses,Row,Column,Phi,Ra,Rb)
set_draw(WindowHandle,'margin')
disp_region(Ellipses,WindowHandle)

Complexity

If $F$ is the area of a region the mean runtime complexity is $O(\sqrt{F})$.

Result

The operator EllipticAxis returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem ('emptyRegionResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

EllipticAxis is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold,Regiongrowing,Connection

Possible Successors

GenEllipse

See also

MomentsRegion2Nd,SelectShape,SetShape

Alternatives

SmallestRectangle2,OrientationRegion

References

R. Haralick, L. Shapiro “Computer and Robot Vision” Addison-Wesley, 1992, pp. 73-75

Module

Region processing

[out] VARIANT EulerNumber HRegionX.EulerNumber ( )
void HOperatorSetX.EulerNumber ([in] IObjectX Regions,
[out] VARIANT EulerNumber )

Calculate the Euler number.

The procedure EulerNumber calculates the Euler number, i.e., the difference between the number of connection components and the number of holes.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

Parameter

▷ Regions (input iconic) ………………………………………..region(-array) ~ HRegionX / IObjectX Region(s) to be examined.
▷ EulerNumber (output control) ……………………………..integer(-array) ~ VARIANT ( integer ) Calculated Euler number.

Result

The operator EulerNumber returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>).
The behavior in case of empty region (the region is the empty set) is set via \texttt{SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

\textbf{Parallelization Information} \hfill \textbf{Possible Predecessors}  
\texttt{EulerNumber} is reentrant and automatically parallelized (on tuple level).  

\textbf{Threshold, RegionGrowing, Connection} \hfill \textbf{Alternatives}  
\texttt{ConnectAndHoles}  

\textbf{Region processing}  

\begin{verbatim}
[out] VARIANT RegionIndex1 HRegionX.FindNeighbors ([in] HRegionX Regions2, [in] long MaxDistance, [out] VARIANT RegionIndex2 )

void HOperatorSetX.FindNeighbors ([in] IObjectX Regions1, [in] IObjectX Regions2, [in] VARIANT MaxDistance, [out] VARIANT RegionIndex1, [out] VARIANT RegionIndex2 )
\end{verbatim}

\textbf{Search direct neighbors.}  
The operator \texttt{FindNeighbors} determines neighboring regions with \texttt{Regions1} and \texttt{Regions2} containing the regions to be examined. \texttt{Regions1} can have three different states:

- \texttt{Regions1} is empty:  
  In this case all regions in \texttt{Regions2} are permutatively checked for neighborhood.

- \texttt{Regions1} consists of one region:  
  The regions of \texttt{Regions1} are compared to all regions in \texttt{Regions2}.

- \texttt{Regions1} consists of the same number of regions as \texttt{Regions2}:  
  Here all regions at the n-th position in \texttt{Regions1} and \texttt{Regions2} are checked for the neighboring relation.

The operator \texttt{FindNeighbors} uses the city block distance between neighboring regions. It can be specified by the parameter \texttt{MaxDistance}. neighboring regions are located at the n-th position in \texttt{Regions1} and \texttt{RegionIndex2}, i.e., the region with index \texttt{RegionIndex1}[n] from \texttt{Regions1} is the neighbor of the region with index \texttt{RegionIndex2}[n] from \texttt{Regions2}.  

\textbf{Attention}  
Covered regions are not found!

\begin{verbatim}
\begin{table}[h]
\centering
\begin{tabular}{ll}
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Regions1} (input iconic)</td>
<td>region(-array) \rightarrow HRegionX / IObjectX \hfill Starting regions.</td>
</tr>
<tr>
<td>\texttt{Regions2} (input iconic)</td>
<td>region(-array) \rightarrow HRegionX / IObjectX \hfill Comparative regions.</td>
</tr>
<tr>
<td>\texttt{MaxDistance} (input control)</td>
<td>integer \rightarrow long / VARIANT \hfill Maximal distance of regions.</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Suggested values</td>
<td>MaxDistance \in {1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 20, 50}</td>
</tr>
<tr>
<td>Typical range of values</td>
<td>1 \leq MaxDistance \leq 1</td>
</tr>
<tr>
<td>Minimum Increment</td>
<td>1</td>
</tr>
<tr>
<td>Recommended Increment</td>
<td>1</td>
</tr>
<tr>
<td>\texttt{RegionIndex1} (output control)</td>
<td>integer \rightarrow VARIANT (integer) \hfill Indices of the found regions from \texttt{Regions1}.</td>
</tr>
<tr>
<td>\texttt{RegionIndex2} (output control)</td>
<td>integer \rightarrow VARIANT (integer) \hfill Indices of the found regions from \texttt{Regions2}.</td>
</tr>
</tbody>
</table>
\end{tabular}
\end{table}
\end{verbatim}

\textbf{Result}  
The operator \texttt{FindNeighbors} returns the value \texttt{TRUE} if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>).
The behavior in case of empty region (the region is the empty set) is set via `SetSystem` (`'emptyRegionResult',<Result>`). If necessary an exception handling is raised.

---

Parallelization Information

FindNeighbors is reentrant and processed without parallelization.

---

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

SpatialRelation, SelectRegionSpatial, ExpandRegion, Interjacent, Boundary

---

Module

Region processing

---

```
[out] VARIANT Index HRegionX.GetRegionIndex ([in] long Row, [in] long Column )
```

```
```

Index of all regions containing a given pixel.

The operator `GetRegionIndex` returns the index of all regions in `Regions` (range of values: 1 to n) containing the test pixel (`Row, Column`), i.e.:

\[ |\text{Regions}_n \cap \{(\text{Row, Column})\}| \neq \emptyset \]

The returned indices can be used, e.g., in `SelectObj` to select the regions containing the test pixel.

Attention

If the regions overlap more than one region might contain the pixel. In this case all these regions are returned. If no region contains the indicated pixel the empty tuple (= no region) is returned.

---

Parameter

- **Regions** (input iconic)  
  Regions to be examined.

- **Row** (input control)  
  Line index of the test pixel. 
  Default Value: 100 (lin)

- **Column** (input control)  
  Column index of the test pixel. 
  Default Value: 100 (lin)

- **Index** (output control)  
  Index of the regions containing the test pixel.

Complexity

If \(F\) is the area of the region and \(N\) is the number of regions the mean runtime complexity is \(O(ln(\sqrt{F} \times N))\).

---

Result

The operator `GetRegionIndex` returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem` (`'noObjectResult',<Result>`). If necessary an exception handling is raised.

---

Parallelization Information

`GetRegionIndex` is reentrant and automatically parallelized (on tuple level).

---

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

GetMbutton, GetMposition, TestRegionPoint

HALCON/COM Reference Manual, 2005-2-1
10.4. FEATURES

Alternatives

SelectRegionPoint

Module

Region processing

```c
[out] VARIANT Thickness HRegionX.GetRegionThickness
(void) VARIANT Histogramm )
```

Access the thickness of a region along the main axis.

The operator `GetRegionThickness` calculates the thickness of the regions along the main axis (see `EllipticAxis`) for each pixel of the section. The thickness at one point on the main axis is defined as the distance between the intersections of the contour with the plumb on the main axis in the respective point which are the furthest apart. Additionally the operator `GetRegionThickness` returns the `Histogramm` of the thicknesses of the region. The length of the histogram corresponds to the largest occurring thickness in the observed region.

**Attention**

Only one region may be passed. If the region has several connection components, only the first one is investigated. All other components are ignored.

**Parameter**

- **Region** (input iconic) ... region ~ HRegionX / IObjectX
  Region to be analysed.
- **Thickness** (output control) ... integer ~ VARIANT (integer )
  Thickness of the region along its main axis.
- **Histogramm** (output control) ... integer ~ VARIANT (integer )
  Histogram of the thickness of the region along its main axis.

**Result**

The operator `GetRegionThickness` returns the value TRUE if exactly one region is passed. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem` ('noObjectResult',<Result>).

**Parallelization Information**

`GetRegionThickness` is reentrant and processed without parallelization.

**Possible Predecessors**

SobelAmp, Threshold, Connection, SelectShape, SelectObj

**See also**

CopyObj, EllipticAxis

Region processing

```c
[out] VARIANT Distance HRegionX.HammingDistance
([in] HRegionX Regions2, [out] VARIANT Similarity )
```

Hamming distance between two regions.

The operator `HammingDistance` returns the hamming distance between two regions, i.e., the number of pixels of the regions which are different (`Distance`), i.e., the number of pixels contained in one region but not in the other:

\[
Distance = |Regions1 \cap Regions2| + |Regions2 \cap Regions1|
\]
The parameter \textit{Similarity} describes the similarity between the two regions based on the hamming distance \textit{Distance}:

\[
\text{Similarity} = 1 - \frac{\text{Distance}}{|\text{Regions1}| + |\text{Regions2}|}
\]

If both regions are empty \textit{Similarity} is set to 0. The regions with the same index from both input parameters are always compared.

\textbf{Attention}

In both input parameters the same number of regions must be passed.

\begin{itemize}
  \item \textbf{Parameter}
  \begin{itemize}
    \item \textbf{Regions1} (input iconic) \texttt{\textunderscore{}array} \texttt{HRegionX / IHObjectX}
      Regions to be examined.
    \item \textbf{Regions2} (input iconic) \texttt{\textunderscore{}array} \texttt{HRegionX / IHObjectX}
      Comparative regions.
    \item \textbf{Distance} (output control) \texttt{\textunderscore{}array} \texttt{integer}
      Hamming distance of two regions.
      \textbf{Restriction:} \texttt{\textunderscore{} Vor } \texttt{\textunderscore{} Restriction}
      \texttt{\textunderscore{} Vor } \texttt{\textunderscore{} Restriction}
    \item \textbf{Similarity} (output control) \texttt{\textunderscore{}array} \texttt{real}
      Similarity of two regions.
      \textbf{Restriction:} \texttt{\textunderscore{} Vor } \texttt{\textunderscore{} Restriction}
  \end{itemize}
\end{itemize}

\textbf{Complexity}

If \( F \) is the area of a region the mean runtime complexity is \( O(\sqrt{F}) \).

\textbf{Result}

\texttt{HammingDistance} returns the value \texttt{TRUE} if the number of objects in both parameters is the same and is not 0. The behavior in case of empty input (no input objects available) is set via the operator \texttt{SetSystem(‘noObjectResult’,<Result>).} The behavior in case of empty region (the region is the empty set) is set via \texttt{SetSystem(‘emptyRegionResult’,<Result>).} If necessary an exception handling handling is raised.

\textbf{Parallelization Information}

\texttt{HammingDistance} is \texttt{reentrant} and automatically \texttt{parallelized} (on tuple level).

\textbf{Possible Predecessors}

\begin{itemize}
  \item \texttt{Threshold}, \texttt{RegionGrowing}, \texttt{Connection}
  \item \texttt{HammingChangeRegion}
  \item \texttt{Intersection}, \texttt{Complement}, \texttt{AreaCenter}
\end{itemize}

\textbf{Alternatives}

Region processing

\begin{verbatim}
[out] VARIANT Distance HRegionX.HammingDistanceNorm
([in] HRegionX Regions2, [in] VARIANT Norm, [out] VARIANT Similarity )

void HOperatorSetX.HammingDistanceNorm ([in] IHObjectX Regions1,
[in] IHObjectX Regions2, [in] VARIANT Norm, [out] VARIANT Distance,
[out] VARIANT Similarity )
\end{verbatim}

\textit{Hamming distance between two regions using normalization.}

The operator \texttt{HammingDistanceNorm} returns the hamming distance between two regions, i.e., the number of pixels of the regions which are different (\textit{Distance}). Before calculating the difference the region in \texttt{Regions1} is normalized onto the regions in \texttt{Regions2}. The result is the number of pixels contained in one region but not in the other:

\[
\text{Distance} = |\text{Norm(Regions1) } \cap \text{Regions2}| + |\text{Regions2 } \cap \text{Norm(Regions1)}|
\]
The parameter `Similarity` describes the similarity between the two regions based on the hamming distance `Distance`:

\[
\text{Similarity} = 1 - \frac{\text{Distance}}{|\text{Norm(Regions1)}| + |\text{Regions2}|}
\]

The following types of normalization are available:

'center': The region is moved so that both regions have the same center of gravity.

If both regions are empty `Similarity` is set to 0. The regions with the same index from both input parameters are always compared.

---

**Parameter**

- **Regions1** (input iconic) 
  
  Region(-array) → HRegionX / IHObjectX
  
  Regions to be examined.

- **Regions2** (input iconic) 
  
  Region(-array) → HRegionX / IHObjectX
  
  Comparative regions.

- **Norm** (input control) 
  
  String(-array) → VARIANT ( string )
  
  Type of normalization.

  **Default Value**: 'center'

  **List of values**: Norm ∈ {'center'}

- **Distance** (output control) 
  
  Integer(-array) → VARIANT ( integer )
  
  Hamming distance of two regions.

  **Restriction**: (Distance ≥ 0)

- **Similarity** (output control) 
  
  Real(-array) → VARIANT ( real )
  
  Similarity of two regions.

  **Restriction**: ((0 ≤ Similarity) ∧ (Similarity ≤ 1))

---

**Complexity**

If \( F \) is the area of a region the mean runtime complexity is \( O(\sqrt{F}) \).

---

**Result**

`hamming_distance_norm` returns the value TRUE if the number of objects in both parameters is the same and is not 0. The behavior in case of empty input (no input objects available) is set via the operator `SetSystem` (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via `SetSystem` (‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

`HammingDistanceNorm` is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, Regiongrowing, Connection

---

**See also**

HammingChangeRegion

---

**Alternatives**

Intersection, Complement, AreaCenter

---

**Module**

Region processing

---

```
[out] VARIANT Row HRegionX.InnerCircle ([out] VARIANT Column,
[out] VARIANT Radius )
```

Largest inner circle of a region.
The operator \texttt{InnerCircle} determines the largest inner circle of a region. This is the biggest discrete circle region that completely fits into the region. For this circle the center (\texttt{Row, Column}) and the radius (\texttt{Radius}) are calculated. If the position of the circle is ambiguous, the "first possible" position (as far upper left as possible) is returned.

The output of the procedure is chosen in such a way that it can be used as an input for the HALCON procedures \texttt{DispCircle}, \texttt{GenCircle}, and \texttt{GenEllipseContourXld}.

If several regions are passed in \texttt{Regions} corresponding tuples are returned as output parameters. In case of an empty input region all parameters have the value 0.0 if no other behavior was set with \texttt{SetSystem}.

\begin{verbatim}
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
regiongrowing(Image,Seg,5,5,6,100)
select_shape(Seg,H,'area','and',100,2000)
inner_circle(H,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius:)
set_draw(WindowHandle,'margin')
disp_region(Circles,WindowHandle)
\end{verbatim}

\textbf{Complexity} 
If $F$ is the area of the region and $R$ is the radius of the inner circle the runtime complexity is $O(\sqrt{F \times R})$.

\textbf{Result} 
The operator \texttt{InnerCircle} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem("noObjectResult",<Result>)}, the behavior in case of empty region is set via \texttt{SetSystem("emptyRegionResult",<Result>)}. If necessary an exception handling is raised.

\textbf{Parallelization Information} 
\texttt{InnerCircle} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

\textbf{Possible Predecessors} 
\texttt{Threshold, Regiongrowing, Connection, RunlengthFeatures}

\textbf{Possible Successors} 
\texttt{GenCircle, DispCircle}

\textbf{See also} 
\texttt{SetShape, SelectShape, SmallestCircle}

\textbf{Alternatives} 
\texttt{ErosionCircle}

\textbf{Module} 
Region processing
Geometric moments of regions.

The operator \texttt{MomentsRegion2Nd} calculates the moments \((M_{20}, M_{02})\) and the product of inertia of the axes through the center parallel to the coordinate axes \((M_{11})\). Furthermore the main axes of inertia \((I_a, I_b)\) are calculated.

Calculation: \(Z_0\) and \(S_0\) are the coordinates of the center of a region \(R\) with the area \(F\). Then the moments \(M_{ij}\) are defined by:

\[
M_{ij} = \sum_{(Z, S) \in R} (Z - Z_0)^i (S - S_0)^j
\]

wherein \(Z\) and \(S\) run through all pixels of the region \(R\).

Furthermore, \(h = \frac{M_{20} + M_{02}}{2}\) then \(I_a\) and \(I_b\) are defined by:

\[
I_a = h + \sqrt{h^2 - M_{20} \cdot M_{02} + M_{11}^2}
\]

\[
I_b = h - \sqrt{h^2 - M_{20} \cdot M_{02} + M_{11}^2}
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see \texttt{SetSystem}).

Parameter

\begin{itemize}
\item \textbf{Regions} (input iconic) \(\text{region(-array)} \sim \text{HRegionX}/\text{IHObjectX}\) Regions to be examined.
\item \textbf{M11} (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\) Product of inertia of the axes through the center parallel to the coordinate axes.
\item \textbf{M20} (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\) Moment of 2nd order (line-dependent).
\item \textbf{M02} (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\) Moment of 2nd order (column-dependent).
\item \textbf{Ia} (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\) The one main axis of inertia.
\item \textbf{Ib} (output control) \(\text{real(-array)} \sim \text{VARIANT( real )}\) The other main axis of inertia.
\end{itemize}

Attention

If \(F\) is the area of the region the mean runtime complexity is \(O(\sqrt{F})\).

Result

The operator \texttt{MomentsRegion2Nd} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem('noObjectResult',<Result>)}. The behavior in case of empty region (region is the empty set) is set via \texttt{SetSystem('emptyRegionResult',<Result>)}. If necessary an exception handling is raised.

Parallelization Information

\texttt{MomentsRegion2Nd} is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

HALCON 6.1.4
Region processing

[out] VARIANT M11  HRegionX::MomentsRegion2NdInvar ([out] VARIANT M20, [out] VARIANT M02 )

void HOperatorSetX::MomentsRegion2NdInvar ( [in] IHObjectX Regions, [out] VARIANT M11, [out] VARIANT M20, [out] VARIANT M02 )

Geometric moments of regions.
The operator MomentsRegion2NdInvar calculates the scaled moments (M20, M02) and the product of inertia of the axes through the center parallel to the coordinate axes (M11).

**Calculation:** Z₀ and S₀ are the coordinates of the center of a region R with the area F. Then the moments Mᵢⱼ are defined by:

\[ M_{i,j} = \frac{1}{F^2} \sum_{(Z,S) \in R} (Z - Z₀)^i(S - S₀)^j \]

wherein Z and S run through all pixels of the region R.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

**Attention**

**Parameter**

- **Regions** (input iconic) region(-array) ~ HRegionX / IHObjectX
  Regions to be examined.
- **M11** (output control) real(-array) ~ VARIANT (real)
  Product of inertia of the axes through the center parallel to the coordinate axes.
- **M20** (output control) real(-array) ~ VARIANT (real)
  Moment of 2nd order (line-dependent).
- **M02** (output control) real(-array) ~ VARIANT (real)
  Moment of 2nd order (column-dependent).

**Complexity**

If \( F \) is the area of the region the mean runtime complexity is \( O(\sqrt{F}) \).

**Result**

The operator MomentsRegion2NdInvar returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem ('noObjectResult', <Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem ('emptyRegionResult', <Result>). If necessary an exception handling is raised.

**Parallelization Information**

MomentsRegion2NdInvar is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection

**See also**

EllipticAxis

MomentsRegion2Nd
10.4. FEATURES

Region processing

**Module**

```plaintext
[out] VARIANT PHI1 HRegionX.MomentsRegion2NdRelInvar
([out] VARIANT PHI2 )
```

```plaintext
void HOperatorSetX.MomentsRegion2NdRelInvar
([in] IHObjectX Regions, [out] VARIANT PHI1, [out] VARIANT PHI2 )
```

Geometric moments of regions.
The operator `MomentsRegion2NdRelInvar` calculates the scaled relative moments (PHI1, PHI2).

**Calculation:** The moments PHI1 and PHI2 are defined by:

\[
PHI1 = V_{20} + V_{02} \\
PHI2 = (V_{20} + V_{02})^2 + V_{11}^2
\]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem`).

**Attention**

- **Regions** (input iconic) ..........................region(-array) \( \sim \) HRegionX / IHObjectX
  Regions to be examined.
- **PHI1** (output control) ..........................real(-array) \( \sim \) VARIANT( real )
  Moment of 2nd order.
- **PHI2** (output control) ..........................real(-array) \( \sim \) VARIANT( real )
  Moment of 2nd order.

**Result**
The operator `MomentsRegion2NdRelInvar` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult',<Result>)`. The behavior in case of empty region (the region is the empty set) is set via `SetSystem('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

**Parallelization Information**

`MomentsRegion2NdRelInvar` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- Threshold, Regiongrowing, Connection
- See also
- EllipticAxis

**Alternatives**

- `MomentsRegion2Nd`
- Module

Region processing

```plaintext
[out] VARIANT M21 HRegionX.MomentsRegion3Rd ([out] VARIANT M12, 
[out] VARIANT M03, [out] VARIANT M30 )
```

```plaintext
void HOperatorSetX.MomentsRegion3Rd
([in] IHObjectX Regions, 
[out] VARIANT M21, [out] VARIANT M12, [out] VARIANT M03, [out] VARIANT M30 )
```

Geometric moments of regions.
The operator `MomentsRegion3Rd` calculates the translation-invariant central moments (M21, M12, M03, M30) of order \( p + q \).

HALCON 6.1.4
CHAPTER 10. REGIONS

Calculation: $x$ and $y$ are the coordinates of the center of a region $R$ with the area $Z$. Then the moments $M_{pq}$ are defined by:

$$M_{pq} = \sum_{i=1}^{n} MZ(x_i, y_i)(x_i - x)^p(y_i - y)^q$$

wherein are $x = \frac{\min_{x_i}}{\max_{x_i}}$ and $y = \frac{\min_{y_i}}{\max_{y_i}}$.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

**Parameter**

- **Regions** (input iconic) ~ region(-array) $\to$ HRegionX / IObjectX
  - Regions to be examined.
- **M21** (output control) ~ VARIANT (real)
  - Moment of 3rd order (line-dependent).
- **M12** (output control) ~ VARIANT (real)
  - Moment of 3rd order (column-dependent).
- **M03** (output control) ~ VARIANT (real)
  - Moment of 3rd order (column-dependent).
- **M30** (output control) ~ VARIANT (real)
  - Moment of 3rd order (line-dependent).

**Complexity**

If $Z$ is the area of the region the mean runtime complexity is $O(\sqrt{Z})$.

**Result**

The operator **MomentsRegion3Rd** returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem** (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via **SetSystem** (‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

**Parallelization Information**

**MomentsRegion3Rd** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- Threshold, RegionGrowing, Connection
- See also

**Alternatives**

- **MomentsRegion2Nd**

**Module**

- Region processing

---

The operator **MomentsRegion3RdInvar** calculates the scale-invariant moments ($M_{21}, M_{12}, M_{03}, M_{30}$).

**Calculation:** Then the moments $M_{ij}$ are defined by:

$$M_{pq} = \frac{\mu_{pq}}{fc^3}$$

HALCON/COM Reference Manual, 2005-2-1
wobei $p + q \geq 2$ und $\mu = \mu_{00} = m_{00}$ sind.

wobei $p + q \geq 2$ and $\mu = \mu_{00} = m_{00}$.

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem`).

If $Z$ is the area of the region the mean runtime complexity is $O(\sqrt{Z})$.

The operator `MomentsRegion3RdInvar` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem` (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via `SetSystem` (‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

MomentsRegion3RdInvar is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

EllipticAxis

Alternatives

MomentsRegion2Nd

Module

Region processing

 Calculation: Then the moments $I_i$ are defined by: $I_1 = \mu_{20} \mu_{02} - \mu_{11}^2$

\[
I_2 = (\mu_{03} - \mu_{21})^2 - 4(\mu_{03} \mu_{12} - \mu_{21} \mu_{03} - \mu_{12}^2)
\]

\[
I_3 = \mu_{20}(\mu_{21} \mu_{03} - \mu_{12}^2) - \mu_{11}(\mu_{30} \mu_{03} - \mu_{21} \mu_{12}) + \mu_{02}(\mu_{30} \mu_{12} - \mu_{21}^2)
\]
$I_4 = \mu_{20}\mu_{02}^3 - 6\mu_{20}\mu_{21}\mu_{12}\mu_{02} + 6\mu_{30}\mu_{12}\mu_{02}(2\mu_{11}^2 - \mu_{20}\mu_{02})$

$+ \mu_{30}\mu_{03}(6\mu_{20}\mu_{11}\mu_{02} - 8\mu_{11}^3) + 9\mu_{21}\mu_{20}\mu_{02}^2 - 18\mu_{21}\mu_{12}\mu_{20}\mu_{11}\mu_{02}$

$+ 6\mu_{21}\mu_{03}\mu_{20}(2\mu_{11}^2 - \mu_{20}\mu_{02}) + 9\mu_{21}\mu_{03}\mu_{20}^2 - 6\mu_{21}\mu_{03}\mu_{11}\mu_{02} + \mu_{23}\mu_{03}^3$

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Attention

Parameter

Regions (input iconic) region(-array) $\sim$ HRegionX / IObjectX

Regions to be examined.

I1 (output control) real(-array) $\sim$ VARIANT (real)

Moment of 2nd order.

I2 (output control) real(-array) $\sim$ VARIANT (real)

Moment of 2nd order.

I3 (output control) real(-array) $\sim$ VARIANT (real)

Moment of 2nd order.

I4 (output control) real(-array) $\sim$ VARIANT (real)

Moment of 3rd order.

Complexity

If $Z$ is the area of the region the mean runtime complexity is $O(\sqrt{Z})$.

Result

The operator MomentsRegionCentral returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

MomentsRegionCentral is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

EllipticAxis

Alternatives

MomentsRegion2Nd

Module

Region processing

Geometric moments of regions.

The operator MomentsRegionCentralInvar calculates the moments ($PSI1, PSI2, PSI3, PSI4$) that are invariant under translation and general linear transformations.

Calculation: Then the moments $\psi_i$ are defined by: $\psi_i = \frac{I_i}{\mu}$
If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).
In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem (‘noObjectResult’,<Result>).

Parameter

Regions (input iconic) ⇒ HRegionX / IHObjectX Region(s) to be examined.

Phi (output control) ⇒ VARIANT (real ) Orientation of region (arc measure).

Restriction: \(((-(\pi)/2) < \text{Phi} \wedge \text{Phi} \leq ((3 \cdot \pi)/2))\)

Complexity

If \( F \) is the area of a region the mean runtime complexity is \( O(\sqrt{F}) \).

Result

The operator OrientationRegion returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem (‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.

Parallelization Information

OrientationRegion is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

DispArrow

Possible Successors

See also

MomentsRegion2Nd, LineOrientation

Alternatives

EllipticAxis, SmallestRectangle2

Module

Region processing

```
[out] VARIANT Distance HRegionX.Roundness ([out] VARIANT Sigma, 
[out] VARIANT Roundness, [out] VARIANT Sides )
```

```
void HOperatorSetX.Roundness ([in] IHObjectX Regions, 
[out] VARIANT Distance, [out] VARIANT Sigma, [out] VARIANT Roundness, 
[out] VARIANT Sides )
```

Shape factors from contour.

The operator Roundness examines the distance between the contour and the center of the area. In particular the mean distance (Distance), the deviation from the mean distance (Sigma) and two shape features derived therefrom are determined. Roundness is the relation between mean value and standard deviation, and Sides indicates the number of polygon pieces if a regular polygon is concerned.

The contour for calculating the features is determined depending on the global neighborhood (see SetSystem).

Calculation:

If \( p \) is the center of the area, \( p - j \) the pixels and \( F \) the area of the contour.

\[
\text{Distance} = \frac{1}{F} \sum ||p - p - j||
\]

\[
\text{Sigma}^2 = \frac{1}{F} \sum (||p - p - j|| - \text{Distance})^2
\]

\[
\text{Roundness} = 1 - \frac{\text{Sigma}}{\text{Distance}}
\]
10.4. FEATURES

\[ Sides = 1.4111 \left( \frac{Distance}{\Sigma} \right)^{0.4724} \]

If more than one region is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input.

In case of empty region all parameters have the value 0.0 if no other behavior was set (see SetSystem).

\[ \text{Attention} \]

Parameter

- Regions (input iconic) \( \rightarrow \) HRegionX / IHOBJECTX Region(s) to be examined.
- Distance (output control) \( \rightarrow \) real(-array) \( \sim \) VARIANT (real)
  Mean distance from the center.
  **Restriction**: \( (\text{Distance} \geq 0.0) \)
- Sigma (output control) \( \rightarrow \) real(-array) \( \sim \) VARIANT (real)
  Standard deviation of Distance.
  **Restriction**: \( (\Sigma \geq 0.0) \)
- Roundness (output control) \( \rightarrow \) real(-array) \( \sim \) VARIANT (real)
  Shape factor for roundness.
  **Restriction**: \( (\text{Roundness} \leq 1.0) \)
- Sides (output control) \( \rightarrow \) real(-array) \( \sim \) VARIANT (real)
  Number of polygon sides.
  **Restriction**: \( (\text{Sides} \geq 0) \)

**Complexity**

If \( F \) is the area of a region the mean runtime complexity is \( O(\sqrt{F}) \).

**Result**

The operator Roundness returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>), the behavior in case of empty region is set via SetSystem('emptyRegionResult',<Result>). If necessary an exception handling is raised.

**Parallelization Information**

Roundness is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection

**Alternatives**

Compactness

**References**


**Module**

Region processing

\[
\text{HRegionX.RunLengthDistribution} \]

\[
\text{void HOperatorSetX.RunLengthDistribution ([in] IHOBJECTX Region,}
\text{[out] VARIANT Foreground, [out] VARIANT Background })
\]

Distribution of runs needed for runlength encoding of a region.

The operator RunLengthDistribution calculates the distribution of the runs of a region of the foreground and background. The frequency of the occurrence of a certain length is calculated. Runs of infinite length are not counted. Therefore the background are the holes of the region. As many values are passed as set by the maximum
length of fore- or background, respectively. The length of both tuples usually differs. The first entry of the tuples is always 0 (no runs of the length 0). If there are no blanks the empty tuple is passed at Background. Analogously the empty tuple is passed in case of an empty region at Foreground.

**Parameter**

- **Region** (input iconic) Region to be examined.
- **Foreground** (output control) Length distribution of the region (foreground).
- **Background** (output control) Length distribution of the background.

**Complexity**

If \( n \) is the number of runs of the region the runtime complexity is \( O(n) \).

**Result**

The operator RunlengthDistribution returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem (‘noObjectResult’,<Result>). If more than one region is passed an exception handling is raised.

**Parallelization Information**

RunlengthDistribution is reentrant and processed without parallelization.

**Possible Predecessors**

Threshold, SelectObj

**See also**

RunlengthFeatures

**Alternatives**

RunlengthFeatures

**Module**

Region processing

```plaintext
[out] VARIANT NumRuns HRegionX.RunlengthFeatures
([out] VARIANT KFactor, [out] VARIANT LFactor, [out] VARIANT MeanLength,
[out] VARIANT Bytes )

void HOperatorSetX.RunlengthFeatures ([in] IHObjectX Regions,
[out] VARIANT NumRuns, [out] VARIANT KFactor, [out] VARIANT LFactor,
[out] VARIANT MeanLength, [out] VARIANT Bytes )
```

**Characteristic values for runlength coding of regions.**

The operator RunlengthFeatures calculates for every input region from Regions the number of runs necessary for storing this region with the aid of runlength coding. Furthermore the so-called ”K-factor” is determined, which indicates by how much the number of runs differs from the ideal of the square in which this value is 1.0.

The K-factor (KFactor) is calculated according to the formula:

\[
KFactor = \frac{\text{NumRuns}}{\sqrt{\text{Area}}}
\]

wherein Area indicates the area of the region. It should be noted that the K-factor can be smaller than 1.0 (in case of long horizontal regions).

The L-factor (LFactor) indicates the mean number of runs for each line index occurring in the region. MeanLength indicates the mean length of the runs. The parameter Bytes indicates how many bytes are necessary for coding the region with runlengths.

**Attention**

All features calculated by the operator RunlengthFeatures are not rotation invariant because the runlength coding depends on the direction. The operator RunlengthFeatures does not serve for calculating shape features but for controlling and analysing the efficiency of the runlength coding.
10.4. FEATURES

Parameter

- **Regions** (input iconic) .................. region(-array) \(\sim\) HRegionX / IHObjectX
  Regions to be examined.

- **NumRuns** (output control) .................. integer(-array) \(\sim\) VARIANT ( integer )
  Number of runs.
  **Restriction:** \((0 \leq \text{NumRuns})\)

- **KFactor** (output control) .................. real(-array) \(\sim\) VARIANT ( real )
  Storing factor in relation to a square.
  **Restriction:** \((0 \leq \text{KFactor})\)

- **LFactor** (output control) .................. real(-array) \(\sim\) VARIANT ( real )
  Mean number of runs per line.
  **Restriction:** \((0 \leq \text{LFactor})\)

- **MeanLength** (output control) ............. real(-array) \(\sim\) VARIANT ( real )
  Mean length of runs.
  **Restriction:** \((0 \leq \text{MeanLength})\)

- **Bytes** (output control) ................... integer(-array) \(\sim\) VARIANT ( integer )
  Number of bytes necessary for coding the region.
  **Restriction:** \((0 \leq \text{Bytes})\)

Complexity

The mean runtime complexity is \(O(1)\).

Result

The operator **RunlengthFeatures** returns the value TRUE if the input is not empty. If necessary an exception handling is raised.

Parallelization Information

**RunlengthFeatures** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- **Threshold**, **RegionGrowing**, **Connection**, **RunlengthFeatures**

See also

- **RunlengthFeatures**, **RunlengthDistribution**

Module

Region processing

Choose all regions containing a given pixel.

The operator **SelectRegionPoint** selects all regions from **Regions** containing the test pixel \((\text{Row}, \text{Column})\), i.e.:

\[|\text{Regions}| \cap \{(\text{Row}, \text{Column})\}| = 1\]

Attention

If the regions overlap more than one region might contain the pixel. In this case all these regions are returned. If no region contains the indicated pixel the empty tuple (= no region) is returned.

Parameter

- **Regions** (input iconic) .................. region \(\sim\) HRegionX / IHObjectX
  Regions to be examined.

- **DestRegions** (output iconic) ............... region \(\sim\) HRegionX / HUntypedObjectX
  All regions containing the test pixel.
 CHAPTER 10. REGIONS

Row (input control) ........................................................ point.y ~ long / VARIANT  
Line index of the test pixel.  
Default Value : 100
(lin)

Column (input control) .................................................. point.x ~ long / VARIANT  
Column index of the test pixel.  
Default Value : 100
(lin)

Example

read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible',',',WindowHandle)
disp_image(Image)
regiongrowing(Image,Seg,3,3,5,0)
set_color(WindowHandle,'red')
set_draw(WindowHandle,'margin')
Button := 1
while (Button = 1)
  fwrite_string(FileId,'Select the region with the mouse (End right button)')
  fwnew_line(FileId)
  get_mbutton(WindowHandle,Row,Column,Button)
  select_region_point(Seg,Single,Row,Column)
  disp_region(Single,WindowHandle)
endwhile

Complexity

If $F$ is the area of the region and $N$ is the number of regions, the mean runtime complexity is $O(ln(\sqrt{F}) \ast N)$.

Result

The operator SelectRegionPoint returns the value TRUE if the parameters are correct. The behavior in case of empty input (no input regions available) is set via the operator SetSystem ('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

SelectRegionPoint is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, Regiongrowing, Connection

See also

GetMbutton, GetMposition

Alternatives

TestRegionPoint

Module

Region processing

Pose relation of regions.

The operator SelectRegionSpatial chooses the regions from Regions2 which are sufficient for the neighboring relation Direction. The regions to be examined have to be passed in Regions1 or Regions2, respectively. Regions1 can have three different states:
• **Regions1** is empty:
  In this case all regions in **Regions2** are permutatively checked for neighborhood.

• **Regions1** consists of one region:
  The regions of **Regions1** are compared to all regions in **Regions2**.

• **Regions1** consists of the same number of regions as **Regions2**:
  The regions at the n-th position in **Regions1** and **Regions2** are each checked for a neighboring relation.

Possible values for **Direction** are:

- **'left'**: **Regions2** is left of **Regions1**
- **'right'**: **Regions2** is right of **Regions1**
- **'above'**: **Regions2** is above **Regions1**
- **'below'**: **Regions2** is below **Regions1**

The operator **SelectRegionSpatial** calculates the centers of the regions to be compared and decides according to the angle between the center straight lines and the x axis whether the direction relation is fulfilled. The relation is fulfilled within the area of -45 degree to +45 degree around the coordinate axes. Thus, the direction relation can be understood in such a way that the center of the second region must be located left (or right, above, below) of the center of the first region. The indices of the regions fulfilling the direction relation are located at the n-th position in **RegionIndex1** and **RegionIndex2**, i.e., the region with the index **RegionIndex2[n]** has the indicated relation with the region with the index **RegionIndex1[n]**. Access to regions via the index can be obtained via the operator **CopyObj**.

---

**Attention**

**Parameter**

- **Regions1** (input iconic) .......................... region(-array)  \(\sim HRegionX / IHObjectX\)
  Starting regions
- **Regions2** (input iconic) .......................... region(-array)  \(\sim HRegionX / IHObjectX\)
  Comparative regions
- **Direction** (input control) .......................... string  \(\sim String / VARIANT\)
  Desired neighboring relation.
  **Default Value**: 'left'
  **List of values**: Direction \(\in\) {'left', 'right', 'above', 'below'}
- **RegionIndex1** (output control) ...................... integer  \(\sim VARIANT\) (integer)
  Indices in the input tuples (**Regions1** or **Regions2**), respectively.
- **RegionIndex2** (output control) ...................... integer  \(\sim VARIANT\) (integer)
  Indices in the input tuples (**Regions1** or **Regions2**), respectively.

**Result**

The operator **SelectRegionSpatial** returns the value TRUE if **Regions2** is not empty. The behavior in case of empty parameter **Regions2** (no input regions available) is set via the operator **SetSystem** ('noObjectResult',<Result>). The behavior in case of empty region (the region is the empty set) is set via **SetSystem** ('emptyRegionResult',<Result>). If necessary an exception handling is raised.

---

**Parallelization Information**

**SelectRegionSpatial** is reentrant and processed without parallelization.

---

**Possible Predecessors**

Threshold, RegionGrowing, Connection

---

**See also**

SpatialRelation, FindNeighbors, CopyObj, ObjToInteger

---

**Alternatives**

AreaCenter, Intersection

---

**Module**

Region processing
Choose regions with the aid of shape features.

The operator SelectShape chooses regions according to shape. For each input region from Regions the indicated features (Features) are calculated. If each (Operation = 'and') or at least one (Operation = 'or') of the calculated features is within the default limits (Min,Max) the region is adapted into the output (duplicated).

Condition: Min_i ≤ Feature_i(Object) ≤ Max_i

Possible values for Features:

'area': Area of the object
'row': Row index of the center
'column': Column index of the center
'width': Width of the region
'height': Height of the region
'row1': Row index of upper left corner
'column1': Column index of upper left corner
'row2': Row index of lower right corner
'column2': Column index of lower right corner
'circularity': Circularity (see Circular)
'compactness': Compactness (see Compactness)
'contlength': Total length of contour (see operator Contlength)
'convexity': Convexity (see Convexity)
'ra': Main radius of the equivalent ellipse (see EllipticAxis)
'rb': Secondary radius of the equivalent ellipse (see EllipticAxis)
'phi': Orientation of the equivalent ellipse (see EllipticAxis)
'anisometry': Anisometry (see Eccentricity)
'bulkiness': Bulkiness (see operator Eccentricity)
'struct_factor': Structur Factor (see operator Eccentricity)
'outer_radius': Radius of smallest surrounding circle (see SmallestCircle)
'inner_radius': Radius of largest inner circle (see InnerCircle)
'dist_mean': Mean distance from the region border to the center (see operator Roundness)
'dist_deviation': Deviation of the distance from the region border from the center (see operator Roundness)
'roundness': Roundness (see operator Roundness)
'num_sides': Number of polygon sides (see operator Roundness)
'connect_num': Number of connection components (see operator ConnectAndHoles)
'holes_num': Number of holes (see operator ConnectAndHoles)
'max_diameter': Maximum diameter of the region (see operator DiameterRegion)
'orientation': Orientation of the region (see operator OrientationRegion)
'euler_number': Euler number (see operator EulerNumber)
'rect2_phi': Orientation of the smallest surrounding rectangle (see operator SmallestRectangle2)
'rect2_len1': Half the length of the smallest surrounding rectangle (see operator SmallestRectangle2)
'rect2_len2': Half the width of the smallest surrounding rectangle (see operator SmallestRectangle2)
'moments_m11': Geometric moments of the region (see operator MomentsRegion2Nd)
10.4. FEATURES

'moments.m20': Geometric moments of the region (see operator MomentsRegion2Nd)
'moments.m02': Geometric moments of the region (see operator MomentsRegion2Nd)
'moments.ia': Geometric moments of the region (see operator MomentsRegion2Nd)
'moments.ib': Geometric moments of the region (see operator MomentsRegion2Nd)
'moments.m11.invar': Geometric moments of the region (see operator MomentsRegion2NdInvar)
'moments.m20.invar': Geometric moments of the region (see operator MomentsRegion2NdInvar)
'moments.m02.invar': Geometric moments of the region (see operator MomentsRegion2NdInvar)
'moments.phi1': Geometric moments of the region (see operator MomentsRegion2NdRelInvar)
'moments.phi2': Geometric moments of the region (see operator MomentsRegion2NdRelInvar)
'moments.m21': Geometric moments of the region (see operator MomentsRegion3Rd)
'moments.m12': Geometric moments of the region (see operator MomentsRegion3Rd)
'moments.m03': Geometric moments of the region (see operator MomentsRegion3Rd)
'moments.m30': Geometric moments of the region (see operator MomentsRegion3Rd)
'moments.m21.invar': Geometric moments of the region (see operator MomentsRegion3RdInvar)
'moments.m12.invar': Geometric moments of the region (see operator MomentsRegion3RdInvar)
'moments.m03.invar': Geometric moments of the region (see operator MomentsRegion3RdInvar)
'moments.m30.invar': Geometric moments of the region (see operator MomentsRegion3RdInvar)
'moments.i1': Geometric moments of the region (see operator MomentsRegionCentral)
'moments.i2': Geometric moments of the region (see operator MomentsRegionCentral)
'moments.i3': Geometric moments of the region (see operator MomentsRegionCentral)
'moments.i4': Geometric moments of the region (see operator MomentsRegionCentral)
'moments.psi1': Geometric moments of the region (see operator MomentsRegionCentralInvar)
'moments.psi2': Geometric moments of the region (see operator MomentsRegionCentralInvar)
'moments.psi3': Geometric moments of the region (see operator MomentsRegionCentralInvar)
'moments.psi4': Geometric moments of the region (see operator MomentsRegionCentralInvar)

If only one feature (Features) is used the value of Operation is meaningless. Several features are processed in the sequence in which they are entered.

Attention Parameter

- Regions (input iconic) .......................... region ~ HRegionX / IHObjectX
  Regions to be examined.
- SelectedRegions (output iconic) ................. region ~ HRegionX / HUntypedObjectX
  Regions fulfilling the condition.
- Features (input control) ......................... string(-array) ~ VARIANT ( string )
  Shape features to be checked.
  Default Value : 'area'

  List of values: features ∈ {'area', 'row', 'column', 'width', 'height', 'row1', 'column1', 'row2',
  'column2', 'circularity', 'compactness', 'contlength', 'convexity', 'ra', 'rb', 'phi', 'anisometry', 'bulkiness',
  'struct_factor', 'outer_radius', 'inner_radius', 'max_diameter', 'dist_mean', 'dist_deviation', 'roundness',
  'num_sides', 'orientation', 'connect_num', 'holes_num', 'euler_number', 'rect2.phi', 'rect2_len1', 'rect2_len2',
  'moments.m11', 'moments.m20', 'moments.m02', 'moments.m11.invar',
  'moments.m20.invar', 'moments.m02.invar', 'moments.phi1', 'moments.phi2', 'moments.m11',
  'moments.m12', 'moments.m03', 'moments.m30', 'moments.m21.invar', 'moments.m12.invar',
  'moments.m03.invar', 'moments.m30.invar', 'moments.i1', 'moments.i2', 'moments.i3', 'moments.i4',
  'moments.psi1', 'moments.psi2', 'moments.psi3', 'moments.psi4'}

- Operation (input control) ...................... string ~ String / VARIANT
  Linkage type of the individual features.
  Default Value : 'and'

  List of values: operation ∈ {'and', 'or'}
Min (input control) .............................................. real(-array) \sim \text{VARIANT} ( \text{integer, real, string} )
Lower limits of the features or 'min'.
 Default Value : 150.0
 Typical range of values : $0.0 \leq \text{Min} \leq 0.0$
 Minimum Increment : 0.001
 Recommended Increment : 1.0

Max (input control) .............................................. real(-array) \sim \text{VARIANT} ( \text{integer, real, string} )
Upper limits of the features or 'max'.
 Default Value : 99999.0
 Typical range of values : $0.0 \leq \text{Max} \leq 0.0$
 Minimum Increment : 0.001
 Recommended Increment : 1.0
 Restriction : (\text{Max} \geq \text{Min})

Example

/* where are the eyes of the ape ? */
read_image(Image 'affe')
threshold(Image, S1, 160, 255)
connection(S1, S2)
select_shape(S2, Eyes, ['area', 'anisometry'], 'and', [500, 1.0], [50000, 1.7])
disp_region(Eyes, WindowHandle)

Result
The operator SelectShape returns the value TRUE if the input is not empty. The behavior in case of empty input (no input objects available) is set via the operator SetSystem('noObjectResult', <Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem ('emptyRegionResult', <Result>). If necessary an exception handling is raised.

Parallelization Information
SelectShape is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Threshold, RegionGrowing, Connection, RunlengthFeatures

Possible Successors
SelectShape, SelectGray, ShapeTrans, ReduceDomain, CountObj

See also
AreaCenter, Circularity, Compactness, Contlength, Convexity, EllipticAxis, Eccentricity, InnerCircle, SmallestCircle, SmallestRectangle1, SmallestRectangle2, Roundness, ConnectAndHoles, DiameterRegion, OrientationRegion, MomentsRegion2Nd, MomentsRegion2NdInvar, MomentsRegion2NdRelInvar, MomentsRegion3Rd, MomentsRegion3RdInvar, MomentsRegionCentral, MomentsRegionCentralInvar, SelectObj

Module
Region processing

Choose regions having a certain relation to each other.

The operator SelectShapeProto selects regions based on certain relations between the regions. Every region from Regions is compared to the union of regions from Pattern. The limits (Min and Max) are specified absolutely or in percent (0..100), depending on the feature. Possible values for Feature are:

HALCON/COM Reference Manual, 2005-2-1
'distance_dilate' The minimum distance in the maximum norm from the edge of Pattern to the edge of every region from Regions is determined (see DistanceRrMinDil).

'distance_contour' The minimum Euclidean distance from the edge of Pattern to the edge of every region from Regions is determined. (see DistanceRrMin).

'distance_center' The Euclidean distance from the center of Pattern to the center of every region from Regions is determined.

covers' It is examined how well the region Pattern fits into the regions from Regions. If there is no shift so that Pattern is a subset of Regions the overlap is 0. If Pattern corresponds to the region after a corresponding shift the overlap is 100. Otherwise the area of the opening of Regions with Pattern is put into relation with the area of Regions (in percent).

'fits' It is examined whether Pattern can be shifted in such a way that it fits in Regions. If this is possible the corresponding region is copied from Regions. The parameters Min and Max are ignored.

'overlaps_abs' The area of the intersection of Pattern and every region in Regions is computed.

'overlaps_rel' The area of the intersection of Pattern and every region in Regions is computed. The relative overlap is the ratio of the area of the intersection and the area of the respective region in Regions (in percent).

--- Attention ---

--- Parameter ---

▷ Regions (input iconic) ......................... region(-array) \(\sim\) HRegionX / IObjectX Regions to be examined.

▷ Pattern (input iconic) ......................... region(-array) \(\sim\) HRegionX / IObjectX Region compared to Regions.

▷ SelectedRegions (output iconic) ............. region(-array) \(\sim\) HRegionX / UntypedObjectX Regions fulfilling the condition.

▷ Feature (input control) ....................... string(-array) \(\sim\) VARIANT (string) Shape features to be checked.

Default Value : 'covers'

List of values : Feature \(\in\) \{ 'distance_center', 'distance_dilate', 'distance_contour', 'covers', 'fits', 'overlaps_abs', 'overlaps_rel' \}

▷ Min (input control) .............................. number \(\sim\) VARIANT (integer, real) Lower border of feature.

Default Value : 50.0

Suggested values : Min \(\in\) \{ 0.0, 1.0, 5.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 99.0, 100.0, 200.0, 400.0 \}

Minimum Increment : 0.001

Recommended Increment : 5.0

▷ Max (input control) .............................. number \(\sim\) VARIANT (integer, real) Upper border of the feature.

Default Value : 100.0

Suggested values : Max \(\in\) \{ 0.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 99.0, 100.0, 200.0, 300.0, 400.0 \}

Minimum Increment : 0.001

Recommended Increment : 5.0

--- Result ---

The operator SelectShapeProto returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem ('noObjectResult', <Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem ('emptyRegionResult', <Result>). If necessary an exception handling is raised.

--- Parallelization Information ---

SelectShapeProto is reentrant and processed without parallelization.

--- Possible Predecessors ---

Connection, DrawRegion, GenCircle, GenRectangle1, GenRectangle2, GenEllipse

--- Possible Successors ---

SelectGray, ShapeTrans, ReduceDomain, CountObj
Select regions of a given shape.

The operator **SelectShapeStd** compares the shape of the given regions with default shapes. If the region has a similar shape it is adopted into the output. Possible values for **Shape** are:

- **'max_area'** The largest region is selected.
- **'rectangle1'** The surrounding rectangle parallel to the coordinate axes is determined via the operator **SmallestRectangle1**. If the area difference in percent is larger than **Percent** the region is adopted.
- **'rectangle1'** The smallest surrounding rectangle with any orientation is determined via the operator **SmallestRectangle2**. If the area difference in percent is larger than **Percent** the region is adopted.

### Parameter

- **Regions** (input iconic) region(-array) ~ HRegionX / IObjectX
  
  Input regions to be selected.

- **SelectedRegions** (output iconic) region(-array) ~ HRegionX / HUntypedObjectX
  
  Regions with desired shape.

- **Shape** (input control) string ~ String / VARIANT
  
  Shape features to be checked.

  **Default Value:** 'max_area'

  **List of values:** Shape ∈ \{ 'max_area', 'rectangle1', 'rectangle2' \}

- **Percent** (input control) real ~ double / VARIANT
  
  Similarity measure.

  **Default Value:** 70.0

  **Suggested values:** Percent ∈ \{ 10.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 100.0 \}

  **Typical range of values:** 0.0 ≤ Percent ≤ 0.0 (lin)

  **Minimum Increment:** 0.1

  **Recommended Increment:** 10.0

### Parallelization Information

**SelectShapeStd** is **reentrant** and processed **without** parallelization.

### Possible Predecessors

- Threshold, RegionGrowing, Connection, SmallestRectangle1, SmallestRectangle2

### See also

- SmallestRectangle1, SmallestRectangle2

### Alternatives

- Intersection, Complement, AreaCenter
10.4. FEATURES

Smallest surrounding circle of a region.

The operator `SmallestCircle` determines the smallest surrounding circle of a region, i.e., the circle with the smallest area of all circles containing the region. For this circle the center (Row,Column) and the radius (Radius) are calculated. The procedure is applied when, for example, the location and size of circular objects (e.g., coins) which, however, are not homogeneous inside or have broken edges due to bad segmentation, has to be determined. The output of the procedure is selected in such a way that it can be used as input for the HALCON procedures `DispCircle` and `GenCircle`.

If several regions are passed in `Regions` corresponding tuples are returned as output parameter. In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem`).

---

### Parameter

- **Regions** (input iconic) .......... region(-array)  \(\sim\) **HRegionX / IHObjectX**
  - Regions to be examined.
- **Row** (output control) .............. circle.center.y(-array)  \(\sim\) **VARIANT** (real)
  - Line index of the center.
- **Column** (output control) .......... circle.center.x(-array)  \(\sim\) **VARIANT** (real)
  - Column index of the center.
- **Radius** (output control) .......... circle.radius(-array)  \(\sim\) **VARIANT** (real)
  - Radius of the surrounding circle.

**Restriction**: \(\text{Radius} \geq 0\)

### Example

```plaintext
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','Margin',WindowHandle)
regiongrowing(Image,Seg,5,5,6,100:)
select_shape(Seg,H,'area','and',100,2000)
smallest_circle(H,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius)
set_draw(WindowHandle,'margin')
disp_region(Circles,WindowHandle)
```

---

### Complexity

If \(F\) is the area of the region, then the mean runtime complexity is \(O(\sqrt{F})\).

### Result

The operator `SmallestCircle` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem` ('noObjectResult',<Result>). The behavior in case of empty region (the region is the empty set) is set via `SetSystem` ('emptyRegionResult',<Result>). If necessary an exception handling is raised.

---

### Parallelization Information

`SmallestCircle` is reentrant and automatically parallelized (on tuple level).

---

### Possible Predecessors

`Threshold`, `Regiongrowing`, `Connection`, `RunlengthFeatures`

### Possible Successors

`GenCircle`, `DispCircle`

See also: `SetShape`, `SelectShape`, `InnerCircle`
**Alternatives**

EllipticAxis, SmallestRectangle1, SmallestRectangle2

**Module**

Region processing

```c
[out] VARIANT Row1 HRegionX.SmallestRectangle1 ([out] VARIANT Column1,
[out] VARIANT Row2, [out] VARIANT Column2 )

void HOperatorSetX.SmallestRectangle1 ([in] IHOBJECTX Regions,
[out] VARIANT Row1, [out] VARIANT Column1, [out] VARIANT Row2,
[out] VARIANT Column2 )
```

**Surrounding rectangle parallel to the coordinate axes.**

The operator SmallestRectangle1 calculates the surrounding rectangle of all input regions (parallel to the coordinate axes). The surrounding rectangle is described by the coordinates of the corner pixels (Row1, Column1, Row2, Column2)

If more than one region is passed in Regions, the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input. In case of empty region all parameters have the value 0 if no other behavior was set (see SetSystem).

**Attention**

In case of empty region the result of Row1, Column1, Row2 and Column2 (all are 0) can lead to confusion.

**Parameter**

- **Regions** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . region(-array) ~ HRegionX / IHOBJECTX
  Regions to be examined.

- **Row1** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . rectangle.origin.y(-array) ~ VARIANT( integer )
  Line index of upper left corner point.

- **Column1** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . rectangle.origin.x(-array) ~ VARIANT( integer )
  Column index of upper left corner point.

- **Row2** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . rectangle.corner.y(-array) ~ VARIANT( integer )
  Line index of lower right corner point.

- **Column2** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . rectangle.corner.x(-array) ~ VARIANT( integer )
  Column index of lower right corner point.

**Complexity**

If $F$ is the area of the region the mean runtime complexity is $O(\sqrt{F})$.

**Result**

The operator SmallestRectangle1 returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem (’noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via SetSystem(’emptyRegionResult’,<Result>). If necessary an exception handling is raised.

**Parallelization Information**

SmallestRectangle1 is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, RegionGrowing, Connection, RunLengthFeatures

**Possible Successors**

DispRectangle1, GenRectangle1

See also

- SelectShape

**Alternatives**

SmallestRectangle2, AreaCenter
Smallest surrounding rectangle with any orientation.

The operator `SmallestRectangle2` determines the smallest surrounding rectangle of a region, i.e., the rectangle with the smallest area of all rectangles containing the region. For this rectangle the center, the inclination and the two radii are calculated.

The procedure is applied when, for example, the location of a scenery of several regions (e.g., printed text on a rectangular paper or in rectangular print (justified lines)) must be found. The parameters of `SmallestRectangle2` are chosen in such a way that they can be used directly as input for the HALCON-procedures `DispRectangle2` and `GenRectangle2`.

If more than one region is passed in `Regions` the results are stored in tuples, the index of a value in the tuple corresponding to the index of a region in the input. In case of empty region all parameters have the value 0.0 if no other behavior was set (see `SetSystem`).

### Example

```c
read_image(Image,'fabrik')
open_window(0,0,-1,-1,'root','visible','WindowHandle')
regiongrowing(Image,Seg,5,5,6,100)
smallest_rectangle2(Seg,Row,Column,Phi,Length1,Length2)
gen_rectangle2(Rectangle,Row,Column,Phi,Length1,Length2)
set_draw(WindowHandle,'margin')
disp_region(Rectangle,WindowHandle)
```

### Complexity

If $F$ is the area of the region and $N$ is the number of supporting points of the convex hull, the runtime complexity is $O(\sqrt{F} + N^2)$.

### Result

The operator `SmallestRectangle2` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem` (‘noObjectResult’,<Result>). The behavior in case of empty region (the region is the empty set) is set via `SetSystem` (‘emptyRegionResult’,<Result>). If necessary an exception handling is raised.
 Parallelization Information

SmallestRectangle2 is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection, RunLengthFeatures

Possible Successors

DispRectangle2, GenRectangle2

See also

SmallestCircle, SetShape

Alternatives

EllipticAxis, SmallestRectangle1

Module

Region processing

 Pose relation of regions with regard to the coordinate axes.

The operator SpatialRelation selects regions located by Percent percent “left”, “right”, “above” or “below” other regions. Regions1 and Regions2 contain the regions to be compared. Regions1 can have three states:

- **Regions1** is empty:
  In this case all regions in Regions2 are permutatively checked for neighborhood.

- **Regions1** consists of one region:
  The regions of Regions1 are compared to all regions in Regions2.

- **Regions1** consists of the same number of regions as Regions2:
  Regions1 and Regions2 are checked for a neighboring relation.

The percentage Percent is interpreted in such a way that the area of the second region has to be located really left/right or above/below the region margins of the first region by at least Percent percent. The indices of the regions that fulfill at least one of these conditions are then located at the n-th position in the output parameters RegionIndex1 and RegionIndex2. Additionally the output parameters Relation1 and Relation2 contain at the n-th position the type of relation of the region pair (RegionIndex1[n], RegionIndex2[n]), i.e., region with index RegionIndex2[n] has the relation Relation1[n] and Relation2[n] with region with index RegionIndex1[n].

Possible values for Relation1 and Relation2 are:

**Relation1**: 'left', 'right' or ''
**Relation2**: 'above', 'below' or ''

In RegionIndex1 and RegionIndex2 the indices of the regions in the tuples of the input regions (Regions1 or Regions2), respectively, are entered as image identifiers. Access to chosen regions via the index can be obtained by the operator CopyObj.

**Attention**
10.4. FEATURES

Parameter

- **Regions1** (input iconic) .......... region(-array)  \(\sim\) HRegionX / HObjectX
  Starting regions.

- **Regions2** (input iconic) .......... region(-array)  \(\sim\) HRegionX / HObjectX
  Comparative regions.

- **Percent** (input control) .............. integer  \(\sim\) long / VARIANT
  Percentage of the area of the comparative region which must be located left/right or above/below the region margins of the starting region.
  
  Default Value : 50
  
  Suggested values : Percent \(\in\) \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}
  
  Typical range of values : 0 \(\leq\) Percent \(\leq\) 0
  
  Minimum Increment : 1
  
  Recommended Increment : 10
  
  Restriction : \((0 \leq \text{Percent}) \wedge (\text{Percent} \leq 100)\)

- **RegionIndex1** (output control) ........ integer  \(\sim\) VARIANT ( integer )
  Indices of the regions in the tuple of the input regions which fulfill the pose relation.

- **RegionIndex2** (output control) ........ integer  \(\sim\) VARIANT ( integer )
  Indices of the regions in the tuple of the input regions which fulfill the pose relation.

- **Relation1** (output control) .............. string  \(\sim\) VARIANT ( string )
  Horizontal pose relation in which RegionIndex2[n] stands with RegionIndex1[n].

- **Relation2** (output control) .............. string  \(\sim\) VARIANT ( string )
  Vertical pose relation in which RegionIndex2[n] stands with RegionIndex1[n].

Result

The operator **SpatialRelation** returns the value TRUE if Regions2 is not empty and Percent is correctly choosen. The behavior in case of empty parameter Regions2 (no input regions available) is set via the operator **SetSystem('noObjectResult',<Result>)**. The behavior in case of empty region (the region is the empty set) is set via **SetSystem('emptyRegionResult',<Result>)**. If necessary an exception handling is raised.

Parallelization Information

**SpatialRelation** is reentrant and processed without parallelization.

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

SelectRegionSpatial, FindNeighbors, CopyObj, ObjToInteger

Alternatives

AreaCenter, Intersection

Module

Region processing

```c
void HRegionX.TestRegionPoint ([in] long Row, [in] long Column )
```

Test if the region consists of the given point.

**TestRegionPoint** tests if at least one input region of Regions consists of the test point \((Row, Column)\).

Attention

In case of empty input (= no region) and **SetSystem('noObjectResult', 'true')** FALSE is returned (no region contains the pixel).

The test pixel is not contained in an empty region (no pixel of the region corresponds to the pixel). If all regions are empty FALSE is also returned, i.e., an empty region behaves as if it did not exist.
CHAPTER 10. REGIONS

Parameter

- **Regions** (input iconic) .......................................................\( \oplus \) region(-array) \( \Rightarrow \) HRegionX / IHObjectX
  Region(s) to be examined.

- **Row** (input control) ............................................................... point.y \( \Rightarrow \) long / VARIANT
  Line index of the test pixel.
  Default Value : 100
  Typical range of values : \( 0 \leq \text{Row} \leq 0 \) (lin)
  Minimum Increment : 1
  Recommended Increment : 1

- **Column** (input control) ...................................................... point.x \( \Rightarrow \) long / VARIANT
  Column index of the test pixel.
  Default Value : 100
  Typical range of values : \( 0 \leq \text{Column} \leq 0 \) (lin)
  Minimum Increment : 1
  Recommended Increment : 1

Complexity

If \( F \) is the area of one region and \( N \) is the number of regions, the runtime complexity is \( O(\ln(\sqrt{F}) \ast N) \).

Result

The operator **TestRegionPoint** returns the value TRUE if a region contains the test pixel. If this is not the case **TestRegionPoint** returns FALSE. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem**('noObjectResult',<Result>). If necessary an exception handling is raised.

Parallelization Information

**TestRegionPoint** is reentrant and processed without parallelization.

Possible Predecessors

Threshold, RegionGrowing, Connection

See also

**SelectRegionPoint**

Alternatives

Union1, Intersection, AreaCenter

Module

Region processing

10.5 Sets

```
[out] HRegionX RegionComplement HRegionX.Complement ( )
void HOperatorSetX.Complement ([in] IHObjectX Region,
[out] HUntypedObjectX RegionComplement )
```

Return the complement of a region.

**Complement** determines the complement of the input region(s).

If the system flag 'clip_region' is 'false' (see **SetSystem**) the complement is done virtually by setting the complement flag of **Region** to TRUE. For succeeding operations the de Morgan laws are applied while calculating results.

If the system flag 'clip_region' is 'true' the difference of the largest image processed so far (see **ResetObjDb**) and the input region is returned.

Parameter

- **Region** (input iconic) .....................................................\( \oplus \) region(-array) \( \Rightarrow \) HRegionX / IHObjectX
  Input region(s).

- **RegionComplement** (output iconic) .................................. region(-array) \( \Rightarrow \) HRegionX / HUntypedObjectX
  Complemented regions.
  Number of elements : (RegionComplement = Region)
Set 5. Sets

Result

Complement always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Complement is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successors

SelectShape

See also

Difference, Union1, Union2, Intersection, ResetObjDb, SetSystem

Module

Region processing

[out] HRegionX RegionDifference

HRegionX.Difference ([in] HRegionX Sub)

void HOperatorSetX.Difference ([in] IObjectX Region,
[in] IObjectX Sub, [out] HUntypedObjectX RegionDifference)

Calculate the difference of two regions.

Difference calculates the set-theoretic difference of two regions:

(Regions in Region) − (Regions in Sub)

The resulting region is defined as the input region (Region) with all points from Sub removed.

Attention

Empty regions are valid for both parameters. On output, empty regions may result. The value of the system flag 'store_empty_region' determines the behavior in this case.

Parameter

▷ Region (input iconic) ............................ region(-array) ～ HRegionX / IObjectX
Regions to be processed.

▷ Sub (input iconic) ............................. region(-array) ～ HRegionX / IObjectX
The union of these regions is subtracted from Region.

▷ RegionDifference (output iconic) .............. region(-array) ～ HRegionX / HUntypedObjectX
Resulting region.

Example

/* provides the region X without the points in Y */
difference(X,Y,RegionDifference)

Complexity

Let \( N \) be the number of regions, \( F_1 \) be their average area, and \( F_2 \) be the total area of all regions in Sub. Then the runtime complexity is \( O(F_1 * \log(F_1) + N * (\sqrt{F_1} + \sqrt{F_2})) \).

Result

Difference always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Difference is reentrant and processed without parallelization.

Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

HALCON 6.1.4
Calculate the intersection of two regions.

Intersection calculates the intersection of the regions in Region1 with the regions in Region2. Each region in Region1 is intersected with all regions in Region2. The order of regions in RegionIntersection is identical to the order of regions in Region1.

Attention

Empty input regions are permitted. Because empty result regions are possible, the system flag 'store empty region' should be set appropriately.

Parameter

- Region1 (input iconic) region(-array) \( \sim \) HRegionX / IObjectX
  Regions to be intersected with all regions in Region2.
- Region2 (input iconic) region(-array) \( \sim \) HRegionX / IObjectX
  Regions with which Region1 is intersected.
- RegionIntersection (output iconic) region(-array) \( \sim \) HRegionX / HUntypedObjectX
  Result of the intersection.

Number of elements: \( RegionIntersection \leq Region1 \)

Complexity

Let \( N \) be the number of regions in Region1, \( F_1 \) be their average area, and \( F_2 \) be the total area of all regions in Region2. Then the runtime complexity is \( O(F_1 \log(F_1) + N \ast (\sqrt{F_1} + \sqrt{F_2})) \).

Result

Intersection always returns TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

Intersection is reentrant and processed without parallelization.

Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring

Possible Successors

SelectShape, DispRegion

See also

Union1, Union2, Complement

Module
10.5. SETS

Union1 computes the union of all input regions and returns the result in RegionUnion.

- **Parameter**
  - Region (input iconic) ..........................region  \( \sim \) HRegionX / IObjectX
    Regions of which the union is to be computed.
  - RegionUnion (output iconic) ..................region  \( \sim \) HRegionX / HIObjectX
    Union of all input regions.

- **Number of elements**: \( (\text{RegionUnion} \leq \text{Region}) \)

- **Example**

  ```
  /* Union of segmentation results: */
  threshold(Image,Region1,128,255)
  dyn_threshold(Image,Mean,Region2,5,'light')
  concat_obj(Region1,Region2,Regions)
  union1(Regions,RegionUnion).
  ```

- **Complexity**

  Let \( F \) be the sum of all areas of the input regions. Then the runtime complexity is \( O(\log(\sqrt{F}) \ast \sqrt{F}) \).

- **Result**

  Union1 always returns TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem ('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem ('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

- **Parallelization Information**

  Union1 is reentrant and processed without parallelization.

- **Possible Predecessors**

  Threshold, Connection, Regiongrowing, Pouring

- **Possible Successors**

  SelectShape, DispRegion

- **See also**

  Intersection, Complement

- **Alternatives**

  Union2

- **Module**

  Region processing

---

Union2 computes the union of the region in Region1 with all regions in Region2. This means that Union2 is not commutative!

- **Parameter**
  - Region1 (input iconic) ..........................region(-array)  \( \sim \) HRegionX / IObjectX
    Region for which the union with all regions in Region2 is to be computed.
  - Region2 (input iconic) ..........................region(-array)  \( \sim \) HRegionX / IObjectX
    Regions which should be added to Region1.
  - RegionUnion (output iconic) ..................region(-array)  \( \sim \) HRegionX / HIObjectX
    Resulting regions.

- **Number of elements**: \( (\text{RegionUnion} = \text{Region1}) \)
Let $F$ be the sum of all areas of the input regions. Then the runtime complexity is $O(\log(\sqrt{F}) \times \sqrt{F})$.

**Result**

Union2 always returns TRUE. The behavior in case of empty input (no regions given) can be set via `SetSystem` ("noObjectResult",<Result>) and the behavior in case of an empty input region via `SetSystem` ("emptyRegionResult",<Result>). If necessary, an exception handling is raised.

**Parallelization Information**

Union2 is reentrant and processed without parallelization.

**Possible Predecessors**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successors**

SelectShape, DispRegion

**See also**

Intersection, Complement

**Alternatives**

Union1

Region processing

### 10.6 Transformation

```c
[out] HRegionX BackgroundRegions HRegionX.BackgroundSeg ( )
void HOperatorSetX.BackgroundSeg ([in] IHObjectX Foreground,
[out] HUntypedObjectX BackgroundRegions )
```

Determine the connected components of the background of given regions.

BackgroundSeg determines connected components of the background of the foreground regions given in Foreground. This operator is normally used after an edge operator in order to determine the regions enclosed by the extracted edges. The connected components are determined using 4-neighborhood.

**Parameter**

- **Foreground** (input iconic) ...................... region(-array) $\leadsto$ HRegionX / IHObjectX
- **BackgroundRegions** (output iconic) ...................... region $\leadsto$ HRegionX / HUntypedObjectX

**Example**

/* Simulation of background_seg: */
background_seg(Foreground,BackgroundRegions):
  complement(Foreground,Background)
  get_system('neighborhood',Save)
  set_system('neighborhood',4)
  connection(Background,BackgroundRegions)
  clear_obj(Background)
  set_system('neighborhood',Save).

/* Segmentation with edge filter: */
read_image(Image,'fabrik')
sobel_dir(Image,Sobel,Dir,'sum_sqrt',3)
threshold(Sobel,Edges,20,255)
skeleton(Edges,Margins)
background_seg(Margins,Regions).
Let $F$ be the area of the background, $H$ and $W$ be the height and width of the image, and $N$ be the number of resulting regions. Then the runtime complexity is $O(H + \sqrt{F} \times \sqrt{N})$.

**Result**

`BackgroundSeg` always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`. If necessary, an exception handling is raised.

**Parallelization Information**

`BackgroundSeg` is reentrant and processed without parallelization.

**Possible Predecessors**

Threshold, Connection, RegionGrowing, Pouring, ClassNdimNorm

**Possible Successors**

SelectShape

**See also**

Threshold, HysteresisThreshold, Skeleton, ExpandRegion, SetSystem, SobelAmp, EdgesImage, Roberts, BandpassImage

**Alternatives**

Complement, Connection

**Module**

Region processing

### Clip Region to a Rectangle

`ClipRegion` clips the input regions to the rectangle given by the four control parameters. `ClipRegion` is more efficient than calling `Intersection` with a rectangle generated by `GenRectangle1`.

#### Parameter

- **Region** (input iconic) .............................. region(-array) $\rightarrow$ `HRegionX` / `IHObjectX` Region to be clipped.
- **RegionClipped** (output iconic) ................. region(-array) $\rightarrow$ `HRegionX` / `HUntypedObjectX` Clipped regions.
- **Row1** (input control) .............................. rectangle.origin.y $\rightarrow$ `long` / `VARIANT` Row coordinate of the upper left corner of the rectangle.
  - **Default Value** : 0
  - **Suggested values** : $Row1 \in \{0, 128, 200, 256\}$
  - **Typical range of values** : $0 \leq Row1 \leq 0$(lin)
- **Column1** (input control) ............................. rectangle.origin.x $\rightarrow$ `long` / `VARIANT` Column coordinate of the upper left corner of the rectangle.
  - **Default Value** : 0
  - **Suggested values** : $Column1 \in \{0, 128, 200, 256\}$
  - **Minimum Increment** : 1
  - **Recommended Increment** : 10
- **Row2** (input control) .............................. rectangle.corner.y $\rightarrow$ `long` / `VARIANT` Row coordinate of the lower right corner of the rectangle.
  - **Default Value** : 256
  - **Suggested values** : $Row2 \in \{128, 200, 256, 512\}$
Column2 (input control) ........................................ rectangle.corner.x  \rightarrow long / VARIANT
Column coordinate of the lower right corner of the rectangle.
Default Value : 256
Suggested values : Column2 \in \{128, 200, 256, 512\}
Typical range of values : 0 \leq Column2 \leq 0(lin)
Minimum Increment : 1
Recommended Increment : 10

\underline{Result} 
ClipRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

\underline{Parallelization Information} 
ClipRegion is reentrant and automatically parallelized (on tuple level).

\underline{Possible Predecessors} 
Threshold, Connection, RegionGrowing, Pouring

\underline{Possible Successors} 
SelectShape, DispRegion

\underline{Alternatives} 
Intersection, GenRectangle1, ClipRegionRel

\underline{Module} 
Region processing

\begin{verbatim}
[out] HRegionX RegionClipped HRegionX.ClipRegionRel ([in] long Top,
void HOperatorSetX.ClipRegionRel ([in] IHObjectX Region,
[out] HUntypedObjectX RegionClipped, [in] VARIANT Top, [in] VARIANT Bottom,
\end{verbatim}

Clip a region relative to its size.

ClipRegionRel clips a region to a rectangle lying within the region. The size of the rectangle is determined by the enclosing rectangle of the region, which is reduced by the values given in the four control parameters. All four parameters must contain numbers larger or equal to zero, and determine by which amount the rectangle is reduced at the top (Top), at the bottom (Bottom), at the left (Left), and at the right (Right). If all parameters are set to zero, the region remains unchanged.

\underline{Parameter} 

\begin{itemize}
\item Region (input iconic) ........................................ region(-array) \rightarrow HRegionX / IHObjectX
Regions to be clipped.
\item RegionClipped (output iconic) ............................... region(-array) \rightarrow HRegionX / HUntypedObjectX
Clipped regions.
\item Top (input control) .............................................. integer \rightarrow long / VARIANT
Number of rows clipped at the top.
Default Value : 1
Suggested values : Top \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\}
(lin)Minimum Increment : 1
Recommended Increment : 1
\item Bottom (input control) ............................................ integer \rightarrow long / VARIANT
Number of rows clipped at the bottom.
Default Value : 1
Suggested values : Bottom \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\}
(lin)Minimum Increment : 1
Recommended Increment : 1
\end{itemize}
10.6. TRANSFORMATION

- **Left** (input control) ................................................................. integer  \rightarrow long / VARIANT
  Number of columns clipped at the left.
  Default Value : 1
  Suggested values : Left \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\}
  (lin) Minimum Increment : 1
  Recommended Increment : 1

- **Right** (input control) ............................................................... integer  \rightarrow long / VARIANT
  Number of columns clipped at the right.
  Default Value : 1
  Suggested values : Right \in \{0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50\}
  (lin) Minimum Increment : 1
  Recommended Increment : 1

\textbf{ClipRegionRel} returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via \texttt{SetSystem(‘noObjectResult’,<Result>)} and the behavior in case of an empty input region via \texttt{SetSystem(‘emptyRegionResult’,<Result>)}. If necessary, an exception handling is raised.

---

**Parallelization Information**

\textbf{ClipRegionRel} is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, Connection, Regiongrowing, Pouring

**Possible Successors**

SelectShape, DispRegion

**Alternatives**

SmallestRectangle1, Intersection, GenRectangle1, ClipRegion

**Module**

---

\texttt{HRegionX} \texttt{ConnectedRegions} \texttt{HRegionX.Connection} ( )

\texttt{void HOperatorSetX.Connection ([in] HIObjectX Region,}
\texttt{ [out] HUntypedObjectX ConnectedRegions )}

*Compute connected components of a region.*

**Connection** determines the connected components of the input regions given in \texttt{Region}. The neighborhood used for this can be set via \texttt{SetSystem(‘neighborhood’,<4/8>)}). The default is 8-neighborhood, which is useful for determining the connected components of the foreground. The maximum number of connected components that is returned by \texttt{Connection} can be set via \texttt{SetSystem(‘maxConnection’,<Num>)}). The default value of 0 causes all connected components to be returned. The inverse operator of \texttt{Connection} is \texttt{Union1}.

---

**Parameter**

- **Region** (input iconic) ......................................................... region(-array)  \rightarrow HRegionX / HIObjectX
  Input region.

- **ConnectedRegions** (output iconic) ................................. region  \rightarrow HRegionX / HUntypedObjectX
  Connected components.

---

**Example**

\texttt{read_image(Image,‘affe’)}
\texttt{set_colored(WindowHandle,12)}
\texttt{threshold(Image,Light,150.0,255.0)}
\texttt{count_obj(Light,Number1)}
\texttt{fwrite_string(‘Number of regions after threshold = ’+Number1)}
\texttt{fnew_line()}
\texttt{disp_region(Light,WindowHandle)}
connection(Light,Many)
count_obj(Many,Number2)
fwrite_string('Number of regions after threshold = '+Number2)
fnew_line()
disp_region(Many,WindowHandle).

---

**Complexity**

Let $F$ be the area of the input region and $N$ be the number of generated connected components. Then the runtime complexity is $O(\sqrt{F} \cdot \sqrt{N})$.

---

**Result**

Connection always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

---

**Parallelization Information**

Connection is reentrant and processed without parallelization.

---

**Possible Predecessors**

AutoThreshold, Threshold, DynThreshold, Erosion1

---

**Possible Successors**

SelectShape, SelectGray, ShapeTrans, SetColored, Dilation1, CountObj, ReduceDomain, AddChannels

---

**See also**

SetSystem, Union1

---

**Alternatives**

BackgroundSeg

---

**Module**

Region processing

```c
[out] HImageX DistanceImage HRegionX.DistanceTransform
```

Compute the distance transformation of a region.

DistanceTransform computes for every point of the input region Region (or its complement, respectively) the distance of the point to the border of the region. The parameter Foreground determines whether the distances are calculated for all points within the region (Foreground = 'true') or for all points outside the region (Foreground = 'false'). The distance is computed for every point of the output image DistanceImage, which has the specified dimensions Width and Height. The input region is always clipped to the extent of the output image. If it is important that the distances within the entire region should be computed, the region should be moved (see MoveRegion) so that it has only positive coordinates and the width and height of the output image should be large enough to contain the region. The extent of the input region can be obtained with SmallestRectangle1.

The parameter Metric determines which metric is used for the calculation of the distances. If Metric = 'city-block', the distance is calculated from the shortest path from the point to the border of the region, where only horizontal and vertical “movements” are allowed. They are weighted with a distance of 1. If Metric = 'chess-board', the distance is calculated from the shortest path to the border, where horizontal, vertical, and diagonal “movements” are allowed. They are weighted with a distance of 1. If Metric = 'octagonal', a combination of these approaches is used, which leads to diagonal paths getting a higher weight. If Metric = 'chamfer-3-4', horizontal and vertical movements are weighted with a weight of 3, while diagonal movements are weighted with a weight of 4. To normalize the distances, the resulting distance image is divided by 3. Since this normalization step takes some time, and one usually is interested in the relative distances of the points, the normalization can be suppressed with Metric = 'chamfer-3-4-unnormalized'. Finally, if Metric = 'euclidean', the computed distance is approximately Euclidean.
10.6. TRANSFORMATION

- **Region** (input iconic) .......................... region(-array)  \(\rightarrow\) HRegionX / HObjectX
  Region for which the distance to the border is computed.

- **DistanceImage** (output iconic) ........................ image  \(\rightarrow\) HImageX / HUntypedObjectX (int4)
  Image containing the distance information.

- **Metric** (input control) .......................... string  \(\rightarrow\) String / VARIANT
  Type of metric to be used for the distance transformation.
  **Default Value:** 'city-block'
  **List of values:** Metric \(\in\) {'city-block', 'chessboard', 'octagonal', 'chamfer-3-4', 'chamfer-3-4-unnormalized', 'euclidean'}

- **Foreground** (input control) ........................ string  \(\rightarrow\) String / VARIANT
  Compute the distance for pixels inside (true) or outside (false) the input region.
  **Default Value:** 'true'
  **List of values:** Foreground \(\in\) {'true', 'false'}

- **Width** (input control) .......................... extent.x  \(\rightarrow\) long / VARIANT
  Width of the output image.
  **Default Value:** 640
  **Suggested values:** Width \(\in\) {160, 192, 320, 384, 640, 768}

- **Height** (input control) .......................... extent.y  \(\rightarrow\) long / VARIANT
  Height of the output image.
  **Default Value:** 480
  **Suggested values:** Height \(\in\) {120, 144, 240, 288, 480, 576}

---

/* Step towards extracting the medial axis of a shape: */
gen_rectangle1 (Rectangle1, 0, 0, 200, 400)
gen_rectangle1 (Rectangle2, 200, 0, 400, 200)
union2 (Rectangle1, Rectangle2, Shape)
distance_transform (Shape, DistanceImage, 'chessboard', 'true', 640, 480)

---

**Complexity**

The runtime complexity is \(O(\text{Width} \times \text{Height})\).

**Result**

**DistanceTransform** returns H_MSG_TRUE if all parameters are correct.

**Parallelization Information**

**DistanceTransform** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, DynThreshold, RegionGrowing

**Possible Successors**

Threshold

**Skeleton**

**See also**


**Module**

Region processing
Eliminate runs of a given length.

**EliminateRuns** eliminates all runs of the run length encoding of the input regions which are shorter than **ElimShorter** or longer as **ElimLonger**.

- **Region** (input iconic) - Region to be clipped.
  - **RegionClipped** (output iconic) - Clipped regions.
- **ElimShorter** (input control) - All runs which are shorter are eliminated.
  - **Default Value**: 3
  - **Suggested values**: **ElimShorter** ∈ \{2, 3, 4, 5, 6, 8, 10, 12, 15\}
  - **Typical range of values**: 1 ≤ **ElimShorter** ≤ 1
  - **Recommended Increment**: 1
- **ElimLonger** (input control) - All runs which are longer are eliminated.
  - **Default Value**: 1000
  - **Suggested values**: **ElimLonger** ∈ \{50, 100, 200, 500, 1000, 2000\}
  - **Typical range of values**: 1 ≤ **ElimLonger** ≤ 1
  - **Recommended Increment**: 10

**Result**

**EliminateRuns** returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via **SetSystem('noObjectResult',<Result>)** and the behavior in case of an empty input region via **SetSystem('emptyRegionResult',<Result>).** If necessary, an exception handling is raised.

**Parallelization Information**

**EliminateRuns** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

Threshold, Connection, RegionGrowing, Pouring

**Possible Successors**

Erosion1, Dilation1, DispRegion

**Alternatives**

ShapeTrans

**Module**

Region processing

---

**ExpandRegion** closes gaps between the input regions, which resulted from the suppression of small regions in a segmentation operator, for example, (mode 'image'), or to separate overlapping regions (mode 'region'). Both
10.6. TRANSFORMATION

uses result from the expansion of regions. The operator works by adding or removing a one pixel wide “strip” to a region.

The expansion takes place only in regions, which are designated as not “forbidden” (parameter ForbiddenArea). The number of iterations is determined by the parameter Iterations. By passing 'maximal', ExpandRegion iterates until convergence, i.e., until no more changes occur. By passing 0 for this parameter, all non-overlapping regions are returned. The two modes of operation ('image' and 'region') are different in the following ways:

'**image**’ The input regions are expanded iteratively until they touch another region or the image border. Because ExpandRegion processes all regions simultaneously, gaps between regions are distributed evenly to all regions. Overlapping regions are split by distributing the area of overlap evenly to both regions.

'**region**’ No expansion of the input regions is performed. Instead, only overlapping regions are split by distributing the area of overlap evenly to the respective regions. Because the intersection with the original region is computed after the shrinking operation gaps in the output regions may result, i.e., the segmentation is not complete. This can be prevented by calling ExpandRegion a second time with the complement of the original regions as “forbidden area.”

---

**Parameter**

- **Regions** (input iconic) .........................region(-array)  \(\mapsto\) HRegionX / IHObjectX
  Regions for which the gaps are to be closed, or which are to be separated.
- **ForbiddenArea** (input iconic) .......................region  \(\mapsto\) HRegionX / IHObjectX
  Regions in which no expansion takes place.
- **RegionExpanded** (output iconic) ......................region(-array)  \(\mapsto\) HRegionX / HUntypedObjectX
  Expanded or separated regions.
- **Iterations** (input control) .........................integer  \(\mapsto\) VARIANT (integer, string)
  Number of iterations.
  Default Value : 'maximal'
  Suggested values : Iterations ∈ \{'maximal', 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100, 200\}
  Typical range of values : 0 \(\leq\) Iterations \(\leq\) 0(lin)
  Minimum Increment : 1
  Recommended Increment : 1
- **Mode** (input control) ......................................string  \(\mapsto\) String / VARIANT
  Expansion mode.
  Default Value : 'image'
  List of values : Mode ∈ \{'image', 'region'\}

---

**Example**

```plaintext
read_image(Image,'fabrik')
threshold(Image,Light,100,255)
disp_region(Light,WindowHandle)
connection(Light,Seg)
expand_region(Seg,[],Exp1,'maximal','image')
set_colored(WindowHandle,12)
set_draw(WindowHandle,'margin')
disp_region(Exp1,WindowHandle)
```

---

**Result**

ExpandRegion always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

---

**Parallelization Information**

ExpandRegion is reentrant and processed without parallelization.

---

**Possible Predecessors**

Pouring, Threshold, DynThreshold, RegionGrowing

HALCON 6.1.4
Fill up holes in regions.

**FillUp** fills up holes in regions. The number of regions remains unchanged. The neighborhood type is set via `SetSystem('neighborhood',<4/8>)` (default: 8-neighborhood).

### Parameter

- **Region** (input iconic) region(-array) \( \rightarrow \) HRegionX / IHObjectX
  - Input regions containing holes.
- **RegionFillUp** (output iconic) region(-array) \( \rightarrow \) HRegionX / HUntypedObjectX
  - Regions without holes.

**Result**

FillUp returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`. If necessary, an exception handling is raised.

### Parallelization Information

FillUp is reentrant and automatically parallelized (on tuple level).

### Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring

### Possible Successors

SelectShape, DispRegion

### See also

Boundary

---

Fill up holes in regions having given shape features.

**FillUpShape** fills up those holes in the input region **Region** having given shape features. The parameter **Feature** determines the shape feature to be used, while **Min** and **Max** determine the range the shape feature has to lie in in order for the hole to be filled up.
10.6. TRANSFORMATION

Parameter

- **Region** (input iconic)  
  Input region(s).

- **RegionFillUp** (output iconic)  
  Output region(s) with filled holes.

- **Feature** (input control)  
  Shape feature used.
  - Default Value: 'area'
  - List of values: Feature ∈ \{area', 'compactness', 'convexity', 'anisometry', 'phi', 'ra', 'rb', 'inner_circle', 'outer_circle'\}

- **Min** (input control)  
  Minimum value for Feature.
  - Default Value: 1.0
  - Suggested values: Min ∈ \{0.0, 1.0, 10.0, 50.0, 100.0, 500.0, 1000.0, 10000.0\}

- **Max** (input control)  
  Maximum value for Feature.
  - Default Value: 100.0
  - Suggested values: Max ∈ \{10.0, 50.0, 100.0, 500.0, 1000.0, 10000.0, 100000.0\}

Result

FillUpShape returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>) and the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.

Parallelization Information

FillUpShape is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, Connection, RegionGrowing, Pouring

Possible Successors

SelectShape, DispRegion

SelectShape, Connection, AreaCenter

Alternatives

FillUp

Module

Region processing

```c
[out] HRegionX OutputRegion  HRegionX.HammingChangeRegion
([in] long Width, [in] long Height, [in] long Distance )

void HOperatorSetX.HammingChangeRegion ([in] IHObjectX InputRegion,
[out] HUntypedObjectX OutputRegion, [in] VARIANT Width, [in] VARIANT Height,
[in] VARIANT Distance )
```

Generate a region having a given Hamming distance.

HammingChangeRegion changes the region in the left upper part of the image given by Width and Height such that the resulting regions have a Hamming distance of Distance to the input regions. This is done by adding or removing Distance points from the input region.

Attention

If Width and Height are chosen too large the resulting region requires a lot of memory.

Parameter

- **InputRegion** (input iconic)  
  Region to be modified.
> **OutputRegion** (output iconic) ...........................................region(-array) \(\rightarrow\) HRegionX / HUntypedObjectX
Regions having the required Hamming distance.

> **Width** (input control) .............................................................. extent.x \(\rightarrow\) long / VARIANT
Width of the region to be changed.
**Default Value**: 100
**Suggested values**: Width \(\in\) \{64, 128, 256, 512\}
**Typical range of values**: \(1 \leq \text{Width} \leq 1\) (lin)
**Minimum Increment**: 1
**Recommended Increment**: 10
**Restriction**: (Width > 0)

> **Height** (input control) .............................................................. extent.y \(\rightarrow\) long / VARIANT
Height of the region to be changed.
**Default Value**: 100
**Suggested values**: Height \(\in\) \{64, 128, 256, 512\}
**Typical range of values**: \(1 \leq \text{Height} \leq 1\) (lin)
**Minimum Increment**: 1
**Recommended Increment**: 10
**Restriction**: (Height > 0)

> **Distance** (input control) ....................................................... integer \(\rightarrow\) long / VARIANT
Hamming distance between the old and new regions.
**Default Value**: 1000
**Suggested values**: Distance \(\in\) \{100, 500, 1000, 5000, 10000\}
**Typical range of values**: \(0 \leq \text{Distance} \leq 0\) (lin)
**Minimum Increment**: 1
**Recommended Increment**: 10
**Restriction**: (Distance \(\geq\) 0) \(\land\) (Distance < (Width \(\cdot\) Height))

---

### Complexity
Memory requirement of the generated region (worst case): \(O(2 \times \text{Width} \times \text{Height})\).

---

### Result
HammingChangeRegion returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via **SetSystem**(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

---

### Parallelization Information
HammingChangeRegion is reentrant and automatically parallelized (on tuple level).

---

### Possible Predecessors
Connection, Regiongrowing, Pouring, ClassNdimNorm

---

### Possible Successors
SelectShape

---

### Module
Region processing

```cpp
[out] HRegionX RegionInterjacent HRegionX.Interjacent
([in] String Mode )

void HOperatorSetX.Interjacent ([in] IHObjectX Region,
[out] HUntypedObjectX RegionInterjacent, [in] VARIANT Mode )
```

Partition the image plane using given regions.

**Interjacent** partitions the image plane using the regions given in **Region**. The result is a region containing the extracted separating lines. The following modes of operation can be used:

- **'medial_axis’** This mode is used for regions that do not touch or overlap. The operator will find separating lines between the regions which partition the background evenly between the input regions. This corresponds to the following calls:

```cpp
Partition the image plane using given regions.
```

---

HALCON/COM Reference Manual, 2005-2-1
complement('full',Region,Tmp) skeleton(Tmp,Result)

'border' If the input regions do not touch or overlap this mode is equivalent to Boundary(Region,Result). i.e., it replaces each region by its boundary. If regions are touching they are aggregated into one region. The corresponding output region then contains the boundary of the aggregated region, as well as the one pixel wide separating line between the original regions. This corresponds to the following calls:

boundary(Region,Tmp,’inner’) union1(Tmp1,Tmp2) skeleton(Tmp2,Result)

'mixed' In this mode the operator behaves like the mode 'medial_axis' for non-overlapping regions. If regions touch or overlap, again separating lines between the input regions are generated on output, but this time including the “touching line” between regions, i.e., touching regions are separated by a line in the output region. This corresponds to the following calls:

erosion1(Region,Mask,Tmp1,1) union1(Tmp1,Tmp2) complement(full,Tmp2,Tmp3) skeleton(Tmp3,Result)

where Mask denotes the following “cross mask”:

```
×
× × ×
×
```

Parameter

- **Region** (input iconic) ..............................................region(-array) ~ HRegionX / IHObjectX
  Regions for which the separating lines are to be determined.
- **RegionInterjacent** (output iconic) .........................region ~ HRegionX / HUntypedObjectX
  Output region containing the separating lines.
- **Mode** (input control) ..............................................string ~ String / VARIANT
  Mode of operation.
  Default Value: ’mixed’
  List of values: Mode ∈ {’medial_axis’, ’border’, ’mixed’}

Example

read_image(Image,’wald1_rot’) mean(Image,Mean,31,31) dyn_threshold(Mean,Seg,20) interjacent(Seg,Graph,’medial_axis’) disp_region(Graph,WindowHandle)

Result

Interjacent always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(’noObjectResult’,<Result>), the behavior in case of an empty input region via SetSystem(’emptyRegionResult’,<Result>), and the behavior in case of an empty result region via SetSystem(’storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.

Parallelization Information

Interjacent is reentrant and processed without parallelization.

Possible Predecessors

Threshold, Connection, Region-growing, Pouring

Possible Successors

SelectShape, DispRegion

See also

ExpandRegion, JunctionsSkeleton, Boundary

Module

Region processing
Find junctions and end points in a skeleton.

**JunctionsSkeleton** detects junctions and end points in a skeleton (see **Skeleton**). The junctions in the input region **Region** are output as a region in **JuncPoints**, while the end points are output as a region in **EndPoints**.

```plaintext
Parameter

- **Region** (input iconic) .................................. region(-array) `HRegionX / IHObjectX`
  Input skeletons.
- **EndPoints** (output iconic) ............................. region(-array) `HRegionX / HUntypedObjectX`
  Extracted end points.
  **Number of elements**: `EndPoints = Region`
- **JuncPoints** (output iconic) ............................. region(-array) `HRegionX / HUntypedObjectX`
  Extracted junctions.
  **Number of elements**: `JuncPoints = Region`
```

Example

```c
/* non-connected branches of a skeleton */
skeleton(Region,Skeleton)
junctions_skeleton(Skeleton,EPoints,JPoints)
difference(S,JPoints,Rows)
set_system(‘neighbourhood’,4)
connection(Rows,Parts).
```

**Complexity**

Let \( F \) be the area of the input region. Then the runtime complexity is \( O(F) \).

**Result**

**JunctionsSkeleton** always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)`, the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`, and the behavior in case of an empty result region via `SetSystem('storeEmptyRegion',<true/false>)`. If necessary, an exception handling is raised.

**Parallelization Information**

**JunctionsSkeleton** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

**Skeleton**

**Possible Successors**

**AreaCenter**, **Connection**, **GetRegionPoints**, **Difference**

**See also**

**Pruning**, **SplitSkeletonRegion**

**Module**

**Region processing**
10.6. TRANSFORMATION

HRegionX CurrMergedRegions HRegionX.MergeRegionsLineScan
([in] HRegionX PrevRegions, [out] HRegionX PrevMergedRegions,

void HOperatorSetX.MergeRegionsLineScan ([in] IHOBJECTX CurrRegions,
[in] IHOBJECTX PrevRegions, [out] HUntypedObjectX CurrMergedRegions,
[out] HUntypedObjectX PrevMergedRegions, [in] VARIANT ImageHeight,

Merge regions from line scan images.

The operator MergeRegionsLineScan connects adjacent regions, which were segmented from adjacent images with the height ImageHeight. This operator was especially designed to process regions that were extracted from images grabbed by a line scan camera. CurrRegions contains the regions from the current image and PrevRegions the regions from the previous one.

With the help of the parameter MergeBorder two cases can be distinguished: If the top (first) line of the current image touches the bottom (last) line of the previous image, MergeBorder must be set to 'top', otherwise set MergeBorder to 'bottom'.

If the operator MergeRegionsLineScan is used recursively, the parameter MaxImagesRegion determines the maximum number of images which are covered by a merged region. All older region parts are removed.

The operator MergeRegionsLineScan returns two region arrays. PrevMergedRegions contains all those regions from the previous input regions PrevRegions, which could not be merged with a current region. CurrMergedRegions collects all current regions together with the merged parts from the previous images. Merged regions will exceed the original image, because the previous regions are moved upward (MergeBorder='top') or downward (MergeBorder='bottom') according to the image height. For this the system parameter 'clip_region' (see also SetSystem) will internally be set to 'false'.

Parameter

▶ CurrRegions (input iconic) region(-array) HRegionX / IHOBJECTX
Current input regions.

▶ PrevRegions (input iconic) region(-array) HRegionX / IHOBJECTX
Merged regions from the previous iteration.

▶ CurrMergedRegions (output iconic) region(-array) HRegionX / HUntypedObjectX
Current regions, merged with old ones where applicable.

▶ PrevMergedRegions (output iconic) region(-array) HRegionX / HUntypedObjectX
Regions from the previous iteration which could not be merged with the current ones.

▶ ImageHeight (input control) integer long / VARIANT
Height of the line scan images.

Default Value : 512
List of values : ImageHeight ∈ {240, 480, 512}

▶ MergeBorder (input control) string String / VARIANT
Image line of the current image, which touches the previous image.

Default Value : 'top'
List of values : MergeBorder ∈ {'top', 'bottom'}

▶ MaxImagesRegion (input control) integer long / VARIANT
Maximum number of images for a single region.

Default Value : 3
Suggested values : MaxImagesRegion ∈ {1, 2, 3, 4, 5}

Result

The operator MergeRegionsLineScan returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

Parallelization Information

MergeRegionsLineScan is reentrant and processed without parallelization.

Module

Region processing
Partition a region horizontally into rectangles.

**PartitionDynamic** partitions the input region into rectangles having an average width of **Distance**. The region is not split exactly into \( n \) parts, but rather starting from the calculated splitting point positions having a minimum number of set pixels in the vertical directions are extracted. The search area is at most plus/minus the width of a rectangle. If **Percent** is passed as 100, this search area is used fully. If it is passed as 0, no search is done.

If the region is smaller than the given size its output remains unchanged. A partition is only done if the size of the region is at least 1.5 times the size of the rectangle given by the parameters.

---

**Parameter**

- **Region** (input iconic) .......................... region(-array)  \( \sim \) HRegionX / IHObjectX
  Region to be partitioned.

- **Partitioned** (output iconic) ......................... region  \( \sim \) HRegionX / HUntypedObjectX
  Partitioned region.

- **Distance** (input control) .......................... real  \( \sim \) double / VARIANT
  Width of the individual rectangles.

- **Percent** (input control) .......................... real  \( \sim \) double / VARIANT
  Maximum shift of the partition point.

  **Default Value**: 20

  **Suggested values**: \( \text{Percent} \in \{0, 10, 20, 30, 40, 50, 70, 90, 100\} \)

  **Typical range of values**: \( 0 \leq \text{Percent} \leq 0 \)

---

**Result**

**PartitionDynamic** returns **TRUE** if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)`, the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`, and the behavior in case of an empty result region via `SetSystem('storeEmptyRegion',<true/false>)`. If necessary, an exception handling is raised.

---

**Parallelization Information**

**PartitionDynamic** is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Threshold, Connection

**See also**

Intersection, SmallestRectangle1, ShapeTrans, ClipRegion

---

**Alternatives**

**PartitionRectangle**

---

**Module**

Region processing

---

**Partition a region into rectangles of equal size.**

**PartitionRectangle** partitions the input region into rectangles having an extent of **Width** times **Height**. The region is always split into rectangles of equal size. Therefore, **Width** and **Height** are adapted to the actual
size of the region. If the region is smaller than the given size its output remains unchanged. A partition is only done if the size of the region is at least 1.5 times the size of the rectangle given by the parameters.

Parameter

- **Region** (input iconic)  
  Region to be partitioned.
- **Partitioned** (output iconic)  
  Partitioned region.
- **Width** (input control)  
  Width of the individual rectangles.
- **Height** (input control)  
  Height of the individual rectangles.

Result

**PartitionRectangle** returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via **SetSystem('noObjectResult',<Result>)**, the behavior in case of an empty input region via **SetSystem('emptyRegionResult',<Result>)**, and the behavior in case of an empty result region via **SetSystem('storeEmptyRegion',<true/false>)**. If necessary, an exception handling is raised.

Parallelization Information

**PartitionRectangle** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Intersection, SmallestRectangle1, ShapeTrans, ClipRegion

Alternatives

PartitionDynamic

Module

Region processing

**Rank operator for regions.**

**RankRegion** calculates the binary rank operator. A filter mask of size **Height x Width** is used. In the process, for each point in the region the number of points of **Region** lying within the filter mask are counted. If this number is greater or equal to **Number**, the current point is added to the output region. If **Number = \( \text{Height} \times \text{Width} \over 2 \)**, is chosen, the median operator is obtained.

Attention

For **Height** and **Width** only odd values > 3 are valid. If invalid parameters are chosen they are converted automatically (without raising an exception handling) to the next larger even values.

Parameter

...
\textbf{Width} (input control) \hspace{1cm} \textbf{Height} (input control) \hspace{1cm} \textbf{Number} (input control)

- **Width**: Width of the filter mask.
  - Default Value: 15
  - Suggested values: Width $\in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}$
  - Typical range of values: $3 \leq \text{Width} \leq 3\text{ (lin)}$
  - Minimum Increment: 2
  - Recommended Increment: 2
  - Restriction: $(\text{Width} \land \text{odd})$

- **Height**: Height of the filter mask.
  - Default Value: 15
  - Suggested values: Height $\in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}$
  - Typical range of values: $3 \leq \text{Height} \leq 3\text{ (lin)}$
  - Minimum Increment: 2
  - Recommended Increment: 2
  - Restriction: $(\text{Height} \land \text{odd})$

- **Number**: Minimum number of points lying within the filter mask.
  - Default Value: 70
  - Suggested values: Number $\in \{5, 10, 20, 40, 60, 80, 90, 120, 150, 200\}$
  - Typical range of values: $1 \leq \text{Number} \leq 1\text{ (lin)}$
  - Minimum Increment: 1
  - Recommended Increment: 10
  - Restriction: $(\text{Number} > 0)$

---

**Example**

```c
read_image(Image, 'affe')
mean_image(Image, Mean, 5, 5)
dyn_threshold(Mean, Points, 25)
rank_region(Points, Textur, 15, 15, 30)
gen_circle(Mask, 10, 10, 3)
opening1(Textur, Mask, Seg).
```

---

**Complexity**

Let $F$ be the area of the input region. Then the runtime complexity is $O(F \ast 8)$. 

**Result**

\textbf{RankRegion} returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via \texttt{SetSystem('noObjectResult',<Result>)} and the behavior in case of an empty input region via \texttt{SetSystem('emptyRegionResult',<Result>). If necessary, an exception handling is raised.}

---

**Parallelization Information**

\textbf{RankRegion} is \textit{reentrant} and automatically \textit{parallelized} (on \textit{tuple level}).

---

**Possible Predecessors**

Threshold, Connection, RegionGrowing, Pouring, ClassNDimNorm

**Possible Successors**

SelectShape, DispRegion

---

**See also**

\textbf{RankImage}, \textbf{MeanImage}

---

**Alternatives**

ClosingRectangle1, ExpandRegion

---

**Module**

Region processing
\begin{verbatim}
[out] HRegionX OutputRegion HRegionX.RemoveNoiseRegion
([in] String Type)

void HOperatorSetX.RemoveNoiseRegion ([in] IHObjectX InputRegion,
[out] HUntypedObjectX OutputRegion, [in] VARIANT Type)
\end{verbatim}

Remove noise from a region.

`RemoveNoiseRegion` removes noise from a region. In mode ’n4’, a structuring element consisting of the four neighbors of a point is generated. A dilation with this structuring element is performed, and the intersection of the result and the input region is calculated. Thus all pixels having no 4-connected neighbor are removed.

\begin{itemize}
\item \textbf{InputRegion} (input iconic) \text{-array} \rightsquigarrow \text{HRegionX / IHObjectX}
\item Regions to be modified.
\item \textbf{OutputRegion} (output iconic) \text{-array} \rightsquigarrow \text{HRegionX / HUntypedObjectX}
\item Less noisy regions.
\item \textbf{Type} (input control) \text{string} \rightsquigarrow \text{String / VARIANT}
\item Mode of noise removal.
\end{itemize}

Default Value: ’n4’

List of values: Type \in \{’n4’, ’n8’, ’n48’\}

\begin{itemize}
\item \textbf{Complexity}
\end{itemize}

Let \( F \) be the area of the input region. Then the runtime complexity is \( O(\sqrt{F} \cdot 4) \).

Result

`RemoveNoiseRegion` returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem(‘noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information

`RemoveNoiseRegion` is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Connection, RegionGrowing, Pouring, ClassNdimNorm

Possible Successors

SelectShape

See also

Dilation1, Intersection, GenRegionPoints

Module

Region processing

\begin{verbatim}
[out] HRegionX RegionTrans HRegionX.ShapeTrans
([in] String Type)

void HOperatorSetX.ShapeTrans ([in] IHObjectX Region,
[out] HUntypedObjectX RegionTrans, [in] VARIANT Type)
\end{verbatim}

Transform the shape of a region.

`ShapeTrans` transforms the shape of the input regions depending on the parameter Type:

’convex’ Convex hull.
’ellipse’ Ellipse with the same moments and area as the input region.
’outer_circle’ Smallest enclosing circle.
’inner_circle’ Largest circle fitting into the region.
’rectangle1’ Smallest enclosing rectangle parallel to the coordinate axes.
’rectangle2’ Smallest enclosing rectangle.
’inner_center’ The point on the skeleton of the input region having the smallest distance to the center of gravity of the input region.
Parameter

- **Region** (input iconic)  region(-array)  \(\rightarrow\)  HRegionX / IHObjectX
  Regions to be transformed.

- **RegionTrans** (output iconic)  region(-array)  \(\rightarrow\)  HRegionX / HUntypedObjectX
  Transformed regions.

- **Type** (input control)  string  \(\rightarrow\)  String / VARIANT
  Type of transformation.
  **Default Value:** 'convex'
  **List of values:** Type \(\in\) \{'convex', 'ellipse', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'inner_center'\}

Let \(F\) be the area of the input region. Then the runtime complexity is \(O(F)\).

---

Result

**ShapeTrans** returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)`. If necessary, an exception handling is raised.

Parallelization Information

**ShapeTrans** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- Connection, RegionGrowing

Possible Successors

- DispRegion, RegionGrowingMean, AreaCenter
- Convexity, EllipticAxis, AreaCenter, SmallestRectangle1, SmallestRectangle2, SetShape, SelectShape, InnerCircle

Module

**Region processing**

```cpp
[out] HRegionX Skeleton HRegionX.Skeleton ( )
void HOperatorSetX.Skeleton ([in] IHObjectX Region,
[out] HUntypedObjectX Skeleton )
```

Compute the skeleton of a region.

**Skeleton** computes the skeleton of the input regions.

Parameter

- **Region** (input iconic)  region(-array)  \(\rightarrow\)  HRegionX / IHObjectX
  Region to be thinned.

- **Skeleton** (output iconic)  region(-array)  \(\rightarrow\)  HRegionX / HUntypedObjectX
  Resulting skeleton.

  **Number of elements:** \((Skeleton = Region)\)

Let \(F\) be the area of the enclosing rectangle of the input region. Then the runtime complexity is \(O(F)\) (per region).

Result

**Skeleton** returns TRUE if all parameters are correct. The behavior in case of empty input (no regions given) can be set via `SetSystem('noObjectResult',<Result>)` and the behavior in case of an empty input region via `SetSystem('emptyRegionResult',<Result>)`. If necessary, an exception handling is raised.

Parallelization Information

**Skeleton** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- SobelAmp, EdgesImage, BandpassImage, Threshold, HysteresisThreshold

Possible Successors

- JunctionsSkeleton, Pruning
10.6. TRANSFORMATION

See also
GraySkeleton, SobelAmp, EdgesImage, Roberts, BandpassImage, Threshold

MorphSkeleton, Thinning

References

Module
Region processing

```
HRegionX.SortRegion ([in] String SortMode, [in] String RowOrCol )
```

```
```

Sorting of regions with respect to their relative position.
The operator **SortRegion** sorts the regions with respect to their relative position. All sorting methods with the exception of ‘character’ use one point of the region. With the help of the parameter **RowOrCol** = ‘row’ these points will be sorted according to their row and then according to their column. By using ‘column’, the column value will be used first. The following values are available for the parameter **SortMode**:

- **'character'** The regions will be treated like characters in a row and will be sorted according to their order in the line: If two regions overlap horizontally, they will be sorted with respect to their column values, otherwise they will be sorted with regard to their row values.
- **'first_point'** The point with the lowest column value in the first row of the region.
- **'last_point'** The point with the highest column value in the last row of the region.
- **'upper_left'** Upper left corner of the surrounding rectangle.
- **'upper_right'** Upper right corner of the surrounding rectangle.
- **'lower_left'** Lower left corner of the surrounding rectangle.
- **'lower_right'** Lower right corner of the surrounding rectangle.

The parameter **Order** determines whether the sorting order is increasing or decreasing: using ‘true’ the order will be increasing, using ‘false’ the order will be decreasing.

\[\text{Attention}\]

**Parameter**

- **Regions** (input iconic) ............................... region \(\rightarrow\) HRegionX / IGameObject
  Regions to be sorted.
- **SortedRegions** (output iconic) .............................. region \(\rightarrow\) HRegionX / HObjectX
  Sorted regions.
- **SortMode** (input control) ................................. string \(\rightarrow\) String / VARIANT
  Kind of sorting.
  Default Value : ‘first_point’
  List of values : SortMode \(\in\) \{'character’, ‘first_point’, ‘last_point’, ‘upper_left’, ‘lower_left’, ‘upper_right’, ‘lower_right’\}
- **Order** (input control) ................................. string \(\rightarrow\) String / VARIANT
  Increasing or decreasing sorting order.
  Default Value : ‘true’
  List of values : Order \(\in\) \{'true’, ‘false’\}
- **RowOrCol** (input control) ................................. string \(\rightarrow\) String / VARIANT
  Sorting first with respect to row, then to column.
  Default Value : ‘row’
  List of values : RowOrCol \(\in\) \{'row’, ‘column’\}

HALCON 6.1.4
If the parameters are correct, the operator \texttt{SortRegion} returns the value TRUE. Otherwise an exception will be raised.

\textbf{Parallelization Information}

\texttt{SortRegion} is \emph{reentrant} and processed \emph{without} parallelization.

\textbf{Possible Successors}

\texttt{DoOcrMulti}, \texttt{DoOcrSingle}

\textbf{Module}

Region processing

\begin{verbatim}
[out] VARIANT BeginRow HRegionX.SplitSkeletonLines
([in] long MaxDistance, [out] VARIANT BeginCol, [out] VARIANT EndRow,
[out] VARIANT EndCol )

void HOperatorSetX.SplitSkeletonLines ([in] IObjectX SkeletonRegion,
[in] VARIANT MaxDistance, [out] VARIANT BeginRow, [out] VARIANT BeginCol,
[out] VARIANT EndRow, [out] VARIANT EndCol )
\end{verbatim}

\begin{description}
\item[	exttt{SplitSkeletonLines}] splits lines represented by one pixel wide, non-branching regions into shorter lines based on their curvature. A line is split if the maximum distance of a point on the line to the line segment connecting its end points is larger than \texttt{MaxDistance} (split \& merge algorithm). The start and end points of the approximating line segments are returned in \texttt{BeginRow}, \texttt{BeginCol}, \texttt{EndRow}, and \texttt{EndCol}.
\item[Attention] The input regions must represent non-branching lines, that is single branches of the skeleton.
\end{description}

\begin{description}
\item[SkeletonRegion] (input iconic) \texttt{region} \texttt{\sim} HRegionX / IObjectX
\item[MaxDistance] (input control) \texttt{integer} \texttt{\sim} long / VARIANT
\begin{description}
\item[Default Value] 3
\item[Suggested values]: \texttt{MaxDistance} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}
\item[Typical range of values]: \(1 \leq \texttt{MaxDistance} \leq 1\) (lin)
\item[Minimum Increment]: 1
\item[Recommended Increment]: 1
\end{description}
\item[BeginRow] (output control) \texttt{line.begin.y} \texttt{\sim} VARIANT(integer)
\item[BeginCol] (output control) \texttt{line.begin.x} \texttt{\sim} VARIANT(integer)
\item[EndRow] (output control) \texttt{line.end.y} \texttt{\sim} VARIANT(integer)
\item[EndCol] (output control) \texttt{line.end.x} \texttt{\sim} VARIANT(integer)
\end{description}

\begin{verbatim}
read_image(Image, ‘fabrik’)
edges_image (Image, ImaAmp, ImaDir, ’lanser2’, 0.5, ’nms’, 8, 16)
threshold (ImaAmp, RawEdges, 8, 255)
skeleton (RawEdges, Skeleton)
junctions_skeleton (Skeleton, EndPoints, JuncPoints)
\end{verbatim}

HALCON/COM Reference Manual, 2005-2-1
difference (Skeleton, JuncPoints, SkelWithoutJunc)
connection (SkelWithoutJunc, SingleBranches)
select_shape (SingleBranches, SelectedBranches, 'area', 'and', 16, 99999)
split_skeleton_lines (SelectedBranches, 3, BeginRow, BeginCol, EndRow, EndCol).

Result

SplitSkeletonLines always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

Parallelization Information

SplitSkeletonLines is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Connection, SelectShape, Skeleton, JunctionsSkeleton, Difference

Possible Successors

SelectLines, PartitionLines, DispLine

See also

SplitSkeletonRegion, DetectEdgeSegments

Module

Region processing

<table>
<thead>
<tr>
<th>SplitSkeletonRegion RegionLines HRegionX.SplitSkeletonRegion</th>
</tr>
</thead>
<tbody>
<tr>
<td>([in] long MaxDistance )</td>
</tr>
<tr>
<td>void HOperatorSetX.SplitSkeletonRegion</td>
</tr>
<tr>
<td>([in] IObjectX SkeletonRegion, [out] HUntypedObjectX RegionLines,</td>
</tr>
<tr>
<td>[in] VARIANT MaxDistance )</td>
</tr>
</tbody>
</table>

Split lines represented by one pixel wide, non-branching regions.

SplitSkeletonRegion splits lines represented by one pixel wide, non-branching regions into shorter lines based on their curvature. A line is split if the maximum distance of a point on the line to the line segment connecting its end points is larger than MaxDistance (split & merge algorithm). However, not the approximating lines are returned, but rather the original lines split into several output regions.

Attention

The input regions must represent non-branching lines, that is single branches of the skeleton.

Parameter

- **SkeletonRegion** (input iconic) region-array ⊳ HRegionX/IObjectX
  - Input lines (represented by 1 pixel wide, non-branching regions).
- **RegionLines** (output iconic) region ⊳ HRegionX/HUntypedObjectX
  - Split lines.
- **MaxDistance** (input control) integer ⊳ long/VARIANT
  - Maximum distance of the line points to the line segment connecting both end points.

Default Value : 3

Suggested values : MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Typical range of values : 1 ≤ MaxDistance ≤ 1(lin)

Minimum Increment : 1

Recommended Increment : 1

Example

read_image(Image, 'fabrik')
edges_image (Image, ImaAmp, ImaDir, 'lanser2', 0.5, 'nms', 8, 16)
threshold (ImaAmp, RawEdges, 8, 255)
skeleton (RawEdges, Skeleton)
junctions_skeleton (Skeleton, EndPoints, JuncPoints)
difference (Skeleton, JuncPoints, SkelWithoutJunc)
connection (SkelWithoutJunc, SingleBranches)
select_shape (SingleBranches, SelectedBranches, ‘area’, ‘and’, 16, 99999)
split_skeleton_region (SelectedBranches, Lines, 3)

\textbf{Result} \hspace{1cm} \text{\textit{SplitSkeletonRegion} always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via} \text{\textit{SetSystem(‘noObjectResult’,<Result>), the behavior in case of an empty input region via} \text{\textit{SetSystem(‘emptyRegionResult’,<Result>), and the behavior in case of an empty result region via} \text{\textit{SetSystem(‘storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.}}}

\textbf{Parallelization Information} \hspace{1cm} \text{\textit{SplitSkeletonRegion} is reentrant and automatically parallelized (on tuple level).}

\textbf{Possible Predecessors} \hspace{1cm} \text{\textit{Connection, SelectShape, Skeleton, JunctionsSkeleton, Difference}}

\textbf{Possible Successors} \hspace{1cm} \text{\textit{CountObj, SelectShape, SelectObj, AreaCenter, EllipticAxis, SmallestRectangle2, GetRegionPolygon, GetRegionContour}}

\textbf{See also} \hspace{1cm} \text{\textit{SplitSkeletonLines, GetRegionPolygon, GenPolygonsXld}}

\textbf{Module} \hspace{1cm} \text{Region processing}
Chapter 11

Segmentation

```plaintext
[out] HRegionX Regions HImageX.AutoThreshold ([in] VARIANT Sigma )
void HOperatorSetX.AutoThreshold ([in] IHObjectX Image,
[out] HUntypedObjectX Regions, [in] VARIANT Sigma )
```

Segment an image using thresholds determined from its histogram.

AutoThreshold segments a single-channel image using multiple thresholding. First the relative histogram of the gray values is determined. Then relevant minima are extracted from the histogram, which are used successively as parameters for a thresholding operation. The thresholds used are 0, 255, and all minima extracted from the histogram (after the histogram has been smoothed). For each gray value interval one region is generated. Thus, the number of regions is the number of minima + 1. The larger the value of Sigma is chosen, the less regions will be extracted. This operator is particularly suited if the regions to be extracted exhibit similar gray values (homogeneous regions).

Parameter

- **Image** (input iconic) ... image(-array) ... HImageX / IHObjectX ( byte )
  Image to be segmented.
- **Regions** (output iconic) ... region ... HRegionX / HUntypedObjectX
  Regions with gray values within the automatically determined intervals.
- **Sigma** (input control) ... number ... VARIANT ( integer, real )
  Sigma for the Gaussian smoothing of the histogram.

  Default Value : 2.0
  Suggested values : Sigma ∈ {0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typical range of values : 0.0 ≤ Sigma ≤ 0.0 (lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.3
  Restriction : (Sigma ≥ 0.0)

Parallelization Information

AutoThreshold is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

AnisotropeDiff, MedianImage, Illuminate

Possible Successors

Connection, SelectShape, SelectGray

Alternatives

GrayHisto, SmoothFunct1DGauss, Threshold

Module

Region processing

597
Segment a black-and-white image using an automatically determined threshold.

**BinThreshold** segments a single-channel gray value image using an automatically determined threshold. First the relative histogram of the gray values is determined. Then relevant minima are extracted from the histogram, which are used as parameters for a thresholding operation. In order to reduce the number of minima, the histogram is smoothed with a Gaussian, as in **AutoThreshold**. The mask size is enlarged until there is only one minimum in the smoothed histogram. The selected region contains the pixels with gray values from 0 to the minimum. This operator is particularly suited for the segmentation of dark characters on a light paper.

### Parameter

- **Image** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . image(-array) \( \sim \) HImageX / IHObjectX (byte, uint2)
  Image to be segmented.
- **Region** (output iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
  Dark regions of the image.

**Parallelization Information**

**BinThreshold** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- AnisotropeDiff, MedianImage, Illuminate

**Possible Successors**

- Connection, SelectShape, SelectGray

**Alternatives**

- AutoThreshold, CharThreshold, GrayHisto, SmoothFunct1DGauss, Threshold

**Module**

Region processing

Perform a threshold segmentation for extracting characters.

The main application of **CharThreshold** is to segment single-channel images of dark characters on bright paper. The operator works as follows: First, a histogram of the gray values in the image **Image** is computed for the points in the region **HistoRegion**. To eliminate noise, the histogram is smoothed with the given **Sigma** (Gaussian smoothing). In the histogram, the background (white paper) corresponds to a large peak at high gray values, while the characters form a small peak at low gray values. In contrast to the operator **BinThreshold**, which locates the minimum between the two peaks, here the threshold for the segmentation is determined in relation to the maximum of the histogram, i.e., the background, with the following condition:

\[
\text{histogram}[\text{threshold}] \times 100.0 < \text{histogram}[\text{maximum}] \times (100.0 - \text{Percent})
\]

For example, if you choose **Percent** = 95 the operator locates the gray value whose frequency is at most 5 percent of the maximum frequency. Because **CharThreshold** assumes that the characters are darker than the background, the threshold is searched for “to the left” of the maximum.

In comparison to **BinThreshold**, this operator should be used if there is no clear minimum between the histogram peaks corresponding to the characters and the background, respectively, or if there is no peak corresponding to the characters at all. This may happen, e.g., if the image contains only few characters or in the case of a non-uniform illumination.
Parameter

- **Image** (input iconic) .......................... image(-array)  Image to be segmented.
- **HistoRegion** (input iconic) .................. region  Region in which the histogram is computed.
- **Characters** (output iconic) ................. region(-array)  Dark regions (characters).
- **Sigma** (input control) ........................ number  Sigma for the Gaussian smoothing of the histogram.
  Default Value: 2.0
  Suggested values: Sigma ∈ {0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typical range of values: 0.0 ≤ Sigma ≤ 0.0(lin)
  Minimum Increment: 0.01
  Recommended Increment: 0.2
- **Percent** (input control) ...................... number  Percentage for the gray value difference.
  Default Value: 95
  Suggested values: Percent ∈ {90, 92, 95, 96, 97, 98, 99, 99.5, 100}
  Typical range of values: 0.0 ≤ Percent ≤ 0.0(lin)
  Minimum Increment: 0.1
  Recommended Increment: 0.5
- **Threshold** (output control) ................. integer(-array)  Calculated threshold.

Parallelization Information

CharThreshold is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

AnisotropeDiff, MedianImage, Illuminate

Possible Successors

Connection, SelectShape, SelectGray

Alternatives

BinThreshold, AutoThreshold, GrayHisto, SmoothFunct1DGauss, Threshold

Module

Region processing

```c
[out] HRegionX Selected
```

```c
void
```

Compare two images pixel by pixel.

`CheckDifference` selects from the input image `Image` those pixels \((g_o = g_{image})\), whose gray value difference to the corresponding pixels in `Pattern` is inside (outside) of the interval \([\text{DiffLowerBound}, \text{DiffLowerBound}]\). The pixels of `Pattern` are translated by \((\text{AddRow}, \text{AddCol})\) with respect to `Image`. Let \(g_p\) be the gray value from `Pattern` translated by \((\text{AddRow}, \text{AddCol})\) w.r.t. \(g_o\).

If the selected mode `Mode` is 'diff_inside', a pixel \(g_o\) is selected if

\[
g_o - g_p - \text{GrayOffset} > \text{DiffLowerBound}\quad \text{and}\quad g_o - g_p - \text{GrayOffset} < \text{DiffUpperBound}.
\]

If the mode is set to 'diff_outside', a pixel \(g_o\) is selected if

\[
g_o - g_p - \text{GrayOffset} < \text{DiffLowerBound}\quad \text{and}\quad g_o - g_p - \text{GrayOffset} > \text{DiffUpperBound}.
\]
\[ g_o - g_p - \text{GrayOffset} \leq \text{DiffLowerBound} \quad \text{or} \quad g_o - g_p - \text{GrayOffset} \geq \text{DiffUpperBound}. \]

This test is performed for all points of the domain (region) of \textit{Image}, intersected with the domain of the translated \textit{Pattern}. All points fulfilling the above condition are aggregated in the output region. The two images may be of different size. Typically, \textit{Pattern} is smaller than \textit{Image}.

\begin{itemize}
  \item \textbf{Image} (input iconic) ................................................. \textit{image}-array
    \textit{Image} to be examined.
  \item \textbf{Pattern} (input iconic) ................................................. \textit{image}-array
    Comparison image.
  \item \textbf{Selected} (output iconic) .............................................. \textit{region}-array
    Points in which the two images are similar/different.
  \item \textbf{Mode} (input control) .................................................. \textit{string}
    Mode: return similar or different pixels.
    \textbf{Default Value}: ‘diff\_outside’
    \textbf{Suggested values}: Mode \in \{‘diff\_inside’, ‘diff\_outside’\}
  \item \textbf{DiffLowerBound} (input control) ..................................... \textit{number}
    Lower bound of the tolerated gray value difference.
    \textbf{Default Value}: -5
    \textbf{Suggested values}: DiffLowerBound \in \{0, -1, -2, -3, -5, -7, -10, -12, -15, -17, -20, -25, -30\}
    \textbf{Typical range of values}: -255 \leq \text{DiffLowerBound} \leq -255(lin)
    \textbf{Minimum Increment}: 1
    \textbf{Recommended Increment}: 2
    \textbf{Restriction}: ((-255 \leq \text{DiffLowerBound}) \land (\text{DiffLowerBound} \leq 255))
  \item \textbf{DiffUpperBound} (input control) ..................................... \textit{number}
    Upper bound of the tolerated gray value difference.
    \textbf{Default Value}: 5
    \textbf{Suggested values}: DiffUpperBound \in \{0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30\}
    \textbf{Typical range of values}: -255 \leq \text{DiffUpperBound} \leq -255(lin)
    \textbf{Minimum Increment}: 1
    \textbf{Recommended Increment}: 2
    \textbf{Restriction}: ((-255 \leq \text{DiffUpperBound}) \land (\text{DiffUpperBound} \leq 255))
  \item \textbf{GrayOffset} (input control) .......................................... \textit{number}
    Offset gray value subtracted from Image.
    \textbf{Default Value}: 0
    \textbf{Suggested values}: GrayOffset \in \{-30, -25, -20, -17, -15, -12, -10, -7, -5, -3, -2, -1, 0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30\}
    \textbf{Typical range of values}: -255 \leq \text{GrayOffset} \leq -255(lin)
    \textbf{Minimum Increment}: 1
    \textbf{Recommended Increment}: 2
    \textbf{Restriction}: ((-255 \leq \text{GrayOffset}) \land (\text{GrayOffset} \leq 255))
  \item \textbf{AddRow} (input control) ............................................. \textit{point.y}
    Row coordinate, by which Pattern is translated.
    \textbf{Default Value}: 0
    \textbf{Suggested values}: AddRow \in \{-200, -100, -20, -10, 0, 10, 20, 100, 200\}
    \textbf{Typical range of values}: -32000 \leq \text{AddRow} \leq -32000(lin)
    \textbf{Minimum Increment}: 1
    \textbf{Recommended Increment}: 1
  \item \textbf{AddCol} (input control) .............................................. \textit{point.x}
    Column coordinate, by which Pattern is translated.
    \textbf{Default Value}: 0
    \textbf{Suggested values}: AddCol \in \{-200, -100, -20, -10, 0, 10, 20, 100, 200\}
    \textbf{Typical range of values}: -32000 \leq \text{AddCol} \leq -32000(lin)
    \textbf{Minimum Increment}: 1
    \textbf{Recommended Increment}: 1
\end{itemize}
Let $F$ be the number of valid pixels. Then the runtime complexity is $O(F)$.

CheckDifference returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

CheckDifference is reentrant and automatically parallelized (on tuple level).

Parallelization Information

Possible Successors

Connection, SelectShape, ReduceDomain, SelectGray, RankRegion, Dilation1, Opening

Alternatives

SubImage, DynThreshold

Module

Region processing

**Complexity**

Let $F$ be the number of valid pixels. Then the runtime complexity is $O(F)$.

**Result**

**CheckDifference**

**Parallelization Information**

**Possible Successors**

Connection, SelectShape, ReduceDomain, SelectGray, RankRegion, Dilation1, Opening

**Alternatives**

SubImage, DynThreshold

**Module**

Region processing

Segment an image using two-dimensional pixel classification.

Class2DimSup classifies the points in two-channel images using a two-dimensional feature space. For each point, two gray values (one from each image) are used as features. The feature space is represented by the input region. The classification is done as follows:

A point from the input region of an image is accepted if the point $(g_c, g_l)$, which is determined by the respective gray values, is contained in the region FeatureSpace. $g_l$ is here a gray value from the image ImageRow, while $g_c$ is the corresponding gray value from ImageCol.

Let $P$ be a point with the coordinates $P = (L,C)$, $g_l$ be the gray value at position $(L,C)$ in the image ImageRow, and $g_c$ be the gray value at position $(L,C)$ in the image ImageCol. Then the point $P$ is aggregated into the output region if

$$(g_c, g_l) \in \text{FeatureSpace}$$

$g_l$ is interpreted as row coordinate and $g_c$ as column coordinate.

For the generation of FeatureSpace, see Histo2Dim. The feature space can be modified by applying region transformation operators, such as RankRegion, Dilation1, ShapeTrans, EllipticAxis, etc., before calling Class2DimSup.

The parameters ImageCol and ImageRow must contain an equal number of images with the same respective size. The image points are taken from the intersection of the domains of both images (see ReduceDomain).

**Parameter**

- **ImageCol** (input iconic) . . . . . . . . . . . . . . . . . image(-array) $\sim$ HImageX / IObjectX (byte, direction, cyclic, int1)
  Input image (first channel).
- **ImageRow** (input iconic) . . . . . . . . . . . . . . . . . image(-array) $\sim$ HImageX / IObjectX (byte, direction, cyclic, int1)
  Input image (second channel).
- **FeatureSpace** (input iconic) . . . . . . . . . . . . . . region(-array) $\sim$ HRegionX / IObjectX
  Region defining the feature space.
- **RegionClass2Dim** (output iconic) . . . . . . . . . . . . region(-array) $\sim$ HRegionX / HUntypedObjectX
  Classified regions.

**Complexity**

Let $F$ be the area of the input region. Then the runtime complexity is $O(256^2 + F)$. 
Class2DimSup always returns TRUE. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

---

**Parallelization Information**

Class2DimSup is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

Histo2Dim, Threshold, DrawRegion, Dilation1, Opening, ShapeTrans

---

**Possible Successors**

Connection, SelectShape, SelectGray

---

**Alternatives**

ClassNdimNorm, ClassNdimBox, Threshold, Histo2Dim

---

**Module**

Region processing

---

**Segment two images by clustering.**

Class2DimUnsup performs a classification with two single-channel images. First a two-dimensional histogram of the two images is computed (Histo2Dim). In this histogram the first maximum is extracted; it serves as the first cluster center. The histogram is computed with the intersection of the domains of both images (see ReduceDomain). After this, all pixels in the images, which are at most Threshold pixels from the cluster center in the maximum norm, are determined. These pixels form one output region. Next, the pixels thus classified are deleted from the histogram so that they are not taken into account for the next class. In this modified histogram again the maximum is extracted; it again serves as a cluster center. The above steps are repeated NumClasses times; thus, NumClasses output regions result. Only pixels defined in both images are returned.

---

**Attention**

Both input images must have the same size.

---

**Parameter**

- **Image1** (input iconic) ............................ image \( \sim \) HImageX / IObjectX (byte)
  
  First input image.

- **Image2** (input iconic) ............................ image \( \sim \) HImageX / IObjectX (byte)
  
  Second input image.

- **Classes** (output iconic) .......................... region \( \sim \) HRegionX / HUntypedObjectX
  
  Segmentation result.

- **Threshold** (input control) ........................ integer \( \sim \) long / VARIANT
  
  Threshold (maximum distance to the cluster’s center).

  Default Value : 15
  
  Suggested values : Threshold \( \in \{0, 2, 5, 8, 12, 17, 20, 30, 50, 70\}\)

- **NumClasses** (input control) ........................ integer \( \sim \) long / VARIANT
  
  Number of classes (cluster centers).

  Default Value : 5
  
  Suggested values : NumClasses \( \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50\}\)

---

**Result**

Class2DimUnsup returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

---

**Parallelization Information**

Class2DimUnsup is reentrant and processed without parallelization.
Classify pixels using hyper-cuboids.

ClassNdimBox classifies the pixels of the multi-channel image given in MultiChannelImage. To do so, the classificator ClassifHandle created with CreateClassBox is used. The classificator can be trained using LearnNdimBox or as described with CreateClassBox. More information on the structure of the classificator can be found also under that operator.

MultiChannelImage is a multi channel image. Its pixel values are used for the classification.

Parameter

- **MultiChannelImage** (input iconic) …… multichannel-image-array \(\sim\) HImageX / IHOBJECTX (byte, direction, cyclic, int1, int2, int4, real)
  
  Multi-channel input image.

- **Regions** (output iconic) …………………………… region \(\sim\) HRegionX / HUntypedObjectX
  
  Segmentation result.

- **ClassifHandle** (input control) ……………………. class_box \(\sim\) HClassBoxX / VARIANT
  
  Classificator’s handle number.

Example

```c
read_image(Bild, 'meer')
disp_image(Image, WindowHandle)
set_color(WindowHandle, 'green')
fwrite_string('Draw the learning region')
fnew_line()
draw_region(Reg1, WindowHandle)
reduce_domain(Image, Reg1, Foreground)
set_color(WindowHandle, 'red')
fwrite_string('Draw Background')
fnew_line()
draw_region(Reg2, WindowHandle)
reduce_domain(Image, Reg2, Background)
fwrite_string('Start to learn')
fnew_line()
create_classif(ClassifHandle)
class_ndim_box(Foreground, Background, Image, ClassifHandle)
fwrite_string('start to classificate')
fnew_line()
class_ndim_box(Image, Res, ClassifHandle)
set_draw(WindowHandle, 'fill')
```
disp_region(Res,WindowHandle)
free_classif(ClassifHandle).

--- Complexity

Let $N$ be the number of hyper-cuboids and $F$ be the area of the input region. Then the runtime complexity is $O(N \times F)$.

--- Result

ClassNdimBox returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

--- Parallelization Information

ClassNdimBox is local and processed completely exclusively without parallelization.

--- Possible Predecessors

CreateClassBox, LearnClassBox, MedianImage, Compose2, Compose3, Compose4

See also

ClassNdimNorm, Class2DimSup

--- Alternatives

ClassNdimNorm, Class2DimSup

--- Module

Region processing

```
HRegionX Regions HImageX.ClassNdimNorm ([in] String Metric,
[in] String SingleMultiple, [in] VARIANT Radius,
[in] VARIANT Center)
```

```
void HOperatorSetX.ClassNdimNorm ([in] IObjectX MultiChannelImage,
[out] HUntypedObjectX Regions, [in] VARIANT Metric,
[in] VARIANT SingleMultiple, [in] VARIANT Radius,
[in] VARIANT Center)
```

Classify pixels using hyper-spheres or hyper-cubes.

ClassNdimNorm classifies the pixels of the multi-channel image given in MultiChannelImage. The result is returned in Regions as one region per classification object. The metric used ('euclid' or 'maximum') is determined by Metric. This parameter must be set to the same value used in LearnNdimNorm. The parameter is used to determine whether one region ('single') or multiple regions ('multiple') have to be generated for each cluster. Radius determines the radii or half edge lengths of the clusters, respectively. Center determines their centers.

--- Parameter

- **MultiChannelImage** (input iconic) 
  multichannel-image-array $\sim$ HImageX / IObjectX (byte) 
  Multi-channel input image.

- **Regions** (output iconic) 
  region $\sim$ HRegionX / HUntypedObjectX 
  Segmentation result.

- **Metric** (input control) 
  string $\sim$ String / VARIANT 
  Metric to be used. 
  **Default Value** : 'euclid' 
  **List of values**: Metric $\in \{\text{euclid'}, \text{maximum'}\}$

- **SingleMultiple** (input control) 
  string $\sim$ String / VARIANT 
  Return one region or one region for each cluster. 
  **Default Value** : 'single' 
  **List of values**: SingleMultiple $\in \{\text{single'}, \text{multiple'}\}$

- **Radius** (input control) 
  number(-array) $\sim$ VARIANT (integer, real) 
  Cluster radii or half edge lengths (returned by LearnNdimNorm).

- **Center** (input control) 
  number(-array) $\sim$ VARIANT (integer, real) 
  Coordinates of the cluster centers (returned by LearnNdimNorm).
Let \( N \) be the number of clusters and \( F \) be the area of the input region. Then the runtime complexity is \( O(N \times F) \).

**Result**

ClassNdNm returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no\_object\_result', 'empty\_region\_result', and 'store\_empty\_region' with SetSystem. If necessary, an exception is raised.

**Parallelization Information**

ClassNdNm is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

LearnNdNm, Compose2, Compose3, Compose4

Possible Successors

Connection, SelectShape, ReduceDomain, SelectGray

See also

DispCircle, DispRectangle1

Alternatives

ClassNdBox, Class2DimSup

Module

Region processing

```plaintext
[out] VARIANT BeginRow HImageX.DetectEdgeSegments ([in] long SobelSize,
[in] long MinAmplitude, [in] long MaxDistance, [in] long MinLength,
[out] VARIANT BeginCol, [out] VARIANT EndRow, [out] VARIANT EndCol )

void HOperatorSetX.DetectEdgeSegments ([in] IHObjectX Image,
[in] VARIANT SobelSize, [in] VARIANT MinAmplitude, [in] VARIANT MaxDistance,
[in] VARIANT MinLength, [out] VARIANT BeginRow, [out] VARIANT BeginCol,
[out] VARIANT EndRow, [out] VARIANT EndCol )
```

Detect straight edge segments.

DetectEdgeSegments detects straight edge segments in the gray image Image. The extracted edge segments are returned as line segments with start point (BeginRow, BeginCol) and end point (EndRow, EndCol). Edge detection is based on the Sobel filter, using 'sum\_abs' as parameter and SobelSize as the filter mask size (see SobelAmp). Only pixels with a filter response larger than MinAmplitude are used as candidates for edge points. These thresholded edge points are thinned and split into straight segments. Due to technical reasons, edge points in which several edges meet are lost. Therefore, DetectEdgeSegments usually does not return closed object contours. The parameter MaxDistance controls the maximum allowed distance of an edge point to its approximating line. For efficiency reasons, the sum of the absolute values of the coordinate differences is used instead of the Euclidean distance. MinLength controls the minimum length of the line segments. Lines shorter than MinLength are not returned.

**Parameter**

- **Image** (input iconic) …………………. (multichannel-)image(-array) \( \sim \) HImageX / IHObjectX (byte)
  - Input image.

- **SobelSize** (input control) ……………………………….. integer \( \sim \) long / VARIANT
  - Mask size of the Sobel operator.
    - **Default Value**: 5
    - **List of values**: SobelSize \( \in \{3, 5, 7, 9, 11, 13\} \)

- **MinAmplitude** (input control) ……………………………….. integer \( \sim \) long / VARIANT
  - Minimum edge strength.
    - **Default Value**: 32
    - **Suggested values**: MinAmplitude \( \in \{10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 80, 90, 100, 110\} \)
    - **Typical range of values**: 1 \( \leq \) MinAmplitude \( \leq \) 1
    - **Minimum Increment**: 1
    - **Recommended Increment**: 1
    - **Restriction**: (MinAmplitude \( \geq \) 0)
MaxDistance (input control) ................................................integer ~ long / VARIANT
Maximum distance of the approximating line to its original edge.

Default Value : 3
Suggested values : MaxDistance ∈ {2, 3, 4, 5, 6, 7, 8}
Typical range of values : 1 ≤ MaxDistance ≤ 1
Minimum Increment : 1
Recommended Increment : 1
Restriction : (MaxDistance ≥ 0)

MinLength (input control) ................................................integer ~ long / VARIANT
Minimum length of resulting line segments.

Default Value : 10
Suggested values : MinLength ∈ {3, 5, 7, 9, 11, 13, 16, 20}
Typical range of values : 1 ≤ MinLength ≤ 1
Minimum Increment : 1
Recommended Increment : 1
Restriction : (MinLength ≥ 0)

BeginRow (output control) ................................. line.begin.y ~ VARIANT ( integer )
Row coordinate of the line segments’ start points.

BeginCol (output control) ................................. line.begin.x ~ VARIANT ( integer )
Column coordinate of the line segments’ start points.

EndRow (output control) ................................. line.end.y ~ VARIANT ( integer )
Row coordinate of the line segments’ end points.

EndCol (output control) ................................. line.end.x ~ VARIANT ( integer )
Column coordinate of the line segments’ end points.

DetectEdgeSegments returns TRUE if all parameters are correct. If the input is empty the behaviour can be
set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception handling is raised.

Parallelization Information
DetectEdgeSegments is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors
SigmaImage, MedianImage

Possible Successors
SelectLines, PartitionLines, SelectLinesLongest, LinePosition, LineOrientation

Alternatives
SobelAmp, Threshold, Skeleton

Module

HRegionX RegionCrossings HImageX.DualThreshold ([in] long MinSize,
[in] double MinGray, [in] double Threshold )

void HOperatorSetX.DualThreshold ([in] IObjectX Image,
[out] HUntypedObjectX RegionCrossings, [in] VARIANT MinSize,
[in] VARIANT MinGray, [in] VARIANT Threshold )

Threshold operator for signed images.

DualThreshold segments the input image into a region with gray values ≥ Threshold (“positive” regions) and
a region with gray values ≤ -Threshold (“negative” regions). “Positive” or “negative” regions having a size of
less than MinSize are suppressed, as well as regions whose maximum gray value is less than MinGray in
absolute value.

The segmentation performed is not complete, i.e., the “positive” and “negative” regions together do not necessarily
cover the entire image: Areas with a gray value between −Threshold and Threshold, −MinGray and
MinGray respectively, are not taken into account.

DualThreshold is usually called after applying a Laplace operator ( Laplace, DerivateGauss or DiffOfGauss) or
the difference of two images (SubImage) to an image.

HALCON/COM Reference Manual, 2005-2-1
The zero crossings of a Laplace image correspond to edges in an image, and are the separating regions of the “positive” and “negative” regions in the Laplace image. They can be determined by calling `DualThreshold` with `Threshold = 1`. The parameter `MinGray` controls the noise invariance, while `MinSize` controls the resolution of the edge detection.

Using byte images only the positive part of the operator is applied. Therefore `DualThreshold` behaves like a standard threshold operator (`Threshold`) with successive `Connection` and `SelectGray`.

---

**Parameter**

- **Image** (input iconic) .............. image(-array) $\sim HImageX / IOBJECTX$ (byte, int2, int4, real )
  Input Laplace image.

- **RegionCrossings** (output iconic) .............. region $\sim HREGIONX / HUPTYPEDOBJECTX$
  “Positive” and “negative” regions.

- **MinSize** (input control) .................. integer $\sim long / VARIANT$
  Regions smaller than MinSize are suppressed.

  **Default Value:** 20
  **Suggested values:** MinSize $\in \{0, 10, 20, 50, 100, 200, 500, 1000\}$
  **Typical range of values:** $0 \leq \text{MinSize} \leq 0$ (lin)
  **Minimum Increment:** 1
  **Recommended Increment:** 10

- **MinGray** (input control) .................. real $\sim double / VARIANT$
  Regions whose maximum absolute gray value is smaller than MinGray are suppressed.

  **Default Value:** 5.0
  **Suggested values:** MinGray $\in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0\}$
  **Typical range of values:** $0.001 \leq \text{MinGray} \leq 0.001$ (lin)
  **Minimum Increment:** 1.0
  **Recommended Increment:** 10.0
  **Restriction:** $(\text{MinGray} > 0)$

- **Threshold** (input control) .................. real $\sim double / VARIANT$
  Regions which have a gray value larger than Threshold (or smaller than -Threshold) are suppressed.

  **Default Value:** 2.0
  **Suggested values:** Threshold $\in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0\}$
  **Typical range of values:** $0.001 \leq \text{Threshold} \leq 0.001$ (lin)
  **Minimum Increment:** 1.0
  **Recommended Increment:** 10.0
  **Restriction:** $(\text{Threshold} \geq 1) \land (\text{Threshold} \leq \text{MinGray})$

---

**Example**

```c
/* Edge detection with the Laplace operator (and edge thinning) */
diff_of_gauss(Image, Laplace, 2.0, 1.6)
/* find "positive" and "negative" regions: */
dual_threshold(Laplace, Region, 20, 2, 1)
/*The zero runnings are the complement to these image section: */
complement('full', Region, NullDurchgaenge).

/* Simulation of OpRef{dual_threshold} */
dual_threshold(Laplace, Result, MinS, MinG, Threshold):
  threshold(Laplace, Tmp1, Threshold, 999999)
  connection(Tmp1, Tmp2)
  select_shape(Tmp2, Tmp3, 'area', 'and', MinS, 999999)
  select_gray(Laplace, Tmp3, Tmp4, 'max', 'and', MinG, 999999)
  threshold(Laplace, Tmp5, -999999, -Threshold)
  connection(Tmp5, Tmp6)
  select_shape(Tmp6, Tmp7, 'area', 'and', MinS, 999999)
  select_gray(Laplace, Tmp7, Tmp8, 'min', 'and', -999999, -MinG)
  concat_obj(Tmp4, Tmp8, Result).
```

---

`DualThreshold` returns TRUE if all parameters are correct. The behavior with respect to the input images and
output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

--- Parallelization Information ---
DualThreshold is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessors ---
MinMaxGray, SobelAmp, GaussImage, ReduceDomain, DiffOfGauss, SubImage, DerivateGauss

--- Possible Successors ---
Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

--- See also ---
Laplace, DiffOfGauss, ExpandRegion

--- Alternatives ---
Threshold, Connection, SelectShape, SelectGray, DynThreshold, CheckDifference

--- Module ---
Region processing

```plaintext

```

Segment an image using a local threshold.

DynThreshold selects from the input image those regions in which the pixels fulfill a threshold condition. Let $g_o = g_{\text{OrigImage}}$ and $g_m = g_{\text{ThresholdImage}}$. Then the condition for LightDark = 'light' is:

$$g_o \geq g_m + \text{Offset}$$

For LightDark = 'dark' the condition is:

$$g_o \leq g_m - \text{Offset}$$

For LightDark = 'equal' it is:

$$g_m - \text{Offset} \leq g_o \leq g_m + \text{Offset}$$

Finally, for LightDark = 'not equal' it is:

$$g_m - \text{Offset} > g_o \lor g_o > g_m + \text{Offset}$$

This means that all points in OrigImage whose gray value is larger then or equal to the gray value in ThresholdImage plus an offset are aggregated into the resulting region.

Typically, the threshold images are smoothed versions of the original image (e.g., by applying MeanImage, GaussImage, etc.). Then the effect of DynThreshold is similar to applying Threshold to a highpass-filtered version of the original image (see HighpassImage).

With DynThreshold contours of an object can be extracted, where the objects' size (diameter) is determined by the mask size of the lowpass filter and the amplitude of the objects' edges:

The larger the mask size is chosen, the larger the found regions get. As a rule of thumb, the mask size should be about twice the diameter of the objects to be extracted. It is important not to set the parameter Offset to zero because in this case too many small regions will be found (noise). Values between 5 and 40 are a sensible choice. The larger Offset is chosen, the smaller the extracted regions get.

All points of the input image fulfilling the above condition are stored jointly in one region. If necessary, the connected components can be obtained by calling Connection.
Attention

If Offset is chosen from $-1$ to 1 usually a very noisy region is generated, requiring large storage. If Offset is chosen too large (> 60, say) it may happen that no points fulfill the threshold condition (i.e., an empty region is returned). If Offset is chosen too small (< -60, say) it may happen that all points fulfill the threshold condition (i.e., a full region is returned).

Parameter

- **OrigImage** (input iconic) .......... image(-array) $\rightsquigarrow$ HIImageX / IHOBJECTX (byte, int2, uint2, int4, real)
  Image to be segmented.

- **ThresholdImage** (input iconic) ... image(-array) $\rightsquigarrow$ HIImageX / IHOBJECTX (byte, int2, uint2, int4, real)
  Image containing the local thresholds.

- **RegionDynThresh** (output iconic) ................. region(-array) $\rightsquigarrow$ HRegionX / HUntypedObjectX
  Segmented regions.

- **Offset** (input control) .......................... number $\rightsquigarrow$ VARIANT (integer, real)
  Offset added to ThresholdImage.
  **Default Value**: 5.0
  **Suggested values**: Offset $\in \{1.0, 3.0, 5.0, 7.0, 10.0, 20.0, 30.0\}$
  **Typical range of values**: $-255.0 \leq \text{Offset} \leq -255.0$ (lin)
  **Minimum Increment**: 0.01
  **Recommended Increment**: 5
  **Restriction**: $((\text{Offset} \leq -255.0) \land (\text{Offset} < 255))$

- **LightDark** (input control) ........................................ string $\rightsquigarrow$ String / VARIANT
  Extract light, dark or similar areas?
  **Default Value**: 'light'
  **List of values**: LightDark $\in \{\text{dark'}, \text{light'}, \text{equal'}, \text{not_equal'}\}$

Example

/* Looking for regions with the diameter D */
mean_image(Image,Mean,D*2+1,D*2+1)
dyn_threshold(Image,Mean,Seg:5,'light')
connection(Seg,Regions).

Complexity

Let $F$ be the area of the input region. Then the runtime complexity is $O(F)$.

Result

DynThreshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

DynThreshold is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessors

MeanImage, SmoothImage, GaussImage

Possible Successors

Connection, SelectShape, ReduceDomain, SelectGray, RankRegion, Dilation1, Opening

See also

MeanImage, SmoothImage, GaussImage, Connection, RankRegion, Dilation1

Alternatives

CheckDifference, HighpassImage, SubImage, Threshold

Module

Region processing
CHAPTER 11. SEGMENTATION

```plaintext
HRegionX RegionExpand HRegionX.ExpandGray ([in] HImageX Image,
[in] HRegionX ForbiddenArea, [in] VARIANT Iterations, [in] String Mode,
[in] VARIANT Threshold )

void HOperatorSetX.ExpandGray ([in] IHObjectX Regions,
[in] IHObjectX Image, [in] IHObjectX ForbiddenArea,
[out] HUntypedObjectX RegionExpand, [in] VARIANT Iterations,
[in] VARIANT Mode, [in] VARIANT Threshold )
```

Fill gaps between regions (depending on gray value or color) or split overlapping regions.

**ExpandGray** closes gaps between the input regions, which resulted from the suppression of small regions in a segmentation operator, (mode ’image’), for example, or separates overlapping regions ’region’). Both uses result from the expansion of regions. The operator works by adding a one pixel wide “strip” to a region, in which the gray values or color are different from the gray values or color of neighboring pixels on the region’s border by at most **Threshold** (in each channel). For images of type ’cyclic’ (e.g., direction images), also points with a gray value difference of at least $255 - \text{Threshold}$ are added to the output region.

The expansion takes place only in regions, which are designated as not “forbidden” (parameter **ForbiddenArea**). The number of iterations is determined by the parameter **Iterations**. By passing ’maximal’, **ExpandGray** iterates until convergence, i.e., until no more changes occur. By passing 0 for this parameter, all non-overlapping regions are returned. The two modes of operation (’image’ and ’region’) are different in the following ways:

- **’image’** The input regions are expanded iteratively until they touch another region or the image border, or the expansion stops because of too high gray value differences. Because **ExpandGray** processes all regions simultaneously, gaps between regions are distributed evenly to all regions with a similar gray value. Overlapping regions are split by distributing the area of overlap evenly to both regions.
- **’region’** No expansion of the input regions is performed. Instead, only overlapping regions are split by distributing the area of overlap evenly to regions having a matching gray value or color.

**Attention** Because regions are only expanded into areas having a matching gray value or color, usually gaps will remain between the output regions, i.e., the segmentation is not complete.

**Parameter**

- **Regions** (input iconic) region(-array) ~ HRegionX / IHObjectX
  Regions for which the gaps are to be closed, or which are to be separated.
- **Image** (input iconic) image ~ HImageX / IHObjectX (byte, cyclic)
  Image (possibly multi-channel) for gray value or color comparison.
- **ForbiddenArea** (input iconic) region ~ HRegionX / IHObjectX
  Regions in which no expansion takes place.
- **RegionExpand** (output iconic) region(-array) ~ HRegionX / HUntypedObjectX
  Expanded or separated regions.
- **Iterations** (input control) string ~ VARIANT (integer, string)
  Number of iterations.
  Default Value : ’maximal’
  Suggested values : **Iterations** ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ’maximal’}
  Typical range of values : $1 \leq \text{Iterations} \leq 1$ (lin)
  Minimum Increment : 1
  Recommended Increment : 1
- **Mode** (input control) string ~ String / VARIANT
  Expansion mode.
  Default Value : ’image’
  List of values : **Mode** ∈ {’image’, ’region’}
```
Threshold (input control) .............................. integer(-array)  \sim  VARIANT ( integer )
Maximum difference between the gray value or color at the region’s border and a candidate for expansion.

Default Value : 32
Suggested values : Threshold ∈ \{ 5, 10, 15, 20, 25, 30, 40, 50 \}
Typical range of values : 1 ≤ Threshold ≤ 1(lin)
Minimum Increment : 1
Recommended Increment : 5

\textbf{Result}
ExpandGray always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem(‘noObjectResult’,<Result>), the behavior in case of an empty input region via SetSystem(‘emptyRegionResult’,<Result>), and the behavior in case of an empty result region via SetSystem(‘storeEmptyRegion’,<true/false>). If necessary, an exception handling is raised.

\textbf{Parallelization Information}
ExpandGray is reentrant and processed without parallelization.

\textbf{Possible Predecessors}
Connection, RegionGrowing, Pouring, ClassNdimNorm

\textbf{Possible Successors}
SelectShape

\textbf{See also}
ExpandGrayRef, ExpandRegion

\textbf{Module}
Region processing

\begin{verbatim}

\end{verbatim}

Fill gaps between regions (depending on gray value or color) or split overlapping regions.

ExpandGrayRef closes gaps between the input regions, which resulted from the suppression of small regions in a segmentation operator, (mode ‘image’), for example, or separates overlapping regions ‘region’). Both uses result from the expansion of regions. The operator works by adding a one pixel wide “strip” to a region, in which the gray values or color are different from a reference gray value or color by at most Threshold (in each channel). For images of type ‘cyclic’ (e.g., direction images), also points with a gray value difference of at least 255 – Threshold are added to the output region.

The expansion takes place only in regions, which are designated as not “forbidden” (parameter ForbiddenArea). The number of iterations is determined by the parameter Iterations. By passing ‘maximal’, ExpandGrayRef iterates until convergence, i.e., until no more changes occur. By passing 0 for this parameter, all non-overlapping regions are returned. The two modes of operation (’image’ and ’region’) are different in the following ways:

’image’ The input regions are expanded iteratively until they touch another region or the image border, or the expansion stops because of too high gray value differences. Because ExpandGrayRef processes all regions simultaneously, gaps between regions are distributed evenly to all regions with a similar gray value. Overlapping regions are split by distributing the area of overlap evenly to both regions.

’region’ No expansion of the input regions is performed. Instead, only overlapping regions are split by distributing the area of overlap evenly to regions having a matching gray value or color.

\textbf{Attention}
Because regions are only expanded into areas having a matching gray value or color, usually gaps will remain between the output regions, i.e., the segmentation is not complete.
**Parameter**

- **Regions** (input iconic) .......................... region(-array)  \~ HRegionX / IHOBJECTX
  Regions for which the gaps are to be closed, or which are to be separated.

- **Image** (input iconic) .......................... image  \~ HImageX / IHOBJECTX (byte, cyclic)
  Image (possibly multi-channel) for gray value or color comparison.

- **ForbiddenArea** (input iconic) .......................... region  \~ HRegionX / IHOBJECTX
  Regions in which no expansion takes place.

- **RegionExpand** (output iconic) .......................... region(-array)  \~ HRegionX / HUntypedObjectX
  Expanded or separated regions.

- **Iterations** (input control) .......................... string  \~ VARIANT (integer, string)
  Number of iterations.
  **Default Value:** 'maximal'
  **Suggested values:** Iterations \\in \{ 'maximal', 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 70, 100, 150, 200, 300, 500 \}
  **Typical range of values:** \( 1 \leq \text{Iterations} \leq 1(\text{lin}) \)
  **Minimum Increment:** 1
  **Recommended Increment:** 1

- **Mode** (input control) .......................... string  \~ String / VARIANT
  Expansion mode.
  **Default Value:** 'image'
  **List of values:** Mode \\in \{ 'image', 'region' \}

- **RefGray** (input control) .......................... integer(-array)  \~ VARIANT (integer)
  Reference gray value or color for comparison.
  **Default Value:** 128
  **Suggested values:** RefGray \\in \{ 1, 10, 20, 50, 100, 128, 200, 255 \}
  **Typical range of values:** \( 1 \leq \text{RefGray} \leq 1(\text{lin}) \)
  **Minimum Increment:** 1
  **Recommended Increment:** 10

- **Threshold** (input control) .......................... integer(-array)  \~ VARIANT (integer)
  Maximum difference between the reference gray value or color and a candidate for expansion.
  **Default Value:** 32
  **Suggested values:** Threshold \\in \{ 4, 10, 15, 20, 25, 30, 40 \}
  **Typical range of values:** \( 1 \leq \text{Threshold} \leq 1(\text{lin}) \)
  **Minimum Increment:** 1
  **Recommended Increment:** 5

**Result**

ExpandGrayRef always returns the value TRUE. The behavior in case of empty input (no regions given) can be set via SetSystem('noObjectResult',<Result>), the behavior in case of an empty input region via SetSystem('emptyRegionResult',<Result>), and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>). If necessary, an exception handling is raised.

**Parallellization Information**

ExpandGrayRef is reentrant and processed without parallelization.

**Possible Predecessors**

Connection, RegionGrowing, Pouring, ClassNdImNorm

**Possible Successors**

SelectShape

**See also**

ExpandGray, ExpandRegion

**Module**

Region processing
Expand a region starting at a given line.

**ExpandLine** generates a region by expansion, starting at a given line (row or column). The expansion is terminated when the current gray value differs by more than **Threshold** from the mean gray value along the line (**ExpandType** = 'mean') or from the previously added gray value (**ExpandType** = 'gradient').

### Parameter

- **Image** (input iconic) .......................... image(-array) ➔ **HImageX / IObjectX** (byte)
  Image to be segmented.
- **RegionExpand** (output iconic) ............... region(-array) ➔ **HRegionX / HUntypedObjectX**
  Extracted segments.
- **Index** (input control) .......................... integer ➔ **long / VARIANT**
  Row or column index.
  Default Value: 256
  Suggested values: **Index** ∈ {16, 64, 128, 200, 256, 300, 400, 511}
  Restriction: (Index ≥ 0)
- **ExpandType** (input control) .................. string ➔ **String / VARIANT**
  Stopping criterion.
  Default Value: 'gradient'
  List of values: **ExpandType** ∈ {'gradient', 'mean'}
- **RowColumn** (input control) .................. string ➔ **String / VARIANT**
  Segmentation mode (row or column).
  Default Value: 'row'
  List of values: **RowColumn** ∈ {'row', 'column'}
- **Threshold** (input control) .................. number ➔ **VARIANT** (integer, real)
  Threshold for the expansion.
  Default Value: 3.0
  Suggested values: **Threshold** ∈ {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 13.0, 17.0, 20.0, 30.0}
  Typical range of values: 1.0 ≤ **Threshold** ≤ 1.0 (lin)
  Minimum Increment: 1.0
  Recommended Increment: 1.0
  Restriction: ((**Threshold** ≥ 0.0) ∧ (**Threshold** ≤ 255.0))

**Parallelization Information**

**ExpandLine** is **reentrant** and automatically **parallelized** (on tuple level).

**Possible Predecessors**

- GaussImage, SmoothImage, AnisotropeDiff, MedianImage, AffineTransImage,
- RotateImage

**Possible Successors**

- Intersection, Opening, Closing

**Alternatives**

- RegionGrowingMean, ExpandGray, ExpandGrayRef

**Module**

Region processing
**Fast selection of gray values within a given gray interval.**

FastThreshold selects the pixels from the input image whose gray values \( g \) fulfill the following condition:

\[
\text{MinGray} \leq g \leq \text{MaxGray}.
\]

To reduce processing time, the selection is done in two steps: At first all pixels along rows with distances MinHeight are processed. In the next step the neighborhood (size MinHeight \( \times \) MinHeight) of all previously selected points are processed.

---

**Parameter**

- **Image** (input iconic) .......................... image(-array) \( \sim \) HImageX / IHObjectX (byte, uint2, direction, cyclic)
  Image to be thresholded.

- **Region** (output iconic) ................................. region(-array) \( \sim \) HRegionX / HUntypedObjectX
  Regions with gray values lying in the specified interval.

- **MinGray** (input control) ................................ number \( \sim \) VARIANT (integer, real)
  Lower threshold for the gray values.

  **Default Value:** 128
  **Suggested values:** \( \text{MinGray} \in \{0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0\} \)
  **Typical range of values:** \( 0.0 \leq \text{MinGray} \leq 0.0(\text{lin}) \)
  **Minimum Increment:** 1
  **Recommended Increment:** 5.0

- **MaxGray** (input control) ................................. number \( \sim \) VARIANT (integer, real)
  Upper threshold for the gray values.

  **Default Value:** 255.0
  **Suggested values:** \( \text{MaxGray} \in \{0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0\} \)
  **Typical range of values:** \( 0.0 \leq \text{MaxGray} \leq 0.0(\text{lin}) \)
  **Minimum Increment:** 1
  **Recommended Increment:** 5.0

- **MinHeight** (input control) ............................... number \( \sim \) long / VARIANT
  Minimum height of objects to be extracted.

  **Default Value:** 20
  **Suggested values:** \( \text{MinHeight} \in \{5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 100\} \)
  **Typical range of values:** \( 2 \leq \text{MinHeight} \leq 2(\text{lin}) \)
  **Minimum Increment:** 1
  **Recommended Increment:** 2

---

**Complexity**

Let \( A \) be the area of the output region and \( \text{height} \) the height of Image. Then the runtime complexity is \( O(A + \text{height} / \text{MinHeight}) \).

---

**Result**

FastThreshold returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

---

**Parallelization Information**

FastThreshold is reentrant and automatically parallelized (on tuple level).
Determine gray value thresholds from a histogram.

HistoToThresh determines gray value thresholds from a histogram for a segmentation of an image using Threshold. The thresholds returned are 0, 255, and all minima extracted from the histogram. Before the thresholds are determined the histogram is smoothed with a Gaussian.

Parameter

- **Histogramm** (input control) 
  Gray value histogram.

- **Sigma** (input control) 
  Sigma for the Gaussian smoothing of the histogram.
  - Default Value: 2.0
  - Suggested values: Sigma ∈ {0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  - Typical range of values: 0.1 ≤ Sigma ≤ 0.1
  - Minimum Increment: 0.01
  - Recommended Increment: 0.2

- **MinThresh** (output control) 
  Minimum thresholds.

- **MaxThresh** (output control) 
  Maximum thresholds.

Parallelization Information

HistoToThresh is reentrant and processed without parallelization.

Possible Predecessors

- Threshold

Possible Successors

- AutoThreshold

Alternatives

Module

Region processing

Perform a hysteresis threshold operation on an image.

HysteresisThreshold performs a hysteresis threshold operation (due to Canny) on an image. All points in the input image Image having a gray value larger than or equal to High are immediately accepted ("secure" points). Conversely, all points with gray values less than Low are immediately rejected. "Potential" points with gray values between both thresholds are accepted if they are connected to "secure" points by a path of "potential"
points having a length of at most \( \text{MaxLength} \) points. This means that “secure” points influence their surroundings (hysteresis). The gray values of the input images remain unchanged. Only the regions of the image may get smaller.

---

**Parameter**

- **Image** (input iconic) ................................. image(-array)  \( \sim \)  \( \text{HImageX/IOObjectX} \)  \( \text{byte, uint2} \)  
  Image to be segmented.

- **RegionHysteresis** (output iconic) .................. region(-array)  \( \sim \)  \( \text{HRegionX/HUntypedObjectX} \)  
  Segmentation result.

- **Low** (input control) ................................. integer  \( \sim \)  \( \text{long/VARIANT} \)  
  Lower threshold for the gray values.
  
  **Default Value:** 30
  
  **Suggested values:** \( \text{Low} \in \{5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100\} \)
  
  **Typical range of values:** \( 0 \leq \text{Low} \leq 0\text{(lin)} \)
  
  **Minimum Increment:** 1
  
  **Recommended Increment:** 5
  
  **Restriction:** \( (0 < \text{Low}) \land (\text{Low} < 255) \)

- **High** (input control) ................................. integer  \( \sim \)  \( \text{long/VARIANT} \)  
  Upper threshold for the gray values.
  
  **Default Value:** 60
  
  **Suggested values:** \( \text{High} \in \{5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130\} \)
  
  **Typical range of values:** \( 0 \leq \text{High} \leq 0\text{(lin)} \)
  
  **Minimum Increment:** 1
  
  **Recommended Increment:** 5
  
  **Restriction:** \( ((0 < \text{High}) \land (\text{High} < 255)) \land (\text{High} > \text{Low}) \)

- **MaxLength** (input control) ........................... integer  \( \sim \)  \( \text{long/VARIANT} \)  
  Maximum length of a path of “potential” points to reach a “secure” point.
  
  **Default Value:** 10
  
  **Suggested values:** \( \text{MaxLength} \in \{1, 2, 3, 5, 7, 10, 12, 14, 17, 20, 25, 30, 35, 40, 50\} \)
  
  **Typical range of values:** \( 1 \leq \text{MaxLength} \leq 1\text{(lin)} \)
  
  **Minimum Increment:** 1
  
  **Recommended Increment:** 5
  
  **Restriction:** \( (\text{MaxLength} > 1) \)

---

**Result**

\( \text{HysteresisThreshold} \) returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags ‘no\_object\_result’, ‘empty\_region\_result’, and ‘store\_empty\_region’ with \( \text{SetSystem} \). If necessary, an exception is raised.

---

**Parallelization Information**

\( \text{HysteresisThreshold} \) is reentrant and automatically parallelized (on tuple level).

---

**See also**

- EdgesImage, SobelAmp, BackgroundSeg
- Alternatives
- DynThreshold, Threshold, Class2DimSup

---

**References**


---

**Module**

Region processing
Train the current classifier using a multi-channel image.

`LearnNdimBox` trains the classifier `ClassifHandle` with the gray values of `MultiChannelImage` using the points in `Foreground` as training sample. The points in `Background` are to be rejected by the classifier. The classifier trained thus can be used in `ClassNdimBox` to segment multi-channel images.

Each pixel is trained once during the training process. For points in `Foreground` the class “0” is used, while for `Background` “1” is used. Pixels are trained by alternating points from `Foreground` with points from `Background`. If one region is smaller than the other, pixels are taken cyclically from the smaller region until the larger region is exhausted. `LearnNdimBox` later accepts only points which can be classified into class “0”.

From a user’s point of view the key difference between `LearnNdimNorm` and `LearnNdimBox` is that in the latter case the rejection class affects the classification process itself. Here, a hyper plane is generated, that separates `Foreground` and `Background` classes, so that no points in feature space are classified incorrectly. As for `LearnNdimNorm`, however, an overlap between `Foreground` and `Background` class is allowed. This has its effect on the return value `Quality`. The larger the overlap, the smaller this value.

### Attention
All channels must be of the same type and have the same size.

### Parameter
- `Foreground` (input iconic) region(-array) \( \sim \) `HRegionX / IObjectX`
  Foreground pixels to be trained.
- `Background` (input iconic) region(-array) \( \sim \) `HRegionX / IObjectX`
  Background pixels to be trained (rejection class).
- `MultiChannelImage` (input iconic) multichannel-image-array \( \sim \) `HImageX / IObjectX`
  Multi-channel training image.
- `ClassifHandle` (input control) class_box \( \sim \) `HClassBoxX / VARIANT`
  Classifier’s handle number.

### Complexity
Let \( N \) be the number of generated hyper-cuboids and \( F \) be the area of the input region. Then the runtime complexity is \( O(N \cdot F) \).

### Result
`ClassNdimBox` returns `TRUE` if all parameters are correct and there is an active classifier. The behavior with respect to the input images can be determined by setting the values of the flags `no_object_result` and `empty_region_result` with `SetSystem`. If necessary, an exception is raised.

### Parallelization Information
`LearnNdimBox` is local and processed completely exclusively without parallelization.

### Possible Predecessors
- `CreateClassBox`, `DrawRegion`

### Possible Successors
- `ClassNdimBox`, `DescriptClassBox`

### Alternatives
- `LearnClassBox`

### Module
Region processing
Construct clusters for \textit{ClassNdimNorm}.

\textbf{LearnNdimNorm} generates classification clusters from the region \textit{Foreground} and the corresponding gray values in the multi-channel image \textit{Image}, which can be used in \textit{ClassNdimNorm}. \textit{Background} determines a class of pixels not to be found in \textit{ClassNdimNorm}. This parameter may be empty (empty tuple).

The parameter \textit{Distance} determines the maximum distance \textit{Radius} of the clusters. It describes the minimum distance between two cluster centers. If the parameter \textit{Distance} is small the (small) hyper-cubes or hyper-spheres can approximate the feature space well. Simultaneously the runtime during classification increases.

The ratio of the number of pixels in a cluster to the total number of pixels (in percent) must be larger than the value of \textit{MinNumberPercent}. otherwise the cluster is not returned. \textit{MinNumberPercent} serves to eliminate outliers in the training set. If it is chosen too large many clusters are suppressed.

Two different clustering procedures can be selected: The minimum distance algorithm (n-dimensional hyper-spheres) and the maximum algorithm (n-dimensional hyper-cubes) for describing the pixels of the image to classify in the n-dimensional histogram (parameter \textit{Metric}). The Euclidian metric usually yields the better results, but takes longer to compute. The parameter \textit{Quality} returns the quality of the clustering. It is a measure of overlap between the rejection class and the classifier classes. Values larger than 0 denote the corresponding ratio of overlap. If no rejection region is given, its value is set to 1. The regions in \textit{Background} do not influence on the clustering. They are merely used to check the results that can be expected.

From a user’s point of view the key difference between \textit{LearnNdimNorm} and \textit{LearnNdimBox} is that in the latter case the rejection class affects the classification process itself. Here, a hyper plane is generated, that separates \textit{Foreground} and \textit{Background} classes, so that no points in feature space are classified incorrectly. As for \textit{LearnNdimNorm}, however, an overlap between \textit{Foreground} and \textit{Background} class is allowed. This has its effect on the return value \textit{Quality}. The larger the overlap, the smaller this value.

\begin{itemize}
  \item \textbf{Foreground} (input iconic) \hspace{1cm} region(-array) \sim HRegionX / IObjectX
    Foreground pixels to be trained.
  \item \textbf{Background} (input iconic) \hspace{1cm} region(-array) \sim HRegionX / IObjectX
    Background pixels to be trained (rejection class).
  \item \textbf{Image} (input iconic) \hspace{1cm} image(-array) \sim HImageX / IObjectX (byte)
    Multi-channel training image.
  \item \textbf{Metric} (input control) \hspace{1cm} string \sim String / VARIANT
    Metric to be used.
    Default Value: ‘euclid’
    List of values: \text{Metric} \in \{‘euclid’, ‘maximum’\}
  \item \textbf{Distance} (input control) \hspace{1cm} number \sim VARIANT (integer, real)
    Maximum cluster radius.
    Default Value: 10.0
    Suggested values: \text{Distance} \in \{1.0, 2.0, 3.0, 4.0, 6.0, 8.0, 10.0, 13.0, 17.0, 24.0, 30.0, 40.0\}
    Typical range of values: 0.0 \leq \text{Distance} \leq 0.0 (in)
    Minimum Increment: 0.01
    Recommended Increment: 1.0
    Restriction: (\text{Radius} > 0.0)
  \item \textbf{MinNumberPercent} (input control) \hspace{1cm} number \sim VARIANT (integer, real)
    The ratio of the number of pixels in a cluster to the total number of pixels (in percent) must be larger than
\end{itemize}
MinNumberPercent (otherwise the cluster is not output).

**Default Value:** 0.01

**Suggested values:** MinNumberPercent ∈ {0.001, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0}

**Typical range of values:** 0.0 ≤ MinNumberPercent ≤ 0.0 (lin)

**Minimum Increment:** 0.01

**Recommended Increment:** 0.1

**Restriction:** ((0 ≤ MinNumberPercent) ∧ (MinNumberPercent ≤ 100))

- **Radius** (output control) ... real ~ VARIANT (real)
  Cluster radii of half edge lengths.

- **Center** (output control) ... real ~ VARIANT (real)
  Coordinates of all cluster centers.

- **Quality** (output control) ... real ~ double / VARIANT
  Overlap of the rejection class with the classified objects (1: no overlapping).

**Restriction:** ((0 ≤ Quality) ∧ (Quality ≤ 1))

---

**LearnNdimNorm** returns TRUE if all parameters are correct. The behavior with respect to the input images can be determined by setting the values of the flags 'no.object_result' and 'empty.region_result' with **SetSystem**. If necessary, an exception is raised.

---

**Parallelization Information**

**LearnNdimNorm** is **local** and processed **completely exclusively** without parallelization.

---

**Possible Predecessors**

MinMaxGray, SobelAmp, GaussImage, ReduceDomain, DiffOfGauss

---

**Possible Successors**

ClassNdimNorm, Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

---

See also **ClassNdimNorm**, **ClassNdimBox**, **Histo2Dim**

---

**References**

P. Haberacker, "’Digitale Bildverarbeitung’"; Hanser-Studienbücher, München, Wien, 1987

---

**Region processing**

```c
[out] HRegionX LocalMaxima HIMageX.LocalMax ( )
void HOperatorSetX.LocalMax ([in] IHObjectX Image,
[out] HUntypedObjectX LocalMaxima )
```

**Detect all local maxima in an image.**

**LocalMax** extracts all points in an image having a gray value larger than the gray value of all its neighbors. The neighborhood used can be set by **SetSystem**(‘neighborhood’, <4/8>).

---

**Parameter**

- **Image** (input iconic) ... image(-array) ~ HIMageX / IHObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real)
  Image to be processed.

- **LocalMaxima** (output iconic) ... region(-array) ~ HRegionX / HUntypedObjectX
  Extracted local maxima as regions.

**Number of elements:** (LocalMaxima = Image)

---

**Parallelization Information**

**LocalMax** is **reentrant** and automatically **parallelized** (on tuple level).

---

**Possible Predecessors**

GaussImage, SmoothImage

---

**Possible Successors**

GetRegionPoints, Connection
See also Monotony, TopographicSketch, CornerResponse, TextureLaws

Alternatives
GraySkeleton, NonmaxSuppressionAmp, Plateaus, PlateausCenter

Module

Region processing

```c
[out] HImageX ImageResult HImageX.NonmaxSuppressionAmp
([in] String Mode )

void HOperatorSetX.NonmaxSuppressionAmp ([in] IHObjectX ImgAmp,
```

Suppress non-maximum points on an edge.

NonmaxSuppressionAmp suppresses in the regions of the image ImgAmp all points whose gray values are not local (directed) maxima. In contrast to NonmaxSuppressionDir, a direction image is not needed. Two modes of operation can be selected:

'hvms' A point is labeled as a local maximum if its gray value is larger than or equal to the gray values within a search space of ± 5 pixels, either horizontally or vertically. Non-maximum points are removed from the region, gray values remain unchanged.

'loc_max' A point is labeled as a local maximum if its gray value is larger than or equal to the gray values of its eight neighbors.

Parameter

- **ImgAmp** (input iconic) image(-array) \(\sim\) HImageX / IHObjectX (byte, uint2)
  Amplitude (gradient magnitude) image.
- **ImageResult** (output iconic) image(-array) \(\sim\) HImageX / HUntypedObjectX (byte, uint2)
  Image with thinned edge regions.
- **Mode** (input control) string \(\sim\) String / VARIANT
  Select horizontal/vertical or undirected NMS.
  Default Value: 'hvms'
  List of values: Mode \(\in\) {'hvms', 'loc_max'}

Result

NonmaxSuppressionAmp returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

NonmaxSuppressionAmp is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors

SobelAmp
Threshold, HysteresisThreshold

Possible Successors

Skeleton

Alternatives
GraySkeleton, LocalMax, GrayDilationRect

References

Suppress non-maximum points on an edge.

NonmaxSuppressionDir suppresses in the regions of the image ImgAmp all points whose gray values are not local (directed) maxima. ImgDir is a direction image giving the direction perpendicular for the local maximum (Unit: 2 degrees, i.e., 50 degrees are coded as 25 in the image). Such images are returned, for example, by EdgesImage. Two modes of operation can be selected:

'nms' Each point in the image is tested whether its gray value is a local maximum perpendicular to its direction. In this mode only the two neighbors closest to the given direction are examined. If one of the two gray values is greater than the gray of the point to be tested, it is suppressed (i.e., removed from the input region. The corresponding gray value remains unchanged).

'inms' Like 'nms'. However, the two gray values for the test are obtained by interpolation from four adjacent points.

Parameter

- **ImgAmp** (input iconic) image-array -> HImageX / IHObjectX (byte, uint2) Amplitude (gradient magnitude) image.
- **ImgDir** (input iconic) image-array -> HImageX / IHObjectX (direction) Direction image.
- **ImageResult** (output iconic) image-array -> HImageX / HUntypedObjectX (byte, uint2) Image with thinned edge regions.
- **Mode** (input control) string Select non-maximum-suppression or interpolating NMS.

Default Value: 'nms'

List of values: Mode ∈ {"nms", 'inms'}

Result

NonmaxSuppressionDir returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with SetSystem. If necessary, an exception is raised.

Parallelization Information

NonmaxSuppressionDir is reentrant and automatically parallelized (on tuple level, channel level).

Possible Predecessors

- EdgesImage, SobelDir

Possible Successors

- Threshold, HysteresisThreshold

See also

Alternatives

- NonmaxSuppressionAmp, GraySkeleton, GrayDilationRect

References


Module

Region processing
Detect all gray value plateaus.

Plateaus extracts all points from Image with a gray value greater or equal to the gray value of its neighbors (8-neighborhood) and returns them in Plateaus. Each maximum is returned as a separate region.

**Parameter**

- **Image** (input iconic) ...... image-array

  Image to be processed.

- **Plateaus** (output iconic) .............................................. region  ~ HRegionX / HUntypedObjectX

  Extracted plateaus as regions (one region for each plateau).

**Parallelization Information**

Plateaus is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

GaussImage, SmoothImage

**Possible Successors**

AreaCenter, GetRegionPoints, SelectShape

See also

Monotony, TopographicSketch, CornerResponse, TextureLaws

**Alternatives**

PlateausCenter, GraySkeleton, NonmaxSuppressionAmp, LocalMax

**Module**

Region processing

---

Detect the centers of all gray value plateaus.

PlateausCenter extracts all points from Image with a gray value greater or equal to the gray value of its neighbors (8-neighborhood) and returns them in Plateaus. If more than one of these points are connected (plateau), their center of gravity is returned. Each maximum is returned as a separate region.

**Parameter**

- **Image** (input iconic) ...... image-array

  Image to be processed.

- **Plateaus** (output iconic) .............................................. region  ~ HRegionX / HUntypedObjectX

  Centers of gravity of the extracted plateaus as regions (one region for each plateau).

**Parallelization Information**

PlateausCenter is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

GaussImage, SmoothImage

**Possible Successors**

AreaCenter, GetRegionPoints, SelectShape

See also

Monotony, TopographicSketch, CornerResponse, TextureLaws

**Alternatives**

Plateaus, GraySkeleton, NonmaxSuppressionAmp, LocalMax
Module

Region processing

```c
[out] HRegionX Regions
```

```c
```

Segment an image by “pouring water” over it.

**Pouring** regards the input image as a “mountain range.” Larger gray values correspond to mountain peaks, while smaller gray values correspond to valley bottoms. **Pouring** segments the input image in several steps. First the local maxima are extracted, i.e., pixels which either alone or in the form of an extended plateau have larger gray values than their immediate neighbors (in 4-neighborhood). In the next step, the maxima thus found are the starting points for an expansion until “valley bottoms” are reached. The expansion is done as long as there are chains of pixels in which the gray value gets smaller (like water running downhill from the maxima in all directions). Again, 4-neighborhood is used, but with a weaker condition (smaller or equal). This means that points at valley bottoms may belong to more than one maximum. These areas are at first not assigned to a region, but rather are split among all competing segments in the last step. The split is done by a uniform expansion of all involved segments, until all ambiguous pixels were assigned. The parameter **Mode** determines which steps are executed. The following values are possible:

- **"all"** This is the normal mode of operation. All steps of the segmentation are performed. The regions are assigned to maxima, and overlapping regions are split.
- **"maxima"** The segmentation only extracts the local maxima of the input image. No corresponding regions are extracted.
- **"regions"** The segmentation extracts the local maxima of the input image and the corresponding regions, which are uniquely determined. Areas which could be assigned to more than one maximum are not split.

In order to prevent the algorithm from splitting a uniform background, which is different from the rest of the image, the parameters **MinGray** and **MaxGray** determine gray value thresholds for regions in the image which should be regarded as background. All parts of the image having a gray value smaller than **MinGray** or larger than **MaxGray** are disregarded for the extraction of the maxima as well as for the assignment of regions. For a complete segmentation of the image, **MinGray** = 0 und **MaxGray** = 255 should be selected. In any case, **MinGray** < **MaxGray** must be observed.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (input iconic)</td>
<td>Image to be segmented.</td>
</tr>
<tr>
<td><strong>Regions</strong> (output iconic)</td>
<td>Segmentation result.</td>
</tr>
<tr>
<td><strong>Mode</strong> (input control)</td>
<td>Mode of operation.</td>
</tr>
<tr>
<td><strong>MinGray</strong> (input control)</td>
<td>All gray values smaller than this threshold are disregarded.</td>
</tr>
<tr>
<td><strong>MaxGray</strong> (input control)</td>
<td>All gray values larger than this threshold are disregarded.</td>
</tr>
</tbody>
</table>

**Default Value**

- **\'all\'**
- **"maxima"**
- **"regions"**

**List of values:** `Mode \in \{ \'all\', \'maxima\', \'regions\' \}`

**Input control**

- **MinGray** (input control)
- **MaxGray** (input control)

**Suggested values:**

- **MinGray** \in \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110\}
- **MaxGray** \in \{100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240, 255\}
Typical range of values: $0 \leq \text{MaxGray} \leq 0(\text{lin})$

Minimum Increment: 1

Recommended Increment: 10

Restriction: $((\text{MaxGray} \leq 255) \land (\text{MaxGray} > \text{MinGray}))$

Example

```c
/* Segmentation of a filtering Image */
read_image(Image,'br2')
mean_image(Image,Mean,11,11)
pouring(Mean,Seg,'all',0,255)
disp_image(Mean,WindowHandle)
set_colored(WindowHandle,12)
disp_region(Seg,WindowHandle).

/* Segmentation of a Image with masking of a dark backround */
read_image(Image,'hand')
mean_image(ImageMean,15,15)
pouring(Mean,Seg,'all',40,255)
disp_image(Mean,WindowHandle)
set_colored(WindowHandle,12)
disp_region(Seg,WindowHandle).

/* Segmentation of a histogram */
read_image(Image,'monkey')
texture_laws(Image,Texture,'el',2,5)
draw_region(Region,draw_region)
reduce_domain(Texture,Region,Testreg)
histo_2dim(Testreg,Texture,Region,Histo)
pouring(Histo,Seg,'all',0,255).
```

Complexity

Let $N$ be the number of pixels in the input image and $M$ be the number of found segments, where the enclosing rectangle of the segment $i$ contains $m_i$ pixels. Furthermore, let $K_i$ be the number of chords in segment $i$. Then the runtime complexity is

$$O(3 \times N + \sum_M (3 \times m_i) + \sum_M (K_i)) .$$

Result

Pouring usually returns the value TRUE. If necessary, an exception is raised.

Parallelization Information

Pouring is processed under mutual exclusion against itself and without parallelization.

Possible Predecessors

GaussImage, SmoothImage

See also

Histo2Dim, ExpandRegion, ExpandGray, ExpandGrayRef

Alternatives

Watersheds, LocalMax

Module

Region processing

HALCON/COM Reference Manual, 2005-2-1
Segment an image using region growing.

Region growing segments images into regions of the same intensity — rastered into rectangles of size \( \text{Row} \times \text{Column} \). In order to decide whether two adjacent rectangles belong to the same region only the gray value of their center points is used. If the gray value difference is less than or equal to \( \text{Tolerance} \) the rectangles are merged into one region.

If \( g_1 \) and \( g_2 \) are two gray values to be examined, they are merged into the same region if:

\[
|g_1 - g_2| < \text{Tolerance}
\]

For images of type ‘cyclic’, the following formulas are used:

\[
(|g_1 - g_2| < \text{Tolerance}) \land (|g_1 - g_2| \leq 127)
\]

\[
(256 - |g_1 - g_2| < \text{Tolerance}) \land (|g_1 - g_2| > 127)
\]

For rectangles larger than one pixel, usually the images should be smoothed with a lowpass filter with a size of at least \( \text{Row} \times \text{Column} \) before calling \text{Region growing} (so that the gray values at the centers of the rectangles are “representative” for the whole rectangle). If the image contains little noise and the rectangles are small, the smoothing can be omitted in many cases. This, of course, makes the whole procedure faster.

The resulting regions are collections of rectangles of the chosen size \( \text{Row} \times \text{Column} \). Only regions containing at least \( \text{MinSize} \) points are returned.

Region growing is a very fast operation, and thus suited for time-critical applications.

\begin{itemize}
\item **Attention**
\end{itemize}

\( \text{Column} \) and \( \text{Row} \) are automatically converted to odd values if necessary.

\begin{itemize}
\item **Parameter**
\end{itemize}

\begin{itemize}
\item \textbf{Image} (input iconic) \ldots \text{-array) \( \sim \) HImageX / IHObjectX (byte, direction, cyclic, int1, int2, int4, real)
\item Image to be segmented.
\item \textbf{Regions} (output iconic) \( \sim \) HRegionX / HUntypedObjectX
\item Extracted segments.
\item \textbf{Row} (input control) \( \sim \) long / VARIANT
\item Vertical distance between tested pixels (height of the raster).
\item Default Value : 3
\item Suggested values : \( \text{Row} \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21\} \)
\item Typical range of values : \( 1 \leq \text{Row} \leq 1(\text{lin}) \)
\item Minimum Increment : 2
\item Recommended Increment : 2
\item Restriction : \( (\text{Row} \land \text{odd}) \)
\item \textbf{Column} (input control) \( \sim \) long / VARIANT
\item Horizontal distance between tested pixels (height of the raster).
\item Default Value : 3
\item Suggested values : \( \text{Column} \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21\} \)
\item Typical range of values : \( 1 \leq \text{Column} \leq 1(\text{lin}) \)
\item Minimum Increment : 2
\item Recommended Increment : 2
\item Restriction : \( (\text{Column} \land \text{odd}) \)
\end{itemize}
**Tolerance** (input control) .......................... number $\sim$ VARIANT (integer, real)
Points with a gray value difference less than or equal to tolerance are accumulated into the same object.

Default Value: 6.0
Suggested values: Tolerance \(\in\) \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 18.0, 25.0\}
Typical range of values: \(1.0 \leq \text{Tolerance} \leq 1.0\) (lin)
Minimum Increment: 0.01
Recommended Increment: 1.0
Restriction: \(((0 \leq \text{Tolerance}) \land (\text{Tolerance} < 127))\)

**MinSize** (input control) .......................... integer $\sim$ long / VARIANT
Minimum size of the output regions.

Default Value: 100
Suggested values: MinSize \(\in\) \{1, 5, 10, 20, 50, 100, 200, 500, 1000\}
Typical range of values: \(1 \leq \text{MinSize} \leq 1\) (lin)
Minimum Increment: 1
Recommended Increment: 5
Restriction: \((\text{MinSize} \geq 1)\)

Example

```
read_image(Image,'fabrik')
mean_image(Image,Mean,Row,Column)
regiongrowing(Mean,Result,Row,Column,6.0,100).
```

**Complexity**
Let \(N\) be the number of found regions and \(M\) the number of points in one of these regions. Then the runtime complexity is \(O(N \cdot \log(M) \cdot M)\).

**Regiongrowing** returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags ‘no_object_result’, ‘empty_region_result’, and ‘store_empty_region’ with SetSystem. If necessary, an exception is raised.

**Parallelization Information**
Regiongrowing is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**
MeanImage, GaussImage, SmoothImage, MedianImage, AnisotropeDiff

**Possible Successors**
SelectShape, ReduceDomain, SelectGray

**Alternatives**
RegiongrowingN, RegiongrowingMean, LabelToRegion

**Module**

Region processing

```c
[out] HRegionX Regions HIMageX.RegiongrowingMean
([in] VARIANT StartRows, [in] VARIANT StartColumns, [in] double Tolerance,
 [in] long MinSize

void HOperatorSetX.RegiongrowingMean ([in] IObjectX Image,
 [out] HUntypedObjectX Regions, [in] VARIANT StartRows,
```

Regiongrowing using mean gray values.

RegiongrowingMean performs a regiongrowing using mean gray values of a region, starting from points given by StartRows and StartColumns. At any point in the process the mean gray value of the current region is calculated. Gray values at the boundary of the region are added to the region if they differ from the current mean by less than Tolerance. Regions smaller than MinSize are suppressed.
If no starting points are given (empty tuples), the expansion process starts at the upper leftmost point, and is continued with the first unprocessed point after a region has been created.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image</td>
<td>Image to be segmented.</td>
</tr>
<tr>
<td>Regions</td>
<td>Extracted segments.</td>
</tr>
<tr>
<td>StartRows</td>
<td>Row coordinates of the starting points. Default Value: [] Typical range of values: 0 ≤ StartRows ≤ 0 (lin) Minimum Increment: 1 Recommended Increment: 1</td>
</tr>
<tr>
<td>StartColumns</td>
<td>Column coordinates of the starting points. Default Value: [] Typical range of values: 0 ≤ StartColumns ≤ 0 (lin) Minimum Increment: 1 Recommended Increment: 1</td>
</tr>
<tr>
<td>Tolerance</td>
<td>Maximum deviation from the mean. Default Value: 5.0 Suggested values: [\text{Tolerance} \in {0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 15.0, 17.0, 20.0, 25.0, 30.0, 40.0}] Typical range of values: 0.1 ≤ Tolerance ≤ 0.1 (lin) Minimum Increment: 0.01 Recommended Increment: 1.0 Restriction: ((\text{Tolerance} &gt; 0.0))</td>
</tr>
<tr>
<td>MinSize</td>
<td>Minimum size of a region. Default Value: 100 Suggested values: [\text{MinSize} \in {0, 10, 30, 50, 100, 500, 1000, 2000}] Typical range of values: 0 ≤ MinSize ≤ 0 (lin) Minimum Increment: 1 Recommended Increment: 100 Restriction: ((\text{MinSize} \geq 0))</td>
</tr>
</tbody>
</table>

Region growing Mean returns TRUE if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags ’no object result’, ’empty region result’, and ’store empty region’ with SetSystem. If necessary, an exception is raised.

Region growing Mean is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
GaussImage, SigmaImage, AnisotropeDiff, MedianImage

Possible Successors
SelectShape, ReduceDomain, Opening, ExpandRegion

Alternatives
Region growing, Region growing N

Module
Region processing
Region growing for multi-channel images.

Region growing N performs a multi-channel region growing. The \( n \) channels give rise to an \( n \)-dimensional feature vector. Neighboring points are aggregated into the same region if the difference of their feature vectors with respect to the given metric lies in the interval \([\text{MinTolerance}, \text{MaxTolerance}]\). Only neighbors of the 4-neighborhood are examined. The following metrics can be used:

Let \( g_A \) denote the gray value in the feature vector \( A \) at point \( a \) of the image, and likewise be \( g_B \) the gray value in the feature vector \( B \) at point a neighboring point \( b \). Let \( g(d) \) be the gray value with index \( d \). Furthermore, let \( \text{MinT} \) denote \( \text{MinTolerance} \) and \( \text{MaxT} \) denote \( \text{MaxTolerance} \).

- **1-norm**: Sum of absolute values
  \[
  \text{MinT} \leq \frac{1}{n} \sum |g_A - g_B| \leq \text{MaxT}
  \]

- **2-norm**: Euclidian distance
  \[
  \text{MinT} \leq \frac{1}{n} \sqrt{\sum (g_A - g_B)^2} \leq \text{MaxT}
  \]

- **3-norm**: \( p \)-Norm with \( p = 3 \)
  \[
  \text{MinT} \leq \frac{1}{n} \sqrt[3]{\sum (g_A - g_B)^3} \leq \text{MaxT}
  \]

- **4-norm**: \( p \)-Norm with \( p = 4 \)
  \[
  \text{MinT} \leq \frac{1}{n} \sqrt[4]{\sum (g_A - g_B)^4} \leq \text{MaxT}
  \]

- **n-norm**: Minkowski distance
  \[
  \text{MinT} \leq \frac{1}{n} \sqrt[n]{\sum (g_A - g_B)^n} \leq \text{MaxT}
  \]

- **max-diff**: Supremum distance
  \[
  \text{MinT} \leq \max \{|g_A - g_B|\} \leq \text{MaxT}
  \]

- **min-diff**: Infimum distance
  \[
  \text{MinT} \leq \min \{|g_A - g_B|\} \leq \text{MaxT}
  \]

- **variance**: Variance of gray value differences
  \[
  \text{MinT} \leq \text{Var}(g_A - g_B) \leq \text{MaxT}
  \]

- **dot-product**: Dot product
  \[
  \text{MinT} \leq \frac{1}{n} \sqrt{\sum (g_A g_B)} \leq \text{MaxT}
  \]

- **correlation**: Correlation
  \[
  m_A = \frac{1}{n} \sum g_A
  \]
  \[
  \text{Var}_A = \frac{1}{n} \sqrt{\sum (g_A - m_A)^2}
  \]
  \[
  m_B = \frac{1}{n} \sum g_B
  \]
  \[
  \text{Var}_B = \frac{1}{n} \sqrt{\sum (g_B - m_B)^2}
  \]
  \[
  \text{MinT} \leq \frac{1}{n^2} \sum \frac{(g_A - m_A)(g_B - m_B)}{(\text{Var}_A \text{Var}_B)} \leq \text{MaxT}
  \]
'mean-diff': Difference of arithmetic means

\[ a = \frac{1}{n} \sum g_A \]
\[ b = \frac{1}{n} \sum g_B \]
\[ MinT \leq |a - b| \leq MaxT \]

'mean-ratio': Ratio of arithmetic means

\[ a = \frac{1}{n} \sum g_A \]
\[ b = \frac{1}{n} \sum g_B \]
\[ MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT \]

'length-diff': Difference of the vector lengths

\[ a = \sqrt{\frac{\sum g_A^2}{n}} \]
\[ b = \sqrt{\frac{\sum g_B^2}{n}} \]
\[ MinT \leq |a - b| \leq MaxT \]

'length-ratio': Ratio of the vector lengths

\[ a = \sqrt{\frac{\sum g_A^2}{n}} \]
\[ b = \sqrt{\frac{\sum g_B^2}{n}} \]
\[ MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT \]

'n-norm-ratio': Ratio of the vector lengths w.r.t the p-norm with \( p = n \)

\[ a = \sqrt[p]{\frac{\sum g_A^p}{n}} \]
\[ b = \sqrt[p]{\frac{\sum g_B^p}{n}} \]
\[ MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT \]

'gray-max-diff': Difference of the maximum gray values

\[ a = \max \{|g_A|\} \]
\[ b = \max \{|g_B|\} \]
\[ MinT \leq |a - b| \leq MaxT \]

'gray-max-ratio': Ratio of the maximum gray values

\[ a = \max \{|g_A|\} \]
\[ b = \max \{|g_B|\} \]
\[ MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT \]

'gray-min-diff': Difference of the minimum gray values

\[ a = \min \{|g_A|\} \]
\[ b = \min \{|g_B|\} \]
\[ MinT \leq |a - b| \leq MaxT \]
'gray-min-ratio': Ratio of the minimum gray values

\[ a = \min \{|g_A|\} \]

\[ b = \min \{|g_B|\} \]

\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'variance-diff': Difference of the variances over all gray values (channels)

\[ \text{MinT} \leq |\text{Var}(g_A) - \text{Var}(g_B)| \leq \text{MaxT} \]

'variance-ratio': Ratio of the variances over all gray values (channels)

\[ \text{MinT} \leq \frac{\text{Var}(g_B)}{\text{Var}(g_A)} \leq \text{MaxT} \]

'mean-abs-diff': Difference of the sum of absolute values over all gray values (channels)

\[ a = \sum_{d,k,k<d} |g_A(d) - g_A(k)| \]

\[ b = \sum_{d,k,k<d} |g_B(d) - g_B(k)| \]

\[ \text{MinT} \leq \frac{|a - b|}{\text{Anzahl der Summen}} \leq \text{MaxT} \]

'mean-abs-ratio': Ratio of the sum of absolute values over all gray values (channels)

\[ a = \sum_{d,k,k<d} |g_A(d) - g_A(k)| \]

\[ b = \sum_{d,k,k<d} |g_B(d) - g_B(k)| \]

\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'max-abs-diff': Difference of the maximum distance of the components

\[ a = \max \{g_A(d), g_A(k)\} \]

\[ b = \max \{g_B(d), g_B(k)\} \]

\[ \text{MinT} \leq |a - b| \leq \text{MaxT} \]

'max-abs-ratio': Ratio of the maximum distance of the components

\[ a = \max \{g_A(d), g_A(k)\} \]

\[ b = \max \{g_B(d), g_B(k)\} \]

\[ \text{MinT} \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq \text{MaxT} \]

'min-abs-diff': Difference of the minimum distance of the components

\[ a = \min \{g_A(d), g_A(k)\}, k < d \]

\[ b = \min \{g_B(d), g_B(k)\}, k < d \]

\[ \text{MinT} \leq |a - b| \leq \text{MaxT} \]
'min-abs-ratio': Ratio of the minimum distance of the components

\[
a = \min \{ g_A(d), g_A(k) \}, k < d \\
b = \min \{ g_B(d), g_B(k) \}, k < d \\
\]

\[
MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT
\]

'plane': The following has to hold for all \(d_1, d_2 \in [1, n]\):

\[
g_A(d_1) > g_A(d_2) \Rightarrow g_B(d_1) > g_B(d_2) \\
g_A(d_1) < g_A(d_2) \Rightarrow g_B(d_1) < g_B(d_2)
\]

Regions with less than \textbf{MinSize} are suppressed.

---

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MultiChannelImage</strong> (input iconic)</td>
<td>multi-channel image to be segmented.</td>
</tr>
<tr>
<td><strong>Regions</strong> (output iconic)</td>
<td>segmented regions.</td>
</tr>
<tr>
<td><strong>Metric</strong> (input control)</td>
<td>metric for the distance of the feature vectors.</td>
</tr>
<tr>
<td>Default Value: 2-norm</td>
<td></td>
</tr>
<tr>
<td><strong>MinTolerance</strong> (input control)</td>
<td>lower threshold for the features' distance.</td>
</tr>
<tr>
<td>Default Value: 0.0</td>
<td></td>
</tr>
<tr>
<td>Suggested values: MinTolerance (\in) {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0, 25.0, 30.0}</td>
<td></td>
</tr>
<tr>
<td>Typical range of values: 0.0 (\leq) MinTolerance (\leq) 0.0 (lin)</td>
<td></td>
</tr>
<tr>
<td>Minimum Increment: 0.01</td>
<td></td>
</tr>
<tr>
<td>Recommended Increment: 1.0</td>
<td></td>
</tr>
<tr>
<td><strong>MaxTolerance</strong> (input control)</td>
<td>upper threshold for the features' distance.</td>
</tr>
<tr>
<td>Default Value: 20.0</td>
<td></td>
</tr>
<tr>
<td>Suggested values: MaxTolerance (\in) {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0, 25.0, 30.0}</td>
<td></td>
</tr>
<tr>
<td>Typical range of values: 0.0 (\leq) MaxTolerance (\leq) 0.0 (lin)</td>
<td></td>
</tr>
<tr>
<td>Minimum Increment: 0.01</td>
<td></td>
</tr>
<tr>
<td>Recommended Increment: 1.0</td>
<td></td>
</tr>
<tr>
<td><strong>MinSize</strong> (input control)</td>
<td>minimum size of the output regions.</td>
</tr>
<tr>
<td>Default Value: 30</td>
<td></td>
</tr>
<tr>
<td>Suggested values: MinSize (\in) {1, 10, 25, 50, 100, 200, 500, 1000}</td>
<td></td>
</tr>
<tr>
<td>Typical range of values: 1 (\leq) MinSize (\leq) 1 (lin)</td>
<td></td>
</tr>
<tr>
<td>Minimum Increment: 1</td>
<td></td>
</tr>
<tr>
<td>Recommended Increment: 5</td>
<td></td>
</tr>
</tbody>
</table>

---

**Result**

\textbf{RegionGrowingN} returns \textbf{TRUE} if all parameters are correct. The behavior with respect to the input images and output regions can be determined by setting the values of the flags 'no_object_result', 'empty_region_result', and 'store_empty_region' with \textbf{SetSystem}. If necessary, an exception is raised.

---

**Parallelization Information**

\textbf{RegionGrowingN} is \textbf{reentrant} and automatically \textbf{parallelized} (on tuple level).

---

**Possible Predecessors**

\textbf{Compose2, Compose3}
Select gray values lying within an interval.

Threshold selects the pixels from the input image whose gray values \( g \) fulfill the following condition:

\[
\text{MinGray} \leq g \leq \text{MaxGray}.
\]

All points of an image fulfilling the condition are returned as one region. If more than one gray value interval is passed (tuples for \text{MinGray} and \text{MaxGray}), one separate region is returned for each interval.

Parameter

- **Image** (input iconic) \( \ldots \) image(-array) \( \sim \) HImageX / IObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, d vf )

  Image to be thresholded.

- **Region** (output iconic) \( \ldots \) \( \sim \) HRegionX / HUntypedObjectX

  Regions with gray values lying in the specified interval.

- **MinGray** (input control) \( \ldots \) number(-array) \( \sim \) VARIANT (integer, real)

  Lower threshold for the gray values.

  Default Value : 128.0

  Suggested values : \( \text{MinGray} \in \{0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0\} \)

  Typical range of values : \( 0.0 \leq \text{MinGray} \leq 0.0\) (lin)

  Minimum Increment : 0.01

  Recommended Increment : 5.0

- **MaxGray** (input control) \( \ldots \) number(-array) \( \sim \) VARIANT (integer, real)

  Upper threshold for the gray values.

  Default Value : 255.0

  Suggested values : \( \text{MaxGray} \in \{0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0\} \)

  Typical range of values : \( 0.0 \leq \text{MaxGray} \leq 0.0\) (lin)

  Minimum Increment : 0.01

  Recommended Increment : 5.0

  Restriction : \( \text{MaxGray} \geq \text{MinGray} \)

Example

```
read_image(Image,’fabrik’)
sobel_dir(Image,EdgeAmp,EdgeDir,’sum_abs’,3)
threshold(EdgeAmp,Seg,50,255,2)
skeleton(Seg,Rand)
connection(Rand,Lines)
select_shape(Lines,Edges,’area’,’and’,10,1000000).
```

Complexity

Let \( F \) be the area of the input region. Then the runtime complexity is \( O(F) \).

Result

Threshold returns TRUE if all parameters are correct. The behavior with respect to the input images and
output regions can be determined by setting the values of the flags 'no object result', 'empty region result', and 'store empty region' with SetSystem. If necessary, an exception is raised.

---

Parallelization Information

Threshold is reentrant and automatically parallelized (on tuple level, domain level).

---

Possible Predecessors

HistoToThresh, MinMaxGray, SobelAmp, GaussImage, ReduceDomain, FillInterlace

---

Possible Successors

Connection, Dilation1, Erosion1, Opening, Closing, RankRegion, ShapeTrans, Skeleton

---

See also

DualThreshold, ZeroCrossing, BackgroundSeg, RegionGrowing

---

Alternatives

Class2DimSup, HysteresisThreshold, DynThreshold

---

Module

Region processing

---


---

Extract level crossings from an image with subpixel accuracy.

ThresholdSubPix extracts the level crossings at the level Threshold of the input image Image with subpixel accuracy. The extracted level crossings are returned as XLD-contours in Border. In contrast to the operator Threshold, ThresholdSubPix does not return regions, but the lines which separate regions with a gray value less than Threshold from regions with a gray value greater than Threshold.

For the extraction, the input image is regarded as a surface, in which the gray values are interpolated bilinearly between the centers of the individual pixels. Consistent with the surface thus defined, level crossing lines are extracted for each pixel and linked into topologically sound contours. This means that the level crossing contours are correctly split at junction points. If the image contains extended areas of constant gray value Threshold, only the border of such areas is returned as level crossings.

---

Parameter

- **Image** (input iconic) ................. image \( \sim \) IHObjectX / IHObjectX (byte, int1, int2, uint2, int4, real) Input image.
- **Border** (output iconic) ...................... xld_cont \( \sim \) HXLDContX / HUntypedObjectX Extracted level crossings.
- **Threshold** (input control) ......................... number \( \sim \) VARIANT (real, integer) Threshold for the level crossings.

Default Value : 128

Suggested values : Threshold \( \in \) \{0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0\}

---

Example

read_image(Image,’fabrik’)
threshold_sub_pix(Image,Border,35)
disp_xld(Border,WindowHandle)

---

Result

ThresholdSubPix usually returns the value TRUE. If necessary, an exception is raised.

---

Parallelization Information

ThresholdSubPix is reentrant and processed without parallelization.

---

See also

ZeroCrossingSubPix
CHAPTER 11. SEGMENTATION

Alternatives

Threshold

Module

Sub-pixel operators

Extract watersheds and basins from an image.

**Watersheds** segments an image based on the topology of the gray values. The image is interpreted as a “mountain range.” Higher gray values correspond to “mountains,” while lower gray values correspond to “valleys.” In the resulting mountain range watersheds are extracted. These correspond to the bright ridges between dark basins. On output, the parameter **Basins** contains these basins, while **Watersheds** contains the watersheds, which are at least one pixel wide (points on the ridge which form a plateau). **Watersheds** always is a single region per input image, while **Basins** contains a separate region for each basin. It is advisable to apply a smoothing operator (e.g., **GaussImage**) to the input image before calling **Watersheds** in order to reduce the number of output regions.

If the image contains many fine structures or is noisy, many output regions result, and thus the runtime increases considerably.

**Parameter**

- **GrayImage** (input iconic) ......... image(-array) \sim HImageX / IHObjectX (byte)
  Images to be segmented.
- **Basins** (output iconic) ............. region \sim HRegionX / HUntypedObjectX
  Segments found (dark basins).
- **Watersheds** (output iconic) ........ region(-array) \sim HRegionX / HUntypedObjectX
  Watersheds between the basins.

**Result**

**Watersheds** always returns TRUE. The behavior with respect to the input images and output regions can be determined by setting the values of the flags **'no_object_result'**, **'empty_region_result'**, and **'store_empty_region'** with **SetSystem**. If necessary, an exception is raised.

**Parallelization Information**

**Watersheds** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

- GaussImage, SmoothImage, InvertImage

**Possible Successors**

- ExpandRegion, SelectShape, ReduceDomain, Opening

**Alternatives**

- Pouring

Region processing

**ZeroCrossing** returns the zero crossings of the input image as a region. A pixel is accepted as a zero crossing if its gray value (in **Image**) is zero, or if at least one of its neighbors of the 4-neighborhood has a different sign.

**HALCON/COM Reference Manual, 2005-2-1**
This operator is intended to be used after edge operators returning the second derivative of the image (e.g., \texttt{LaplaceOfGauss}), which were possibly followed by a smoothing operator. In this case, the zero crossings are (candidates for) edges.

\textbf{Parameter}

\begin{itemize}
  \item \textbf{Image} (input iconic) \hspace{1cm} \texttt{image(-array)} \rightarrow \texttt{HImageX / IObjectX (int2, int4, real)}
  \begin{itemize}
    \item Input image.
  \end{itemize}
  \item \textbf{RegionCrossing} (output iconic) \hspace{1cm} \texttt{region(-array)} \rightarrow \texttt{HRegionX / HUntypedObjectX}
  \begin{itemize}
    \item Zero crossings (as region).
  \end{itemize}
\end{itemize}

\textbf{Result}

\texttt{ZeroCrossing} usually returns the value \texttt{TRUE}. If necessary, an exception is raised.

\textbf{Parallelization Information}

\texttt{ZeroCrossing} is \textit{reentrant} and automatically \textit{parallelized} (on \textit{tuple} level).

\textbf{Possible Predecessors}

\begin{itemize}
  \item \texttt{Laplace}, \texttt{LaplaceOfGauss}, \texttt{DerivateGauss}
\end{itemize}

\textbf{Possible Successors}

\begin{itemize}
  \item \texttt{Connection}, \texttt{Skeleton}, \texttt{Boundary}, \texttt{SelectShape}, \texttt{FillUp}
\end{itemize}

\textbf{Alternatives}

\begin{itemize}
  \item \texttt{Threshold}, \texttt{DualThreshold}
\end{itemize}

\textbf{Module}

\texttt{Region processing}

\begin{verbatim}
[out] HXLDContX ZeroCrossings HImageX.ZeroCrossingSubPix ( )

void HOperatorSetX.ZeroCrossingSubPix ([in] IObjectX Image, [out] HUntypedObjectX ZeroCrossings )
\end{verbatim}

Extract zero crossings from an image with subpixel accuracy.

\texttt{ZeroCrossingSubPix} extracts the zero crossings of the input image \texttt{Image} with subpixel accuracy. The extracted zero crossings are returned as XLD-contours in \texttt{ZeroCrossings}. Thus, \texttt{ZeroCrossingSubPix} can be used as a sub-pixel precise edge extractor if the input image is a Laplace-filtered image (see \texttt{Laplace}, \texttt{LaplaceOfGauss}, \texttt{DerivateGauss}).

For the extraction, the input image is regarded as a surface, in which the gray values are interpolated bilinearly between the centers of the individual pixels. Consistent with the surface thus defined, zero crossing lines are extracted for each pixel and linked into topologically sound contours. This means that the zero crossing contours are correctly split at junction points. If the image contains extended areas of constant gray value 0, only the border of such areas is returned as zero crossings.

\textbf{Parameter}

\begin{itemize}
  \item \textbf{Image} (input iconic) \hspace{1cm} \texttt{image(-array)} \rightarrow \texttt{HImageX / IObjectX (int1, int2, int4, real)}
  \begin{itemize}
    \item Input image.
  \end{itemize}
  \item \textbf{ZeroCrossings} (output iconic) \hspace{1cm} \texttt{xld_cont} \rightarrow \texttt{HXLDContX / HUntypedObjectX}
  \begin{itemize}
    \item Extracted zero crossings.
  \end{itemize}
\end{itemize}

\textbf{Example}

\begin{verbatim}
/* Detection zero crossings of the Laplacian-of-Gaussian of an aerial image */
read_image(Image,’mreut’)
derivate_gauss(Image,Laplace,3,’laplace’)
zero_crossing_sub_pix(Laplace,ZeroCrossings)
disp_xld(ZeroCrossings,WindowHandle)

/* Detection of edges, i.e., zero crossings of the Laplacian-of-Gaussian that have a large gradient magnitude, in an aerial image */
read_image(Image,’mreut’)
\end{verbatim}

HALCON 6.1.4
Sigma := 1.5
/* Compensate the threshold for the fact that derivate_gauss(...,'gradient')
calculates a Gaussian-smoothed gradient, in which the edge amplitudes
are too small because of the Gaussian smoothing, to correspond to a true
edge amplitude of 20. */
Threshold := 20/(Sigma*sqrt(2*3.1415926))
derivate_gauss(Image,Gradient,Sigma,'gradient')
threshold(Gradient,Region,Threshold,255)
reduce_domain(Image,Region,ImageReduced)
derivate_gauss(ImageReduced,Laplace,Sigma,'laplace')
zero_crossing_sub_pix(Laplace,Edges)
disp_xld(Edges,WindowHandle)

Result
ZeroCrossingSubPix usually returns the value TRUE. If necessary, an exception is raised.

Parallelization Information
ZeroCrossingSubPix is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
Laplace, LaplaceOfGauss, DiffOfGauss, DerivateGauss

Alternatives
ZeroCrossing

Module
Sub-pixel operators
Chapter 12

System

12.1 Database

```java
[out] long NumOfTuples HSystemX.CountRelation 
([in] String RelationName )

void HOperatorSetX.CountRelation ([in] VARIANT RelationName, 
[out] VARIANT NumOfTuples )
```

Number of entries in the HALCON database.

The operator `CountRelation` counts the number of entries in one of the four relations of the HALCON database. The HALCON database is organized as follows:

There are two basic relations for region-data and image-matrices. The HALCON objects region and image are constructed from elements from these two relations: a region consists of a pointer to a tuple in the region-data relation. An image consists also of a pointer to a tuple in the region-data relation (like a region) and additionally of one or more pointers to tuples in the matrix relation. If there is more than one matrix pointer, the image is called a multi-channel image.

Both regions and images are called objects. A region can be considered as the special case of an iconic object having no image matrices. For reasons of an efficient memory management, the tuples of the region-data relation and the image-matrix relation will be used by different objects together. Therefore there may be for example more images than image matrices. Only the two lowlevel relations are of relevance to the memory consumption. Image objects (regions as well as images) consist only of references on region and matrix data and therefore only need a couple of bytes of memory.

Possible values for `RelationName`:

- 'image': Image matrices. One matrix may also be the component of more than one image (no redundant storage).
- 'region': Regions (the full and the empty region are always available). One region may of course also be the component of more than one image object (no redundant storage).
- 'XLD': eXtended Line Description: Contours, Polygons, parallels, lines, etc. XLD data types don’t have gray values and are stored with subpixel accuracy.
- 'object': Iconic objects. Composed of a region (called region) and optionally image matrices (called image).
- 'tuple': In the compact mode, tuples of iconic objects are stored as a surrogate in this relation. Instead of working with the individual object keys, only this tuple key is used. It depends on the host language, whether the objects are passed individually (Prolog and C++) or as tuples (C, Smalltalk, Lisp, OPS-5).

Certain database objects will be created already by the operator `ResetObjDb` and therefore have to be available all the time (the undefined gray value component, the objects 'full' (FULL_REGION in HALCON/C) and 'empty' (EMPTY_REGION in HALCON/C) as well as the herein included empty and full region). By calling `GetChannelInfo`, the operator therefore appears correspondingly also as 'creator' of the full and empty region. The procedure can be used for example to check the completeness of the `ClearObj` operation.

Attention
CHAPTER 12. SYSTEM

Parameter

▷ RelationName (input control) .......................... string  ~> String / VARIANT
  Relation of interest of the HALCON database.
  Default Value: 'object'
  List of values: RelationName ∈ {'image', 'region', 'XLD', 'object', 'tuple'}
▷ NumOfTuples (output control) .......................... integer  ~> long / VARIANT
  Number of tuples in the relation.

draw_image(512,512,3)
count_relation('image',I1)
count_relation('region',R1)
count_relation('XLD',X1)
count_relation('object',O1)
count_relation('tuple',T1)
read_image(X,'monkey')
count_relation('image',I2)
count_relation('region',R2)
count_relation('XLD',X2)
count_relation('object',O2)
count_relation('tuple',T2)

/*
Result:
  I1 = 1 (undefined image)
  R1 = 2 (full and empty region)
  X1 = 0 (no XLD data)
  O1 = 2 (full and empty objects)
  T1 = 0 (always 0 in the normal mode)
  I2 = 2 (additionally the image 'monkey')
  R2 = 2 (read_image uses the full region)
  X2 = 0 (no XLD data)
  O2 = 3 (additionally the image object X)
  T2 = 0
*/

If the parameter is correct, the operator CountRelation returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

CountRelation is reentrant and processed without parallelization.

Possible Predecessors

ResetObjDb

See also ClearObj

Module

System

[out] VARIANT UsedModules HSystemX.GetModules ([out] long ModuleKey )

void HOperatorSetX.GetModules ([out] VARIANT UsedModules,
[out] VARIANT ModuleKey )

Query of used modules and the module key.

GetModules returns the module numbers of all operators used up to this point. Each operator belongs to one module (maximum 32). Each module has a name, which is returned in UsedModules. Based on the used
modules, a key is generated that is needed for the licence manager. \texttt{GetModules} is normally called at the end of a programm to check the used modules.

\begin{verbatim}
// GetModules is normally called at the end of a programm to check the used modules.

Parameter

- \texttt{UsedModules} (output control) ................................................. \texttt{string} \sim \texttt{VARIANT ( string )}
  Names of used modules.

- \texttt{ModuleKey} (output control) .................................................. \texttt{integer} \sim \texttt{long / VARIANT}
  Key for licence manager.

Parallelization Information

\texttt{GetModules} is \textit{reentrant} and processed \textit{without} parallelization.

Operators not requiring licensing

\begin{verbatim}
void \texttt{HSystemX.ResetObjDb} ([in] long \texttt{DefaultImageWidth},
[in] long \texttt{DefaultImageHeight}, [in] long \texttt{DefaultChannels})

void \texttt{HOperatorSetX.ResetObjDb} ([in] \texttt{VARIANT DefaultImageWidth},
[in] \texttt{VARIANT DefaultImageHeight}, [in] \texttt{VARIANT DefaultChannels})
\end{verbatim}

Initialization of the HALCON system.

The operator \texttt{ResetObjDb} initializes the HALCON system. With this procedure the four relations (grayvalue data, region data, iconic objects and object tuplets) which are necessary for image processing with HALCON will be installed (see also \texttt{CountRelation}). In case the relations already exist, all tuplets in the relations will be deallocatted!

The parameters \texttt{DefaultImageWidth} and \texttt{DefaultImageHeight} provide the initial values for the global maximum image size. If the first created object is an image, (e.g. \texttt{ReadImage}), the set values will be overruled in Standard-HALCON by the size of this picture. Instead of this, in Parallel HALCON the set values will only be changed, if they are smaller than the size of the created object. If on the other hand the first object to be created is a region, both in Standard- and in Parallel HALCON the values will only be adjusted in case the new image is larger than the set values. This is not only the case for the first image which is created or read: the global image size will always be enlarged, if larger images are created.

The global image size is of consequence for the opening of windows (\texttt{OpenWindow}) and the clipping of regions. Whenever the clip mode is activated, regions will be clipped according to the global image size (\texttt{SetSystem ('clipRegion','true')}). This can lead to problems if images of various sizes are used. In this case only the fact that a region is smaller or of the same size as the largest image can be guaranteed.

The parameter \texttt{DefaultChannels} returns the most frequent number of channels of an image object. This value can be set to 0 if for the most part regions are used. If more channels than those having been set at the initialization are necessary for one image, the number will be enlarged dynamically for this image. If less channels than those having been set at the initialization are necessary for the image, the superfluous channels will be set as undefined. For the user this will seem as if they were non existent, however, memory is allocated unnecessarily.

The parameter values can be queried using the operator \texttt{GetSystem}.

\begin{verbatim}
Attention

If the operator \texttt{ResetObjDb} is not called at the beginning of a HALCON session, HALCON will be initialized automatically by the operator \texttt{ResetObjDb(128,128,0)}. In case the operator \texttt{ResetObjDb} is called again, all image objects in the database will be deallocatted.

Parameter

- \texttt{DefaultImageWidth} (input control) ................................. \texttt{integer} \sim \texttt{long / VARIANT}
  Default image width (in pixels).
  \textbf{Default Value} : 128
  \textbf{Suggested values} : \texttt{DefaultImageWidth} \in \{64, 128, 256, 512, 525, 1024\}

- \texttt{DefaultImageHeight} (input control) ................................. \texttt{integer} \sim \texttt{long / VARIANT}
  Default image height (in pixels).
  \textbf{Default Value} : 128
  \textbf{Suggested values} : \texttt{DefaultImageHeight} \in \{64, 128, 256, 512, 768, 1024\}
\end{verbatim}

HALCON 6.1.4
DefaultChannels (input control) .............................................. integer ~ long / VARIANT
Usual number of channels by which the system constant ’max_channels’ is limited.

Default Value : 0
Suggested values : DefaultChannels ∈ {0, 1, 2, 3, 4, 5, 6, 7}

The operator ResetObjDb returns the value TRUE if the parameter values are correct. Otherwise an exception will be raised.

Parallelization Information
ResetObjDb is reentrant and processed without parallelization.

See also GetChannelInfo, CountRelation

Module

12.2 Error-Handling

State of the HALCON control modes.

Executing the operator GetCheck the user can inquire what kind of control modes are currently activated and which are not. Check gives the tuplet containing the names of the control modes (see also SetCheck) which are preceded by a tilde (˜, e.g. ’˜data’), if the corresponding control is deactivated.

Attention

Parameter

Check (output control) ....................................................... string ~ VARIANT( string )
Tuplet of the currently activated control modes.

GetCheck always returns the value TRUE.

Parallelization Information
GetCheck is reentrant and processed without parallelization.

Possible Predecessors
SetCheck

See also

SetCheck

Module

System

Inquiry after the error text of a HALCON error number.

The operator GetErrorText returns the error text for the corresponding HALCON error number. This is indeed the same text which will be given during an exception. The operator GetErrorText is especially useful if the error treatment is programmed by the users themselves (see also SetCheck(::’˜give_error’:)).

Attention

Unknown error numbers will trigger a standard message.
**12.2. ERROR-HANDLING**

- **Parameter**
  - **ErrorNumber** (input control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) integer \(\sim long / VARIANT\)
    
    - **Restriction**: \((1 \leq ErrorNumber) \land (ErrorNumber \leq 36000)\)

- **Parameter**
  - **ErrorText** (output control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) string \(\sim String / VARIANT\)

**Result**

The operator `GetErrorText` always returns the value TRUE.

**Parallelization Information**

`GetErrorText` is reentrant and processed without parallelization.

**Possible Predecessors**

- **SetCheck**

**Attention**

**Parameter**

- **ErrorNumber** (input control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) integer \(\sim long / VARIANT\)
  
  - **Restriction**: \((1 \leq ErrorNumber) \land (ErrorNumber \leq 36000)\)

- **ErrorText** (output control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) string \(\sim String / VARIANT\)

**Result**

The operator `GetErrorText` always returns the value TRUE.

**Parallelization Information**

`GetErrorText` is reentrant and processed without parallelization.

**Possible Predecessors**

- **SetCheck**

**Attention**

- **Parameter**
  - **Class** (input control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) string \(\sim String / VARIANT\)

  - **Default Value**: `mode`

  - **List of values**: `Class \in \{ 'mode', 'procedure', 'input_control', 'output_control', 'parameter_values', 'input_gray_window', 'output_gray_window', 'input_region_window', 'db', 'output_region_window', 'halt', 'timeout', 'button_window', 'button_window', 'button_click', 'button_notify', 'log_file', 'error', 'internal' \}

  - **Value** (output control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) string \(\sim VARIANT(\) string, integer, real \) State of the control mode.

**Result**

The operator `GetSpy` returns the value TRUE if the parameter `Class` is correct. Otherwise an exception is raised.

**Parallelization Information**

`GetSpy` is reentrant and processed without parallelization.

**Possible Predecessors**

- **ResetObjDb**

**See also**

- **SetSpy**, **QuerySpy**

**Module**

**System**

```c
[out] VARIANT Value HSystemX.GetSpy ([in] String Class )
void HOperatorSetX.GetSpy ([in] VARIANT Class, [out] VARIANT Value )
```

Current configuration of the HALCON debugging-tool.

The operator `GetSpy` returns the current configuration of spy, the HALCON debugging tool. The available control modes (possible choices for `Class`) as well as the corresponding tuning possibilities (possible values for `Value`) can be called up by using the operator `QuerySpy`. You will find a more detailed description under `SetSpy`.

**Attention**

- **Parameter**
  - **Class** (input control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) string \(\sim String / VARIANT\)

  - **Default Value**: `mode`

  - **List of values**: `Class \in \{ 'mode', 'procedure', 'input_control', 'output_control', 'parameter_values', 'input_gray_window', 'output_gray_window', 'input_region_window', 'db', 'output_region_window', 'halt', 'timeout', 'button_window', 'button_window', 'button_click', 'button_notify', 'log_file', 'error', 'internal' \}

  - **Value** (output control) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) \(\ldots\) string \(\sim VARIANT(\) string, integer, real \) State of the control mode.

**Result**

The operator `GetSpy` returns the value TRUE if the parameter `Class` is correct. Otherwise an exception is raised.

**Parallelization Information**

`GetSpy` is reentrant and processed without parallelization.

**Possible Predecessors**

- **ResetObjDb**

**See also**

- **SetSpy**, **QuerySpy**

**Module**

**System**

HALCON 6.1.4
Inquiring for possible settings of the HALCON debugging tool.

The operator `QuerySpy` returns all possible settings of spy, the HALCON debugging tool, i.e. all the available control modes (`Classes`) as well as the corresponding possible ways of setting (`Values`). For a more detailed description of spy see operator `SetSpy`.

Attention

The values of `Values` cannot be used as direct input for `SetSpy`, as they are transmitted as a symbolic constant.

Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Classes</strong></td>
<td>(output control) string <code>~</code> VARIANT (string)</td>
</tr>
<tr>
<td><strong>Values</strong></td>
<td>(output control) string <code>~</code> VARIANT (string)</td>
</tr>
</tbody>
</table>

Result

`querySpy` always returns the value TRUE.

Parallelization Information

`QuerySpy` is reentrant, local, and processed without parallelization.

Possible Predecessors

ResetObjDb

See also

SetSpy, GetSpy

Module

System

Activating and deactivating of HALCON control modes.

With the help of the operator `SetCheck` different control modes of the HALCON system can be activated or deactivated. If a certain control mode is activated, parameters etc. will be checked at runtime. Whenever an inconsistency is hereby detected, the program will be interrupted by an exception.

It is recommendable to activate the control modes during the development of a program and to deactivate them only after a successfully concluded testrun. For if the control mode is deactivated and an error occurs, the system may react in an unpredictable way.

The HALCON system provides various possible control modes which can be activated and deactivated independently. By calling the operator `SetCheck` with the name (`Check`) of the desired control mode, this control mode is activated; the control mode is deactivated by passing its name prefixed with a tilde ('
', z.B. '"data"
').

Available control modes:

**'color':** If this control mode is activated, only colors may be used which are supported by the display for the currently active window. Otherwise an error message is displayed.

In case of deactivated control mode and non existent colors, the nearest color is used (see also `SetColor`, `SetGray`, `SetRgb`).

**'text':** If this control mode is activated, it will check the coordinates during the setting of the text cursor as well as during the display of strings (`WriteString`) to the effect whether a part of a sign would lie outside the windowframe (a fact which is not forbidden in principle by the system).

If the control mode is deactivated, the text will be clipped at the windowframe.
12.2. ERROR-HANDLING

'parameter': (For HALCON/PRO only)
If this control mode is activated, output parameter may not be instantiated by calling a procedure.
Otherwise a normal unification mechanism is used.

'data': (For program development)
Checks the consistency of image objects (regions and grayvalue components).

'interface': If this control mode is activated, the interface between the host language and the HALCON proce-
dures will be checked in course (e.g. typifying and counting of the values).

'database': This is a consistency check of the database (e.g. checks whether an object which shall be canceled
does indeed exist or not.)

'give_error': Determines whether errors shall trigger exceptions or not. If this control modes is deactivated,
the application program must provide a suitable error treatment itself. Please note that errors which are
not reported usually lead to undefined output parameters which may cause an unpredictable reaction of the
program.

'father': If this control mode is activated by calling the operator OpenWindow, HALCON allows only the usage
of the number of another HALCON window as the "father" of the new window; otherwise it allows also
other X Window IDs.

'region': For program development)
Checks the consistency of chords (this may lead to a notable speed reduction of routines).

'clear': Normally, if a list of objects shall be canceled by using ClearObj, an exception will be raised, in case
individual objects do not or no longer exist. If the 'clear’ mode is activated, such objects will be ignored.

'all': Activates all control modes.

'none': Deactivates all control modes.

'default': Original setting.

Attention

Parameter

Check (input control) ......................................................... string(-array) \rightarrow\text{VARIANT}(\text{string})
Desired control mode.
Default Value: 'default'
List of values: Check \in \{ 'color', 'text', 'database', 'data', 'interface', 'give_error', 'father', 'region', 'clear',
'all', 'none', 'default' \}

Result

The operator SetCheck returns the value TRUE, if the parameters are correct. Otherwise an exception will be
raised.

Parallelization Information

SetCheck is reentrant and processed without parallelization.

See also

GetCheck, SetColor, SetRgb, SetHsi, WriteString

Module

System

void HSystemX.SetSpy ([in] String Class, [in] VARIANT Value )
void HOperatorSetX.SetSpy ([in] VARIANT Class, [in] VARIANT Value )

Control of the HALCON Debugging Tools.
The operator SetSpy is the HALCON debugging tool. This tool allows the flexible control of the input and
output data of HALCON-operators - in graphical as well as in textual form. The datacontrol is activated by using

SetSpy(::'mode',on').
and deactivated by using

\[
\text{SetSpy}(:, \text{mode}', \text{off}')
\]

The debugging tool can further be activated with the help of the environment variable HALCONSPY. The definition of this variable corresponds to calling up ‘mode’ and ‘on’.

The following control modes can be tuned (in any desired combination of course) with the help of `Class/Value`:

<table>
<thead>
<tr>
<th>Class Meaning / Value</th>
<th>Meaning / Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>operator</code></td>
<td>When a routine is called, its name and the names of its parameters will be given (in TRIAS notation).</td>
<td><code>on</code> or <code>off</code></td>
</tr>
<tr>
<td><code>input_control</code></td>
<td>When a routine is called, the names and values of the input control parameters will be given.</td>
<td><code>on</code> or <code>off</code></td>
</tr>
<tr>
<td><code>output_control</code></td>
<td>When a routine is called, the names and values of the output control parameters are given.</td>
<td><code>on</code> or <code>off</code></td>
</tr>
<tr>
<td><code>parameter_values</code></td>
<td>Additional information on <code>input_control</code> and <code>output_control</code>: indicates how many values per parameter shall be displayed at most (maximum tuplet length of the output).</td>
<td>Tuplet length (integer)</td>
</tr>
<tr>
<td><code>db</code></td>
<td>Information concerning the 4 relations in the HALCON-database. This is especially valuable in looking for forgotten <code>ClearObj</code>.</td>
<td><code>on</code> or <code>off</code></td>
</tr>
<tr>
<td><code>input_gray_window</code></td>
<td>Any reading access of the gray-value component of an (input) image object will cause the gray-value component to be shown in the indicated window (Window-ID; 'none' will deactivate this control).</td>
<td>Window-ID (integer) or 'none'</td>
</tr>
<tr>
<td><code>output_gray_window</code></td>
<td>As soon as the gray-value component of an (output) image object is set, spy will show this gray-value component in the indicated window (Window-ID; 'none' will deactivate this control).</td>
<td>Window-ID (integer) or 'none'</td>
</tr>
<tr>
<td><code>input_region_window</code></td>
<td>Any reading access of the region of an (input) iconic object will cause this region to be shown in the indicated window (Window-ID; 'none' will deactivate this control).</td>
<td>Window-ID (integer) or 'none'</td>
</tr>
<tr>
<td><code>output_region_window</code></td>
<td>As soon as the region of an (output) iconic object is set, spy will show this region in the indicated window (Window-ID; 'none' will deactivate this control).</td>
<td>Window-ID (integer) or 'none'</td>
</tr>
<tr>
<td><code>time</code></td>
<td>Processing time of the operator</td>
<td><code>on</code> or <code>off</code></td>
</tr>
<tr>
<td><code>halt</code></td>
<td>Determines whether there is a halt after every individual action (‘multiple’) or only at the end of each operator (‘single’). The parameter is only effective if the halt has been activated by ‘timeout’ or ‘button.window’.</td>
<td><code>single</code> or <code>multiple</code></td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>After every output there will be a halt of the indicated number of seconds.</td>
<td>Seconds (real)</td>
</tr>
</tbody>
</table>
'button_window'  Alternative to 'timeout': after every output spy waits until the cursor indicates ('button_click' = 'false') or clicks into ('button_click' = 'true') the indicated window. (Window-ID; 'none' will deactivate this control).
    Value: Window-ID (integer) or 'none'
    default: 'none'

'button_click'  Additional option for 'button_window': determines whether or not a mouse-click has to be waited for after an output.
    Value: 'on' or 'off'
    default: 'off'

'button_notify'  If 'button_notify' is activated, spy generates a beep after every output. This is useful in combination with 'button_window'.
    Value: 'on' or 'off'
    default: 'off'

'log_file'  Spy can hereby divert the text output into a file having been opened with open_file.
    Value: a file handle (see OpenFile)

'error'  If 'error' is activated and an internal error occurs, spy will show the internal procedures (file/line) concerned.
    Value: 'on' or 'off'
    default: 'off'

'internal'  If 'internal' is activated, spy will display the internal procedures and their parameters (file/line) while an HALCON-operator is processed.
    Value: 'on' or 'off'
    default: 'off'

Attention

Parameter

Class (input control)  string  \(\sim\)  String / VARIANT

Control mode

Default Value: 'mode'

List of values: Class \(\in\) { 'mode', 'operator', 'input_control', 'output_control', 'parameter_values', 'input_gray_window', 'db', 'time', 'output_gray_window', 'output_region_window', 'input_region_window', 'halt', 'timeout', 'button_window', 'button_click', 'button_notify', 'log_file', 'error', 'internal' }

Value (input control)  string  \(\sim\)  VARIANT ( string, integer, real )

State of the control mode to be set.

Default Value: 'on'

Suggested values: Value \(\in\) { 'on', 'off', 1, 2, 3, 4, 5, 10, 50, 0.0, 1.0, 2.0, 5.0, 10.0 }

Result

The operator SetSpy returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

Parallelization Information

SetSpy is reentrant, local, and processed without parallelization.

Possible Predecessors

ResetObjDb

See also

GetSpy, QuerySpy

Module

System
12.3  Information

Display the manual information of a procedure on the screen.

The operator DispInfo together with the previewer DispProgram shows the manual entry of the indicated procedure on the screen. The individual files (ProcName.Extension) are located in the HALCON-Directory ”doc/ps/reference/LANGUAGE/*”, whereby the value for LANGUAGE will be determined by the currently used host language. The directory can also be generated with the help of the operator SetSystem (::‘referenceDir’,’Pfad’):

\[
\]

\[
\]

Parameter

\(\triangleright\) ProcName (input control) \[\text{proc.name } \sim \text{String} / \text{VARIANT}\]

Name of the seeked procedure.

Default Value: ’read_image’

\(\triangleright\) Machine (input control) \[\text{string}\]

Name of the computer to which the data shall be transmitted or empty string.

Default Value: ”

\(\triangleright\) DispProgram (input control) \[\text{string}\]

Name of the program which shall display the help text.

Default Value: ’ghostview’

Suggested values: DispProgram ∈ \{'gs’, ’ghostview’\}

\(\triangleright\) Extension (input control) \[\text{string}\]

Extension of the helpfile.

Default Value: ’ps’

Result

If the parameter values are correct and the file and the program are available, the operator DispInfo returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

DispInfo is reentrant and processed without parallelization.

See also

OpenWindow

Image / region / XLD management

Get information concerning the chapters on procedures.

The operator GetChapterInfo gives information concerning the chapters on procedures. If instead of Chapter the empty string is transmitted, the routine will provide in Info the names of all chapters. If on the other hand a certain chapter or a chapter and its subchapter(s) are indicated (by a tuple of names), the corresponding subchapters or - in case there are no further subchapters - the names of the corresponding procedures will be given. The organization of the chapters on procedures is the same as the organization of chapters and subchapters in the HALCON-manual. Please note: The chapters on procedures respectively the subchapters concerning an individual procedure can be called by using the operator GetOperatorInfo(::<Name>,’chapter’,Info:).

The Online-texts will be taken from the files english.hlp, english.sta, english.num and english.idx, which will be searched by HALCON in the currently used directory or the directory ’help_dir’ (see also GetSystem and SetSystem).
12.3. INFORMATION

Parameter

▷ **Chapter** (input control) .................................................. string(-array)  \(\sim\) VARIANT (string)  
Procedure class or subclass of interest.

Default Value: ”

▷ **Info** (output control) .................................................. string  \(\sim\) VARIANT (string)  
Procedure classes (Chapter = ”) or procedure subclasses respectively procedures.

Result

If the parameter values are correct and the helpfile is available, the operator *GetChapterInfo* returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

*GetChapterInfo* is processed completely exclusively without parallelization.

Possible Predecessors

GetSystem, SetSystem

See also

GetOperatorInfo, GetSystem, SetSystem

Module

Image / region / XLD management

Get keywords which are assigned to procedures.

The operator *GetKeywords* returns all the keywords in the online-texts corresponding to those procedures which have the indicated substring *ProcName* in their name. If instead of *ProcName* the empty string is transmitted, the operator *GetKeywords* returns all keywords. The keywords of an individual procedure can also be called by using the operator *GetOperatorInfo*. The online-texts will be taken from the files *english.hlp*, *english.sta*, *english.num*, *english.key* and *english.idx*, which are searched by HALCON in the currently used directory and in the directory 'help_dir' (see also *GetSystem* and *SetSystem*).

Attention

Parameter

▷ **ProcName** (input control) .................................................. proc_name  \(\sim\) String / VARIANT  
Substring in the names of those procedures for which keywords are needed.

Default Value: ’get_keywords’

▷ **Keywords** (output control) .................................................. string  \(\sim\) VARIANT (string)  
Keywords for the procedures.

Result

The operator *GetKeywords* returns the value TRUE if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

Parallelization Information

*GetKeywords* is processed completely exclusively without parallelization.

Possible Predecessors

GetChapterInfo

See also

GetOperatorName, SearchOperator, GetParamInfo

Alternatives

GetOperatorInfo

Module

Image / region / XLD management
Get information concerning a HALCON-procedure.

With the help of the operator GetOperatorInfo the online-texts concerning a certain procedure can be called (see also GetOperatorName). The form of information available for all procedures (Slot) can be called using the operator QueryOperatorInfo. For the time being the following slots are available:

'short': Short description of the procedure.

'abstract': Description of the procedure.

'procedure_class': Name(s) of the chapter(s) in the procedure hierarchy (chapter, subchapter in the HALCON manual).

'functionality': Functionality is equivalent to the object class to which the procedure can be assigned.

'keywords': Keywords of the procedure (optional).

'exmaple': Example for the use of the procedure (optional). The operator 'example.LANGUAGE' (LANGUAGE \in \{c,c++,smalltalk,triias\}) calls up examples for a certain language if available. If the language is not indicated or if no example is available in this language, the TRIAS-example will be returned.

'complexity': Complexity of the procedure (optional).

'effect': Not in use so far.

'alternatives': Alternative procedures (optional).

'see_also': Procedures containing further information (optional).

'predecessor': Possible and sensible predecessor.

'successor': Possible and sensible successor.

'return_value': Return value of the procedure (TRUE, FALSE, FAIL, VOID or EXCEPTION).

'attention': Restrictions and advice concerning the correct use of the procedure (optional).

'parameter': Names of the parameter of the procedure (see also GetParamInfo).

'references': Literary references (optional).

'module': The module to which the operator is assigned.

'html_path': The directory where the HTML documentation of the operator resides.

'parameter_relations': Assurances concerning the parameters (optional).

The texts will be taken from the files english.hlp, english.stat, english.key, english.num und english.idx which will be searched by HALCON in the currently used directory or in the directory 'help_dir' (respectively 'user_help_dir') (see also GetSystem and SetSystem). By adding '.latex' after the slotname, the text of slots containing textual information can be made available in \LaTeX notation.

\begin{verbatim}
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcName</td>
<td>Name of the procedure on which more information is needed.</td>
</tr>
<tr>
<td>Default Value</td>
<td>'get_operator_info'</td>
</tr>
<tr>
<td>Slot</td>
<td>Desired information.</td>
</tr>
<tr>
<td>Default Value</td>
<td>'abstract'</td>
</tr>
<tr>
<td>List of values</td>
<td>Slot \in {'short', 'abstract', 'procedure_class', 'functionality', 'effect', 'complexity', 'predecessor', 'successor', 'alternatives', 'see_also', 'keywords', 'example', 'attention', 'return_value', 'references', 'sourcefiles', 'deffile', 'module', 'html_path'}'</td>
</tr>
<tr>
<td>Information</td>
<td>Information (empty if no information is available)</td>
</tr>
</tbody>
</table>
\end{verbatim}
The operator \texttt{GetOperatorInfo} returns the value \texttt{TRUE} if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

\textbf{Parallelization Information} \texttt{GetOperatorInfo} is processed \textit{completely exclusively} without parallelization.

\textbf{Possible Predecessors} \texttt{GetKeywords, SearchOperator, GetOperatorName, QueryOperatorInfo, QueryParamInfo, GetParamInfo}

\textbf{Possible Successors} \texttt{GetParamNames, GetParamNum, GetParamTypes}

\textbf{See also} \texttt{QueryOperatorInfo, GetParamInfo, GetOperatorName, GetParamNum, GetParamTypes}

\textbf{Alternatives} \texttt{GetParamNames}

\textbf{Module} \textit{Image / region / XLD management}

\begin{verbatim}
[out] VARIANT ProcNames \texttt{HInfoX.GetOperatorName ([in] String Pattern )}

void \texttt{HOperatorSetX.GetOperatorName ([in] VARIANT Pattern, [out] VARIANT ProcNames )}
\end{verbatim}

Get procedures with the given string as a substring of their name.

The operator \texttt{GetOperatorName} takes a string (\texttt{Pattern}) as input and searches all HALCON-procedures having this string as a substring in their name. If an empty string is entered, the names of all procedures available will be returned.

\textbf{Attention}

\textbf{Parameter}

\begin{itemize}
\item \texttt{Pattern} (input control) \texttt{String / VARIANT} Substring of the seeked names (empty \texttt{<==>} all names).
\item \texttt{Default Value} : \texttt{’info’}
\item \texttt{ProcNames} (output control) \texttt{VARIANT ( string )} Detected procedure names.
\end{itemize}

\textbf{Result}

The operator \texttt{GetOperatorName} returns the value \texttt{TRUE} if the helpfiles are available. Otherwise an exception handling is raised.

\textbf{Parallelization Information} \texttt{GetOperatorName} is \textit{reentrant, local}, and processed \textit{without} parallelization.

\textbf{Possible Successors} \texttt{GetOperatorInfo, GetParamNames, GetParamNum, GetParamTypes}

\textbf{See also} \texttt{GetOperatorInfo, GetParamNames, GetParamNum, GetParamTypes}

\textbf{Alternatives} \texttt{SearchOperator}

\textbf{Module} \textit{Image / region / XLD management}
Get information concerning the procedure parameters.

The operator `GetParamInfo` is used for calling up the online-texts assigned to a parameter of an indicated procedure. The form of information available for each parameter (Slot), can be called up by using the operator `QueryParamInfo`. At the moment the following slots are available:

- **description**: Description of the parameter.
- **description.latex**: Description of the parameter in \LaTeX notation.
- **parameter_class**: Parameter classes: 'input_object', 'output_object', 'input_control' oder 'output_control'.
- **type_list**: Permitted type(s) of data for parameter values (for control parameters only). Value: 'real', 'integer' oder 'string'.
- **default_type**: Default-type for parameter values (for control parameters only). This type of parameter is the one HALCON/C uses in the "simple mode". If 'none' is indicated, the "tuple mode" must be used. Value: 'real', 'integer', 'string' oder 'none'.
- **sem_type**: Semantic type of the parameter. This is important to allow the assignment of the parameters to object classes in object-oriented languages (C++, Smalltalk). If more than one parameter belongs semantically to one type, this fact is indicated as well. So far the following objects are supported:
  - object, image, region, xld,
  - xld_cont, xld_para, xld_poly, xld_ext_para, xld_mod_para,
  - integer, real, number, string,
  - channel, grayval, window,
  - histogram, distribution,
  - point(.x, .y), extent(.x, .y),
  - angle(.rad oder .deg),
  - circle(.center.x, .center.y, .radius),
  - arc(.center.x, .center.y, .angle.rad, .begin.x, .begin.y),
  - ellipse(.center.x, .center.y, .angle.rad, .radius1, .radius2),
  - line(.begin.x, .begin.y, .end.x, .end.y)
  - rectangle(.origin.x, .origin.y, .corner.x, .corner.y bzw. .extent.x, .extent.y),
  - polygon(.x, .y), contour(.x, .y),
  - coordinates(.x, .y), chord(.x1, .x2, .y),
  - chain(.begin.x, .begin.y, .code).
- **default_value**: Default-value for the parameter (for input-control parameters only). It is the question of mere information only (the parameter value must be transmitted explicitly, even if the default-value is used). This entry serves only as a notice, a point of departure for own experiments. The values have been selected so that they normally do not cause any errors but generate something that makes sense.
- **multi_value**: 'true', if more than one value is permitted in this parameter position, otherwise 'false'.
- **multichannel**: 'true', in case the input image object may be multichannel.
- **mixed_type**: For control parameters exclusively and only if value tuples ('multi_value'-'true') and various types of data are permitted for the parameter values ('type_list' having more than one value). In this case Slot indicates, whether values of various types may be mixed in one tuple ('true' or 'false').
- **values**: Selection of values (optional).
- **value_list**: In case a parameter can take only a limited number of values, this fact will be indicated explicitly (optional).
- **valuemin**: Minimum value of a value interval.
- **valuemax**: Maximum value of a value interval.
- **valuefunction**: Function describing the course of the values for a series of tests (lin, log, quadr, ...).
- **steprec**: Recommended step width for the parameter values in a series of tests.

```c
[out] VARIANT Information HInfoX.GetParamInfo ([in] String ProcName,
[in] String ParamName, [in] String Slot )

void HOperatorSetX.GetParamInfo ([in] VARIANT ProcName,
```
'steprec': Minimum step width of the parameter values in a series of tests.

'valuenumber': Expression describing the number of parameters as such or in relation to other parameters.

'assertion': Expression describing the parameter values as such or in relation to other parameters.

The online-texts will be taken from the files english.hlp, english.sta, english.key, english.num and english.idx which will be searched by HALCON in the currently used directory or the directory 'help_dir' (see also GetSystem and SetSystem).

Attention

Parameter

ProcName (input control) .................................................. proc_name  ~ String / VARIANT
Name of the procedure on whose parameter more information is needed.
Default Value: 'get_param_info'

ParamName (input control) .................................................. string  ~ String / VARIANT
Name of the parameter on which more information is needed.
Default Value: 'Slot'

Slot (input control) ....................................................... string  ~ String / VARIANT
Desired information.
Default Value: 'description'
List of values: Slot ∈ {'description', 'type_list', 'default_type', 'sem_type', 'default_value', 'values', 'value_list', 'valuemin', 'valuemax', 'valuefunction', 'valuenumber', 'assertion', 'steprec', 'stepmin', 'mixed_type', 'multivalue', 'multichannel'}

Information (output control) ............................................ string(-array)  ~ VARIANT( string )
Information (empty in case there is no information available).

Parallelization Information

GetParamInfo is processed completely exclusively without parallelization.

Possible Predecessors

GetKeywords, SearchOperator

See also

QueryParamInfo, GetOperatorInfo, GetOperatorName

Alternatives

GetParamNames, GetParamNum, GetParamTypes

Module

Image / region / XLD management

Get the names of the parameters of a HALCON-procedure.
For the HALCON-procedure indicated in ProcName the operator GetParamNames returns the names of the input objects, the output objects, of the input control parameters and the output control parameters.

ProcName (input control) .................................................. proc_name  ~ String / VARIANT
Name of the procedure.
Default Value: 'get_param_names'

InpObjPar (output control) .............................................. string  ~ VARIANT( string )
Names of the input objects.
The operator **GetParamNames** returns the value TRUE if the name of the procedure exists and the helpfiles are available. Otherwise an exception handling is raised.

**Parallelization Information**

GetParamNames is reentrant and processed without parallelization.

**Possible Predecessors**

GetKeywords, SearchOperator, GetOperatorName, GetOperatorInfo

**Possible Successors**

GetParamNum, GetParamTypes

**See also**

GetParamNum, GetParamTypes, GetOperatorName

**Alternatives**

GetOperatorInfo, GetParamInfo

**Module**

Image / region / XLD management

```c
[out] String HInfoX.GetParamNum ([in] String ProcName,
[out] long InpObjPar, [out] long OutpObjPar, [out] long InpCtrlPar,
[out] long OutpCtrlPar, [out] String Type )

void HOperatorSetX.GetParamNum ([in] VARIANT ProcName,
[out] VARIANT CName, [out] VARIANT InpObjPar, [out] VARIANT OutpObjPar,
[out] VARIANT InpCtrlPar, [out] VARIANT OutpCtrlPar, [out] VARIANT Type )
```

The operator **GetParamNum** returns the number of the input and output object parameters, as well as the input and output control parameters for the indicated HALCON-procedure. Further, you will receive the name of the C-function (**CName**) called by the procedure. The output parameter **Type** indicates, whether the procedure is a system procedure or an user procedure.

**Parameter**

- **ProcName** (input control) .......................................................... proc_name ~ String / VARIANT
  Name of the procedure.
  **Default Value** : 'get_paramnum'

- **CName** (output control) ......................................................... string ~ String / VARIANT
  Name of the called C-function.

- **InpObjPar** (output control) .................................................. integer ~ long / VARIANT
  Number of the input object parameters.

- **OutpObjPar** (output control) ................................................ integer ~ long / VARIANT
  Number of the output object parameters.

- **InpCtrlPar** (output control) ................................................ integer ~ long / VARIANT
  Number of the input control parameters.

- **OutpCtrlPar** (output control) ................................................ integer ~ long / VARIANT
  Number of the output control parameters.

- **Type** (output control) ......................................................... string ~ String / VARIANT
  System procedure or user procedure.
  **Suggested values** : Type ∈ {'system', 'user'}
The operator \texttt{GetParamNum} returns the value \texttt{TRUE} if the name of the procedure exists. Otherwise an exception handling is raised.

GetParamNum is \textit{reentrant} and processed \textit{without} parallelization.

Possible Predecessors

GetKeywords, SearchOperator, GetOperatorName, GetOperatorInfo

Possible Successors

GetParamTypes

See also

GetParamNames, GetParamTypes, GetOperatorName

Alternatives

GetOperatorInfo, GetParamInfo

Image / region / XLD management

\begin{verbatim}
[out] VARIANT InpCtrlParType HInfoX.GetParamTypes ([in] String ProcName, [out] VARIANT OutpCtrlParType )
void HOperatorSetX.GetParamTypes ([in] VARIANT ProcName, [out] VARIANT InpCtrlParType, [out] VARIANT OutpCtrlParType )
\end{verbatim}

Get default data type for the control parameters of a HALCON-procedure.

The operator \texttt{GetParamTypes} returns the default data type for each input and output control parameter. The default type of a parameter is the type used in \textquote{\texttt{\textit{simple mode}}} in HALCON/C. This concerns parameters which allow more than one type as for example \texttt{WriteString}. Hereby the types of input parameters are combined in the variable \texttt{InpCtrlParType}, whereas the types of output parameters are combined in the variable \texttt{OutpCtrlParType}. The following types are possible:

\begin{itemize}
    \item \texttt{integer}': an integer.
    \item \texttt{integer tuple}': an integer or a tuple of integers.
    \item \texttt{real}': a floating point number.
    \item \texttt{real tuple}': a floating point number or a tuple of floating point numbers.
    \item \texttt{string}': a string.
    \item \texttt{string tuple}': a string or a tuple of strings.
    \item \texttt{no default}': individual value of which the type cannot be determined.
    \item \texttt{no default tuple}': individual value or tuple of values of which the type cannot be determined.
    \item \texttt{default}': individual value of unknown type, whereby the systems assumes it to be an \texttt{integer}'.
\end{itemize}

Parameter

\begin{itemize}
    \item \texttt{ProcName} (input control) \texttt{String / VARIANT}
        \texttt{Name of the procedure.}
    \item \texttt{InpCtrlParType} (output control) \texttt{VARIANT ( string )}
        \texttt{Default type of the input control parameters.}
    \item \texttt{OutpCtrlParType} (output control) \texttt{VARIANT ( string )}
        \texttt{Default type of the output control parameters.}
\end{itemize}

Parallelization Information

\texttt{GetParamTypes} is \textit{reentrant} and processed \textit{without} parallelization.
CHAPTER 12. SYSTEM

See also GetParamNames, GetParamNum, GetOperatorInfo, GetOperatorName

Alternatives

GetParamInfo

Module

Image / region / XLD management

[out] VARIANT Slots HInfoX.QueryOperatorInfo ( )
void HOperatorSetX.QueryOperatorInfo ([out] VARIANT Slots )

Query slots concerning information with relation to the operator GetOperatorInfo.
The operator QueryOperatorInfo returns the names of those online texts (Slots) which are available online for each procedure. The information itself can be called up using

GetOperatorInfo(<Procedurname>,<Parametername>,<Slot>:<Information>).

Attention

Parameter

 Slots (output control) ................. string ~ VARIANT( string )
Slotnames of the operator GetOperatorInfo.

Result

The operator QueryOperatorInfo always returns the value TRUE.

Parallelization Information

QueryOperatorInfo is local and processed completely exclusively without parallelization.

Possible Successors

GetOperatorInfo

See also

GetOperatorInfo

Module

Image / region / XLD management

[out] VARIANT Slots HInfoX.QueryParamInfo ( )
void HOperatorSetX.QueryParamInfo ([out] VARIANT Slots )

Query slots of the online-information concerning the operator GetParamInfo.
The operator QueryParamInfo returns the names of those pieces of information (Slots) which are available online for each parameter (online texts). The online texts themselves can be called up using

GetParamInfo(<Procedurname>,<Parametername>,<Slot>:<Information>).

Attention

Parameter

 Slots (output control) ................. string ~ VARIANT( string )
Slotnames for the operator GetParamInfo.

Result

QueryParamInfo always returns the value TRUE.

Parallelization Information

QueryParamInfo is reentrant, local, and processed without parallelization.

Possible Successors

GetParamInfo

HALCON/COM Reference Manual, 2005-2-1
Search names of all procedures assigned to one keyword.

The operator **SearchOperator** returns the names of all procedures whose online-texts include the keyword **Keyword** (see also **GetOperatorInfo**). All available keywords are called by using the operator **GetKeywords (::: '': <keywords>)**. The online-texts are taken from the files english.hlp, english.sta, english.key, english.num and Halcon.idx, which are searched by HALCON in the currently used directory or the directory 'help_dir' (see also **GetSystem** and **GetSystem**).

**Attention**

**Parameter**

- **Keyword** (input control) .................................................. string ~ String / VARIANT
  
  Keyword for which corresponding procedures are searched.

  **Default Value**: 'Information'

- **ProcNames** (output control) .......................................... string ~ VARIANT( string )
  
  Procedures whose slot 'keyword' contains the keyword.

**Result**

The operator **SearchOperator** returns the value TRUE if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

**Parallelization Information**

**SearchOperator** is processed completely exclusively without parallelization.

**Possible Predecessors**

**GetKeywords**

**See also**

**GetKeywords, GetOperatorInfo, GetParamInfo**

Image / region / XLD management

### 12.4 Operating-System

Elapsed processing time since the last call of **CountSeconds**.

The operator **CountSeconds** helps to measure the runtime. The first call of the procedure normally returns a zero. Each further call returns the processing time which has elapsed in the meantime in seconds. The method by which the time is measured can be set with **SetSystem('clockMode',...)**.

**Attention**

The time measurement is not exact and depends on the load of the computer.

**Parameter**

- **Seconds** (output control) .......................................... real ~ double / VARIANT
  
  Processtime since the program start.
Example

```c
count_seconds(::Start) > /* program segment to be measured */
count_seconds(::End) >
eval(::End - Start:RunTime).
```

**Result**
The operator `CountSeconds` always returns the value TRUE.

**Parallelization Information**
`CountSeconds` is reentrant and processed without parallelization.

**See also**
`SetSystem` Module

**System**

```c
#include "halcon_hal.h"

void HSystemX.SystemCall ([in] String Command )
void HOperatorSetX.SystemCall ([in] VARIANT Command )
```

Executes a system command.

The operator `SystemCall` executes the system command specified by the string pointed to by `Command` (C-procedure "’system’"). If the string is empty, an interactive shell will be started (’csh -i’).

**Attention**

**Parameter**

► `Command` (input control) ..........................string  ~ String / VARIANT
Command to be called by the system.

**Default Value**: ‘ls’

**Result**
If the entered operator can be executed by the system, the operator `SystemCall` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**
`SystemCall` is reentrant and processed without parallelization.

**Possible Predecessors**
`CountSeconds`

**See also**
`WaitSeconds`

**Module**

```c
#include "halcon_hal.h"

void HSystemX.WaitSeconds ([in] double Seconds )
void HOperatorSetX.WaitSeconds ([in] VARIANT Seconds )
```

Delaying the execution of the program.

The operator `WaitSeconds` delays the execution by the number of seconds indicated in `Seconds`.

**Attention**
### 12.5. Parallelization

**Parameter**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Real double / VARIANT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td></td>
</tr>
</tbody>
</table>

- **Seconds** (input control)
  - Number of seconds by which the execution of the program will be delayed.
  - **Default Value**: 10
  - **Restriction**: \((\text{Seconds} \geq 0)\)

**Result**

The operator `WaitSeconds` always returns the value `TRUE`.

**Parallelization Information**

`WaitSeconds` is *reentrant* and processed *without* parallelization.

**Possible Successors**

- `SystemCall`
- See also

**Module**

- `System`

#### 12.5 Parallelization

```cpp
void HSystemX.CheckParHwPotential ([in] long AllInpPars )
void HOperatorSetX.CheckParHwPotential ([in] VARIANT AllInpPars )
```

**CheckParHwPotential** is necessary for an efficient automatic parallelization, which is used by HALCON to better utilize multiprocessor hardware in order to speed up the processing of operators. As the parallelization of operators is done automatically, there is no need for the user to explicitly prepare or change programs for their parallelization. Thus, all HALCON-based programs can be used unchanged on multiprocessor hardware and nevertheless utilize the potential of parallel hardware. **CheckParHwPotential** checks a given hardware with respect to a parallel processing of HALCON operators. At this, it examines every operator, which can be sped up in principle by an automatic parallelization. Each examined operator is processed several times - both sequentially and in parallel - with a changing set of input parameter values/images. The latter helps to evaluate dependencies between an operator’s input parameter characteristics (e.g. the size of an input image) and the efficiency of its parallel processing. At this, AllInpPars is used in the following way: In the normal case, i.e. if `AllInpPars` contains the default value 0 (“false”), only those input parameters are examined which are supposed to show influence on the processing time. Other parameters are not examined so that the whole process is sped up. However, in some rare cases, the internal implementation of a HALCON operator might change from one HALCON release to another. Then, a parameter which did not show any direct influence on the processing time in former releases, may now show such an influence. In this case it is necessary to set `AllInpPars` to 1 (“true”) in order to force the examination of all input parameters. If this happens, the HALCON release notes will most likely contain an appropriate note about this fact. Overall, **CheckParHwPotential** performs several test loops and collects a lot of hardware-specific informations, which enable HALCON to optimize the automatic parallelization for a given hardware. The hardware information is stored so that it can be used again in future HALCON sessions. Thus, it is sufficient, to start **CheckParHwPotential** once on each multiprocessor machine that is used for parallel processing. Of course, it should be started again, if the hardware of the machine changes, for example, by installing a new cpu, or if the operating system of the machine changes, or if the machine gets a new host name. The latter is necessary, because HALCON identifies the machine-specific parallelization information by the machine’s host name. If the same multiprocessor machine is used with different operating systems, such as Windows and Linux, it is necessary to start **CheckParHwPotential** once for each operating system in order to correctly measure the rather strong influence of the operating system on the potential of exploiting multiprocessor hardware. Under Windows, HALCON stores the parallelization knowledge, which belongs to a specific machine, in the machine’s registry. At this, it uses a machine-specific registry key, which can be used by different users simultaneously. In the normal case, this key can be written or changed by any user under Windows NT. However, under Windows 2000 the key may only be changed by users with administrator privileges or by users which at least belong to the “power user” group. For all other users **CheckParHwPotential** shows no effect (but does not return an error). Under Linux/UNIX the parallelization information is stored in a file in the HALCON installation directory (HALCONROOT). Again this means that **CheckParHwPotential** must be called by users with
the appropriate privileges, here by users which have write access to the HALCON directory. If HALCON is used within a network under Linux/UNIX, the denoted file contains the information about every computer in the network for which the hardware check has been successfully completed.

Attention

During its test loops CheckParHwPotential has to start every examined operator several times. Thus, the processing of CheckParHwPotential can take rather a long time. CheckParHwPotential bases on the automatic parallelization of operators which is exclusively supported by Parallel HALCON. Thus, CheckParHwPotential always returns an appropriate error, if it used with a non-parallel HALCON version. CheckParHwPotential must be called by users with the appropriate privileges for storing the parallelization information permanently (see the operator’s description above for more details about this subject).

Parameter

\[ \text{AllInpPars} \]

(input control) ……………………………………………………….integer / long / VARIANT
Check every input parameter?

Default Value : 0

List of values : AllInpPars \in \{0, 1\}

Result

CheckParHwPotential returns TRUE if all parameters are correct.

Parallelization Information

CheckParHwPotential is \textit{local} and processed \textit{completely exclusively} without parallelization.

Possible Successors

StoreParKnowledge

See also

StoreParKnowledge, LoadParKnowledge

Module

System

\begin{verbatim}
void HSystemX.LoadParKnowledge ([in] String FileName )
void HOperatorSetX.LoadParKnowledge ([in] VARIANT FileName )
\end{verbatim}

Load knowledge about automatic parallelization from file.

LoadParKnowledge supports the automatic parallelization of HALCON operators, which is used to better utilize multiprocessor hardware in order to speed up the processing of operators. To parallelize the processing of operators automatically HALCON needs some specific knowledge about the used hardware. This hardware-specific knowledge can be obtained by using the operator CheckParHwPotential. In the normal case, HALCON stores this knowledge in a specific file in the HALCON installation directory (Linux/UNIX) or within the “registry” (Windows). This enables HALCON to use the knowledge again later on. With LoadParKnowledge it is possible to load this knowledge explicitly from an ASCII file. At this, FileName denotes the name of this file (incl. path and file extension). The file must conform to a specific syntax and must have been stored beforehand by using StoreParKnowledge. While reading the file LoadParKnowledge checks whether its content was written for the currently used computer and whether the contained parallelization information regards the currently used HALCON version (and revision). If this is the case, LoadParKnowledge adopts the information so that it will also be used with further HALCON sessions. Otherwise, the information is ignored and LoadParKnowledge returns an appropriate error message.

Parameter

\[ \text{FileName} \]

(input control) …………………………………………………….filename.named / String / VARIANT
Name of parallelization knowledge file.

Default Value : ”

Result

LoadParKnowledge returns TRUE if all parameters are correct.

Parallelization Information

LoadParKnowledge is \textit{local} and processed \textit{completely exclusively} without parallelization.

Possible Predecessors

StoreParKnowledge

HALCON/COM Reference Manual, 2005-2-1
Store knowledge about automatic parallelization in file.

`StoreParKnowledge` supports the automatic parallelization of HALCON operators, which is used to better utilize multiprocessor hardware in order to speed up the processing of operators. To parallelize the processing of operators automatically HALCON needs some specific knowledge about the used hardware. This hardware-specific knowledge can be obtained by calling the operator `CheckParHwPotential`. There, HALCON stores the knowledge in a specific file in the HALCON installation directory (Linux/UNIX) or within the “registry” (Windows). This enables HALCON to use the knowledge again later on. With `StoreParKnowledge` it is possible to store this knowledge explicitly as an ASCII file. At this, `FileName` denotes the name of this file (incl. path and file extension). The stored knowledge can be read again later on by using `LoadParKnowledge`.

**Parameter**

- **FileName** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . filename.named  
  Name of parallelization knowledge file.

  Default Value : ”

**Result**

`StoreParKnowledge` returns TRUE if all parameters are correct.

**Parallelization Information**

`StoreParKnowledge` is local and processed completely exclusively without parallelization.

**Possible Predecessors**

- `CheckParHwPotential`

**Possible Successors**

- `LoadParKnowledge`

**See also**

- `LoadParKnowledge`, `CheckParHwPotential`

**Module**

System

### 12.6 Parameters

The operator `GetSystem` returns information concerning the currently activated HALCON system parameters. Some of these parameters can be changed dynamically by using the operator `SetSystem`. They are marked by a + in the list below. By passing the string ”?” as the parameter `Query`, the names of all system parameters are provided with `Information`.

The following system parameters can be queried:

**Versions**

- `version`: HALCON version number, e.g.: 6.0
- `last_update`: Date of creation of the HALCON library
- `revision`: Revision number of the HALCON library, e.g.: 1
Upper Limits

- **max_contour_length**: Maximum number of contour respectively polygone control points of a region.
- **max_images**: Maximum total of images.
- **max_channels**: Maximum number of channels of an image.
- **max_obj_per_par**: Maximum number of image objects which may be used during one call up per parameter.
- **max_inp_obj_par**: Maximum number of input parameters.
- **max_outp_obj_par**: Maximum number of output parameters.
- **max_inp_ctrl_par**: Maximum number of input control parameters.
- **max_outp_ctrl_par**: Maximum number of output control parameters.
- **max_window**: Maximum number of windows.
- **max_window_types**: Maximum number of window systems.
- **max_proc**: Maximum number of HALCON procedures (system defined + user defined).

Graphic

- **flush_graphic**: Determines, whether the flush operation is called or not after each HALCON procedure.
- **int2_bits**: Number of significant bits of int2 images. This number is used when scaling the gray values. If the values is -1 the gray values will be automatically scaled (default).
- **int_zooming**: Determines if the zooming of images is done with integer arithmetic or with floating point arithmetic.
- **draw_mode**: (no description available)
- **ignore_colormap**: (no description available)
- **backing_store**: Storage of the window contents in case of overlaps.
- **icon_name**: (no description available)
- **window_name**: (no description available)
- **default_font**: Name of the font to set at opening the window.
- **single_lut**: (no description available)
- **update_lut**: (no description available)
- **x_package**: Number of bytes which are sent to the X server during each transfer of data.
- **num_gray_4**: Number of colors reserved under X Window concerning the output of graylevels (DispChannel) on a machine with 4 bitplanes (16 colors).
- **num_gray_6**: Number of colors reserved under X Window concerning the output of graylevels (DispChannel) on a machine with 6 bitplanes (64 colors).
- **num_gray_8**: Number of colors reserved under X Window concerning the output of graylevels (DispChannel) on a machine with 8 bitplanes (256 colors).
- **num_gray_percentage**: HALCON reserves a certain amount of the available colors under X Window for the representation of graylevels (DispImage). This shall interfere with other X applications as little as possible. However, if HALCON does not succeed in reserving a minimum percentage of 'num_gray_percentage' of the necessary colors on the X server, a certain amount of the lookup-table will be claimed for the HALCON graylevels regardless of the consequences for other applications. This may result in undesired shifts of color when switching between HALCON windows and windows of other applications, or if (outside HALCON) a window-dump is generated. The number of the real graylevels to be reserved depends on the number of available bitplanes on the outputmachine (see also 'num_gray_x'). Naturally no colors will be reserved on monochrome machines - the graylevels will instead be dithered when displayed. If graylevel displays are used, only different shades of gray will be applied (‘black’, ‘white’, ‘gray’, etc.). Machines with 2 or 3 bitplanes will be considered monochrome machines, machines with 5 (7) bitplanes like machines with 4 (6) bitplanes, and machines having more than 8 bitplanes like machines with 8 bitplanes. A special case are machines providing a 24 bit display (true color machines). Naturally no colors are reserved for the display of graylevels in this case. Note: Before the first window on a machine with x bitplanes is opened, num_gray_x indicates the number of colors which have to be reserved for the display of graylevels, afterwards, however, it will indicate the number of colors which actually have been reserved.
- **num_graphic_percentage**: (no description available)
- **num_graphic_2**: Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 2 bitplanes (4 colors).
+`num_graphic_4`': Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 4 bitplanes (16 colors).

+`num_graphic_6`': Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 6 bitplanes (64 colors).

+`num_graphic_8`': Number of the HALCON graphic colors reserved under X Window (for DispRegion etc.) on a machine with 8 bitplanes (256 colors).

**Image Processing**

+`neighborhood`': Using the 4 or 8 neighborhood.

+`only_lines`': (no description available)

+`init_new_image`': Initialization of images before applying grayvalue transformations.

+`no_object_result`': Behavior for an empty object lists.

+`empty_region_result`': Reaction of procedures concerning input objects with empty regions which actually are not useful for such objects (e.g. certain region features, segmentation, etc.). Possible return values:
  
  - `true`: the error will be ignored if possible
  - `false`: the procedure returns FALSE
  - `fail`: the procedure returns FAIL
  - `void`: the procedure returns VOID
  - `exception`: an exception is raised

+`store_empty_region`': Storing of objects with empty regions.

+`clip_region`': Clipping of output regions so that they fit the global image size.

+`width`': Global maximum image width - in Standard-HALCON this value contains the maximum image width of all HALCON image objects which are currently stored in memory. In Parallel HALCON this value contains the maximum image width of all HALCON image objects which are or were in memory since the start of the current HALCON session (this also includes objects which may be deleted meanwhile).

+`height`': Global maximum image height - in Standard-HALCON this value contains the maximum image height of all HALCON image objects which are currently stored in memory. In Parallel HALCON this value contains the maximum image height of all HALCON image objects which are or were in memory since the start of the current HALCON session (this also includes objects which may be deleted meanwhile).

+`obj_images`': Current number of grayvalue components per image object.

+`current_runlength_number`': Currently used number of chords which can be used for the encoding of regions.

**Parallelization**

+`parallelize_operators`': Determines whether Parallel HALCON uses an automatic parallelization to speed up the processing of operators on multiprocessor machines.

+`reentrant`': Denotes whether Parallel HALCON currently supports reentrancy (default case), or whether this feature has been switched off. Reentrancy is necessary for the automatic parallelization of Parallel HALCON and for calling and processing multiple HALCON operators in parallel within multithreaded applications.

+`processor_num`': Returns the number of processors which Parallel HALCON has found on the hardware it is running on. This also indicates the number of processors which is used by Parallel HALCON for the automatic parallelization of operators.

**File**

+`flush_file`': Buffering of file output.

+`ocr_trainf_version`': This parameter returns the file format used for writing an OCR training file. The operators WriteOcrTrainf, WriteOcrTrainfImage and ConcatOcrTrainf write training data in ASCII format for version number 1 or in binary format for version number 2.

**Directories**

+`image_dir`': Path which will searched for the image file after the default directory (see also: ReadImage).

+`lut_dir`': Path for the default directory for color tables (see also: SetLut).
+"reference_dir": Path for the default directory concerning the postscript HALCON documentation.
+"help_dir": Path for the default help directory for the online help files:
   {german,english},{hlp,sta,idx,num,key}.

Other
+"do_low_error": Flag, if low level error should be printed.
'num_proc': Total number of the available HALCON procedures ('num_sys_proc' + 'num_user_proc').
'num_sys_proc': Number of the system procedures (supported procedures).
'num_user_proc': Number of the user defined procedures (see also 'Extension Packages' manual).
'byte_order': Byte order of the processor ('msb_first' or 'lsb_first').
'operating_system': Name of the operating system of the computer on which the HALCON process is being executed.
'operating_system_version': Version number of the operating system of the computer on which the HALCON process is being executed.
+"clock_mode": Method used for measuring the time in CountSeconds ('processor_time', 'elapsed_time', or 'performance_counter').
+"max_connection": Maximum number of regions returned by Connection.
+"alloca_temp_single_block": Flag for kind of temporary memory management.
+"alloca_max_blocksize": Maximum size of memory blocks to be allocated within temporary memory management. (No effect, if 'alloca_max_blocksize' == -1 or 'alloca_temp_single_block' == 'true')
+"extern_alloc_funct": Pointer to external function for memory allocation of result images.
+"extern_free_funct": Pointer to external function for memory deallocation of result images.
"temp_mem": Amount of temporary memory used by the last operator. This feature is only supported by Standard-HALCON. Instead of this, Parallel HALCON returns the amount of temporary memory used by GetSystem itself.
+"language": Language used for error messages ('english' or 'german').

\[ Attention \]
\[ Parameter \]

▷ Query (input control) ........................................ string(-array) \rightsquigarrow\mbox{VARIANT}(\mbox{string})
Desired system parameter.

Default Value: 'width'
List of values: Query \in \{'?', 'version', 'last_update', 'revision', 'max_images', 'max_channels', 'max_obj_per_par', 'max_max_obj_per_par', 'max_max_max_obj_per_par', 'max_max_max_max_obj_per_par', 'max_window', 'max_window_types', 'max_proc', 'flush_graphics', 'int2_bits', 'int_zooming', 'draw_mode', 'ignore_colormap', 'backing_store', 'icon_name', 'window_name', 'default_font', 'single_lut', 'update_lut', 'x_package', 'num_gray4', 'num_gray6', 'num_gray8', 'num_gray_percentage', 'num_graphic2', 'num_graphic4', 'num_graphic8', 'num_graphic_percentage', 'neighborhood', 'only_lines', 'init_new_image', 'no_object_result', 'empty_region_result', 'store_empty_region', 'clip_region', 'width', 'height', 'obj_images', 'current_runlength_number', 'flush_file', 'ocr_trainf_version', 'image_dir', 'lut_dir', 'reference_dir', 'help_dir', 'num_proc', 'num_sys_proc', 'num_user_proc', 'temp_mem', 'alloca_temp_single_block', 'alloca_max_blocksize', 'extern_alloc_funct', 'extern_free_funct', 'do_low_error', 'reentrant', 'parallelize_operators', 'processor_num', 'clock_mode', 'max_connection', 'byte_order', 'operating_system', 'operating_system_version', 'language'}

▷ Information (output control) ......................... integer(-array) \rightsquigarrow\mbox{VARIANT}(\mbox{integer, real, string})
Current value of the system parameter.

\[ Result \]
The operator GetSystem returns the value TRUE if the parameters are correct. Otherwise an exception is raised.

\[ Parallelization Information \]
GetSystem is local and processed completely exclusively without parallelization.

\[ Possible Predecessors \]
ResetObjDb

\[ Possible Successors \]
SetSystem

HALCON/COM Reference Manual, 2005-2-1
12.6. PARAMETERS

See also

SetSystem

Module

System

void HSystemX.SetSystem ([in] VARIANT Systemparameter, [in] VARIANT Value )

void HOperatorSetX.SetSystem ([in] VARIANT Systemparameter, [in] VARIANT Value )

Setting of HALCON system parameters.

The operator SetSystem allows to change different system parameters with relation to the runlength.

Available system parameters:

'neighborhood': This parameter is used with all procedures which examine neighborhood relations: Connection, GetRegionContour, GetRegionChain, GetRegionPolygon, GetRegionThickness, Boundary, PaintRegion, DispRegion, FillUp, Contlength, ShapeHistoAll.

Value: 4 or 8

default: 8

'default_font': Whenever a window is opened, a font will be set for the text output, whereby the 'default_font' will be used. If the preset font cannot be found, another font name can be set before opening the window.

Value: Filename of the fonts

default: fixed

'single_lut': (no description available) default: 'false'

'update_lut' Determines whether the HALCON color tables are adapted according to their environment or not.

Value: 'true' or 'false'

default: 'false'

'image_dir': Image files (e.g. ReadImage and ReadSequence) will be looked for in the currently used directory and in 'image_dir' (if no absolute paths are indicated). More than one directory name can be indicated (searchpaths), seperated by semicolons (Windows NT) or colons (Unix). The path can also be determined using the environment variable HALCONIMAGES.

Value: Name of the filepath

default: '$HALCONROOT/images' bzw. '%HALCONROOT%/images'

'lut_dir': Color tables (SetLut) which are realized as an ASCII-file will be looked for in the currently used directory and in 'lut_dir' (if no absolute paths are indicated). If HALCONROOT is set, HALCON will search the color tables in the sub-directory ''lut''.

Value: Name of the filepath

default: '$HALCONROOT/lut' bzw. '%HALCONROOT%/lut'

'reference_dir': The HTML and postscript sources of the HALCON documentation (especially the HALCON manual itself) will be looked for in the currently used directory or in 'reference_dir'. This system parameter is necessary for example using the operator DispInfo. This parameter can also be set by the environment variable HALCONROOT before initializing HALCON. In this case the variable must indicate the directory above the help directories (that is the HALCON-Homedirectory): e.g.: '/usr/local/halcon'

Value: Name of the filepath

default: '$HALCONROOT/doc/ps/reference/c' bzw. '%HALCONROOT%/doc/ps/reference/c'

'help_dir': The online text files german or english.hlp, .sta, .key, .num and .idx will be looked for in the currently used directory or in 'help_dir'. This system parameter is necessary for instance using the operators GetOperatorInfo and GetParamInfo. This parameter can also be set by the environment variable HALCONROOT before initializing HALCON. In this case the variable must indicate the directory above the helpdirectories (that is the HALCON-Homedirectory): e.g.: '/usr/local/halcon'

Value: Name of the filepath

default: '$HALCONROOT/help' bzw. '%HALCONROOT%/help'

'only_lines': (no description available) default: 'true'

HALCON 6.1.4
'init_new_image': Determines whether new images shall be set to 0 before using filters. This is not necessary if always the whole image is filtered or if the data of not filtered image areas are unimportant.
Value: 'true' or 'false'
default: 'true'

'no_object_result': Determines how operations processing iconic objects shall react if the object tuplet is empty (= no objects). Available values for Value:
'true': the error will be ignored
'false': the procedure returns FALSE
'fail': the procedure returns FAIL
'void': the procedure returns VOID
'exception': an exception is raised
default: 'true'

'empty_region_result': Controls the reaction of procedures concerning input objects with empty regions which actually are not useful for such objects (e.g. certain region features, segmentation, etc.). Available values for Value:
'true': the error will be ignored if possible
'false': the procedure returns FALSE
'fail': the procedure returns FAIL
'void': the procedure returns VOID
'exception': an exception is raised
default: 'true'

'store_empty_region': Quite a number of operations will lead to the creation of objects with an empty region (= no image points) (e.g. Intersection, Threshold, etc.). This parameter determines whether the object with an empty region will be returned as a result ('true') or whether it will be ignored ('false') that is no result will be returned.
Value: 'true' or 'false'
default: 'true'

'backing_store': Determines whether the window content will be refreshed in case of overlapping of the windows. Some implementations of X Window are faulty; in order to avoid these errors, the storing of contents can be deactivated. It may be recommendable in some cases to deactivate the security mechanism, if e.g. performance / memory is what matters.
Value: true or false
default: true

'flush_graphic': After each HALCON procedure which creates a graphic output, a flush operation will be executed in order to display the data immediately on screen. This is not necessary with all programs (e.g. if everything is done with the help of the mouse). In this case 'flush_graphic' can be set to 'false' to improve the runlength.
Value: 'true' or 'false'
default: 'true'

'flush_file': This parameter determines whether the output into a file (also to the terminal) shall be buffered or not. If the output is to be buffered, in general the data will be displayed on the terminal only after entering the operator FnewLine.
Value: 'true' or 'false'
default: 'true'

'ocr_trainf_version': This parameter determines the format that is used for writing an OCR training file. The operators WriteOcrTrainf, WriteOcrTrainfImage and ConcatOcrTrainf write training data in ASCII format for version number 1 or in binary format for version number 2. The binary version is faster in reading and writing data and stores training files more packed. The ASCII format is compatible to older HALCON releases.
Value: 1, 2
default: 2

'x_package': The output of image data via the network may cause errors owing to the heavy load on the computer or on the network. In order to avoid this, the data are transmitted in small packages. If the computer is used locally, these units can be enlarged at will. This can lead to a notably improved output performance.
Value: package size (in bytes)
default: 20480
'int2_bits': Number of significant bits of int2 images. This number is used when scaling the gray values. If the values is -1 the gray values will be automatically scaled (default).

**Value**: -1 or 9..15
**default**: -1

'num_gray_4': Number of colors to be reserved under X Window to allow the output of graylevels disp_channel on a machine with 4 bitplanes (16 colors).

Attention! This value may only be changed before the first window has been opened on the machine.

**Value**: 2 - 12
**default**: 8

'num_gray_6': Number of colors to be reserved under X Window to allow the output of graylevels disp_channel on a machine with 6 bitplanes (64 colors).

Attention! This value may only be changed before the first window has been opened on the machine.

**Value**: 2 - 62
**default**: 50

'num_gray_8': Number of colors to be reserved under X Window to allow the output of graylevels disp_channel on a machine with 8 bitplanes (256 colors).

Attention! This value may only be changed before the first window has been opened on the machine.

**Value**: 2 - 254
**default**: 140

'num_gray_percentage': Under X Window HALCON reserves a part of the available colors for the representation of gray values (DispChannel). This shall interfere with other X applications as little as possible. However, if HALCON does not succeed in reserving a minimum percentage of 'num_gray_percentage' of the necessary colors on the X server, a certain amount of the lookup table will be claimed for the HALCON graylevels regardless of the consequences. This may result in undesired shifts of color when switching between HALCON windows and windows of other applications, or (outside HALCON) if a window-dump is generated. The number of the real graylevels to be reserved depends on the number of available bitplanes on the outputmachine (see also 'num_gray_x'). Naturally no colors will be reserved on monochrome machines - the graylevels will instead be dithered when displayed. If graylevel-displays are used, only different shades of gray will be applied ('black', 'white', 'gray', etc.). Machines with 2 or 3 bitplanes will be considered monochrome machines, machines with 5 (7) bitplanes like machines with 4 (6) bitplanes, and machines having more than 8 bitplanes like machines with 8 bitplanes. A special case are machines providing a 24 bit display (true color machines). Naturally no colors are reserved for the display of graylevels in this case.

Note: This value may only be changed before the first window has been opened on the machine. For before opening the first window on a machine with x bitplanes, num_gray_x indicates the number of colors which have to be reserved for the display of graylevels, afterwards, however, it will indicate the number of colors which actually have been reserved.

**Value**: 0 - 100
**default**: 30

'num_graphic_percentage': (no description available) default: 60

'draw_mode': (no description available) default: 'complement'

'int_zooming': (no description available) default: 'true'

'ignore_colormap': (no description available) default: 'false'

'icon_name': (no description available) default: 'default'

'num_graphic_2': Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators DispRegion etc.) on a machine with 2 bitplanes (4 colors).

Attention: This value may only be changed before the first window has been opened on the machine.

**Value**: 0 - 2
**default**: 2

'num_graphic_4': Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators DispRegion etc.) on a machine with 4 bitplanes (16 colors).

Attention: This value may only be changed before the first window has been opened on the machine.

**Value**: 0 - 14
**default**: 5

'num_graphic_6': Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators DispRegion etc.) on a machine with 6 bitplanes (64 colors).
Attention: This value may only be changed before the first window has been opened on the machine.

**Value:** 0 - 62
Default: 10

**'num_graphic_colors':** Number of the graphic colors to be reserved by HALCON under X Window (concerning the operators `DispRegion` etc.) on a machine with 8 bitplanes (256 colors).

Attention: This value may only be changed before the first window has been opened on the machine.

**Value:** 0 - 64
Default: 20

HALCON reserves the first `num_graphic_colors` colors form this list of color names as graphic colors.

As a default HALCON uses this same list which is also returned by using `QueryAllColors`. However, the list can be changed individually: hereby a tuplet of color names will be returned as value. It is recommendable that such a tuplet always includes the colors 'black' and 'white', and optionally also 'red', 'green' and 'blue'. If 'default' is set as Value, HALCON returns to the initial setting. Note: On graylevel machines not the first x colors will be reserved, but the first x shades of gray from the list.

Attention: This value may only be changed before the first window has been opened on the machine.

**Value:** Tuplets of X Window color names
Default: see also `QueryAllColors`.

**'current_runlength_number':** Regions will be stored internally in a certain runlengthcode. This parameter can determine the maximum number of chords which may be used for representing a region. Please note that some procedures raise the number on their own if necessary.

The value can be enlarged as well as reduced.

**Value:** maximum number of chords
Default: 50000

**'clip_region':** Determines whether the regions of iconic objects of the HALCON database will be clipped to the currently used image size or not. This is the case for example in procedures like `GenCircle`, `GenRectangle1` or `Dilation1`.

See also: `ResetObjDb`

**Value:** 'true' or 'false'
Default: 'true'

**'do_low_error':** Determines whether the HALCON should print low level error or not.

**Value:** 'true' or 'false'
Default: 'false'

**'reentrant':** Determines whether HALCON must be reentrant for being used within a parallel programming environment (e.g. a multithreaded application). This parameter is only of importance for Parallel HALCON, which can process several operators concurrently. Thus, the parameter is ignored by the sequentially working HALCON-Version. If it is set to 'true', Parallel HALCON internally uses synchronization mechanisms to protect shared data objects from concurrent accesses. Though this is inevitable with any effectively parallel working application, it may cause undesired overhead, if used within an application which works purely sequentially. The latter case can be signalled by setting 'reentrant' to 'false'. This switches off all internal synchronization mechanisms and thus reduces overhead. Of course, Parallel HALCON then is no longer thread-safe, which causes another side-effect: Parallel HALCON will then no longer use the internal parallelization of operators, because this needs reentrancy. Setting 'reentrant' to 'true' resets Parallel HALCON to its default state, i.e. it is reentrant (and thread-safe) and it uses the automatic parallelization to speed up the processing of operators on multiprocessor machines.

**Value:** 'true' or 'false'
Default: Parallel HALCON: 'true', otherwise: 'false'

**'parallelize_operators':** Determines whether Parallel HALCON uses an automatic parallelization to speed up the processing of operators on multiprocessor machines. This feature can be switched off by setting 'parallelize_operators' to 'false'. Even then Parallel HALCON will remain reentrant (and thread-safe), unless the parameter 'reentrant' is changed via `SetSystem` accordingly. Changing 'parallelize_operators' can be helpful, for example, if HALCON-operators are called by a multithreaded application, which also does the scheduling and load-balancing of operators and data by itself. Then it may be undesired that HALCON performs additional parallelization steps, which may disturb the application’s scheduling and load-balancing concepts. The parameter 'parallelize_operators' is only supported by Parallel HALCON and thus ignored by the sequentially working HALCON-Version.

**Value:** 'true' or 'false'
Default: Parallel HALCON: 'true', otherwise: 'false'
'clock_mode' Determines the method used for measuring the current time in CountSeconds. For Value='processor_time', the time used by the HALCON process is measured. For Value='elapsed_time', the true elapsed time is measured. For Value='performance_counter', under Windows the time is measured by using a performance counter, which results in a much higher accuracy than the system clock, which is used for Value='processor_time'. On Unix systems, the modes 'processor_time' and 'performance_counter' are identical, because no performance counters are available on Unix systems.

'max_connection' Determines the maximum number of regions returned by Connection. For Value=0, all regions are returned.

'alloctmp_single_block' Kind of temporary memory management. Value: 'true' or 'false' default: 'false'

'alloctmp_max_blocksize' Maximum size of memory blocks to be allocated within temporary memory management. (No effect, if 'alloctmp_max_blocksize' == -1 or 'alloctmp_single_block' == 'true') Value: -1 or >= 0 default: -1

'extern_alloc_func' Pointer to external function for memory allocation of result images. default: 0

'extern_free_func' Pointer to external function for memory deallocation of result images. default: 0

'language' Language used for error messages. Value: 'english' or 'german'. default: 'english'

Parameter

Systemparameter (input control) ... string(-array) ~ VARIANT ( string )
Name of the system parameter to be changed.
Default Value : 'image_dir'

Value (input control) ... string(-array) ~ VARIANT ( integer, real, string )
New value of the system parameter.
Default Value : 'true'
Suggested values : Value ∈ {'true', 'false', 0, 4, 8, 100, 140, 255}

The operator SetSystem returns the value TRUE if the parameters are correct. Otherwise an exception will be raised.

Parallelization Information
SetSystem is local and processed completely exclusively without parallelization.

Possible Predecessors
ResetObjDb, GetSystem, SetCheck
See also

GetSystem, SetCheck, CountSeconds

Module

System

12.7 Serial

void HSerialX.ClearSerial ([in] String Channel )

void HOperatorSetX.ClearSerial ([in] VARIANT SerialHandle, [in] VARIANT Channel )

Clear the buffer of a serial connection.
ClearSerial discards data written to the serial device referred to by SerialHandle, but not transmitted (Channel = 'output'), or data received, but not read (Channel = 'input'), or performs both these operations at once (Channel = 'in_out').

Parameter

▷ SerialHandle (input control) serial_id ~ HSerialX / VARIANT Serial interface handle.
▷ Channel (input control) string ~ String / VARIANT Buffer to be cleared.
   Default Value: 'input'
   List of values: Channel ∈ {'input', 'output', 'in_out'}

Result

If the parameters are correct and the buffers of the serial device could be cleared, the operator ClearSerial returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

ClearSerial is reentrant and processed without parallelization.

Possible Predecessors

OpenSerial

Possible Successors

ReadSerial, WriteSerial

See also

ReadSerial

Module

System

void HMiscX.CloseAllSerials ( )
void HOperatorSetX.CloseAllSerials ( )

Close all serial devices.

CloseAllSerials closes all serial devices that have been opened with OpenSerial.

Result

CloseAllSerials returns always TRUE.

Parallelization Information

CloseAllSerials is reentrant and processed without parallelization.

Possible Predecessors

OpenSerial

See also

OpenSerial, CloseFile

Alternatives

CloseSerial

Module

System

void HOperatorSetX.CloseSerial ([in] VARIANT SerialHandle )

Close a serial device.

CloseSerial closes a serial device that was opened with OpenSerial.
12.7. SERIAL

Parameter

- **SerialHandle** (input control) ........................................... serial_id ~> HSerialX / VARIANT
  Serial interface handle.

Result

If the parameters are correct and the device could be closed, the operator **CloseSerial** returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

**CloseSerial** is reentrant and processed without parallelization.

Possible Predecessors

OpenSerial

See also

OpenSerial, CloseFile

Module

System

Get the parameters of a serial device.

**GetSerialParam** returns the current parameter settings of the serial device passed in **SerialHandle**. For a description of the parameters of a serial device, see **SetSerialParam**.

Parameter

- **SerialHandle** (input control) ........................................... serial_id ~> HSerialX / VARIANT
  Serial interface handle.
- **BaudRate** (output control) .............................................. integer ~> long / VARIANT
  Speed of the serial interface.
- **DataBits** (output control) ............................................. integer ~> long / VARIANT
  Number of data bits of the serial interface.
- **FlowControl** (output control) ........................................... string ~> String / VARIANT
  Type of flow control of the serial interface.
- **Parity** (output control) ................................................ string ~> String / VARIANT
  Parity of the serial interface.
- **StopBits** (output control) .............................................. integer ~> long / VARIANT
  Number of stop bits of the serial interface.
- **TotalTimeOut** (output control) ...................................... integer ~> long / VARIANT
  Total timeout of the serial interface in ms.
- **InterCharTimeOut** (output control) ................................. integer ~> long / VARIANT
  Inter-character timeout of the serial interface in ms.

Result

If the parameters are correct and the parameters of the device could be read, the operator **GetSerialParam** returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

**GetSerialParam** is reentrant and processed without parallelization.

Possible Predecessors

OpenSerial

Possible Successors

GetSerialParam, ReadSerial, WriteSerial
CHAPTER 12. SYSTEM

SetSerialParam

Module System

void HSerialX.OpenSerial ([in] String PortName )

void HOoperatorSetX.OpenSerial ([in] VARIANT PortName, [out] VARIANT SerialHandle )

Open a serial device.

OpenSerial opens a serial device. The name of the device is determined by the parameter PortName and is operating system specific. On Windows NT machines, ‘COM1’-‘COM4’ is typically used, while on Unix systems the serial devices usually are named ‘/dev/tty*’. The parameters of the serial device, e.g., its speed or number of data bits, are set to the system default values for the respective device after the device has been opened. They can be set or changed by calling SetSerialParam.

Parameter

▷ PortName (input control) ........................................ filename.named String / VARIANT

Name of the serial port.

Default Value : ‘COM1’


▷ SerialHandle (output control) ......................... serial_id ~ HSerialX / VARIANT

Serial interface handle.

Result

If the parameters are correct and the device could be opened, the operator OpenSerial returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

OpenSerial is reentrant and processed without parallelization.

Possible Successors

SetSerialParam, ReadSerial, WriteSerial, CloseSerial

See also

SetSerialParam, GetSerialParam, OpenFile

Module System

[out] VARIANT Data HSerialX.ReadSerial ([in] long NumCharacters )


Read from a serial device.

ReadSerial tries to read NumCharacters from the serial device given in SerialHandle. The read characters are returned in Data as a tuple of integers. This allows to read NUL characters, which would otherwise be interpreted as the end of a string. If the timeout of the serial device has been set to a value greater than 0 with SetSerialParam, ReadSerial waits at most as long for the arrival of the first character as indicated by the timeout. Otherwise, the operator returns immediately. In any case, the number of characters available at the time of return are passed back to the caller, i.e., fewer characters than requested can be returned. This can be checked by the length of the tuple Data.

Parameter

▷ SerialHandle (input control) .............................. serial_id ~ HSerialX / VARIANT

Serial interface handle.
12.7. SERIAL

- **NumCharacters** (input control) ........................................................ integer \(\sim\) long / VARIANT
  Number of characters to read.
  **Default Value:** 1
  **Suggested values:** NumCharacters \(\in\) \{1, 2, 3, 4, 5, 10, 20, 40, 100\}

- **Data** (output control) ........................................................ integer(-array) \(\sim\) VARIANT( integer )
  Read characters (as tuple of integers).

---

**Result**

If the parameters are correct and the read from the device was successful, the operator **ReadSerial** returns the value TRUE. Otherwise an exception is raised.

---

**Parallelization Information**

**ReadSerial** is reentrant and processed without parallelization.

---

**Possible Predecessors**

- **OpenSerial**
- **WriteSerial**

---

**System**

```c
void HSerialX.SetSerialParam ([in] VARIANT BaudRate,
[in] VARIANT StopBits, [in] VARIANT TotalTimeOut,
[in] VARIANT InterCharTimeOut )

void HOperatorSetX.SetSerialParam ([in] VARIANT SerialHandle,
[in] VARIANT BaudRate, [in] VARIANT DataBits, [in] VARIANT FlowControl,
[in] VARIANT InterCharTimeOut )
```

Set the parameters of a serial device.

**SetSerialParam** can be used to set the parameters of a serial device. The parameter **BaudRate** determines the input and output speed of the device. It should be noted that not all devices support all possible speeds. The number of sent and received data bits is set with **DataBits**. The parameter **FlowControl** determines if and what kind of data flow control should be used. In the latter case, a choice between software control (‘xon_xoff’) and hardware control (‘cts_rts’, ‘dtr_dsr’) can be made. If and what kind of parity check of the transmitted data should be performed can be determined by **Parity**. The number of stop bits sent is set with **StopBits**. Finally, two timeout for reading from the serial device can be set. The parameter **TotalTimeOut** determines the maximum time, which may pass in **ReadSerial** until the first character arrives, independent of the actual number of characters requested. The parameter **InterCharTimeOut** determines the time which may pass between the reading of individual characters, if multiple characters are requested with **ReadSerial**. If one of the timeouts is set to -1, a read waits an arbitrary amount of time for the arrival of characters. Thus, on Windows NT systems, a total timeout of **TotalTimeOut + nInterCharTimeOut** results if n characters are to be read. On Unix systems, only one of the two timeouts can be set. Thus, if both timeouts are passed larger than -1, only the total timeout is used. The unit of both timeouts is milliseconds. For each parameter, the current values can be left in effect by passing ‘unchanged’.

---

- **SerialHandle** (input control) .................................................... serial_id \(\sim\) HSerialX / VARIANT
  Serial interface handle.

- **BaudRate** (input control) .................................................... integer \(\sim\) VARIANT( integer, string )
  Speed of the serial interface.
  **Default Value:** ‘unchanged’
  **List of values:** BaudRate \(\in\) \{50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200,
  38400, 57600, 76800, 115200, 153600, 230400, 307200, 460800, ‘unchanged’\}
DataBits (input control) ............................................ integer \(\sim\) VARIANT (integer, string)
Number of data bits of the serial interface.
Default Value: ‘unchanged’
List of values: DataBits \(\in\) \{5, 6, 7, 8, ‘unchanged’\}

FlowControl (input control) ........................................ string \(\sim\) String / VARIANT
Type of flow control of the serial interface.
Default Value: ‘unchanged’

Parity (input control) .............................................. string \(\sim\) String / VARIANT
Parity of the serial interface.
Default Value: ‘unchanged’
List of values: Parity \(\in\) \{‘none’, ‘odd’, ‘even’, ‘unchanged’\}

StopBits (input control) ............................................. integer \(\sim\) VARIANT (integer, string)
Number of stop bits of the serial interface.
Default Value: ‘unchanged’
List of values: StopBits \(\in\) \{1, 2, ‘unchanged’\}

TotalTimeOut (input control) .................................... integer \(\sim\) VARIANT (integer, string)
Total timeout of the serial interface in ms.
Default Value: ‘unchanged’
Suggested values: TotalTimeOut \(\in\) \{-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, ‘unchanged’\}

InterCharTimeOut (input control) ................................. integer \(\sim\) VARIANT (integer, string)
Inter-character timeout of the serial interface in ms.
Default Value: ‘unchanged’
Suggested values: InterCharTimeOut \(\in\) \{-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, ‘unchanged’\}

Result

If the parameters are correct and the parameters of the device could be set, the operator SetSerialParam returns the value TRUE. Otherwise an exception is raised.

Parallelization Information

SetSerialParam is reentrant and processed without parallelization.

Possible Predecessors
OpenSerial, GetSerialParam

Possible Successors
ReadSerial, WriteSerial

See also
GetSerialParam

Module
System

void HSerialX.WriteSerial ([in] VARIANT Data )
void HOperatorSetX.WriteSerial ([in] VARIANT SerialHandle, [in] VARIANT Data )

Write to a serial connection.
WriteSerial writes the characters given in Data to the serial device given by SerialHandle. The data to be written is passed as a tuple of integers. This allows to write NUL characters, which would otherwise be interpreted as the end of a string. WriteSerial always waits until all data has been transmitted, i.e., a timeout for writing cannot be set.

Parameter

SerialHandle (input control) ........................................ serial_id \(\sim\) HSerialX / VARIANT
Serial interface handle.
12.8. SOCKETS

Data (input control) .................................................. integer(-array) \texttt{\sim} \texttt{VARIANT ( integer )}
Characters to write (as tuple of integers).

Result

If the parameters are correct and the write to the device was successful, the operator \texttt{WriteSerial} returns the
value TRUE. Otherwise an exception is raised.

Parallelization Information
\texttt{WriteSerial} is reentrant and processed \textit{without} parallelization.

Possible Predecessors

OpenSerial

See also

ReadSerial

Module

12.8 Sockets

\begin{verbatim}
void \texttt{HOperatorSetX.CloseSocket ([in] \texttt{VARIANT Socket })}
\end{verbatim}

Close a socket.

\texttt{CloseSocket} closes a socket that was previously opened with \texttt{OpenSocketAccept}, \texttt{OpenSocketConnect}, or \texttt{SocketAcceptConnect}. For a detailed example, see \texttt{OpenSocketAccept}.

Parameter

\begin{verbatim}
\texttt{Socket (input control) ..................................................socket_id \texttt{\sim} \texttt{HSocketX / \texttt{VARIANT}}}
\end{verbatim}

\texttt{CloseSocket} is reentrant and processed \textit{without} parallelization.

See also

OpenSocketAccept, OpenSocketConnect, SocketAcceptConnect

Module

\begin{verbatim}
void \texttt{HOperatorSetX.GetNextSocketDataType ([in] \texttt{VARIANT Socket, [out] \texttt{VARIANT DataType })}
\end{verbatim}

Determine the HALCON data type of the next socket data.

\texttt{GetNextSocketDataType} returns the data type of the next data that are present on the socket \texttt{Socket} and returns it in \texttt{DataType}. The possible values for \texttt{DataType} are:

\begin{itemize}
  \item \texttt{no\_data}': No data are present.
  \item \texttt{no\_halcon\_data}': Some data are present, but they are not HALCON data.
  \item \texttt{tuple}': The next data is a tuple.
  \item \texttt{region}': The next data is a region object.
  \item \texttt{image}': The next data is an image object.
  \item \texttt{xld\_cont}': The next data is an XLD contour.
  \item \texttt{xld\_poly}': The next data is an XLD polygon.
  \item \texttt{xld\_para}': The next data is an XLD parallel.
  \item \texttt{xld\_mod\_para}': The next data is a modified XLD parallel.
  \item \texttt{xld\_ext\_para}': The next data is an extended XLD parallel.
\end{itemize}
CHAPTER 12. SYSTEM

Parameter

- **Socket** (input control) .......................... socket id  \( \sim \) HSocketX / VARIANT
  Socket number.
- **DataType** (output control) .......................... string  \( \sim \) String / VARIANT
  Data type of next HALCON data.

Parallelization Information

GetNextSocketDataType is reentrant and processed without parallelization.

See also
SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple

Module

```
[out] VARIANT Timeout HSocketX.GetSocketTimeout ( )
```

Get the timeout of a socket.

GetSocketTimeout returns the timeout for the socket connection that is passed in Socket. For a description of possible values of Timeout see SetSocketTimeout.

Parameter

- **Socket** (input control) .......................... socket id  \( \sim \) HSocketX / VARIANT
  Socket number.
- **Timeout** (output control) .......................... number  \( \sim \) VARIANT ( real, string )
  Socket timeout.

Parallelization Information

GetSocketTimeout is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketAccept, OpenSocketConnect, SocketAcceptConnect

See also

SetSocketTimeout

Module

```
void HSocketX.OpenSocketAccept ([in] long Port )
```

Open a socket that accepts connection requests.

OpenSocketAccept opens a socket that accepts incoming connection requests by other HALCON processes. This operator is the necessary first step in the establishment of a communication channel between two HALCON processes. The socket listens for incoming connection requests on the port number given by Port. The accepting socket is returned in AcceptingSocket. OpenSocketAccept returns immediately without waiting for a request from another process, done by calling OpenSocketConnect in the other process. This allows multiple other processes to connect to the particular HALCON process that calls OpenSocketAccept. To accept an incoming connection request, SocketAcceptConnect must be called after another process has called OpenSocketConnect.
12.8. SOCKETS

Parameter

- **Port** (input control) ..................................................... integer ~ long / VARIANT
  Port number.
  **Default Value:** 3000
  **Typical range of values:** 1024 ≤ Port ≤ 1024
  **Minimum Increment:** 1
  **Recommended Increment:** 1
- **AcceptingSocket** (output control) ............................... socket_id ~ HSocketX / VARIANT
  Socket number.

Example

```c
/* Process 1 */
dev_set_colored (12)
open_socket_accept (3000, AcceptingSocket)
/* Busy wait for an incoming connection */
dev_error_var (Error, 1)
dev_set_check (‘˜give_error’)
OpenStatus := 5
while (OpenStatus # 2)
    socket_accept_connect (AcceptingSocket, ‘false’, Socket)
    OpenStatus := Error
    wait_seconds (0.2)
endwhile
dev_set_check (‘give_error’)  
/* Connection established */
receive_image (Image, Socket)
threshold (Image, Region, 0, 63)
send_region (Region, Socket)
area_center (ConnectedRegions, Area, Row, Column)
send_tuple (Socket, Area)
send_tuple (Socket, Row)
send_tuple (Socket, Column)
close_socket (Socket)
close_socket (AcceptingSocket)

/* Process 2 */
dev_set_colored (12)
open_socket_connect (‘localhost’, 3000, Socket)
read_image (Image, ‘fabrik’)  
send_image (Image, Socket)
receive_region (Region, Socket)
connection (Region, ConnectedRegions)
send_region (ConnectedRegions, Socket)
receive_tuple (Socket, Area)
receive_tuple (Socket, Row)
receive_tuple (Socket, Column)
close_socket (Socket)
```

Parallelization Information

**OpenSocketAccept** is reentrant and processed without parallelization.

Possible Successors

See also

OpenSocketConnect, CloseSocket, SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple
CHAPTER 12. SYSTEM

Module System

void HSocketX.OpenSocketConnect ([in] String HostName, [in] long Port )
void HOperatorSetX.OpenSocketConnect ([in] VARIANT HostName, [in] VARIANT Port, [out] VARIANT Socket )

Open a socket to an existing socket.

OpenSocketConnect opens a connection to an accepting socket on the computer HostName, which listens on port Port. The listening socket in the other HALCON process must have been created earlier with the operator OpenSocketAccept. The socket thus created is returned in Socket. To establish the connection, the HALCON process, in which the accepting socket resides, must call SocketAcceptConnect. For a detailed example, see OpenSocketAccept.

Parameter

downarrow HostName (input control) ........................................ string \( \sim \) String / VARIANT
Hostname of the computer to connect to.

Default Value: 'localhost'
downarrow Port (input control) ............................................ integer \( \sim \) long / VARIANT
Port number.

Default Value: 3000
Typical range of values: \( 1024 \leq \text{Port} \leq 1024 \)
Minimum Increment: 1
Recommended Increment: 1
downarrow Socket (output control) ........................................... socket_id \( \sim \) HSocketX / VARIANT
Socket number.

Parallelization Information

OpenSocketConnect is reentrant and processed without parallelization.

Possible Successors
SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple

See also
OpenSocketAccept, SocketAcceptConnect, CloseSocket

Module System

void HImageX.ReceiveImage ([in] long Socket )
[out] HImageX Image HSocketX.ReceiveImage ()

Receive an image over a socket connection.

ReceiveImage reads an image object that was sent over the socket connection determined by Socket by another HALCON process using the operator SendImage. If no image has been sent, the HALCON process calling ReceiveImage blocks until enough data arrives. For a detailed example, see OpenSocketAccept.

Parameter

downarrow Image (output iconic) image(-array) \( \sim \) HImageX / HUntypedObjectX (byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf)
Received image.
downarrow Socket (input control) socket_id \( \sim \) long / HSocketX / VARIANT
Socket number.
12.8. SOCKETS

ReceiveImage is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

See also

SendImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System

void HRegionX.ReceiveRegion ([in] long Socket )

[out] HRegionX Region HSocketX.ReceiveRegion ( )

void HOperatorSetX.ReceiveRegion ([out] HUntypedObjectX Region,
[in] VARIANT Socket )

Receive regions over a socket connection.

ReceiveRegion reads a region object that was sent over the socket connection determined by Socket by another HALCON process using the operator SendRegion. If no regions have been sent, the HALCON process calling ReceiveRegion blocks until enough data arrives. For a detailed example, see OpenSocketAccept.

Parameter

▷ Region (output iconic) ................................. region(-array) \mapsto HRegionX / HUntypedObjectX

Received regions.

▷ Socket (input control) ............................... socket_id \mapsto long / HSocketX / VARIANT

Socket number.

Parallelization Information

ReceiveRegion is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

See also

SendRegion, SendImage, ReceiveImage, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System

[out] String Tuple HSocketX.ReceiveTuple ( )

void HOperatorSetX.ReceiveTuple ([in] VARIANT Socket,
[out] VARIANT Tuple )

Receive a tuple over a socket connection.

ReceiveTuple reads a tuple that was sent over the socket connection determined by Socket by another HALCON process using the operator SendTuple. If no tuple has been sent, the HALCON process calling ReceiveTuple blocks until enough data arrives. For a detailed example, see OpenSocketAccept.

Parameter

▷ Socket (input control) ............................... socket_id \mapsto HSocketX / VARIANT

Socket number.

▷ Tuple (output control) .............................. string \mapsto String / VARIANT

Received tuple.

Parallelization Information

ReceiveTuple is reentrant and processed without parallelization.
CHAPTER 12. SYSTEM

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

See also

SendTuple, SendImage, ReceiveImage, SendRegion, ReceiveRegion, GetNextSocketDataType

Module

System

```
void HXLDX.ReceiveXld ([in] long Socket )
[out] HUntypedObjectX XLD HSocketX.ReceiveXld ( )
void HOperatorSetX.ReceiveXld ([out] HUntypedObjectX XLD, [in] VARIANT Socket )
```

Receive an XLD object over a socket connection.

ReceiveXld reads an XLD object that was sent over the socket connection determined by Socket by another HALCON process using the operator SendXld. If no XLD object has been sent, the HALCON process calling ReceiveXld blocks until enough data arrives. For a detailed example, see SendXld.

Parameter

- **XLD (output iconic)** ........................................... xld(-array) ~ HUntypedObjectX
  Received XLD object.
- **Socket (input control)** ..................................... socket_id ~ long / HSocketX / VARIANT
  Socket number.

Parallelization Information

ReceiveXld is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

SendXld, SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System

```
void HImageX.SendImage ([in] long Socket )
void HSocketX.SendImage ([in] HImageX Image )
void HOperatorSetX.SendImage ([in] IObjectX Image, [in] VARIANT Socket )
```

Send an image over a socket connection.

SendImage sends an image object over the socket connection determined by Socket. The receiving HALCON process must call ReceiveImage to read the image from the socket. For a detailed example, see OpenSocketAccept.

Parameter

- **Image (input iconic)** ........................................... image(-array) ~ HImageX / IObjectX ( byte, direction, cyclic, int1, int2, uint2, int4, real, complex, dvf )
  Image to be sent.
- **Socket (input control)** ..................................... socket_id ~ long / HSocketX / VARIANT
  Socket number.

Parallelization Information

SendImage is reentrant and processed without parallelization.
12.8. SOCKETS

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

See also

ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HRegionX.SendRegion</td>
<td>([in] long Socket)</td>
</tr>
<tr>
<td>HSocketX.SendRegion</td>
<td>([in] HRegionX Region)</td>
</tr>
<tr>
<td>HOperatorSetX.SendRegion</td>
<td>([in] IHObjectX Region, [in] VARIANT Socket )</td>
</tr>
</tbody>
</table>

Send regions over a socket connection.

SendRegion sends a region object over the socket connection determined by Socket. The receiving HALCON process must call ReceiveRegion to read the regions from the socket. For a detailed example, see OpenSocketAccept.

Parameter

- **Region** (input iconic)  
  region-array  
  Regions to be sent.

- **Socket** (input control)  
  socket-id  
  Socket number.

Parallelization Information

SendRegion is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

See also

ReceiveRegion, SendImage, ReceiveImage, SendTuple, ReceiveTuple, GetNextSocketDataType

Module

System

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSocketX.SendTuple</td>
<td>([in] String Tuple)</td>
</tr>
<tr>
<td>HOperatorSetX.SendTuple</td>
<td>([in] VARIANT Socket, [in] VARIANT Tuple )</td>
</tr>
</tbody>
</table>

Send a tuple over a socket connection.

SendTuple sends a tuple over the socket connection determined by Socket. The receiving HALCON process must call ReceiveTuple to read the tuple from the socket. For a detailed example, see OpenSocketAccept.

Parameter

- **Socket** (input control)  
  socket-id  
  Socket number.

- **Tuple** (input control)  
  string  
  Tuple to be sent.

Parallelization Information

SendTuple is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketConnect, SocketAcceptConnect

See also

ReceiveTuple, SendImage, ReceiveImage, SendRegion, ReceiveRegion, GetNextSocketDataType

HALCON 6.1.4
Send an XLD object over a socket connection.

SendXld sends an XLD object over the socket connection determined by Socket. The receiving HALCON process must call ReceiveXld to read the XLD object from the socket.

**Parameter**

- **XLD**(input iconic) ... xld(-array) \(\sim\) IHXLDX / IHObjectX
  - XLD object to be sent.
- **Socket**(input control) ... socket \(\sim\) long / HSocketX / VARIANT
  - Socket number.

**Example**

```c
/* Process 1 */
dev_set_colored (12)
open_socket_accept (3000, AcceptingSocket)
socket_accept_connect (AcceptingSocket, ‘true’, Socket)
receive_image (Image, Socket)
edges_sub_pix (Image, Edges, ‘canny’, 1.5, 20, 40)
send_xld (Edges, Socket)
receive_xld (Polygons, Socket)
split_contours_xld (Polygons, Contours, ‘polygon’, 1, 5)
gen_parallels_xld (Polygons, Parallels, 10, 30, 0.15, ‘true’)
send_xld (Parallels, Socket)
receive_xld (ModParallels, Socket)
send_xld (ExtParallels, Socket)
stop ()
close_socket (Socket)
close_socket (AcceptingSocket)

/* Process 2 */
dev_set_colored (12)
open_socket_connect (‘localhost’, 3000, Socket)
read_image (Image, ‘mreut’) 
send_image (Image, Socket)
receive_xld (Edges, Socket)
gen_polygons_xld (Edges, Polygons, ‘ramer’, 2)
send_xld (Polygons, Socket)
split_contours_xld (Polygons, Contours, ‘polygon’, 1, 5)
receive_xld (Parallels, Socket)
mod_parallels_xld (Parallels, Image, ModParallels, ExtParallels,
  0.4, 160, 220, 10)
send_xld (ModParallels, Socket)
send_xld (ExtParallels, Socket)
stop ()
close_socket (Socket)
```

**Parallelization Information**

SendXld is reentrant and processed without parallelization.
Set the timeout of a socket.

SetSocketTimeout sets the timeout for the socket connection that is passed in `Socket`. The `Timeout` is only used for reading data from the socket and for calls to `SocketAcceptConnect`. If problems during the transmission of the data cause a timeout, the underlying protocol cannot synchronize itself with the data any longer. Therefore, in these cases, the only possibility to put the system into a consistent state is to close both sockets and to open them anew. It should be noted that sometimes while reading data no error message will be returned if the sending socket is closed while the receiving socket is waiting for data. In these cases, empty data are returned (either objects or tuples).

The timeout is given in seconds as a floating point number. It can also be set to `infinite`, causing the read calls to wait indefinitely.

**Parameter**

- `Socket` (input control) socket number.
- `Timeout` (input control) number

**Default Value:** `infinite`

**Suggested values:** `Timeout ∈ {‘infinite’, 0, 1, 2, 3, 4, 5, 10, 30, 60}`

**Parallelization Information**

SetSocketTimeout is *reentrant* and processed without parallelization.

**Possible Predecessors**

OpenSocketAccept, OpenSocketConnect, SocketAcceptConnect

**Possible Successors**

SocketAcceptConnect, ReceiveImage, ReceiveRegion, ReceiveXld

**See also**

GetSocketTimeout

**Module**

System

**Possible Successors**

OpenSocketConnect, SocketAcceptConnect

**See also**

ReceiveXld, SendImage, ReceiveImage, ReceiveRegion, SendTuple, ReceiveTuple, GetNextSocketDataType

**Module**

System
SocketAcceptConnect is another socket Socket, which is used for a two-way communication with another HALCON process. After this connection has been established, data can be exchanged between the two processes by calling the appropriate send or receive operators. For a detailed example, see OpenSocketAccept.

Parameter

- **AcceptingSocket** (input control) socket_id ~ HSocketX / VARIANT
  Socket number of the accepting socket.
- **Wait** (input control) wait ~ String / VARIANT
  Should the operator wait until a connection request arrives?
  List of values: \text{Wait} \in \{ "true", "false" \}
- **Socket** (output control) socket_id ~ long / VARIANT
  Socket number.

Parallelization Information

SocketAcceptConnect is reentrant and processed without parallelization.

Possible Predecessors

OpenSocketAccept

Possible Successors

SendImage, ReceiveImage, SendRegion, ReceiveRegion, SendTuple, ReceiveTuple

See also

OpenSocketConnect, CloseSocket

Module

System
Chapter 13

Tools

13.1 Affine-Transformations

Apply an arbitrary affine 2D transformation to points.

AffineTransPoint2D applies an arbitrary affine 2D transformation, i.e., scaling, rotation, translation, and slant (skewing), to the input points \((Px, Py)\) and returns the resulting points in \((Qx, Qy)\). The affine transformation is described by the homogeneous transformation matrix given in HomMat2D. This corresponds to the following equation (input and output points as homogeneous vectors):

\[
\begin{pmatrix}
Qx \\
Qy \\
1
\end{pmatrix} = HomMat2D \cdot 
\begin{pmatrix}
Px \\
Py \\
1
\end{pmatrix}
\]

If the points to transform are specified in standard image coordinates, their row coordinates must be passed in \(Px\) and their column coordinates in \(Py\). This is necessary to obtain a right-handed coordinate system for the image. In particular, this assures that rotations are performed in the correct direction. Note that the \((x,y)\) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

The transformation matrix can be created using the operators HomMat2dIdentity, HomMat2dRotate, HomMat2dTranslate, etc., or can be the result of operators like VectorAngleToRigid.

For example, if HomMat2D corresponds to a rigid transformation, i.e., if it consists of a rotation and a translation, the points are transformed as follows:

\[
\begin{pmatrix}
Qx \\
Qy \\
1
\end{pmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} Px \\
Py \\
1
\end{pmatrix} = \begin{bmatrix} R \cdot (Px) + t \\
Py \\
1
\end{pmatrix}
\]

Parameter

- HomMat2D (input control) ....................... affine2d \(\Rightarrow HHomMat2dX / \text{VARIANT} (\text{real})\)
  Input transformation matrix.
  Number of elements : 6

- Px (input control) ................................. point.x(-array) \(\Rightarrow \text{VARIANT} (\text{real}, \text{integer})\)
  Input point(s) (x or row coordinate).
  Default Value : 64
  Suggested values : \(Px \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
CHAPTER 13. TOOLS

Py (input control) \[\text{point.y(-array)} \sim \text{VARIANT (real, integer)}\]
Input point(s) (y or column coordinate).
Default Value: 64
Suggested values: \(\text{Py} \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

Qx (output control) \[\text{point.x(-array)} \sim \text{VARIANT (real)}\]
Output point(s) (x or row coordinate).

Qy (output control) \[\text{point.y(-array)} \sim \text{VARIANT (real)}\]
Output point(s) (y or column coordinate).

Result
If the parameters are valid, the operator \texttt{AffineTransPoint2D} returns TRUE. If necessary, an exception is raised.

Parallelization Information
\texttt{AffineTransPoint2D} is \textit{reentrant} and processed \textit{without} parallelization.

Possible Predecessors
\texttt{HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant}

Possible Successors
\texttt{HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant}

Apply an arbitrary affine 3D transformation to points.

\texttt{HomMat3D} applies an arbitrary affine 3D transformation, i.e., scaling, rotation, and translation, to the input points \((\text{Px,Py,Pz})\) and returns the resulting points in \((\text{Qx,Qy,Qz})\). The affine transformation is described by the homogeneous transformation matrix given in \texttt{HomMat3D}. This corresponds to the following equation (input and output points as homogeneous vectors):

\[
\begin{bmatrix}
\text{Qx} \\
\text{Qy} \\
\text{Qz} \\
1
\end{bmatrix}
= \text{HomMat3D} \cdot
\begin{bmatrix}
\text{Px} \\
\text{Py} \\
\text{Pz} \\
1
\end{bmatrix}
\]

The transformation matrix can be created using the operators \texttt{HomMat3dIdentity, HomMat3dScale, HomMat3dRotate, HomMat3dTranslate}, etc., or be the result of \texttt{PoseToHomMat3d}.

For example, if \texttt{HomMat3D} corresponds to a rigid transformation, i.e., if it consists of a rotation and a translation, the points are transformed as follows:

\[
\begin{bmatrix}
\text{Qx} \\
\text{Qy} \\
\text{Qz} \\
1
\end{bmatrix}
= \begin{bmatrix}
\text{R} & \text{t} \\
0 & 0 & 0 & 1
\end{bmatrix}
\cdot
\begin{bmatrix}
\text{Px} \\
\text{Py} \\
\text{Pz} \\
1
\end{bmatrix}
= \begin{bmatrix}
\text{R} \cdot \text{Px} + \text{t} \\
\text{R} \cdot \text{Py} + \text{t} \\
\text{R} \cdot \text{Pz} + \text{t} \\
1
\end{bmatrix}
\]

Parameter

\texttt{HomMat3D} (input control) \[\text{affine3d} \sim \text{HHomMat3dX / VARIANT (real)}\]
Input transformation matrix.
Number of elements: 12

\texttt{Px} (input control) \[\text{point3d.x(-array)} \sim \text{VARIANT (real, integer)}\]
Input point(s) (x coordinate).
Default Value: 64
Suggested values: \(\text{Px} \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

HALCON/COM Reference Manual, 2005-2-1
13.1. AFFINE-TRANSFORMATIONS

Py (input control) ....................................... point3d.y(-array) \[\sim\] VARIANT (real, integer)
Input point(s) (y coordinate).
Default Value : 64
Suggested values : \(Py \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

Pz (input control) ....................................... point3d.z(-array) \[\sim\] VARIANT (real, integer)
Input point(s) (z coordinate).
Default Value : 64
Suggested values : \(Pz \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

Qx (output control) ....................................... point3d.x(-array) \[\sim\] VARIANT (real)
Output point(s) (x coordinate).

Qy (output control) ....................................... point3d.y(-array) \[\sim\] VARIANT (real)
Output point(s) (y coordinate).

Qz (output control) ....................................... point3d.z(-array) \[\sim\] VARIANT (real)
Output point(s) (z coordinate).

Result
If the parameters are valid, the operator AffineTransPoint3D returns TRUE. If necessary, an exception is raised.

Parallelization Information
AffineTransPoint3D is reentrant and processed without parallelization.

Possible Predecessors
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, PoseToHomMat3d

Possible Successors
HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, Project3DPoint

Basic operators

[out] VARIANT HomMat2D HImageX.DvfToHomMat2d ( )
void HHomMat2dX.DvfToHomMat2d ([in] HImageX VectorField )
void HOperatorSetX.DvfToHomMat2d ([in] IHObjectX VectorField, [out] VARIANT HomMat2D )

Approximate an affine map from a displacement vector field.
DvfToHomMat2d approximates an affine map from the displacement vector field VectorField. The affine map is returned in HomMat2D.

If the displacement vector field has been computed from the original image \(I_{\text{orig}}\) and the second image \(I_{\text{res}}\), the internally stored transformation matrix (see AffineTransImage) contains a map that describes how to transform the first image \(I_{\text{orig}}\) to the second image \(I_{\text{res}}\).

Parameter

VectorField (input iconic) ........................................ image \[\sim\] HImageX / IHObjectX (dvf)
Input image.

HomMat2D (output control) ................................... affine2d \[\sim\] VARIANT / HHomMat2dX (real)
Output transformation matrix.

Number of elements : 6

Parallelization Information
DvfToHomMat2d is reentrant and processed without parallelization.

Possible Predecessors
OpticalFlowMatch

Possible Successors
AffineTransImage

Alternatives
VectorToHomMat2d

HALCON 6.1.4
Multiply two homogeneous 2D transformation matrices.

HomMat2dCompose composes a new 2D transformation matrix by multiplying the two input matrices:

\[
\text{HomMat2dCompose} = \text{HomMat2dLeft} \cdot \text{HomMat2dRight}
\]

For example, if the two input matrices correspond to rigid transformations, i.e., to transformations consisting of a rotation and a translation, the resulting matrix is calculated as follows:

\[
\text{HomMat2dCompose} = \begin{bmatrix}
R_l & t_l \\
0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
R_r & t_r \\
0 & 1
\end{bmatrix} = \begin{bmatrix}
R_l \cdot R_r & R_l \cdot t_r + t_l \cdot t_r \\
0 & 1
\end{bmatrix}
\]

Parameter

- **HomMat2dLeft** (input control) affine2d \(\sim\) VARIANT( real )
  Left input transformation matrix.
  Number of elements : 6

- **HomMat2dRight** (input control) affine2d \(\sim\) HHomMat2dX / VARIANT( real )
  Right input transformation matrix.
  Number of elements : 6

- **HomMat2dCompose** (output control) affine2d \(\sim\) VARIANT( real )
  Output transformation matrix.
  Number of elements : 6

Result

If the parameters are valid, the operator HomMat2dCompose returns TRUE. If necessary, an exception is raised.

Parallelization Information

HomMat2dCompose is reentrant and processed without parallelization.

Possible Predecessors

HomMat2dCompose, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Possible Successors

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Basic operators

Generate the homogeneous transformation matrix of the identical 2D transformation.

HomMat2dIdentity generates the homogeneous transformation matrix HomMat2dIdentity describing the identical 2D transformation:

\[
\text{HomMat2dIdentity} = \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. Thus, `HomMat2DIdentity` is stored as the tuple [1,0,0,1,0].

**Parameter**

- `HomMat2DIdentity` (output control) affine2d ~ HHomMat2dX / VARIANT (real)
  - Transformation matrix.
  - Number of elements: 6

**Result**

`HomMat2DIdentity` always returns TRUE.

**Parallelization Information**

`HomMat2DIdentity` is reentrant and processed without parallelization.

**Possible Successors**

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

**Module**

**Basic operators**

```c
[out] VARIANT HomMat2DInvert HHomMat2dX.HomMat2dInvert ( )

void HOperatorSetX.HomMat2dInvert ([in] VARIANT HomMat2D, [out] VARIANT HomMat2DInvert )
```

Invert a homogeneous 2D transformation matrix.

`HomMat2dInvert` inverts the homogeneous 2D transformation matrix given by `HomMat2D`. The resulting matrix is returned in `HomMat2DInvert`.

**Parameter**

- `HomMat2D` (input control) affine2d ~ HHomMat2dX / VARIANT (real)
  - Input transformation matrix.
  - Number of elements: 6

- `HomMat2DInvert` (output control) affine2d ~ VARIANT (real)
  - Output transformation matrix.
  - Number of elements: 6

**Result**

`HomMat2DInvert` returns TRUE if the parameters are valid and the input matrix is invertible. Otherwise, an exception is raised.

**Parallelization Information**

`HomMat2DInvert` is reentrant and processed without parallelization.

**Possible Predecessors**

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

**Possible Successors**

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

**Module**

**Basic operators**

```c

```

Add a rotation to a homogeneous 2D transformation matrix.
**HomMat2dRotate** adds a rotation by the angle \( \Phi \) to the homogeneous 2D transformation matrix \( \text{HomMat2D} \) and returns the resulting matrix in \( \text{HomMat2dRotate} \). The rotation is described by a \( 2 \times 2 \) rotation matrix \( R \). It is performed relative to the global (i.e., fixed) coordinate system, which corresponds to the following chain of transformation matrices:

\[
\text{HomMat2dRotate} = \begin{bmatrix} R & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \text{HomMat2D} = \begin{bmatrix} \cos(\Phi) & -\sin(\Phi) \\ \sin(\Phi) & \cos(\Phi) \end{bmatrix}
\]

The point \((Px, Py)\) is the fixed point of the transformation, i.e., this point remains unchanged when transformed using \( \text{HomMat2dRotate} \). To obtain this behavior, first a translation is added to the input transformation matrix that moves the fixed point onto the origin of the global coordinate system. Then, the rotation is added, and finally a translation that moves the fixed point back to its original position. This corresponds to the following chain of transformations:

\[
\text{HomMat2dRotate} = \begin{bmatrix} 1 & 0 & +Px \\ 0 & 1 & +Py \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -Px \\ 0 & 1 & -Py \\ 0 & 0 & 1 \end{bmatrix} \cdot \text{HomMat2D}
\]

**Attention**

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix

\[
\begin{bmatrix} ra & rb & tc \\ rd & re & tf \\ 0 & 0 & 1 \end{bmatrix}
\]

is stored as the tuple \([ra, rb, tc, rd, re, tf]\).

---

**Parameter**

- **HomMat2D** (input control) .................. \( \text{affine2d} \) \( \rightsquigarrow \) \( \text{HHomMat2dX} \) / \( \text{VARIANT( real )} \)
  - Input transformation matrix.
  - **Number of elements**: 6
- **\( \Phi \)** (input control) .................. \( \text{angle.rad} \) \( \rightsquigarrow \) \( \text{VARIANT( real, integer )} \)
  - Rotation angle.
  - **Default Value**: 0.78
  - **Suggested values**: \( \Phi \in \{0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14\} \)
  - **Typical range of values**: \( 0 \leq \Phi \leq 0 \)
- **\( Px \)** (input control) .................. \( \text{point.x} \) \( \rightsquigarrow \) \( \text{VARIANT( real, integer )} \)
  - Fixed point of the transformation (x coordinate).
  - **Default Value**: 0
  - **Suggested values**: \( Px \in \{0, 16, 32, 64, 128, 256, 512, 1024\} \)
- **\( Py \)** (input control) .................. \( \text{point.y} \) \( \rightsquigarrow \) \( \text{VARIANT( real, integer )} \)
  - Fixed point of the transformation (y coordinate).
  - **Default Value**: 0
  - **Suggested values**: \( Py \in \{0, 16, 32, 64, 128, 256, 512, 1024\} \)
- **HomMat2dRotate** (output control) .................. \( \text{affine2d} \) \( \rightsquigarrow \) \( \text{VARIANT( real )} \)
  - Output transformation matrix.
  - **Number of elements**: 6

---

If the parameters are valid, the operator \( \text{HomMat2dRotate} \) returns TRUE. If necessary, an exception is raised.

**Parallelization Information**

**HomMat2dRotate** is reentrant and processed without parallelization.

**Possible Predecessors**

\( \text{HomMat2dIdentity}, \text{HomMat2dTranslate}, \text{HomMat2dScale}, \text{HomMat2dRotate}, \text{HomMat2dSlant} \)

**Possible Successors**

\( \text{HomMat2dTranslate}, \text{HomMat2dScale}, \text{HomMat2dRotate}, \text{HomMat2dSlant} \)

---

**Module**

Basic operators

HALCON/COM Reference Manual, 2005-2-1
Add a scaling to a homogeneous 2D transformation matrix.

`HomMat2dScale` adds a scaling by the scale factors `Sx` and `Sy` to the homogeneous 2D transformation matrix `HomMat2D` and returns the resulting matrix in `HomMat2DScale`. The scaling is described by a $2 \times 2$ scaling matrix $S$. It is performed relative to the global (i.e., fixed) coordinate system; this corresponds to the following chain of transformation matrices:

$$
\text{HomMat2DScale} = \begin{bmatrix}
S & 0 \\
0 & 0
\end{bmatrix} \cdot \text{HomMat2D}
$$

The point $(Px, Py)$ is the fixed point of the transformation, i.e., this point remains unchanged when transformed using `HomMat2DScale`. To obtain this behavior, first a translation is added to the input transformation matrix that moves the fixed point onto the origin of the global coordinate system. Then, the scaling is added, and finally a translation that moves the fixed point back to its original position. This corresponds to the following chain of transformations:

$$
\text{HomMat2DScale} = \begin{bmatrix}
1 & 0 +Px \\
0 & 1 +Py \\
0 & 0
\end{bmatrix} \cdot \begin{bmatrix}
S & 0 \\
0 & 0
\end{bmatrix} \cdot \begin{bmatrix}
1 & 0 -Px \\
0 & 1 -Py \\
0 & 0
\end{bmatrix} \cdot \text{HomMat2D}
$$

Attention

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix

$$
\begin{bmatrix}
ra & rb & tc \\
rd & re & tf \\
0 & 0 & 1
\end{bmatrix}
$$

is stored as the tuple $[ra, rb, tc, rd, re, tf]$.

---

Parameter

- `HomMat2D` (input control) ......................... affine2d $\mapsto \text{HHomMat2dX} / \text{VARIANT}$ (real)
  - Input transformation matrix.
  - Number of elements : 6
- `Sx` (input control) ................................. number $\mapsto \text{VARIANT}$ (real, integer)
  - Scale factor along the x-axis.
  - Default Value : 2
  - Suggested values : $Sx \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 16\}$
  - Restriction : $(Sx \neq 0)$
- `Sy` (input control) ................................. number $\mapsto \text{VARIANT}$ (real, integer)
  - Scale factor along the y-axis.
  - Default Value : 2
  - Suggested values : $Sy \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 16\}$
  - Restriction : $(Sy \neq 0)$
- `Px` (input control) ................................. point.x $\mapsto \text{VARIANT}$ (real, integer)
  - Fixed point of the transformation (x coordinate).
  - Default Value : 0
  - Suggested values : $Px \in \{0, 16, 32, 64, 128, 256, 512, 1024\}$
- `Py` (input control) ................................. point.y $\mapsto \text{VARIANT}$ (real, integer)
  - Fixed point of the transformation (y coordinate).
  - Default Value : 0
  - Suggested values : $Py \in \{0, 16, 32, 64, 128, 256, 512, 1024\}$
Add a slant to a homogeneous 2D transformation matrix.

`HomMat2dSlant` adds a slant by the angle `Theta` to the homogeneous 2D transformation matrix `HomMat2D` and returns the resulting matrix in `HomMat2DSlant`. A slant is an affine transformation in which one coordinate axis remains fixed, while the other coordinate axis is rotated counterclockwise by an angle `Theta`. The parameter `Axis` determines which coordinate axis is slanted. For `Axis = 'x'`, the x-axis is slanted and the y-axis remains fixed, while for `Axis = 'y'` the y-axis is slanted and the x-axis remains fixed. The slanting is performed relative to the global (i.e., fixed) coordinate system; this corresponds to the following chains of transformation matrices:

For `Axis = 'x'`:

\[
\text{HomMat2DSlant} = \begin{bmatrix}
\cos(\Theta) & 0 & 0 \\
\sin(\Theta) & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \text{HomMat2D}
\]

For `Axis = 'y'`:

\[
\text{HomMat2DSlant} = \begin{bmatrix}
1 & -\sin(\Theta) & 0 \\
0 & \cos(\Theta) & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \text{HomMat2D}
\]

The point `(Px, Py)` is the fixed point of the transformation, i.e., this point remains unchanged when transformed using `HomMat2DSlant`. To obtain this behavior, first a translation is added to the input transformation matrix that moves the fixed point onto the origin of the global coordinate system. Then, the slant is added, and finally a translation that moves the fixed point back to its original position. This corresponds to the following chain of transformations for `Axis = 'x'`:

\[
\text{HomMat2DSlant} = \begin{bmatrix}
1 & 0 & +Px \\
0 & 1 & +Py \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
\cos(\Theta) & 0 & 0 \\
\sin(\Theta) & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
1 & 0 & -Px \\
0 & 1 & -Py \\
0 & 0 & 1
\end{bmatrix} \cdot \text{HomMat2D}
\]

Attention

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix

\[
\begin{bmatrix}
ra & rb & tc \\
rd & re & tf \\
0 & 0 & 1
\end{bmatrix}
\]
is stored as the tuple \([ra, rb, tc, rd, re, tf]\).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HomMat2D (input control)</td>
<td>affine2d</td>
<td>Input transformation matrix.</td>
</tr>
<tr>
<td>Number of elements</td>
<td>: 6</td>
<td></td>
</tr>
<tr>
<td>Theta (input control)</td>
<td>angle.rad</td>
<td>Slant angle.</td>
</tr>
<tr>
<td>Default Value</td>
<td>: 0.78</td>
<td></td>
</tr>
<tr>
<td>Suggested values</td>
<td>(\Theta \in {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14})</td>
<td></td>
</tr>
<tr>
<td>Typical range of values</td>
<td>(0 \leq \Theta \leq 0)</td>
<td></td>
</tr>
<tr>
<td>Axis (input control)</td>
<td>string</td>
<td>Coordinate axis that is slanted.</td>
</tr>
<tr>
<td>Default Value</td>
<td>: 'x'</td>
<td></td>
</tr>
<tr>
<td>List of values</td>
<td>(Axis \in {'x', 'y'})</td>
<td></td>
</tr>
<tr>
<td>Px (input control)</td>
<td>point.x</td>
<td>Fixed point of the transformation (x coordinate).</td>
</tr>
<tr>
<td>Default Value</td>
<td>: 0</td>
<td></td>
</tr>
<tr>
<td>Suggested values</td>
<td>(Px \in {0, 16, 32, 64, 128, 256, 512, 1024})</td>
<td></td>
</tr>
<tr>
<td>Py (input control)</td>
<td>point.y</td>
<td>Fixed point of the transformation (y coordinate).</td>
</tr>
<tr>
<td>Default Value</td>
<td>: 0</td>
<td></td>
</tr>
<tr>
<td>Suggested values</td>
<td>(Py \in {0, 16, 32, 64, 128, 256, 512, 1024})</td>
<td></td>
</tr>
<tr>
<td>HomMat2dSlant (output control)</td>
<td>affine2d</td>
<td>Output transformation matrix.</td>
</tr>
<tr>
<td>Number of elements</td>
<td>: 6</td>
<td></td>
</tr>
</tbody>
</table>

Result

If the parameters are valid, the operator \texttt{HomMat2dSlant} returns TRUE. If necessary, an exception is raised.

Parallellization Information

\texttt{HomMat2dSlant} is reentrant and processed without parallelization.

Possible Predecessors

\texttt{HomMat2dIdentity, HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant}

Possible Successors

\texttt{HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant}

Basic operators

\begin{verbatim}
[out] double Sx HHomMat2dX.HomMat2dToAffinePar ([out] double Sy, [out] double Phi, [out] double Theta, [out] double Tx, [out] double Ty)
\end{verbatim}

Compute the affine transformation parameters from a homogeneous 2D transformation matrix.

\texttt{HomMat2dToAffinePar} computes the affine transformation parameters corresponding to the homogeneous 2D transformation matrix \texttt{HomMat2D}. The parameters \(Sx\) and \(Sy\) determine how the transformation scales the original x- and y-axes, respectively. The two scaling factors are always positive. The angle \(Theta\) describes whether the transformed coordinate axes are orthogonal (\(Theta = 0\)) or slanted. If \(|Theta| > \pi/2\), the transformation contains a reflection. The angle \(Phi\) determines the rotation of the transformed x-axis with respect to the original x-axis. The parameters \(Tx\) and \(Ty\) determine the translation of the two coordinate systems. The matrix \texttt{HomMat2D} can be constructed from the six transformation parameters by the following operator sequence:
This is equivalent to the following chain of transformation matrices:

\[
\begin{pmatrix}
1 & 0 & +T_x \\
0 & 1 & +T_y \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
\cos(\Phi) & -\sin(\Phi) & 0 \\
\sin(\Phi) & \cos(\Phi) & 0 \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
1 & -\sin(\Theta) & 0 \\
\cos(\Theta) & 0 & 0 \\
0 & 0 & 1
\end{pmatrix}
\cdot
\begin{pmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

Parameter

- **HomMat2D** (input control) ................. affine2d ~ HHomMat2dX / VARIANT (real)
  Input transformation matrix.
  **Number of elements:** 6

- **Sx** (output control) ....................... real ~ double / VARIANT
  Scaling factor along the x direction.

- **Sy** (output control) ....................... real ~ double / VARIANT
  Scaling factor along the y direction.

- **Phi** (output control) .................... angle.rad ~ double / VARIANT
  Rotation angle.

- **Theta** (output control) ................... angle.rad ~ double / VARIANT
  Slant angle.

- **Tx** (output control) ..................... point.x ~ double / VARIANT
  Translation along the x direction.

- **Ty** (output control) ..................... point.y ~ double / VARIANT
  Translation along the y direction.

Result

If the matrix **HomMat2D** is non-degenerate, **HomMat2dToAffinePar** returns TRUE. Otherwise, an exception is raised.

Parallelization Information

**HomMat2dToAffinePar** is reentrant and processed without parallelization.

Possible Predecessors

VectorToHomMat2d, VectorToRigid, VectorToSimilarity

Possible Successors

HomMat2dTranslate, HomMat2dScale, HomMat2dRotate, HomMat2dSlant

Module

Basic operators

Add a translation to a homogeneous 2D transformation matrix.

**HomMat2dTranslate** adds a translation by the vector \( t = (T_x, T_y) \) to the homogeneous 2D transformation matrix **HomMat2D** and returns the resulting matrix in **HomMat2dTranslate**. The translation is performed relative to the global (i.e., fixed) coordinate system; this corresponds to the following chain of transformation matrices:

\[
\text{HomMat2dTranslate} = \begin{bmatrix}
1 & 0 & t \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\cdot
\text{HomMat2D}
\]
Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix
\[
\begin{bmatrix}
ra & rb & tc \\
rd & re & tf \\
0 & 0 & 1
\end{bmatrix}
\]
is stored as the tuple \([ra, rb, tc, rd, re, tf]\).

**Parameter**

- **HomMat2D** (input control) .......................... affine2d \(\Rightarrow HHomMat2dX / \text{VARIANT( real )}

  * Input transformation matrix.
  * **Number of elements**: 6

- **Tx** (input control) ...................................... point.x \(\Rightarrow \text{VARIANT( real, integer )}

  * Translation along the x-axis.
  * **Default Value**: 64
  * **Suggested values**: \(Tx \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

- **Ty** (input control) ...................................... point.y \(\Rightarrow \text{VARIANT( real, integer )}

  * Translation along the y-axis.
  * **Default Value**: 64
  * **Suggested values**: \(Ty \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

- **HomMat2DTranslate** (output control) .......................... affine2d \(\Rightarrow \text{VARIANT( real )}

  * Output transformation matrix.
  * **Number of elements**: 6

If the parameters are valid, the operator **HomMat2dTranslate** returns TRUE. If necessary, an exception is raised.

**Parallelization Information**

**HomMat2dTranslate** is *reentrant* and processed *without* parallelization.

**Possible Predecessors**

- HomMat2dIdentity
- HomMat2dTranslate
- HomMat2dScale
- HomMat2dRotate
- HomMat2dSlant

**Possible Successors**

- HomMat2dTranslate
- HomMat2dScale
- HomMat2dRotate
- HomMat2dSlant

**Module**

```plaintext
[out] VARIANT HomMat3dCompose HHomMat3dX.HomMat3dCompose
(\[in\] VARIANT HomMat3dLeft )

```

**Multiply two homogeneous 3D transformation matrices.**

**HomMat3dCompose** composes a new 3D transformation matrix by multiplying the two input matrices:

\[
\text{HomMat3dCompose} = \text{HomMat3dLeft} \cdot \text{HomMat3dRight}
\]

For example, if the two input matrices correspond to rigid transformations, i.e., to transformations consisting of a rotation and a translation, the resulting matrix is calculated as follows:

\[
\text{HomMat3dCompose} = \begin{bmatrix}
R_l & t_l \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
R_r & t_r \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
R_l \cdot R_r & R_l \cdot t_r + t_l \\
0 & 0 & 1
\end{bmatrix}
\]
CHAPTER 13. TOOLS

Parameter

⊲ HomMat3DLeft (input control) .......................................................... affine3d $\sim$ VARIANT (real )
  Left input transformation matrix.
  Number of elements: 12

⊲ HomMat3DRight (input control) ................................. affine3d $\sim$ HHomMat3dX / VARIANT (real )
  Right input transformation matrix.
  Number of elements: 12

⊲ HomMat3DCompose (output control) .......................... affine3d $\sim$ VARIANT (real )
  Output transformation matrix.
  Number of elements: 12

Result

If the parameters are valid, the operator HomMat3dCompose returns TRUE. If necessary, an exception is raised.

Parallelization Information

HomMat3dCompose is reentrant and processed without parallelization.

Possible Predecessors

HomMat3dCompose, HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, PoseToHomMat3d

Possible Successors

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

See also

AffineTransPoint3D, HomMat3dIdentity, HomMat3dRotate, HomMat3dTranslate, PoseToHomMat3d, HomMat3dToPose

Module

Basic operators

void HHomMat3dX.HomMat3dIdentity ( )
void HOperatorSetX.HomMat3dIdentity ([out] VARIANT HomMat3DIdentity )

Generate the homogeneous transformation matrix of the identical 3D transformation.

HomMat3dIdentity generates the homogeneous transformation matrix HomMat3DIdentity describing the identical 3D transformation:

$$
\text{HomMat3DIdentity} = \\
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
$$

Attention

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. Thus, HomMat3DIdentity is stored as the tuple [1,0,0,0,1,0,0,0,0,1,0].

Parameter

⊲ HomMat3dIdentity (output control) .................. affine3d $\sim$ HHomMat3dX / VARIANT (real )
  Transformation matrix.
  Number of elements: 12

Result

HomMat3dIdentity always returns TRUE.

Parallelization Information

HomMat3dIdentity is reentrant and processed without parallelization.

Possible Successors

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate

HALCON/COM Reference Manual, 2005-2-1
13.1. AFFINE-TRANSFORMATIONS

PoseToHomMat3d

Basic operators

[out] VARIANT HomMat3DInvert HHomMat3dX.HomMat3dInvert ( )
void HOperatorSetX.HomMat3dInvert ([in] VARIANT HomMat3D,
[out] VARIANT HomMat3DInvert )

Invert a homogeneous 3D transformation matrix.
HomMat3DInvert inverts the homogeneous 3D transformation matrix given by HomMat3D. The resulting matrix is returned in HomMat3DInvert.

Parameter

▷ HomMat3D (input control) .................. affine3d ~ HHomMat3dX / VARIANT ( real )
   Input transformation matrix.
   Number of elements : 12
▷ HomMat3DInvert (output control) .................. affine3d ~ VARIANT ( real )
   Output transformation matrix.
   Number of elements : 12

Result

HomMat3DInvert returns TRUE if the parameters are valid and the input matrix is invertible. Otherwise, an exception is raised.

Parallelization Information

HomMat3DInvert is reentrant and processed without parallelization.

Possible Predecessors

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, PoseToHomMat3d

Possible Successors

HomMat3dTranslate, HomMat3dScale, HomMat3dRotate, HomMat3dToPose

See also

AffineTransPoint3D, HomMat3dIdentity, HomMat3dRotate, HomMat3dTranslate, PoseToHomMat3d, HomMat3dToPose, HomMat3dCompose

Basic operators

[out] VARIANT HomMat3DRotate HHomMat3dX.HomMat3dRotate
[in] VARIANT Pz )
void HOperatorSetX.HomMat3dRotate ([in] VARIANT HomMat3D,
in] VARIANT Pz, [out] VARIANT HomMat3dRotate )

Add a rotation to a homogeneous 3D transformation matrix.
HomMat3dRotate adds a rotation around the Axis-axis by the angle Phi to the homogeneous 3D transformation matrix HomMat3D and returns the resulting matrix in HomMat3dRotate. The rotation is described by a 3×3 rotation matrix R. It is performed relative to the global (i.e., fixed) coordinate system; this corresponds to the following chain of transformation matrices:

Axis = ‘x’:

\[
\begin{bmatrix}
R_x & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 1
\end{bmatrix}
\cdot HomMat3D
\]

\[
R_x = \begin{bmatrix}
1 & 0 & 0 \\
0 & \cos(\Phi) & -\sin(\Phi) \\
0 & \sin(\Phi) & \cos(\Phi)
\end{bmatrix}
\]

HALCON 6.1.4
**Chapter 13. Tools**

Axis = 'y':

\[
\text{HomMat3DRotate} = \begin{bmatrix} R_y & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \text{HomMat3D} \quad \text{R}_y = \begin{bmatrix} \cos(\Phi) & 0 & \sin(\Phi) \\ 0 & 1 & 0 \\ -\sin(\Phi) & 0 & \cos(\Phi) \end{bmatrix}
\]

Axis = 'z':

\[
\text{HomMat3DRotate} = \begin{bmatrix} R_z & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \text{HomMat3D} \quad \text{R}_z = \begin{bmatrix} \cos(\Phi) & -\sin(\Phi) & 0 \\ \sin(\Phi) & \cos(\Phi) & 0 \\ 0 & 0 & 1 \end{bmatrix}
\]

The point \((P_x, P_y, P_z)\) is the fixed point of the transformation, i.e., this point remains unchanged when transformed using \text{HomMat3DRotate}. To obtain this behavior, first a translation is added to the input transformation matrix that moves the fixed point onto the origin of the global coordinate system. Then, the rotation is added, and finally a translation that moves the fixed point back to its original position. This corresponds to the following chain of transformations:

\[
\text{HomMat3DRotate} = \begin{bmatrix} 1 & 0 & 0 & +P_x \\ 0 & 1 & 0 & +P_y \\ 0 & 0 & 1 & +P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 & 0 & 0 \\ 0 & R & 0 & 0 \\ 0 & 0 & R & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \text{HomMat3D}
\]

Attention

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix

\[
\begin{bmatrix} r_a & r_b & r_c & t_d \\ r_e & r_f & r_g & t_h \\ r_i & r_j & r_k & t_l \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]

is stored as the tuple \([r_a, r_b, r_c, t_d, r_e, r_f, r_g, t_h, r_i, r_j, r_k, t_l]\).

Parameter

- **HomMat 3D** (input control) .................. affine3d \(\sim HHomMat3dX / \text{VARIANT (real)}\)  
  Input transformation matrix.
  
  **Number of elements**: 12

- **Phi** (input control) .................. angle.rad \(\sim \text{VARIANT (real, integer)}\)  
  Rotation angle.
  
  **Default Value**: 0.78
  
  **Suggested values**: \(\Phi \in \{0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14\}\)
  
  **Typical range of values**: \(0 \leq \Phi \leq 0\)

- **Axis** (input control) .................. string \(\sim \text{String / VARIANT}\)  
  Axis, to be rotated around.
  
  **Default Value**: 'x'
  
  **Suggested values**: \(\text{Axis} \in \{'x', 'y', 'z'\}\)

- **P x** (input control) .................. point3d.x \(\sim \text{VARIANT (real, integer)}\)  
  Fixed point of the transformation (x coordinate).
  
  **Default Value**: 0
  
  **Suggested values**: \(P_x \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

- **P y** (input control) .................. point3d.y \(\sim \text{VARIANT (real, integer)}\)  
  Fixed point of the transformation (y coordinate).
  
  **Default Value**: 0
  
  **Suggested values**: \(P_y \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)

- **P z** (input control) .................. point3d.z \(\sim \text{VARIANT (real, integer)}\)  
  Fixed point of the transformation (z coordinate).
  
  **Default Value**: 0
  
  **Suggested values**: \(P_z \in \{0, 16, 32, 64, 128, 256, 512, 1024\}\)
13.1. AFFINE-TRANSFORMATIONS

\textbf{HomMat3dRotate} (output control) \hdots \texttt{affine3d} \sim \textbf{VARIANT} (\text{real})

Output transformation matrix.

\textbf{Number of elements} : 12

\textbf{Result}

If the parameters are valid, the operator \textbf{HomMat3dRotate} returns TRUE. If necessary, an exception is raised.

\textbf{Parallelization Information}

\textbf{HomMat3dRotate} is reentrant and processed without parallelization.

\textbf{Possible Predecessors}

\textbf{HomMat3dIdentity}, \textbf{HomMat3dTranslate}, \textbf{HomMat3dScale}, \textbf{HomMat3dRotate}

\textbf{Possible Successors}

\textbf{HomMat3dTranslate}, \textbf{HomMat3dScale}, \textbf{HomMat3dRotate}

\textbf{See also}

\textbf{HomMat3dInvert}, \textbf{HomMat3dIdentity}, \textbf{HomMat3dRotate}, \textbf{HomMat3dTranslate}, \textbf{PoseToHomMat3d}, \textbf{HomMat3dToPose}, \textbf{HomMat3dCompose}

\textbf{Module}

Basic operators

\begin{verbatim}

\end{verbatim}

Add a scaling to a homogeneous 3D transformation matrix.

\textbf{HomMat3dScale} adds a scaling by the scale factors \textit{Sx}, \textit{Sy}, and \textit{Sz} to the homogeneous 3D transformation matrix \textbf{HomMat3D} and returns the resulting matrix in \textbf{HomMat3DScale}. The scaling is described by a 3\times3 scaling matrix \textit{S}. It is performed relative to the global (i.e., fixed) coordinate system; this corresponds to the following chain of transformation matrices:

\[
\text{HomMat3DScale} = \begin{bmatrix}
S & 0 & 0 \\\n0 & S & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \text{HomMat3D}
\]

The point \((\text{Px}, \text{Py}, \text{Pz})\) is the fixed point of the transformation, i.e., this point remains unchanged when transformed using \textbf{HomMat3dScale}. To obtain this behavior, first a translation is added to the input transformation matrix that moves the fixed point onto the origin of the global coordinate system. Then, the scaling is added, and finally a translation that moves the fixed point back to its original position. This corresponds to the following chain of transformations:

\[
\text{HomMat3DScale} = \begin{bmatrix}
1 & 0 & 0 & +\text{Px} \\
0 & 1 & 0 & +\text{Py} \\
0 & 0 & 1 & +\text{Pz} \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
S & 0 & 0 \\
0 & S & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
1 & 0 & 0 & -\text{Px} \\
0 & 1 & 0 & -\text{Py} \\
0 & 0 & 1 & -\text{Pz} \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot \text{HomMat3D}
\]

\textbf{Attention}

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix

\[
\begin{bmatrix}
ra & rb & rc & td \\
re & rf & rg & th \\
ri & rj & rk & tl \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

is stored as the tuple \([ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl]\).
Parameter

- **HomMat3D** (input control) .......................... affine3d  \( \sim HHomMat3dX / \text{VARIANT} (\text{real}) \\
  \text{Input transformation matrix.} \\
  \text{Number of elements} : 12

- **Sx** (input control) ................................. number  \( \sim \text{VARIANT} (\text{real, integer}) \\
  \text{Scale factor along the x-axis.} \\
  \text{Default Value} : 2 \\
  \text{Suggested values} : Sx \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 112\} \\
  \text{Restriction} : (Sx \neq 0)

- **Sy** (input control) ................................. number  \( \sim \text{VARIANT} (\text{real, integer}) \\
  \text{Scale factor along the y-axis.} \\
  \text{Default Value} : 2 \\
  \text{Suggested values} : Sy \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 112\} \\
  \text{Restriction} : (Sy \neq 0)

- **Sz** (input control) ................................. number  \( \sim \text{VARIANT} (\text{real, integer}) \\
  \text{Scale factor along the z-axis.} \\
  \text{Default Value} : 2 \\
  \text{Suggested values} : Sz \in \{0.125, 0.25, 0.5, 1, 2, 4, 8, 112\} \\
  \text{Restriction} : (Sz \neq 0)

- **Px** (input control) ................................. point3d.x  \( \sim \text{VARIANT} (\text{real, integer}) \\
  \text{Fixed point of the transformation (x coordinate).} \\
  \text{Default Value} : 0 \\
  \text{Suggested values} : Px \in \{0, 16, 32, 64, 128, 256, 512, 1024\}

- **Py** (input control) ................................. point3d.y  \( \sim \text{VARIANT} (\text{real, integer}) \\
  \text{Fixed point of the transformation (y coordinate).} \\
  \text{Default Value} : 0 \\
  \text{Suggested values} : Py \in \{0, 16, 32, 64, 128, 256, 512, 1024\}

- **Pz** (input control) ................................. point3d.z  \( \sim \text{VARIANT} (\text{real, integer}) \\
  \text{Fixed point of the transformation (z coordinate).} \\
  \text{Default Value} : 0 \\
  \text{Suggested values} : Pz \in \{0, 16, 32, 64, 128, 256, 512, 1024\}

- **HomMat3DScale** (output control) ................. affine3d  \( \sim \text{VARIANT} (\text{real}) \\
  \text{Output transformation matrix.} \\
  \text{Number of elements} : 12

Result

**HomMat3DScale** returns TRUE if all three scale factors are not 0. If necessary, an exception is raised.

Parallelization Information

**HomMat3DScale** is reentrant and processed without parallelization.

Possible Predecessors

**HomMat3dIdentity, HomMat3dTranslate, HomMat3dScale, HomMat3dRotate**

Possible Successors

**HomMat3dTranslate, HomMat3dScale, HomMat3dRotate**

Module

---

Add a translation to a homogeneous 3D transformation matrix.

**HomMat3dTranslate** adds a translation by the vector \( t = (Tx, Ty, Tz) \) to the homogeneous 3D transformation matrix **HomMat3D** and returns the resulting matrix in **HomMat3DTranslate**. The translation is performed

---

HALCON/COM Reference Manual, 2005-2-1
relative to the global (i.e., fixed) coordinate system; this corresponds to the following chain of transformation matrices:

\[
\text{HomMat3DTranslate} = \begin{bmatrix}
1 & 0 & 0 & t \\
0 & 1 & 1 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
T_x \\
T_y \\
T_z
\end{bmatrix}
\]

\[
\text{HomMat3D} = \begin{bmatrix}
x & y & z & 1
\end{bmatrix}
\]

Attention

Note that homogeneous matrices are stored row-by-row as a tuple; the last row is not stored because it is identical for all homogeneous matrices that describe an affine transformation. For example, the homogeneous matrix

\[
\begin{bmatrix}
ra & rb & rc & td \\
re & rf & rg & th \\
ri & rj & rk & tl \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

is stored as the tuple \([ra, rb, rc, td, re, rf, rg, th, ri, rj, rk, tl]\).
Compute a rigid affine transformation from points and angles.

VectorAngleToRigid computes a rigid affine transformation, i.e., a transformation consisting of a rotation and a translation, from a point correspondence and two corresponding angles and returns it in form of the homogeneous transformation matrix \( \text{HomMat2D} \). The matrix consists of 2 components: a rotation matrix \( R \) and a translation vector \( t \) (also see HomMat2dRotate and HomMat2dTranslate):

\[
\text{HomMat2D} = \begin{bmatrix} R & t \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0 & 0 & 1 \end{bmatrix} = H(t) \cdot H(R)
\]

The coordinates of the original point are passed in \((\text{Row1}, \text{Column1})\), while the corresponding angle is passed in \(\text{Angle1}\). The coordinates of the transformed point are passed in \((\text{Row2}, \text{Column2})\), while the corresponding angle is passed in \(\text{Angle2}\). The following equation describes the transformation of the point using homogeneous vectors:

\[
\begin{bmatrix} \text{Row2} \\ \text{Column2} \\ 1 \end{bmatrix} = \text{HomMat2D} \cdot \begin{bmatrix} \text{Row1} \\ \text{Column1} \\ 1 \end{bmatrix}
\]

In particular, the operator VectorAngleToRigid is useful to construct a rigid affine transformation from the results of the matching operators (e.g., FindShapeModel or BestMatchRotMg), which transforms a reference image to the current image or (if the parameters are passed in reverse order) from the current image to the reference image.

\( \text{HomMat2D} \) can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

---

**Parameter**

- **Row1** (input control) .......................................................... point.y \( \sim \) VARIANT (real, integer)
  Row coordinate of the original point.
- **Column1** (input control) ....................................................... point.x \( \sim \) VARIANT (real, integer)
  Column coordinate of the original point.
- **Angle1** (input control) .......................................................... angle.rad \( \sim \) VARIANT (real, integer)
  Angle of the original point.
- **Row2** (input control) .......................................................... point.y \( \sim \) VARIANT (real, integer)
  Row coordinate of the transformed point.
- **Column2** (input control) ....................................................... point.x \( \sim \) VARIANT (real, integer)
  Column coordinate of the transformed point.
- **Angle2** (input control) .......................................................... angle.rad \( \sim \) VARIANT (real, integer)
  Angle of the transformed point.
- **HomMat2D** (output control) ................................................. affine2d \( \sim \) HHomMat2dX / VARIANT (real)
  Output transformation matrix.

**Number of elements**: 6

---

**Example**

- draw_rectangle2 (WindowID, RowTempl, ColumnTempl, PhiTempl, Length1, Length2)
- gen_rectangle2 (Rectangle, RowTempl, ColumnTempl, PhiTempl, Length1, Length2)
- reduce_domain (ImageTempl, Rectangle, ImageReduced)
- create_template_rot (ImageReduced, 4, 0, rad(360), rad(1), ‘sort’, ‘original’, TemplateID)
13.1. AFFINE-TRANSFORMATIONS

while (true)
    best_match_rot_mg (Image, TemplateID, 0, rad(360), 30, 'true', 4, Row, Column, Angle, ErrMatch)
    if (ErrMatch<255)
        vector_angle_to_rigid (Row, Column, Angle, RowTempl, ColumnTempl, 0, HomMat2D)
        affine_trans_image (Image, ImageAffinTrans, HomMat2D, 'constant', 'false')
    endif
endwhile

clear_template (TemplateID)

--- Parallelization Information ---
VectorAngleToRigid is reentrant and processed without parallelization.
--- Possible Predecessors ---
BestMatchRotMg, BestMatchRot
--- Possible Successors ---
HomMat2dInvert, AffineTransImage, AffineTransRegion, AffineTransContourXld, AffineTransPolygonXld, AffineTransPoint2D
--- See also ---
DvfToHomMat2d
--- Alternatives ---
VectorToRigid
--- Module ---
Basic operators


Approximate an affine transformation from point correspondences.

VectorToHomMat2d approximates an affine transformation from at least three point correspondences and returns it in form of the homogeneous transformation matrix HomMat2D (siehe HomMat2dToAffinePar for the content of the homogeneous transformation matrix).

The point correspondences are passed in the tuples (Rows1,Columns1) and (Rows2,Columns2), where corresponding points must be at the same index positions in the tuples. If more than three point correspondences are passed the transformation is overdetermined. In this case, the returned transformation is the transformation that minimizes the distances between the input points (Rows1,Columns1) and the transformed points (Rows2,Columns2), as described in the following equation (points in form of homogeneous vectors):

\[
\sum_i \left( \begin{array}{c} \text{Rows2}[i] \\ \text{Columns2}[i] \\ 1 \end{array} \right) - \text{HomMat2D} \cdot \left( \begin{array}{c} \text{Rows1}[i] \\ \text{Columns1}[i] \\ 1 \end{array} \right) \right]^2 = \text{minimal}
\]

HomMat2D can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

--- Parameter ---

Row1 (input control) .......................................................... point.y  \sim\  \text{VARIANT}\left(\text{real}\right)
Row coordinates of the starting points.

Column1 (input control) ...................................................... point.x  \sim\  \text{VARIANT}\left(\text{real}\right)
Column coordinates of the starting points.
VectorToHomMat2d is reentrant and processed without parallelization.

Possible Successors

See also

AffineTransImage, OpticalFlowMatch

Alternatives

DvToFHomMat2d

Module

Basic operators

Approximate a rigid affine transformation from point correspondences.

VectorToRigid approximates a rigid affine transformation, i.e., a transformation consisting of a rotation and a translation, from at least two point correspondences and returns it in form of the homogeneous transformation matrix HomMat2D. The matrix consists of 2 components: a rotation matrix $R$ and a translation vector $t$ (also see HomMat2dRotate and HomMat2dTranslate):

$$\text{HomMat2D} = \begin{bmatrix} R \ t \\ 0 \ 0 \ 0 \ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0 & 0 \end{bmatrix} = H(t) \cdot H(R)$$

The point correspondences are passed in the tuples (Rows1, Columns1) and (Rows2, Columns2), where corresponding points must be at the same index positions in the tuples. The transformation is always overdetermined. Therefore, the returned transformation is the transformation that minimizes the distances between the original points (Rows1,Columns1) and the transformed points (Rows2,Columns2), as described in the following equation (points in form of homogeneous vectors):

$$\sum_i \left\| \begin{bmatrix} \text{Rows2[i]} \\ \text{Columns2[i]} \\ 1 \end{bmatrix} - \text{HomMat2D} \cdot \begin{bmatrix} \text{Rows1[i]} \\ \text{Columns1[i]} \\ 1 \end{bmatrix} \right\|^2 = \text{minimum}$$

HomMat2D can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.
13.1. AFFINE-TRANSFORMATIONS

- **Columns2** (input control) ........................................................ point.x  ~ VARIANT (real)
  Column coordinates of the transformed points.

- **HomMat2D** (output control) .............................................. affine2d  ~ HHomMat2dX / VARIANT (real)
  Output transformation matrix.

  **Number of elements** : 6

  **Parallelization Information**
  VectorToRigid is reentrant and processed without parallelization.

  **Possible Successors**
  AffineTransImage, AffineTransRegion, AffineTransContourXld, AffineTransPolygonXld, AffineTransPoint2D

  **See also**
  DvfToHomMat2d

  **Alternatives**
  VectorToHomMat2d, VectorToSimilarity

**Module**

**Basic operators**

```c
```

```c
```

Approximate a similarity transformation from point correspondences.

**VectorToSimilarity** approximates a similarity transformation, i.e., a transformation consisting of a scaling, a rotation, and a translation, from at least two point correspondences and returns it in form of the homogeneous transformation matrix HomMat2D. The matrix consists of 3 components: a scaling matrix S, a rotation matrix R, and a translation vector t (also see HomMat2dScale, HomMat2dRotate, and HomMat2dTranslate):

\[
\text{HomMat2D} = \begin{bmatrix} R \cdot S \cdot t \\ 0 \ 0 \ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} R & 0 \\ 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S & 0 \\ 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} = H(t) \cdot H(R) \cdot H(S)
\]

The point correspondences are passed in the tuples (Rows1, Columns1) and (Rows2, Columns2), where corresponding points must be at the same index positions in the tuples. If more than two point correspondences are passed the transformation is overdetermined. In this case, the returned transformation is the transformation that minimizes the distances between the original points (Rows1, Columns1) and the transformed points (Rows2, Columns2), as described in the following equation (points as homogeneous vectors):

\[
\sum_i \left\| \begin{bmatrix} \text{Rows2[i]} \\ \text{Columns2[i]} \\ 1 \end{bmatrix} - \text{HomMat2D} \cdot \begin{bmatrix} \text{Rows1[i]} \\ \text{Columns1[i]} \\ 1 \end{bmatrix} \right\|^2 = \text{minimum}
\]

**HomMat2D** can be used directly with operators that transform data using affine transformations, e.g., AffineTransImage.

**Parameter**

- **Rows1** (input control) ........................................................ point.y  ~ VARIANT (real)
  Row coordinates of the original points.

- **Columns1** (input control) ................................................ point.x  ~ VARIANT (real)
  Column coordinates of the original points.

- **Rows2** (input control) ................................................... point.y  ~ VARIANT (real)
  Row coordinates of the transformed points.

HALCON 6.1.4
CHAPTER 13. TOOLS

- **Columns2** (input control) .......................... point.x \( \sim \) VARIANT( real )
  Column coordinates of the transformed points.

- **HomMat2D** (output control) .......................... affine2d \( \sim \) HHomMat2dX / VARIANT( real )
  Output transformation matrix.

**Number of elements**: 6

**Parallelization Information**

*VectorToSimilarity* is reentrant and processed without parallelization.

**Possible Successors**

AffineTransImage, AffineTransRegion, AffineTransContourXld,
AffineTransPolygonXld, AffineTransPoint2D

**See also**

DvfToHomMat2d

**Alternatives**

VectorToHomMat2d, VectorToRigid

Basic operators

### 13.2 Background-Estimator

```c
void HMiscX.CloseAllBgEsti ( )
void HOperatorSetX.CloseAllBgEsti ( )
```

Delete all background estimation data sets.

**CloseAllBgEsti** deletes the background estimation data sets and releases all used memory.

**Attention**

Since all estimaters are closed by CloseAllBgEsti, all handles become invalid.

**Result**

If it is possible to close the background estimation data sets the operator CloseAllBgEsti returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

CloseAllBgEsti is local and processed completely exclusively without parallelization.

**See also**

CreateBgEsti

**Alternatives**

CloseBgEsti

**Module**

Background estimation

```c
void HOperatorSetX.CloseBgEsti ([in] VARIANT BgEstiHandle )
```

Delete the background estimation data set.

**CloseBgEsti** deletes the background estimation data set and releases all used memory.

**Parameter**

- **BgEstiHandle** (input control) .......................... bg_estimation \( \sim \) HBgEstiX / VARIANT
  ID of the BgEsti data set.

**Example**

/* read Init-Image: */
read_image(InitImage,'Init_Image')

HALCON/COM Reference Manual, 2005-2-1
/* initialize BgEsti-Dataset with fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,'on',?10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
/* - end of background estimation - */
close_bg_est(BgEstiHandle).

____ Result ____
CloseBgEsti returns TRUE if all parameters are correct.
____ Parallelization Information ____
CloseBgEsti is local and processed completely exclusively without parallelization.
____ Possible Predecessors ____
RunBgEsti
____ See also ____
CreateBgEsti
____ Module ____
Background estimation

[out] long BgEstiHandle HImageX.CreateBgEsti ([in] double Syspar1,
in] double Syspar2, [in] String GainMode, [in] double Gain1,
in] double Gain2, [in] String AdaptMode, [in] double MinDiff,
void HBgEstiX.CreateBgEsti ([in] HImageX InitializeImage, 
in] double Syspar1, [in] double Syspar2, [in] String GainMode, 
in] double Gain1, [in] double Gain2, [in] String AdaptMode, 
in] double MinDiff, [in] long StatNum, [in] double ConfidenceC, 
in] double TimeC )
void HOperatorSetX.CreateBgEsti ([in] IObjectX InitializeImage, 
in] VARIANT Syspar1, [in] VARIANT Syspar2, [in] VARIANT GainMode, 
in] VARIANT Gain1, [in] VARIANT Gain2, [in] VARIANT AdaptMode, 
in] VARIANT TimeC, [out] VARIANT BgEstiHandle )

Generate and initialize a data set for the background estimation.
CreateBgEsti creates a new data set for the background estimation and initializes it with the appropriate parameters. The estimated background image is part of this data set. The newly created set automatically becomes the current set.

InitializeImage is used as an initial prediction for the background image. For a good prediction an image of the observed scene without moving objects should be passed in InitializeImage. That way the foreground adaptation rate can be held low. If there is no empty scene image available, a homogenous gray image can be used instead. In that case the adaptation rate for the foreground image must be raised, because initially most of the image
will be detected as foreground. The initialization image must to be of type byte or real. Because of processing single-channel images, data sets must be created for every channel. Size and region of InitializeImage determines size and region for all background estimations (RunBgEsti) that are performed with this data set. Syspar1 and Syspar2 are the parameters of the Kalman system matrix. The system matrix describes the system of the gray value changes according to Kalman filter theory. The background estimator implements a different system for each pixel.

GainMode defines whether a fixed Kalman gain should be used for the estimation or whether the gain should adapt itself depending on the difference between estimation and actual value. If GainMode is set to 'fixed', then Gain1 is used as Kalman gain for pixels predicted as foreground and Gain2 as gain for pixels predicted as background. Gain1 should be smaller than Gain2, because adaptation of the foreground should be slower than adaptation of the background. Both Gain1 and Gain2 should be smaller than 1.0.

If GainMode is set to 'frame', then tables for foreground and background estimation are computed containing Kalman gains for all the 256 possible grayvalue changes. Gain1 and Gain2 then denote the number of frames necessary to adapt the difference between estimated value and actual value. So with a fixed time for adaptation (i.e. number of frames) the needed Kalman gain grows with the grayvalue difference. Gain1 should therefore be larger than Gain2. Different gains for different grayvalue differences are useful if the background estimator is used for generating an 'empty' scene assuming that there are always moving objects in the observed area. In that case the adaptation time for foreground adaptation (Gain1) must not be too big. Gain1 and Gain2 should be bigger than 1.0.

AdaptMode denotes, whether the foreground/background decision threshold applied to the grayvalue difference between estimation and actual value is fixed or whether it adapts itself depending on the grayvalue deviation of the background pixels.

If AdaptMode is set to 'off', the parameter MinDiff denotes a fixed threshold. The parameters StatNum, ConfidenceC and TimeC are meaningless in this case.

If AdaptMode is set to 'on', then MinDiff is interpreted as a base threshold. For each pixel an offset is added to this threshold depending on the statistical evaluation of the pixel value over time. StatNum holds the number of data sets (past frames) that are used for computing the grayvalue variance (FIR-Filter). ConfidenceC is used to determine the confidence interval. The confidence interval determines the values of the background statistics if background pixels are hidden by a foreground object and thus are detected as foreground. According to the student t-distribution the confidence constant is 4.30 (3.25, 2.82, 2.26) for a confidence interval of 99.8% (99.0%, 98.0%, 95.0%). TimeC holds a time constant for the exp-function that raises the threshold in case of a foreground estimation of the pixel. That means, the threshold is raised in regions where movement is detected in the foreground. That way larger changes in illumination are tolerated if the background becomes visible again. The main reason for increasing this tolerance is the impossibility for a prediction of illumination changes while the background is hidden. Therefore no adaptation of the estimated background image is possible.

---

**Attention**

If GainMode was set to 'frame', the run-time can be extremely long for large values of Gain1 or Gain2, because the values for the gains’ table are determined by a simple binary search.

---

**Parameter**

- **InitializeImage** (input iconic) ......................... image  \( \sim HimageX / IObjectX(\text{byte, real}) \) initialization image.
- **Syspar1** (input control) .......................................................... real  \( \sim \text{double / VARIANT} \)
  1. system matrix parameter.
  - **Default Value**: 0.7
  - **Suggested values**: Syspar1 \( \in \{0.65, 0.7, 0.75\}\)
  - **Typical range of values**: 0.05 \( \leq \text{Syspar1} \leq 0.05\)
  - **Recommended Increment**: 0.05
- **Syspar2** (input control) .......................................................... real  \( \sim \text{double / VARIANT} \)
  2. system matrix parameter.
  - **Default Value**: 0.7
  - **Suggested values**: Syspar2 \( \in \{0.65, 0.7, 0.75\}\)
  - **Typical range of values**: 0.05 \( \leq \text{Syspar2} \leq 0.05\)
  - **Recommended Increment**: 0.05
13.2. BACKGROUND-ESTIMATOR

GainMode (input control) ........................................ string ~ String / VARIANT
Gain type.
Default Value: ‘fixed’
List of values: GainMode ∈ {'fixed', 'frame'}

Gain1 (input control) ........................................ real ~ double / VARIANT
Kalman gain / foreground adaptation time.
Default Value: 0.002
Suggested values: Gain1 ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
Restriction: (0.0 ≤ Gain1)

Gain2 (input control) ........................................ real ~ double / VARIANT
Kalman gain / background adaptation time.
Default Value: 0.02
Suggested values: Gain2 ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
Restriction: (0.0 ≤ Gain2)

AdaptMode (input control) ..................................... string ~ String / VARIANT
Threshold adaptation.
Default Value: ‘on’
List of values: AdaptMode ∈ {'on', 'off'}

MinDiff (input control) ....................................... real ~ double / VARIANT
Foreground/background threshold.
Default Value: 7.0
Suggested values: MinDiff ∈ {3.0, 5.0, 7.0, 9.0, 11.0}
Recommended Increment: 0.2

StatNum (input control) ..................................... integer ~ long / VARIANT
Number of statistic data sets.
Default Value: 10
Suggested values: StatNum ∈ {5, 10, 20, 30}
Recommended Increment: 5

ConfidenceC (input control) ................................ real ~ double / VARIANT
Confidence constant.
Default Value: 3.25
Suggested values: ConfidenceC ∈ {4.30, 3.25, 2.82, 2.62}
Recommended Increment: 0.01
Restriction: (0.0 < ConfidenceC)

TimeC (input control) ....................................... real ~ double / VARIANT
Constant for decay time.
Default Value: 15.0
Suggested values: TimeC ∈ {10.0, 15.0, 20.0}
Recommended Increment: 5.0
Restriction: (0.0 < TimeC)

BgEstiHandle (output control) ......................... bg_estimation ~ long / HBgEstiX / VARIANT
ID of the BgEsti data set.

Example

/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize 1. BgEsti-Dataset with fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
               'on',7.0,10,3.25,15.0,BgEstiHandle1)
/* initialize 2. BgEsti-Dataset with frame orientated gains and fixed threshold */
create_bg_esti(InitImage,0.7,0.7,'frame',30.0,4.0,
               'off',9.0,10,3.25,15.0,BgEstiHandle2).

Result

CreateBgEsti returns TRUE if all parameters are correct.
### Parallelization Information

`CreateBgEsti` is local and processed completely exclusively without parallelization.

### Possible Successors

### RunBgEsti

### SetBgEstiParam, CloseBgEsti

Background estimation

**Example**

```cpp
/* read Init-Image:*/
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with fixed gains and threshold adaptation: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02 ,
'on',7.0,10,3.25,15.0,BgEstiHandle)
```

### Module

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BgEstiHandle</strong></td>
<td>(input control)</td>
</tr>
<tr>
<td></td>
<td>bg_estimation</td>
</tr>
<tr>
<td><strong>Syspar1</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
<tr>
<td><strong>Syspar2</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
<tr>
<td><strong>GainMode</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td><strong>Gain1</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
<tr>
<td><strong>Gain2</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
<tr>
<td><strong>AdaptMode</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td><strong>MinDiff</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
<tr>
<td><strong>StatNum</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>integer</td>
</tr>
<tr>
<td><strong>ConfidenceC</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
<tr>
<td><strong>TimeC</strong></td>
<td>(output control)</td>
</tr>
<tr>
<td></td>
<td>real</td>
</tr>
</tbody>
</table>

### GetBgEstiParam

`GetBgEstiParam` returns the parameters of the data set. The returned parameters are the same as in `CreateBgEsti` and `SetBgEstiParam` (see these for an explanation).

```cpp
[out] double HBgEstiX.GetBgEstiParam ((out) double Syspar2,
[out] String GainMode, (out) double Gain1, (out) double Gain2,
[out] String AdaptMode, (out) double MinDiff, (out) long StatNum,
[out] double ConfidenceC, (out) double TimeC )
```

```cpp
void HOperatorSetX.GetBgEstiParam ([in] VARIANT BgEstiHandle,
[out] VARIANT Syspar1, (out) VARIANT Syspar2, [out] VARIANT GainMode,
[out] VARIANT Gain1, [out] VARIANT Gain2, [out] VARIANT AdaptMode,
[out] VARIANT MinDiff, [out] VARIANT StatNum, [out] VARIANT ConfidenceC,
[out] VARIANT TimeC )
```
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
/* change only the gain parameter in dataset: */
get_bg_esti_params(BgEstiHandle,par1,par2,par3,par4,
par5,par6,par7,par8,par9,par10)
set_bg_esti_params(BgEstiHandle,par1,par2,par3,0.004,
0.08,par6,par7,par8,par9,par10)
/* read the next image in sequence: */
read_image(Image3,'Image_3')
/* estimate the Background: */
run_bg_esti(Image3,Region3,BgEstiHandle)
/* display the foreground region: */
disp_region(Region3,WindowHandle)
/* etc. */

Result
GetBgEstiParams returns TRUE if all parameters are correct.

Parallelization Information
GetBgEstiParams is reentrant and processed without parallelization.

Possible Predecessors
CreateBgEsti

Possible Successors
RunBgEsti
SetBgEstiParams

See also
SetBgEstiParams

Module
Background estimation

void HI mageX.GiveBgEsti ([in] long BgEstiHandle )
[out] HI mageX BackgroundImage HBgEstiX.GiveBgEsti ( )

void HOperatorSetX.GiveBgEsti ([out] HUntypedObjectX BackgroundImage,
[in] VARIANT BgEstiHandle )

Return the estimated background image.
GiveBgEsti returns the estimated background image of the current BgEsti data set. The background image has the same type and size as the initialization image passed in CreateBgEsti.

Parameter

▷ BackgroundImage (output iconic) .................image ~ HI mageX / HUntypedObjectX ( byte, real )
Estimated background image of the current data set.

▷ BgEstiHandle (input control) ....................bg estimation ~ long / HBgEstiX / VARIANT
ID of the BgEsti data set.
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with 
fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, 
'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* give the background image from the aktive dataset: */
give_bg_esti(BgImage,BgEstiHandle)
/* display the background image: */
disp_image(BgImage,WindowHandle).


GiveBgEsti returns TRUE if all parameters are correct.

Parallelization Information
GiveBgEsti is local and processed completely exclusively without parallelization.

Possible Predecessors
RunBgEsti

Possible Successors
RunBgEsti,CreateBgEsti,UpdateBgEsti

See also
RunBgEsti,UpdateBgEsti,CreateBgEsti

Module
Background estimation

Estimate the background and return the foreground region.
RunBgEsti adapts the background image stored in the BgEsti data set using a Kalman filter on each pixel and returns a region of the foreground (detected moving objects).

For every pixel an estimation of its grayvalue is computed using the values of the current data set and its stored background image and the current image (PresentImage). By comparison to the threshold (fixed or adaptive, see CreateBgEsti) the pixels are classified as either foreground or background.

The background estimation processes only single-channel images. Therefore the background has to be adapted separately for every channel.

The background estimation should be used on half- or even quarter-sized images. For this, the input images (and the initialization image!) has to be reduced using ZoomImageFactor. The advantage is a shorter run-time on one hand and a low-band filtering on the other. The filtering eliminates high frequency noise and results in a more reliable estimation. As a result the threshold (see CreateBgEsti) can be lowered. The foreground region returned by RunBgEsti then has to be enlarged again for further processing.
The passed image (**PresentImage**) must have the same type and size as the background image of the current
data set (initialized with **CreateBgEsti**).

**Parameter**

- **PresentImage** (input iconic) \( \text{image} \sim HImageX / IObjectX \) (byte, real)
  Current image.
- **ForegroundRegion** (output iconic) \( \text{region} \sim HRegionX / HUntypedObjectX \)
  Region of the detected foreground.
- **BgEstiHandle** (input control) \( \text{bg estimation} \sim \text{long} / HBgEstiX / \text{VARIANT} \)
  ID of the BgEsti data set.

**Example**

```c
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with 
 fixed gains and threshold adaption */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02, 
 'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
```

**Result**

**RunBgEsti** returns TRUE if all parameters are correct.

**Parallelization Information**

**RunBgEsti** is reentrant and processed without parallelization.

**Possible Predecessors**

**CreateBgEsti, UpdateBgEsti**

**Possible Successors**

**RunBgEsti, GiveBgEsti, UpdateBgEsti**

**See also**

**SetBgEstiParams, CreateBgEsti, UpdateBgEsti, GiveBgEsti**

**Module**

Background estimation
void HBgEstiX.SetBgEstiParams ([in] double Syspar1,
[in] double Syspar2, [in] String GainMode, [in] double Gain1,
[in] double Gain2, [in] String AdaptMode, [in] double MinDiff,

void HOperatorSetX.SetBgEstiParams ([in] VARIANT BgEstiHandle,
[in] VARIANT Syspar1, [in] VARIANT Syspar2, [in] VARIANT GainMode,
[in] VARIANT Gain1, [in] VARIANT Gain2, [in] VARIANT AdaptMode,
[in] VARIANT TimeC )

Change the parameters of the data set.

SetBgEstiParams is used to change the parameters of the data set. The parameters passed by SetBgEstiParams are the same as in CreateBgEsti (see there for an explanation).

The image format cannot be changed! To do this, a new data set with an initialization image of the appropriate format has to be created.

To exchange the background image completely, use UpdateBgEsti. The current image then has to be passed for both the input image and the update region.

Attention

If GainMode was set to ’frame’, the run-time can be extremely long for large values of Gain1 or Gain2, because the values for the gains’ table are determined by a simple binary search.

Parameter

▷ BgEstiHandle (input control) ............................................. bg_estimation ~ HBgEstiX / VARIANT
ID of the BgEsti data set.

▷ Syspar1 (input control) ................................................real ~ double / VARIANT
1. system matrix parameter.
Default Value : 0.7
Suggested values : Syspar1 ∈ {0.65, 0.7, 0.75}
Typical range of values : 0.05 ≤ Syspar1 ≤ 0.05
Recommended Increment : 0.05

▷ Syspar2 (input control) ................................................real ~ double / VARIANT
2. system matrix parameter.
Default Value : 0.7
Suggested values : Syspar2 ∈ {0.65, 0.7, 0.75}
Typical range of values : 0.05 ≤ Syspar2 ≤ 0.05
Recommended Increment : 0.05

▷ GainMode (input control) ............................................. string ~ String / VARIANT
Gain type.
Default Value : ’fixed’
List of values : GainMode ∈ {’fixed’, ’frame’}

▷ Gain1 (input control) ............................................. real ~ double / VARIANT
Kalman gain / foreground adaptation time.
Default Value : 0.002
Suggested values : Gain1 ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
Restriction : (0.0 ≤ Gain1)

▷ Gain2 (input control) ............................................. real ~ double / VARIANT
Kalman gain / background adaptation time.
Default Value : 0.02
Suggested values : Gain2 ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
Restriction : (0.0 ≤ Gain2)

▷ AdaptMode (input control) ............................................. string ~ String / VARIANT
Threshold adaptation.
Default Value : ’on’
List of values : AdaptMode ∈ {’on’, ’off’}
MinDiff (input control) ...................................................... real \sim double / VARIANT
Foreground/background threshold.
Default Value : 7.0
Suggested values : MinDiff \in \{3.0, 5.0, 7.0, 9.0, 11.0\}
Recommended Increment : 0.2

StatNum (input control) .................................................... integer \sim long / VARIANT
Number of statistic data sets.
Default Value : 10
Suggested values : StatNum \in \{5, 10, 20, 30\}
Recommended Increment : 5

ConfidenceC (input control) ................................................. real \sim double / VARIANT
Confidence constant.
Default Value : 3.25
Suggested values : ConfidenceC \in \{4.30, 3.25, 2.82, 2.62\}
Recommended Increment : 0.01
Restriction : (0.0 < ConfidenceC)

TimeC (input control) ....................................................... real \sim double / VARIANT
Constant for decay time.
Default Value : 15.0
Suggested values : TimeC \in \{10.0, 15.0, 20.0\}
Recommended Increment : 5.0
Restriction : (0.0 < TimeC)

Example

/* read Init-Image:*/
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with
fixed gains and threshold adaption: */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02 ,
'on',7.0,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* display the foreground region: */
disp_region(Region1,WindowHandle)
/* read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* display the foreground region: */
disp_region(Region2,WindowHandle)
/* etc. */
/* change parameter in dataset: */
set_bg_esti_params(BgEstiHandle,0.7,0.7,'fixed',
0.004,0.08,'on',9.0,10,3.25,20.0)
/* read the next image in sequence: */
read_image(Image3,'Image_3')
/* estimate the Background: */
run_bg_esti(Image3,Region3,BgEstiHandle)
/* display the foreground region: */
disp_region(Region3,WindowHandle)
/* etc. */

Result
SetBgEstiParams returns TRUE if all parameters are correct.

Parallelization Information
SetBgEstiParams is reentrant and processed without parallelization.
CHAPTER 13. TOOLS

### Possible Predecessors

CreateBgEsti

### Possible Successors

RunBgEsti

See also

UpdateBgEsti

---

**Module**

Background estimation

---

```c
void HImageX.UpdateBgEsti ([in] HRegionX UpdateRegion,
[in] long BgEstiHandle )

void HRegionX.UpdateBgEsti ([in] HImageX PresentImage,
[in] HRegionX UpdateRegion )

void HOperatorSetX.UpdateBgEsti ([in] IHObjectX PresentImage,
[in] IHObjectX UpdateRegion, [in] VARIANT BgEstiHandle )
```

**Change the estimated background image.**

**UpdateBgEsti** overwrites the image stored in the current BgEsti data set with the grayvalues of **PresentImage** within the bounds of **UpdateRegion**. This can be used for a "hard" adaptation: Image regions with a sudden change in (known) background can be adapted very fast this way.

---

**Attention**

The passed image (**PresentImage**) must have the same type and size as the background image of the current data set (initialized with **CreateBgEsti**).

---

**Parameter**

- **PresentImage** (input iconic) ......................... image  \(\sim\) HImageX / IHObjectX ( byte, real )
  Current image.
- **UpdateRegion** (input iconic) ...............................region  \(\sim\) HRegionX / IHObjectX
  Region describing areas to change.
- **BgEstiHandle** (input control) ......................... bg estimation  \(\sim\) long / HBgEstiX / VARIANT
  ID of the BgEsti data set.

---

**Example**

```c
/* read Init-Image: */
read_image(InitImage,'Init_Image')
/* initialize BgEsti-Dataset with fixed gains and threshold adaption */
create_bg_esti(InitImage,0.7,0.7,'fixed',0.002,0.02,
'on',7,10,3.25,15.0,BgEstiHandle)
/* read the next image in sequence: */
read_image(Image1,'Image_1')
/* estimate the Background: */
run_bg_esti(Image1,Region1,BgEstiHandle)
/* use the Region and the information of a knowledge base */
/* to calculate the UpdateRegion */
update_bg_esti(Image1,UpdateRegion,BgEstiHandle)
/* then read the next image in sequence: */
read_image(Image2,'Image_2')
/* estimate the Background: */
run_bg_esti(Image2,Region2,BgEstiHandle)
/* etc. */
```

---

**Result**

**UpdateBgEsti** returns TRUE if all parameters are correct.
Parallelization Information

UpdateBgEsti is reentrant and processed without parallelization.

Possible Predecessors

RunBgEsti

Possible Successors

RunBgEsti

See also

RunBgEsti, GiveBgEsti

Module

Background estimation

13.3 Barcode

| out | VARIANT Characters HBarcodeX.Decode1DBarCode |
| in  | VARIANT BarCodeElements, in VARIANT BarCodeDescr, |
| out | VARIANT Reference, out long IsCorrect |

void HOperatorSetX.Decode1DBarCode (in VARIANT BarCodeElements, |
| in  | VARIANT BarCodeDescr, out VARIANT Characters, out VARIANT Reference, |
| out | VARIANT IsCorrect ) |

Decoding of a sequence of elements of a bar code.

Decode1DBarCode decodes a sequence of elements which have been extracted by Find1DBarCode or Get1DBarCode into a sequence of characters. As input the widths of the elements (in pixels) are used. The discrete form as it is returned from Discrete1DBarCode can optionally be used. Otherwise Decode1DBarCode creates the discrete form automatically.

The result of Decode1DBarCode is a sequence of Characters and the corresponding reference numbers (Reference). In addition a parity check is applied, the result of which is returned in IsCorrect. If all characters are used to store user data (i.e. no parity is used) the value of this parameter has to be ignored.

Attention

For bar codes of type Pharmacode the reading direction cannot be determined from the bar code data because the bar code does not specify extra characters (like start, stop, or parity check characters) that enable the determination of a reading direction, and hence the bar code can always be decoded in both directions. Therefore, for PharmaCodes the results of decoding the code in both possible reading directions are returned as two values in Characters as well as Reference. The decision which Element of Characters and Reference contains the correct reading direction must be made by the caller based on the orientation of the bar code (as returned in the parameter Orientation in Find1DBarCode and Find1DBarCodeRegion). The respective first element of Characters and Reference corresponds to the standard reading direction in the orientation given by Orientation, while the second element corresponds to the opposite reading direction. If, for example, the orientation is 0, the first element corresponds to the reading direction from right to left. If the orientation is \( \pi/2 \), the first element corresponds to the reading direction from top to bottom.

Parameter

- **BarCodeElements** (input control) number \( \Rightarrow \) VARIANT (real) Widths of the elements of the bar code.
- **BarCodeDescr** (input control) barcode_Id \( \Rightarrow \) VARIANT (string, integer, real) Description of a bar code class.
- **Characters** (output control) string \( \Rightarrow \) VARIANT (string) Decoded characters in standard interpretation.
- **Reference** (output control) integer \( \Rightarrow \) VARIANT (integer) Decoded characters as numbers.
- **IsCorrect** (output control) integer \( \Rightarrow \) long / VARIANT Information whether the bar code is correct.

List of values: IsCorrect \( \in \{0, 1\} \)
The operator `Decode1DBarCode` returns the value TRUE if the transferred bar code description is correct and the element list can be decoded.

`Decode1DBarCode` is reentrant and processed without parallelization.

Possible Predecessors

`Find1DBarCode`, `Get1DBarCode`

Module

Barcode reader

```plaintext
[out] VARIANT SymbolCharacters HBarcodeX.Decode2DBarCode
[[in] VARIANT BarCodeDescr, [in] VARIANT BarCodeDimension,
[in] VARIANT BarCodeData, [out] VARIANT CorrSymbolData,
[out] VARIANT DecodedData, [out] long DecodingError,
[out] VARIANT StructuredAppend )

void HOperatorSetX.Decode2DBarCode ([in] VARIANT BarCodeDescr,
[out] VARIANT SymbolCharacters, [out] VARIANT CorrSymbolData,
[out] VARIANT DecodedData, [out] VARIANT DecodingError,
[out] VARIANT StructuredAppend )
```

Decoding 2D bar code data.

`Decode2DBarCode` decodes (binary) data of a 2D bar code that has been extracted with the help of `Get2DBarCode` or `Get2DBarCodePos`. The parameter `BarCodeData` contains the data values, `BarCodeDimension` describes the size of the data matrix (see operator `Get2DBarCode`), and `BarCodeDescr` describes the 2D bar code class (see operator `Gen2DBarCodeDescr`).

First, the binary data are converted into a stream of 8-bit characters (`SymbolCharacters`). This stream consists of the actual, possibly erroneous data and of additional characters used by the subsequent error correcting routine. If all errors could be corrected, the resulting, still undecoded data is returned in the parameter `CorrSymbolData`. The parameter `DecodingError` contains the number of corrected errors, or a negative number if the correction failed.

Then, the data is decoded and returned as a tuple of ASCII characters in the parameter `DecodedData`. If the symbol is part of a group of symbols (ECC 200: “Structured Append”), the parameter `StructuredAppend` contains the number (place) of the symbol in the group, the overall number of symbols in the group, and a group identifier. If the symbol is not part of a group, the first two entries of `StructuredAppend` contain a 1.

Parameter

- **BarCodeDescr** (input control).................. `barcode_2d` ~ VARIANT (string, integer, real) Description of the bar code class.
- **BarCodeDimension** (input control)............... `integer` ~ VARIANT (integer) Tuple with the dimension of the examined symbol. In the case of ECC 200: width, height, symbol code.
- **BarCodeData** (input control).................... `integer` ~ VARIANT (integer) Tuple with the data values of the examined symbol.
- **SymbolCharacters** (output control).............. `string` ~ VARIANT (string) Data and error codewords of the symbol.
- **CorrSymbolData** (output control)............... `integer` ~ VARIANT (integer) Corrected data codewords of the symbol.
- **DecodedData** (output control).................. `integer` ~ VARIANT (integer) Decoded data characters as numbers.
- **DecodingError** (output control)................ `integer` ~ long / VARIANT Number of errors during the decoding process.
- **StructuredAppend** (output control)............. `integer` ~ VARIANT (integer) If the symbol belongs to a group (“structured append”): position in the group, total symbol number, group (“file”) id.
The return value signals incorrect parameters as well as a failure in the decoding procedure. The error code 8812, e.g., signals that the decoded data stream contained an invalid data word. Up to now, user-defined control words can not be handled by the operator. If such control words are detected in the decoded data stream, the error code 8813 is returned.

**Parallelization Information**

*Decode2DBarCode* is reentrant and processed without parallelization.

**Possible Predecessors**

Get2DBarCode

Barcode reader

---

**[out] VARIANT DiscreteBarCode HBarcodex.Discrete1DBarCode**


**void HOperatorSetX.Discrete1DBarCode**


*Generate a discrete bar code from the elements widths.*

*Discrete1DBarCode* converts the list of element widths (output from *Find1DBarCode* or *Get1DBarCode*) into a discrete bar code. Thus every element is then represented by its number of modules (1,2,...) and no longer as its width in pixels.

This operator is used if the bar code type is not available so that *Decode1DBarCode* cannot be applied, thus the user wants to find the bar code with the help of HALCON operators and then himself decode the bar code. To create the bar code description the operator *Gen1DBarCodeDescrGen* is used and with *Find1DBarCode* the element widths are extracted. Then *Discrete1DBarCode* is used to create the list of the multiple of the modules.

**Parameter**

▷ *Elements* (input control) ........................................... number ~ VARIANT (real)
   List of elements widths of the bar code.

▷ *BarCodeDescr* (input control) .................................barcode_id ~ VARIANT (string, integer, real)
   Description of a bar code class.

▷ *DiscreteBarCode* (output control) .............................number ~ VARIANT (integer)
   Widths of elements as multiple of modules.

**Result**

The operator *Discrete1DBarCode* returns the value TRUE if the transferred bar code description is correct and the element list can be discretized.

**Parallelization Information**

*Discrete1DBarCode* is reentrant and processed without parallelization.

**Possible Predecessors**

*Find1DBarCode, Get1DBarCode*

See also

*Decode1DBarCode*

**Module**

Barcode reader
Look for one bar code in an image.

**Find1DBarCode** looks for a bar code in an image. As input it needs the description of the bar code as generated by **Gen1DBarCodeDescr** or **Gen1DBarCodeDescrGen**. With this description the kind of bar code to be extracted is specified.

The result of the operator are the widths of the elements and the bar code region with its orientation. If a bar code was found **BarcodeFound** returns the value 1. Otherwise it returns 0.

To control the internal image processing the parameters **GenericName** and **GenericValue** are used. This is done by passing the names of the control parameters to be changed in **GenericName** as a list of strings. In **GenericValue** the values are passed at the corresponding index positions.

Normally none of the values have to be changed because the operator can adapt automatically to changing situations. Only in the case of difficult capturing conditions or with special bar codes a change should be applied. Here is the list of all parameters which can be changed:

- **amplitude_sobel** Minimum amplitude for edge extraction: The first step of the search for the bar code is based on the sobel filter. In contrast to thresholding this has the advantage of being independent of different illumination conditions. In the case of a low contrast (difference between bright and dark elements) this value can be chosen lower to find all parts of the bar code. If the contrast is very good the value can be set to a higher value. This results in a better runtime because less pixels have to be evaluated. The effect of this parameter can be checked with the operator **SobelDir** with edge type 'sum_abs'.

  Default value: 70

  Used for: Search for bar code region

- **min_size_element** During the analysis of elements (or parts of them) this parameter is used to eliminate very small objects which do not belong to the bar code. You have to be aware that with a low image quality elements can be divided into smaller pieces. The parameter has to be larger than the smallest piece. With good image quality the value can be larger to reduce runtime. In any case the value may not be larger than the length of one element (in pixels). The area calculated includes only the border of the elements and is thus smaller. If the value is too large parts of the bar code may be missing.

  Default value: 30

  Used for: Search for bar code region

- **max_size_element** Similar to the minimal size this parameter is used to suppress large objects. You have to know that with a low resolution multiple elements can be extracted as one intermediate region. The value has thus to be larger than the size of this region. This can be the size of many elements. The area internally calculated includes only the border of the elements. Furthermore the value should be smaller than the total size of the bar code.

  Default value: 15000

  Used for: Search for bar code region
'angle_range' This parameter has the same function as the parameter 'sum_angles' but here it is used during the initial search for elements of the bar code. It has influence on the estimation of the orientation of the elements. For low quality images this value has to be large. A small value results in a reduced runtime. The value is given in degrees. There is no need to change the parameter.

Default value: 24

Used for: Search for bar code region

'correct_angle' This parameter is used to decide if a region can be part of a bar code. For this the orientation of all boundary points are calculated. If enough points have the same orientation the region is accepted for further processing. Increasing this value reduces the amount of accepted regions and improves speed. For low image quality this value has to be low.

Default value: 0.5 (corresponds 50 percent)

Used for: Search for bar code region

'dilation_factor' After the initial step of finding elements these are combined into a bar code region (CodeRegion). This is implemented using the closing operator. The mask size is automatically estimated. If this estimation is wrong it can be corrected using this parameter. If the bar code elements have large gaps the value has to be larger than one. If the bar code is very close to other objects and is wrongly connected to this object the value can be smaller than 1. The parameter is independent of the image quality and runtime.

Default value: 1

Used for: Search for bar code region

'sum_angles' This parameter is used for the final estimation of the orientation of the bar code region (value of Orientation). For this at first the orientation of the boundaries of all elements are calculated. The most frequent orientation is calculated. Using the neighboring orientations the mean orientation is calculated. The parameter determines the amount of neighboring angles used for summing up. The value is given in degrees.

Default value: 40

Used for: Search for bar code region

'min_area_bar_code' This value is used for the final decision if the region is a bar code. The parameter specifies the minimum area of the bar code. This area used here consists only of the boundary pixels of the elements and is thus smaller than the area of CodeRegion. If areas were found which are too small the value has to be increased. The parameter can also be used if small areas with the same orientation as the bar code elements were found. The parameter is independent of the image quality and runtime.

Default value: 1000

Used for: Search for bar code region

'sigma_project' For the extraction of the element widths the gray values of the bar code region are projected in the direction of the elements. The data calculated this way is smoothed to reduce noise. For bar codes with large gaps between the elements this value can be increased if extra elements were detected. The value should be larger than 0.5.

Default value: 0.7

Used for: Extraction of element widths

'amplitude_project' For the calculation of the element widths from the gray projections this parameter controls the minimum amplitude (gray value difference) between dark and light elements. With a lot of noise (e.g. scratches) this value can be increased. For low contrast images the value has to be low.
Default value: 3

Used for: Extraction of element widths

'width_project' The gray value projections are calculated only in a part of the bar code region along several scanlines (see below). This parameter specifies the half width of the area that is used to calculate the gray value projections. If the orientation of the region is not estimated correctly this value must be small. If there are distortions (e.g., scratches) this value can be increased. For very narrow bar codes the value can be set to a small value.

Default value: 5

Used for: Extraction of element widths

'add_length_project' Because the length of the bar code is sometimes too short to get elements at the beginning and at the end the region is extended based on this parameter. If the bar code lies near to another object resulting in a connection of both areas this value can be decreased.

Default value: 5

Used for: Extraction of element widths

'interpolation_project' With this parameter the type of interpolation used for the gray projection is determined. With a value of 1 a bilinear interpolation is used. With 0 no interpolation is used. In this case the runtime is shorter but it might result in a wrong extraction of the elements.

Default value: 1

Used for: Extraction of element widths

'num_scanlines' This value determines the maximum number of scanlines in which the bar code is searched. This value should be chosen relatively large if it is expected that bar codes with severe disturbances must be read. If num_scanlines is set to a value greater than 1, the default value of width_project is 5, otherwise 25. This choice of default values for width_project enables a robust extraction of the bar code even if only a single scanline is used.

Default value: 9

Used for: Extraction of element widths

'max_extra_elements' With this parameter, elements extracted in the vicinity of the bar code that are parallel to the bars of the bar code can be eliminated. Hence, this parameter enables the correct decoding of the bar code if there are extraneous bars close to the bar code. The value of max_extra_elements is used for both ends of the bar code separately, so that a total of 2*max_extra_elements extraneous elements (bars and spaces) can be eliminated. Hence, on each side of the bar code up to max_extra_elements/2 extraneous bars can be eliminated.

Default value: 6

Used for: Extraction of element widths

Please note that these parameters should normally not be changed.
13.3. BARCODE

GenericName (input control) ......................... string(-array) ~ VARIANT (string)
Names of optional control parameters.

Default Value: []

List of values: GenericName ∈ {'amplitude_sobel', 'min_size_element', 'max_size_element',
'angle_range', 'correct_angle', 'dilation_factor', 'sum_angles', 'sigma_project', 'amplitude_project',
'width_project', 'add_length_project', 'interpolation_project', 'num_scanlines', 'max_extra_elements'}

GenericValue (input control) ......................... number(-array) ~ VARIANT (integer, real)
Values of optional control parameters.

Default Value: []

BarcodeFound (output control) ......................... integer ~ long / VARIANT
Information whether the bar code was found.

List of values: BarcodeFound ∈ {0, 1}

BarcodeElements (output control) ....................... number ~ VARIANT (real)
Widths of elements.

Orientation (output control) .............................. angle.rad ~ double / VARIANT
Orientation of bar code.

The operator Find1DBarCode returns the value TRUE if the parameters are correct.

Find1DBarCode is reentrant and processed without parallelization.

Possible Predecessors
Gen1DBarCodeDescr, Gen1DBarCodeDescrGen

Possible Successors
Decode1DBarCode, Discrete1DBarCode

See also
SobelDir, GrayProjections, GenMeasureRectangle2

Alternatives
Find1DBarCodeRegion

Module
Barcode reader

Find1DBarCodeRegion looks for multiple bar code regions in an image.

Find1DBarCodeRegion looks for multiple barcodes in the image. In contrast to Find1DBarCode this
operator is used if an image contains more than one bar code. Here only the regions but not the widths of the
elements are extracted. For every region the orientation in radians is calculated.

The control of the image processing is identical to Find1DBarCode. The description of the parameters
GenericName and GenericValue can be found at this operator.

Parameter

Image (input iconic) ......................... image ~ HImageX / IObjectX (byte, uint2)
Image with bar codes inside.

CodeRegion (output iconic) ......................... region(-array) ~ HRegionX / HUntypedObjectX
Regions of bar codes.
CHAPTER 13. TOOLS

**BarCodeDescr** (input control) .................................................. barcode_id ~ VARIANT (string, integer, real)
Description of a bar code class.

**GenericName** (input control) .................................................. string(-array) ~ VARIANT (string)
Names of optional parameters.

Default Value : []
List of values : GenericName ∈ {'amplitude_sobel', 'min_size_element', 'max_size_element', 'angle_range', 'correct_angle', 'dilation_factor', 'sum_angles'}

**GenericValue** (input control) .................................................. number(-array) ~ VARIANT (integer, real, string)
Values of optional parameters.

Default Value : []

**Orientation** (output control) .................................................. real(-array) ~ VARIANT (real)
Orientation of bar code.

The operator **Find1DBarCodeRegion** returns the value TRUE if the parameters are correct and at least one bar code is found.

---

### Parallelization Information

**Find1DBarCodeRegion** is reentrant and processed without parallelization.

---

### Possible Predecessors

Gen1DBarCodeDescr, Gen1DBarCodeDescrGen

---

### Possible Successors

Get1DBarCode, CountObj, SelectObj, ReduceDomain

---

### See also

SobelDir

---

### Alternatives

Find1DBarCode

---

### Module

Barcode reader

---

```plaintext
[out] HRegionX CodeRegion HImageX.Find2DBarCode
([in] VARIANT BarCodeDescr, [in] VARIANT GenParamNames,
[in] VARIANT GenParamValues, [out] VARIANT CodeRegDescr )

[out] HRegionX CodeRegion HBarcodeX.Find2DBarCode ([in] HImageX Image,
[in] VARIANT BarCodeDescr, [in] VARIANT GenParamNames,
[in] VARIANT GenParamValues, [out] VARIANT CodeRegDescr )

void HOperatorSetX.Find2DBarCode ([in] HOBJECTX Image,
[out] HUntypedObjectX CodeRegion, [in] VARIANT BarCodeDescr,
[in] VARIANT GenParamNames, [in] VARIANT GenParamValues,
[out] VARIANT CodeRegDescr )
```

**Find regions that might contain a 2D bar code.**

**Find2DBarCode** searches in the image **Image** for regions that might contain a 2D bar code. Candidate regions are returned in the array of regions **CodeRegion**. Whether such a region really contains a (readable) 2D bar code can only be determined with the help of the operator **Get2DBarCode** (or **Get2DBarCodePos**).

Besides regions that might contain a 2D bar code, the operator passes further, internal information about the regions to the operator for extracting the data from a symbol (**Get2DBarCode** or **Get2DBarCodePos**). This information is stored in a region descriptor and returned in the parameter **CodeRegDescr**.

Depending on the method for printing the bar code (`mode`), different image processing steps are performed. Together with other information about the 2D bar code, the actual printing method is part of a descriptor created with the help of the operator **Gen2DBarCodeDescr** and passed in the parameter **BarCodeDescr**.

In the case of difficult conditions, additional control parameters can be passed to the operator with the help of the (optional) generic parameters **GenParamNames** nd **GenParamValues**, in the form of descriptor-value pairs. One group of parameters describes the appearance of the symbols in the actual images, e.g., the expected size of data elements in pixels. The operator will then check candidate regions against the specified criteria. Suitable
values can be determined from the real images, e.g., by manually selecting a region with a bar code and then applying the corresponding operator. If the specification of such parameters does not lead to successfully extracted regions, the user can influence the underlying image processing operators directly with the help of a second group of parameters. As the image processing steps taken depend on the printing method, different parameters are to be applied.

- Parameter that will be passed on in CodeRegDescr:
  - *module_width*  Mean size of a module (data element) in pixels.
    - From this quantity, a set of other parameters will be derived. A module should have a size of 4 to 16 pixels in order to be recognized successfully.
    - GenParamValues: > 0
      - default: 10
  - Parameters for the printing method *printed*:
    - The following parameters are used to limit the set of candidate regions with the help of region features. To prevent the use of a certain feature, the corresponding parameter must be set to -1.
      - *anisometry_max*  Maximum anisometry (see operator Eccentricity).
        - GenParamValues: > 1  (-1: feature not used)
        - default: 1.45
      - *compactness_min*  Minimum compactness (see operator Compactness).
        - GenParamValues: > 1  (-1: feature not used)
        - default: 1.2
      - *compactness_max*  Maximum compactness.
        - GenParamValues: ≥ 'compactness_min'  (-1: feature not used)
        - default: 3.0
      - *circularity_max*  Maximum circularity (see operator Circularity).
        - GenParamValues: 0...1.0  (-1: feature not used)
        - default: 0.8
      - *circularity_min*  Minimum circularity.
        - GenParamValues: ≤ 'circularity_max'  (-1: feature not used)
        - default: 0.45
      - *deviation_min*  Minimum deviation of gray values (see operator Intensity).
        - GenParamValues: > 1.0  (-1: feature not used)
        - default: 20.0

The following parameters are automatically derived from the size of a module, *module_width*, and from parameters of the bar code descriptor BarCodeDescr (see operator Gen2DBarCodeDescr). However, they can be adjusted individually, too.

ATTENTION: If you want to set *module_width* together with other parameters, make sure set *module_width* BEFORE the others!

- *mean_mask_size_1*  Mask size for the first smoothing of the image.
  - GenParamValues: ≥ 3
  - default: 2.5 * 'module_width'

- *mean_mask_size_2*  Mask size for the second smoothing of the image.
  - GenParamValues: ≥ 5
  - default: 10.0 * 'module_width'

- *area_min*  Minimum size of the symbol in pixels squared.
  - GenParamValues: > 0
  - default: 0.25 * 'module_width'² * 'columns_min' * 'rows_min'

- *area_max*  Maximum size of the symbol in pixels squared.
  - GenParamValues: ≥ 'area_min'
  - default: 4.00 * 'module_width'² * 'columns_min' * 'rows_min'
'closing_mask_rad'  Mask size for calling the operator ClosingCircle with a test region.

\[
\text{GenParamValues: } > 0
\]
default: 'module_width'

- Parameters for the printing method 'engraved_darkfield':
  'compactness_min'  Minimum compactness (see operator Compactness).

\[
\text{GenParamValues: } > 1.0  \quad (-1: \text{feature not used})
\]
default: 1.2

'compactness_max'.  

\[
\text{GenParamValues: } \geq 'compactness_min'  \quad (-1: \text{feature not used})
\]
default: 3.0

'edge_thresh'  Maximum compactness.

\[
\text{GenParamValues: } 0 \ldots 255
\]
default: 120

'region_rect2_rel'  Relation between the areas of the region and the smallest surrounding rectangle.

\[
\text{GenParamValues: } < 1.0
\]
default: 0.7

'median_mask_rad'  Mask size for the initial median filtering (see operator MedianImage).

\[
\text{GenParamValues: } > 0
\]
default: 0.1 * 'module_width' + 0.5

- Parameters for the printing method 'engraved_lightfield':
  'anisometry_max'  Maximum anisometry (see operator Eccentricity).

\[
\text{GenParamValues: } > 1  \quad (-1: \text{feature not used})
\]
default: 1.45

'compactness_min'  Minimum compactness (see operator Compactness).

\[
\text{GenParamValues: } > 1  \quad (-1: \text{feature not used})
\]
default: 1.2

'compactness_max'  Maximum compactness.

\[
\text{GenParamValues: } \geq 'compactness_min'  \quad (-1: \text{feature not used})
\]
default: 3.0

'circularity_max'  Maximum circularity (see operator Circularity).

\[
\text{GenParamValues: } 0 \ldots 1.0  \quad (-1: \text{feature not used})
\]
default: 0.8

'circularity_min'  Minimum circularity.

\[
\text{GenParamValues: } \leq 'circularity_max'  \quad (-1: \text{feature not used})
\]
default: 0.45

'deviation_min'  Minimum deviation of gray values (see operator Intensity).

\[
\text{GenParamValues: } > 1.0  \quad (-1: \text{feature not used})
\]
default: 20.0

'mean_mask_size'  Mask size for smoothing the image.

\[
\text{GenParamValues: } \geq 3
\]
default: 10 * 'module_width'

'area_min'  Minimum size of the symbol in pixels squared.

\[
\text{GenParamValues: } > 0
\]
default: 0.25 * 'module_width'^2 * 'columns_min' * 'rows_min'

'area_max'  Maximum size of the symbol in pixels squared.

\[
\text{GenParamValues: } \geq 'area_min'
\]
default: 4.00 * 'module_width'^2 * 'columns_min' * 'rows_min'
13.3. BARCODE

'closing_mask_rad' Mask size for calling the operator ClosingCircle with a test region.

```
GenParamValues: > 0
  default: 'module_width'
```

'opening_mask_rad' Mask size for calling the operator OpeningCircle with a test region.

```
GenParamValues: > 0
  default: 1.5 * 'module_width'
```

---

**Image** (input iconic) ................................. image  \( \sim \) HImageX / HOBJECTX

Image of one or more bar codes.

**CodeRegion** (output iconic) ......................... region(-array)  \( \sim \) HRegionX / HUntypedObjectX

Regions that might contain a bar code.

**BarCodeDescr** (input control) ...................... barcode_2d  \( \sim \) VARIANT (string, integer, real)

Description of a 2D bar code class to look for.

**GenParamNames** (input control) ...................... string(-array)  \( \sim \) VARIANT (string)

List of names of (optional) generic parameters controlling the image processing.

Default Value: []

List of values: GenParamNames \( \in \) {'module_width', 'anisometry_max', 'compactness_min', 'compactness_max', 'circularity_min', 'circularity_max', 'deviation_min', 'mean_mask_size', 'mean_mask_size1', 'mean_mask_size2', 'area_min', 'area_max', 'closing_mask_rad', 'opening_mask_rad', 'compactness_min', 'compactness_max', 'edge_thresh', 'region_rect2_rel', 'median_mask_rad'}

**GenParamValues** (input control) ...................... number(-array)  \( \sim \) VARIANT (integer, real, string)

List of values of generic parameters controlling the image processing.

Default Value: []

List of values: GenParamValues \( \in \) {1, 1.1, 1.2, 1.3, 1.4, 1.5, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20}

**CodeRegDescr** (output control) ...................... string  \( \sim \) VARIANT (string, integer, real)

Additional parameters describing the bar code regions. They can be used for extracting the data (see Decode2DBarCode).

---

**Result**

The operator Find2DBarCode returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

---

**Parallelization Information**

Find2DBarCode is reentrant and processed without parallelization.

---

**Possible Predecessors**

Gen2DBarCodeDescr

---

**Possible Successors**

Get2DBarCode, Get2DBarCodePos

---

**Module**

Barcode reader

---

```cpp
void HOperatorSetX.Gen1DBarCodeDescr ( [ in ] VARIANT CodeName,
[ in ] VARIANT MinCharacters, [ in ] VARIANT MaxCharacters,
[ out ] VARIANT BarCodeDescr )
```

Generate a description of a 1D bar code.

Gen1DBarCodeDescr generates a description of a one dimensional bar code. This description is used for the search (Find1DBarCode or Find1DBarCodeRegion) and the decoding (Decode1DBarCode) of the bar code. Gen1DBarCodeDescr is therefore the first operator in a program sequence for bar code processing. Gen1DBarCodeDescr has only to be called once at the beginning of a program. The descriptor can be used multiple times. In the case of different types of bar codes the operator has to be called once for each type.

HALCON 6.1.4
You have to be aware that this description contains only basic informations about the bar code. Thus an arbitrary description can be used to extract almost every type of bar code. On the other hand a specific description is important for the decoding of a bar code type.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CodeName</strong> (input control)</td>
<td>Name of bar code.</td>
</tr>
<tr>
<td><strong>MinCharacters</strong> (input control)</td>
<td>Minimum number of characters (if not fixed). Default Value: 6 Suggested values: MinCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30}</td>
</tr>
<tr>
<td><strong>MaxCharacters</strong> (input control)</td>
<td>Maximum number of characters (if not fixed). Default Value: 10 Suggested values: MaxCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 50}</td>
</tr>
</tbody>
</table>
| **BarCodeDescr** (output control) | Description of a bar code class. Result The operator Gen1DBarCodeDescr returns the value TRUE if the transfered bar code name is correct. Parallelization Information Gen1DBarCodeDescr is reentrant and processed without parallelization. Possible Successors Find1DBarCode, Find1DBarCodeRegion Alternatives Gen1DBarCodeDescrGen Module Barcode reader

---

Generate a generic description of a 1D bar code. Gen1DBarCodeDescrGen generates a generic description of a one dimensional bar code. This operator is used if the bar code description cannot be generated by Gen1DBarCodeDescr but nonetheless has to be found by using the operators Find1DBarCode or Find1DBarCodeRegion. Note that in this case the bar code cannot be decoded by Decode1DBarCode. Thus in this case the decoding has to be done by the user.

The values for DiscreteCode and ElementSizes have to be set correctly in any case. The other values are not so important for the extraction because they can be estimated by the system automatically.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MinCodeLength</strong> (input control)</td>
<td>Minimum length of the code in modules (including start and stop elements). Default Value: 30 List of values: MinCodeLength ∈ {-1, 30, 60, 90, 110, 130, 150, 200}</td>
</tr>
</tbody>
</table>

---

HALCON/COM Reference Manual, 2005-2-1
13.3. BARCODE

- **MaxCodeLength** (input control) ........................................... integer $\sim$ long / VARIANT
  Maximum length of the code in modules (including start and stop elements).
  Default Value: 30
  List of values: MaxCodeLength $\in$ {-1, 30, 60, 90, 110, 130, 150, 200}

- **ElementSizes** (input control) ........................................... integer $\sim$ long / VARIANT
  Number of different element sizes.
  Default Value: 2
  List of values: ElementSizes $\in$ {1, 2, 3, 4, 5, 6, 7}

- **StartElement** (input control) ........................................... integer(-array) $\sim$ VARIANT (integer)
  List of elements of the start sequence. The width of an element is given as the number of modules. Gaps are
given as negative values.
  Default Value: [1,-1]

- **StopElement** (input control) ........................................... integer(-array) $\sim$ VARIANT (integer)
  List of elements of the stop sequence. The width of an element is given as the number of modules. Gaps are
given as negative values.
  Default Value: [1,-1]

- **MaxSizeRatio** (input control) ........................................... number $\sim$ double / VARIANT
  Maximum ratio length to height.
  Default Value: 2.5
  List of values: MaxSizeRatio $\in$ {-1, 2, 3, 4, 5, 6}

- **DiscreteCode** (input control) ........................................... string $\sim$ String / VARIANT
  Discrete code (ignore white elements).
  Default Value: 'false'
  List of values: DiscreteCode $\in$ {'true', 'false'}

- **BarCodeDescr** (output control) ........................................... barcode_Id $\sim$ VARIANT (string, integer, real)
  Description of a bar code class.

The operator Gen1DBarCodeDescrGen returns the value TRUE if the transfered values are correct.

---

Parallelization Information

Gen1DBarCodeDescrGen is reentrant and processed without parallelization.

Possible Successors

Find1DBarCode, Find1DBarCodeRegion

Alternatives

Gen1DBarCodeDescr

---

Module

Barcode reader

```c
([in] String CodeType, [in] VARIANT GenParamNames,
[in] VARIANT GenParamValues )

void HOperatorSetX.Gen2DBarCodeDescr ([in] VARIANT CodeType,
[in] VARIANT GenParamNames, [in] VARIANT GenParamValues,
[out] VARIANT BarCodeDescr )
```

Generate a generic description of a 2D bar code class.

Gen2DBarCodeDescr creates a generic description of a 2D bar code. This description consists of the type of
the bar code and further characteristic features of the code.

The parameter CodeType specifies the type of the 2D bar code. Currently, the following types are supported:

'Data Matrix ECC 200' Data Matrix type ECC 200.

The bar code can be described more closely using the parameters GenParamNames and GenParamValues,
which form pairs of descriptors and values. The following parameters describe the size and the form of the symbols:
'columns'  Exact number of columns of the symbol.
  GenParamValues:  10, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40,  
  44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144
  default: -

'columns_min'  Minimum number of columns.
  GenParamValues:  10...144 (even)
  default:  10

'columns_max'  Maximum number of columns.
  GenParamValues:  10...144 (even)

'rows'  Exact number of rows of the symbol.
  GenParamValues:  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40,  
  44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144
  default: -

'rows_min'  Minimum number of rows.
  GenParamValues:  8...144 (even)
  default:  8

'rows_max'  Maximum number of rows.
  GenParamValues:  8...144 (even)
  default:  144

'shape'  Form of the symbol.
  GenParamValues:  'square', 'rectangle', 'any'
  default:  'any'

The following parameters describe the printing method and the appearance of the bar code:

'foreground'  Appearance of bits corresponding to a logical 1 (foreground).
  2D bar codes can be printed dark on light or vice versa. If the appearance of bits corresponding to a logical 1 can be specified in advance, computing time will be significantly reduced, especially when looking for regions that might contain a bar code (operator Find2DBarCode).
  GenParamValues:  'any':  unknown appearance  
  'dark','black':  symbols are printed dark on light  
  'light','white':  symbols are printed light on dark
  default:  'any'

'mode'  Method used for printing the 2D bar code.
  By specifying this parameter, image processing can be optimized.
  GenParamValues:  'printed':  Standard black-and-white printing. Modules appear as square regions; neighboring modules with the same value form one region.  
  'engraved_darkfield':  Modules with the value 1 are engraved into the surface, e.g. by a laser. Using dark field lighting these modules appear as dark round dots with a white corona; neighboring modules with the value 1 are therefore separated by gaps.  
  'engraved_lightfield':  Similar to 'engraved_darkfield', but using light field lighting. Modules with the value 1 therefore appear as light dots with a dark corona; neighboring modules with the value 1 are again separated by gaps.
  default:  'printed'
13.3. BARCODE

Parameter

- **CodeType** (input control) ........................................ string \( \sim \) String / VARIANT
  Type of the 2D bar code.
  Default Value : 'Data Matrix ECC 200'
  List of values : CodeType \( \in \{ \) 'Data Matrix ECC 200' \( \} \)

- **GenParamNames** (input control) ................................... string \( \sim \) VARIANT ( string )
  List of names of generic parameters describing the 2D bar code class.
  Default Value : []
  List of values : GenParamNames \( \in \{ \) 'columns', 'columns_min', 'columns_max', 'rows', 'rows_min', 'rows_max', 'foreground', 'shape', 'mode' \( \} \)

- **GenParamValues** (input control) ................................. integer \( \sim \) VARIANT ( integer, string, real )
  List of values of the generic parameters describing the 2D bar code class.
  Default Value : []
  List of values : GenParamValues \( \in \{ \) 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 32, 36, 40, 44, 48, 52, 64, 72, 80, 88, 96, 104, 120, 132, 144, 'dark', 'light', 'square', 'rectangle', 'printed', 'engraved_darkfield', 'engraved_lightfield', 'any' \( \} \)

- **BarCodeDescr** (output control) ................................. bar code_2d \( \sim \) VARIANT ( string, integer, real )
  Description of the 2D bar code class.

Result

The operator **Gen2DBarCodeDescr** returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

Parallelization Information

**Gen2DBarCodeDescr** is reentrant and processed without parallelization.

Possible Successors

Find2DBarCode

Barcode reader

```
[out] VARIANT BarCodeElements HImageX.Get1DBarCode
([in] VARIANT BarCodeDescr, [in] VARIANT GenericName,

[out] VARIANT BarCodeElements HBarcodeX.Get1DBarCode
([in] HImageX BarCodeRegion, [in] VARIANT BarCodeDescr,
[in] VARIANT GenericName, [in] VARIANT GenericValue,
[in] double Orientation )

void HOperatorSetX.Get1DBarCode ([in] IObjectX BarCodeRegion,
[in] VARIANT BarCodeDescr, [in] VARIANT GenericName,
[in] VARIANT GenericValue, [in] VARIANT Orientation,
[out] VARIANT BarCodeElements )
```

Extract the widths of the elements inside a bar code region.

**Get1DBarCode** extracts the widths of the elements of a bar code inside the specified region.

The control of the processing is identical to **Find1DBarCode**. The description of the parameters **GenericName** and **GenericValue** can be found at this operator.

Parameter

- **BarCodeRegion** (input iconic) ................................. image \( \sim \) HImageX / IObjectX ( byte, uint2 )
  Region of bar code.

- **BarCodeDescr** (input control) ................................. bar code_1d \( \sim \) VARIANT ( string, integer, real )
  Description of a bar code class.

- **GenericName** (input control) ................................. string(-array) \( \sim \) VARIANT ( string )
  Names of optional parameters.
  Default Value : []
  List of values : GenericName \( \in \{ \) 'sigma_project', 'amplitude_project', 'width_project', 'add_length_project', 'interpolation_project', 'num_scanlines', 'max_extra_elements' \( \} \)

HALCON 6.1.4
CHAPTER 13. TOOLS

- **GenericValue** (input control) ......................... number(-array) \(\sim\) VARIANT (integer, real)
  Values of optional parameters.
  **Default Value:** []

- **Orientation** (input control) .......................... angle.rad \(\sim\) double / VARIANT
  Orientation of bar code.

- **BarCodeElements** (output control) .................... number \(\sim\) VARIANT (real)
  Widths of elements.

---

The operator **Get1DBarCode** returns the value TRUE if the parameters are correct.

---

**Parallelization Information**

Get1DBarCode is reentrant and processed without parallelization.

---

**Possible Predecessors**

Find1DBarCodeRegion, SelectObj, ReduceDomain

---

**Possible Successors**

Decode1DBarCode

---

**Alternatives**

Find1DBarCode

---

**Module**

Barcode reader

---

Extract the values of the data elements (in ECC 200: "modules") inside a bar code region ("Data Matrix symbol"). **Get2DBarCode** extracts the 2D bar code from a candidate symbol region given in **BarCodeRegion** and the corresponding image given in **Image**. **BarCodeDescr** contains a descriptor of the expected bar code class, which is created with the help of the operator **Gen2DBarCodeDescr**. **CodeRegDescr** contains further information regarding the candidate region. This descriptor is created by the operator **Find2DBarCode**.

If **BarCodeRegion** contains a 2D bar code, its data, i.e. the bits of the matrix, are written line by line into a tuple and returned in the parameter **BarCodeData**. Positive values stand for a logical one, negative values for a logical zero. A value of zero signals that the data element could not be classified unambiguously.

The size of the data field is returned in the parameter **BarCodeDimension**. The first two entries specify the width and the height of the data field, the third contains a so-called symbol index which also describes its size. The size of the data field is not to be confused with the size of the symbol which in turn encompasses additional finder and alignment patterns.

If no 2D bar code could be extracted the operator returns a corresponding error code.

In difficult cases, the user can influence image processing with the help of the (optional) generic parameters **GenParamNames** and **GenParamValues**. The width of a module in pixels ("module_width") does not need to
be specified, it is part of the region descriptor CodeRegDescr. As in the case of the operator Find2DBarCode, for the different printing methods different image processing steps are applied, leading to different parameters.

As a short overview, image processing falls into the following parts: First, the region BarCodeRegion is divided into a dark and a light region by applying a thresholding operator. Because there must be a “silent” area around each 2D bar code, this area can be examined to determine whether the bar code is printed black-on-white or vice versa. With this information, the dark and the light region can be assigned to the logical data values. Next, BarCodeRegion is approximated by a rectangle. Then, the special finder patterns that each each symbol must contain are searched for in the border areas of the rectangle. From the finder pattern, the size of the data matrix and the size and the position of the modules (data elements) can be determined. Finally, the value of the modules is determined.

- Parameters controlling the classification of modules as being dark or light:

  `'module_rad_ratio’` Part of the module that is to be used for its classification.

  This parameter controls which part of the module area (in percent, measured from the center) is used to determine its value. With a `'module_rad_ratio’` of 1.0, the whole area of the module will be examined. With very small `'module_rad_ratio’`, only one point in the middle will be examined.

  GenParamValues: 0 < `'module_rad_ratio’` ≤ 1

  default: printing method `'printed’`, `'engraved_lightfield’": 0.3
  printing method `'engraved_darkfield’": 0.7

  `'use_grayvals’` Method for the classification of modules (Selectable only for the printing methods `'printed’` and `'engraved_lightfield’`; for the printing method `'engraved_darkfield’`, the gray-value-based method is always used!).

  The parameter `'use_grayvals’` controls by which method the modules are classified. If `'use_grayvals’` is set to ‘no’, the module’s value is determined by examining whether the part of the module specified by `'module_rad_ratio’` is part of the region that has been assigned the value 1. If more than half of the module is part of this region, the module is interpreted as a logical one.

  In the case `'use_grayvals’` is set to ‘yes’, the gray values inside `'module_rad_ratio’` are examined. For this, the known border parts of the symbol are used to create classifiers with the features “minimum gray value”, “maximum gray value”, and “range of gray values”. A feature will be automatically excluded if it cannot separate the known dark and light regions. It can also be excluded manually with the help of the parameters below. The value of the modules is then determined by applying the trained classifiers. In case of ambiguous results, the majority decision is adopted.

  If none of the gray value features can separate the test regions, the operator switches automatically to the region-based method in case the printing method is set to `'printed’`. This can be detected by examining the data values returned in BarCodeData: The region-based method yields multiples of 4, the gray-value-based method values between -3 and +3. If separation fails when the printing method is `'engraved_darkfield’`, an error code is returned. Note, that classification performance can be enhanced by choosing a suitable value for `'module_rad_ratio’`.

  GenParamValues: ‘yes’, ‘no’

  default: ‘yes’

  `'use_grayval_min’` Use the feature “gray value minimum” for classifying module values.

  GenParamValues: ‘yes’, ‘no’

  default: ‘yes’

  `'use_grayval_max’` Use the feature “gray value maximum” for classifying module values.

  GenParamValues: ‘yes’, ‘no’

  default: ‘printed’, ‘engraved_lightfield’: ‘yes’

  ‘engraved_darkfield’: ‘no’

  `'use_grayval_range’` Use the feature “gray value range” for classifying module values.

  GenParamValues: ‘yes’, ‘no’

  default: ‘yes’

- Parameters for the printing methods `'printed’` and `'engraved_lightfield’`:

  `'enlarge_region_rad’` Enlarge the symbol region.
'thresh_percent' Area of the histogram in which to look for an optimal threshold.
The threshold for separating dark and light regions will be searched for in the middle 'thresh_percent' percent of the gray value histogram. If 'thresh_percent' is set to 0, a threshold exactly in the middle of the histogram is selected. If 'thresh_percent' is set to 100, the whole histogram will be searched. Note, that the threshold influences not only the size of the data region, but also the position of the surrounding rectangle and with it, slightly, the position of the modules.

\[ \text{GenParamValues: } > 0 \ (\leq 0: \text{enlargment prohibited}) \]
\[ \text{default: } 2.5 \]

'thresh_step_width' Step width for the search of the optimal threshold.
The histogram area specified in the parameter 'thresh_percent' will be stepped through using 'thresh_step_width' until an optimal separation of dark and light regions is reached. The optimum is declared to be found when the number of individual regions is at its maximum. The smaller 'thresh_step_width' is, the better a threshold can be determined, however at the cost of rising computing time, of course.

\[ \text{GenParamValues: } > 0 \]
\[ \text{default: } 50 \]

'smooth_cont' Parameter for approximating the region by a polygon (rectangle).
The parameter 'smooth_cont' controls to which degree the contour is smoothed before approximating it with a polygon (see operator \text{SegmentContoursXld}).

\[ \text{GenParamValues: } 0,3,5,7,9,\ldots \]
\[ \text{default: } 5 \]

'max_line_dist1', 'max_line_dist2' Distance between the contour and its approximation.
The parameters 'max_line_dist1' and 'max_line_dist2' control the maximal distance between the contour and its approximation (see operator \text{SegmentContoursXld}).

\[ \text{GenParamValues: } \geq 0 \]
\[ \text{default: } 4 \]

'min_cont_len' Minimum length of the contour elements in pixel.
Contour elements shorter than 'min_cont_len' will be ignored when approximating the region by a rectangle. In the case of small symbols or low resolution, it can be helpful to decrease this parameter.

\[ \text{GenParamValues: } > 0 \]
\[ \text{default: } 50 \]

'border_width_min' Minimum width of the border search area.
The border area of the symbol region must contain the so-called finder patterns (ECC 200: two lines with the value 1, two with alternating values). To find these patterns, a search region is constructed along the sides of the rectangle. Starting with 'border_width_min', the width of the search region is increased until the finder patterns have been detected, or until 'border_width_max' has been reached.

\[ \text{GenParamValues: } > 0 \]
\[ \text{default: } 2 \]

'border_width_max' Maximum width of the border search area.

\[ \text{GenParamValues: } \geq \ 'border_width_min' \]
\[ \text{default: } 5 \]

'border_width' Exact width of the border search area.
Alternatively, the exact width of this search area can be specified in the parameter 'border_width'. This automatically sets 'border_width_min' and 'border_width_max'.

\[ \text{GenParamValues: } > 0 \]
\[ \text{default: } - \]

Parameters for the printing method 'engraved\_darkfield':
### 13.3. BARCODE

**median_mask_rad**  
Mask size for the initial median filtering (see operator `MedianImage`).  
- **GenParamValues**: 3,5,...,501  
- **default**: `'module_width'*0.1 + 0.5

**gray_erosion_size**  
Mask size for the subsequent gray value erosion (see operator `GrayErosionRect`).  
- **GenParamValues**: 3,5,...,511  
- **default**: `'module_width'*0.18 + 3.5

**measure_sigma**  
Parameter for the extraction of linear edge segment pairs (see operator `MeasurePairs`).  
- **GenParamValues**: 0.4...100  
- **default**:  
  - `'module_width' < 20`: 0.7  
  - `'module_width' ≥ 20`: 1.4

**measure_thresh**  
Parameter for the extraction of linear edge segment pairs (see operator `MeasurePairs`).  
- **GenParamValues**: 1...255  
- **default**: 2.0

---

**Attention**  
The operator `Get2DBarCode` returns error codes to signal that the candidate region did not contain a valid 2D bar code. As the region candidates extracted by the operator `Find2DBarCode` can perfectly well encompass regions without a bar code, every call to `Get2DBarCode` should be surrounded by error handling code (see example).

---

#### Parameter

- **BarCodeRegion** (input iconic)  
  - Description: Region that might contain a 2D bar code.
- **Image** (input iconic)  
  - Description: Corresponding image.
- **BarCodeDescr** (input control)  
  - Description: Description of the bar code class.
- **CodeRegDescr** (input control)  
  - Description: Additional parameters describing the bar code region. They can be used for extracting the data.
- **GenParamNames** (input control)  
  - Description: List of names of (optional) generic control parameters.
- **GenParamValues** (input control)  
  - Description: List of values of the generic parameters.
- **BarCodeDimension** (output control)  
  - Description: Tuple with the dimension of the extracted symbol. In the case of ECC 200: data field width, height, symbol index.
- **BarCodeData** (output control)  
  - Description: Tuple with the data values of the extracted symbol. value > 0: logical 1, value < 0: logical 0, value = 0: module could not be classified.

---

**Result**  
The return value can signal incorrect parameters as well as the failure to extract a 2D bar code. Such a failure can be due to different causes: If no surrounding rectangle was found, the operator returns the error code 8808. Error code 8807 signals that the rectangle does not lie completely inside the image. If no finder pattern was detected inside the border search area, the error code 8810 is returned. If the two continuous lines forming the ‘L’ of the finder pattern could not be found, the error code 8809 is returned. In the case of the printing method 'engraved darkfield', the...
error code 8811 signals that none of the gray value features can separate the test regions, i.e. that modules cannot be classified.

Parallelization Information
Get2DBarCode is reentrant and processed without parallelization.

Possible Predecessors
Find2DBarCode

Possible Successors
Decode2DBarCode

Alternatives
Get2DBarCodePos

Module
Barcode reader

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BarCodeRegion</td>
<td>region in HRegionX / IObjectX that might contain a bar code.</td>
</tr>
<tr>
<td>Image</td>
<td>image in HImageX / IObjectX (byte) corresponding image.</td>
</tr>
<tr>
<td>BarCodeDescr</td>
<td>barcode_2d in VARIANT (string, integer, real) description of the 2D bar code class.</td>
</tr>
</tbody>
</table>

Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.

Get2DBarCodePos extracts the 2D bar code data from a candidate symbol region given in BarCodeRegion and the corresponding image given in Image elements. In contrast to Get2DBarCode, this operator also returns the positions of the data elements in the image in the parameters DataElementRow and DataElementCol. This can be used to check the result of the operator, e.g., by displaying the data in the real image. For a description of the other parameters, see Get2DBarCode.

Attention
The operator Get2DBarCodePos returns error codes to signal that the candidate region did not contain a valid 2D bar code. As the region candidates extracted by the operator Find2DBarCode can perfectly well encompass regions without a bar code, every call to Get2DBarCodePos should be surrounded by error handling code (see example).
13.4. CALIBRATION

- **CodeRegDescr** (input control) .......................... string  \(\sim\) VARIANT (string, integer, real)
  Additional parameters describing the bar code region. They can be used for extracting the data.

- **GenParamNames** (input control) ........................... string  \(\sim\) VARIANT (string)
  List of names of (optional) generic control parameters.
  **Default Value:** []
  **List of values:** GenParamNames \(\in\) \{'module\_rad\_ratio', 'use\_grayvals', 'use\_grayval\_min', 'use\_grayval\_max', 'use\_grayval\_range', 'enlarge\_region\_rad', 'thresh\_percent', 'thresh\_step\_width', 'smooth\_cont', 'max\_line\_dist1', 'max\_line\_dist2', 'min\_cont\_len', 'border\_width\_min', 'border\_width\_max', 'border\_width', 'median\_mask\_rad', 'gray\_erosion\_size', 'measure\_sigma', 'measure\_thresh'\}

- **GenParamValues** (input control) ......................... number  \(\sim\) VARIANT (integer, real)
  List of values of the generic parameters.
  **Default Value:** []
  **Suggested values:** GenParamValues \(\in\) \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 'yes', 'no', -1, 1.5, 2.5, 3.5, 4.5, 5.5, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 30, 40, 50\}

- **BarCodeDimension** (output control) ........................ integer  \(\sim\) VARIANT (integer)
  Tuple with the dimension of the extracted symbol. In the case of ECC 200: data field width, height, symbol index.

- **BarCodeData** (output control) .......................... integer  \(\sim\) VARIANT (integer)
  Tuple with the data values of the extracted symbol. value > 0: logical 1, value < 0: logical 0, value = 0: module could not be classified.

- **DataElementRow** (output control) ........................ real  \(\sim\) VARIANT (real)
  Tuple with the row positions of the data elements of the extracted symbol in the image.

- **DataElementCol** (output control) ........................ real  \(\sim\) VARIANT (real)
  Tuple with the column positions of the data elements of the extracted symbol in the image.

---

**Result**

The return value can signal incorrect parameters as well as the failure to extract a 2D bar code. Such a failure can be due to different causes: If no surrounding rectangle was found, the operator returns the error code 8808. Error code 8807 signals that the rectangle does not lie completely inside the image. If no finder pattern was detected inside the border search area, the error code 8810 is returned. If the two continuous lines forming the 'L' of the finder pattern could not be found, the error code 8809 is returned. In the case of the printing method 'engraved\_darkfield', the error code 8811 signals that none of the gray value features can separate the test regions, i.e., that modules cannot be classified.

---

**Parallelization Information**

Get2DBarCodePos is reentrant and processed without parallelization.

---

**Possible Predecessors**

Find2DBarCode

---

**Possible Successors**

Decode2DBarCode

---

See also

---

**Alternatives**

Get2DBarCode

---

Module

---

Barcode reader

---

**HALCON 6.1.4**

---

**13.4 Calibration**

```
[out] VARIANT X HMiscX.CaltabPoints ( [in] String CalTabDescrFile, 
[out] VARIANT Y, [out] VARIANT Z )
void HOperatorSetX.CaltabPoints ( [in] VARIANT CalTabDescrFile, 
```

Read the mark center points from the calibration plate description file.
CaltabPoints reads the mark center points from the calibration plate description file CalTabDescrFile (see CreateCaltab) and returns their coordinates in X, Y and Z. The mark center points are 3D coordinates in the calibration plate coordinate system and describe the 3D model of the calibration plate. The calibration plate coordinate system is located in the middle of the surface of the calibration plate, its z-axis points into the calibration plate, its x-axis to the right, and its y-axis downwards.

The mark center points are typically used as input parameters for the operator CameraCalibration. This operator projects the model points into the image, minimizes the distance between the projected points and the observed 2D coordinates in the image (see FindMarksAndPose), and from this computes the exact values for the interior and exterior camera parameters.

**Parameter**

- **CalTabDescrFile** (input control) .................. string  ~ String / VARIANT
  File name of the calibration plate description.
  **Default Value** : 'caltab.descr'
- **X** (output control) .......................... real  ~ VARIANT( real )
  X coordinates of the mark center points in the coordinate system of the calibration plate.
- **Y** (output control) .......................... real  ~ VARIANT( real )
  Y coordinates of the mark center points in the coordinate system of the calibration plate.
- **Z** (output control) .......................... real  ~ VARIANT( real )
  Z coordinates of the mark center points in the coordinate system of the calibration plate.

**Example**

```plaintext
* read_image(Image1, 'calib-01')
* find calibration pattern
find_caltab(Image1, Caltab1, 'caltab.descr', 3, 112, 5)
* find calibration marks and start poses
StartCamPar := [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576]
find_marks_and_pose(Image1,Caltab1,'caltab.descr', StartCamPar,
                     128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1,
                     StartPose1)
* read 3D positions of calibration marks
caltab_points('caltab.descr', NX, NY, NZ) >
* camera calibration
camera_calibration(NX, NY, NZ, RCoord1, CCoord1, StartCamPar,
                     StartPose1, 'all', CamParam, FinalPose, Errors)
* visualize calibration result
disp_image(Image1, WindowHandle)
disp_caltab('caltab.descr', CamParam, FinalPose, 1.0)
```

**Result**

CaltabPoints returns TRUE if all parameter values are correct and the file CalTabDescrFile has been read successfully. If necessary, an exception handling is raised.

**Parallelization Information**

CaltabPoints is **reentrant** and processed without parallelization.

**Possible Successors**

CameraCalibration

**See also**

FindCaltab, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, Project3DPoint, GetLineOfSight, CreateCaltab

**Module**

Camera calibration
Determine all camera parameters by a simultaneous minimization process.

CameraCalibration performs the calibration of a camera. For this, known 3D model points (with coordinates \(NX, NY, NZ\)) are projected in the image and the sum of the squared distance between these projections and the corresponding image points (with coordinates \(NRow, NCol\)) is minimized.

If the minimization converges, the exact interior (\(CamParam\)) and exterior (\(NFinalPose\)) camera parameters are determined by this minimization algorithm. The parameters \(StartCamParam\) and \(NStartPose\) are used as initial values for the minimization process. Since this algorithm simultaneously handles correspondences between image and model points from different images, it is also called multi-image calibration.

In general, camera calibration means the exact determination of the parameters that model the (optical) projection of any 3D world point \(P^w\) into a (sub-)pixel \([r,c]\) in the image. This is important, if the original 3D pose of an object has to be computed using an image (e.g., measuring of industrial parts).

**Used 3D camera model**

The projection consists of multiple steps: First, the point \(p^w\) is transformed from world into camera coordinates (points as homogeneous vectors, compare AffineTransPoint3D):

\[
\begin{pmatrix}
 p^c \\
 1
\end{pmatrix} =
\begin{pmatrix}
 x \\
 y \\
 z \\
 1
\end{pmatrix} =
\begin{bmatrix}
 R & t \\
 0 & 0 & 0 & 1
\end{bmatrix}
\begin{pmatrix}
 p^w \\
 1
\end{pmatrix}
\]

Then, the point is projected into the image plane, i.e., onto the sensor chip. If the underlying camera model is a pinhole camera with radial distortions, i.e., if the focal length passed in \(StartCamParam\) is greater than 0, the projection is described by the following equations:

\[
u = \text{Focus} \cdot \frac{x}{z} \quad \text{and} \quad v = \text{Focus} \cdot \frac{y}{z}
\]

In contrast, if the focal length is passed as 0 in \(StartCamParam\), the camera model of a telecentric camera with radial distortions is used, i.e., it is assumed that the optics of the lens of the camera performs a parallel projection. In this case, the corresponding equations are:

\[u = x \quad \text{and} \quad v = y\]

The following equations compensate for radial distortion:

\[
\begin{align*}
\tilde{u} &= \frac{2u}{1 + \sqrt{1 - 4\kappa(u^2 + v^2)}} \\
\tilde{v} &= \frac{2v}{1 + \sqrt{1 - 4\kappa(u^2 + v^2)}}
\end{align*}
\]

Finally, the point is transformed from the image plane coordinate system into the image coordinate system, i.e., the pixel coordinate system:

\[
\begin{align*}
c &= \frac{\tilde{u}}{S_x} + C_x \\
r &= \frac{\tilde{v}}{S_y} + C_y
\end{align*}
\]
Camera parameters

The total of 14 camera parameters can be divided into the interior and exterior camera parameters:

Interior camera parameters: These parameters describe the characteristics of the used camera, especially the dimension of the sensor itself and the projection properties of the used combination of lens, camera, and frame grabber. The camera model (as described above) contains the following 8 parameters:

Focus: Focal length of the lens. 0 for telecentric lenses.
Kappa ($\kappa$): Distortion coefficient to model the pillow- or barrel-shaped distortions caused by the lens.
Sx: Scale factor, corresponds to the horizontal distance between two neighboring cells on the sensor. Attention: This value increases, if the image is subsampled!
Sy: Scale factor, corresponds to the vertical distance between two neighboring cells on the sensor. Since in most cases the image signal is sampled line-synchronously, this value is determined by the dimension of the sensor and needn’t be estimated for pinhole cameras by the calibration process. Attention: This value increases, if the image is subsampled!
Cx: Column coordinate of the image center point (center of the radial distortion).
Cy: Row coordinate of the image center point (center of the radial distortion).
ImageWidth: Width of the sampled image. Attention: This value decreases, if the image is subsampled!
ImageHeight: Height of the sampled image. Attention: This value decreases, if the image is subsampled!

Exterior camera parameters: These 6 parameters describe the 3D pose, i.e., the position and orientation, of the world coordinate system relative to the camera coordinate system. Three parameters describe the translation, three the rotation. See CreatePose for more information about 3D poses. Note that CameraCalibration operates with all types of 3D poses for NStartPose.

When using the standard calibration plate, the world coordinate system is defined by the coordinate system of the calibration plate, which is located in the middle of the surface of the calibration plate, its z-axis pointing into the calibration plate, its x-axis to the right, and its y-axis downwards.

Additional information about the calibration process

The use of CameraCalibration leads to some questions, which are dealt with in the following sections:

How to generate a appropriate calibration plate? The simplest method to determine the interior parameters of a camera is the use of the standard calibration plate as generated by the operator CreateCaltab. You can obtain high-precision calibration plates in various sizes and materials from your local distributor. In case of small distances between object and lens it may be sufficient to print the calibration pattern by a laser printer and to mount it on a cardboard. Otherwise – especially by using a wide-angle lens – it is possible to print the PostScript file on a large ink-jet printer and to mount it on a aluminum plate. It is very important that the mark coordinates in the calibration plate description file correspond to the real ones on the calibration plate with high accuracy. Thus, the calibration plate description file has to be modified in accordance with the measurement of the calibration plate!

How to take a set of suitable images? If you use the standard calibration plate, you can proceed in the following way: With the combination of lens (fixed distance!), camera, and frame grabber to be calibrated a set of images of the calibration plate has to be taken, see OpenFramegrabber and GrabImage. The following items have to be considered:

- At least a total of 10 to 20 images should be taken into account.
- The calibration plate has to be completely visible (incl. border!).
- Reflections etc. on the calibration plate should be avoided.
- Within the set of images the calibration plate should appear in different positions and orientations: Once left in the image, once right, once (left and right) at the bottom, once (left or right) at the top, from different distances etc. At this, the calibration plate should be rotated around its x- and/or y-axis, so the perspective distortions of the calibration pattern are clearly visible. Thus, the exterior camera parameters (camera pose with regard of the calibration plate) should be set to a large variety of different values!
- The calibration plate should fill at least a quarter of the whole image to ensure the robust detection of the marks.

How to extract the calibration marks in the images? If a standard calibration plate is used, you can use the operators FindCaltab and FindMarksAndPose to determine the coordinates of the calibration marks in each image and to compute a rough estimate for the exterior camera parameters. The concatenation of
these values can directly be used as initial values for the exterior camera parameters (NStartPose) in CameraCalibration.

Obviously, images in which the segmentation of the calibration plate (FindCaltab) has failed or the calibration marks haven’t been determined successfully by FindMarksAndPose should not be used.

**How to find suitable initial values for the interior camera parameters?** The operators FindMarksAndPose (determination of initial values for the exterior camera parameters) and CameraCalibration require initial values for the interior camera parameters. These parameters can be provided by an appropriate text file (see ReadCamPar), which can be generated by WriteCamPar or can be edited manually.

The following should be considered for the initial values of the single parameters:

**Focus:** The initial value is the nominal focal length of the used lens, e.g., 0.008 m.

**Kappa:** Use 0.0 as initial value.

**Sx:** The initial value for the horizontal distance between two neighboring cells depends on the dimension of the used chip of the camera (see technical specifications of the camera). Generally, common chips are either 1/3"-Chips (e.g., SONY XC-73, SONY XC-777), 1/2"-Chips (e.g., SONY XC-999, Panasonic WV-CD50), or 2/3"-Chips (e.g., SONY DXC-151, SONY XC-77). Notice: The value of Sx increases if the image is subsampled! Appropriate initial values are:

<table>
<thead>
<tr>
<th>Chip Type</th>
<th>Full Imaging (768*576)</th>
<th>Subsampling (384*288)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3&quot;-Chip</td>
<td>0.0000055 m</td>
<td>0.0000011 m</td>
</tr>
<tr>
<td>1/2&quot;-Chip</td>
<td>0.0000086 m</td>
<td>0.00000172 m</td>
</tr>
<tr>
<td>2/3&quot;-Chip</td>
<td>0.0000110 m</td>
<td>0.00000220 m</td>
</tr>
</tbody>
</table>

The value for Sx is calibrated, since the video signal of a camera normally isn’t sampled pixel-synchronously.

**Sy:** Since most off-the-shelf cameras have quadratic pixels, the same values for Sy are valid as for Sx. In contrast to Sx the value for Sy will not be calibrated for pinhole cameras, because the video signal of a camera normally is sampled line-synchronously. Thus, the initial value is equal to the final value. Appropriate initial values are:

<table>
<thead>
<tr>
<th>Chip Type</th>
<th>Full Imaging (768*576)</th>
<th>Subsampling (384*288)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3&quot;-Chip</td>
<td>0.0000055 m</td>
<td>0.0000011 m</td>
</tr>
<tr>
<td>1/2&quot;-Chip</td>
<td>0.0000086 m</td>
<td>0.00000172 m</td>
</tr>
<tr>
<td>2/3&quot;-Chip</td>
<td>0.0000110 m</td>
<td>0.00000220 m</td>
</tr>
</tbody>
</table>

**Cx and Cy:** Initial values for the coordinates of the image center is the half image width and half image height. Notice: The values of Cx and Cy decrease if the image is subsampled! Appropriate initial values are:

<table>
<thead>
<tr>
<th>Image Type</th>
<th>Full Imaging (768*576)</th>
<th>Subsampling (384*288)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cx</td>
<td>384.0</td>
<td>192.0</td>
</tr>
<tr>
<td>Cy</td>
<td>288.0</td>
<td>144.0</td>
</tr>
</tbody>
</table>

**ImageWidth and ImageHeight:** These two parameters are determined by the used frame grabber and therefore are not calibrated. Appropriate initial values are, for example:

<table>
<thead>
<tr>
<th>Image Type</th>
<th>Full Imaging (768*576)</th>
<th>Subsampling (384*288)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageWidth</td>
<td>768</td>
<td>384</td>
</tr>
<tr>
<td>ImageHeight</td>
<td>576</td>
<td>288</td>
</tr>
</tbody>
</table>

**Which camera parameters have to be estimated?** The input parameter EstimateParams is used to select which camera parameters to estimate. Usually this parameter is set to 'all', i.e., all 6 exterior camera parameters (translation and rotation) and all interior camera parameters are determined. If the interior camera parameters already have been determined (e.g., by a previous call to CameraCalibration) it is often desired to only determine the pose of world coordinate system in camera coordinates (i.e., the exterior camera parameters). In this case, EstimateParams can be set to 'pose'. This has the same effect as EstimateParams = ['alpha','beta','gamma','transx','transy','transz']. Otherwise, EstimateParams contains a tuple of strings indicating the combination of parameters to estimate.

**What is the order within the individual parameters?** The length of the tuple NStartPose corresponds to the number of calibration images, e.g., using 15 images leads to a length of the tuple NStartPose equal to...
15 · 7 = 105 (15 times the 7 exterior camera parameters). The first 7 values correspond to the pose of the calibration plate in the first image, the next 7 values to the pose in the second image, etc.

This fixed number of calibration images has to be considered within the tuples with the coordinates of the 3D model marks and the extracted 2D marks. If 15 images are used, the length of the tuples \( \text{NRow} \) and \( \text{NCol} \) is 15 times the length of the tuples with the coordinates of the 3D model marks (\( \text{NX}, \text{NY}, \) and \( \text{NZ} \)). If every image consists of 49 marks, the length of the tuples \( \text{NRow} \) and \( \text{NCol} \) is 15 · 49 = 735, while the length of the tuples \( \text{NX}, \text{NY}, \) and \( \text{NZ} \) is 49. The order of the values in \( \text{NRow} \) and \( \text{NCol} \) is "image after image", i.e., using 49 marks the first 3D model point corresponds to the 1st, 50th, 99th, 148th, 197th, 246th, etc. extracted 2D mark.

The 3D model points can be read from a calibration plate description file using the operator \text{CaltabPoints}. Initial values for the poses of the calibration plate can be determined by applying \text{FindMarksAndPose} for each image. The tuple \( \text{NStartPose} \) is set by the concatenation of all these poses.

**What is the meaning of the output parameters?** If the camera calibration process is finished successfully, i.e., the minimization process has converged, the output parameters \( \text{CamParam} \) and \( \text{NFinalPose} \) contain the computed exact values for the interior and exterior camera parameters. The length of the tuple \( \text{NFinalPose} \) corresponds to the length of the tuple \( \text{NStartPose} \).

The representation types of \( \text{NFinalPose} \) correspond to the representation type of the first tuple of \( \text{NStartPose} \) (see \text{CreatePose}). You can convert the representation type by \text{ConvertPoseType}.

The computed average errors (\( \text{Errors} \)) give an impression of the accuracy of the calibration. The error values (deviations in x and y coordinates) are measured in pixels.

**Must I use a planar calibration object?** No. The operator \text{CameraCalibration} is designed in a way that the input tuples \( \text{NX}, \text{NY}, \text{NZ}, \text{NRow}, \) and \( \text{NCol} \) can contain any 3D/2D correspondences, see the above paragraph explaining the order of the single parameters.

Thus, it makes no difference how the required 3D model marks and the corresponding extracted 2D marks are determined. On one hand, it is possible to use a 3D calibration pattern, on the other hand, you also can use any characteristic points (natural landmarks) with known position in the world. By setting \text{EstimateParams} to 'pose', it is thus possible to compute the pose of an object in camera coordinates! For this, at least three 3D/2D-correspondences are necessary as input. \( \text{NStartPose} \) can, e.g., be generated directly as shown in the program example for \text{CreatePose}.

---

**Attention**

The minimization process of the calibration depends on the initial values of the interior (\( \text{StartCamParam} \)) and exterior (\( \text{NStartPose} \)) camera parameters. The computed average errors \( \text{Errors} \) give an impression of the accuracy of the calibration. The errors (deviations in x and y coordinates) are measured in pixels.

**Parameter**

- \( \text{NX} \) (input control) real \( \sim \) \text{VARIANT} (real) Ordered tuple with all x coordinates of the calibration marks (in meters).
- \( \text{NY} \) (input control) real \( \sim \) \text{VARIANT} (real) Ordered tuple with all y coordinates of the calibration marks (in meters).
- \( \text{NZ} \) (input control) real \( \sim \) \text{VARIANT} (real) Ordered tuple with all z coordinates of the calibration marks (in meters).
- \( \text{NRow} \) (input control) real \( \sim \) \text{VARIANT} (real) Ordered tuple with all row coordinates of the extracted calibration marks (in pixels).
- \( \text{NCol} \) (input control) real \( \sim \) \text{VARIANT} (real) Ordered tuple with all column coordinates of the extracted calibration marks (in pixels).
- \( \text{StartCamParam} \) (input control) real \( \sim \) \text{VARIANT} (integer, real) Initial values for the interior camera parameters.
- \( \text{NStartPose} \) (input control) pose \( \sim \) \text{VARIANT} (real, integer) Ordered tuple with all initial values for the exterior camera parameters.
- \( \text{EstimateParams} \) (input control) string \( \sim \) \text{VARIANT} (string, integer) Camera parameters to be estimated.

**Default Value:** 'all'

**List of values:** \text{EstimateParams} \( \in \{ \text{all', 'pose', 'alpha', 'beta', 'gamma', 'transx', 'transy', 'transz', 'focus', 'kappa', 'cx', 'cy', 'sx', 'sy' \} \)
13.4. CALIBRATION

- **CamParam** (output control) .......................................................... number  \( \sim \) VARIANT (integer, real)
  Interior camera parameters.
- **NFinalPose** (output control) ....................................................... pose  \( \sim \) VARIANT (real, integer)
  Ordered tuple with all exterior camera parameters.
- **Errors** (output control) .............................................................. real  \( \sim \) VARIANT
  Average error distances in pixels.

---

**Example**

* read calibration images
  ```
  read_image(Image1, ‘calib-01’)
  read_image(Image2, ‘calib-02’)
  read_image(Image3, ‘calib-03’)
  * find calibration pattern
  find_caltab(Image1, Caltab1, ‘caltab.descr’, 3, 112, 5)
  find_caltab(Image2, Caltab2, ‘caltab.descr’, 3, 112, 5)
  find_caltab(Image3, Caltab3, ‘caltab.descr’, 3, 112, 5)
  * find calibration marks and start poses
  find_marks_and_pose(Image1, Caltab1, ‘caltab.descr’,
                      [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576],
                      128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1,
                      StartPose1)
  find_marks_and_pose(Image2, Caltab2, ‘caltab.descr’,
                      [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576],
                      128, 10, 18, 0.9, 15.0, 100.0, RCoord2, CCoord2,
                      StartPose2)
  find_marks_and_pose(Image3, Caltab3, ‘caltab.descr’,
                      [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576],
                      128, 10, 18, 0.9, 15.0, 100.0, RCoord3, CCoord3,
                      StartPose3)
  * read 3D positions of calibration marks
    caltab_points(‘caltab.descr’, NX, NY, NZ)
  * camera calibration
    camera_calibration(NX, NY, NZ, [RCoord1, RCoord2, RCoord3],
                       [CCoord1, CCoord2, CCoord3],
                       [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576],
                       [StartPose1, StartPose2, StartPose3], ‘all’,
                       CamParam, NFinalPose, Errors)
  * write interior camera parameters to file
    write_cam_par(CamParam, ‘campar.dat’)"

---

**Result**

*CameraCalibration* returns TRUE if all parameter values are correct and the desired camera parameters have been determined by the minimization algorithm. If necessary, an exception handling is raised.

---

**Parallelization Information**

*CameraCalibration* is reentrant and processed without parallelization.

---

**Possible Predecessors**

- FindMarksAndPose
- CaltabPoints
- ReadCamPar

---

**Possible Successors**

- WritePose
- PoseToHomMat3d
- DispCaltab
- SimCaltab

---

**See also**

- FindCaltab
- FindMarksAndPose
- SimCaltab
- WriteCamPar
- ReadCamPar
- CreatePose
- ConvertPoseType
- WritePose
- ReadPose
- PoseToHomMat3d
- HomMat3dToPose
- CaltabPoints
- CreateCaltab

---

**Module**

Camera calibration

HALCON 6.1.4
Determine new camera parameters in accordance to the specified radial distortion.

`ChangeRadialDistortionCamPar` modifies the interior camera parameters in accordance to the specified radial distortion `Kappa`. Via `Mode` one of the following modes can be selected:

- **'fixed'**: Only `Kappa` is modified, the other interior camera parameters remain unchanged. In general, this leads to a change of the visible part of the scene.
- **'fullsize'**: The scale factors $S_x$ and $S_y$ and the image center point $[C_x, C_y]^T$ are modified in order to preserve the visible part of the scene. Thus, all points visible in the original image are also visible in the modified (rectified) image. In general, this leads to undefined pixels in the modified image.
- **'adaptive'**: A trade-off between the other modes: The visible part of the scene is slightly reduced to prevent undefined pixels in the modified image. Similarly to 'fullsize', the scale factors and the image center point are modified.

In all modes the radial distortion coefficient $\kappa$ in `CamParOut` is set to `Kappa`. The transformation of a pixel in the modified image into the image plane using `CamParOut` results in the same point as the transformation of a pixel in the original image via `CamParIn`.

```c
[out] VARIANT CamParOut HMiscX.ChangeRadialDistortionCamPar
```

```c
void HOperatorSetX.ChangeRadialDistortionCamPar ([in] VARIANT Mode, 
```

In all modes the radial distortion coefficient $\kappa$ in `CamParOut` is set to `Kappa`. The transformation of a pixel in the modified image into the image plane using `CamParOut` results in the same point as the transformation of a pixel in the original image via `CamParIn`.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode</td>
<td>Input control string / VARIANT</td>
</tr>
<tr>
<td>Default Value</td>
<td>'adaptive'</td>
</tr>
<tr>
<td>Suggested values</td>
<td>$\text{Mode} \in {\text{&quot;fullsize&quot;}, \text{&quot;adaptive&quot;}, \text{&quot;fixed&quot;}}$</td>
</tr>
<tr>
<td>CamParIn</td>
<td>Input control real / VARIANT</td>
</tr>
<tr>
<td>Interior camera parameters (original).</td>
<td></td>
</tr>
<tr>
<td>Kappa</td>
<td>Input control real / VARIANT</td>
</tr>
<tr>
<td>Desired radial distortion.</td>
<td></td>
</tr>
<tr>
<td>Default Value</td>
<td>0.0</td>
</tr>
<tr>
<td>CamParOut</td>
<td>Output control real / VARIANT</td>
</tr>
<tr>
<td>Interior camera parameters (modified).</td>
<td></td>
</tr>
</tbody>
</table>

`ChangeRadialDistortionCamPar` returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

`ChangeRadialDistortionCamPar` is reentrant and processed without parallelization.

Possible Predecessors

CameraCalibration, ReadCamPar

Possible Successors

ChangeRadialDistortionImage, ChangeRadialDistortionContoursXld

See also

CameraCalibration, ReadCamPar, ChangeRadialDistortionImage, ChangeRadialDistortionContoursXld

Module

Camera calibration
13.4. Calibration

```haskell
HLDContX.ChangeRadialDistortionContoursXld
  ([in] VARIANT CamParIn, [in] VARIANT CamParOut)
```

```haskell
void HOperatorSetX.ChangeRadialDistortionContoursXld
  ([in] IHObjectX Contours, [out] HUntypedObjectX ContoursRectified,
   [in] VARIANT CamParIn, [in] VARIANT CamParOut)
```

Change the radial distortion of contours.

ChangeRadialDistortionContoursXld changes the radial distortion of the input contours Contours in accordance to the interior camera parameters CamParIn and CamParOut. Each subpixel of an input contour is transformed into the image plane using CamParIn and subsequently projected into a subpixel of the corresponding contour in ContoursRectified using CamParOut.

If CamParOut was computed via ChangeRadialDistortionCamPar, the contours ContoursRectified are equivalent to Contours obtained with a lens with a modified radial distortion. If \( \kappa \) is 0 the contours are rectified. A subsequent pose estimation (determination of the exterior camera parameters) is not affected by this operation.

Parameter

- **Contours** (input iconic) \( \rightsquigarrow \) HLDContX / IHObjectX
  Original contours.
- **ContoursRectified** (output iconic) \( \rightsquigarrow \) HLDContX / HUntypedObjectX
  Resulting contours with modified radial distortion.
- **CamParIn** (input control) \( \rightsquigarrow \) VARIANT (real)
  Interior camera parameter for Contours.
- **CamParOut** (input control) \( \rightsquigarrow \) VARIANT (real)
  Interior camera parameter for ContoursRectified.

Parallelization Information

ChangeRadialDistortionContoursXld is reentrant and processed without parallelization.

Possible Predecessors

ChangeRadialDistortionCamPar, GenContoursSkeletonXld, EdgesSubPix, SmoothContoursXld

Possible Successors

GenPolygonsXld, SmoothContoursXld

See also

ChangeRadialDistortionCamPar, CameraCalibration, ReadCamPar, ChangeRadialDistortionImage

Module

Camera calibration

```haskell
HImageX.ImageRectified HImageX.ChangeRadialDistortionImage
  ([in] HRegionX Region, [in] VARIANT CamParIn, [in] VARIANT CamParOut)
```

```haskell
void HOperatorSetX.ChangeRadialDistortionImage
  ([in] IHObjectX Image, [in] IHObjectX Region,
   [out] HUntypedObjectX ImageRectified, [in] VARIANT CamParIn,
   [in] VARIANT CamParOut)
```

Change the radial distortion of an image.

ChangeRadialDistortionImage changes the radial distortion of the input image Image in accordance to the interior camera parameters CamParIn and CamParOut. The image size remains unchanged. Each pixel of the output image is transformed into the image plane using CamParOut and subsequently projected into a subpixel of Image using CamParIn. The resulting gray value is determined by bilinear interpolation. If the subpixel is outside of Image, the corresponding pixel in ImageRectified is set to ‘black’ and eliminated from the image domain.
If `CamParOut` was computed via `ChangeRadialDistortionCamPar`, `ImageRectified` is equivalent to `Image` obtained with a lens with a modified radial distortion. If $\kappa$ is 0 the image is rectified. A subsequent pose estimation (determination of the exterior camera parameters) is not affected by this operation.

Parameter

- **Image** (input iconic) ……… (multichannel-)image(-array) $\leadsto$ `HImageX / IHObjectX (byte, uint2)`
  
  Original image.

- **Region** (input iconic) ………………………………….region $\leadsto$ `HRegionX / IHObjectX`
  
  Region of interest in `Image`.

- **ImageRectified** (output iconic) ……… (multichannel-)image(-array) $\leadsto$ `HImageX / HUntypedObjectX (byte, uint2)`
  
  Resulting image with modified radial distortion.

- **CamParIn** (input control) ……………………………real $\leadsto$ `VARIANT (real)`
  
  Interior camera parameter for `Image`.

- **CamParOut** (input control) ……………………………real $\leadsto$ `VARIANT (real)`
  
  Interior camera parameter for `Image`.

Result `ChangeRadialDistortionImage` returns TRUE if all parameter values are correct and the input is not empty. If necessary, an exception handling is raised.

Parallelization Information

`ChangeRadialDistortionImage` is reentrant and processed without parallelization.

Possible Predecessors

- `ChangeRadialDistortionCamPar`, `ReadImage`, `GrabImage`

Possible Successors

- `EdgesImage`, `Threshold`

See also

- `ChangeRadialDistortionCamPar`, `CameraCalibration`, `ReadCamPar`, `ChangeRadialDistortionContoursXld`

Module

Camera calibration

```plaintext
[out] HXLDCntX ContoursTrans

```

Transform an XLD contour into the plane $z=0$ of a world coordinate system.

The operator `ContourToWorldPlaneXld` transforms contour points given in `Contours` into the plane $z=0$ in a world coordinate system and returns the 3D contour points in `ContoursTrans`. The world coordinate system is chosen by passing its 3D pose relative to the camera coordinate system in `WorldPose`. In `CamParam` you must pass the interior camera parameters (see `WriteCamPar` for the sequence of the parameters and the underlying camera model).

In many cases `CamParam` and `WorldPose` are the result of calibrating the camera with the operator `CameraCalibration`. See below for an example.

With the parameter `Scale` you can scale the resulting 3D coordinates. The parameter `Scale` must be specified as the ratio desired unit/original unit. The original unit is determined by the coordinates of the calibration object. If the original unit is meters (which is the case if you use the standard calibration plate), you can set the desired unit directly by selecting 'm', 'cm', 'mm' or 'µm' for the parameter `Scale`.

Internally, the operator first computes the line of sight between the projection center and the image point in the camera coordinate system, taking into account the radial distortions. The line of sight is then transformed into the world coordinate system specified in `WorldPose`. By intersecting the plane $z=0$ with the line of sight the 3D coordinates of the transformed contour `ContoursTrans` are obtained.
13.4. CALIBRATION

Parameter

- **Contours** (input iconic) ........................ xld_cont(-array)  $\rightarrow$ HXLDContX / HObjectX
  Input XLD contours to be transformed in image coordinates.
- **ContoursTrans** (output iconic) ................. xld_cont(-array)  $\rightarrow$ HXLDContX / HUntypedObjectX
  Transformed XLD contours in world coordinates.
- **CamParam** (input control) ........................ number  $\rightarrow$ VARIANT (integer, real)
  Interior camera parameters.
  Number of elements : 8
- **WorldPose** (input control) ........................ pose  $\rightarrow$ VARIANT (integer, real)
  3D pose of the world coordinate system in camera coordinates.
  Number of elements : 7
- **Scale** (input control) .............................. number  $\rightarrow$ VARIANT (integer, real, string)
  Scale oder dimension
  Default Value : 'm'
  Suggested values : Scale $\in \{ 'm', 'cm', 'mm', 'microns', \mu m, 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144 \}$

Example

* perform camera calibration (with standard calibration plate)
camera_calibration(NX, NY, NZ, NRow, NCol, StartCamParam, NStartPose, 'all', FinalCamParam, NFinalPose, Errors)
* world coordinate system is defined by calibration plate in first image
FinalPose1 := NFinalPose[0:6]
* compensate thickness of plate
set_origin_pose(FinalPose1, 0, 0, 0.0006, WorldPose)
* transform contours into world coordinate system (unit mm)
contour_to_world_plane_xld(Contours, ContoursTrans, FinalCamParam, WorldPose, 'mm')

Result

ContourToWorldPlaneXld returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

ContourToWorldPlaneXld is reentrant and processed without parallelization.

Possible Predecessors

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration, SetOriginPose

See also

ImagePointsToWorldPlane

Module

Camera calibration

```
[out] VARIANT PoseOut HPoseX.ConvertPoseType ([in] VARIANT PoseIn,
[in] String OrderOfTransform, [in] String OrderOfRotation,
[in] String ViewOfTransform )
void HOperatorSetX.ConvertPoseType ([in] VARIANT PoseIn,
[in] VARIANT OrderOfTransform, [in] VARIANT OrderOfRotation,
[in] VARIANT ViewOfTransform, [out] VARIANT PoseOut )
```

Change the representation type of a 3D pose.

ConvertPoseType converts the 3D pose PoseIn into a 3D pose PoseOut with a different representation type. See CreatePose for details about 3D poses, their representation types, and the meaning of the parameters OrderOfTransform, OrderOfRotation, and ViewOfTransform.
Note that ConvertPoseType only changes the representation of a 3D pose, but not the rigid transformation described by the pose.

- **PoseIn** (input control) ........................ pose \(\sim\) VARIANT (integer, real)  
  Original 3D pose.  
  **Number of elements:** 7

- **OrderOfTransform** (input control) .................. string \(\sim\) String / VARIANT  
  Order of rotation and translation.  
  **Default Value:** 'Rp+T'  
  **Suggested values:** OrderOfTransform \(\in\) \{'Rp+T', 'R(p-T)\}

- **OrderOfRotation** (input control) .................. string \(\sim\) String / VARIANT  
  Meaning of the rotation values.  
  **Default Value:** 'gba'  
  **Suggested values:** OrderOfRotation \(\in\) \{'gba', 'abg', 'rodriguez'\}

- **ViewOfTransform** (input control) ................. string \(\sim\) String / VARIANT  
  View of transformation.  
  **Default Value:** 'point'  
  **Suggested values:** ViewOfTransform \(\in\) \{'point', 'coordinate system'\}

- **PoseOut** (output control) ............................ pose \(\sim\) VARIANT (integer, real)  
  3D transformation.  
  **Number of elements:** 7

---

**Example**

* get pose (exterior camera parameters):  
  read_pose ('campose.dat', Pose)  
* convert pose to a pose with desired semantic  
  convert_pose_type (Pose, 'Rp+T', 'abg', 'point', Pose2)

---

**Result**

ConvertPoseType returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

---

**Parallelization Information**

ConvertPoseType is reentrant and processed without parallelization.

---

**Possible Predecessors**

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration

---

**Possible Successors**

WritePose

---

**See also**

CreatePose, GetPoseType, WritePose, ReadPose

---

**Module**

Camera calibration

---

```c
void HMiscX.CreateCaltab ([in] double Width,  
[in] String CalTabDescrFile, [in] String CalTabFile )

void HOperatorSetX.CreateCaltab ([in] VARIANT Width,  
[in] VARIANT CalTabDescrFile, [in] VARIANT CalTabFile )
```

Generate a calibration plate description file and a corresponding PostScript file.

CreateCaltab generates the description of a standard calibration plate for HALCON. This calibration plate consists of 49 black circular marks on a white plane, which are surrounded by a black frame. The parameter **Width** sets the width (equal to the height) of the whole calibration plate in meters. Using a width of 0.8 m, the distance between two neighboring marks becomes 10 cm, and the mark radius and the frame width are set to 2.5 cm. The calibration plate coordinate system is located in the middle of the surface of the calibration plate, its z-axis points into the calibration plate, its x-axis to the right, and it y-axis downwards.
The file \texttt{CalTabDescrFile} contains the calibration plate description, e.g., the number of rows and columns of the calibration plate, the geometry of the surrounding frame (see \texttt{FindCaltab}), and the coordinates and the radius of all calibration plate marks given in the calibration plate coordinate system. A file generated by \texttt{CreateCaltab} looks like the following (comments are marked by a \texttt{"#"} at the beginning of a line):

```
# Description of the standard calibration plate
# used for the camera calibration in HALCON
#
# 7 rows X 7 columns
# Distance between mark centers [meter]: 0.1
#
# Number of marks per row
r 7
#
# Number of marks per column
c 7
#
# Quadratic frame (with outer and inner border) around calibration plate
w 0.025
o -0.41 0.41 0.41 -0.41
i -0.4 0.4 0.4 -0.4
#
# calibration marks: x y radius [Meter]
#
# calibration marks at y = -0.3 m
-0.3 -0.3 0.025
-0.2 -0.3 0.025
-0.1 -0.3 0.025
 0 -0.3 0.025
 0.1 -0.3 0.025
 0.2 -0.3 0.025
 0.3 -0.3 0.025
#
# calibration marks at y = -0.2 m
-0.3 -0.2 0.025
-0.2 -0.2 0.025
-0.1 -0.2 0.025
 0 -0.2 0.025
 0.1 -0.2 0.025
 0.2 -0.2 0.025
 0.3 -0.2 0.025
#
# calibration marks at y = -0.1 m
-0.3 -0.1 0.025
-0.2 -0.1 0.025
-0.1 -0.1 0.025
 0 -0.1 0.025
 0.1 -0.1 0.025
 0.2 -0.1 0.025
 0.3 -0.1 0.025
#
# calibration marks at y = 0 m
-0.3 0 0.025
-0.2 0 0.025
-0.1 0 0.025
 0 0 0.025
 0.1 0 0.025
```
The file `CalTabFile` contains the corresponding PostScript description of the calibration plate.

**Attention**

Depending on the accuracy of the used output device (e.g., laser printer), the printed calibration plate may not match the values in the calibration plate description file `CalTabDescrFile` exactly. Thus, the coordinates of the calibration marks in the calibration plate description file may have to be corrected!

**Parameter**

- **Width** (input control) ......................................................... real $\sim$ double / VARIANT
  Width of the calibration plate in meters.
  Default Value: 0.8
  Suggested values: `Width \in \{1.2, 0.8, 0.6, 0.4, 0.2, 0.1\}`
  Recommended Increment: 0.1
  Restriction: $(0.0 < \text{Width})$

- **CalTabDescrFile** (input control) ........................................ string $\sim$ String / VARIANT
  File name of the calibration plate description.
  Default Value: `caltab.descr`

- **CalTabFile** (input control) ........................................... string $\sim$ String / VARIANT
  File name of the PostScript file.
  Default Value: `caltab.ps`

**Example**

* create calibration plate with width = 80 cm
  `create_caltab(0.8, 'caltab.descr', 'caltab.ps')`

**Result**

`CreateCaltab` returns TRUE if all parameter values are correct and both files have been written successfully. If necessary, an exception handling is raised.

**Parallelization Information**

`CreateCaltab` is processed completely exclusively without parallelization.
Create a 3D pose.

CreatePose creates the 3D pose Pose. A pose describes a rigid 3D transformation, i.e., a transformation consisting of an arbitrary translation and rotation, with 6 parameters: TransX, TransY, and TransZ specify the translation along the x-, y-, and z-axis, respectively, while RotX, RotY, and RotZ describe the rotation.

3D poses are typically used in two ways: First, to describe the position and orientation of one coordinate system relative to another (e.g., the pose of a part’s coordinate system relative to the camera coordinate system - in short: the pose of the part relative to the camera) and secondly, to describe how coordinates can be transformed between two coordinate systems (e.g., to transform points from part coordinates into camera coordinates).

Representation of orientation (rotation)

A 3D rotation around an arbitrary axis can be represented by 3 parameters in multiple ways. HALCON lets you choose between three of them with the parameter OrderOfRotation: If you pass the value ‘gba’, the rotation is described by the following chain of rotations around the three axes (see HomMat3dRotate for the content for the rotation matrices R_x, R_y, and R_z):

\[ R_{gba} = R_x(\text{RotX}) \cdot R_y(\text{RotY}) \cdot R_z(\text{RotZ}) \]

Please note that you can “read” this chain in two ways: If you start from the right, the rotations are always performed relative to the global (i.e., fixed or “old”) coordinate system. Thus, \( R_{gba} \) can be read as follows: First rotate around the z-axis, then around the “old” y-axis, and finally around the “old” x-axis. In contrast, if you read from the left to the right, the rotations are performed relative to the local (i.e., “new”) coordinate system. Then, \( R_{gba} \) corresponds to the following: First rotate around the x-axis, then the “new” y-axis, and finally around the “new(est)” z-axis.

Reading \( R_{gba} \) from right to left corresponds to the following sequence of operator calls:

```
hom_mat3d_identity (HomMat3DIdent)
hom_mat3d_rotate (HomMat3DIdent, RotZ, ’z’, 0, 0, 0, HomMat3DRotZ)
```

When passing ‘abg’ in OrderOfRotation, the rotation corresponds to the following chain:

\[ R_{abg} = R_y(\text{RotZ}) \cdot R_x(\text{RotY}) \cdot R_z(\text{RotX}) \]

If you pass ‘rodriguez’ in OrderOfRotation, the rotation parameters RotX, RotY, and RotZ are interpreted as the x-, y-, and z-component of the so-called Rodriguez rotation vector. The direction of the vector defines the (arbitrary) axis of rotation. The length of the vector usually defines the rotation angle with positive orientation.
Here, a variation of the Rodriguez vector is used, where the length of the vector defines the tangent of half the rotation angle:

\[ \mathbf{R}_{\text{rodriguez}} = \text{rotate around } \begin{pmatrix} \text{RotX} \\ \text{RotY} \\ \text{RotZ} \end{pmatrix} \text{ by } 2 \cdot \arctan(\sqrt{\text{RotX}^2 + \text{RotY}^2 + \text{RotZ}^2}) \]

**Corresponding homogeneous transformation matrix**

You can obtain the homogeneous transformation matrix corresponding to a pose with the operator `PoseToHomMat3d`. In the standard definition, this is the following homogeneous transformation matrix, which can be split into two separate matrices, one for the translation \( \mathbf{H}(\mathbf{t}) \) and one for the rotation \( \mathbf{H}(\mathbf{R}) \):

\[
\mathbf{H}_{\text{pose}} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}(\text{RotX, RotY, RotZ}) & \text{TransX} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \text{TransY} \\ \text{TransZ} \end{bmatrix} = \mathbf{H}(\mathbf{t}) \cdot \mathbf{H}(\mathbf{R})
\]

**Transformation of coordinates**

The following equation describes how a point can be transformed from coordinate system 1 into coordinate system 2 with a pose, or more exactly, with the corresponding homogeneous transformation matrix \( ^2\mathbf{H}_1 \) (input and output points as homogeneous vectors, see also `AffineTransPoint3D`). Note that to transform points from coordinate system 1 into system 2, you use the transformation matrix that describes the pose of coordinate system 1 relative to system 2.

\[
\begin{pmatrix} \mathbf{p}_2^g \\ 1 \end{pmatrix} = ^2\mathbf{H}_1 \cdot \begin{pmatrix} \mathbf{p}_1^g \\ 1 \end{pmatrix} = \mathbf{R}(\text{RotX, RotY, RotZ}) \cdot \mathbf{p}_1^g + \begin{pmatrix} \text{TransX} \\ \text{TransY} \\ \text{TransZ} \end{pmatrix}
\]

This corresponds to the following operator calls:

```plaintext```
pose_to_hom_mat3d(PoseOf1In2, HomMat3DFrom1In2)
affine_trans_point_3d(HomMat3DFrom1In2, P1X, P1Y, P1Z, P2X, P2Y, P2Z)
```

**Non-standard pose definitions**

So far, we described the standard pose definition. To create such poses, you select the (default) values \( 'Rp+T' \) for the parameter `OrderOfTransform` and \( 'point' \) for `ViewOfTransform`. By specifying other values for these parameters, you can create non-standard poses types, which we describe briefly below. Please note that these representation types are only supported for backwards compatibility; we strongly recommend to use the standard types.

If you select \( 'R(p-T)' \) for `OrderOfTransform`, the created pose corresponds to the following chain of transformations, i.e., the sequence of rotation and translation is reversed and the translation is negated:

\[
\mathbf{H}_{R(p-T)} = \begin{bmatrix} \mathbf{R}(\text{RotX, RotY, RotZ}) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -\text{TransX} \\ 0 & 1 & -\text{TransY} \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{R}) \cdot \mathbf{H}(-\mathbf{t})
\]

If you select \( 'coordinate_system' \) for `ViewOfTransform`, the sequence of transformations remains constant, but the rotation angles are negated. Please note that, contrary to its name, this is not equivalent to transforming a coordinate system!
13.4. CALIBRATION

\[
H_{\text{Coordinate system}} = \begin{bmatrix}
1 & 0 & 0 & \text{TransX} \\
0 & 1 & 0 & \text{TransY} \\
0 & 0 & 1 & \text{TransZ}
\end{bmatrix} \cdot \begin{bmatrix}
\text{R}(\text{-RotX}, \text{-RotY}, \text{-RotZ}) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Returned data structure

The created 3D pose is returned in \text{Pose}, which is a tuple of length seven. The first three elements hold the translation parameters \text{TransX}, \text{TransY}, and \text{TransZ}. The last element codes the representation type of the pose, which you selected with the parameters \text{OrderOfTransform}, \text{OrderOfRotation}, and \text{ViewOfTransform}. The following table lists the possible combinations. As already noted, we recommend to use only the representation types with \text{OrderOfTransform} = 'Rp+T' and \text{ViewOfTransform} = 'point' (codes 0, 2, and 4).

<table>
<thead>
<tr>
<th>\text{OrderOfTransform}</th>
<th>\text{OrderOfRotation}</th>
<th>\text{ViewOfTransform}</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>'Rp+T'</td>
<td>'gba'</td>
<td>'point'</td>
<td>0</td>
</tr>
<tr>
<td>'Rp+T'</td>
<td>'abg'</td>
<td>'point'</td>
<td>2</td>
</tr>
<tr>
<td>'Rp+T'</td>
<td>'rodriguez'</td>
<td>'point'</td>
<td>4</td>
</tr>
<tr>
<td>'Rp+T'</td>
<td>'gba'</td>
<td>'coordinate system'</td>
<td>1</td>
</tr>
<tr>
<td>'Rp+T'</td>
<td>'abg'</td>
<td>'coordinate system'</td>
<td>3</td>
</tr>
<tr>
<td>'Rp+T'</td>
<td>'rodriguez'</td>
<td>'coordinate system'</td>
<td>5</td>
</tr>
<tr>
<td>'R(p-T)'</td>
<td>'gba'</td>
<td>'point'</td>
<td>8</td>
</tr>
<tr>
<td>'R(p-T)'</td>
<td>'abg'</td>
<td>'point'</td>
<td>10</td>
</tr>
<tr>
<td>'R(p-T)'</td>
<td>'rodriguez'</td>
<td>'point'</td>
<td>12</td>
</tr>
<tr>
<td>'R(p-T)'</td>
<td>'gba'</td>
<td>'coordinate system'</td>
<td>9</td>
</tr>
<tr>
<td>'R(p-T)'</td>
<td>'abg'</td>
<td>'coordinate system'</td>
<td>11</td>
</tr>
<tr>
<td>'R(p-T)'</td>
<td>'rodriguez'</td>
<td>'coordinate system'</td>
<td>13</td>
</tr>
</tbody>
</table>

You can convert poses into other representation types using \text{ConvertPoseType} and query the type using \text{GetPoseType}.

- **TransX** (input control) .................................................. real \sim double / VARIANT
  Translation along the x-axis (in [m]).
  Default Value : 0.1
  Suggested values : \text{TransX} \in \{-1.0, -0.75, -0.5, -0.25, -0.2, -0.1, -0.5, -0.25, -0.125, -0.01, 0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0\}

- **TransY** (input control) .................................................. real \sim double / VARIANT
  Translation along the y-axis (in [m]).
  Default Value : 0.1
  Suggested values : \text{TransY} \in \{-1.0, -0.75, -0.5, -0.25, -0.2, -0.1, -0.5, -0.25, -0.125, -0.01, 0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0\}

- **TransZ** (input control) .................................................. real \sim double / VARIANT
  Translation along the z-axis (in [m]).
  Default Value : 0.1
  Suggested values : \text{TransZ} \in \{-1.0, -0.75, -0.5, -0.25, -0.2, -0.1, -0.5, -0.25, -0.125, -0.01, 0, 0.01, 0.125, 0.25, 0.5, 0.1, 0.2, 0.25, 0.5, 0.75, 1.0\}

- **RotX** (input control) .................................................. real \sim double / VARIANT
  Rotation around x-axis or x component of the Rodriguez vector (in [°] or without unit).
  Default Value : 90
  Suggested values : \text{RotX} \in \{90, 180, 270\}
  Typical range of values : 0 \leq \text{RotX} \leq 0

- **RotY** (input control) .................................................. real \sim double / VARIANT
  Rotation around y-axis or y component of the Rodriguez vector (in [°] or without unit).
  Default Value : 90
  Suggested values : \text{RotY} \in \{90, 180, 270\}
  Typical range of values : 0 \leq \text{RotY} \leq 0
RotZ (input control) ..................................................... real \sim double / VARIANT
Rotation around z-axis or z component of the Rodriguez vector (in ° or without unit).

Default Value: 90
Suggested values: RotZ \in \{90, 180, 270\}
Typical range of values: 0 \leq RotZ \leq 0

OrderOfTransform (input control) ........................................ string \sim String / VARIANT
Order of rotation and translation.

Default Value: ’Rp+T’
Suggested values: OrderOfTransform \in \{’Rp+T’, ’R(p-T)’\}

OrderOfRotation (input control) ........................................ string \sim String / VARIANT
Meaning of the rotation values.

Default Value: ’gba’
Suggested values: OrderOfRotation \in \{’gba’, ’abg’, ’rodriguez’\}

ViewOfTransform (input control) ..................................... string \sim String / VARIANT
View of transformation.

Default Value: ’point’
Suggested values: ViewOfTransform \in \{’point’, ’coordinate_system’\}

Pose (output control) ................................................. pose \sim VARIANT (integer, real)
3D pose.

Number of elements: 7

--- Example ---

* goal: calibration with non-standard calibration object
* read start values for interior camera parameters
read_cam_par(’campar.dat’, CamParam)
* (read 3D world points [WorldPointsX, WorldPointsY, WorldPointsZ],
* extract corresponding 2D image points [PixelsRow, PixelsColumn])
* task: create starting value for the exterior camera parameters, i.e., the
  * pose of the calibration object in the calibration images
* first image: calibration object placed at a distance of 0.5 and 0.1
  * ’below’ the camera coordinate system
  * orientation ’read from left to right’: rotated 30 degrees
  * around the optical axis of the camera (z-axis),
  * then tilted 10 degrees around the new y-axis
create_pose(0.1, 0.0, 0.5, 30, 10, 0, ’Rp+T’, ’abg’, ’point’, StartPose1)
* (accumulate all poses in StartPoses = [StartPose1, StartPose2, ...])
* perform the calibration
camera_calibration(WorldPointsX, WorldPointsY, WorldPointsZ,
  PixelsRow, PixelsColumn, CamParam, StartPoses, ’pose’,
  FinalCamParam, FinalPoses, Errors)

--- Result ---
CreatePose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information
CreatePose is reentrant and processed without parallelization.

Possible Successors
PoseToHomMat3d, WritePose, CameraCalibration, HandEyeCalibration

See also
HomMat3dRotate, HomMat3dTranslate, ConvertPoseType, GetPoseType, HomMat3dToPose,
PoseToHomMat3d, WritePose, ReadPose

Alternatives
ReadPose, HomMat3dToPose

Module
Camera calibration
Project and visualize the 3D model of the calibration plate in the image.

`DispCaltab` is used to visualize the calibration marks and the connecting lines between the marks of the used calibration plate (`CalTabDescrFile`) in the window specified by `WindowHandle`. For this, the 3D model of the calibration plate is projected into the image plane using the interior (`CamParam`) and exterior camera parameters (`CaltabPose`, i.e., the pose of the calibration plate in camera coordinates). The underlying camera model (pinhole or telecentric camera with radial distortion) is described in `WriteCamPar` and `CameraCalibration`.

Typically, `DispCaltab` is used to verify the result of the camera calibration (see `CameraCalibration`) by superimposing it onto the original image. The current line width can be set by `SetLineWidth`, the current color can be set by `SetColor`.

The parameter `ScaleFac` influences the number of supporting points to approximate the elliptic contours of the calibration marks. You should increase the number of supporting points, if the image part in the output window is displayed with magnification (see `SetPart`).

Parameter

- **WindowHandle** (input control) ................................. window ~ HWindowX / VARIANT
  Window in which the calibration plate should be visualized.
- **CalTabDescrFile** (input control) ................................. string ~ String / VARIANT
  File name of the calibration plate description.
  Default Value: 'caltab.descr'
- **CamParam** (input control) ................................. number ~ VARIANT ( integer, real )
  Interior camera parameters.
  Number of elements: 8
- **CaltabPose** (input control) ................................. pose ~ VARIANT ( integer, real )
  Exterior camera parameters (3D pose of the calibration plate in camera coordinates).
  Number of elements: 7
- **ScaleFac** (input control) ...................................... real ~ double / VARIANT
  Scaling factor for the visualization.
  Default Value: 1.0
  Suggested values: ScaleFac ∈ {0.5, 1.0, 2.0, 3.0}
  Recommended Increment: 0.05
  Restriction: (0.0 < ScaleFac)

Example

```c
* read calibration image
read_image(Image1, 'calib-01')
* find calibration pattern
find_caltab(Image1, Caltab1, 'caltab.descr', 3, 112, 5)
* find calibration marks and start poses
StartCamPar := [Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]
find_marks_and_pose(Image1, Caltab1, 'caltab.descr', StartCamPar,
                   128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1,
                   StartPose1)
* read 3D positions of calibration marks
caltab_points('caltab.descr', NX, NY, NZ)
* camera calibration
camera_calibration(NX, NY, NZ, RCoord1, CCoord1, StartCamPar,
                    StartPose1, 11, CamParam, FinalPose, Errors)
* visualize calibration result
disp_image(Image1, WindowHandle)
set_color(WindowHandle, 'red')
disp_caltab('caltab.descr', CamParam, FinalPose, 1.0)
```

HALCON 6.1.4
**DispCaltab** returns TRUE if all parameter values are correct. If necessary, an exception handling is raised. **Parallelization Information**

**DispCaltab** is **local** and processed **completely exclusively** without parallelization.

**Possible Predecessors**

CameraCalibration, ReadCamPar, ReadPose

See also

FindMarksAndPose, CameraCalibration, SimCaltab, WriteCamPar, ReadCamPar, CreatePose, WritePose, ReadPose, Project3DPoint, GetLineOfSight

**Module**

Camera calibration

Segment the standard calibration plate region in the image.

**FindCaltab** is used to determine the region of a plane calibration plate with circular marks in the input image **Image**. First the input image is smoothed (see **GaussImage**); the size of the used filter mask is given by **SizeGauss**. Afterwards, a thresholding operator (see **Threshold**) with minimum gray value **MarkThresh** and maximum gray value 255 is applied. Among the extracted connected regions the most convex region with almost correct number of holes (corresponding to the dark marks of the calibration plate) is selected. Holes with a diameter smaller than the expected size of the marks **MinDiamMarks** are eliminated to reduce the impact of noise. The number of marks is read from the calibration plate description file **CalTabDescrFile**. The complete explanation of this file can be found within the description of **CreateCaltab**.

**Parameter**

- **Image** (input iconic)  
  Input image.

- **Caltab** (output iconic)  
  Output region.

- **CalTabDescrFile** (input control)  
  File name of the calibration plate description.

  **Default Value:** 'caltab.descr'

- **SizeGauss** (input control)  
  Filter size of the Gaussian.

  **Default Value:** 3

  **List of values:** SizeGauss ∈ {0, 3, 5, 7, 9, 11}

- **MarkThresh** (input control)  
  Threshold value for mark extraction.

  **Default Value:** 112

  **List of values:** MarkThresh ∈ {48, 64, 80, 96, 112, 128, 144, 160}

- **MinDiamMarks** (input control)  
  Expected minimal diameter of the marks on the calibration plate.

  **Default Value:** 5

  **List of values:** MinDiamMarks ∈ {3, 5, 9, 15, 30, 50, 70}

**Example**

```plaintext
* read calibration image
read_image(Image, 'calib-01')
* find calibration pattern
find_caltab(Image, Caltab, 'caltab.descr', 3, 112, 5)
```
FindCaltab returns TRUE if all parameter values are correct and an image region is found. The behavior in case of empty input (no image given) can be set via SetSystem('noObjectResult',<Result>): and the behavior in case of an empty result region via SetSystem('storeEmptyRegion',<true/false>:). If necessary, an exception handling is raised.

FindCaltab is reentrant and processed without parallelization.

Possible Predecessors
ReadImage

Possible Successors
FindMarksAndPose

See also
FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, CaltabPoints, CreateCaltab

Camera calibration

 Extract the 2D calibration marks from the image and calculate initial values for the exterior camera parameters.
FindMarksAndPose is used to determine the necessary input data for the subsequent camera calibration (see CameraCalibration): First, the 2D center points RCoord, CCoord of the calibration marks within the region CalTabRegion of the input image Image are extracted and ordered. Secondly, a rough estimate for the exterior camera parameters (StartPose) is computed, i.e., the 3D pose (= position and orientation) of the calibration plate relative to the camera coordinate system (see CreatePose for more information about 3D poses).

In the input image Image an edge detector is applied (see EdgesImage, mode 'lanser2') to the region CalTabRegion, which can be found by applying the operator FindCaltab. The filter parameter for this edge detection can be tuned via Alpha. In the edge image closed contours are searched for: The number of closed contours must correspond to the number of calibration marks as described in the calibration plate description file CalTabDescrFile and the contours have to be elliptically shaped. Contours shorter than MinContLength are discarded, just as contours enclosing regions with a diameter larger than MaxDiamMarks (e.g., the border of the calibration plate).

For the detection of contours a threshold operator is applied to amplitude of the edge detector. All points with a high amplitude (i.e., borders of marks) are selected.

First, the threshold value is set to StartThresh. If the search for the closed contours or the successive pose estimate fails, this threshold value is successively decreased by DeltaThresh down to a minimum value of MinThresh.

Each of the found contours is refined with subpixel accuracy (see EdgesSubPix) and subsequently approximated by an ellipse. The center points of these ellipses represent a good approximation of the desired 2D image coordinates [RCoord,CCoord] of the calibration mark center points. The order of the values within these tuples is in row-major order beginning at the upper left in the image. This order must correspond to the order of the 3D coordinates of the calibration marks in the calibration plate description file CalTabDescrFile, since this fixes the correspondences between extracted image marks and known model marks!
Based on the ellipse parameters for each calibration mark, a rough estimate for the exterior camera parameters is finally computed. For this purpose the fixed correspondences between extracted image marks and known model marks are used. The estimate \texttt{StartPose} describes the pose of the calibration plate in the camera coordinate system as required by the operator \texttt{CameraCalibration}.

\begin{verbatim}
    Parameter
    \begin{itemize}
        \item \texttt{Image} (input iconic) \(\leadsto\) \texttt{HImageX / IObjectX} (byte, uint2)
            Input image.
        \item \texttt{CalTabRegion} (input iconic) \(\leadsto\) \texttt{HRegionX / IObjectX}
            Region of the calibration plate.
        \item \texttt{CalTabDescrFile} (input control) \(\leadsto\) \texttt{String / VARIANT}
            File name of the calibration plate description.
            \textbf{Default Value:} \texttt{’caltab.descr’}
        \item \texttt{StartCamParam} (input control) \(\leadsto\) \texttt{VARIANT} (integer, real)
            Initial values for the interior camera parameters.
        \item \texttt{StartThresh} (input control) \(\leadsto\) \texttt{VARIANT} (long)
            Initial threshold value for contour detection.
            \textbf{Default Value:} 128
            \textbf{List of values:} \texttt{StartThresh} \(\in\) \{80, 96, 112, 128, 144, 160\}
        \item \texttt{DeltaThresh} (input control) \(\leadsto\) \texttt{VARIANT} (long)
            Loop value for successive reduction of \texttt{StartThresh}.
            \textbf{Default Value:} 10
            \textbf{List of values:} \texttt{DeltaThresh} \(\in\) \{6, 8, 10, 12, 14, 16, 18, 20, 22\}
        \item \texttt{MinThresh} (input control) \(\leadsto\) \texttt{VARIANT} (long)
            Minimum threshold for contour detection.
            \textbf{Default Value:} 18
            \textbf{List of values:} \texttt{MinThresh} \(\in\) \{8, 10, 12, 14, 16, 18, 20, 22\}
        \item \texttt{Alpha} (input control) \(\leadsto\) \texttt{VARIANT} (double)
            Filter parameter for contour detection, see \texttt{EdgesImage}.
            \textbf{Default Value:} 0.9
            \textbf{Suggested values:} \texttt{Alpha} \(\in\) \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1\}
            \textbf{Typical range of values:} 0.2 \(\leq\) \texttt{Alpha} \(\leq\) 0.2
            \textbf{Restriction:} \((\texttt{Alpha} > 0.0)\)
        \item \texttt{MinContLength} (input control) \(\leadsto\) \texttt{VARIANT} (double)
            Minimum length of the contours of the marks.
            \textbf{Default Value:} 15.0
            \textbf{List of values:} \texttt{MinContLength} \(\in\) \{10.0, 15.0, 20.0, 30.0, 40.0, 100.0\}
            \textbf{Restriction:} \((\texttt{MinContLength} > 0.0)\)
        \item \texttt{MaxDiamMarks} (input control) \(\leadsto\) \texttt{VARIANT} (double)
            Maximum expected diameter of the marks.
            \textbf{Default Value:} 100.0
            \textbf{List of values:} \texttt{MaxDiamMarks} \(\in\) \{50.0, 100.0, 150.0, 200.0, 300.0\}
            \textbf{Restriction:} \((\texttt{MaxDiamMarks} > 0.0)\)
        \item \texttt{RCoord} (output control) \(\leadsto\) \texttt{VARIANT} (real)
            Tuple with row coordinates of the detected marks.
        \item \texttt{CCoord} (output control) \(\leadsto\) \texttt{VARIANT} (real)
            Tuple with column coordinates of the detected marks.
        \item \texttt{StartPose} (output control) \(\leadsto\) \texttt{VARIANT} (pose)
            Estimation for the exterior camera parameters.
            \textbf{Number of elements:} 7
    \end{itemize}
\end{verbatim}

\begin{verbatim}
* read calibration image
read_image(Image, ‘calib-01’)
* find calibration pattern
find_caltab(Image, Caltab1, ‘caltab.descr’, 3, 112, 5)
* find calibration marks and start pose
\end{verbatim}

HALCON/COM Reference Manual, 2005-2-1
find_marks_and_pose(Image, Caltab, ‘caltab.descr’,
[0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576],
128, 10, 18, 0.9, 15.0, 100.0, RCoord, CCoord, StartPose)

Result

FindMarksAndPose returns TRUE if all parameter values are correct and an estimation for the exterior camera parameters has been determined successfully. If necessary, an exception handling is raised.

Parallelization Information

FindMarksAndPose is reentrant and processed without parallelization.

Possible Predecessors

FindCaltab

Possible Successors

CameraCalibration

See also

FindCaltab, CameraCalibration, DispCaltab, SimCaltab, ReadCamPar, ReadPose,
CreatePose, PoseToHomMat3d, CaltabPoints, CreateCaltab, EdgesSubPix, EdgesImage

Module

Camera calibration

void HImageX.GenImageToWorldPlaneMap ([in] VARIANT CamParam,
[in] VARIANT WorldPose, [in] long WidthIn, [in] long HeightIn,
[in] long WidthMapped, [in] long HeightMapped, [in] VARIANT Scale,
[in] String Interpolation )

[out] HImageX Map HPoseX.GenImageToWorldPlaneMap
([in] VARIANT CamParam, [in] VARIANT WorldPose, [in] long WidthIn,
[in] long HeightIn, [in] long WidthMapped, [in] long HeightMapped,
[in] VARIANT Scale, [in] String Interpolation )

void HOperatorSetX.GenImageToWorldPlaneMap
([out] HUntypedObjectX Map, [in] VARIANT CamParam, [in] VARIANT WorldPose,
[in] VARIANT WidthIn, [in] VARIANT HeightIn, [in] VARIANT WidthMapped,

Generate a projection map that describes the mapping between the image plane and a the plane z=0 of a world coordinate system.

GenImageToWorldPlaneMap generates a projection map Map, which describes the mapping between the image plane and the plane z=0 (plane of measurements) in a world coordinate system. This map can be used to rectify an image with the operator MapImage. The rectified image shows neither radial nor perspective distortions; it corresponds to an image acquired by a distortion-free camera that looks perpendicularly onto the plane of measurements. The world coordinate system is chosen by passing its 3D pose relative to the camera coordinate system in WorldPose. In CamParam you must pass the interior camera parameters (see WriteCamPar for the sequence of the parameters and the underlying camera model).

In many cases CamParam and WorldPose are the result of calibrating the camera with the operator CameraCalibration. See below for an example.

The size of the images to be mapped can be specified by the parameters WidthIn and HeightIn. The pixel position of the upper left corner of the output image is determined by the origin of the world coordinate system. The size of the output image can be choosen by the parameters WidthMapped, HeightMapped, and Scale. WidthMapped and HeightMapped must be given in pixels.

With the parameter Scale you can specify the size of a pixel in the transformed image. There are two typical scenarios: First, you can scale the image such that pixel coordinates in the transformed image directly correspond to metric units, e.g., that one pixel corresponds to one micron. This is useful if you want to perform measurements in the transformed image, which will then directly result in metric results. The second scenario is to scale the image such that its content appears in a size similar to the original image. This is useful, e.g., if you want to perform shape-based matching in the transformed image.

HALCON 6.1.4
Scale must be specified as the ratio desired pixel size/original unit. A pixel size of 1µm means that a pixel in the transformed image corresponds to the area 1µm × 1µm in the plane of measurements. The original unit is determined by the coordinates of the calibration object. If the original unit is meters (which is the case if you use the standard calibration plate), you can use the parameter values ‘m’, ‘cm’, ‘mm’, ‘microns’, or ‘µm’ to directly set the unit of pixel coordinates in the transformed image.

The parameter Interpolation specifies whether bilinear interpolation (‘bilinear’) should be applied between the pixels in the input image or whether the gray value of the nearest neighboring pixel (‘none’) should be used.

The mapping function is stored in the output image Map. Map has the same size as the resulting images after the mapping. If no interpolation is chosen, Map consists of one image containing one channel, in which for each pixel of the resulting image the linearized coordinate of the pixel of the input image is stored that is the nearest neighbor to the transformed coordinates. If bilinear interpolation is chosen, Map consists of one image containing five channels. In the first channel for each pixel in the resulting image the linearized coordinates of the pixel in the input image is stored that is in the upper left position relative to the transformed coordinates. The four other channels contain the weights of the four neighboring pixels of the transformed coordinates, which are used for the bilinear interpolation, in the following order:

2 4 5

The second channel, for example, contains the weights of the pixels that lie to the upper left relative to the transformed coordinates. If several images have to be mapped using the same camera parameters, GenImageToWorldPlaneMap in combination with MapImage is much more efficient than the operator ImageToWorldPlane because the mapping function needs to be computed only once.

Example

* perform camera calibration (with standard calibration plate)
camera_calibration(NX, NY, NZ, NRow, NCol, StartCamParam, NStartPose, ‘all’, FinalCamParam, NFinalPose, Errors)

* world coordinate system is defined by calibration plate in first image FinalPosel := NFinalPose[0:6]
* compensate thickness of plate
set_origin_pose(FinalPose1, 0, 0, 0.000073, WorldPose)
* goal: rectify images
* first determine parameters such that the entire image content is visible
* -> transform image boundary into world plane, determine smallest rectangle around it
get_image_pointer1(Image, Pointer, Type, Width, Height)
gen_rectangle1(ImageRect, 0, 0, Height-1, Width-1)
gen_contour_region_xld(ImageRect, ImageBorder, 'border')
contour_to_world_plane_xld(ImageBorder, ImageBorderWCS, FinalCamParam, WorldPose, 1)
smallest_rectangle1_xld(ImageBorderWCS, MinY, MinX, MaxY, MaxX)
* -> move the pose to the upper left corner of the surrounding rectangle
set_origin_pose(WorldPose, MinX, MinY, 0, PoseForEntireImage)
* -> determine the scaling factor such that the center pixel has the same size in the original and in the rectified image
* method: transform corner points of the pixel into the world coordinate system, compute their distances, and use their mean as the scaling factor
image_points_to_world_plane(FinalCamParam, PoseForEntireImage, [Height/2, Height/2, Height/2+1], [Width/2, Width/2+1, Width/2], 1, WorldPixelX, WorldPixelY)
distance_pp(WorldPixelY[0], WorldPixelX[0], WorldPixelY[1], WorldPixelX[1], WorldLength1)
distance_pp(WorldPixelY[0], WorldPixelX[0], WorldPixelY[2], WorldPixelX[2], WorldLength2)
ScaleForSimilarPixelSize := (WorldLength1+WorldLength2)/2
* -> determine output image size such that entire input image fits into it
ExtentX := MaxX-MinX
ExtentY := MaxY-MinY
WidthRectifiedImage := ExtentX/ScaleForSimilarPixelSize
HeightRectifiedImage := ExtentY/ScaleForSimilarPixelSize
* create mapping with the determined parameters
gen_image_to_world_plane_map(Map, FinalCamParam, PoseForEntireImage, Width, Height, WidthRectifiedImage, HeightRectifiedImage, ScaleForSimilarPixelSize, 'bilinear')
* transform grabbed images with the created map
while(1)
  grab_image_async(Image, FGHandle, -1)
  map_image(Image, Map, RectifiedImage)
endwhile

Result
GenImageToWorldPlaneMap returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information
GenImageToWorldPlaneMap is reentrant and processed without parallelization.

Possible Predecessors
CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration, SetOriginPose

Possible Successors
MapImage

See also
MapImage, ContourToWorldPlaneXld, ImagePointsToWorldPlane

Alternatives
ImageToWorldPlane

HALCON 6.1.4
Camera calibration

Generate a projection map that describes the mapping of images corresponding to a changing radial distortion. 

GenRadialDistortionMap computes the mapping of images corresponding to a changing radial distortion in accordance to the interior camera parameters CamParIn and CamParOut, which can be obtained, e.g., using the operator CameraCalibration. CamParIn and CamParOut contain the old and the new camera parameters including the old and the new radial distortion, respectively (also see WriteCamPar for the sequence of the parameters and the underlying camera model). Each pixel of the potential output image is transformed into the image plane using CamParOut and subsequently projected into a subpixel position of the potential input image using CamParIn.

The mapping function is stored in the output image Map. The size of Map is given by the camera parameters and therefore defines the size of the images to be mapped using MapImage. If no interpolation is chosen (Interpolation = 'none'), Map consists of one image containing one channel, in which for each pixel of the output image the linearized coordinate of the pixel of the input image is stored that is the nearest neighbor to the transformed coordinates. If bilinear interpolation is chosen (Interpolation = 'bilinear'), Map consists of one image containing five channels. In the first channel for each pixel in the resulting image the linearized coordinate of the pixel in the input image is stored that is in the upper left position relative to the transformed coordinates. The four other channels contain the weights of the four neighboring pixels of the transformed coordinates, which are used for the bilinear interpolation, in the following order:

```
2 3
4 5
```

The second channel, for example, contains the weights of the pixels that lie to the upper left relative to the transformed coordinates.

If CamParOut was computed via ChangeRadialDistortionCamPar, the mapping describes the effect of a lens with a modified radial distortion. If $\kappa$ is 0, the mapping corresponds to a rectification.

If several images have to be mapped using the same camera parameters, GenRadialDistortionMap in combination with MapImage is much more efficient than the operator ChangeRadialDistortionImage because the transformation must be computed only once.

**Parameter**

- **Map** (output iconic) .................. (multichannel-)image $\sim$ HImageX / HUntypedObjectX (int4, uint2)
  Image containing the mapping data.

- **CamParIn** (input control) ........................................ number $\sim$ VARIANT (integer, real)
  Old camera parameters.

  **Number of elements**: 8

- **CamParOut** (input control) ................................. number $\sim$ VARIANT (integer, real)
  New camera parameters.

  **Number of elements**: 8

- **Interpolation** (input control) .............................. string $\sim$ String / VARIANT
  Type of interpolation.

  **Default Value**: 'bilinear'

  **List of values**: Interpolation $\in$ {'none', 'bilinear'}

**Result**

GenRadialDistortionMap returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

HALCON/COM Reference Manual, 2005-2-1
Compute the line of sight corresponding to a point in the image.

GetLineOfSight computes the line of sight corresponding to a pixel (Row, Column) in the image. The line of sight is a (straight) line in the camera coordinate system, which is described by two points \((PX, PY, PZ)\) and \((QX, QY, QZ)\) on the line. A pinhole or telecentric camera model with radial distortions described by the interior camera parameters CamParam is used (see WriteCamPar). If a pinhole camera is used, the second point lies on the focal plane, i.e., the output parameter \(QZ\) is equivalent to the focal length of the camera. The equation of the line of sight is given by

\[
\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} PX \\ PY \\ PZ \end{pmatrix} + l \cdot \begin{pmatrix} QX - PX \\ QY - PY \\ QZ - PZ \end{pmatrix}
\]

The advantage of representing the line of sight as two points is that it is easier to transform the line in 3D. To do so, all that is necessary is to apply the operator AffineTransPoint3D to the two points.
Example

* get interior camera parameters
  read_cam_par('campar.dat', CamParam)
* inverse projection
  get_line_of_sight([50, 100], [100, 200], CamParam, PX, PY, PZ, QX, QY, QZ)

GetLineOfSight returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information
GetLineOfSight is reentrant and processed without parallelization.

Possible Predecessors
ReadCamPar, CameraCalibration

Possible Successors
AffineTransPoint3D
See also
CameraCalibration, DispCaltab, ReadCamPar, Project3DPoint, AffineTransPoint3D

Module
Camera calibration

Get the representation type of a 3D pose.

With GetPoseType, the representation type of the 3D pose Pose can be queried. See CreatePose for details about 3D poses, their representation types, and the meaning of the parameters OrderOfTransform, OrderOfRotation, and ViewOfTransform.

Parameter

- **Pose** (input control) 3D pose.
  Number of elements: 7
- **OrderOfTransform** (output control) Order of rotation and translation.
- **OrderOfRotation** (output control) Meaning of the rotation values.
- **ViewOfTransform** (output control) View of transformation.

Result
CreatePose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information
GetPoseType is reentrant and processed without parallelization.

Possible Predecessors
CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration

Possible Successors
ConvertPoseType
See also
CreatePose, ConvertPoseType, WritePose, ReadPose

Module
Camera calibration
Perform a hand-eye calibration.

The operator **HandEyeCalibration** determines the 3D pose of a robot (“hand”) relative to a camera (“eye”). With this information, the results of image processing can be transformed into the coordinate system of the robot, which can then, e.g., grasp an inspected part. There are two possible configurations of robot-camera (hand-eye) systems: The camera can be mounted on the robot or be stationary and observe the robot. Note that the term robot is used in place for a mechanism that moves objects. Thus, you can use **HandEyeCalibration** to calibrate many different systems, from pan-tilt heads to multi-axis manipulators.

A hand-eye calibration is performed similarly to the calibration of the external camera parameters (see **CameraCalibration**): You acquire a set of images of a calibration object, determine correspondences between known model points and their projection in the images and pass them to **HandEyeCalibration** via the parameters NX, NY, NZ, NRow, NCol, and **MPointsOfImage**. If you use the standard calibration plate, the correspondences can be determined very easily with the operators **FindCalTab** and **FindMarksAndPose**. Furthermore, you must specify the internal camera parameters in **CamParam**.

In contrast to the camera calibration, the calibration object is not moved manually. This task is delegated to the robot, which either moves the camera (mounted camera) or the calibration object (stationary camera). The robot’s movements are assumed to be known and therefore are also used as an input for the calibration (parameter **MRelPoses**).

The two hand-eye configurations are discussed in more detail below, followed by general information about the process of hand-eye calibration.

**Moving camera (mounted on a robot)**

In this configuration, the calibration object remains stationary and the camera is moved to different positions by the robot. The main idea behind the hand-eye calibration is that the information extracted from a calibration image, i.e., the pose of the calibration object relative to the camera (i.e., the external camera parameters), can be seen as a chain of poses or homogeneous transformation matrices, from the calibration object via the base of the robot to its tool (end-effector) and finally to the camera:

\[
\begin{align*}
{\text{moving camera:}} \\
\mathbf{H}_{\text{cal}} & = {\mathcal{c}} \cdot \mathbf{H}_{\text{tool}} \cdot {\text{tool}} \cdot \mathbf{H}_{\text{base}} \cdot {\text{base}} \\
& = {\text{CamStartPose}} \cdot \mathbf{MRelPoses} \cdot \mathbf{BaseStartPose} \\
& = \mathbf{CamFinalPose} \cdot \mathbf{BaseFinalPose}
\end{align*}
\]

From the set of calibration images, the operator **HandEyeCalibration** determines the two transformations at the ends of the chain, i.e., the pose of the robot tool in camera coordinates (\(\mathbf{H}_{\text{tool}}\)) and the pose of the calibration object in the robot base coordinate system (\(\mathbf{H}_{\text{cal}}\)). In the input parameters **CamStartPose** and **BaseStartPose**, you must specify suitable starting values for these transformations, which are constant over all calibration images. **HandEyeCalibration** then returns the calibrated values in **CamFinalPose** and **BaseFinalPose**.

In contrast, the transformation in the middle of the chain, \(\mathbf{H}_{\text{base}}\), is known but changes for each calibration image, because it describes the pose of the robot moving the camera, or to be more exact its inverse pose (pose of
the base coordinate system in robot tool coordinates). You must specify the (inverse) robot poses in the calibration images in the parameter \( \text{MRelPoses} \).

Internally, \texttt{HandEyeCalibration} uses a Newton-type algorithm to minimize an error function based on normal equations. Analogously to the calibration of the camera itself (see \texttt{CameraCalibration}), the hand-eye calibration becomes more robust if you use many calibration images that were acquired with different robot poses.

**Stationary camera**

In this configuration, the robot grasps the calibration object and moves it in front of the camera. Again, the information extracted from a calibration image, i.e., the pose of the calibration object in camera coordinates (i.e., the external camera parameters), are equal to a chain of poses or homogeneous transformation matrices, this time from the calibration object via the robot’s tool to its base and finally to the camera:

\[
\text{Stationary camera: } \quad \text{cam}_c H_{\text{cal}} = \text{cam}_c H_{\text{base}} \cdot \text{base}_t H_{\text{tool}} \cdot \text{tool}_t H_{\text{cal}}
\]

Analogously to the configuration with a moving camera, the operator \texttt{HandEyeCalibration} determines the two transformations at the ends of the chain, here the pose of the robot base coordinate system in camera coordinates (\( \text{cam}_c H_{\text{base}} \)) and the pose of the calibration object relative to the robot tool (\( \text{tool}_t H_{\text{cal}} \)). In the input parameters \texttt{CamStartPose} and \texttt{BaseStartPose}, you must specify suitable starting values for these transformations. \texttt{HandEyeCalibration} then returns the calibrated values in \texttt{CamFinalPose} and \texttt{BaseFinalPose}. Please note that the names of the parameters \texttt{BaseStartPose} and \texttt{BaseFinalPose} are misleading for this configuration!

The transformation in the middle of the chain, \( \text{base}_t H_{\text{tool}} \), describes the pose of the robot moving the calibration object, i.e., the pose of the tool relative to the base coordinate system. You must specify the robot poses in the calibration images in the parameter \( \text{MRelPoses} \).

**Additional information about the calibration process**

The following sections discuss individual questions arising from the use of \texttt{HandEyeCalibration}. They are intended to be a guideline for using the operator in an application, as well as to help understanding the operator.

**How do I get 3D model points and their projections?**

3D model points given in the world coordinate system (\( \text{NX}, \text{NY}, \text{NZ} \)) and their associated projections in the image (\( \text{NRow}, \text{NCol} \)) form the basis of the hand-eye calibration. In order to be able to perform a successful hand-eye calibration, you need images of the 3D model points that were obtained for sufficiently many different poses of the manipulator.

In principle, you can use arbitrary known points for the calibration. However, it is usually most convenient to use the standard calibration plate, e.g., the one that can be generated with \texttt{CreateCaltab}. In this case, you can use the operators \texttt{FindCaltab} and \texttt{FindMarksAndPose} to extract the position of the calibration plate and of the calibration marks and the operator \texttt{CaltabPoints} to access the 3D coordinates of the calibration marks (see also the description of \texttt{CameraCalibration}).

The parameter \( \text{MPointsOfImage} \) specifies the number of 3D model points used for each pose of the manipulator, i.e., for each image. With this, the 3D model points, which are stored in a linearized fashion in \( \text{NX}, \text{NY}, \text{NZ} \), and their corresponding projections (\( \text{NRow}, \text{NCol} \)) can be associated with the corresponding pose of the manipulator (\( \text{MRelPoses} \)). Note that in contrast to the operator \texttt{CameraCalibration} the 3D coordinates of the model points must be specified for each calibration image, not only once.

**How do I acquire a suitable set of images?**

If a standard calibration plate is used, the following procedure should be used:

- At least 10 to 20 images from different positions should be taken in which the position of the camera with respect to the calibration plate is sufficiently different. The position of the calibration plate (moving camera: relative to the robot’s tool; stationary camera: relative to the robot’s base) must not be changed between images.
- In each image, the calibration plate must be completely visible (including its border).
- No reflections or other disturbances should be visible on the calibration plate.
The set of images must show the calibration plate from very different positions of the manipulator. The calibration plate can and should be visible in different parts of the images. Furthermore, it should be slightly to moderately rotated around its x- or y-axis, in order to clearly exhibit distortions of the calibration marks. In other words, the corresponding exterior camera parameters (pose of the calibration plate in camera coordinates) should take on many different values.

- In each image, the calibration plate should fill at least one quarter of the entire image, in order to ensure the robust detection of the calibration marks.
- The interior camera parameters of the camera to be used must have been determined earlier and must be passed in \texttt{CamParam} (see \texttt{CameraCalibration}). Note that changes of the image size, the focal length, the aperture, or the focus effect a change of the interior camera parameters.
- The camera must not be modified between the acquisition of the individual images, i.e., focal length, aperture, and focus must not be changed, because all calibration images use the same interior camera parameters. Please make sure that the focus is sufficient for the expected changes of the distance the camera from the calibration plate. Therefore, bright lighting conditions for the calibration plate are important, because then you can use smaller apertures, which result in larger depth of focus.

**How do I obtain suitable starting values?** Depending on the used hand-eye configuration, you need starting values for the following poses:

**Moving camera**

\begin{align*}
\text{BaseStartPose} &= \text{pose of the calibration object in robot base coordinates} \\
\text{CamStartPose} &= \text{pose of the robot tool in camera coordinates}
\end{align*}

**Stationary camera**

\begin{align*}
\text{BaseStartPose} &= \text{pose of the calibration object in robot tool coordinates} \\
\text{CamStartPose} &= \text{pose of the robot base in camera coordinates}
\end{align*}

The camera’s coordinate system is oriented such that its optical axis corresponds to the z-axis, the x-axis points to the right, and the y-axis downwards. The coordinate system of the standard calibration plate is located in the middle of the surface of the calibration plate, its z-axis points into the calibration plate, its x-axis to the right, and it y-axis downwards.

For more information about creating a 3D pose please refer to the description of \texttt{CreatePose}, which also contains a short example.

In fact, you need a starting value only for one of the two poses (\texttt{BaseStartPose} or \texttt{CamStartPose}). The other can be computed from one of the calibration images. This means that you can pick the pose that is easier to determine and let HALCON compute the other one for you.

The main idea is to exploit the fact that the two poses for which we need starting values are connected via the already described chain of transformations, here shown for a configuration with a moving camera:

\[
\begin{align*}
\text{moving camera:} & \quad \text{CamStartPose} \quad \text{MRelPoses} \quad \text{BaseStartPose} \\
\text{base}_{\text{cal}} &= \text{base}_{\text{tool}} \cdot \text{tool}_{\text{base}} \cdot \text{base}_{\text{cal}} \\
\text{CamStartPose} &= \left( \text{base}_{\text{tool}} \cdot \text{tool}_{\text{base}} \right)^{-1} \cdot \text{base}_{\text{cal}} \\
\text{moving camera:} & \quad \text{BaseStartPose} \quad \text{CamStartPose}
\end{align*}
\]

In this configuration, it is typically easy to determine a starting value for \textit{cam}\text{tool} \,(\text{CamStartPose}). Thus, we solve the equation for \textit{base}\text{cal} \,(\text{BaseStartPose}):

\[
\text{moving camera:} \quad \text{base}_{\text{cal}} = \left( \text{cam}_{\text{tool}} \cdot \text{tool}_{\text{base}} \right)^{-1} \cdot \text{cam}_{\text{cal}}
\]

Thus, to compute \texttt{BaseStartPose} you need one of the robot poses (e.g., the one in the first image), your estimate for \texttt{CamStartPose}, and the pose of the calibration object in camera coordinates in the selected image. If you use the standard calibration plate, you typically already obtained its pose when applying the operator \texttt{FindMarksAndPose} to determine the projections of the marks. An example program can be found below.

For a configuration with a stationary camera, the chain of transformations is:
**Chapter 13. Tools**

**Stationary camera:**

\[
\begin{align*}
\text{Stationary camera:} & \quad \text{\(c_{am}H_{cal} = c_{am}H_{base} \cdot h_{base}H_{tool} \cdot t_{tool}H_{cal}\)} \\
\text{CamStartPose} & \quad \text{MRelPoses} \quad \text{BaseStartPose}
\end{align*}
\]

In this configuration, it is typically easier to determine a starting value for \(t_{tool}H_{cal}\) (BaseStartPose). Thus, we solve the equation for \(c_{am}H_{base}\) (CamStartPose):

\[
\begin{align*}
\text{Stationary camera:} & \quad \text{\(c_{am}H_{base} = c_{am}H_{cal} (baseH_{tool} \cdot t_{tool}H_{cal})^{-1}\)} \\
& = \text{\(c_{am}H_{cal} \cdot c_{cal}H_{tool} \cdot t_{tool}H_{base}\)}
\end{align*}
\]

Thus, to compute CamStartPose you need one of the robot poses (e.g., the one in the first image), your estimate for BaseStartPose, and the pose of the calibration object in camera coordinates in the selected image. If you use the standard calibration plate, you typically already obtained its pose when applying the operator FindMarksAndPose to determine the projections of the marks. An example program can be found below.

**How do I obtain the poses of the robot?**

In the parameter MRelPoses you must pass the poses of the robot in the calibration images (moving camera: pose of the robot base in robot tool coordinates; stationary camera: pose of the robot tool in robot base coordinates) in a linearized fashion. We recommend to create the robot poses in a separate program and save in files using WritePose. In the calibration program you can then read and accumulate them in a tuple as shown in the example program below. Besides, we recommend to save the pose of the robot tool in robot base coordinates independent of the hand-eye configuration. When using a moving camera, you then invert the read poses before accumulating them. This is also shown in the example program.

Via the cartesian interface of the robot, you can typically obtain the pose of the tool in base coordinates in a notation that corresponds to the pose representations with the codes 0 or 2 (OrderOfRotation = 'gba' or 'abg', see CreatePose). In this case, you can directly use the pose values obtained from the robot as input for CreatePose.

If the cartesian interface of your robot describes the orientation in a different way, e.g., with the representation ZYZ (\(R_z(\phi_3) \cdot R_y(\phi_2) \cdot R_z(\phi_1)\)), you can create the corresponding homogeneous transformation matrix step by step using the operators HomMat3dRotate and HomMat3dTranslate and then convert the matrix into a pose using HomMat3dToPose. The following example code creates a pose from the ZYZ representation described above:

\[
\begin{align*}
\text{hommat3d_identity (HomMat3DIdent)} & \\
\text{hommat3d_rotate (HomMat3DIdent, \(\varphi_3\), 'z', 0, 0, 0, HomMat3DRotZ)} & \\
\text{hommat3d_rotate (HomMat3DRotZ, \(\varphi_2\), 'y', 0, 0, 0, HomMat3DRotYZ)} & \\
\text{hommat3d_rotate (HomMat3DRotYZ, \(\varphi_1\), 'z', 0, 0, 0, HomMat3DRotZYZ)} & \\
\text{hommat3d_translate (HomMat3DRotZYZ, Tx, Ty, Tz, baseHtool)} & \\
\text{hommat3d_to_pose (baseHtool, RobPose)}
\end{align*}
\]

Please note that the hand-eye calibration only works if the robot poses MRelPoses are specified with high accuracy!

**How can I exclude individual pose parameters from the estimation?**

HandEyeCalibration estimates a maximum of 12 pose parameters, i.e., 6 parameters each for the two computed poses CamFinalPose and BaseFinalPose. However, it is possible to exclude some of these pose parameters from the estimation. This means that the starting values of the poses remain unchanged and are assumed constant for the estimation of all other pose parameters. The parameter ToEstimate is used to determine which pose parameters should be estimated. In ToEstimate, a list of keywords for the parameters to be estimated is passed. The possible values are:

**BaseFinalPose:**

- 'baseTx' = translation along the x-axis
- 'baseTy' = translation along the y-axis
- 'baseTz' = translation along the z-axis
- 'baseRa' = rotation around the x-axis
- 'baseRb' = rotation around the y-axis
- 'baseRg' = rotation around the z-axis
In order to estimate all 12 pose parameters, you can pass the keyword 'all' (or of course a tuple containing all 12 keywords listed above).

It is useful to exclude individual parameters from the estimation if some of the pose parameters have already been measured exactly.

**Which terminating criteria can be used for the error minimization?** The error minimization terminates either after a fixed number of iterations or if the error falls below a given minimum error. The parameter `StopCriterion` is used to choose between these two alternatives. If 'CountIterations' is passed, the algorithm terminates after `MaxIterations` iterations.

If `StopCriterion` is passed as 'MinError', the algorithm runs until the error falls below the error threshold given in `MinError`. If, however, the number of iterations reaches the number given in `MaxIterations`, the algorithm terminates with an error message.

**What is the order of the individual parameters?** The length of the tuple `MPointsOfImage` corresponds to the number of different positions of the manipulator and thus to the number of calibration images. The parameter `MPointsOfImage` determines the number of model points used in the individual positions. If the standard calibration plate is used, this means 49 points per position (image). If for example 15 images were acquired, `MPointsOfImage` is a tuple of length 15, where all elements of the tuple have the value 49.

The number of calibration images, which is determined by the length of `MPointsOfImage`, must also be taken into account for the tuples for the 3D model points and for the extracted 2D marks, respectively. Hence, for 15 calibration images with 49 model points each, the tuples `NX, NY, NZ, NRow, and NCol` must contain 15 · 49 = 735 values each. These tuples are ordered according to the image the respective points lie in, i.e., the first 49 values correspond to the 49 model points in the first image. The order of the 3D model points and the extracted 2D model points must be the same in each image.

The length of the tuple `MRelPoses` also depends on the number of calibration images. If, for example, 15 images and therefore 15 poses are used, the length of the tuple `MRelPoses` is 15 · 7 = 105 (15 times 7 pose parameters). The first seven parameters thus determine the pose of the manipulator in the first image, and so on.

**What do the output parameters mean?** If `StopCriterion` was set to 'CountIterations', the output parameters `BaseFinalPose` and `CamFinalPose` are returned even if the algorithm didn’t converge. If, however, `StopCriterion` was set to 'MinError', the error must fall below 'MinError' in order for output parameters to be returned.

The representation type of `BaseFinalPose` and `CamFinalPose` is the same as in the corresponding starting values. It can be changed with the operator `ConvertPoseType`. The description of the different representation types and of their conversion can be found with the documentation of the operator `CreatePose`.

The parameter `NumErrors` contains a list of (numerical) errors from the individual iterations of the algorithm. Based on the evolution of the errors, it can be decided whether the algorithm has converged for the given starting values. The error values are returned as 3D deviations in meters. Thus, the last entry of the error list corresponds to an estimate of the accuracy of the returned pose parameters.

---

**Attention**

The quality of the calibration depends on the accuracy of the input parameters (position of the calibration marks, robot poses `MRelPoses`, and the starting positions `BaseStartPose, CamStartPose`). Based on the returned error measures `NumErrors`, it can be decided, whether the algorithm has converged. Furthermore, the accuracy of the returned pose can be estimated. The error measures are 3D differences in meters.

---

**Parameter**

- **NX** (input control) .............................................. real  ~  VARIANT ( real )  
  Linear list containing all the x coordinates of the calibration points (in the order of the images).

- **NY** (input control) .............................................. real  ~  VARIANT ( real )  
  Linear list containing all the y coordinates of the calibration points (in the order of the images).
NZ (input control) .................................................... real  \(\sim\) VARIANT (real)
Linear list containing all the z coordinates of the calibration points (in the order of the images).

NRow (input control) .................................................... real  \(\sim\) VARIANT
Linear list containing all row coordinates of the calibration points (in the order of the images).

NCol (input control) .................................................... real  \(\sim\) VARIANT
Linear list containing all the column coordinates of the calibration points (in the order of the images).

MPointsOfImage (input control) .................................... integer  \(\sim\) VARIANT
Number of the calibration points for each image.

MRelPoses (input control) ......................................... pose  \(\sim\) VARIANT (integer, real)
Measured 3D pose of the robot for each image (moving camera: robot base in robot tool coordinates;
stationary camera: robot tool in robot base coordinates).

BaseStartPose (input control) ...................................... pose  \(\sim\) VARIANT (integer, real)
Starting value for the 3D pose of the calibration object in robot base coordinates (moving camera) or in robot
tool coordinates (stationary camera), respectively.

CamStartPose (input control) ...................................... pose  \(\sim\) VARIANT (integer, real)
Starting value for the 3D pose of the robot tool (moving camera) or robot base (stationary camera),
respectively, in camera coordinates.

CamParam (input control) .......................................... number  \(\sim\) VARIANT (integer, real)
Interior camera parameters.

ToEstimate (input control) ........................................ string  \(\sim\) VARIANT (string)
Parameters to be estimated (max. 12 degrees of freedom).

Default Value: ‘all’
List of values: ToEstimate \(\in\) \{'all', 'baseTx', 'baseTy', 'baseTz', 'baseRa', 'baseRb', 'baseRg',
'camTx', 'camTy', 'camTz', 'camRa', 'camRb', 'camRg'\}

StopCriterion (input control) ..................................... string  \(\sim\) String / VARIANT
Type of stopping criterion.

Default Value: ‘CountIterations’
List of values: StopCriterion \(\in\) \{'CountIterations', 'MinError'\}

MaxIterations (input control) ..................................... integer  \(\sim\) long / VARIANT
Maximum number of iterations to be executed.

Default Value: 15
Suggested values: MaxIterations \(\in\) \{10, 15, 20, 25, 30\}

MinError (input control) .......................................... real  \(\sim\) double / VARIANT
Minimum error used as the stopping criterion.

Default Value: 0.0005
Suggested values: MinError \(\in\) \{0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}

BaseFinalPose (output control) .................................... pose  \(\sim\) VARIANT (integer, real)
Computed 3D pose for the 3D pose of the calibration object in robot base coordinates (moving camera) or in
robot tool coordinates (stationary camera), respectively.

CamFinalPose (output control) .................................... pose  \(\sim\) VARIANT (integer, real)
Computed 3D pose of the robot tool (moving camera) or robot base (stationary camera), respectively,
in camera coordinates.

NumErrors (output control) ....................................... real(-array)  \(\sim\) VARIANT
Error measures for each iteration.

Example

read_cam_par(‘campar.dat’, CamParam)
CalDescr := ‘caltab.descr’
caltab_points(CalDescr, X, Y, Z)
* process all calibration images
for i := 0 to NumImages-1 by 1
    read_image(Image, ‘calib_’+i$‘02d’)
* find marks on the calibration plate in every image
    find_caltab(Image, CalPlate, CalDescr, 3, 150, 5)
    find_marks_and_pose(Image, CalPlate, CalDescr, CamParam, 128, 10,
                        RCoordTmp, CCoordTmp, StartPose)
* accumulate 2D and 3D coordinates of the marks
13.4. CALIBRATION

RCoord := [RCoord, RCoordTmp]
CCoord := [CCoord, CCoordTmp]
XCoord := [XCoord, X]
YCoord := [YCoord, Y]
ZCoord := [ZCoord, Z]
NumMarker := [NumMarker, |RCoordTmp|]

* read pose of the robot tool in robot base coordinates
read_pose(‘robpose_’+i$’02d’+’.dat’, RobPose)

* moving camera? invert pose
if (IsMovingCameraConfig=’true’)
    pose_to_hom_mat3d(RobPose, base_H_tool)
    hom_mat3d_invert(base_H_tool, tool_H_base)
    hom_mat3d_to_pose(tool_H_base, RobPose)
endif

* accumulate robot poses
MRelPoses := [MRelPoses, RobPose]

* store the pose of the calibration plate in the first image and the
* corresponding pose of the robot for later use
if (i=0)
    cam_P_cal := StartPose
    RelPose0 := RobPose
endif
endfor

* obtain starting values: read one, compute the other
if (IsMovingCameraConfig=’true’)
    * mov. camera: read pose of robot tool in camera coordinates
    * compute pose of calibration plate in robot base coordinates
    read_pose(‘cam_P_tool.dat’, CamStartPose)
    BaseStartPose = inverse(CamStartPose * RelPose0) * cam_P_cal
    pose_to_hom_mat3d(CamStartPose, cam_H_tool)
    pose_to_hom_mat3d(RelPose0, tool_H_base)
    pose_to_hom_mat3d(StartPose, cam_H_cal)
    hom_mat3d_compose(cam_H_tool, tool_H_base, cam_H_base)
    hom_mat3d_invert(cam_H_base, base_H_cam)
    hom_mat3d_compose(base_H_cam, cam_H_cal, base_H_cal)
    hom_mat3d_to_pose(base_H_cal, BaseStartPose)
else
    * stat. camera: read pose of calibration plate in robot tool coordinates
    * compute pose of robot base in camera coordinates
    read_pose(‘tool_P_cal.dat’, BaseStartPose)
    CamStartPose = cam_P_cal * inverse(RelPose0 * BaseStartPose)
    pose_to_hom_mat3d(BaseStartPose, tool_H_cal)
    pose_to_hom_mat3d(RelPose0, base_H_tool)
    pose_to_hom_mat3d(StartPose, cam_H_cal)
    hom_mat3d_compose(base_H_tool, tool_H_cal, base_H_cal)
    hom_mat3d_invert(base_H_call, cal_H_base)
    hom_mat3d_compose(cam_H_cal, cal_H_base, cam_H_cal)
    hom_mat3d_to_pose(cam_H_base, CamStartPose)
endif

* perform hand-eye calibration
*hand_eye_calibration(XCoord, YCoord, ZCoord, RCoord, CCoord, NumMarker,
    MRelPoses, BaseStartPose, CamStartPose, CamParam,
    "all", "CountIterations", 20, 0.000670,
    BaseFinalPose, CamFinalPose, NumErrors)

* measure some point P in camera coordinates (cam_px, cam_py, cam_pz)
*
* transform point into robot base coordinates: \( \text{base}_p = \text{base}_{\text{H_cam}} \times \text{cam}_p \)
  if (IsMovingCameraConfig='true')
    * mov. camera: \( \text{base}_{\text{H_cam}} = \text{base}_{\text{H_tool}} \times \text{tool}_{\text{H_cam}} \)
    * \( \text{base}_p = \text{RobPose} \times \text{inverse(CamFinalPose)} \)
      pose_to_hom_mat3d(CamFinalPose, cam_H_tool)
      hom_mat3d_invert(cam_H_tool, tool_H_cam)
    * obtain current robot pose RobPose from robot
      pose_to_hom_mat3d(RobPose, base_H_tool)
      hom_mat3d-compose(base_H_tool, tool_H_cam, base_H_cam)
  else
    * stat. camera: \( \text{base}_p = \text{inverse(CamFinalPose)} \)
      pose_to_hom_mat3d(CamFinalPose, cam_H_base)
      hom_mat3d_invert(cam_H_base, base_H_cam)
  endif

affine_trans_point_3d(base_H_cam, cam_px, cam_py, cam_pz,
  base_px, base_py, base_pz)

---

**Result**

`HandEyeCalibration` returns TRUE if all parameter values are correct and the method converges with an error less than the specified minimum error (if `StopCriterion` = 'MinError'). If necessary, an exception handling is raised.

---

**Parallelization Information**

`HandEyeCalibration` is reentrant and processed without parallelization.

---

**Possible Predecessors**

FindMarksAndPose

---

**Possible Successors**

WritePose, ConvertPoseType, PoseToHomMat3d, DispCaltab, SimCaltab

---

See also

FindCaltab, FindMarksAndPose, DispCaltab, SimCaltab, WriteCamPar, ReadCamPar, CreatePose, ConvertPoseType, WritePose, ReadPose, PoseToHomMat3d, HomMat3dToPose, CaltabPoints, CreateCaltab

---

**Module**

Camera calibration

---

```plaintext
[out] VARIANT Pose HHomMat3dX.HomMat3dToPose ( )
void HOperatorSetX.HomMat3dToPose ([in] VARIANT HomMat3D,
[out] VARIANT Pose )
```

Convert a homogeneous transformation matrix into a 3D pose.

`HomMat3dToPose` converts a homogeneous transformation matrix into the corresponding 3D pose with type code 0. For details about 3D poses and the corresponding transformation matrices please refer to `CreatePose`.

A typical application of `HomMat3dToPose` is that a 3D pose was converted into a homogeneous transformation matrix to further transform it, e.g., with `HomMat3dRotate` or `HomMat3dTranslate`, and now must be converted back into a pose to use it as input for operators like `ImagePointsToWorldPlane`.

---

**Parameter**

* **HomMat3D** (input control) .......................... affine3d ~ HHomMat3dX / VARIANT( real )
  Homogeneous transformation matrix.
  Number of elements : 12
* **Pose** (output control) ................................. pose ~ VARIANT( integer, real )
  Equivalent 3D pose.
  Number of elements : 7

---

**Example**

HALCON/COM Reference Manual, 2005-2-1
camera_calibration(WorldPointsX, WorldPointsY, WorldPointsZ,
     PixelsRow, PixelsColumn, CamParam, StartPose, 6,
     FinalCamParam, FinalPose, Errors)
* transform FinalPose to homogeneous transformation matrix
pose_to_hom_mat3d(FinalPose, cam_H_cal)
* rotate it 90 degree around the y-axis to obtain a world coordinate system
* whose y- and z-axis lie in the plane of the calibration plate while the
* x-axis point ‘upwards’: cam_H_w = cam_H_cal * RotY(90)
hom_mat3d_identity(HomMat3DIdent)
hom_mat3d_rotate(HomMat3DIdent, deg(90), ‘y’, 0, 0, 0,
     HomMat3DRotateY)
hom_mat3d_compose(cam_H_cal, HomMat3DRotateY, cam_H_w)
* transform back to pose
hom_mat3d_to_pose(cam_H_w, cam_P_w)
* use pose to transform an image point into the world coordinate system
image_points_to_world_plane(FinalCamParam, cam_P_w, 87, 23.5, 1,
     w_px, w_py)

Result
HomMat3dToPose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised

Parallelization Information
HomMat3dToPose is reentrant and processed without parallelization.

Possible Predecessors
HomMat3dRotate, HomMat3dTranslate, HomMat3dInvert

Possible Successors
CameraCalibration, WritePose, DispCaltab, SimCaltab

See also
CreatePose, CameraCalibration, DispCaltab, SimCaltab, WritePose, ReadPose,
PoseToHomMat3d, Project3DPoint, GetLineOfSight, HomMat3dRotate,
HomMat3dTranslate, HomMat3dInvert, AffineTransPoint3D

Module
Camera calibration

Transform image points into the plane z=0 of a world coordinate system.
The operator ImagePointsToWorldPlane transforms image points which are given in Rows and Cols into the plane z=0 in a world coordinate system and returns their 3D coordinates in X and Y. The world coordinate system is chosen by passing its 3D pose relative to the camera coordinate system in WorldPose. In CamParam you must pass the interior camera parameters (see WriteCamPar for the sequence of the parameters and the underlying camera model).

In many cases CamParam and WorldPose are the result of calibrating the camera with the operator CameraCalibration. See below for an example.

With the parameter Scale you can scale the resulting 3D coordinates. The parameter Scale must be specified as the ratio desired unit/original unit. The original unit is determined by the coordinates of the calibration object. If the original unit is meters (which is the case if you use the standard calibration plate), you can set the desired unit directly by selecting 'm', 'cm', 'mm' or 'µm' for the parameter Scale.

Internally, the operator first computes the line of sight between the projection center and the image contour points in the camera coordinate system, taking into account the radial distortions. The line of sight is then transformed
into the world coordinate system specified in WorldPose. By intersecting the plane z=0 with the line of sight the 3D coordinates X and Y are obtained.

---

**Parameter**

- **CamParam** (input control) ........................................... number \(\sim\) VARIANT \(\text{(integer, real)}\)
  Intermediate camera parameters.
  **Number of elements**: 8

- **WorldPose** (input control) ........................................... pose \(\sim\) VARIANT \(\text{(integer, real)}\)
  3D pose of the world coordinate system in camera coordinates.
  **Number of elements**: 7

- **Rows** (input control) ........................................... coordinates.y \(\sim\) VARIANT \(\text{(integer, real)}\)
  Row coordinates of the points to be transformed.
  **Default Value**: 100.0

- **Cols** (input control) ........................................... coordinates.x \(\sim\) VARIANT \(\text{(integer, real)}\)
  Column coordinates of the points to be transformed.
  **Default Value**: 100.0

- **Scale** (input control) ........................................... number \(\sim\) VARIANT \(\text{(integer, real, string)}\)
  Scale or dimension
  **Default Value**: \(\text{\textquoteleft} m\text{\textquoteleft}\)
  **Suggested values**: \(\text{Scale} \in \{\text{\textquoteleft}m\text{\textquoteleft},\text{\textquoteleft}cm\text{\textquoteleft},\text{\textquoteleft}mm\text{\textquoteleft},\text{\textquoteleft}microns\text{\textquoteleft},\mu\text{m}, 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144\}\)

- **X** (output control) ........................................... coordinates.x \(\sim\) VARIANT \(\text{(real)}\)
  X coordinates of the points in the world coordinate system.

- **Y** (output control) ........................................... coordinates.y \(\sim\) VARIANT \(\text{(real)}\)
  Y coordinates of the points in the world coordinate system.

---

**Example**

```
* perform camera calibration (with standard calibration plate)
camera_calibration(NX, NY, NZ, NRow, NCol, StartCamParam, NStartPose, 'all', FinalCamParam, NFinalPose, Errors)
* world coordinate system is defined by calibration plate in first image
FinalPose1 := NFinalPose[0:6]
* compensate thickness of plate
set_origin_pose(FinalPose1, 0, 0, 0.0006, WorldPose)
* transform image points into world coordinate system (unit mm)
image_points_to_world_plane(FinalCamParam, WorldPose, PointRows, PointCols, 'mm', PointXCoord, PointYCoord)
```

---

**Result**

ImagePointsToWorldPlane returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

---

**Parallelization Information**

ImagePointsToWorldPlane is reentrant and processed without parallelization.

---

**Possible Predecessors**

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration, SetOriginPose

---

**See also**

ContourToWorldPlaneXld

---

**Module**

Camera calibration
Rectify an image by transforming it into the plane z=0 of a world coordinate system.

ImageToWorldPlane rectifies an image `Image` by transforming it into the plane z=0 (plane of measurements) in a world coordinate system. The resulting rectified image `ImageWorld` shows neither radial nor perspective distortions; it corresponds to an image acquired by a distortion-free camera that looks perpendicularly onto the plane of measurements. The world coordinate system is chosen by passing its 3D pose relative to the camera coordinate system in `WorldPose`. In `CamParam` you must pass the interior camera parameters (see `WriteCamPar` for the sequence of the parameters and the underlying camera model).

In many cases `CamParam` and `WorldPose` are the result of calibrating the camera with the operator `CameraCalibration`. See below for an example.

The pixel position of the upper left corner of the output image `ImageWorld` is determined by the origin of the world coordinate system. The size of the output image `ImageWorld` can be chosen by the parameters `Width`, `Height`, and `Scale`. `Width`, `Height` and `Scale` must be given in pixels.

With the parameter `Scale` you can specify the size of a pixel in the transformed image. There are two typical scenarios: First, you can scale the image such that pixel coordinates in the transformed image directly correspond to metric units, e.g., that one pixel corresponds to one micron. This is useful if you want to perform measurements in the transformed image, which will then directly result in metric results. The second scenario is to scale the image such that its content appears in a size similar to the original image. This is useful, e.g., if you want to perform shape-based matching in the transformed image.

`Scale` must be specified as the ratio `desired pixel size/original unit`. A pixel size of 1µm means that a pixel in the transformed image corresponds to the area 1µm × 1µm in the plane of measurements. The original unit is determined by the coordinates of the calibration object. If the original unit is meters (which is the case if you use the standard calibration plate), you can use the parameter values `m`, `cm`, `mm`, `microns`, or `µm` to directly set the unit of pixel coordinates in the transformed image.

The parameter `Interpolation` specifies, whether bilinear interpolation (`bilinear`) should be applied between the pixels in the input image or whether the gray value of the nearest neighboring pixel (`none`) should be used.

If several images have to be rectified using the same parameters, `GenImageToWorldPlaneMap` in combination with `MapImage` is much more efficient than the operator `ImageToWorldPlane` in combination `MapImage`. In `CamParam`, `Height`, `Width`, `Scale`, `WorldPose`, and `CamParam` can be chosen by the parameters.

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Image</code></td>
<td>(input iconic) (multichannel-)image(-array) ~ HImageX / IQObjectX (byte, uint2) Input image.</td>
</tr>
<tr>
<td><code>ImageWorld</code></td>
<td>(output iconic) (multichannel-)image(-array) ~ HImageX / IUntypedObjectX (byte, uint2) Transformed image.</td>
</tr>
<tr>
<td><code>CamParam</code></td>
<td>(input control) number ~ VARIANT (integer, real) Interior camera parameters. Number of elements: 8</td>
</tr>
<tr>
<td><code>WorldPose</code></td>
<td>(input control) pose ~ VARIANT (integer, real) 3D pose of the world coordinate system in camera coordinates. Number of elements: 7</td>
</tr>
<tr>
<td><code>Width</code></td>
<td>(input control) extent.x ~ long / VARIANT Width of the resulting image in pixels.</td>
</tr>
<tr>
<td><code>Height</code></td>
<td>(input control) extent.y ~ long / VARIANT Height of the resulting image in pixels.</td>
</tr>
</tbody>
</table>
Scale (input control) .............................................. number \(\sim\) VARIANT (integer, real, string)
Scale or unit
Default Value: 'm'
Suggested values: Scale \(\in\) \{'m', 'cm', 'mm', 'microns', \(\mu\)m, 1.0, 0.01, 0.001, 1.0e-6, 0.0254, 0.3048, 0.9144\}

Interpolation (input control) .................................. string \(\sim\) String / VARIANT
Type of interpolation.
Default Value: 'bilinear'
List of values: Interpolation \(\in\) \{'none', 'bilinear'\}

--- Example ---

* perform camera calibration (with standard calibration plate)
camera_calibration(NX, NY, NZ, NRow, NCol, StartCamParam, NStartPose, 'all', FinalCamParam, NFinalPose, Errors)
* world coordinate system is defined by calibration plate in first image
FinalPose1 := NFinalPose[0:6]
* compensate thickness of plate
set_origin_pose(FinalPose1, 0, 0, 0.0006, WorldPose)
* goal: rectify image
* first determine parameters such that the entire image content is visible
* and that objects have a similar size before and after the rectification
* -> transform image boundary into world plane, determine smallest
* rectangle around it
global_pointer(Image, Pointer, Type, Width, Height)
gen_rectangle1(Image, Pointer, 0, 0, Width-1, Height-1)
gen_contour_region_xld(ImageRect, ImageBorder, 'border')
contour_to_world_plane_xld(ImageBorder, ImageBorderWCS, FinalCamParam, WorldPose, 1)
smallest_rectangle1_xld(ImageBorderWCS, MinY, MinX, MaxY, MaxX)
* -> move the pose to the upper left corner of the surrounding rectangle
set_origin_pose(WorldPose, MinX, MinY, 0, PoseForEntireImage)
* -> determine the scaling factor such that the center pixel has the same
* size in the original and in the rectified image
* method: transform corner points of the pixel into the world
* coordinate system, compute their distances, and use their
* mean as the scaling factor
image_points_to_world_plane(FinalCamParam, PoseForEntireImage,
[Height/2, Height/2, Height/2+1],
[Width/2, Width/2+1, Width/2],
1, WorldPixelX, WorldPixelY)
distance_pp(WorldPixelY[0], WorldPixelX[0], WorldPixelY[1], WorldPixelX[1],
WorldLength1)
distance_pp(WorldPixelY[0], WorldPixelX[0], WorldPixelY[2], WorldPixelX[2],
WorldLength2)
ScaleForSimilarPixelSize := (WorldLength1+WorldLength2)/2
* -> determine output image size such that entire input image fits into it
ExtentX := MaxX-MinX
ExtentY := MaxY-MinY
WidthRectifiedImage := ExtentX/ScaleForSimilarPixelSize
HeightRectifiedImage := ExtentY/ScaleForSimilarPixelSize
* transform the image with the determined parameters
image_to_world_plane(Image, RectifiedImage, FinalCamParam,
PoseForEntireImage, WidthRectifiedImage,
HeightRectifiedImage, ScaleForSimilarPixelSize,
'bilinear')

--- Result ---
ImageToWorldPlane returns TRUE if all parameter values are correct. If necessary, an exception handling is
13.4. CALIBRATION

Parallelization Information

ImageToWorldPlane is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration, SetOriginPose

See also

ContourToWorldPlaneXld, ImagePointsToWorldPlane

Alternatives

GenImageToWorldPlaneMap, MapImage

Camera calibration

Convert a 3D pose into a homogeneous transformation matrix.

PoseToHomMat3d converts a 3D pose Pose, e.g., the exterior camera parameters, into the equivalent homogeneous transformation matrix HomMat3D. For details about 3D poses and the corresponding transformation matrices please refer to CreatePose.

A typical application of PoseToHomMat3d is that you want to further transform the pose, e.g., rotate or translate it using HomMat3dRotate or HomMat3dTranslate. In case of the exterior camera parameters, this can be necessary if the calibration plate cannot be placed such that its coordinate system coincides with the desired world coordinate system.

Parameter

- **Pose** (input control) ................................................. pose ~ VARIANT (integer, real ) 3D pose.
  Number of elements : 7
- **HomMat3D** (output control) ...................................... affine3d ~ VARIANT (real ) Equivalent homogeneous transformation matrix.
  Number of elements : 12

Example

* read interior camera parameters
  read_cam_par(’campar.dat’, CamParam)
* read exterior camera parameters
  read_pose(’startpose.dat’, StartPose)
* (read 3D world points [WorldPointsX,WorldPointsY,WorldPointsZ],
  * extract corresponding 2D image points [PixelsRow,PixelsColumn])
* calibration of exterior camera parameters:
  camera_calibration(WorldPointsX, WorldPointsY, WorldPointsZ,
    PixelsRow, PixelsColumn, CamParam, StartPose, ’pose’,
    FinalCamParam, FinalPose, Errors)
* transform FinalPose to homogeneous transformation matrix
  pose_to_hom_mat3d(FinalPose, cam_H_cal)
* rotate it 90 degree around its y-axis to obtain a world coordinate system
  * whose y- and z-axis lie in the plane of the calibration plate while the
    * x-axis point ‘upwards’: cam_H_w = cam_H_cal * RotY(90)
  hom_mat3d_identity(HomMat3DIdent)
  hom_mat3d_rotate(HomMat3DIdent, deg(90), ’y’, 0, 0, 0,
    HomMat3DRotateY)
  hom_mat3d_compose(cam_H_cal, HomMat3DRotateY, cam_H_w)
PoseToHomMat3d returns TRUE if all parameter values are correct. If necessary, an exception handling is raised parallelization Information

PoseToHomMat3d is reentrant and processed without parallelization.

Possible Predecessors

CameraCalibration, ReadPose

Possible Successors

AffineTransPoint3D, HomMat3dInvert, HomMat3dTranslate, HomMat3dRotate, HomMat3dToPose

See also

CreatePose, CameraCalibration, WritePose, ReadPose, HomMat3dToPose, Project3DPoint, GetLineOfSight, HomMat3dRotate, HomMat3dTranslate, HomMat3dInvert, AffineTransPoint3D

Module

Camera calibration

Project 3D points into (sub-)pixel image coordinates.

Project3DPoint projects one or more 3D points (with coordinates \(X\), \(Y\), and \(Z\)) into the image plane (in pixels) and returns the result in Row and Column. The coordinates \(X\), \(Y\), and \(Z\) are given in the camera coordinate system, i.e., they describe the position of the points relative to the camera.

The interior camera parameters \(\text{CamParam}\) describe the projection characteristics of the camera (see WriteCamPar).

Parameter

- **X** (input control) \(\text{real} \approx \text{VARIANT( real )}\)
  - X coordinates of the 3D points to be projected in the camera coordinate system.
- **Y** (input control) \(\text{real} \approx \text{VARIANT( real )}\)
  - Y coordinates of the 3D points to be projected in the camera coordinate system.
- **Z** (input control) \(\text{real} \approx \text{VARIANT( real )}\)
  - Z coordinates of the 3D points to be projected in the camera coordinate system.
- **CamParam** (input control) \(\text{number} \approx \text{VARIANT( integer, real )}\)
  - Interior camera parameters.
  - **Number of elements** : 8
- **Row** (output control) \(\text{real} \approx \text{VARIANT( real )}\)
  - Row coordinates of the projected points (in pixels).
  - **Default Value** : 'ProjectedRow'
- **Column** (output control) \(\text{real} \approx \text{VARIANT( real )}\)
  - Column coordinates of the projected points (in pixels).
  - **Default Value** : 'ProjectedCol'

Example

* read pose of the world coordinate system in camera coordinates
  read_pose('worldpose.dat', \text{WorldPose})
* convert pose into transformation matrix
  pose_to_hom_mat3d(\text{WorldPose}, \text{HomMat3D})
* transform 3D points from world into the camera coordinate system
  affine_trans_point_3d([[3.0, 3.2], [4.5, 4.5], [5.8, 6.2]], \text{HomMat3D}, \text{X}, \text{Y}, \text{Z})
13.4. CALIBRATION

* read interior camera parameters
read_cam_par('campar.dat', CamParam)
* project 3D points into image
project_3d_point(X, Y, Z, CamParam, Row, Column)

Result

Project3DPoint returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

Project3DPoint is reentrant and processed without parallelization.

Possible Predecessors

ReadCamPar, AffineTransPoint3D

Possible Successors

GenRegionPoints, GenRegionPolygon, DispPolygon

See also

CameraCalibration, DispCaltab, ReadCamPar, GetLineOfSight, AffineTransPoint3D

Module

Camera calibration

Read the interior camera parameters from text file.

ReadCamPar is used to read the interior camera parameters CamParam from a text file with name CamParFile. CamParam is a tuple that contains the interior camera parameters in the following sequence (see WriteCamPar for a description of the corresponding camera models):

[Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]

The format of the text file is a (HALCON-independent) generic parameter description. It allows to group arbitrary sets of parameters hierarchically. The description of a single parameter within a parameter group consists of the following 3 lines:

Name : Shortname : Actual value ;
Type : Lower bound (optional) : Upper bound (optional) ;
Description (optional) ;

WriteCamPar expects in the file CamParFile the parameter group Camera:Parameter. This parameter group consists of the 8 parameters Focus, Sx, Sy, Cx, Cy, Kappa (κ), ImageWidth and ImageHeight. Comments are marked by a '#' at the beginning of a line. A suitable file can look like the following:

# INTERNAL CAMERA PARAMETERS
ParGroup: Camera: Parameter;
"Internal camera parameters"

Focus: foc: 0.00806039;
DOUBLE:0.0; ;
"Focal length of the lens [meter]"

Sx: sx: 1.0629e-05;
DOUBLE:0.0; ;
"Width of a cell on the chip [meter]"

Sy: sy: 1.1e-05;
DOUBLE:0.0;
"Height of a cell on the chip [meter]";

Cx:cx: 378.236;
DOUBLE:0.0;
"X-coordinate of the image center [pixel]";

Cy:cy: 297.587;
DOUBLE:0.0;
"Y-coordinate of the image center [pixel]";

Kappa:kappa: -2253.5;
DOUBLE::;
"Radial distortion coefficient [1/(meter*meter)]";

ImageWidth:imgw: 768;
INT:0:2048;
"Width of the used calibration images [pixel]";

ImageHeight:imgh: 576;
INT:0:2048;
"Height of the used calibration images [pixel]";

Parameter

▷ CamParFile (input control) ........................................string 〜 String / VARIANT

File name of interior camera parameters.

Default Value: 'campar.dat'
List of values: CamParFile ∈ {'campar.dat', 'campar.initial', 'campar.final'}

▷ CamParam (output control) ........................................number 〜 VARIANT (integer, real)

Interior camera parameters.
Number of elements: 8

Example

* get interior camera parameters:
read_cam_par('campar.dat', CamParam)

Result

ReadCamPar returns TRUE if all parameter values are correct and the file has been read successfully. If necessary an exception handling is raised.

Parallelization Information

ReadCamPar is reentrant and processed without parallelization.

Possible Successors

FindMarksAndPose, SimCaltab, CreateCaltab, DispCaltab, CameraCalibration

See also

FindCaltab, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab,
WriteCamPar, WritePose, ReadPose, Project3DPoint, GetLineOfSight

Module

[out] VARIANT Pose HPoseX.ReadPose ([in] String PoseFile )

void HOperatorSetX.ReadPose ([in] VARIANT PoseFile, [out] VARIANT Pose )

Read a 3D pose from a text file.

ReadPose is used to read the 3D pose Pose from a text file with the name PoseFile.
A pose describes a rigid 3D transformation, i.e., a transformation consisting of an arbitrary translation and rotation, with 6 parameters, three for the translation, three for the rotation. With a seventh parameter different pose types can be indicated (see CreatePose).

A suitable file can be generated by the operator WritePose and looks like the following:

```plaintext
# 3D POSE PARAMETERS: rotation and translation
# Used representation type:
f 0
# Rotation angles [deg] or Rodriguezvector:
r -17.8134 1.83816 0.288092
# Translation vector (x y z [m]):
t 0.280164 0.150644 1.7554
```

Parameter PoseFile (input control) .............................................. filename ~ String / VARIANT

File name of the exterior camera parameters.

Default Value : 'campose.dat'

List of values : PoseFile ∈ {'campose.dat', 'campose.initial', 'campose.final'}

Parameter Pose (output control) .............................................. pose ~ VARIANT ( integer, real ) 3D pose.

Number of elements : 7

Result

ReadPose returns TRUE if all parameter values are correct and the file has been read successfully. If necessary an exception handling is raised.

Parallelization Information

ReadPose is reentrant and processed without parallelization.

Possible Predecessors

ReadCamPar

Possible Successors

PoseToHomMat3d, CameraCalibration, DispCaltab, SimCaltab

See also

CreatePose, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, WritePose, PoseToHomMat3d, HomMat3dToPose

Module Camera calibration

```plaintext
```

Translate the origin of a 3D pose.

SetOriginPose translates the origin of the 3D pose PoseIn by the vector given by DX, DY, and DZ and returns the result in PoseNewOrigin. Note that the translation is performed relative to the coordinate system of the pose itself. For example, if PoseIn describes the pose of an object in camera coordinates, PoseNewOrigin is obtained by translating the object’s coordinate system by DX along its own x-axis (and so on for the other axes) and not along the x-axis of the camera coordinate system. This corresponds to the following chain of transformations:

\[
\begin{pmatrix}
1 & 0 & 0 & DX \\
0 & 1 & 1 & DY \\
0 & 0 & 1 & DZ \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
A typical application of this operator when defining a world coordinate system by placing the standard calibration plate on the plane of measurements. In this case, the external camera parameters returned by CameraCalibration correspond to a coordinate system that lies above the measurement plane, because the coordinate system of the calibration plate is located on its surface and the plate has a certain thickness. To correct the pose, call SetOriginPose with the translation vector (0,0,D), where D is the thickness of the calibration plate.

Parameter

- **PoseIn** (input control) ................. pose ~ VARIANT (integer, real)
  
  original 3D pose.

  **Number of elements**: 7

- **DX** (input control) .................. real ~ double / VARIANT
  
  translation of the origin in x-direction.

  **Default Value**: 0

- **DY** (input control) .................. real ~ double / VARIANT
  
  translation of the origin in y-direction.

  **Default Value**: 0

- **DZ** (input control) .................. real ~ double / VARIANT
  
  translation of the origin in z-direction.

  **Default Value**: 0

- **PoseNewOrigin** (output control) ........ pose ~ VARIANT (integer, real)
  
  new 3D pose after applying the translation.

  **Number of elements**: 7

Result

SetOriginPose returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

SetOriginPose is reentrant and processed without parallelization.

Possible Predecessors

CreatePose, HomMat3dToPose, CameraCalibration, HandEyeCalibration

Possible Successors

WritePose, PoseToHomMat3d, ImagePointsToWorldPlane, ContourToWorldPlaneXld

Module

Camera calibration

```
void HImageX.SimCaltab ([in] String CalTabDescrFile,
[in] long GrayCaltab, [in] long GrayMarks, [in] double ScaleFac );
```

```
void HOperatorSetX.SimCaltab ([out] HUntypedObjectX SimImage,
[in] VARIANT GrayBackground, [in] VARIANT GrayCaltab, [in] VARIANT GrayMarks,
[in] VARIANT ScaleFac );
```

Simulate an image with calibration plate.

SimCaltab is used to generate a simulated calibration image. The calibration plate description is read from the file CalTabDescrFile and will be projected into the image plane using the given camera parameters (interior camera parameters CamParam and exterior camera parameters CaltabPose), see also Project3DPoint.

In the simulated image only the calibration plate is shown. The image background is set to the gray value GrayBackground, the calibration plate background is set to GrayCaltab, and the calibration marks are set to the gray value GrayMarks. The parameter ScaleFac influences the number of supporting points to approximate the elliptical contours of the calibration marks, see also DispCaltab. Increasing the number of supporting points causes a more accurate determination of the mark boundary, but increases the computation time, too. For each pixel of the simulated image, which touches a subpixel-boundary of this kind, the gray value is set linearly between GrayMarks and GrayCaltab dependent on the proportion Inside/Outside.

By applying the operator SimCaltab you can generate synthetic calibration images (with known camera parameters!) to test the quality of the calibration algorithm (see CameraCalibration).
13.4. CALIBRATION

Parameter

- **SimImage** (output iconic) ......................... image \( \sim HImageX / HUntypedObjectX \) (byte)  
  Simulated calibration image.

- **CalTabDescrFile** (input control) .................. string \( \sim String / VARIANT \)  
  File name of the calibration plate description.
  Default Value: ’caltab.descr’

- **CamParam** (input control) ........................... number \( \sim VARIANT \) (integer, real)  
  Interior camera parameters.
  Number of elements: 8

- **CaltabPose** (input control) .......................... pose \( \sim VARIANT \) (integer, real)  
  Exterior camera parameters (3D pose of the calibration plate in camera coordinates).
  Number of elements: 7

- **GrayBackground** (input control) ...................... integer \( \sim long / VARIANT \)  
  Gray value of image background.
  Default Value: 128
  Suggested values: GrayBackground \( \in \{0, 32, 64, 96, 128, 160\} \)
  Restriction: \((0 \leq \text{GrayBackground} \leq 255)\)

- **GrayCaltab** (input control) ............................ integer \( \sim long / VARIANT \)  
  Gray value of calibration plate.
  Default Value: 224
  Suggested values: GrayCaltab \( \in \{144, 160, 176, 192, 208, 224, 240\} \)
  Restriction: \((0 \leq \text{GrayCaltab} \leq 255)\)

- **GrayMarks** (input control) ............................ integer \( \sim long / VARIANT \)  
  Gray value of calibration marks.
  Default Value: 80
  Suggested values: GrayMarks \( \in \{16, 32, 48, 64, 80, 96, 112\} \)
  Restriction: \((0 \leq \text{GrayMarks} \leq 255)\)

- **ScaleFac** (input control) ............................. real \( \sim double / VARIANT \)  
  Scaling factor to reduce oversampling.
  Default Value: 1.0
  Suggested values: ScaleFac \( \in \{1.0, 0.5, 0.25, 0.125\} \)
  Recommended Increment: 0.05
  Restriction: \((1.0 \geq \text{ScaleFac})\)

Example

* read calibration image
read_image(Image1, ’calib-01’)
* find calibration pattern
find_caltab(Image1, Caltab1, ’caltab.descr’, 3, 112, 5)
* find calibration marks and initial pose
StartCamPar := [Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]
find_marks_and_pose(Image1, Caltab1, ’caltab.descr’, StartCamPar,
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1,
                    StartPose1)
* read 3D positions of calibration marks
caltab_points(’caltab.descr’, NX, NY, NZ)
* camera calibration
camera_calibration(NX, NY, NZ, RCoord1, CCoord1, StartCamPar,
                    StartPose1, 11, CamParam, FinalPose, Errors)
* simulate calibration image
sim_caltab(Image1Sim, ’caltab.descr’, CamParam, FinalPose, 128, 224, 80, 1)

Result

SimCaltab returns TRUE if all parameter values are correct. If necessary, an exception handling is raised.

Parallelization Information

SimCaltab is processed under mutual exclusion against itself and without parallelization.
Write the interior camera parameters to text file.

WriteCamPar is used to write the interior camera parameters \texttt{CamParam} to a text file with name \texttt{CamParFile}. \texttt{CamParam} is a tuple that contains the interior camera parameters in the following sequence:

\begin{itemize}
\item [\textit{Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight}]
\end{itemize}

The interior camera parameters describe the projection process of the used combination of camera, lens, and frame grabber; they can be determined calibrating the camera, see \texttt{CameraCalibration}.

The projection of a point \(p^c\) given in camera coordinates into a (sub-)pixel \([r,c]\) in the image consists of multiple steps: First, the point is projected into the image plane, i.e., onto the sensor chip. If the underlying camera model is a \textit{pinhole camera with radial distortions}, i.e., if the focal length passed in \texttt{CamParam} is greater than 0, the projection is described by the following equations:

\[
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]

\[
\begin{aligned}
  u &= \text{Focus} \cdot \frac{x}{z} \\
  v &= \text{Focus} \cdot \frac{y}{z}
\end{aligned}
\]

In contrast, if the focal length is passed as 0 in \texttt{CamParam}, the camera model of a \textit{telecentric camera with radial distortions} is used, i.e., it is assumed that the optics of the lens of the camera performs a parallel projection. In this case, the corresponding equations are:

\[
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\]

\[
\begin{aligned}
  u &= x \\
  v &= y
\end{aligned}
\]

The following equations compensate for radial distortion:

\[
\begin{aligned}
  \tilde{u} &= \frac{2u}{1 + \sqrt{1 - 4\kappa(u^2 + v^2)}} \\
  \tilde{v} &= \frac{2v}{1 + \sqrt{1 - 4\kappa(u^2 + v^2)}}
\end{aligned}
\]

Finally, the point is transformed from the image plane coordinate system into the image coordinate system, i.e., the pixel coordinate system:

\[
\begin{aligned}
  c &= \frac{\tilde{u}}{S_x} + C_x \\
  r &= \frac{\tilde{v}}{S_y} + C_y
\end{aligned}
\]

The format of the text file is a (HALCON-independent) generic parameter description. It allows to group arbitrary sets of parameters hierarchically. The description of a single parameter within a parameter group consists of the following 3 lines:
Name : Shortname : Actual value ;
Type : Lower bound (optional) : Upper bound (optional) ;
Description (optional) ;

The operator WriteCamPar writes the parameter group Camera:Parameter into the text file CamParFile. This parameter group consists of the 8 interior camera parameters, but in a different sequence than in CamParam (see ReadCamPar for an example).

Parameter

- **CamParam** (input control) ........................................... number ~ VARIANT ( integer, real )
  Interior camera parameters.
  Number of elements : 8

- **CamParFile** (input control) ........................................... string ~ String / VARIANT
  File name of interior camera parameters.
  Default Value : 'campar.dat'
  List of values : CamParFile ∈ {'campar.dat', 'campar.initial', 'campar.final'}

Example

```plaintext
* read calibration images
read_image(Image1, 'calib-01')
read_image(Image2, 'calib-02')
read_image(Image3, 'calib-03')

* find calibration pattern
find_caltab(Image1, Caltab1, 'caltab.descr', 3, 112, 5)
find_caltab(Image2, Caltab2, 'caltab.descr', 3, 112, 5)
find_caltab(Image3, Caltab3, 'caltab.descr', 3, 112, 5)

* find calibration marks and start poses
StartCamPar := [Focus, Kappa, Sx, Sy, Cx, Cy, ImageWidth, ImageHeight]
find_marks_and_pose(Image1, Caltab1, 'caltab.descr', StartCamPar,
                     128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1,
                     StartPose1)
find_marks_and_pose(Image2, Caltab2, 'caltab.descr', StartCamPar,
                     128, 10, 18, 0.9, 15.0, 100.0, RCoord2, CCoord2,
                     StartPose2)
find_marks_and_pose(Image3, Caltab3, 'caltab.descr', StartCamPar,
                     128, 10, 18, 0.9, 15.0, 100.0, RCoord3, CCoord3,
                     StartPose3)

* read 3D positions of calibration marks
caltab_points('caltab.descr', NX, NY, NZ)

* camera calibration
camera_calibration(NX, NY, NZ, [RCoord1, RCoord2, RCoord3],
                     [CCoord1, CCoord2, CCoord3], StartCamPar,
                     [StartPose1, StartPose2, StartPose3], 'all',
                     CamParam, NFinalPose, Errors)

* write interior camera parameters to file
write_cam_par(CamParam, 'campar.dat')
```

Result

WriteCamPar returns TRUE if all parameter values are correct and the file has been written successfully. If necessary an exception handling is raised.

Parallelization Information

WriteCamPar is local and processed completely exclusively without parallelization.

Possible Predecessors

CameraCalibration

See also

FindCaltab, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab,
ReadCamPar, WritePose, ReadPose, Project3DPoint, GetLineOfSight

Module

Camera calibration
Write a 3D pose to a text file.

WritePose is used to write a 3D pose Pose into a text file with the name PoseFile.

A pose describes a rigid 3D transformation, i.e., a transformation consisting of an arbitrary translation and rotation, with 6 parameters, three for the translation, three for the rotation. With a seventh parameter different pose types can be indicated (see CreatePose).

A file generated by WritePose looks like the following:

```
# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
f 0

# Rotation angles [deg] or Rodriguez vector:
r -17.8134 1.83816 0.288092

# Translation vector (x y z [m]):
t 0.280164 0.150644 1.7554
```

Parameter

- **Pose** (input control) .......................... pose \(\sim\) VARIANT (integer, real) 3D pose.
- Number of elements: 7
- **PoseFile** (input control) ....................... filename \(\sim\) String / VARIANT File name of the exterior camera parameters.
  - Default Value: 'campose.dat'
  - List of values: PoseFile \(\in\) {'campose.dat', 'campose.initial', 'campose.final'}

Example

```
* read calibration images
read_image(Image1, 'calib-01')
read_image(Image2, 'calib-02')
read_image(Image3, 'calib-03')
* find calibration pattern
find_caltab(Image1, Caltab1, 'caltab.descr', 3, 112, 5)
find_caltab(Image2, Caltab2, 'caltab.descr', 3, 112, 5)
find_caltab(Image3, Caltab3, 'caltab.descr', 3, 112, 5)
* find calibration marks and start poses
StartCamPar := [0.008, 0.0, 0.000011, 0.000011, 384, 288, 768, 576]
find_marks_and_pose(Image1, Caltab1, 'caltab.descr', StartCamPar,
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord1, CCoord1,
                    StartPose1)
find_marks_and_pose(Image2, Caltab2, 'caltab.descr', StartCamPar,
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord2, CCoord2,
                    StartPose2)
find_marks_and_pose(Image3, Caltab3, 'caltab.descr', StartCamPar,
                    128, 10, 18, 0.9, 15.0, 100.0, RCoord3, CCoord3,
                    StartPose3)
```

HALCON/COM Reference Manual, 2005-2-1
13.5 Fourier-Descriptor

WritePose returns TRUE if all parameter values are correct and the file has been written successfully. If necessary an exception handling is raised.

Parallelization Information
WritePose is local and processed completely exclusively without parallelization.

Possible Predecessors
CameraCalibration, HomMat3dToPose
See also
CreatePose, FindMarksAndPose, CameraCalibration, DispCaltab, SimCaltab, ReadPose, PoseToHomMat3d, HomMat3dToPose

Module

13.5 Fourier-Descriptor

Normalizing of the Fourier coefficients with respect to the displacement of the starting point.
The operator AbsInvarFourierCoeff normalizes the Fourier coefficients with regard to the displacements of the starting point. These occur when an object is rotated. The contour tracer GetRegionContour starts with recording the contour in the upper lefthand corner of the region and follows the contour clockwise. If the object is rotated, the starting value for the contour point chain is different which leads to a phase shift in the frequency space. The following two kinds of normalizing are available:

- **abs_amount**: The phase information will be eliminated; the normalizing does not retain the structure, i.e. if the AZ-invariants are backtransformed, no similarity with the pattern can be recognized anymore.
- **az_invar1**: AZ-invariants of the 1st order execute the normalizing with respect to displacing the starting point so that the structure is retained; they are however more prone to local and global disturbances, in particular to projective distortions.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RealInvar</strong> (input control) real</td>
</tr>
<tr>
<td><strong>ImaginaryInvar</strong> (input control) real</td>
</tr>
<tr>
<td><strong>CoefP</strong> (input control) integer</td>
</tr>
<tr>
<td><strong>CoefQ</strong> (input control) integer</td>
</tr>
</tbody>
</table>

Default Value : 1
Suggested values : CoefP ∈ {1, 2}
Restriction : (CoefP ≥ 1)
CHAPTER 13. TOOLS

- **AZInvar** (input control) string ~ String / VARIANT
  Order of the AZ-invariants.
  
  **Default Value**: ‘abs_amount’
  
  **List of values**: AZInvar ∈ {'abs_amount', 'az_invar1'}

- **RealAbsInvar** (output control) real ~ VARIANT(real)
  Real parts of the normalized Fourier coefficients.

- **ImaginaryAbsInvar** (output control) real ~ VARIANT(real)
  Imaginary parts of the normalized Fourier coefficients.

---

**Parallelization Information**

AbsInvarFourierCoeff is reentrant and processed without parallelization.

---

**Possible Predecessors**

InvarFourierCoeff

---

**Possible Successors**

Fourier1DimInv, MatchFourierCoeff

---

**Module**

Fourier descriptors

---

**Parallelization Information**

Fourier1Dim is reentrant and processed without parallelization.

---

**Calculate the Fourier coefficients of a parameterized contour.**

The operator **Fourier1Dim** calculates the Fourier coefficients of a parameterized contour by using a valid parameter scale. This parameter scale may, for instance, be created with the help of the procedure PrepContourFourier. This function serves to calculate the Fourier coefficients of closed contours which are treated like complex-valued curves. Therefore, in order to determine the Fourier coefficients, the Fourier transform for periodical functions is used. Hereby the parameter MaxCoef determines the absolute value +1 of the maximal number of Fourier coefficients, i.e. if \( n \) coefficients are indicated, the procedure will calculate coefficients ranging from \(-n\) to \(n\). The contour will be approximated without loss, if \( n \approx \text{number of the contour points} \), whereby \( n = 100 \) approximates the contour so well that an error can hardly be distinguished; \( n \in [40, 50] \) however is sufficient for most applications. If the parameter MaxCoef is set to 0, all coefficients will be determined.

---

**Parameter**

- **Rows** (input control) contour.y ~ VARIANT(integer)
  Row coordinates of the contour.

- **Columns** (input control) contour.x ~ VARIANT(integer)
  Column coordinates of the contour.

- **ParContour** (input control) real ~ VARIANT(real)
  Parameter scale.

- **MaxCoef** (input control) integer ~ long / VARIANT
  Desired number of Fourier coefficients or all of them (0).
  **Default Value**: 50
  **Suggested values**: MaxCoef ∈ {0, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 400}
  **Restriction**: \((\text{MaxCoef} \geq 0)\)

---

**Parallelization Information**

Fourier1Dim is reentrant and processed without parallelization.
13.5. FOURIER-DESCRIPTOR

Fourier descriptors

Possible Predecessors
PrepContourFourier

Possible Successors
InvarFourierCoeff, DispPolygon

Module

Fourier descriptors

One dimensional Fourier synthesis (inverse Fourier transform).
Backtransformation of Fourier coefficients respectively of Fourier descriptors. The number of values to be backtransformed should not exceed the length of the transformed contour.

Parameter

- **RealCoef** (input control) .......................................................... real  \( \rightsquigarrow \) VARIANT (real)
  Real parts.
- **ImaginaryCoef** (input control) .................................................. real  \( \rightsquigarrow \) VARIANT (real)
  Imaginary parts.
- **MaxCoef** (input control) ........................................................... integer  \( \rightsquigarrow \) long / VARIANT
  Input of the steps for the backtransformation.

Default Value: 100
Suggested values: MaxCoef \( \in \{5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 400\}\)
Restriction: (MaxCoef \( \geq \) 1)
- **Rows** (output control) .............................................................. contour.y  \( \rightsquigarrow \) VARIANT (real)
  Row coordinates.
- **Columns** (output control) .......................................................... contour.x  \( \rightsquigarrow \) VARIANT (real)
  Column coordinates.

Parallelization Information

Fourier1DimInv is reentrant and processed without parallelization.

Possible Predecessors
InvarFourierCoeff, Fourier1Dim

Possible Successors
DispPolygon

Module

Fourier descriptors

Normalize the Fourier coefficients.
Elimination of affine information from the Fourier coefficients, determination of affine invariants. The Fourier coefficients will be normalized suitably so that all affine correlated contours will be projected to one and the same contour. The following levels of affine mappings are available:
1. Translations (InvarType = 'transl_invar')
2. + Rotations (InvarType = 'congr_invar')
3. + Scalings (InvarType = 'simil_invar')
4. + Slanting (InvarType = 'affine_invar')

The control parameter InvarType indicates up to which level the affine representation shall be normalized. Please note that indicating a certain level implies that the normalizing is executed with regard to all levels below. For most applications a subsequent normalizing of the starting point is recommended!

Parameter

RealCoef (input control) .................................................. real  \(\leadsto\) VARIANT (real)
Real parts of the Fourier coefficients.

ImaginaryCoef (input control) ................................. real  \(\leadsto\) VARIANT (real)
Imaginary parts of the Fourier coefficients.

NormPar (input control) ................................ integer  \(\leadsto\) long / VARIANT
Input of the normalizing coefficients.
Default Value: 1
Suggested values: NormPar \(\in\) \{1, 2\}
Restriction: (NormPar \(\geq\) 1)

InvarType (input control) ............................................... string  \(\leadsto\) String / VARIANT
Indicates the level of the affine mappings.
Default Value: 'affine_invar'
List of values: InvarType \(\in\) \{'affine_invar', 'simil_invar', 'congr_invar', 'transl_invar'\}

RealInvar (output control) ....................................... real  \(\leadsto\) VARIANT (real)
Real parts of the normalized Fourier coefficients.

ImaginaryInvar (output control) ......................... real  \(\leadsto\) VARIANT (real)
Imaginary parts of the normalized Fourier coefficients.

Parallelization Information

InvarFourierCoeff is reentrant and processed without parallelization.

Possible Predecessors

Fourier1Dim

Possible Successors

InvarFourierCoeff

Module

Fourier descriptors

Similarity of two contours.

The operator MatchFourierCoeff calculates the Euclidean distance between two contours which are available as Fourier coefficients. In order to avoid that the higher frequencies are in some way too dominant, the following attenuation can be used:

none: No attenuation.

I/index: Absolute amounts of the Fourier coefficients will be divided by their index.

I/(index*index): Absolute amounts of the Fourier coefficients will be divided by their square index.
The higher the result value, the greater the differences between the pattern and the test contour. If the number of coefficients is not the same, only the first $n$ coefficients will be compared. The parameter $\text{MaxCoef}$ indicates the number of the coefficients to be compared. If $\text{MaxCoef}$ is set to zero, all coefficients will be used.

\begin{itemize}
\item[$\triangleright$] $\text{RealCoef1}$ (input control) \hspace{1cm} real $\sim$ VARIANT (real)
Real parts of the pattern Fourier coefficients.
\item[$\triangleright$] $\text{ImaginaryCoef1}$ (input control) \hspace{1cm} real $\sim$ VARIANT (real)
Imaginary parts of the pattern Fourier coefficients.
\item[$\triangleright$] $\text{RealCoef2}$ (input control) \hspace{1cm} real $\sim$ VARIANT (real)
Real parts of the Fourier coefficients to be compared.
\item[$\triangleright$] $\text{ImaginaryCoef2}$ (input control) \hspace{1cm} real $\sim$ VARIANT (real)
Imaginary parts of the Fourier coefficients to be compared.
\item[$\triangleright$] $\text{MaxCoef}$ (input control) \hspace{1cm} integer $\sim$ long / VARIANT
Total number of Fourier coefficients.
\end{itemize}

**Default Value**: 50

**Suggested values**: $\text{MaxCoef} \in \{0, 5, 10, 15, 20, 30, 40, 50, 70, 100, 200, 400\}$

**Restriction**: $(\text{MaxCoef} \geq 0)$

\begin{itemize}
\item[$\triangleright$] $\text{Damping}$ (input control) \hspace{1cm} string $\sim$ String / VARIANT
Kind of attenuation.
\item[$\triangleright$] $\text{Distance}$ (output control) \hspace{1cm} real $\sim$ double / VARIANT
Similarity of the contours.
\end{itemize}

**Parallelization Information**

$\text{MatchFourierCoeff}$ is reentrant and processed without parallelization.

**Possible Predecessors**

$\text{InvarFourierCoeff}$

**Module**

Fourier descriptors

\begin{verbatim}

\end{verbatim}

Transformation of the origin into the centre of gravity.

The operator $\text{MoveContourOrig}$ relocates the input contour so that the origin lies in the centre of gravity.

\begin{itemize}
\item[$\triangleright$] $\text{Rows}$ (input control) \hspace{1cm} contour.y $\sim$ VARIANT (integer)
Row coordinates of the contour.
\item[$\triangleright$] $\text{Columns}$ (input control) \hspace{1cm} contour.x $\sim$ VARIANT (integer)
Column coordinates of the contour.
\item[$\triangleright$] $\text{RowsMoved}$ (output control) \hspace{1cm} contour.y $\sim$ VARIANT (integer)
Row coordinates of the displaced contour.
\item[$\triangleright$] $\text{ColumnsMoved}$ (output control) \hspace{1cm} contour.x $\sim$ VARIANT (integer)
Column coordinates of the displaced contour.
\end{itemize}

**Parallelization Information**

$\text{MoveContourOrig}$ is processed completely exclusively without parallelization.

**Possible Predecessors**

$\text{GetRegionContour}$

**Possible Successors**

$\text{PrepContourFourier}$
Fourier descriptors

Parameterize the passed contour.
The operator PrepContourFourier parameterizes the transmitted contour in order to prepare it for the one-dimensional Fourier transformation. Hereby the contour must be available in closed form. Three parameter functions are available for the control parameter TransMode:

- **arc**: Parameterization by the radian.
- **signed_area**: Parameterization by the signed area.
- **unsigned_area**: Parameterization by the absolute area.

Please note that in contrast to the signed or unsigned area the affine mapping of the radian will not be transformed linearly.

### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rows</strong></td>
<td>(input control)contour.y ~ VARIANT(integer) Row indices of the contour.</td>
</tr>
<tr>
<td><strong>Columns</strong></td>
<td>(input control)contour.x ~ VARIANT(integer) Column indices of the contour.</td>
</tr>
<tr>
<td><strong>TransMode</strong></td>
<td>(input control)string ~ String/VARIANT Kind of parameterization.</td>
</tr>
<tr>
<td><strong>Default Value</strong>:</td>
<td>'signed_area'</td>
</tr>
<tr>
<td><strong>Suggested values</strong>:</td>
<td>TransMode ∈ {'arc', 'unsigned_area', 'signed_area'}</td>
</tr>
<tr>
<td><strong>ParContour</strong></td>
<td>(output control)real ~ VARIANT(real) Parameterized contour.</td>
</tr>
</tbody>
</table>

### Parallelization Information

PrepContourFourier is reentrant and processed without parallelization.

### Possible Predecessors

- MoveContourOrig

### Possible Successors

- Fourier1Dim

### Module

Fourier descriptors

### 13.6 Function

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FunctionAbsolute</strong></td>
<td>HFunction1dX.AbsFunct1D( ) Absolute value of the y values.</td>
</tr>
<tr>
<td><strong>Function</strong></td>
<td>HOperatorSetX.AbsFunct1D([in] VARIANT Function, [out] VARIANT FunctionAbsolute)</td>
</tr>
</tbody>
</table>

AbsFunct1D calculates the absolute values of all y values of Function.
13.6. FUNCTION

Parameter

- **Function** (input control) ................. function\_id \( \rightsquigarrow \) \texttt{HFunction1dX / VARIANT ( real, integer )}
  
  Input function.

- **FunctionAbsolute** (output control) ........ function\_id \( \rightsquigarrow \) \texttt{HFunction1dX / VARIANT ( real, integer )}
  
  Function with the absolute values of the y values.

Parallelization Information

\texttt{AbsFunct1D} is reentrant and processed without parallelization.

Possible Predecessors

CreateFunct1DPairs, CreateFunct1DArray

Module

Tools

```c
void \texttt{HFunction1dX.CreateFunct1DArray} ([in] VARIANT YValues )
```

```c
void \texttt{HOperatorSetX.CreateFunct1DArray} ([in] VARIANT YValues, [out] VARIANT Function )
```

Create a function from a sequence of y-values.

\texttt{CreateFunct1DArray} creates a one-dimensional function from a set of y-values \texttt{YValues}. The resulting function can then be processed and analyzed with the operators for 1d functions. \texttt{YValues} is interpreted as follows: the first value of \texttt{YValues} is the function value at zero, the second value is the function value at one, etc. Thus, the values define a function at equidistant x values (with distance 1), starting at 0.

Alternatively, the operator \texttt{CreateFunct1DPairs} can be used to create a function. \texttt{CreateFunct1DPairs} also allows to define a function with non-equidistant x values by specifying them explicitly. Thus to get the same definition as with \texttt{CreateFunct1DArray}, one would pass a tuple of x values to \texttt{CreateFunct1DPairs} that has the same length as \texttt{YValues} and contains values starting at 0 and increasing by 1 in each position. Note, however, that \texttt{CreateFunct1DPairs} leads to a different internal representation of the function which needs more storage (because all (x,y) pairs are stored) and sometimes cannot be processed as efficiently as functions created by \texttt{CreateFunct1DArray}.

Parallelization Information

\texttt{CreateFunct1DArray} is reentrant and processed without parallelization.

Possible Successors

WriteFunct1D, GnuplotPlotFunct1D, YRangeFunct1D, GetPairFunct1D, TransformFunct1D

See also

Funct1DToPairs

Alternatives

CreateFunct1DPairs, ReadFunct1D

Module

Tools

```c
void \texttt{HFunction1dX.CreateFunct1DPairs} ([in] VARIANT XValues, [in] VARIANT YValues )
```

```c
void \texttt{HOperatorSetX.CreateFunct1DPairs} ([in] VARIANT XValues, [in] VARIANT YValues, [out] VARIANT Function )
```

Create a function from a set of (x,y) pairs.
CreateFunct1DPairs creates a one-dimensional function from a set of pairs of \((x, y)\) values. The \(XValues\) of the functions have to be passed in ascending order. The resulting function can then be processed and analyzed with the operators for 1d functions.

Alternatively, functions can be created with the operator CreateFunct1DArray. In contrast to this operator, \(x\) values with arbitrary positions can be specified with CreateFunct1DPairs. Hence, it is the more general operator. It should be noted, however, that because of this generality the processing of a function created with CreateFunct1DPairs cannot be carried out as efficiently as for equidistant functions. In particular, not all operators accept such functions. If necessary, a function can be transformed into an equidistant function with the operator SampleFunct1D.

\[
\begin{align*}
\text{Parameter} & \\
\text{XValues (input control)} & \text{number(-array) } \sim \text{VARIANT (real, integer)} \\
\text{YValues (input control)} & \text{number(-array) } \sim \text{VARIANT (real, integer)} \\
\text{Function (output control)} & \text{function} \\
\end{align*}
\]

CreateFunct1DPairs is \textit{reentrant} and processed without parallelization.

\[
\begin{align*}
\text{Possible Successors} & \\
\text{WriteFunct1D, GnuplotPlotFunct1D, YRangeFunct1D, GetPairFunct1D} & \text{See also} \\
\text{Funct1DToPairs} & \text{Alternatives} \\
\text{CreateFunct1DArray, ReadFunct1D} & \text{Module} \\
\text{Tools} & \\
\end{align*}
\]

\[
\begin{align*}
\text{[out] VARIANT Distance HFunction1dX.DistanceFunct1D} \\
([\text{in}] \text{HFunction1dX Function2, [in] VARIANT Mode, [in] VARIANT Sigma} )
\end{align*}
\]

\[
\begin{align*}
\text{void HOperatorSetX.DistanceFunct1D ([in] VARIANT Function1,} \\
\text{[in] VARIANT Function2, [in] VARIANT Mode, [in] VARIANT Sigma,} \\
\text{[out] VARIANT Distance} )
\end{align*}
\]

Compute the distance of two functions.

\[
\text{DistanceFunct1D} \text{ calculates the distance of two functions. The two functions may differ in length.}
\]\n
\[
\begin{align*}
\text{Parameter} & \\
\text{Function1 (input control)} & \text{function} \\
\text{Function2 (input control)} & \text{function} \\
\text{Mode (input control)} & \text{string(-array) } \sim \text{VARIANT (string)} \\
\text{Default Value : 'length'} \\
\text{List of values : Mode } \in \{ \text{'length', 'mean'} \} \\
\text{Sigma (input control)} & \text{number(-array) } \sim \text{VARIANT (real)} \\
\text{Default Value : 0.0} \\
\text{Suggested values : Sigma } \in \{ 0.0, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0, 15.0, 20.0, 25.0, 30.0, 40.0, 50.0 \} \\
\text{Distance (output control)} & \text{real } \sim \text{VARIANT (real, integer)} \\
\end{align*}
\]

DistanceFunct1D is \textit{reentrant} and processed without parallelization.
13.6. FUNCTION

Access to the x/y values of a function.

Funct1DToPairs splits the input function Function into tuples for the x and y values.

**Parallelization Information**
Funct1DToPairs is reentrant and processed without parallelization.

**Possible Predecessors**
CreateFunct1DPairs, CreateFunct1DArray

**Tools**

```
[out] VARIANT XValues HFunction1dX.Funct1DToPairs
([out] VARIANT YValues )
```

```
void HOperatorSetX.Funct1DToPairs ([in] VARIANT Function,
[out] VARIANT XValues, [out] VARIANT YValues )
```

Access a function value using the index of the control points.

GetPairFunct1D accesses a function value of Function. This is done by specifying the index of one or more control points of the function.

**Parallelization Information**
GetPairFunct1D is reentrant and processed without parallelization.

**Possible Predecessors**
CreateFunct1DPairs, CreateFunct1DArray

**Tools**

```
[out] VARIANT X HFunction1dX.GetPairFunct1D ([in] VARIANT Index,
[out] VARIANT Y )
```

```
void HOperatorSetX.GetPairFunct1D ([in] VARIANT Function,
```
CHAPTER 13. TOOLS

Compute the positive and negative areas of a function.

IntegrateFunct1D integrates the function Function (see CreateFunct1DArray and CreateFunct1DPairs) and returns the integral of the positive and negative parts of the function in Positive and Negative, respectively. Hence, the integral of the function is the difference Positive - Negative. The integration is done on the interval on which the function is defined. For the integration, the function is interpolated linearly.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function (input control)</td>
</tr>
<tr>
<td>Positive (output control)</td>
</tr>
<tr>
<td>Negative (output control)</td>
</tr>
</tbody>
</table>

Parallelization Information

IntegrateFunct1D is reentrant and processed without parallelization.

Possible Predecessors

CreateFunct1DArray, CreateFunct1DPairs

See also

CreateFunct1DArray, CreateFunct1DPairs

Module

Tools

Calculate transformation parameters between two functions.

MatchFunct1DTrans calculates the transformation parameters between two functions given as the tuples Function1 and Function2 (see CreateFunct1DArray and CreateFunct1DPairs). The following model is used for the transformation between the two functions:

\[ y_1(x) = a_1 y_2(a_3 x + a_4) + a_2. \]

The transformation parameters are determined by a least-squares minimization of the following function:

\[ \sum_{i=0}^{n-1} (y_1(x_i) - a_1 y_2(a_3 x_i + a_4) + a_2)^2. \]

The values of the function \( y_2 \) are obtained by linear interpolation. The parameter Border determines the values of the function Function2 outside of its domain. For Border=’zero’ these values are set to 0, for Border=’constant’ they are set to the corresponding value at the border, for Border=’mirror’ they are mirrored at the border, and for Border=’cyclic’ they are continued cyclically. The calculated transformation parameters are returned as a 4-tuple in Params. If some of the parameter values are known, the respective parameters can
be excluded from the least-squares adjustment by setting the corresponding value in the tuple `UseParams` to the value 'false'. In this case, the tuple `ParamsConst` must contain the known value of the respective parameter. If a parameter is used for the adjustment (`UseParams = 'true'`), the corresponding parameter in `ParamsConst` is ignored. On output, `MatchFunct1DTrans` additionally returns the sum of the squared errors `ChiSquare` of the resulting function, i.e., the function obtained by transforming the input function with the transformation parameters, as well as the covariance matrix `Covar` of the transformation parameters `Params`. These parameters can be used to decide whether a successful matching of the functions was possible.

- **Function 1** (input control) .......................... `function_id` ➔ `HFunction1dX / VARIANT` (real, integer) 
  Function 1.

- **Function 2** (input control) .......................... `function_id` ➔ `HFunction1dX / VARIANT` (real, integer) 
  Function 2.

- **Border** (input control) ................................. `string` ➔ `String / VARIANT` 
  Border treatment for function 2.

  - **Default Value:** 'constant'
  - **List of values:** `Border` ∈ {'zero', 'constant', 'mirror', 'cyclic'}

- **ParamsConst** (input control) ............................ `number` ➔ `VARIANT` (real) 
  Values of the parameters to remain constant.

  - **Default Value:** `[1.0,0.0,1.0,0.0]`
  - **Number of elements:** 4

- **UseParams** (input control) .............................. `string` ➔ `VARIANT` (string) 
  Should a parameter be adapted for it?

  - **Default Value:** `['true','true','true','true']`
  - **List of values:** `UseParams` ∈ {'true', 'false'}

  - **Number of elements:** 4

- **Params** (output control) ............................... `number` ➔ `VARIANT` (real) 
  Transformation parameters between the functions.

  - **Number of elements:** 4

- **ChiSquare** (output control) ............................ `number` ➔ `double / VARIANT` 
  Quadratic error of the output function.

- **Covar** (output control) ............................... `number` ➔ `VARIANT` (real) 
  Covariance Matrix of the transformation parameters.

  - **Number of elements:** 16

**Parallelization Information**

`MatchFunct1DTrans` is reentrant and processed without parallelization.

**Possible Predecessors**

`CreateFunct1DArray, CreateFunct1DPairs`

See also

`GrayProjections`

**Module**

Tools

```c
[out] HFunction1dX FunctionInverted HFunction1dX.NegateFunct1D ( )

void HOperatorSetX.NegateFunct1D ([in] VARIANT Function,
[out] VARIANT FunctionInverted )
```

Negation of the y values.

`NegateFunct1D` negates all y values of `Function`. 

**Parameter**

- **Function** (input control) ............................. `function_id` ➔ `HFunction1dX / VARIANT` (real, integer) 
  Input function.

- **FunctionInverted** (output control) .................... `function_id` ➔ `HFunction1dX / VARIANT` (real, integer) 
  Function with the negated y values.
Parallelization Information

NegateFunct1D is reentrant and processed without parallelization.

Possible Predecessors

CreateFunct1DPairs, CreateFunct1DArray

Module

Tools

[out] long Length HFunction1dX.NumPointsFunct1D ( )

void HOperatorSetX.NumPointsFunct1D ([in] VARIANT Function,
[out] VARIANT Length )

Number of control points of the function.

NumPointsFunct1D calculates the number of control points of Function.

Parameter

▷ Function (input control) ................. function Id ~ HFunction1dX / VARIANT ( real, integer )
Input function.

▷ Length (output control) .................. integer ~ long / VARIANT
Number of control points.

Parallelization Information

NumPointsFunct1D is reentrant and processed without parallelization.

Possible Predecessors

CreateFunct1DPairs, CreateFunct1DArray

Module

Tools

void HFunction1dX.ReadFunct1D ([in] String FileName )

void HOperatorSetX.ReadFunct1D ([in] VARIANT FileName,
[out] VARIANT Function )

Read a function from a file.

The operator ReadFunct1D reads the contents of FileName and converts it into the function Function. The file has be generated by WriteFunct1D.

Parameter

▷ FileName (input control) ................... filename ~ String / VARIANT
Name of the file to be read.

▷ Function (output control) .......... function Id(-array) ~ HFunction1dX / VARIANT ( real, integer )
Function from the file.

Result

If the parameters are correct the operator ReadFunct1D returns the value TRUE. If the file could not be opened ReadFunct1D returns FAIL. Otherwise an exception handling is raised.

Parallelization Information

ReadFunct1D is reentrant and processed without parallelization.

See also

WriteFunct1D, GnuPlotPlotCtrl, WriteImage, WriteRegion, OpenFile

Alternatives

FreadString, ReadTuple

Module

Basic operators
Sample a function equidistantly in an interval.

SampleFunct1D samples the input function Function in the interval \([XMin, XMax]\) at equidistant points with the distance \(XDist\). The last point lies in the interval if \(XMax - XMin\) is not an integer multiple of \(XDist\). To obtain the samples, the input function is interpolated linearly. The parameter Border determines the values of the function Function outside of its domain. For Border='zero' these values are set to 0, for Border='constant' they are set to the corresponding value at the border, for Border='mirror' they are mirrored at the border, and for Border='cyclic' they are continued cyclically.

Parameter

- **Function** (input control) ................. function \(1\text{d}\)  \(\sim\) HFunction1dX / VARIANT ( real, integer )
  Input function.
- **XMin** (input control) ......................... number  \(\sim\) VARIANT ( real, integer )
  Minimum x value of the output function.
- **XMax** (input control) ......................... number  \(\sim\) VARIANT ( real, integer )
  Maximum x value of the output function.
  Restriction: \((XMax > XMin)\)
- **XDist** (input control) ......................... number  \(\sim\) VARIANT ( real, integer )
  Distance of the samples.
  Restriction: \((XDist > 0)\)
- **Border** (input control) ...................... string  \(\sim\) String / VARIANT
  Border treatment for the input function.
  Default Value: 'constant'
  List of values: Border \(\in\) \{'zero', 'constant', 'mirror', 'cyclic'\}
- **SampledFunction** (output control) ........ function \(1\text{d}\)  \(\sim\) HFunction1dX / VARIANT ( real )
  Sampled function.

Parallelization Information

SampleFunct1D is reentrant and processed without parallelization.

Possible Predecessors

TransformFunct1D, CreateFunct1DArray, CreateFunct1DPairs

Tools

Parameter

- **Function** (input control) ................. function \(1\text{d}\)  \(\sim\) HFunction1dX / VARIANT ( real, integer )
  Input function.
- **Mult** (input control) ....................... number  \(\sim\) double / VARIANT
  Factor for scaling of the y values.
  Default Value: 2
  Suggested values: Mult \(\in\) \{0.1, 0.3, 0.5, 1, 2, 5, 10\}
Add (input control) ..........................number \sim double / VARIANT
Constant which is added to the y values.
Default Value : 0
Suggested values : Add \in \{-10, -5, 1, 0, 5, 10\}

FunctionScaled (output control) .......... functionId \sim HFunction1dX / VARIANT (real, integer)
Transformed function.

Parallelization Information
ScaleYFunct1D is reentrant and processed without parallelization.

Possible Predecessors
CreateFunct1DPairs, CreateFunct1DArray

Possible Successors
MatchFunct1DTrans, DistanceFunct1D

Tools

\[
\begin{align*}
\text{[out]} & \quad \text{HFunction1dX SmoothedFunction} & \quad \text{HFunction1dX.SmoothFunct1DGauss} \\
& \quad ([\text{in}] \quad \text{double Sigma} )
\end{align*}
\]


Smooth an equidistant function with a Gaussian function.
The operator SmoothFunct1DGauss smooths a one-dimensional function with a Gaussian function.

Parameter

Function (input control) ......................... functionId \sim HFunction1dX / VARIANT (real, integer)
Function to be smoothed.

Sigma (input control) .......................... number \sim double / VARIANT
Sigma of the Gaussian function for the smoothing.
Default Value : 2.0
Suggested values : Sigma \in \{0.5, 1.0, 2.0, 3.0, 4.0, 5.0\}
Typical range of values : 0.1 \leq Sigma \leq 0.1(lin)
Minimum Increment : 0.01
Recommended Increment : 0.2

SmoothedFunction (output control) .......... functionId \sim HFunction1dX / VARIANT (real )
Smoothed function.

Parallelization Information
SmoothFunct1DGauss is reentrant and processed without parallelization.

Possible Predecessors
CreateFunct1DPairs, CreateFunct1DArray

Possible Successors
MatchFunct1DTrans, DistanceFunct1D

Tools

\[
\begin{align*}
\text{[out]} & \quad \text{HFunction1dX SmoothedFunction} & \quad \text{HFunction1dX.SmoothFunct1DMean} \\
& \quad ([\text{in}] \quad \text{long SmoothSize, [in] long Iterations} )
\end{align*}
\]


Smooth a 1D function by averaging its values.
The operator SmoothFunct1DMean smooths a one dimensional function by applying an average (mean) filter multiple times.
13.6. **FUNCTION**

- **Function** (input control) \( \ldots \) \( \text{function}_1 \ldots \text{function}_n \) \( \sim HFunction1dX / \text{VARIANT} (\text{real}, \text{integer}) \)
  - 1D function.

- **SmoothSize** (input control) \( \ldots \) \( \text{integer} \) \( \sim \text{long} / \text{VARIANT} \)
  - Size of the averaging mask.
  - **Default Value**: 10
  - **Suggested values**: \( \text{SmoothSize} \in \{1, 3, 5, 7, 9, 11, 13, 15, 21, 31, 51\} \)
  - **Typical range of values**: \( 1 \leq \text{SmoothSize} \leq 1(\text{lin}) \)
  - **Minimum Increment**: 1
  - **Recommended Increment**: 1
  - **Restriction**: \((\text{SmoothSize} > 0)\)

- **Iterations** (input control) \( \ldots \) \( \text{integer} \) \( \sim \text{long} / \text{VARIANT} \)
  - Number of iterations for the smoothing.
  - **Default Value**: 3
  - **Suggested values**: \( \text{Iterations} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \)
  - **Typical range of values**: \( 1 \leq \text{Iterations} \leq 1(\text{lin}) \)
  - **Minimum Increment**: 1
  - **Recommended Increment**: 1
  - **Restriction**: \((\text{Iterations} \geq 1)\)

- **SmoothedFunction** (output control) \( \ldots \) \( \text{function}_1 \ldots \text{function}_n \) \( \sim HFunction1dX / \text{VARIANT} (\text{real}) \)
  - Smoothed function.

---

**Parallelization Information**

- **SmoothFunct1DMean** is reentrant and processed without parallelization.

---

**Possible Predecessors**

- CreateFunct1DArray

---

**Alternatives**

- SmoothFunct1DGauss

---

**Module**

**Tools**

```c
[out] HFunction1dX TransformedFunction HFunction1dX.TransformFunct1D
([in] VARIANT Params )

void HOperatorSetX.TransformFunct1D ([in] VARIANT Function,
[in] VARIANT Params, [out] VARIANT TransformedFunction )
```

Transform a function using given transformation parameters.

**TransformFunct1D** transforms the input function **Function** using the transformation parameters given in **Params**. The function **Function** is passed as a tuple (see CreateFunct1DArray and CreateFunct1DPairs). The following model is used for the transformation between the two functions (see MatchFunct1DTrans):

\[
y(t) = a_1 y(a_3 x + a_4) + a_5.
\]

The output function **TransformedFunction** is obtained by transforming the x and y values of the input function separately with the above formula, i.e., the output function is not sampled again. Therefore, the parameter \( a_3 \) is restricted to \( a_3 \neq 0.0 \). To resample a function, the operator **SampleFunct1D** can be used.

---

**Parameter**

- **Function** (input control) \( \ldots \) \( \text{function}_1 \ldots \text{function}_n \) \( \sim HFunction1dX / \text{VARIANT} (\text{real}, \text{integer}) \)
  - Input function.

- **Params** (input control) \( \ldots \) \( \text{number} \) \( \sim \text{VARIANT} (\text{real}) \)
  - Transformation parameters between the functions.
  - **Number of elements**: 4

- **TransformedFunction** (output control) \( \ldots \) \( \text{function}_1 \ldots \text{function}_n \) \( \sim HFunction1dX / \text{VARIANT} (\text{real}, \text{integer}) \)
  - Transformed function.
### Parallelization Information

*TransformFunct1D* is *reentrant* and processed *without* parallelization.

### Possible Predecessors

*CreateFunct1DPairs, CreateFunct1DArray, MatchFunct1DTrans*

---

**Tools**

```c
void HFunction1dX.WriteFunct1D ([in] String FileName )

void HOperatorSetX.WriteFunct1D ([in] VARIANT Function,
[in] VARIANT FileName )
```

*Write a function to a file.*

The operator *WriteFunct1D* writes the contents of *Function* to a file. The data is written in an ASCII format. Therefore, the file can be exchanged between different architectures. The data can be read by the operator *ReadFunct1D*. There is no specific extension for this kind of file.

<table>
<thead>
<tr>
<th>Parameter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Function</em> (input control) .................. function_1d $\sim$ HFunction1dX / VARIANT ( real, integer )</td>
<td>Function to be written.</td>
</tr>
<tr>
<td><em>FileName</em> (input control) .................. filename $\sim$ String / VARIANT</td>
<td>Name of the file to be written.</td>
</tr>
</tbody>
</table>

*Result*

If the parameters are correct the operator *WriteFunct1D* returns the value TRUE. If the file could not be opened *WriteFunct1D* returns FAIL. Otherwise an exception handling is raised.

### Parallelization Information

*WriteFunct1D* is *reentrant* and processed *without* parallelization.

### Possible Predecessors

*CreateFunct1DPairs, CreateFunct1DArray*

**See also**

*ReadFunct1D, WriteImage, WriteRegion, OpenFile*

### Alternatives

*WriteTuple, FwriteString*

---

**Basic operators**

```c
[out] double XMin HFunction1dX.XRRangeFunct1D ([out] double XMax )

void HOperatorSetX.XRRangeFunct1D ([in] VARIANT Function,
[out] VARIANT XMin, [out] VARIANT XMax )
```

*Smallest and largest x value of the function.*

*XRangeFunct1D* calculates the smallest and the largest x value of *Function*.

<table>
<thead>
<tr>
<th>Parameter</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Function</em> (input control) .................. function_1d $\sim$ HFunction1dX / VARIANT ( real, integer )</td>
<td>Input function.</td>
</tr>
<tr>
<td><em>XMin</em> (output control) ..................... number $\sim$ double / VARIANT</td>
<td>Smallest x value.</td>
</tr>
<tr>
<td><em>XMax</em> (output control) ..................... number $\sim$ double / VARIANT</td>
<td>Largest x value.</td>
</tr>
</tbody>
</table>

### Parallelization Information

*XRangeFunct1D* is *reentrant* and processed *without* parallelization.

---

HALCON/COM Reference Manual, 2005-2-1
13.7 Geometry

**Possible Predecessors**
CreateFunct1DPairs, CreateFunct1DArray

**Module**

```c
[out] double YMin HFunction1DX.YRangeFunct1D ([out] double YMax )
void HOperatorSetX.YRangeFunct1D ([in] VARIANT Function,
[out] VARIANT YMin, [out] VARIANT YMax )
```

Adjust smallest and largest y value of the function.

**YRangeFunct1D** calculates the smallest and the largest y value of `Function`.

**Parameter**

- **Function** (input control) ................. function_1d ~ HFunction1DX / VARIANT ( real, integer )
  Input function.
- **YMin** (output control) .............................. number ~ double / VARIANT
  Smallest y value.
- **YMax** (output control) .............................. number ~ double / VARIANT
  Largest y value.

**Parallelization Information**

**YRangeFunct1D** is reentrant and processed without parallelization.

**Possible Predecessors**
CreateFunct1DPairs, CreateFunct1DArray

**Module**

13.7 Geometry

```c
[out] VARIANT Angle HMiscX.AngleLl ([in] VARIANT RowA1,
[in] VARIANT ColumnB2 )
void HOperatorSetX.AngleLl ([in] VARIANT RowA1, [in] VARIANT ColumnA1,
[out] VARIANT Angle )
```

Calculate the angle between two lines.

The operator **AngleLl** calculates the angle between two lines. As input the rows and columns of the first line (RowA1, ColumnA1, RowA2, ColumnA2) and of the second line (RowB1, ColumnB1, RowB2, ColumnB2) are expected. The calculation in done as follows: We interpret the lines as vectors with starting points RowA1, ColumnA1 and RowB1, ColumnB1 and end points RowA2, ColumnA2 and RowB2, ColumnB2, respectively. Turning the vector A counterclockwise onto the vector B (the center of rotation is the intersection point of the two lines) yields the angle. The result depends on the order of the points and on the order of the lines. The parameter **Angle** returns the angle in radians. The angles range from \(-\pi \leq \text{Angle} \leq \pi\).

**Parameter**

- **RowA1** (input control) ........................ point.y(-array) ~ VARIANT ( real, integer )
  Row of the first point of the first line.
- **ColumnA1** (input control) ........................ point.x(-array) ~ VARIANT ( real, integer )
  Column of the first point of the first line.
- **RowA2** (input control) ........................ point.y(-array) ~ VARIANT ( real, integer )
  Row of the second point of the first line.
CHAPTER 13. TOOLS

\[ ColumnA2 \] (input control) .......................... point.x(-array)  \to  \text{VARIANT} (\text{real}, \text{integer})
\quad Column of the second point of the first line.

\[ RowB1 \] (input control) .......................... point.y(-array)  \to  \text{VARIANT} (\text{real}, \text{integer})
\quad Row of the first point of the second line.

\[ ColumnB1 \] (input control) .......................... point.x(-array)  \to  \text{VARIANT} (\text{real}, \text{integer})
\quad Column of the first point of the second line.

\[ RowB2 \] (input control) .......................... point.y(-array)  \to  \text{VARIANT} (\text{real}, \text{integer})
\quad Row of the second point of the second line.

\[ ColumnB2 \] (input control) .......................... point.x(-array)  \to  \text{VARIANT} (\text{real}, \text{integer})
\quad Column of the second point of the second line.

\[ Angle \] (output control) .......................... number(-array)  \to  \text{VARIANT}
\quad Angle between the lines.

--- Example ---

RowA1 := 255  
ColumnA1 := 10  
RowA2 := 255  
ColumnA2 := 501  
disp_line (WindowHandle, RowA1, ColumnA1, RowA2, ColumnA2)
RowB1 := 255  
ColumnB1 := 255  
for i := 1 to 360 by 1
  RowB2 := 255 + \sin(\text{rad}(i)) \times 200  
  ColumnB2 := 255 + \cos(\text{rad}(i)) \times 200  
  disp_line (WindowHandle, RowB1, ColumnB1, RowB2, ColumnB2)
  angle_ll (RowA1, ColumnA1, RowA2, ColumnA2,  
            RowB1, ColumnB1, RowB2, ColumnB2, Angle)
endfor

--- Result ---
\[ AngleLl \] returns TRUE.

--- Parallelization Information ---
\[ AngleLl \] is reentrant and processed without parallelization.

--- Alternatives ---
\[ AngleLx \]

--- Module ---

Basic operators

\[
\begin{align*}
\text{[out]} \text{VARIANT Angle } & \text{HMiscX.AngleLx} ([\text{in}] \text{VARIANT Row1,} \\
& \text{[in]} \text{VARIANT Column1, [in]} \text{VARIANT Row2, [in]} \text{VARIANT Column2}) \\
\text{void } & \text{HOperatorSetX.AngleLx} ([\text{in}] \text{VARIANT Row1, [in]} \text{VARIANT Column1,} \\
& \text{[in]} \text{VARIANT Row2, [in]} \text{VARIANT Column2, [out]} \text{VARIANT Angle} )
\end{align*}
\]

Calculate the angle between one line and the vertical axis.

The operator \text{AngleLx} calculates the angle between one line and the abscissa. As input the row and column of the line \text{(Row1,Column1,Row2,Column2)} are expected. The calculation is done as follows: We interpret the line as a vector with starting point \text{Row1,Column1} and end point \text{Row2,Column2}. Turning the vector counter clockwise onto the abscissa (center of rotation is the intersection point of the abscissa) yields the angle. The result is dependant on the order of points of line. The parameters \text{Angle} returns the angle in radians. The angles range from $-\pi \leq \text{Angle} \leq \pi$. 

HALCON/COM Reference Manual, 2005-2-1
13.7. GEOMETRY

Parameter

- **Row1** (input control) 
  Row of the first point of the line.
- **Column1** (input control) 
  Column of the first point of the line.
- **Row2** (input control) 
  Row of the second point of the line.
- **Column2** (input control) 
  Column of the second point of the line.
- **Angle** (output control) 
  Angle between the line and the abscissa.

Example

RowX1 := 255
ColumnX1 := 10
RowX2 := 255
ColumnX2 := 501

disp_line (WindowHandle, RowX1, ColumnX1, RowX2, ColumnX2)
Row1 := 255
Column1 := 255
for i := 1 to 360 by 1
  Row2 := 255 + sin(rad(i)) * 200
  Column2 := 255 + cos(rad(i)) * 200
  disp_line (WindowHandle, Row1, Column1, Row2, Column2)
  angle_lx (Row1, Column1, Row2, Column2, Angle)
endfor

Result

AngleLx returns TRUE.

Parallelization Information

AngleLx is reentrant and processed without parallelization.

Alternatives

AngleL1

Module

Basic operators

```plaintext
[out] VARIANT DistanceMin
HRegionX.DistanceLr ([in] VARIANT Row1,
[in] VARIANT Column1, [in] VARIANT Row2, [in] VARIANT Column2,
[out] VARIANT DistanceMax )

void HOperatorSetX.DistanceLr ([in] IHObjectX Region,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2, [out] VARIANT DistanceMin, [out] VARIANT DistanceMax )
```

Calculate the distance between one line and one region.

The operator DistanceLr calculates the orthogonal distance between one line and one region. As input the coordinates of 2 points that the line represent (Row1,Column1, Row2,Column2) and one region are expected. The parameters DistanceMin and DistanceMax return the result of the calculation.

Attention

Due to efficiency of DistanceLr holes are ignored. Furthermore, if the lines intersects the region a minimal distance larger than 0.5 can be returned.

HALCON 6.1.4
CHAPTER 13. TOOLS

Parameter

- **Region** (input iconic) .................. region \(\sim HRegionX / IHObjectX\)
  Input region.

- **Row1** (input control) ................ point.y(-array) \(\sim\) VARIANT (real, integer)
  Row of the first point of the line.

- **Column1** (input control) ............. point.x(-array) \(\sim\) VARIANT (real, integer)
  Column of the first point of the line.

- **Row2** (input control) ................ point.y(-array) \(\sim\) VARIANT (real, integer)
  Row of the second point of the line.

- **Column2** (input control) .......... point.x(-array) \(\sim\) VARIANT (real, integer)
  Column of the second point of the line.

- **DistanceMin** (output control) ........ number(-array) \(\sim\) VARIANT
  Minimal distance between the line and the region

- **DistanceMax** (output control) ........ number(-array) \(\sim\) VARIANT
  Maximal distance between the line and the region

Example

```
dev_close_window ()
read_image (Image, 'fabrik')
develop_window (0, 0, 512, 512, 'white', WindowHandle)
threshold (Image, Region, 180, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and',
            5000, 100000000)
develop_clear_window ()
develop_set_color ('black')
develop_display (SelectedRegions)
develop_set_color ('red')
Row1 := 100
Row2 := 400
for Col := 50 to 400 by 4
   disp_line (WindowHandle, Row1, Col+100, Row2, Col)
distance_lr (SelectedRegions, Row1, Col+100, Row2, Col, DistanceMin, DistanceMax)
endfor
```

Result

DistanceLr returns TRUE.

Parallelization Information

DistanceLr is reentrant and processed without parallelization.

See also

HammingDistance, SelectRegionPoint, TestRegionPoint, SmallestRectangle2

Alternatives

DistancePr, DistanceSr, DiameterRegion

Module

Basic operators
Calculate the distance between one point and one line.

The operator `DistancePl` calculates the orthogonal distance between a point (`Row`, `Column`) and a line, given by two arbitrary points of the line. The result is passed in `Distance`.

Parameter

- **Row** (input control) ........................................ point.y(-array) ~> VARIANT( real, integer )  
  Row of the point.

- **Column** (input control) ..................................... point.x(-array) ~> VARIANT( real, integer )  
  Column of the point.

- **Row1** (input control) ....................................... point.y(-array) ~> VARIANT( real, integer )  
  Row of the first point of the line.

- **Column1** (input control) .................................... point.x(-array) ~> VARIANT( real, integer )  
  Column of the first point of the line.

- **Row2** (input control) ....................................... point.y(-array) ~> VARIANT( real, integer )  
  Row of the second point of the line.

- **Column2** (input control) .................................... point.x(-array) ~> VARIANT( real, integer )  
  Column of the second point of the line.

- **Distance** (output control) ................................. number(-array) ~> VARIANT  
  Distance between the points

Example

```
read_image(Image, 'mreut')
dev_open_window(0, 0, 512, 512, 'white', WindowHandle)
dev_display(Image)
dev_set_color('black')
threshold(Image, Region, 180, 255)
dev_clear_window()
dev_display(Region)
connection(Region, ConnectedRegions)
select_shape(ConnectedRegions, SelectedRegions, 'area', 'and', 10000, 100000000)
get_region_contour(SelectedRegions, Rows, Columns)
RowLine1 := 5
Colline1 := 300
RowLine2 := 300
Colline2 := 400
NumberTuple := |Rows|
dev_set_color('red')
disp_line(WindowHandle, RowLine1, Colline1, RowLine2, Colline2)
dev_set_color('green')
for i := 1 to NumberTuple by 5
  disp_line(WindowHandle, Rows[i], Columns[i]-2, Rows[i], Columns[i]+2)
  disp_line(WindowHandle, Rows[i]-2, Columns[i], Rows[i]+2, Columns[i])
  distance_pl(Rows[i], Columns[i], RowLine1, Colline1, RowLine2, Colline2, Distance)
endfor
```

Result

`DistancePl` returns TRUE.
Parallelization Information

DistancePl is reentrant and processed without parallelization.

See also DistancePp, DistancePr

Alternatives DistancePs

Module Basic operators

Calculate the distance between two points.

The operator DistancePp calculates the distance between pairs of points according to the following formula:

\[
\text{Distance} = \sqrt{(\text{Row1} - \text{Row2})^2 + (\text{Column1} - \text{Column2})^2}
\]

The result is passed in Distance.

Parameter

- **Row1** (input control) point.y(-array) \(\sim\) VARIANT (real, integer) Row of the first point.
- **Column1** (input control) point.x(-array) \(\sim\) VARIANT (real, integer) Column of the first point.
- **Row2** (input control) point.y(-array) \(\sim\) VARIANT (real, integer) Row of the second point.
- **Column2** (input control) point.x(-array) \(\sim\) VARIANT (real, integer) Column of the second point.
- **Distance** (output control) number(-array) \(\sim\) VARIANT Distance between the points

Example

```c
dev_close_window ()
read_image (Image, 'mreut')
devo_pen_open (0, 0, 512, 512, 'white', WindowHandle)
devo_display (Image)
devo_set_color ('black')
threshold (Image, Region, 180, 255)
devo_clear_window ()
devo_display (Region)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 10000, 100000000)
get_region_contour (SelectedRegions, Rows, Columns)
RowPoint := 80
ColPoint := 250
NumberTuple := |Rows|
devo_set_color ('red')
set_draw (WindowHandle, 'margin')
disp_circle (WindowHandle, RowPoint, ColPoint, 10)
devo_set_color ('green')
for i := 1 to NumberTuple by 10
    disp_line (WindowHandle, Rows[i], Columns[i]-2, Rows[i], Columns[i]+2)
```
13.7. GEOMETRY

```c
disp_line (WindowHandle, Rows[i]-2, Columns[i], Rows[i]+2, Columns[i])
distance_pp (RowPoint, ColPoint, Rows[i], Columns[i], Distance)
endfor
```

---

**Result**

`DistancePp` returns TRUE.

---

**Parallelization Information**

`DistancePp` is reentrant and processed without parallelization.

---

**See also**

`DistancePl`, `DistancePr`

---

**Alternatives**

`DistancePs`

---

**Module**

Basic operators

---

**Out** VARIANT `DistanceMin`

HRegionX.DistancePr ([in] VARIANT Row, [in] VARIANT Column, [out] VARIANT DistanceMax)

**void** HOperatorSetX.DistancePr ([in] IHObjectX Region, [in] VARIANT Row, [in] VARIANT Column, [out] VARIANT DistanceMin, [out] VARIANT DistanceMax)

---

**Calculate the distance between one point and one region.**

The operator `DistancePr` calculates the distance between one point and one region. As input the column und Row of the point (`Row, Column`) and one region are expected. If the pint is inside the region the minimal distance is zero. The parameters `DistanceMin` and `DistanceMax` return the result of the calculation.

---

**Parameter**

- **Region** (input iconic) . . . . . . . . . . . . . . . . . . . . . . . . . . . . region ∼ HRegionX / IHObjectX
  Input region.
- **Row** (input control) . . . . . . . . . . . . . . . . . . . . . point.y(-array) ∼ VARIANT (real, integer)
  Row of the point.
- **Column** (input control) . . . . . . . . . . . . . . . . . . . . point.x(-array) ∼ VARIANT (real, integer)
  Column of the point.
- **DistanceMin** (output control) . . . . . . . . . . . . . . . number(-array) ∼ VARIANT
  Minimal distance between the point and the region
- **DistanceMax** (output control) . . . . . . . . . . . . . . . number(-array) ∼ VARIANT
  Maximal distance between the point and the region

---

**Example**

```c
dev_close_window ()
read_image (Image, ‘mreut’)
dev_open_window (0, 0, 512, 512, ‘white’, WindowHandle)
dev_set_color (‘black’)
threshold (Image, Region, 180, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, ‘area’, ‘and’,
           10000, 100000000)
Row1 := 255
Column1 := 255
dev_clear_window ()
dev_display (SelectedRegions)
dev_set_color (‘red’)
for i := 1 to 360 by 1
  Row2 := 255 + sin(rad(i)) * 200
```

HALCON 6.1.4
Column2 := 255 + cos(rad(i)) * 200
disp_line (WindowHandle, Row1, Column1, Row2, Column2)
distance_pr (SelectedRegions, Row2, Column2,
DistanceMin, DistanceMax)
endfor

<table>
<thead>
<tr>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistancePr returns <strong>TRUE</strong>.</td>
</tr>
</tbody>
</table>

**Parallelization Information**

DistancePr is **reentrant** and processed **without** parallelization.

**See also**

HammingDistance, SelectRegionPoint, TestRegionPoint, SmallestRectangle2

**Alternatives**

DistanceLr, DistanceSr, DiameterRegion

**Module**

Basic operators

```
[out] VARIANT DistanceMin HMiscX.DistancePs ([in] VARIANT Row,
[in] VARIANT Column, [in] VARIANT Row1, [in] VARIANT Column1,
[in] VARIANT Row2, [in] VARIANT Column2, [out] VARIANT DistanceMax )

void HOperatorSetX.DistancePs ([in] VARIANT Row, [in] VARIANT Column,
[in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2,
[in] VARIANT Column2, [out] VARIANT DistanceMin, [out] VARIANT DistanceMax )
```

Calculate the distances between a point and a line segment.

The operator **DistancePs** calculates the minimal and maximal distance between a point (Row, Column) and a line segment which is represented by the start point (Row1, Column1) and the end point (Row2, Column2). **DistanceMax** is the maximal distance between the point and the end points of the line segment. **DistanceMin** is identical to **DistancePl** in the case that the point is “between” the two endpoints. Otherwise the minimal distance to one of the endpoints is used.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
</table>

- **Row** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the first point.
- **Column** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the first point.
- **Row1** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the first point of the line segment.
- **Column1** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the first point of the line segment.
- **Row2** (input control) point.y(-array) ~ VARIANT (real, integer)
  Row of the second point of the line segment.
- **Column2** (input control) point.x(-array) ~ VARIANT (real, integer)
  Column of the second point of the line segment.
- **DistanceMin** (output control) number(-array) ~ VARIANT
  Minimal distance between the point and the line segment.
- **DistanceMax** (output control) number(-array) ~ VARIANT
  Maximal distance between the point and the line segment.

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
</table>

```
read_image (Image, 'mreut')
dev_open_window (0, 0, 512, 512, 'white', WindowHandle)
dev_display (Image)
```
13.7. GEOMETRY

dev_set_color (‘black’)
threshold (Image, Region, 180, 255)
dev_clear_window ()
dev_display (Region)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, ‘area’, ‘and’,
10000, 100000000)
get_region_contour (SelectedRegions, Rows, Columns)
RowLine1 := 400
ColLine1 := 50
RowLine2 := 50
ColLine2 := 450
NumberTuple := |Rows|
dev_set_color (‘red’)
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
dev_set_color (‘green’)
for i := 1 to NumberTuple by 10
    disp_line (WindowHandle, Rows[i], Columns[i]-2, Rows[i], Columns[i]+2)
    disp_line (WindowHandle, Rows[i]-2, Columns[i], Rows[i] +2, Columns[i])
    distance_ps (Rows[i], Columns[i], RowLine1, ColLine1, RowLine2, ColLine2,
        DistanceMin, DistanceMax)
endfor

Result
DistancePs returns TRUE.

Parallelization Information
DistancePs is reentrant and processed without parallelization.

See also
DistancePp, DistancePr

Alternatives
DistancePl

Module
Basic operators

[out] VARIANT MinDistance HRegionX.DistanceRrMin
([in] HRegionX Regions2, [out] VARIANT Row1, [out] VARIANT Column1,
[out] VARIANT Row2, [out] VARIANT Column2 )
void HOperatorSetX.DistanceRrMin ([in] IObjectX Regions1,
[in] IObjectX Regions2, [out] VARIANT MinDistance, [out] VARIANT Row1,
[out] VARIANT Column1, [out] VARIANT Row2, [out] VARIANT Column2 )

Minimum distance between the contour pixels of two regions each.
The operator DistanceRrMin calculates the minimum distance of pairs of regions. If several regions are
passed in Regions1 and Regions2 the distance between the contour pixels of each i-th element is calculated
and then forms the i-th entry in the output parameter MinDistance. The calculation is carried out by comparing
all contour pixels. The Euclidean distance is used. The parameters (Row1,Column1) and (Row2, Column2)
indicate the position on the contour of Regions1 and Regions2, respectively, the distance between which is
the minimum distance.

Attention
Each region must consist of exactly one connection component. Both input parameters must contain the same
number of regions. The regions must not be empty. If the regions overlap the distance is indicated as 0.0. In this
case the positions are not reliable.
CHAPTER 13. TOOLS

Parameter

- **Regions1** (input iconic) .............................................. region(-array) ~ HRegionX / IObjectX
  Regions to be examined.
- **Regions2** (input iconic) .............................................. region(-array) ~ HRegionX / IObjectX
  Regions to be examined.
- **MinDistance** (output control) ................................... real(-array) ~ VARIANT (real)
  Minimum distance between contours of the regions.
  **Restriction:** \(0 \leq \text{MinDistance}\)
- **Row1** (output control) .............................................. point.y(-array) ~ VARIANT (integer)
  Line index on contour in **Regions1**.
- **Column1** (output control) ........................................... point.x(-array) ~ VARIANT (integer)
  Column index on contour in **Regions1**.
- **Row2** (output control) .............................................. point.y(-array) ~ VARIANT (integer)
  Line index on contour in **Regions2**.
- **Column2** (output control) ........................................... point.x(-array) ~ VARIANT (integer)
  Column index on contour in **Regions2**.

Complexity

If \(N_1, N_2\) are the lengths of the contours the runtime complexity is \(O(N_1 \times N_2)\).

Result

The operator **DistanceRrMin** returns the value TRUE if the input is not empty. Otherwise an exception handling is raised.

Parallelization Information

**DistanceRrMin** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Threshold, RegionGrowing, Connection

Alternatives

**DistanceRrMinDil**, **Dilation1**, **Intersection**

Region processing

Minimum distance between two regions with the help of dilatation.

The operator **DistanceRrMinDil** calculates the minimum distance between pairs of regions. If several regions are passed in **Regions1** and **Regions2** the distance between the \(i\)-th elements in each case is calculated. It then forms the \(i\)-th entry in the output parameter **MinDistance**. The calculation is carried out with the help of dilatation with the Golay element 'h'. The result is:

\[
\text{Numberiterations} \times 2 - 1
\]

The mask 'h' has the effect that precisely the maximum metrics are calculated.

Attention

Both parameters must contain the same number of regions. The regions must not be empty.

Parameter

- **Regions1** (input iconic) .............................................. region(-array) ~ HRegionX / IObjectX
  Regions to be examined.
- **Regions2** (input iconic) .............................................. region(-array) ~ HRegionX / IObjectX
  Regions to be examined.
13.7. GEOMETRY

MinDistance (output control) ........................................ integer(-array) ^ VARIANT ( integer )
MinDistance : (-1 <= MinDistance)

The operator DistanceRrMinDil returns the value TRUE if the input is not empty. Otherwise an exception handling is raised.

Parallelization Information

DistanceRrMinDil is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

Regiongrowing, Connection

Alternatives

DistanceRrMin, Dilation1, Intersection

Module

Region processing

Calculate the distances between one line segment and one line.

The operator DistanceSl calculates the minimal and maximal orthogonal distance between one line segment and one line. As input the columns and rows of the line segment (RowA1, ColumnA1, RowA2, ColumnA2) and of the line (RowB1, ColumnB1, RowB2, ColumnB2) are expected. The parameters DistanceMin and DistanceMax return the result of the calculation. If the line segments are intersecting DistanceMin returns zero.

Parameter

RowA1 (input control) ........................................ point.y(-array) ^ VARIANT ( real, integer )
Row of the first point of the line segment.

ColumnA1 (input control) ........................................ point.x(-array) ^ VARIANT ( real, integer )
Column of the first point of the line segment.

RowA2 (input control) ........................................ point.y(-array) ^ VARIANT ( real, integer )
Row of the second point of the line segment.

ColumnA2 (input control) ........................................ point.x(-array) ^ VARIANT ( real, integer )
Column of the second point of the line segment.

RowB1 (input control) ........................................ point.y(-array) ^ VARIANT ( real, integer )
Row of the first point of the line.

ColumnB1 (input control) ........................................ point.x(-array) ^ VARIANT ( real, integer )
Column of the first point of the line.

RowB2 (input control) ........................................ point.y(-array) ^ VARIANT ( real, integer )
Row of the second point of the line.

ColumnB2 (input control) ........................................ point.x(-array) ^ VARIANT ( real, integer )
Column of the second point of the line.

DistanceMin (output control) ........................................ number(-array) ^ VARIANT
Minimal distance between the line segment and the line

DistanceMax (output control) ........................................ number(-array) ^ VARIANT
Maximal distance between the line segment and the line

HALCON 6.1.4
### Example

```plaintext
dev_set_color ('black')
RowLine1 := 400
ColLine1 := 200
RowLine2 := 200
ColLine2 := 400
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
dev_set_color ('green')
```

```plaintext
n := 0
for Rows := 40 to 200 by 4
    disp_line (WindowHandle, Rows+n, Columns+n, Rows, Columns+n)
    distance_sl (Rows+n, Columns+n, Rows, Columns+n, RowLine1, ColLine1, RowLine2, ColLine2, DistanceMin, DistanceMax)
    n := n+10
endfor
```

---

### Result

**DistanceSl** returns TRUE.

---

### Parallelization Information

**DistanceSl** is reentrant and processed without parallelization.

---

**See also**

DistancePs, DistancePp

---

### Alternatives

DistancePl

---

### Module

Basic operators

---

```plaintext
[out] VARIANT DistanceMin HRegionX.DistanceSr ([in] VARIANT Row1, [in] VARIANT Column1, [in] VARIANT Row2, [in] VARIANT Column2, [out] VARIANT DistanceMax )
```

### Calculate the distance between one line segment and one region.

The operator **DistanceSr** calculates the distance between one line segment and one region. `Row1, Column1, Row2, Column2` are the initial and end coordinates of the line segment. The parameters `DistanceMin` and `DistanceMax` contain the resulting distances.

---

### Attention

Due to efficiency of **DistanceSr** holes are ignored. Furthermore, if the lines intersects the region a minimal distance larger than 0.5 can be returned.

---

### Parameter

- **Region** (input iconic) ............................................. region ~ HRegionX / IObjectX Input region.
- **Row1** (input control) ........................................... point.y(-array) ~> VARIANT ( real, integer ) Row of the first point of the line segment.
- **Column1** (input control) ................................. point.x(-array) ~> VARIANT ( real, integer ) Column of the first point of the line segment.
- **Row2** (input control) ........................................... point.y(-array) ~> VARIANT ( real, integer ) Row of the second point of the line segment.
13.7. GEOMETRY

- **Column2**: (input control) \( \text{point.x}(-\text{array}) \) \( \rightarrow \) VARIANT (real, integer)
  Column of the second point of the line segment.

- **DistanceMin**: (output control) \( \text{number}(-\text{array}) \) \( \rightarrow \) VARIANT
  Minimal distance between the line segment and the region

- **DistanceMax**: (output control) \( \text{number}(-\text{array}) \) \( \rightarrow \) VARIANT
  Maximal distance between the line segment and the region

---

Example

```halcon
read_image (Image, ‘fabrik’)
dev_open_window (0, 0, 512, 512, ‘white’, WindowHandle)
dev_display (Image)
threshold (Image, Region, 180, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, ‘area’, ‘and’, 5000, 100000000)
dev_clear_window ()
dev_set_color (‘black’)
dev_display (SelectedRegions)
dev_set_color (‘red’)
Row1 := 100
Row2 := 400
n := 0
for Col := 50 to 400 by 5
  disp_line (WindowHandle, Row1+n, Col, Row2-n, Col+100)
distance_sr (SelectedRegions, Row1+n, Col, Row2-n, Col+100, DistanceMin, DistanceMax)
n := n+5
endfor
```

Result

DistanceSr returns TRUE.

Parallelization Information

DistanceSr is reentrant and processed without parallelization.

See also

HammingDistance, SelectRegionPoint, TestRegionPoint, SmallestRectangle2

Alternatives

DistanceLr, DistancePr, DiameterRegion

Module

Basic operators

```
[out] VARIANT DistanceMin HMiscX.DistanceSs ([in] VARIANT RowA1,
[in] VARIANT ColumnB2, [out] VARIANT DistanceMax )
void HOperatorSetX.DistanceSs ([in] VARIANT RowA1,
[in] VARIANT ColumnB2, [out] VARIANT DistanceMin, [out] VARIANT DistanceMax )
```

Calculate the distances between two line segments.

The operator DistanceSs calculates the minimal and maximal distance between two line segments. As input the rows and columns of the first line segments (RowA1,ColumnA1,RowA2,ColumnA2) and of the second line segment (RowB1,ColumnB1,RowB2,ColumnB2) are used. The parameters DistanceMin and DistanceMax return the result of the calculation. If the line segments are intersecting DistanceMin returns zero.
Parameter

- **RowA1** (input control) \( \ldots \) point.y(-array) \( \sim \) VARIANT (real, integer)
  Row of the first point of the line segment.
- **ColumnA1** (input control) \( \ldots \) point.x(-array) \( \sim \) VARIANT (real, integer)
  Column of the first point of the line segment.
- **RowA2** (input control) \( \ldots \) point.y(-array) \( \sim \) VARIANT (real, integer)
  Row of the second point of the line segment.
- **ColumnA2** (input control) \( \ldots \) point.x(-array) \( \sim \) VARIANT (real, integer)
  Column of the second point of the line segment.
- **RowB1** (input control) \( \ldots \) point.y(-array) \( \sim \) VARIANT (real, integer)
  Row of the first point of the line.
- **ColumnB1** (input control) \( \ldots \) point.x(-array) \( \sim \) VARIANT (real, integer)
  Column of the first point of the line.
- **RowB2** (input control) \( \ldots \) point.y(-array) \( \sim \) VARIANT (real, integer)
  Row of the second point of the line.
- **ColumnB2** (input control) \( \ldots \) point.x(-array) \( \sim \) VARIANT (real, integer)
  Column of the second point of the line.
- **DistanceMin** (output control) \( \ldots \) number(-array) \( \sim \) VARIANT
  Minimal distance between the line segments
- **DistanceMax** (output control) \( \ldots \) number(-array) \( \sim \) VARIANT
  Maximal distance between the line segments

Example

```plaintext
dev_set_color (‘black’)
RowLine1 := 400
ColLine1 := 200
RowLine2 := 240
ColLine2 := 400
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
dev_set_color (‘red’)
n := 0
for Rows := 40 to 200 by 4
  disp_line (WindowHandle, Rows, Columns, Rows+n, Columns+n)
  distance_ss (Rows, Columns, Rows+n, Columns+n, RowLine1, ColLine1,
  RowLine2, ColLine2, DistanceMin, DistanceMax)
  n := n+8
endfor
```

Result

`DistanceSs` returns TRUE.

Parallelization Information

`DistanceSs` is reentrant and processed without parallelization.

See also

`DistanceP1`, `DistancePs`

Alternatives

`DistancePp`

Module

Basic operators
Points of an ellipse corresponding to specific angles.

GetPointsEllipse returns the points (RowPoint,ColPoint) on the specified ellipse corresponding to the angles in Angle, which refer to the main axis of the ellipse. The ellipse itself is characterized by the center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2.

### Parameter

- **Angle** (input control) 
  Angles corresponding to the resulting points [rad].
  Default Value: 0
  Restriction: \((\text{Angle} \geq 0) \land (\text{Angle} \leq 6.283185307)\)

- **Row** (input control) 
  Row coordinate of the center of the ellipse.

- **Column** (input control) 
  Column coordinate of the center of the ellipse.

- **Phi** (input control) 
  Orientation of the main axis [rad].
  Restriction: \((\text{Phi} \geq 0) \land (\text{Phi} \leq 6.283185307)\)

- **Radius1** (input control) 
  Length of the larger half axis.
  Restriction: \((\text{Radius}1 > 0)\)

- **Radius2** (input control) 
  Length of the smaller half axis.
  Restriction: \((\text{Radius}2 \geq 0)\)

- **RowPoint** (output control) 
  Row coordinates of the points on the ellipse.

- **ColPoint** (output control) 
  Column coordinates of the points on the ellipse.

### Example

draw_ellipse(WindowHandle,Row,Column,Phi,Radius1,Radius2)
get_points_ellipse([[0,3.14]],Row,Column,Phi,Radius1,Radius2,RowPoint,ColPoint)

### Result

GetPointsEllipse returns TRUE if all parameter values are correct. If necessary, an exception is raised.

### Parallelization Information

GetPointsEllipse is reentrant and processed without parallelization.

### Possible Predecessors

FitEllipseContourXld, DrawEllipse, GenEllipseContourXld

### See also

GenEllipseContourXld

### Module

Basic operators
Calculate the intersection point of two lines.

The operator \texttt{IntersectionLI} calculates the intersection point of two lines. As input the columns and rows of the lines (RowA1,ColumnA1,RowA2,ColumnA2) and (RowB1,ColumnB1,RowB2,ColumnB2) are expected. The parameters Row and Column return the result of the calculation. If the lines are parallel IsParallel is 1 else 0. In addition the values of Row and Column are undefined.

\begin{itemize}
  \item \textbf{Attention}\ 
  If the lines are parallel the values of Row and Column are undefined.
\end{itemize}

\textbf{Parameter}

\begin{itemize}
  \item \textbf{RowA1} (input control) \texttt{point.y(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Row of the first point of the first line.
  \item \textbf{ColumnA1} (input control) \texttt{point.x(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Column of the first point of the first line.
  \item \textbf{RowA2} (input control) \texttt{point.y(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Row of the second point of the first line.
  \item \textbf{ColumnA2} (input control) \texttt{point.x(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Column of the second point of the first line.
  \item \textbf{RowB1} (input control) \texttt{point.y(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Row of the first point of the second line.
  \item \textbf{ColumnB1} (input control) \texttt{point.x(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Column of the first point of the second line.
  \item \textbf{RowB2} (input control) \texttt{point.y(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Row of the second point of the second line.
  \item \textbf{ColumnB2} (input control) \texttt{point.x(-array)} \leadsto \texttt{VARIANT( real, integer )}
    Column of the second point of the second line.
  \item \textbf{Row} (output control) \texttt{point.y(-array)} \leadsto \texttt{VARIANT}
    Row of the intersection point.
  \item \textbf{Column} (output control) \texttt{point.x(-array)} \leadsto \texttt{VARIANT}
    Column of the intersection point.
  \item \textbf{IsParallel} (output control) \texttt{number(-array)} \leadsto \texttt{VARIANT}
    Are the two lines parallel?
\end{itemize}

\textbf{Example}

\begin{verbatim}
  dev_set_color ('black')
  RowLine1 := 350
  ColLine1 := 250
  RowLine2 := 300
  ColLine2 := 300
  Rows := 300
  Columns := 50
  disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
  n := 0
  for Rows := 40 to 200 by 4
    dev_set_color ('red')
    disp_line (WindowHandle, Rows, Columns, Rows+n, Columns+n)
    intersection_ll (Rows, Columns, Rows+n, Columns+n, RowLine1, ColLine1,
\end{verbatim}
dev_set_color ('blue')
disp_line (WindowHandle, Row, Column-2, Row, Column+2)
disp_line (WindowHandle, Row-2, Column, Row+2, Column)
n := n+8
endfor

Result
IntersectionLl returns TRUE.

Parallelization Information
IntersectionLl is reentrant and processed without parallelization.

Basic operators

\[
\begin{align*}
\text{[out]} & \quad \text{VARIANT} \quad \text{RowProj} & \quad \text{HMiscX.ProjectionPl} & ( & \text{[in]} & \quad \text{VARIANT} \quad \text{Row}, \\
& \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Column}, & \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Row1}, & \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Column1}, \\
& \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Row2}, & \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Column2}, & \quad \text{[out]} & \quad \text{VARIANT} \quad \text{ColProj} )
\end{align*}
\]

\[
\begin{align*}
\text{void} & \quad \text{HOperatorSetX.ProjectionPl} & ( & \text{[in]} & \quad \text{VARIANT} \quad \text{Row}, & \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Column}, \\
& \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Row1}, & \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Column1}, & \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Row2}, \\
& \quad \text{[in]} & \quad \text{VARIANT} \quad \text{Column2}, & \quad \text{[out]} & \quad \text{VARIANT} \quad \text{RowProj}, & \quad \text{[out]} & \quad \text{VARIANT} \quad \text{ColProj} )
\end{align*}
\]

Calculate the projection of a point onto a line.
The operator ProjectionPl calculates the projection of a point (Row,Column) onto a line which is represented by the start point (Row1,Column1) and the end point (Row2,Column2). RowProj is the row of the projection point and ColProj is the column of the projection point.

Parameter

\begin{itemize}
\item Row (input control) \quad \text{point.y(-array)} \quad \text{~ Variant ( real, integer )}
\item Column (input control) \quad \text{point.x(-array)} \quad \text{~ Variant ( real, integer )}
\item Row1 (input control) \quad \text{point.y(-array)} \quad \text{~ Variant ( real, integer )}
\item Column1 (input control) \quad \text{point.x(-array)} \quad \text{~ Variant ( real, integer )}
\item Row2 (input control) \quad \text{point.y(-array)} \quad \text{~ Variant ( real, integer )}
\item Column2 (input control) \quad \text{point.x(-array)} \quad \text{~ Variant ( real, integer )}
\item RowProj (output control) \quad \text{number(-array)} \quad \text{~ Variant ( real, integer )}
\item ColProj (output control) \quad \text{number(-array)} \quad \text{~ Variant ( real, integer )}
\end{itemize}

Example

dev_set_color ('black')
RowLine1 := 400
ColLine1 := 200
RowLine2 := 240
ColLine2 := 400
Rows := 300
Columns := 50
disp_line (WindowHandle, RowLine1, ColLine1, RowLine2, ColLine2)
n := 0
for Rows := 40 to 200 by 4
  dev_set_color ('red')
  disp_circle (WindowHandle, Rows+n, Columns, 2)
  projection_pl (Rows+n, Columns, RowLine1, ColLine1, RowLine2, ColLine2, RowProj, ColProj)
  dev_set_color ('blue')
  disp_line (WindowHandle, RowProj-2, ColProj, RowProj+2, ColProj)
  disp_line (WindowHandle, RowProj, ColProj-2, RowProj, ColProj+2)
  n := n+8
endfor

--- Result

ProjectionPl returns TRUE.

--- Parallelization Information

ProjectionPl is reentrant and processed without parallelization.

--- Module

Basic operators

13.8 Hough

```c
[out] HImageX HoughImage HRegionX.HoughCircleTrans
([in] VARIANT Radius )
```

Return the Hough-Transform for circles with a given radius.

The operator `HoughCircleTrans` calculates the Hough transform for circles with a certain `Radius` in the regions passed by `Region`. Hereby the centres of all possible circles in the parameter space (the Hough or accumulator space respectively) will be accumulated for each point in the image space. Circle hypotheses supported by many points in the input region thereby generate a maximum in the area showing the circle’s centre in the output image (`HoughImage`). The circles’ centres in the image space can be deduced from the coordinates of these maximums by subtracting the `Radius`. If more than one radius is transmitted, all Hough images will be shifted according to the maximal radius.

--- Parameter

- **Region** (input iconic) .............................................region ⇆ HRegionX / IHOBJECTX
  Binary edge image in which the circles are to be detected.
- **HoughImage** (output iconic) ..........................image(-array) ⇆ HImageX / HUntypedObjectX (int2)
  Hough transform for circles with a given radius.
- **Radius** (input control) .................................integer(-array) ⇆ VARIANT (integer)
  Radius of the circle to be searched in the image.

Default Value : 12
Minimum Increment : 1
Recommended Increment : 1
Number of elements : (1 ≤ Radius ≤ 500)

--- Result

The operator `HoughCircleTrans` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem ('noObjectResult',<Result>)`, the behavior in case of empty region is set via `SetSystem ('emptyRegionResult',<Result>)`. If necessary an exception handling is raised.

--- Parallelization Information

HoughCircleTrans is reentrant and processed without parallelization.

HALCON/COM Reference Manual, 2005-2-1
Centres of circles for a specific radius.

HoughCircleTrans detects the centres of circles in regions with the help of the Hough transform for circles with a specific radius.

Parameter

▷ RegionIn (input iconic) ........................................... region ↷ HRegionX / IHObjectX
   Binary edge image in which the circles are to be detected.

▷ RegionOut (output iconic) ................................. region(-array) ↷ HRegionX / HUntypedObjectX
   Centres of those circles which are included in the edge image by Percent percent.
   Number of elements : \((\text{RegionOut} = (\text{Radius} \cdot \text{Percent}) \cdot \text{Mode})\)

▷ Radius (input control) ............................................ integer(-array) ↷ VARIANT ( integer )
   Radius of the circle to be searched in the image.
   Default Value : 12
   Typical range of values : \(2 \leq \text{Radius} \leq 2\) (lin)
   Minimum Increment : 1
   Recommended Increment : 1
   Number of elements : \((1 \leq \text{Radius}) \leq 500\)

▷ Percent (input control) ............................................ integer(-array) ↷ VARIANT ( integer )
   Indicates the percentage (approximately) of the (ideal) circle which must be present in the edge image RegionIn.
   Default Value : 60
   Typical range of values : \(10 \leq \text{Percent} \leq 10\) (lin)
   Minimum Increment : 1
   Recommended Increment : 5
   Number of elements : \((1 \leq \text{Percent}) \leq 100\)

▷ Mode (input control) ............................................ integer(-array) ↷ VARIANT ( integer )
   The modus defines the position of the circle in question:
   0 - the radius is equivalent to the outer border of the set pixels.
   1 - the radius is equivalent to the centres of the circle lines’ pixels.
   2 - both 0 and 1 (a little more fuzzy, but more reliable in contrast to circles set slightly differently, necessitates 50 \% more processing capacity compared to 0 or 1 alone).
   List of values : \(\text{Mode} \in \{0, 1, 2\}\)
   Number of elements : \((1 \leq \text{Mode}) \leq 3\)

Result

The operator HoughCircles returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator SetSystem(‘noObjectResult’, <Result>), the behavior in case of empty region is set via SetSystem(‘emptyRegionResult’, <Result>). If necessary an exception handling is raised.

Parallelization Information

HoughCircles is reentrant and processed without parallelization.

Module

Region processing
### Produce the Hough transform for lines within regions.

The operator `HoughLineTrans` calculates the Hough transform for lines in those regions transmitted by `Region`. Thereby the angles and the lengths of the lines’ normal vectors are registered in the parameter space (the Hough- or accumulator space respectively). This means that the parameterization is executed according to the HNF.

The result is registered in a newly generated Int2-Image (`HoughImage`), whereby the x-axis is equivalent to the angle between the normal vector and the x-axis (in the original image), and the y-axis is equivalent to the distance of the line from the origin.

The angle ranges from -90 to 180 degrees and will be registered with a resolution of 1/AngleResolution, which means that one pixel in x-direction is equivalent to 1/AngleResolution and that the `HoughImage` has a width of 270 * AngleResolution + 1 pixel. The height of the `HoughImage` corresponds to the diagonal of the surrounding rectangle of the input region.

The maxima in the result image are equivalent to the parameter values of the lines in the original image.

#### Parameter

- **Region** (input iconic) .............................. region \( \rightarrow \) `HRegionX / IObjectX`
  Binary edge image in which lines are to be detected.

- **HoughImage** (output iconic) ....................... image \( \rightarrow \) `HImageX / HUntypedObjectX (int2)`
  Hough transform for lines.

- **AngleResolution** (input control) ...................... integer \( \rightarrow \) long / VARIANT
  Adjusting the resolution in the angle area.

  **Default Value**: 4

  **List of values**: `AngleResolution \in \{1, 2, 4, 8\}`

#### Result

The operator `HoughLineTrans` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator `SetSystem('noObjectResult'<Result>)`, the behavior in case of empty region is set via `SetSystem('emptyRegionResult'<Result>)`. If necessary an exception handling is raised.

#### Parallelization Information

`HoughLineTrans` is reentrant and processed without parallelization.

### Possible Predecessors

- `Threshold`, `Skeleton`

### Possible Successors

- `Threshold`, `LocalMax`

### See also

- `HoughCircleTrans`, `GenRegionHline`

### Module

- Region processing

---

```c
[out] HImageX HoughImage HRegionX.HoughLineTrans
([in] long AngleResolution )

void HOperatorSetX.HoughLineTrans ([in] IObjectX Region,
```

## Compute the Hough transform for lines using local gradient direction.

```c
[out] HImageX HoughImage HImageX.HoughLineTransDir
([in] long DirectionUncertainty, [in] long AngleResolution )

void HOperatorSetX.HoughLineTransDir ([in] IObjectX ImageDir,
[out] HUntypedObjectX HoughImage, [in] VARIANT DirectionUncertainty,
[in] VARIANT AngleResolution )
```
The operator \texttt{HoughLineTransDir} calculates the Hough transform for lines in those regions passed in the domain of \texttt{ImageDir}. To do so, the angles and the lengths of the lines' normal vectors are registered in the parameter space (the so-called Hough or accumulator space).

In contrast to \texttt{HoughLineTrans}, additionally the edge direction in \texttt{ImageDir} (e.g., returned by \texttt{SobelDir} or \texttt{EdgesImage}) is taken into account. This results in a more efficient computation and in a reduction of the noise in the Hough space.

The parameter \texttt{DirectionUncertainty} describes how much the edge direction of the individual points within a line is allowed to vary. For example, with \texttt{DirectionUncertainty = 10} a horizontal line (i.e., edge direction = 0 degrees) may contain points with an edge direction between -10 and +10 degrees. The higher \texttt{DirectionUncertainty} is chosen, the higher the computation time will be. For \texttt{DirectionUncertainty = 180} \texttt{HoughLineTransDir} shows the same behavior as \texttt{HoughLineTrans}, i.e., the edge direction is ignored. \texttt{DirectionUncertainty} should be chosen at least as high as the step width of the edge direction stored in \texttt{ImageDir}. The minimum step width is 2 degrees (defined by the image type 'direction').

The result is stored in a newly generated UINT2-Image (\texttt{HoughImage}), where the x-axis (i.e., columns) represents the angle between the normal vector and the x-axis of the original image, and the y-axis (i.e., rows) represents the distance of the line from the origin.

The angle ranges from -90 to 180 degrees and will be stored with a resolution of 1/\texttt{AngleResolution}, which means that one pixel in x-direction is equivalent to 1/\texttt{AngleResolution} degrees and that the \texttt{HoughImage} has a width of \(270 \times \texttt{AngleResolution} + 1\) pixels. The height of the \texttt{HoughImage} corresponds to the length of the diagonal of the surrounding rectangle of the input region.

The local maxima in the result image are equivalent to the parameter values of the lines in the original image.

\begin{itemize}
\item \textbf{ImageDir (input iconic)} \hspace{1cm} \texttt{image \sim HI mageX / HIObjectX ( direction )}
\hspace{1cm} Image containing the edge direction. The edges must be described by the image domain.
\item \textbf{HoughImage (output iconic)} \hspace{1cm} \texttt{image \sim HImageX / HUntypedObjectX ( uint2 )}
\hspace{1cm} Hough transform.
\item \textbf{DirectionUncertainty (input control)} \hspace{1cm} \texttt{angle.deg \sim long / VARIANT}
\hspace{1cm} Uncertainty of the edge direction (in degrees).
\hspace{1cm} Default Value : 2
\hspace{1cm} Typical range of values : \(2 \leq \text{DirectionUncertainty} \leq 2\)
\hspace{1cm} Minimum Increment : 2
\item \textbf{AngleResolution (input control)} \hspace{1cm} \texttt{integer \sim long / VARIANT}
\hspace{1cm} Resolution in the angle area (in 1/degrees).
\hspace{1cm} Default Value : 4
\hspace{1cm} List of values : \texttt{AngleResolution \in \{1, 2, 4, 8\}}
\end{itemize}

The operator \texttt{HoughLineTransDir} returns the value TRUE if the input is not empty. The behavior in case of empty input is set via the operator \texttt{SetSystem('noObjectResult',<Result>)}. If necessary an exception handling is raised.

\begin{itemize}
\item \textbf{Parallelization Information}
\item \texttt{HoughLineTransDir} is \textit{reentrant} and processed \textit{without} parallelization.
\end{itemize}

\begin{itemize}
\item \textbf{Possible Predecessors} \hspace{1cm} EdgesImage, SobelDir, Threshold, HysteresisThreshold, NonmaxSuppressionDir, ReduceDomain
\item \textbf{Possible Successors} \hspace{1cm} GaussImage, Threshold, LocalMax, PlateausCenter
\end{itemize}

\begin{itemize}
\item \textbf{See also} \hspace{1cm} \texttt{HoughLineTrans, HoughLines, HoughLinesDir}
\item \textbf{Module} \hspace{1cm} \texttt{Tools}
\end{itemize}
Detect lines in edge images with the help of the Hough transform and returns it in HNF.

The operator **HoughLines** allows the selection of linelike structures in a region, whereby it is not necessary that the individual points of a line are connected. This process is based on the Hough transform. The lines are returned in HNF, that is by the direction and length of their normal vector.

The parameter **AngleResolution** defines the degree of exactness concerning the determination of the angles. It amounts to $1/\text{AngleResolution}$ degree. The parameter **Threshold** determines by how many points of the original region a line’s hypothesis has to be supported at least in order to be taken over into the output. The parameters **AngleGap** and **DistGap** define a neighborhood of the points in the Hough image in order to determine the local maxima. The lines are returned in HNF.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RegionIn</strong></td>
<td>(input iconic) Binary edge image in which the lines are to be detected.</td>
</tr>
<tr>
<td><strong>AngleResolution</strong></td>
<td>(input control) Adjusting the resolution in the angle area. Default Value: 4 List of values: $\text{AngleResolution} \in {1, 2, 4, 8}$</td>
</tr>
<tr>
<td><strong>Threshold</strong></td>
<td>(input control) Threshold value in the Hough image. Default Value: 100</td>
</tr>
<tr>
<td><strong>AngleGap</strong></td>
<td>(input control) Minimal distance of two maxima in the Hough image (direction: angle). Default Value: 5</td>
</tr>
<tr>
<td><strong>DistGap</strong></td>
<td>(input control) Minimal distance of two maxima in the Hough image (direction: distance). Default Value: 5</td>
</tr>
<tr>
<td><strong>Angle</strong></td>
<td>(output control) Angles (in radians) of the detected lines’ normal vectors. Typical range of values: $-1.5707963 \leq \text{Angle} \leq 1.5707963$</td>
</tr>
<tr>
<td><strong>Dist</strong></td>
<td>(output control) Distance of the detected lines from the origin.</td>
</tr>
</tbody>
</table>

**Result**

The operator **HoughLines** returns the value TRUE if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator **SetSystem**('noObjectResult',<Result>). the behavior in case of empty region is set via **SetSystem**('emptyRegionResult',<Result>). If necessary an exception handling is raised.

**Parallelization Information**

**HoughLines** is reentrant and processed without parallelization.

**Possible Predecessors**

**Possible Successors**

**See also**

**Module**

Region processing
Detect lines in edge images with the help of the Hough transform using local gradient direction and return them in normal form.

The operator `HoughLinesDir` selects line-like structures in a region based on the Hough transform. The individual points of a line can be unconnected. The region is given by the domain of `ImageDir`. The lines are returned in Hessian normal form (HNF), that is by the direction and length of their normal vector.

In contrast to `HoughLines`, additionally the edge direction in `ImageDir` (e.g., returned by `SobelDir` or `EdgesImage`) is taken into account. This results in a more efficient computation and in a reduction of the noise in the Hough space.

The parameter `DirectionUncertainty` describes how much the edge direction of the individual points within a line is allowed to vary. For example, with `DirectionUncertainty = 10` a horizontal line (i.e., edge direction = 0 degrees) may contain points with an edge direction between -10 and +10 degrees. The higher `DirectionUncertainty` is chosen, the higher the computation time will be. For `DirectionUncertainty = 180` `HoughLinesDir` shows the same behavior as `HoughLines`, i.e., the edge direction is ignored. `DirectionUncertainty` should be chosen at least as high as the step width of the edge direction stored in `ImageDir`. The minimum step width is 2 degrees (defined by the image type 'direction').

The parameter `AngleResolution` defines how accurately the angles are determined. The accuracy amounts to \(1/\text{AngleResolution}\) degrees. A subsequent smoothing of the Hough space results in an increased stability. The smoothing filter can be selected by `Smoothing`, the degree of smoothing by the parameter `FilterSize` (see `MeanImage` or `GaussImage` for details). The parameter `Threshold` determines by how many points of the original region a line's hypothesis must at least be supported in order to be selected into the output. The parameters `AngleGap` and `DistGap` define a neighborhood of the points in the Hough image in order to determine the local maxima: `AngleGap` describes the minimum distance of two maxima in the Hough image in angle direction and `DistGap` in distance direction, respectively. Thus, maxima exceeding `Threshold` but lying close to an even higher maximum are eliminated. This can particularly be helpful when searching for short and long lines simultaneously. Besides the unsmoothed Hough image `HoughImage`, the lines are returned in HNF (`Angle`, `Dist`). If the parameter `GenLines` is set to 'true', additionally those regions in `ImageDir` are returned that contributed to the local maxima in Hough space. They are stored in the parameter `Lines`.

---

### Parameter

- **ImageDir** (input iconic) .................. image \(\sim HImageX / IHObjectX\) (direction) Image containing the edge direction. The edges are described by the image domain.
- **HoughImage** (output iconic) .............. image \(\sim HImageX / HUntypedObjectX\) (uint2) Hough transform.
- **Lines** (output iconic) ...................... region \(\sim HRegionX / HUntypedObjectX\) Regions of the input image that contributed to the local maxima.
- **DirectionUncertainty** (input control) .......... `angle.deg` \(\sim long /\) VARIANT Uncertainty of edge direction (in degrees).
  - **Default Value**: 2
  - **Typical range of values**: \(2 \leq \text{DirectionUncertainty} \leq 2\)
  - **Minimum Increment**: 2
- **AngleResolution** (input control) ................ integer \(\sim long /\) VARIANT Resolution in the angle area (in degrees).
  - **Default Value**: 4
  - **List of values**: `AngleResolution \in \{1, 2, 4, 8\}`
CHAPTER 13. TOOLS

Smoothing (input control) ................................................. string ~ String / VARIANT
Smoothing filter for hough image.
Default Value: 'mean'
List of values: Smoothing ∈ {'none', 'mean', 'gauss'}

FilterSize (input control) .............................................. integer ~ long / VARIANT
Required smoothing filter size.
Default Value: 5
List of values: FilterSize ∈ {3, 5, 7, 9, 11}

Threshold (input control) ............................................ integer ~ long / VARIANT
Threshold value in the Hough image.
Default Value: 100

AngleGap (input control) ............................................. integer ~ long / VARIANT
Minimum distance of two maxima in the Hough image (direction: angle).
Default Value: 5

DistGap (input control) ............................................. integer ~ long / VARIANT
Minimum distance of two maxima in the Hough image (direction: distance).
Default Value: 5

GenLines (input control) ............................................. string ~ String / VARIANT
Create line regions if 'true'.
Default Value: 'true'
List of values: GenLines ∈ {'true', 'false'}

Angle (output control) ............................................. hesseline.angle.rad ~ VARIANT( real )
Angles (in radians) of the detected lines' normal vectors.
Typical range of values: -1.5707963 ≤ Angle ≤ -1.5707963

Dist (output control) ............................................. hesseline.distance ~ VARIANT( real )
Distance of the detected lines from the origin.
Number of elements: (Dist = Angle)

The operator HoughLines returns the value TRUE if the input is not empty. The behavior in case of empty
input (no input regions available) is set via the operator SetSystem('noObjectResult',<Result>). If
necessary an exception handling is raised.

Parallelization Information
HoughLinesDir is reentrant and processed without parallelization.

Possible Predecessors
EdgesImage, SobelDir, Threshold, NonmaxSuppressionDir, ReduceDomain, Skeleton

Possible Successors
GenRegionHline, SelectMatchingLines

See also
HoughLineTransDir, HoughLineTrans, GenRegionHline, HoughCircles

Module
Region processing

Select those lines from a set of lines (in HNF) which fit best into a region.

Lines which fit best into a region can be selected from a set of lines which are available in HNF with the help of
the operator SelectMatchingLines; the region itself is also transmitted as a parameter (RegionIn). The
The width of the lines can be indicated by the parameter \textit{LineWidth}. The selected lines will be returned in HNF and as regions (\textit{RegionLines}).

The lines are selected iteratively in a loop: At first, the line showing the greatest overlap with the input region is selected from the set of input lines. This line will then be taken over into the output set whereby all points belonging to that line will not be considered in the further steps determining overlaps. The loop will be left when the maximum overlap value of the region and the lines falls below a certain threshold value (\textit{Thresh}). The selected lines will be returned as regions as well as in HNF.

\begin{itemize}
  \item \textbf{RegionIn} (input iconic) \texttt{\ldots\ldots\ldots\ldots region $\sim$ HRegionX / HObjectX} Region in which the lines are to be matched.
  \item \textbf{RegionLines} (output iconic) \texttt{\ldots\ldots\ldots\ldots \text{region(-array)} $\sim$ HRegionX / HUntypedObjectX} Region array containing the matched lines.
  \item \textbf{AngleIn} (input control) \texttt{\ldots\ldots\ldots\ldots \text{hesseline.angle.rad(-array)} $\sim$ VARIANT (real)} Angles (in radians) of the normal vectors of the input lines.
    \begin{itemize}
      \item \textbf{Typical range of values}: $-1.5707963 \leq \text{AngleIn} \leq -1.5707963$
    \end{itemize}
  \item \textbf{DistIn} (input control) \texttt{\ldots\ldots\ldots\ldots \text{hesseline.distance(-array)} $\sim$ VARIANT (real)} Distances of the input lines form the origin.
    \begin{itemize}
      \item \textbf{Number of elements} ($\text{DistIn} = \text{AngleIn}$)
    \end{itemize}
  \item \textbf{LineWidth} (input control) \texttt{\ldots\ldots\ldots\ldots \text{integer} $\sim$ long / VARIANT} Widths of the lines.
    \begin{itemize}
      \item \textbf{Default Value}: 7
    \end{itemize}
  \item \textbf{Thresh} (input control) \texttt{\ldots\ldots\ldots\ldots \text{integer} $\sim$ long / VARIANT} Threshold value for the number of line points in the region.
    \begin{itemize}
      \item \textbf{Default Value}: 100
    \end{itemize}
  \item \textbf{AngleOut} (output control) \texttt{\ldots\ldots\ldots\ldots \text{hesseline.angle.rad(-array)} $\sim$ VARIANT (real)} Angles (in radians) of the normal vectors of the selected lines.
    \begin{itemize}
      \item \textbf{Typical range of values}: $-1.5707963 \leq \text{AngleOut} \leq -1.5707963$
      \item \textbf{Number of elements} ($\text{AngleOut} \leq \text{AngleIn}$)
    \end{itemize}
  \item \textbf{DistOut} (output control) \texttt{\ldots\ldots\ldots\ldots \text{hesseline.distance(-array)} $\sim$ VARIANT (real)} Distances of the selected lines from the origin.
    \begin{itemize}
      \item \textbf{Number of elements} ($\text{DistOut} = \text{AngleOut}$)
    \end{itemize}
\end{itemize}

The operator \texttt{SelectMatchingLines} returns the value \texttt{TRUE} if the input is not empty. The behavior in case of empty input (no input regions available) is set via the operator \texttt{SetSystem ('noObjectResult',<Result>)}, the behavior in case of empty region is set via \texttt{SetSystem ('emptyRegionResult',<Result>)}. If necessary an exception handling is raised.

\begin{itemize}
  \item \textbf{Parallelization Information}
  \item \texttt{SelectMatchingLines} is \textit{reentrant} and processed \textit{without} parallelization.
\end{itemize}

\subsection*{Possible Predecessors}
\texttt{HoughLines}

\section*{13.9 Image-Comparison}

\begin{verbatim}
void HMiscX.ClearAllVariationModels ( )
void HOperatorSetX.ClearAllVariationModels ( )
\end{verbatim}

Free the memory of all variation models.

\texttt{ClearAllVariationModels} frees the memory of all variation models that were created by calling \texttt{CreateVariationModel}. After calling \texttt{ClearAllVariationModels}, no model can be used any longer.

\begin{itemize}
  \item \textbf{Result}
  \item \texttt{ClearAllVariationModels} always returns \texttt{TRUE}.
\end{itemize}
ClearAllVariationModels is processed completely exclusively without parallelization.

Possible Predecessors
CreateVariationModel

Alternatives
ClearVariationModel

Module
Image / region / XLD management

**void** **HOperatorSetX.ClearVariationModel** ([in] VARIANT ModelID )

Free the memory of a variation model.

ClearVariationModel frees the memory of a variation model that was created by CreateVariationModel. After calling CreateVariationModel, the model can no longer be used. The handle ModelID becomes invalid.

Parameter

▷ ModelID (input control) .................. variation model \(\sim\) HVariationModelX / VARIANT ID of the variation model.

Result

ClearVariationModel returns TRUE if all parameters are correct.

Parallelization Information

ClearVariationModel is processed completely exclusively without parallelization.

Possible Predecessors
CreateVariationModel

Alternatives
ClearAllVariationModels

Module
Image / region / XLD management

**[out]** HRegionX Region **HVariationModelX.CompareVariationModel**

([in] HImageX Image )

**[out]** HRegionX Region **HImageX.CompareVariationModel**

([in] HVariationModelX ModelID )

**void** **HOperatorSetX.CompareVariationModel** ([in] IHObjectX Image, [out] HUntypedObjectX Region, [in] VARIANT ModelID )

Compare an image to a variation model.

CompareVariationModel compares the input image Image to the variation model given by ModelID. Before CompareVariationModel can be called, the two internal threshold images of the variation model must have been created with PrepareVariationModel. Let \(c(x,y)\) denote the input image Image and \(t_u,t_l\) denote the two threshold images (see PrepareVariationModel). Then the output region Region contains all points that differ substantially from the model, i.e., the points that fulfill the following condition:

\[
c(x,y) > t_u(x,y) \lor c(x,y) < t_l(x,y) .
\]

Parameter

▷ Image (input iconic) ..................image(-array) \(\sim\) HImageX / IHObjectX ( byte, int2, uint2 ) Image of the object to be trained.

▷ Region (output iconic) ..................region(-array) \(\sim\) HRegionX / HUntypedObjectX Region containing the points that differ substantially from the model.
13.9. IMAGE-COMPARISON

ModelID (input control) ........................................... variation_model  \sim HVariationModelX / VARIANT
ID of the variation model.

Example

open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1, 'default', 'model.seq', 'default', -1, -1, FGHandle)
read_region (Region, 'model.reg')
area_center (Region, Area, RowRef, ColumnRef)
read_shape_model ('model.shape', TemplateID)
read_variation_model ('model.var', ModelID)
for K := 1 to 10000 by 1
  grab_image (Image, FGHandle)
  find_shape_model (Image, TemplateID, 0, rad(360), 0.5, 1, 0.5, 'true', 4, 0.9, Row, Column, Angle, Score)
  disp_obj (Image, WindowHandle)
  if (|Score| = 1)
    vector_angle_to_rigid (Row, Column, Angle, RowRef, ColumnRef, 0, HomMat2D)
    affine_trans_image (Image, ImageTrans, HomMat2D, 'constant', 'false')
  compare_variation_model (ImageTrans, RegionDiff, ModelID)
  disp_obj (RegionDiff, WindowHandle)
endif
endfor
clear_shape_model (TemplateID)
clear_variation_model (ModelID)
close_framegrabber (FGHandle)

Result

CompareVariationModel returns TRUE if all parameters are correct and if the internal threshold images have been generated with PrepareVariationModel.

Parallelization Information

CompareVariationModel is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessors

PrepareVariationModel

Alternatives

DynThreshold

Module

Image / region / XLD management

void HVariationModelX.CreateVariationModel ([in] long Width, [in] long Height, [in] String Type, [in] String Mode )


Create a variation model for image comparison.

CreateVariationModel creates a variation model that can be used for image comparison. The handle for the variation model is returned in ModelID.

Typically, the variation model is used to discriminate correctly manufactured objects (“good objects”) from incorrectly manufactured objects (“bad objects”). It is assumed that the discrimination can be done solely based on the gray values of the object.
The variation model consists of an ideal image of the object to which the images of the objects to be tested are compared later on with \texttt{CompareVariationModel} and an image that represents the amount of gray value variation at every point of the object. The size of the images with which the object model is trained and with which the model is compared later on is passed in \texttt{Width} and \texttt{Height}, respectively. The image type of the images used for training and comparison is passed in \texttt{Type}.

The variation model is trained using multiple images of good objects. Therefore, it is essential that the training images show the objects in the same position and rotation. If this cannot be guaranteed by external means, the pose of the object can, for example, be determined by using matching (see \texttt{FindShapeModel}). The image can then be transformed to a reference pose with \texttt{AffineTransImage}.

The parameter \texttt{Mode} is used to determine how the image of the ideal object and the corresponding variation image are computed. For \texttt{Mode='standard'}, the ideal image of the object is computed as the mean of all training images at the respective image positions. The corresponding variation image is computed as the standard deviation of the training images at the respective image positions. This mode has the advantage that the variation model can be trained iteratively, i.e., as soon as an image of a good object becomes available, it can be trained with \texttt{TrainVariationModel}. The disadvantage of this mode is that great care must be taken to ensure that only images of good objects are trained, because the mean and standard deviation are not robust against outliers, i.e., if an image of a bad object is trained inadvertently, the accuracy of the ideal object image and that of the variation image might be degraded.

If it cannot be avoided that the variation model is trained with some images of objects that can contain errors, \texttt{Mode} can be set to \textquote{robust}. In this mode, the image of the ideal object is computed as the median of all training images at the respective image positions. The corresponding variation image is computed as a suitably scaled median absolute deviation of the training images and the median image at the respective image positions. This mode has the advantage that it is robust against outliers. It has the disadvantage that it cannot be trained iteratively, i.e., all training images must be accumulated using \texttt{ConcatObj} and be trained with \texttt{TrainVariationModel} in a single call.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default Value</th>
<th>Suggested values</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Width}</td>
<td>Width of the images to be compared.</td>
<td>640</td>
<td>{160, 192, 320, 384, 640, 768}</td>
</tr>
<tr>
<td>\texttt{Height}</td>
<td>Height of the images to be compared.</td>
<td>480</td>
<td>{120, 144, 240, 288, 480, 576}</td>
</tr>
<tr>
<td>\texttt{Type}</td>
<td>Type of the images to be compared.</td>
<td>\textquote{byte}</td>
<td>{'byte', 'int2', 'uint2'}</td>
</tr>
<tr>
<td>\texttt{Mode}</td>
<td>Method used for computing the variation model.</td>
<td>\textquote{standard}</td>
<td>{'standard', \textquote{robust}'}</td>
</tr>
<tr>
<td>\texttt{ModelID}</td>
<td>ID of the variation model.</td>
<td></td>
<td>\texttt{HVariationModelX} / \texttt{VARIANT}</td>
</tr>
</tbody>
</table>

\texttt{CreateVariationModel} returns \texttt{TRUE} if all parameters are correct.

\texttt{CreateVariationModel} is processed \textit{completely exclusively} without parallelization.

\texttt{TrainVariationModel} \texttt{ClearVariationModel, FindShapeModel, AffineTransImage}

\texttt{Image / region / XLD management}
13.9. IMAGE-COMPARISON

Return the images used for image comparison by a variation model.

GetVariationModel returns the images of the ideal object and the corresponding variation image of the variation model ModelID in Image and VarImage, respectively. The returned images can be used to check whether an image of a bad object has been trained with TrainVariationModel. This can be seen from the variation image. If an image of a bad object has been trained, the variation image typically has large variations in areas that should exhibit no variations.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong> (output iconic)</td>
</tr>
<tr>
<td><strong>VarImage</strong> (output iconic)</td>
</tr>
<tr>
<td><strong>ModelID</strong> (input control)</td>
</tr>
</tbody>
</table>

GetVariationModel returns TRUE if all parameters are correct.

**Parallelization Information**

GetVariationModel is reentrant and processed without parallelization.

**Possible Predecessors**

TrainVariationModel

**See also**

PrepareVariationModel, CompareVariationModel

**Module**

Image / region / XLD management

Prepare a variation model for comparison with an image.

PrepareVariationModel prepares a variation model for the image comparison with CompareVariationModel. This is done by converting the ideal image and the variation image that have been trained with TrainVariationModel into two threshold images and storing them in the variation model. These threshold images are used in CompareVariationModel to speed up the comparison of the current image to the variation model.

Two thresholds are used to compute the threshold images. The parameter **AbsThreshold** determines the minimum amount of gray levels by which the image of the current object must differ from the image of the ideal object. The parameter **VarThreshold** determines a factor relative to the variation image for the minimum difference of the current image and the ideal image. Let \( i(x, y) \) be the ideal image, \( v(x, y) \) the variation image, \( a = \text{AbsThreshold} \), and \( b = \text{VarThreshold} \). Then the two threshold images \( t_u, t_l \) are computed as follows:

\[
  t_u(x, y) = i(x, y) + \max\{a, bv(x, y)\} \quad t_l(x, y) = i(x, y) - \max\{a, bv(x, y)\}.
\]

If the current image \( c(x, y) \) is compared to the variation model using CompareVariationModel, the output region contains all points that differ substantially from the model, i.e., that fulfill the following condition:

\[
c(x, y) > t_u(x, y) \lor c(x, y) < t_l(x, y).
\]
**Parameter**

- **ModelID** (input control) .................................................. variation \_model $\leadsto$ HVariationModelX / VARIANT ID of the variation model.
- **AbsThreshold** (input control) ........................................... number $\leadsto$ VARIANT (real, integer)
  Absolute minimum threshold for the differences between the image and the variation model.
  - Default Value: 10
  - Suggested values: \( \text{AbsThreshold} \in \{0, 5, 10, 15, 20, 30, 40, 50\} \)
  - Restriction: \( \text{AbsThreshold} \geq 0 \)
- **VarThreshold** (input control) ......................................... number $\leadsto$ VARIANT (real, integer)
  Threshold for the differences based on the variation of the variation model.
  - Default Value: 2
  - Suggested values: \( \text{VarThreshold} \in \{1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5\} \)
  - Restriction: \( \text{VarThreshold} \geq 0 \)

**Result**

PrepareVariationModel returns TRUE if all parameters are correct.

**Parallelization Information**

PrepareVariationModel is processed under *mutual exclusion* against itself and without parallelization.

**Possible Predecessors**

TrainVariationModel

**Possible Successors**

CompareVariationModel

**See also**

CreateVariationModel

**Module**

Image / region / XLD management

```
void HVariationModelX.ReadVariationModel ([in] String FileName )

void HOperatorSetX.ReadVariationModel ([in] VARIANT FileName, [out] VARIANT ModelID )
```

Read a variation model from a file.

The operator ReadVariationModel reads a variation model, which has been written with WriteVariationModel, from the file FileName.

**Parameter**

- **FileName** (input control) .................................................filename $\leadsto$ String / VARIANT File name.
- **ModelID** (output control) .............................................. variation \_model $\leadsto$ HVariationModelX / VARIANT ID of the variation model.

**Result**

If the file name is valid, the operator ReadVariationModel returns TRUE. If necessary an exception handling is raised.

**Parallelization Information**

ReadVariationModel is *reentrant* and processed without parallelization.

**Possible Successors**

CompareVariationModel

**See also**

WriteVariationModel

**Module**

Image / region / XLD management
void HVariationModelX.TrainVariationModel ([in] HImageX Images )
void HImageX.TrainVariationModel ([in] HVariationModelX ModelID )
void HOperatorSetX.TrainVariationModel ([in] IHOBJECTX Images, [in] VARIANT ModelID )

Train a variation model.

TrainVariationModel trains the variation model that is passed in ModelID with one or more images, which are passed in Images.

As described for CreateVariationModel, a variation model that has been created using the mode 'standard' can be trained iteratively, i.e., as soon as images of good objects become available, they can be trained with TrainVariationModel. The ideal image of the object is computed as the mean of all previous training images and the images that are passed in Images. The corresponding variation image is computed as the standard deviation of the training images and the images that are passed in Images.

If the variation model has been created using the mode 'robust', the model cannot be trained iteratively, i.e., all training images must be accumulated using ConcatObj and be trained with TrainVariationModel in a single call. If any images have been trained previously, the training information of the previous call is discarded. The image of the ideal object is computed as the median of all training images passed in Images. The corresponding variation image is computed as a suitably scaled median absolute deviation of the training images and the median image.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Images (input iconic) ..................... image(-array) ~ HImageX / IHOBJECTX ( byte, int2, uint2 )</td>
</tr>
<tr>
<td>Images of the object to be trained.</td>
</tr>
<tr>
<td>ModelID (input control) ......................... variation_model ~ HVariationModelX / VARIANT</td>
</tr>
<tr>
<td>ID of the variation model.</td>
</tr>
</tbody>
</table>

Example

```
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1, 'default', 'model.seq', 'default', -1, -1, FGHandle)
grab_image (Image, FGHandle)
get_image_pointer1 (Image, Pointer, Type, Width, Height)
disp_obj (Image, WindowHandle)
draw_region (Region, WindowHandle)
reduce_domain (Image, Region, ImageReduced)
area_center (Region, Area, RowRef, ColumnRef)
create_shape_model (ImageReduced, 4, 0, rad(360), rad(1), 'none', 'use_polarity', 40, 10, TemplateID)
create_variation_model (Width, Height, Type, 'standard', ModelID)
for K := 1 to 100 by 1
  grab_image (Image, FGHandle)
  find_shape_model (Image, TemplateID, 0, rad(360), 0.5, 1, 0.5, 'true', 4, 0.9, Row, Column, Angle, Score)
  if (|Score| = 1)
    vector_angle_to_rigid (Row, Column, Angle, RowRef, ColumnRef, 0, HomMat2D)
    affine_trans_image (Image, ImageTrans, HomMat2D, 'constant', 'false')
    train_variation_model (ImageTrans, ModelID)
  endif
endfor
prepare_variation_model (ModelID, 10, 4)
write_region (Region, 'model.reg')
write_shape_model (TemplateID, 'model.shape')
write_variation_model (ModelID, 'model.var')
clear_shape_model (TemplateID)
```

HALCON 6.1.4
clear_variation_model (ModelID)
close_framegrabber (FGHandle)

---

**TrainVariationModel** returns TRUE if all parameters are correct.

---

**Parallelization Information**

*TrainVariationModel* is processed under *mutual exclusion* against itself and without parallelization.

---

**Possible Predecessors**

CreateVariationModel, FindShapeModel, AffineTransImage, ConcatObj

---

**Possible Successors**

PrepareVariationModel

---

See also

PrepareVariationModel, CompareVariationModel, ClearVariationModel

---

**Module**

Image / region / XLD management

```c
void HVariationModelX.WriteVariationModel ([in] String FileName )
void HOperatorSetX.WriteVariationModel ([in] VARIANT ModelID, [in] VARIANT FileName )
```

Write a variation model to a file.

*WriteVariationModel* writes a variation model to the file *FileName*. The model can be read with *ReadVariationModel*.

---

**Parameter**

- **ModelID** (input control) \( \leadsto HVariationModelX / VARIANT \) ID of the variation model.
- **FileName** (input control) \( \leadsto String / VARIANT \) File name.

---

If the file name is valid (write permission), the operator *WriteVariationModel* returns TRUE. If necessary an exception handling is raised.

---

**Parallelization Information**

*WriteVariationModel* is *reentrant* and processed *without* parallelization.

---

**Possible Predecessors**

TrainVariationModel

---

**See also**

PrepareVariationModel

---

**Module**

Image / region / XLD management

---

**13.10 Kalman-Filter**

```c
```

Estimate the current state of a system with the help of the Kalman filtering.

HALCON/COM Reference Manual, 2005-2-1
The operator FilterKalman returns an estimate of the current state (or also a prediction of a future state) of a discrete, stochastically disturbed, linear system. In practice, Kalman filters are used successfully in image processing in the analysis of image sequences (background identification, lane tracking with the help of line tracing or region analysis, etc.). A short introduction concerning the theory of the Kalman filters will be followed by a detailed description of the routine FilterKalman itself.

KALMAN FILTER: A discrete, stochastically disturbed, linear system is characterized by the following markers:

- State \( x(t) \): Describes the current state of the system (speeds, temperatures, ...).
- Parameter \( u(t) \): Inputs from outside into the system.
- Measurement \( y(t) \): Measurements gained by observing the system. They indicate the state of the system (or at least parts of it).
- An output function describing the dependence of the measurements on the state.
- A transition function indicating how the state changes with regard to time, the current value and the parameters.

The output function and the transition function are linear. Their application can therefore be written as a multiplication with a matrix.

The transition function is described with the help of the transition matrix \( A(t) \) and the parameter matrix \( C(t) \), the initial function is described by the measurement matrix \( G(t) \). Hereby \( C(t) \) characterizes the dependency of the new state on the old, \( G(t) \) indicates the dependency on the parameters. In practice it is rarely possible (or at least too time consuming) to describe a real system and its behaviour in a complete and exact way. Normally only a relatively small number of variables will be used to simulate the behaviour of the system. This leads to an error, the so called system error (also called system disturbance) \( w(t) \).

The output function, too, is usually not exact. Each measurement is faulty. The measurement errors will be called \( w(t) \). Therefore the following system equations arise:

\[
\begin{align*}
x(t + 1) &= A(t)x(t) + G(t)u(t) + v(t) \\
y(t) &= c(t)x(t) + w(t)
\end{align*}
\]

The system error \( v(t) \) and the measurement error \( w(t) \) are not known. As far as systems are concerned which are interpreted with the help of the Kalman filter, these two errors are considered as Gaussian distributed random vectors (therefore the expression "stochastically disturbed systems"). Therefore the system can be calculated, if the corresponding expected values for \( v(t) \) and \( w(t) \) as well as the covariance matrices are known.

The estimation of the state of the system is carried out in the same way as in the Gaussian-Markov-estimation. However, the Kalman filter is a recursive algorithm which is based only on the current measurements \( y(t) \) and the latest state \( x(t) \). The latter implicitly also includes the knowledge about earlier measurements.

A suitable estimate value \( x(0) \), which is interpreted as the expected value of a random variable for \( x(0) \), must be indicated for the initial value \( x(0) \). This variable should have an expected error value of 0 and the covariance matrix \( P(0) \) which also has to be indicated. At a certain time \( t \) the expected values of both disturbances \( v(t) \) and \( w(t) \) should be 0 and their covariances should be \( Q(t) \) and \( R(t) \). \( x(t), v(t) \) and \( w(t) \) will usually be assumed to be not correlated (any kind of noise-process can be modelled - however the development of the necessary matrices by the user will be considerably more demanding). The following conditions must be met by the searched estimate values \( x^t \):

- The estimate values \( x^t \) are linearly dependent on the actual value \( x(t) \) and on the measurement sequence \( y(0), y(1), \ldots, y(t) \).
- \( x^t \) being hereby considered to meet its expectations, i.e. \( Ex^t = Ex(t) \).
- The grade criterion for \( x^t \) is the criterion of minimal variance, i.e. the variance of the estimation error defined as \( x(t) - x^t \), being as small as possible.

After the initialization

\[
\hat{x}(0) = x_0, \quad \hat{P}(0) = P_0
\]

at each point in time \( t \) the Kalman filter executes the following calculation steps:
CHAPTER 13. TOOLS

As mentioned above, it is much more demanding to model any kind of noise processes. If for example the system noise and the measurement noise are correlated with the corresponding covariance matrix \( L \), the equations for the Kalman gain and the error covariance matrix have to be modified:

\[
(K - III) \quad K(t) = \frac{P(t)C'(t)}{C(t)P(t)C'(t) + R(t)} \\
(K - IV) \quad x' = \hat{x}(t) + K(t)(y(t) - C(t)\hat{x}(t)) \\
(K - V) \quad \hat{P}(t) = P(t) - K(t)C(t)\hat{P}(t) \\
(K - I) \quad \hat{x}(t + 1) = A(t)x' + G(t)u(t) \\
(K - II) \quad \hat{P}(t + 1) = A(t)\hat{P}(t)A'(t) + Q(t)
\]

This means that the user himself has to establish the linear system equations from (K-I) up to (K-V) with respect to the actual problem. The user must therefore develop a mathematical model upon which the solution to the problem can be based. Statistical characteristics describing the inaccuracies of the system as well as the measurement errors, which are to be expected, thereby have to be estimated if they cannot be calculated exactly. Therefore the following individual steps are necessary:

1. Developing a mathematical model
2. Selecting characteristic state variables
3. Establishing the equations describing the changes of these state variables and their linearization (matrices \( A \) and \( G \))
4. Establishing the equations describing the dependency of the measurement values of the system on the state variables and their linearization (matrix \( C \))
5. Developing or estimating of statistical dependencies between the system disturbances (matrix \( Q \))
6. Developing or estimating of statistical dependencies between the measurement errors (matrix \( R \))
7. Initialization of the initial state

As mentioned above, the initialization of the system (point 7) hereby necessitates to indicate an estimate \( x_0 \) of the state of the system at the time 0 and the corresponding covariance matrix \( P_0 \). If the exact initial state is not known, it is recommendable to set the components of the vector \( x_0 \) to the average values of the corresponding range, and to set high values for \( P_0 \) (about the size of the squares of the range). After a few iterations (when the number of the accumulated measurement values in total has exceeded the number of the system values), the values which have been determined in this way are also useable.

If on the other hand the initial state is known exactly, all entries for \( P_0 \) have to be set to 0, because \( P_0 \) describes the covariances of the error between the estimated value \( x_0 \) and the actual value \( x(0) \).

THE FILTER ROUTINE:
A Kalman filter is dependent on a range of data which can be organized in four groups:

**Model parameter:** transition matrix \( A \), control matrix \( G \) including the parameter \( u \) and the measurement matrix \( C \)

**Model stochastic:** system-error covariance matrix \( Q \), system-error - measurement-error covariance matrix \( L \), and measurement-error covariance matrix \( R \)

**Measurement vector:** \( y \)

**History of the system:** extrapolation vector \( \hat{x} \) and extrapolation-error covariance matrix \( \hat{P} \)
Thereby many systems can work without input "from outside", i.e. without $G$ and $u$. Further, system errors and measurement errors are normally not correlated ($L$ is dropped).

Actually the data necessary for the routine will be set by the following parameters:

**Dimension**: This parameter includes the dimensions of the status vector, the measurement vector and the controller vector. Dimension thereby is a vector $[n,m,p]$, whereby $n$ indicates the number of the state variables, $m$ the number of the measurement values and $p$ the number of the controller members. For a system without determining control (i.e. without influence "from outside") therefore $[n,m,0]$ has to be passed.

**Model**: This parameter includes the lined up matrices (vectors) $A,C,Q,G,u$ and (if necessary) $L$ having been stored in row-major order. Model therefore is a vector of the length $n \times n + n \times m + n \times n + n \times p + p$. The last summand is dropped, in case the system errors and measurement errors are not correlated, i.e. there is no value for $L$.

**Measurement**: This parameter includes the matrix $R$ which has been stored in row-major order, and the measurement vector $y$ lined up. Measurement therefore is a vector of the dimension $m \times m + m$.

**PredictionIn / PredictionOut**: These two parameters include the matrix $\hat{P}$ (the extrapolation-error covariance matrix) which has been stored in row-major order and the extrapolation vector $\hat{x}$ lined up. This means, they are vectors of the length $n \times n + n$. **PredictionIn** therefore is an input parameter, which must contain $\hat{P}(t)$ and $\hat{x}(t)$ at the current time $t$. With **PredictionOut** the routine returns the corresponding predictions $\hat{P}(t+1)$ and $\hat{x}(t+1)$.

**Estimate**: With this parameter the routine returns the matrix $\hat{P}$ (the estimation-error covariance matrix) which has been stored in row-major order and the estimated state $\hat{x}$ lined up. Estimate therefore is a vector of the length $n \times n + n$.

Please note that the covariance matrices ($Q, R, \hat{P}, \bar{P}$) must of course be symmetric.

---

### Attention

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimension</td>
<td>The dimensions of the state vector, the measurement and the controller vector.</td>
</tr>
<tr>
<td>Model</td>
<td>The lined up matrices $A, C, Q, G, u$, possibly $G$ and $u$, and if necessary $L$ which have been stored in row-major order.</td>
</tr>
<tr>
<td>Measurement</td>
<td>The matrix $R$ stored in row-major order and the measurement vector $y$ lined up.</td>
</tr>
<tr>
<td>PredictionIn</td>
<td>The matrix $\hat{P}$ (the extrapolation-error covariances) stored in row-major order and the extrapolation vector $\hat{x}$ lined up.</td>
</tr>
<tr>
<td>PredictionOut</td>
<td>The matrix $\hat{P}$ (the estimation-error covariances) stored in row-major order and the extrapolation vector $\hat{x}$ lined up.</td>
</tr>
<tr>
<td>Estimate</td>
<td>The matrix $\hat{P}$ (the estimation-error covariances) stored in row-major order and the estimated state $\hat{x}$ lined up.</td>
</tr>
</tbody>
</table>

---

// Typical procedure:
// 1. To initialize the variables, which describe the model, e.g. with
read_kalman(‘kalman.init’, Dim, Mod, Meas, Pred)

HALCON 6.1.4
// Generation of the first measurements (typical of the first image of an
// image series) with an appropriate problem-specific routine (there is a
// fictitious routine extract_features in example):
extract_features(Image1, Meas, Meas1)
// first Kalman-Filtering:
filter_kalman(Dim, Mod, Meas1, Pred, Pred1, Est1)
// To use the estimate value (if need be the prediction too)
// with a problem-specific routine (here use_est):
use_est(Est1)
// To get the next measurements (e.g. from the next image):
extract_next_features(Image2, Meas1, Meas2)
// if need be Update of the model parameter (a constant model)
// second Kalman-Filtering:
filter_kalman(Dim, Mod, Meas2, Pred1, Pred2, Est2)
use_est(Est2)
extract_next_features(Image3, Meas2, Meas3).
// etc.

Result

If the parameter values are correct, the operator FilterKalman returns the value TRUE. Otherwise an exception
handling will be raised.

Parallelization Information

FilterKalman is reentrant and processed without parallelization.

Possible Predecessors
ReadKalman, SensorKalman

Possible Successors
UpdateKalman

See also
ReadKalman, UpdateKalman, SensorKalman

References

W. Hartinger: "Entwurf eines anwendungsunabhangigen Kalman-Filters mit Untersuchungen im Bereich der
Bildfolgenanalyse"; Diplomarbeit; Technische Universitat Munchen, Institut fur Informatik, Lehrstuhl Prof.
Radig; 1991.

R.E. Kalman: "A New Approach to Linear Filtering and Prediction Problems"; Transactions ASME, Ser.D: Jour-
nal of Basic Engineering; Vol. 82, S.34-45; 1960.

R.E. Kalman, P. Falb, M.A. Arbib: "Topics in Mathematical System Theory"; McGraw-Hill Book Company,
New York; 1969.

K.-P. Karmann, A. von Brandt: "Moving Object Recognition Using an Adaptive Background Memory"; Time-
Varying Image Processing and Moving Object Recognition 2 (ed.: V. Cappellini), Proc. of the 3rd Interantional

Module

<table>
<thead>
<tr>
<th>out</th>
<th>VARIANT</th>
<th>Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMiscX.ReadKalman</td>
<td>(in) String FileName,</td>
<td></td>
</tr>
<tr>
<td>[out] VARIANT Model, [out] VARIANT Measurement, [out] VARIANT Prediction</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

void HOperatorSetX.ReadKalman | (in) VARIANT FileName, |
| [out] VARIANT Dimension, [out] VARIANT Model, [out] VARIANT Measurement, |
| [out] VARIANT Prediction |

Read the description file of a Kalman filter.

The operator ReadKalman reads the description file FileName of a Kalman filter. Kalman filters return an
estimate of the current state (or even the prediction of a future state) of a discrete, stochastically disturbed, linear
system. They are successfully used in image processing, especially in the analysis of image sequences. A Kalman
filtering is based on a mathematical model of the system to be examined which at any point in time has the following characteristics:

**Model parameter:** transition matrix $A$, control matrix $G$ including the controller output $u$ and the measurement matrix $C$

**Model stochastic:** system-error covariance matrix $Q$, system-error - measurement-error covariance matrix $L$ and measurement-error covariance matrix $R$

**Estimate of the initial state of the system:** state $x_0$ and corresponding covariance matrix $P_0$

Many systems do not need entries “‘from outside’”, and therefore $G$ and $u$ can be dropped. Further, system errors and measurement errors are normally not correlated ($L$ is dropped). The characteristics mentioned above can be stored in an ASCII-file and then can be read with the help of the operator `ReadKalman`. This ASCII-file must have the following structure:

```
Dimension row
+ content row
+ matrix $A$
+ matrix $C$
+ matrix $Q$
[[ + matrix $G$ + vector $u$ ]
[ + matrix $L$ ]
+ matrix $R$
[ + matrix $P_0$ ]
[ + vector $x_0$ ]
```

The dimension row thereby is always of the following form:

```
n = <integer>  m = <integer>  p = <integer>
```

whereby $n$ indicates the number of the state variables, $m$ the number of the measurement values and $p$ the number of the controller members (see also `Dimension`). The maximal dimension will hereby be limited by a system constant (= 30 for the time being).

The content row has the following form:

```
A * C * Q * G * u * L * R * P * x
```

and describes the following content of the file. Instead of ‘*’, ‘+’ (= parameter is available) respectively ‘-’ (= parameter is missing) have to be set. Please note that only the parameters marked by [...] in the above list may be left out in the description file. If the initial state estimate $a_0$ is missing (i.e. ‘x-’), the components of the vector will supposed to be 0.0. If the covariance matrix $P_0$ of the initial state estimate is missing (i.e. ’P-’), the error will be supposed to be tremendous. In this case the matrix elements will be set to 10000.0. This value seems to be very high, however, it is only sufficient if the range of components of the state vector $x$ is smaller to the tenth power. $(r \times s)$ matrices will be stored per row in the following form:

```
< Kommentar, d.h. string >
< a_{11} > < a_{12} > \cdots < a_{1s} >
\vdots 
< a_{r1} > < a_{r2} > \cdots < a_{rs} >
```

(the spaces and line feed characters can be chosen at will),

vectors will be stored correspondingly in the following form:

```
< comment, i.e.string >
< a_1 > \cdots < a_k >
```

The following parameter values are returned by the operator `ReadKalman`:

**Dimension:** This parameter includes the dimensions of the status vector, the measurement vector and the controller vector. `Dimension` thereby is a vector $[n,m,p]$, whereby $n$ indicates the number of the state variables, $m$ the number of the measurement values and $p$ the number of the controller members. For a system without determining control (i.e. without influence “’from outside’”) therefore `Dimension = [n,m,0]`. 

HALCON 6.1.4
**Model**: This parameter includes the lined up matrices (vectors) $A, C, Q, G, u$ and (if necessary) $L$ having been stored in row-major order. **Model** therefore is a vector of the length $n \times n + n \times m + n \times n + n \times p + p \times [n \times m]$. The last summand is dropped, in case the system errors and measurement errors are not correlated, i.e. there is no value for $L$.

**Measurement**: This parameter includes the matrix $R$ which has been stored in row-major order. **Measurement** therefore is vector of the dimension $m \times m$.

**Prediction**: This parameter includes the matrix $P_0$ (the error covariance matrix of the initial state estimate) and the initial state estimate $x_0$ lined up. This means, it is a vector of the length $n \times n + n$.

---

**Attention**

---

**Parameter**

- **FileName** (input control) ........................................ filename $\sim$ String / VARIANT
  - Description file for a Kalman filter.
  - **Default Value**: 'kalman.init'

- **Dimension** (output control) ...................................... integer $\sim$ VARIANT (integer)
  - The dimensions of the state vector, the measurement vector and the controller vector.

- **Model** (output control) ........................................... real $\sim$ VARIANT (real)
  - The lined up matrices $A, C, Q$, possibly $G$ and $u$, and if necessary $L$ stored in row-major order.

- **Measurement** (output control) .................................. real $\sim$ VARIANT (real)
  - The matrix $R$ stored in row-major order.

- **Prediction** (output control) .................................... real $\sim$ VARIANT (real)
  - The matrix $P_0$ (error covariance matrix of the initial state estimate) stored in row-major order and the initial state estimate $x_0$ lined up.

---

**Example**

%An example of the description-file:
%
% n=3 m=1 p=0
% A+C+Q+G-u-L-R+P+x+
% transition matrix A:
% 1 1 0.5
% 0 1 1
% 0 0 1
% measurement matrix C:
% 1 0
% system-error covariance matrix Q:
% 54.3 37.9 48.0
% 37.9 34.3 42.5
% 48.0 42.5 43.7
% measurement-error covariance matrix R:
% 1.2
% estimation-error covariance matrix (for the initial estimate) P0:
% 0 0
% 0 180.5 0
% 0 0 100
% initial estimate x0:
% 0 100 0
%
%the result of read_kalman with the upper descriptionfile
%as inputparameter:
%
%Dimension  = [3,1,0]
%Model      = [1.0,1.0,0.5,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
%Measurement = [1.2]
%Prediction  = [0.0,0.0,0.0,0.0,0.0,180.5,0.0,0.0,0.0,100.0,0.0,0.0,100.0,0.0,0.0,0.0,0.0].

HALCON/COM Reference Manual, 2005-2-1
13.10. KALMAN-FILTER

Result

If the description file is readable and correct, the operator `ReadKalman` returns the value TRUE. Otherwise an exception handling will be raised.

**Parallelization Information**

`ReadKalman` is reentrant and processed without parallelization.

**Possible Successors**

`FilterKalman`  

See also

`UpdateKalman`  

Module

Tools

```
[out] VARIANT MeasurementOut HMiscX.SensorKalman ([in] long Dimension,  
[in] VARIANT MeasurementIn )  
void HOperatorSetX.SensorKalman ([in] VARIANT Dimension,  
```

Interactive input of measurement values for a Kalman filtering.

The operator `SensorKalman` supports the interactive input of measurement values for a Kalman filtering. Kalman filters return an estimate of the current state (or even the prediction of a future state) of a discrete, stochastically disturbed, linear system. They are successfully used in image processing, especially in the analysis of image sequences.

Each filtering is hereby based on certain measurement values. How these values are extracted from images or sensor data depends strongly on the individual application and therefore must be entirely up to the user. However, the operator `SensorKalman` allows an interactive input of (fictitious) measurement values $y$ and the corresponding measurement-error covariance matrix $R$. Especially the testing of Kalman filters during the development can hereby be facilitated.

The parameters `MeasurementIn` and `MeasurementOut` include the matrix $R$ which has been stored in row-major order and the measurement vector $y$ lined up, i.e. they are vectors of the length $\text{Dimension} \times \text{Dimension} + \text{Dimension}$

**Attention**

**Parameter**

- **Dimension** (input control) ........................................... integer $\rightarrow$ long / VARIANT  
  Number of measurement values.  
  Default Value : 1  
  Typical range of values : $0 \leq \text{Dimension} \leq 0$

- **MeasurementIn** (input control) ...................................... real $\rightarrow$ VARIANT( real )  
  The matrix $R$ stored in row-major order and the measurement vector $y$ lined up.  
  Default Value : [1.2,1.0]  
  Typical range of values : $0.0 \leq \text{MeasurementIn} \leq 0.0$

- **MeasurementOut** (output control) ............................... real $\rightarrow$ VARIANT( real )  
  The matrix $R$ stored in row-major order and the measurement vector $y$ lined up.

**Result**

If the parameters are correct, the operator `SensorKalman` returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

`SensorKalman` is reentrant and processed without parallelization.

**Possible Successors**

`FilterKalman`  

See also

`FilterKalman`, `ReadKalman`, `UpdateKalman`
CHAPTER 13. TOOLS

Module

Read an update file of a Kalman filter.

The operator `UpdateKalman` reads the update file `FileName` of a Kalman filter. Kalman filters return an estimate of the current state (or even the prediction of a future state) of a discrete, stochastically disturbed, linear system.

A Kalman filtering is based on a mathematical model of the system to be examined which at any point in time has the following characteristics:

**Model parameter:** transition matrix $A$, control matrix $G$ including the controller output $u$ and the measurement matrix $C$

**Model stochastic:** system-error covariance matrix $Q$, system-error-measurement-error covariance matrix $L$ and measurement-error covariance matrix $R$

**Measurement vector:** $y$

**History of the system:** extrapolation vector $\hat{x}$ and extrapolation-error covariance matrix $\hat{P}$

Many systems do not need entries "from outside" and therefore $G$ and $u$ can be dropped. Further, system errors and measurement errors are normally not correlated ($L$ is dropped). Some of the characteristics mentioned above may change dynamically (from one iteration to the next). The operator `UpdateKalman` serves to modify parts of the system according to an update file (ASCII) with the following structure (see also `ReadKalman`):

- **Dimension row**
  - matrix $A$ + matrix $C$ + matrix $Q$ + matrix $G$ + vector $u$ + matrix $L$ + matrix $R$

The dimension row thereby has the following form:

$$n = \text{<integer>} \quad m = \text{<integer>} \quad p = \text{<integer>}$$

whereby $n$ indicates the number of the state variables, $m$ the number of the measurement values and $p$ the number of the controller members (see also `DimensionIn` / `DimensionOut`). The maximal dimension will hereby be limited by a system constant (= 30 for the time being). As in this case changes should take effect at a valid model, the dimensions $n$ and $m$ are invariant (and will only be indicated for purposes of control).

The content row has the following form:

$$A \ast C \ast Q \ast G \ast u \ast L \ast R$$

and describes the further content of the file. Instead of ‘*’ , ‘+’ (= parameter is available) respectively ‘*’ (= parameter is missing) has to be set. In contrast to description files for `ReadKalman`, the system description needs not be complete in this case. Only those parts of the system which are changed must be indicated. The indication of estimated values is unnecessary, as these values must stem from the latest filtering according to the structure of the filter.

$(r \times s)$ matrices will be stored in row-major order in the following form:
<comment, i.e. string>

\[ \begin{align*}
\begin{array}{cccc}
\angle a_{11} & \angle a_{12} & \cdots & \angle a_{1s} \\
\vdots & \vdots & \ddots & \vdots \\
\angle a_{r1} & \angle a_{r2} & \cdots & \angle a_{rs}
\end{array}
\end{align*} \]

(the spaces/line feed characters can be chosen at will),

vectors will be stored correspondingly in the following form:

<comment, i.e. string>

\[ \begin{align*}
\begin{array}{cccc}
\angle a_1 & \cdots & \angle a_k \\
\end{array}
\end{align*} \]

The following parameter values of the operator \texttt{ReadKalman} will be changed:

\textbf{DimensionIn/ DimensionOut}: These parameters include the dimensions of the state vector, measurement vector and controller vector and therefore are vectors \([n, m, p]\), whereby \(n\) indicates the number of the state variables, \(m\) the number of the measurement values and \(p\) the number of the controller members. \(n\) and \(m\) are invariant for a given system, i.e. they must not differ from corresponding input values of the update file. For a system without influence "from outside" \(p = 0\).

\textbf{ModelIn/ ModelOut}: These parameters include the lined up matrices (vectors) \(A, C, Q, G, u\) and if necessary \(L\) which have been stored in row-major order. \texttt{ModelIn/ ModelOut} therefore are vectors of the length \(n \times n + n \times m + n \times n + n \times p + p \times n \times m\). The last summand is dropped if system errors and measurement errors are not correlated, i.e. no value has been set for \(L\).

\textbf{MeasurementIn/ MeasurementOut}: These parameters include the matrix \(R\) stored in row-major order, and therefore are vectors of the dimension \(m \times m\).

\begin{itemize}
  \item \textbf{Attention}
\end{itemize}

\begin{itemize}
  \item \textbf{Parameter}
\end{itemize}

- **FileName** (input control) \(\ldots\) \texttt{filename} \(\sim\) \texttt{String / VARIANT}
  Update file for a Kalman filter.
  \textbf{Default Value}: 'kalman.updt'

- **DimensionIn** (input control) \(\ldots\) \texttt{DimensionIn} \(\sim\) \texttt{VARIANT(integer)}
  The dimensions of the state vector, measurement vector and controller vector.
  \textbf{Default Value}: \([3,1,0]\)
  \textbf{Typical range of values}: \(0 \leq \text{DimensionIn} \leq 0\)

- **ModelIn** (input control) \(\ldots\) \texttt{ModelIn} \(\sim\) \texttt{VARIANT(real)}
  The lined up matrices \(A, C, Q, \) possibly \(G\) and \(u\), and if necessary \(L\) which all have been stored in row-major order.
  \textbf{Default Value}: \([1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]\)
  \textbf{Typical range of values}: \(0.0 \leq \text{ModelIn} \leq 0.0\)

- **MeasurementIn** (input control) \(\ldots\) \texttt{MeasurementIn} \(\sim\) \texttt{VARIANT(real)}
  The matrix \(R\) stored in row-major order.
  \textbf{Default Value}: \([1,2]\)
  \textbf{Typical range of values}: \(0.0 \leq \text{MeasurementIn} \leq 0.0\)

- **DimensionOut** (output control) \(\ldots\) \texttt{DimensionOut} \(\sim\) \texttt{VARIANT(integer)}
  The dimensions of the state vector, measurement vector and controller vector.

- **ModelOut** (output control) \(\ldots\) \texttt{ModelOut} \(\sim\) \texttt{VARIANT(real)}
  The lined up matrices \(A, C, Q, \) possibly \(G\) and \(u\), and if necessary \(L\) which all have been stored in row-major order.

- **MeasurementOut** (output control) \(\ldots\) \texttt{MeasurementOut} \(\sim\) \texttt{VARIANT(real)}
  The matrix \(R\) stored in row-major order.
%The following values are describing the system

%DimensionIn = [3,1,0]
%ModelIn = [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,
%54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
%MeasurementIn = [1,2]
%
%An example of the Updatefile:
%
%n=3 m=1 p=0
%A+C-Q-G-u-L-R-
%transitions at time t=15:
%2 1 1
%0 2 2
%0 0 2
%
%the results of update_kalman:
%
%DimensionOut = [3,1,0]
%ModelOut = [2.0,1.0,1.0,0.0,2.0,2.0,0.0,0.0,2.0,1.0,0.0,0.0,
%54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
%MeasurementOut = [1,2]

Result

If the update file is readable and correct, the operator UpdateKalman returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

UpdateKalman is reentrant and processed without parallelization.

Possible Successors

FilterKalman

See also

ReadKalman, FilterKalman, SensorKalman

Module

13.11 Measure

Delete all measure objects.

CloseAllMeasures deletes all measure objects that have been created using GenMeasureRectangle2 or GenMeasureArc. The memory used for the measure objects is freed.

Result

CloseAllMeasures always returns TRUE.

Parallelization Information

CloseAllMeasures is reentrant and processed without parallelization.

Possible Predecessors

GenMeasureRectangle2, GenMeasureArc, MeasurePos, MeasurePairs

Alternatives

CloseMeasure

Module

Sub-pixel operators
void HOperatorSetX.CloseMeasure ([in] VARIANT MeasureHandle )

Delete a measure object. CloseMeasure deletes the measure object given by MeasureHandle. The memory used for the measure object is freed.

Parameter

▷ MeasureHandle (input control) ........................................measure.jd ~ HMeasureX / VARIANT
 Measure object handle.

Result

If the parameter values are correct the operator CloseMeasure returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

CloseMeasure is reentrant and processed without parallelization.

Possible Predecessors

GenMeasureRectangle2, GenMeasureArc, MeasurePos, MeasurePairs

See also

CloseAllMeasures

Module

Sub-pixel operators

Extract straight edge pairs perpendicular to a rectangle or an annular arc.

FuzzyMeasurePairing serves to extract straight edge pairs that lie perpendicular to the major axis of a rectangle or an annular arc. In addition to MeasurePos it uses fuzzy member functions to evaluate and select the edge pairs.

The extraction algorithm is identical to FuzzyMeasurePos. In addition, the edges are grouped to pairs: If Transition = 'positive', the edge points with a dark-to-light transition in the direction of the major axis of the
rectangle or the annular arc are returned in `RowEdgeFirst` and `ColumnEdgeFirst`. In this case, the corresponding edges with a light-to-dark transition are returned in `RowEdgeSecond` and `ColumnEdgeSecond`. If `Transition = 'negative'`, the behavior is exactly opposite. If `Transition = 'all'`, the first detected edge defines the transition for `RowEdgeFirst` and `ColumnEdgeFirst`.

Having extracted subpixel edge locations, the edges are paired. The features of a possible edge pair are evaluated by a fuzzy function, set by `SetFuzzyMeasure`. Which edge pairs are selected can be determined with the parameter `FuzzyThresh`, which constitutes a threshold on the weight over all fuzzy sets, i.e., the geometric mean of the weights of the defined fuzzy membership functions. As an extension to `FuzzyMeasurePairs`, the pairing algorithm can be restricted by `Pairing`. Currently only `no restriction` is available, which returns all possible edge pairs, allowing interleaving and inclusion of pairs. Finally, the best scored `NumPairs` edge pairs are returned, whereas 0 indicates to return all possible found edge combinations.

The selected edges are returned as single points, which lie on the major axis of the rectangle or annular arc. The corresponding edge amplitudes are returned in `AmplitudeFirst` and `AmplitudeSecond`, the fuzzy scores in `FuzzyScore`. In addition, the distance between each edge pair is returned in `IntraDistance`, corresponding to the distance between `EdgeFirst[i]` and `EdgeSecond[i]`.

---

**Attention**

`FuzzyMeasurePairing` only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle or annular arc are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle or annular arc is as close to perpendicular as possible to the edges in the image.

It should be kept in mind that `FuzzyMeasurePairing` ignores the domain of `Image` for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.

---

**Parameter**

- **Image** (input iconic) ........................................ image  \(\sim\) HIImageX / IHObjectX (byte, uint2)  
  Input image.
- **MeasureHandle** (input control) ............................... measure_id  \(\sim\) long / HMeasureX / VARIANT  
  Measure object handle.
- **Sigma** (input control) ......................................... number  \(\sim\) double / VARIANT  
  Sigma of Gaussian smoothing.
  - Default Value : 1.0
  - Suggested values : \(\Sigma \in \{0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0\}\)
  - Typical range of values : \(0.4 \leq \Sigma \leq 0.4\) (lin)
  - Minimum Increment : 0.01
  - Recommended Increment : 0.1
  - Restriction : \((\Sigma \geq 0.4)\)
- **AmpThresh** (input control) ................................... number  \(\sim\) double / VARIANT  
  Minimum edge amplitude.
  - Default Value : 30.0
  - Suggested values : \(\text{AmpThresh} \in \{5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0\}\)
  - Typical range of values : \(1 \leq \text{AmpThresh} \leq 1\) (lin)
  - Minimum Increment : 0.5
  - Recommended Increment : 2
- **FuzzyThresh** (input control) ............................... number  \(\sim\) double / VARIANT  
  Minimum fuzzy value.
  - Default Value : 0.5
  - Suggested values : \(\text{FuzzyThresh} \in \{0.1, 0.3, 0.5, 0.7, 0.9\}\)
  - Typical range of values : \(0.0 \leq \text{FuzzyThresh} \leq 0.0\) (lin)
  - Recommended Increment : 0.1
- **Transition** (input control) .................................. string  \(\sim\) String / VARIANT  
  Select the first gray value transition of the edge pairs.
  - Default Value : 'all'
  - List of values : \('all', 'positive', 'negative'\)
- **Pairing** (input control) ..................................... string  \(\sim\) String / VARIANT  
  Constraint of pairing.
  - Default Value : 'no restriction'
  - List of values : \('no restriction'\)
13.11. MEASURE

- **NumPairs** (input control)  
  Number of edge pairs.
  
  **Default Value**: 10
  
  **Suggested values**: `NumPairs ∈ {0, 1, 10, 20, 50}`
  
  **Recommended Increment**: 1

- **RowEdgeFirst** (output control)
  Row coordinate of the first edge.

- **ColumnEdgeFirst** (output control)
  Column coordinate of the first edge.

- **AmplitudeFirst** (output control)
  Edge amplitude of the first edge (with sign).

- **RowEdgeSecond** (output control)
  Row coordinate of the second edge.

- **ColumnEdgeSecond** (output control)
  Column coordinate of the second edge.

- **AmplitudeSecond** (output control)
  Edge amplitude of the second edge (with sign).

- **RowPairCenter** (output control)
  Row coordinate of the center of the edge pair.

- **ColumnPairCenter** (output control)
  Column coordinate of the center of the edge pair.

- **FuzzyScore** (output control)
  Fuzzy evaluation of the edge pair.

- **IntraDistance** (output control)
  Distance between the edges of the edge pair.

**Result**

If the parameter values are correct, the operator **FuzzyMeasurePairing** returns the value TRUE. Otherwise, an exception handling is raised.

---

**Parallelization Information**

**FuzzyMeasurePairing** is **reentrant** and processed without parallelization.

**Possible Predecessors**

GenMeasureRectangle2, GenMeasureArc, SetFuzzyMeasure

**Possible Successors**

CloseMeasure

**See also**

FuzzyMeasurePos, MeasurePos

**Alternatives**

EdgesSubPix, FuzzyMeasurePairs, MeasurePairs

---

**Module**

Sub-pixel operators
Extract straight edge pairs perpendicular to a rectangle or an annular arc.

FuzzyMeasurePairs serves to extract straight edge pairs which lie perpendicular to the major axis of a rectangle or an annular arc. In addition to MeasurePairs it uses fuzzy member functions to evaluate and select the edge pairs.

The extraction algorithm is identical to FuzzyMeasurePos. In addition, the edges are grouped to pairs: If Transition = 'positive', the edge points with a dark-to-light transition in the direction of the major axis of the rectangle or annular arc are returned in RowEdgeFirst and ColumnEdgeFirst. In this case, the corresponding edges with a light-to-dark transition are returned in RowEdgeSecond and ColumnEdgeSecond. If Transition = 'negative', the behavior is exactly opposite. If Transition = 'all', the first detected edge defines the transition for RowEdgeFirst and ColumnEdgeFirst.

Having extracted subpixel edge locations, the edges are paired. The pairing algorithm groups the edges such that interleavings and inclusions of pairs are prohibited. The features of an edge pair are evaluated by a fuzzy function, which can be set by SetFuzzyMeasure or SetFuzzyMeasureNormPair. Which edge pairs are selected can be determined with the parameter FuzzyThresh, which constitutes a threshold on the weight over all fuzzy sets, i.e., the geometric mean of the weights of the defined fuzzy member functions.

The selected edges are returned as single points, which lie on the major axis of the rectangle or annular arc. The corresponding edge amplitudes are returned in AmplitudeFirst and AmplitudeSecond, the fuzzy scores in FuzzyScore. In addition, the distance between each edge pair is returned in IntraDistance and the distance between consecutive edge pairs is returned in InterDistance. Here, IntraDistance[i] corresponds to the distance between EdgeFirst[i] and EdgeSecond[i], while InterDistance[i] corresponds to the distance between EdgeSecond[i] and EdgeFirst[i+1], i.e., the tuple InterDistance contains one element less than the tuples of the edge pairs.

Attention

FuzzyMeasurePairs only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle or annular arc are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle or an annular arc is as close to perpendicular as possible to the edges in the image.

It should be kept in mind that FuzzyMeasurePairs ignores the domain of Image for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.

HALCON/COM Reference Manual, 2005-2-1
13.11. MEASURE

Parameter

- **Image** (input iconic)  
  Image  \( \sim \) HImageX / HObjectX (byte, uint2)  
  Input image.

- **MeasureHandle** (input control)  
  Measure object handle.

- **Sigma** (input control)  
  Sigma of Gaussian smoothing.  
  Default Value: 1.0  
  Suggested values: \( \Sigma \in \{0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0\} \)  
  Typical range of values: 0.4 \( \leq \Sigma \leq 0.4 \) (lin)  
  Minimum Increment: 0.01  
  Recommended Increment: 0.1  
  Restriction: \( (\Sigma \geq 0.4) \)

- **AmpThresh** (input control)  
  Minimum edge amplitude.  
  Default Value: 30.0  
  Suggested values: \( \text{AmpThresh} \in \{5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0\} \)  
  Typical range of values: 1 \( \leq \text{AmpThresh} \leq 1 \) (lin)  
  Minimum Increment: 0.5  
  Recommended Increment: 2

- **FuzzyThresh** (input control)  
  Minimum fuzzy value.  
  Default Value: 0.5  
  Suggested values: \( \text{FuzzyThresh} \in \{0.1, 0.3, 0.5, 0.7, 0.9\} \)  
  Typical range of values: 0.0 \( \leq \text{FuzzyThresh} \leq 0.0 \) (lin)  
  Recommended Increment: 0.1

- **Transition** (input control)  
  Select the first gray value transition of the edge pairs.  
  Default Value: 'all'  
  List of values: Transition \( \in \{\text{'all'}, \text{'positive'}, \text{'negative'}\} \)

- **RowEdgeFirst** (output control)  
  Row coordinate of the first edge point.

- **ColumnEdgeFirst** (output control)  
  Column coordinate of the first edge point.

- **AmplitudeFirst** (output control)  
  Edge amplitude of the first edge (with sign).

- **RowEdgeSecond** (output control)  
  Row coordinate of the second edge point.

- **ColumnEdgeSecond** (output control)  
  Column coordinate of the second edge point.

- **AmplitudeSecond** (output control)  
  Edge amplitude of the second edge (with sign).

- **RowEdgeCenter** (output control)  
  Row coordinate of the center of the edge pair.

- **ColumnEdgeCenter** (output control)  
  Column coordinate of the center of the edge pair.

- **FuzzyScore** (output control)  
  Fuzzy evaluation of the edge pair.

- **IntraDistance** (output control)  
  Distance between edges of an edge pair.

- **InterDistance** (output control)  
  Distance between consecutive edge pairs.

Result

If the parameter values are correct the operator FuzzyMeasurePairs returns the value TRUE. Otherwise an exception handling is raised.
**Parallelization Information**

**FuzzyMeasurePairs** is reentrant and processed without parallelization.

**Possible Predecessors**

GenMeasureRectangle2, GenMeasureArc, SetFuzzyMeasure

**Possible Successors**

CloseMeasure

**See also**

FuzzyMeasurePos, MeasurePos

**Alternatives**

EdgesSubPix, FuzzyMeasurePairing, MeasurePairs

**Module**

Sub-pixel operators

```plaintext
```

```plaintext
```

```plaintext
```

Extract straight edges perpendicular to a rectangle or an annular arc.

**FuzzyMeasurePos** extracts straight edges which lie perpendicular to the major axis of a rectangle or an annular arc. In addition to **MeasurePos** it uses fuzzy member functions to evaluate and select the edges.

The algorithm works by averaging the gray values in “slices” perpendicular to the major axis of the rectangle or annular arc in order to obtain a one-dimensional edge profile. The sampling is done at subpixel positions in the image **Image** at integer row and column distances (in the coordinate frame of the rectangle) from the center of the rectangle. Since this involves some calculations which can be used repeatedly in several measurements, the operator **GenMeasureRectangle2** is used to perform these calculations only once, thus increasing the speed of **FuzzyMeasurePos** significantly. Since there is a trade-off between accuracy and speed in the subpixel calculations of the gray values, and thus in the accuracy of the extracted edge positions, different interpolation schemes can be selected in **GenMeasureRectangle2**. (The interpolation only influences rectangles not aligned with the image axes and annular arcs.) The measure object generated with **GenMeasureRectangle2** is passed in **MeasureHandle**.

After the one-dimensional edge profile has been calculated, subpixel edge locations are computed by convolving the profile with the derivatives of a Gaussian smoothing kernel of standard deviation **Sigma**. Salient edges can be selected with the parameter **AmpThresh**, which constitutes a threshold on the amplitude, i.e., the absolute value of the first derivative of the edge. Additionally, it is possible to select only positive edges, i.e., edges which constitute a dark-to-light transition in the direction of the major axis of the rectangle (**Transition = 'positive'**), only negative edges, i.e., light-to-dark transitions (**Transition = 'negative'**), or both types of edges (**Transition = 'all'**). Finally, it is possible to select which edge points are returned.

Having extracted subpixel edge locations, features of these edges are evaluated by a corresponding fuzzy function, which can be set by **SetFuzzyMeasure**. Which edges are selected can be determined with the parameter **FuzzyThresh**, which constitutes a threshold on the weight over all fuzzy sets, i.e., the geometric mean of the weights of the defined sets.

The selected edges are returned as single points, which lie on the major axis of the rectangle or annular arc, in (RowEdge, ColumnEdge). The corresponding edge amplitudes are returned in **Amplitude**, the fuzzy scores.
in **FuzzyScore**. In addition, the distance between consecutive edge points is returned in **Distance**. Here, **Distance[i]** corresponds to the distance between **Edge[i]** and **Edge[i+1]**, i.e., the tuple **Distance** contains one element less than the tuples **RowEdge** and **ColumnEdge**.

**Attention**

**FuzzyMeasurePos** only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle is as close to perpendicular as possible to the edges in the image.

It should be kept in mind that **FuzzyMeasurePos** ignores the domain of **Image** for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong></td>
<td>(input iconic) Input image.</td>
</tr>
<tr>
<td><strong>MeasureHandle</strong></td>
<td>(input control) Measure object handle.</td>
</tr>
<tr>
<td><strong>Sigma</strong></td>
<td>(input control) Sigma of Gaussian smoothing. Default Value: 1.0 Suggested values: Sigma ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0} Typical range of values: 0.4 ≤ Sigma ≤ 0.4(lin) Minimum Increment: 0.01 Recommended Increment: 0.1 Restriction: (Sigma ≥ 0.4)</td>
</tr>
<tr>
<td><strong>AmpThresh</strong></td>
<td>(input control) Minimum edge amplitude. Default Value: 30.0 Suggested values: AmpThresh ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0} Typical range of values: 1 ≤ AmpThresh ≤ 1(lin) Minimum Increment: 2 Recommended Increment: 0.5</td>
</tr>
<tr>
<td><strong>FuzzyThresh</strong></td>
<td>(input control) Minimum fuzzy value. Default Value: 0.5 Suggested values: FuzzyThresh ∈ {0.1, 0.3, 0.5, 0.6, 0.7, 0.9} Typical range of values: 0.0 ≤ FuzzyThresh ≤ 0.0(lin) Recommended Increment: 0.1</td>
</tr>
<tr>
<td><strong>Transition</strong></td>
<td>(input control) Select light/dark or dark/light edges. Default Value: 'all' List of values: Transition ∈ {'all', 'positive', 'negative'}</td>
</tr>
<tr>
<td><strong>RowEdge</strong></td>
<td>(output control) Row coordinate of the edge point.</td>
</tr>
<tr>
<td><strong>ColumnEdge</strong></td>
<td>(output control) Column coordinate of the edge point.</td>
</tr>
<tr>
<td><strong>Amplitude</strong></td>
<td>(output control) Edge amplitude of the edge (with sign).</td>
</tr>
<tr>
<td><strong>FuzzyScore</strong></td>
<td>(output control) Fuzzy evaluation of the edges.</td>
</tr>
<tr>
<td><strong>Distance</strong></td>
<td>(output control) Distance between consecutive edges.</td>
</tr>
</tbody>
</table>

**Result**

If the parameter values are correct the operator **FuzzyMeasurePos** returns the value TRUE. Otherwise an exception handling is raised.

**Parallelization Information**

**FuzzyMeasurePos** is reentrant and processed without parallelization.
Possible Predecessors
GenMeasureRectangle2, GenMeasureArc, SetFuzzyMeasure

Possible Successors
CloseMeasure

See also
FuzzyMeasurePairing, FuzzyMeasurePairs, MeasurePairs

Alternatives
EdgesSubPix, MeasurePos

Module
Sub-pixel operators

void HMeasureX.GenMeasureArc ([in] VARIANT CenterRow,
in] long Height, [in] String Interpolation )

void HOperatorSetX.GenMeasureArc ([in] VARIANT CenterRow,
in] VARIANT CenterCol, [in] VARIANT Radius, [in] VARIANT AngleStart,
in] VARIANT Height, [in] VARIANT Interpolation,
[out] VARIANT MeasureHandle )

Prepare the extraction of straight edges perpendicular to an annular arc.

GenMeasureArc prepares the extraction of straight edges which lie perpendicular to an annular arc. Here, annular arc denotes a circular arc with an associated width. The center of the arc is passed in the parameters CenterRow and CenterCol, its radius in Radius, the starting angle in AngleStart, and its angular extent relative to the starting angle in AngleExtent. If AngleExtent > 0, an arc with counterclockwise orientation is generated, otherwise an arc with clockwise orientation. The radius of the annular arc, i.e., half its width, is determined by AnnulusRadius.

The edge extraction algorithm is described in the documentation of the operator MeasurePos. As discussed there, different types of interpolation can be used for the calculation of the one-dimensional gray value profile. For Interpolation = 'nearest neighbor', the gray values in the measurement are obtained from the gray values of the closest pixel, i.e., by constant interpolation. For Interpolation = 'bilinear', bilinear interpolation is used, while for Interpolation = 'bicubic', bicubic interpolation is used.

With GenMeasureArc, all computations which can be used for multiple measurements are removed from the actual measurement. To effect this, an optimized data structure, a so-called measure object, is constructed and returned in MeasureHandle. In order to perform the measurement at the optimum speed, the size of the images for which subsequent measurements will be made must already be specified in GenMeasureArc with the parameters Width and Height. This technique increases the speed of the actual measurement significantly.

The system parameter 'intZooming' (see SetSystem) affects the accuracy and speed of the calculations used to construct the measure object. If 'intZooming' is set to 'true', the internal calculations are performed using fixed point arithmetic, leading to much shorter execution times. However, the geometric accuracy is slightly lower in this mode. If 'intZooming' is set to 'false', the internal calculations are performed using floating point arithmetic, leading to the maximum geometric accuracy, but also to significantly increased execution times.

Parameter

> CenterRow (input control) ............................... point.y ⇒ VARIANT( integer, real )
Row coordinate of the center of the arc.

Default Value : 100.0
Suggested values : CenterRow ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
 Typical range of values : 0.0 ≤ CenterRow ≤ 0.0(lin)
Minimum Increment : 1.0
Recommended Increment : 10.0
13.11. MEASURE

- **CenterCol** (input control) ........................................... point.x  ~ VARIANT (integer, real)
  Column coordinate of the center of the arc.
  Default Value: 100.0
  Suggested values: CenterCol ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typical range of values: 0.0 ≤ CenterCol ≤ 0.0 (lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0

- **Radius** (input control) ............................................ number  ~ VARIANT (integer, real)
  Radius of the arc.
  Default Value: 50.0
  Suggested values: Radius ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typical range of values: 0.0 ≤ Radius ≤ 0.0 (lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0

- **AngleStart** (input control) ..................................... angle.rad  ~ VARIANT (integer, real)
  Start angle of the arc in radians.
  Default Value: 0.0
  Suggested values: AngleStart ∈ {-3.14159265359, -2.35619449019, -1.5707963268, -0.785398163398, 0.0, 0.785398163398, 1.5707963268, 2.35619449019, 3.14159265359}
  Typical range of values: -3.14159265359 ≤ AngleStart ≤ -3.14159265359 (lin)
  Minimum Increment: 0.0314159265359
  Recommended Increment: 0.314159265359
  Restriction: (AngleExtent ≠ 0.0)

- **AngleExtent** (input control) ................................. angle.rad  ~ VARIANT (integer, real)
  Angular extent of the arc in radians.
  Default Value: 6.28318530718
  Suggested values: AngleExtent ∈ {-6.28318530718, -5.49778714378, -4.71238898038, -3.926990817, -3.14159265359, -2.35619449019, -1.5707963268, -0.785398163398, 0.785398163398, 1.5707963268, 2.35619449019, 3.14159265359, 3.926990817, 4.71238898038, 5.49778714378, 6.28318530718}
  Typical range of values: -6.28318530718 ≤ AngleExtent ≤ -6.28318530718 (lin)
  Minimum Increment: 0.0314159265359
  Recommended Increment: 0.314159265359
  Restriction: (AngleExtent ≠ 0.0)

- **AnnulusRadius** (input control) .............................. number  ~ VARIANT (integer, real)
  Radius (half width) of the annulus.
  Default Value: 10.0
  Suggested values: AnnulusRadius ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typical range of values: 0.0 ≤ AnnulusRadius ≤ 0.0 (lin)
  Minimum Increment: 1.0
  Recommended Increment: 10.0
  Restriction: (AnnulusRadius ≤ Radius)

- **Width** (input control) ............................................. extent.x  ~ long / VARIANT
  Width of the image to be processed subsequently.
  Default Value: 512
  Suggested values: Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
  Typical range of values: 0 ≤ Width ≤ 0 (lin)
  Minimum Increment: 1
  Recommended Increment: 16

- **Height** (input control) ......................................... extent.y  ~ long / VARIANT
  Height of the image to be processed subsequently.
  Default Value: 512
  Suggested values: Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}
  Typical range of values: 0 ≤ Height ≤ 0 (lin)
  Minimum Increment: 1
  Recommended Increment: 16

- **Interpolation** (input control) ................................. string  ~ String / VARIANT
  Type of interpolation to be used.
  Default Value: 'nearest_neighbor'
  List of values: Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}
Measure object handle.

result

If the parameter values are correct, the operator \texttt{GenMeasureArc} returns the value TRUE. Otherwise an exception handling is raised.

\textbf{Parallelization Information}

\texttt{GenMeasureArc} is reentrant and processed without parallelization.

\textbf{Possible Predecessors}

\texttt{DrawCircle}

\textbf{Possible Successors}

\texttt{MeasurePos, MeasurePairs, FuzzyMeasurePos, FuzzyMeasurePairs, FuzzyMeasurePairing}

\textbf{See also}

\texttt{GenMeasureRectangle2}

\textbf{Alternatives}

\texttt{EdgesSubPix}

\textbf{Module}

Sub-pixel operators

\begin{verbatim}
void \texttt{HMeasureX.GenMeasureRectangle2} ([in] VARIANT Row,
|in] VARIANT Column, [in] VARIANT Phi, [in] VARIANT Length1,
|in] VARIANT Length2, [in] long Width, [in] long Height,
|in] String Interpolation )

void \texttt{HOperatorSetX.GenMeasureRectangle2} ([in] VARIANT Row,  
|in] VARIANT Column, [in] VARIANT Phi, [in] VARIANT Length1,  
|in] VARIANT Length2, [in] VARIANT Width, [in] VARIANT Height,  
|in] VARIANT Interpolation, [out] VARIANT MeasureHandle )
\end{verbatim}

Prepare the extraction of straight edges perpendicular to a rectangle.

\texttt{GenMeasureRectangle2} prepares the extraction of straight edges which lie perpendicular to the major axis of a rectangle. The center of the rectangle is passed in the parameters \texttt{Row} and \texttt{Column}, the direction of the major axis of the rectangle in \texttt{Phi}, and the length of the two axes, i.e., half the diameter of the rectangle, in \texttt{Length1} and \texttt{Length2}.

The edge extraction algorithm is described in the documentation of the operator \texttt{MeasurePos}. As discussed there, different types of interpolation can be used for the calculation of the one-dimensional gray value profile. For \texttt{Interpolation = 'nearest neighbor'}, the gray values in the measurement are obtained from the gray values of the closest pixel, i.e., by constant interpolation. For \texttt{Interpolation = 'bilinear'}, bilinear interpolation is used, while for \texttt{Interpolation = 'bicubic'}, bicubic interpolation is used.

With \texttt{GenMeasureRectangle2}, all computations which can be used for multiple measurements are removed from the actual measurement. To effect this, an optimized data structure, a so-called measure object, is constructed and returned in \texttt{MeasureHandle}. In order to perform the measurement at the optimum speed, the size of the images for which subsequent measurements will be made must already be specified in \texttt{GenMeasureRectangle2} with the parameters \texttt{Width} and \texttt{Height}. This technique increases the speed of the actual measurement significantly.

The system parameter \texttt{'int zooming'} (see \texttt{SetSystem}) affects the accuracy and speed of the calculations used to construct the measure object. If \texttt{'int zooming'} is set to \texttt{'true'}, the internal calculations are performed using fixed point arithmetic, leading to much shorter execution times. However, the geometric accuracy is slightly lower in this mode. If \texttt{'int zooming'} is set to \texttt{'false'}, the internal calculations are performed using floating point arithmetic, leading to the maximum geometric accuracy, but also to significantly increased execution times.
Parameter

- **Row** (input control) .......................... rectangle2.center.y  ~ V A R I A N T ( integer, real )
  Row coordinate of the center of the rectangle.
  Default Value : 50.0
  Suggested values : Row ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typical range of values : 0.0 ≤ Row ≤ 0.0
  Minimum Increment : 1.0
  Recommended Increment : 10.0

- **Column** (input control) ..................... rectangle2.center.x  ~ V A R I A N T ( integer, real )
  Column coordinate of the center of the rectangle.
  Default Value : 100.0
  Suggested values : Column ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typical range of values : 0.0 ≤ Column ≤ 0.0
  Minimum Increment : 1.0
  Recommended Increment : 10.0

- **Phi** (input control) .......................... rectangle2.angle.rad  ~ V A R I A N T ( integer, real )
  Angle of longitudinal axis of the rectangle to horizontal (radians).
  Default Value : 0.0
  Suggested values : Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
  Typical range of values : -1.178097 ≤ Phi ≤ -1.178097
  Minimum Increment : 0.001
  Recommended Increment : 0.1
  Restriction : (-pi < Phi) ∧ (Phi ≤ pi)

- **Length1** (input control) ...................... rectangle2.hwidth  ~ V A R I A N T ( integer, real )
  Half width of the rectangle.
  Default Value : 200.0
  Suggested values : Length1 ∈ {3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0}
  Typical range of values : 0.0 ≤ Length1 ≤ 0.0
  Minimum Increment : 1.0
  Recommended Increment : 10.0

- **Length2** (input control) ...................... rectangle2.hheight  ~ V A R I A N T ( integer, real )
  Half height of the rectangle.
  Default Value : 100.0
  Suggested values : Length2 ∈ {1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0}
  Typical range of values : 0.0 ≤ Length2 ≤ 0.0
  Minimum Increment : 1.0
  Recommended Increment : 10.0

- **Width** (input control) .......................... extent.x  ~ long / V A R I A N T
  Width of the image to be processed subsequently.
  Default Value : 512
  Suggested values : Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
  Typical range of values : 0 ≤ Width ≤ 0
  Minimum Increment : 1
  Recommended Increment : 16

- **Height** (input control) ........................... extent.y  ~ long / V A R I A N T
  Height of the image to be processed subsequently.
  Default Value : 512
  Suggested values : Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}
  Typical range of values : 0 ≤ Height ≤ 0
  Minimum Increment : 1
  Recommended Increment : 16

- **Interpolation** (input control) ................. string  ~ String / V A R I A N T
  Type of interpolation to be used.
  Default Value : 'nearest_neighbor'
  List of values : Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

- **MeasureHandle** (output control) ............... measure_id  ~ HMeasureX / V A R I A N T
  Measure object handle.
GenMeasureRectangle2 serves to extract Transition is, e.g., is set to RowEdgeSecond ColumnEdgeFirst = Transition Transition and . In this case, the corresponding edges RowEdgeFirst returns the value TRUE. Otherwise . If cannot be selected high enough to suppress consecutive edge s of the same transition. For strongest’ RowEdgeFirst = strongest’ and . In addition the edges are grouped to pairs: If positive’ , the behavior is exactly opposite. If negative’ , the first detected edge defines the transition for RowEdgeFirst and ColumnEdgeFirst. If more than one consecutive edge with the same transition i s found, the first one is used as a pair element. This behavior ma y cause problems in applications in which the 'all', only the first of the extracted edge pairs is returned, while it is set to 'last', only the last one is returned.

The extraction algorithm is identical to MeasurePos. In addition the edges are grouped to pairs: If Transition = positive’, the edge points with a dark-to-light transition in the direction of the major axis of the rectangle are returned in RowEdgeFirst and ColumnEdgeFirst. In this case, the corresponding edges with a light-to-dark transition are returned in RowEdgeSecond and ColumnEdgeSecond. If Transition = negative’, the behavior is exactly opposite. If Transition = all’, the first detected edge defines the transition for RowEdgeFirst and ColumnEdgeFirst. If more than one consecutive edge with the same transition is found, the first one is used as a pair element. This behavior may cause problems in applications in which the threshold Threshold cannot be selected high enough to suppress consecutive edges of the same transition. For these applications, a second pairing mode exists that only selects the respective strongest edges of a sequence of consecutive rising and falling edges. This mode is selected by appending strongest’ to any of the above modes for Transition, e.g., negative strongest’. Finally, it is possible to select which edge pairs are returned. If Select is set to all’, all edge pairs are returned. If it is set to first’, only the first of the extracted edge pairs is returned, while it is set to last’, only the last one is returned.

Extract straight edge pairs perpendicular to a rectangle or annular arc.

MeasurePairs serves to extract straight edge pairs which lie perpendicular to the major axis of a rectangle or annular arc.

The extraction algorithm is identical to MeasurePos. In addition the edges are grouped to pairs: If Transition = positive’, the edge points with a dark-to-light transition in the direction of the major axis of the rectangle are returned in RowEdgeFirst and ColumnEdgeFirst. In this case, the corresponding edges with a light-to-dark transition are returned in RowEdgeSecond and ColumnEdgeSecond. If Transition = negative’, the behavior is exactly opposite. If Transition = all’, the first detected edge defines the transition for RowEdgeFirst and ColumnEdgeFirst. If more than one consecutive edge with the same transition is found, the first one is used as a pair element. This behavior may cause problems in applications in which the threshold Threshold cannot be selected high enough to suppress consecutive edges of the same transition. For these applications, a second pairing mode exists that only selects the respective strongest edges of a sequence of consecutive rising and falling edges. This mode is selected by appending strongest’ to any of the above modes for Transition, e.g., negative strongest’. Finally, it is possible to select which edge pairs are returned. If Select is set to all’, all edge pairs are returned. If it is set to first’, only the first of the extracted edge pairs is returned, while it is set to last’, only the last one is returned.
The extracted edges are returned as single points which lie on the major axis of the rectangle. The corresponding edge amplitudes are returned in `AmplitudeFirst` and `AmplitudeSecond`. In addition, the distance between each edge pair is returned in `IntraDistance` and the distance between consecutive edge pairs is returned in `InterDistance`. Here, `IntraDistance[i]` corresponds to the distance between `EdgeFirst[i]` and `EdgeSecond[i]`, while `InterDistance[i]` corresponds to the distance between `EdgeSecond[i]` and `EdgeFirst[i+1]`, i.e., the tuple `InterDistance` contains one element less than the tuples of the edge pairs.

**Attention**

`MeasurePairs` only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle is as close to perpendicular as possible to the edges in the image.

It should be kept in mind that `MeasurePairs` ignores the domain of `Image` for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.

### Parameter

- **Image** (input iconic)  
  Input image.

- **MeasureHandle** (input control)  
  Measure object handle.

- **Sigma** (input control)  
  Sigma of gaussian smoothing.

  **Default Value**: 1.0

  **Suggested values**: `Sigma ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}`

  **Typical range of values**: `0.4 ≤ Sigma ≤ 0.4(lin)`

  **Minimum Increment**: 0.01

  **Recommended Increment**: 0.1

  **Restriction**: `(Sigma ≥ 0.4)`

- **Threshold** (input control)  
  Minimum edge amplitude.

  **Default Value**: 30.0

  **Suggested values**: `Threshold ∈ {5.0, 10.0, 20.0, 30.0, 50.0, 60.0, 70.0, 90.0, 110.0}`

  **Typical range of values**: `1 ≤ Threshold ≤ 1(lin)`

  **Minimum Increment**: 0.5

  **Recommended Increment**: 2

- **Transition** (input control)  
  Type of gray value transition that determines how edges are grouped to edge pairs.

  **Default Value**: `'all'`

  **List of values**: `Transition ∈ {'all', 'positive', 'negative', 'all_strongest', 'positive_strongest', 'negative_strongest'}`

- **Select** (input control)  
  Selection of edge pairs.

  **Default Value**: `'all'`

  **List of values**: `Select ∈ {'all', 'first', 'last'}`

- **RowEdgeFirst** (output control)  
  Row coordinate of the center of the first edge.

- **ColumnEdgeFirst** (output control)  
  Column coordinate of the center of the first edge.

- **AmplitudeFirst** (output control)  
  Edge amplitude of the first edge (with sign).

- **RowEdgeSecond** (output control)  
  Row coordinate of the center of the second edge.

- **ColumnEdgeSecond** (output control)  
  Column coordinate of the center of the second edge.

- **AmplitudeSecond** (output control)  
  Edge amplitude of the second edge (with sign).
MeasurePairs (output control) ............................................real  ~ VARIANT( real )
Distance between edges of an edge pair.

InterDistance (output control) ............................................real  ~ VARIANT( real )
Distance between consecutive edge pairs.

If the parameter values are correct the operator MeasurePairs returns the value TRUE. Otherwise an exception handling is raised.

Extract straight edges perpendicular to a rectangle or annular arc.

MeasurePos extracts straight edges which lie perpendicular to the major axis of a rectangle or annular arc. The algorithm works by averaging the gray values in “slices” perpendicular to the major axis of the rectangle or annular arc in order to obtain a one-dimensional edge profile. The sampling is done at subpixel positions in the image Image at integer row and column distances (in the coordinate frame of the rectangle) from the center of the rectangle. Since this involves some calculations which can be used repeatedly in several mesurements, the operator GenMeasureRectangle2 or GenMeasureArc is used to perform these calculations only once, thus increasing the speed of MeasurePos significantly. Since there is a trade-off between accuracy and speed in the subpixel calculations of the gray values, and thus in the accuracy of the extracted edge positions, different interpolation schemes can be selected in GenMeasureRectangle2. (The interpolation only influences rectangles not aligned with the image axes.) The measure object generated with GenMeasureRectangle2 is passed in MeasureHandle.

After the one-dimensional edge profile has been calculated, subpixel edge locations are computed by convolving the profile with the derivatives of a Gaussian smoothing kernel of standard deviation Sigma. Salient edges can be selected with the parameter Threshold, which constitutes a threshold on the amplitude, i.e., the absolute value of the first derivative of the edge. Additionally, it is possible to select only positive edges, i.e., edges which constitute a dark-to-light transition in the direction of the major axis of the rectangle or the arc (Transition = 'positive'), only negative edges, i.e., light-to-dark transitions (Transition = 'negative'), or both types of edges (Transition = 'all'). Finally, it is possible to select which edge points are returned. If Select is set to 'all',...
all edge points are returned. If it is set to 'first', only the first of the extracted edge points is returned, while it is set to 'last', only the last one is returned.

The extracted edges are returned as single points which lie on the major axis of the rectangle or arc in (RowEdge,ColumnEdge). The corresponding edge amplitudes are returned in Amplitude. In addition, the distance between consecutive edge points is returned in Distance. Here, Distance[i] corresponds to the distance between Edge[i] and Edge[i+1], i.e., the tuple Distance contains one element less than the tuples RowEdge and ColumnEdge.

Attention
MeasurePos only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle or arc are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle or arc is as close to perpendicular as possible to the edges in the image.

It should be kept in mind that MeasurePos ignores the domain of Image for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.

Parameter

- **Image** (input iconic) ............................ image  \(\sim\) HImageX / IHObjectX (byte, uint2)
  Input image.

- **MeasureHandle** (input control) ........................ measurejd  \(\sim\) long / HMeasureX / VARIANT
  Measure object handle.

- **Sigma** (input control) ............................. number  \(\sim\) double / VARIANT
  Sigma of gaussian smoothing.
  Default Value : 1.0
  Suggested values : Sigma \(\in\) \{0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0\}
  Typical range of values : 0.4 \(\leq\) Sigma \(\leq\) 0.4(lin)
  Minimum Increment : 0.01
  Recommended Increment : 0.1
  Restriction : (Sigma \(\geq\) 0.4)

- **Threshold** (input control) .......................... number  \(\sim\) double / VARIANT
  Minimum edge amplitude.
  Default Value : 30.0
  Suggested values : Threshold \(\in\) \{5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0\}
  Typical range of values : 1 \(\leq\) Threshold \(\leq\) 1(lin)
  Minimum Increment : 2
  Recommended Increment : 0.5

- **Transition** (input control) ........................ string  \(\sim\) String / VARIANT
  Light/dark or dark/light edge.
  Default Value : 'all'
  List of values : Transition \(\in\) \{'all', 'positive', 'negative'\}

- **Select** (input control) ............................ string  \(\sim\) String / VARIANT
  Selection of end points.
  Default Value : 'all'
  List of values : Select \(\in\) \{'all', 'first', 'last'\}

- **RowEdge** (output control) .......................... point.y  \(\sim\) VARIANT (real)
  Row coordinate of the center of the edge.

- **ColumnEdge** (output control) ......................... point.x  \(\sim\) VARIANT (real)
  Column coordinate of the center of the edge.

- **Amplitude** (output control) ........................ real  \(\sim\) VARIANT (real)
  Edge amplitude of the edge (with sign).

- **Distance** (output control) .......................... real  \(\sim\) VARIANT (real)
  Distance between consecutive edges.

Result
If the parameter values are correct the operator MeasurePos returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
MeasurePos is reentrant and processed without parallelization.
CHAPTER 13. TOOLS

Possible Predecessors

GenMeasureRectangle2

Possible Successors

CloseMeasure

See also

MeasurePairs, FuzzyMeasurePairs, FuzzyMeasurePairing

Alternatives

EdgesSubPix, FuzzyMeasurePos

Sub-pixel operators

Extract a gray value profile perpendicular to a rectangle or annular arc.

\textbf{MeasureProjection} extracts a one-dimensional gray value profile perpendicular to a rectangle or annular arc. This is done by averaging the gray values in “slices” perpendicular to the major axis of the rectangle or arc. The sampling is done at subpixel positions in the image \textit{Image} at integer row and column distances (in the coordinate frame of the rectangle) from the center of the rectangle. Since this involves some calculations which can be used repeatedly in several projections, the operator \textit{GenMeasureRectangle2} is used to perform these calculations only once, thus increasing the speed of \textit{MeasureProjection} significantly. Since there is a trade-off between accuracy and speed in the subpixel calculations of the gray values, different interpolation schemes can be selected in \textit{GenMeasureRectangle2} (the interpolation only influences rectangles not aligned with the image axes). The measure object generated with \textit{GenMeasureRectangle2} is passed in \textit{MeasureHandle}.

\begin{itemize}
  \item [\textbf{Attention}] It should be kept in mind that \textit{MeasureProjection} ignores the domain of \textit{Image} for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.
\end{itemize}

\begin{itemize}
  \item [\textbf{Parameter}] \begin{itemize}
    \item [\textbf{Image} (input iconic)] image \sim \textit{HImageX / IOObjectX (byte, uint2)}
    Input image.
    \item [\textbf{MeasureHandle} (input control)] measure jd \sim \textit{long / HMeasureX / VARIANT}
    Measure object handle.
    \item [\textbf{GrayValues} (output control)] number \sim \textit{VARIANT (real)}
    Gray value profile.
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item [\textbf{Result}] If the parameter values are correct the operator \textit{MeasureProjection} returns the value TRUE. Otherwise an exception handling is raised.
\end{itemize}

\begin{itemize}
  \item [\textbf{Parallelization Information}] \textit{MeasureProjection} is \textit{reentrant} and processed \textit{without} parallelization.
\end{itemize}
MeasureThresh extracts points for which the gray value within an one-dimensional gray value profile is equal to the specified threshold Threshold. The gray value profile is projected onto the major axis of the measure rectangle which is passed with the parameter MeasureHandle, so the threshold points calculated within the gray value profile correspond to certain image coordinates on the rectangle’s major axis. These coordinates are returned as the operator results in RowThresh and ColumnThresh.

If the gray value profile intersects the threshold line for several times, the parameter Select determines which values to return. Possible settings are 'first', 'last', 'first last' (first and last) or 'all'. For the last two cases Distance returns the distances between the calculated points.

The gray value profile is created by averaging the gray values along all line segments, which are defined by the measure rectangle as follows:

1. The segments are perpendicular to the major axis of the rectangle,
2. they have an integer distance to the center of the rectangle,
3. the rectangle bounds the segments.

For every line segment, the average of the gray values of all points with an integer distance to the major axis is calculated. Due to translation and rotation of the measure rectangle with respect to the image coordinates the input image Image is in general sampled at subpixel positions.

Since this involves some calculations which can be used repeatedly in several projections, the operator GenMeasureRectangle2 is used to perform these calculations only once in advance. Here, the measure object MeasureHandle is generated and different interpolation schemes can be selected.

Attention MeasureThresh only returns meaningful results if the assumptions that the edges are straight and perpendicular to the major axis of the rectangle are fulfilled. Thus, it should not be used to extract edges from curved objects, for example. Furthermore, the user should ensure that the rectangle is as close to perpendicular as possible to the edges in the image.

It should be kept in mind that MeasureThresh ignores the domain of Image for efficiency reasons. If certain regions in the image should be excluded from the measurement a new measure object with appropriately modified parameters should be generated.

Parameters

- **Image** (input iconic) 
  - image \( \sim \) HImageX / IObjectX (byte, uint2)
  - Input image.
- **MeasureHandle** (input control) 
  - measure \( \sim \) long / HMeasureX / VARIANT
  - Measure object handle.
- **Sigma** (input control) 
  - number \( \sim \) double / VARIANT
  - Sigma of gaussian smoothing.

  Default Value: 1.0
  Suggested values: \( \Sigma \in \{0.0, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0\} \)
  Typical range of values: \( 0.4 \leq \Sigma \leq 0.4 \text{(lin)} \)
  Minimum Increment: 0.01
  Recommended Increment: 0.1
  Restriction: \( (\Sigma \geq 0.0) \)

HALCON 6.1.4
CHAPTER 13. TOOLS

- **Threshold** (input control) .................................................. number \(\sim\) double / VARIANT
  - Default Value: 128.0
  - Typical range of values: \(0 \leq \text{Threshold} \leq 0\) (lin)
  - Minimum Increment: 1
  - Recommended Increment: 0.5

- **Select** (input control) .................................................. string \(\sim\) String / VARIANT
  - Selection of points.
  - Default Value: 'all'
  - List of values: Select \(\in\) \{'all', 'first', 'last', 'first_last'\}

- **RowThresh** (output control) ............................................. point.y \(\sim\) VARIANT (real)
  - Row coordinates of points with threshold value.

- **ColumnThresh** (output control) ......................................... point.x \(\sim\) VARIANT (real)
  - Column coordinates of points with threshold value.

- **Distance** (output control) .............................................. real \(\sim\) VARIANT (real)
  - Distance between consecutive points.

If the parameter values are correct the operator MeasureThresh returns the value TRUE. Otherwise, an exception handling is raised.

---

**Parallelization Information**

MeasureThresh is reentrant and processed without parallelization.

---

**Possible Predecessors**

GenMeasureRectangle2

---

**Possible Successors**

CloseMeasure

---

**Alternatives**

MeasurePos, EdgesSubPix, MeasurePairs

---

**Module**

Sub-pixel operators

---

```cpp
void HMeasureX.ResetFuzzyMeasure ([in] String SetType )

void HOperatorSetX.ResetFuzzyMeasure ([in] VARIANT MeasureHandle, [in] VARIANT SetType )
```

Reset a fuzzy member function.

ResetFuzzyMeasure discards a fuzzy member function of the fuzzy set SetType. This member function should have been set by SetFuzzyMeasure before.

---

**Parameter**

- **MeasureHandle** (input control) ............................... measure jd \(\sim\) HMeasureX / VARIANT
  - Measure object handle.

- **SetType** (input control) ............................................. string \(\sim\) String / VARIANT
  - Selection of the fuzzy set.
  - Default Value: 'contrast'
  - List of values: SetType \(\in\) \{'position', 'position_pair', 'size', 'gray', 'contrast'\}

---

**Parallelization Information**

ResetFuzzyMeasure is reentrant and processed without parallelization.

---

**Possible Predecessors**

SetFuzzyMeasure

---

**Possible Successors**

FuzzyMeasurePos, FuzzyMeasurePairs

---

SetFuzzyMeasure, SetFuzzyMeasureNormPair

---

See also

HALCON/COM Reference Manual, 2005-2-1
Module Sub-pixel operators

```c
void HMeasureX.SetFuzzyMeasure ([in] String SetType, [in] HFunction1dX Function )
void HOperatorSetX.SetFuzzyMeasure ([in] VARIANT MeasureHandle, [in] VARIANT SetType, [in] VARIANT Function )
```

Specify a fuzzy member function. `SetFuzzyMeasure` specifies a fuzzy member function passed in `Function`. The specified fuzzy functions enable `FuzzyMeasurePos` and `FuzzyMeasurePairs / FuzzyMeasurePairing` to evaluate and select the detected edge candidates. For this purpose, weighting characteristics for different edge features can be defined by one function each. Such a specified feature is called fuzzy set. Specifying no function for a fuzzy set means not to use this feature for the final edge evaluation. Setting a second fuzzy function to a set means to discard the first defined function and replace it by the second one. A previously defined fuzzy member function can be discarded completely by `ResetFuzzyMeasure`.

Functions for five different fuzzy set types selected by the `SetType` parameter can be defined, the sub types of a set being mutual exclusive:

- `'contrast'` will use the fuzzy function to evaluate the amplitudes of the edge candidates. When extracting edge pairs, the fuzzy evaluation is obtained by the geometric average of the fuzzy contrast scores of both edges.
- The fuzzy function of `'position'` evaluates the distance of each edge candidate to the reference point of the measure object, generated by `GenMeasureArc` or `GenMeasureRectangle2`. The reference point is located at the beginning whereas `'position: center'` or `'position: end'` sets the reference point to the middle or the end of the one-dimensional gray value profile instead. If the fuzzy position evaluation depends on the position of the object along the profile, `'position: first edge'` / `'position: last edge'` sets the reference point at the position of the first/last extracted edge. When extracting edge pairs the position of a pair is referenced by the geometric average of the fuzzy position scores of both edges.
- Similar to `'position'`, `'position: pair'` evaluates the distance of each edge pair to the reference point of the measure object. The position of a pair is defined by the center point between both edges. The object’s reference can be set by `'position: pair: center'`, `'position: pair: end'` and `'position: first pair'`, `'position: last pair'`, respectively. Contrary to `'position'`, this set is only used by `FuzzyMeasurePairs / FuzzyMeasurePairing`.
- `'size'` denotes a fuzzy set that evaluates the normed distance of the two edges of a pair in pixels. This set is only used by `FuzzyMeasurePairs / FuzzyMeasurePairing`. Specifying an upper bound for the size by terminating the member function with a corresponding fuzzy value of 0.0 will speed up `FuzzyMeasurePairs / FuzzyMeasurePairing` because not all possible pairs need to be considered.
- `'gray'` sets a fuzzy function to weight the mean projected gray value between two edges of a pair. This set is only used by `FuzzyMeasurePairs / FuzzyMeasurePairing`.

A fuzzy member function is defined as a piecewise linear function by at least two pairs of values, sorted in an ascending order by their x value. The x values represent the edge feature and must lie within the parameter space of the set type, i.e., in case of `'contrast'` and `'gray'` feature and, e.g., byte images within the range $0.0 \leq x \leq 255.0$. In case of `'size'` x has to satisfy $0.0 \leq x$ whereas in case of `'position'` x can be any real number. The y values of the fuzzy function represent the weight of the corresponding feature value and have to satisfy the range of $0.0 \leq y \leq 1.0$. Outside of the function’s interval, defined by the smallest and the greatest x value, the y values of the interval borders are continued constantly. Such Fuzzy member functions can be generated by `CreateFunct1DPairs`

If more than one set is defined, `FuzzyMeasurePos / FuzzyMeasurePairs / FuzzyMeasurePairing` yield the overall fuzzy weighting by the geometric middle of the weights of each set.

Parameter

- **MeasureHandle** (input control) ......................... measure_id ~> HMeasureX / VARIANT Measure object handle.
SetType (input control) .............................................. string \( \sim \) String / VARIANT
Selection of the fuzzy set.

Default Value: 'contrast'
List of values: SetType \( \in \) \{ 'position', 'position_center', 'position_end', 'position_first_edge', 'position_last_edge', 'position_pair_center', 'position_pair_end', 'position_first_pair', 'position_last_pair', 'size', 'gray', 'contrast' \}

Function (input control) ......................... function_id \( \sim \) HFunction1dX / VARIANT ( real, integer )
Fuzzy member function.

/* how to use a fuzzy function */
...
gen_measure_rectangle2 (50, 100, 0, 200, 100, 512, 512, 'nearest_neighbor', MeasureHandle)
/* create a generalized fuzzy function to evaluate edge pairs
 * (30% uncertainty). */
create_funct_1d_pairs ([0.7,1.0,1.3], [0.0,1.0,0.0], SizeFunction)
/* and transform it to expected size of 13.45 pixels */
transform_funct_1d (SizeFunction, [1.0,0.0,13.45,0.0], TransformedFunction)
set_fuzzy_measure (MeasureHandle, 'size', SizeFunction)

fuzzy_measure_pairs (Image, MeasureHandle, 1, 30, 0.5, 'all', RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, RowEdgeCenter, ColumnEdgeCenter, FuzzyScore, IntraDistance, InterDistance)

Parallelization Information
SetFuzzyMeasure is reentrant and processed without parallelization.

Possible Predecessors
GenMeasureArc, GenMeasureRectangle2, CreateFunct1DPairs, TransformFunct1D

Possible Successors
FuzzyMeasurePos, FuzzyMeasurePairs

See also
ResetFuzzyMeasure, FuzzyMeasurePairing

Alternatives
SetFuzzyMeasureNormPair

Module

void HMeasureX.SetFuzzyMeasureNormPair ([in] VARIANT PairSize, [in] String SetType, [in] HFunction1dX Function )


Specify a normalized fuzzy member function for edge pairs.
SetFuzzyMeasureNormPair specifies a normalized fuzzy member function passed in Function. The specified fuzzy functions enable FuzzyMeasurePos, FuzzyMeasurePairs and FuzzyMeasurePairing to evaluate and select the detected candidates of edges and edge pairs. For this purpose, weighting characteristics for different edge features can be defined by one function each. Such a specified feature is called fuzzy set. Specifying no function for a fuzzy set means not to use this feature for the final edge evaluation. Setting a second fuzzy function to a fuzzy set means to discard the first defined function and replace it by the second one. In difference
to `SetFuzzyMeasure`, the abscissa \( x \) of these member functions must be defined relative to the desired size \( s \) of the edge pairs (passed in `PairSize`). This enables a generalized usage of the defined functions. A previously defined normalized fuzzy member function can be discarded completely by `ResetFuzzyMeasure`.

Functions for three different fuzzy set types selected by the `SetType` parameter can be defined, the sub types of a set being mutually exclusive:

- **'size'** denotes a fuzzy set that evaluates the normalized distance of two edges of a pair in pixels:
  
  \[ x = \frac{d}{s} (x \geq 0) . \]

  Specifying an upper bound \( x_{\text{max}} \) for the size by terminating the member function with a corresponding fuzzy value of 0.0 will speed up `FuzzyMeasurePairs` / `FuzzyMeasurePairing` because not all possible pairs must be considered. Additionally, this fuzzy set can also be specified as a normalized size difference by 'size\_diff':

  \[ x = \frac{s - d}{s} (x \leq 1) \]

  and a absolute normalized size difference by 'size\_abs\_diff':

  \[ x = \frac{|s - d|}{s} (0 \leq x \leq 1) . \]

- **'position'** evaluates the signed distance \( p \) of each edge candidate to the reference point of the measure object, generated by `GenMeasureArc` or `GenMeasureRectangle2`:

  \[ x = \frac{p}{s} . \]

  The reference point is located at the beginning whereas 'position\_center' or 'position\_end' sets the reference point to the middle or the end of the one-dimensional gray value profile, instead. If the fuzzy position valuation depends on the position of the object along the profile 'position\_first\_edge' / 'position\_last\_edge' sets the reference point at the position of the first/last extracted edge. When extracting edge pairs, the position of a pair is referenced by the geometric average of the fuzzy position scores of both edges.

- **'position\_pair'** evaluates the signed distance of each edge pair to the reference point of the measure object. The position of a pair is defined by the center point between both edges. The object’s reference can be set by 'position\_pair\_center', 'position\_pair\_end' and 'position\_first\_pair', 'position\_last\_pair', respectively. Contrary to 'position', this set is only used by `FuzzyMeasurePairs` / `FuzzyMeasurePairing`.

A normalized fuzzy member function is defined as a piecewise linear function by at least two pairs of values, sorted in an ascending order by their \( x \) value. The \( y \) values of the fuzzy function represent the weight of the corresponding feature value and must satisfy the range of 0.0 \( \leq y \leq 1.0 \). Outside of the function’s interval, defined by the smallest and the greatest \( x \) value, the \( y \) values of the interval borders are continued constantly. Such Fuzzy member functions can be generated by `CreateFunct1DPairs`.

If more than one set is defined, `FuzzyMeasurePos` / `FuzzyMeasurePairs` / `FuzzyMeasurePairing` yield the overall fuzzy weighting by the geometric mean of the weights of each set.

---

**Parameter**

- **`MeasureHandle`** (input control) ........................ measure\_id \( \sim \) `HMeasureX` / `VARIANT`
  
  Measure object handle.

- **`PairSize`** (input control) .............................. number \( \sim \) `VARIANT` ( real, integer )
  
  Favored width of edge pairs.

  **Default Value** : 10.0

  **List of values** : `PairSize \in \{4.0, 6.0, 8.0, 10.0, 15.0, 20.0, 30.0\}`

  **Minimum Increment** : 0.1

  **Recommended Increment** : 1.0

- **`SetType`** (input control) ................................ string \( \sim \) `String` / `VARIANT`
  
  Selection of the fuzzy set.

  **Default Value** : 'size\_abs\_diff'

  **List of values** : `SetType \in \{'size', 'size\_diff', 'size\_abs\_diff', 'position', 'position\_center', 'position\_end', 'position\_first\_edge', 'position\_last\_edge', 'position\_pair\_center', 'position\_pair\_end', 'position\_first\_pair', 'position\_last\_pair'\}`

- **`Function`** (input control) ............................ function\_id \( \sim \) `HFunction1dX` / `VARIANT` ( real, integer )
  
  Fuzzy member function.

HALCON 6.1.4
/* how to use a fuzzy function */
...
gen_measure_rectangle2 (50, 100, 0, 200, 100, 512, 512, 'nearest_neighbor',
 MeasureHandle)

/* create a generalized fuzzy function to evaluate edge pairs */
create_funct_1d_pairs ([0.7, 1.0, 1.3], [0.0, 1.0, 0.0], SizeFunction)
/* and set it for an expected pair size of 13.45 pixels */
set_fuzzy_measure_norm_pair (MeasureHandle, 13.45, 'size', SizeFunction)

fuzzy_measure_pairs (Image, MeasureHandle, 1, 30, 0.5, 'all', RowEdgeFirst,
 ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, RowEdgeCenter, ColumnEdgeCenter, FuzzyScore, IntraDistance, InterDistance)

---

Parallelization Information

SetFuzzyMeasureNormPair is reentrant and processed without parallelization.

---

Possible Predecessors
GenMeasureArc, GenMeasureRectangle2, CreateFunct1DPairs

---

Possible Successors
FuzzyMeasurePairs, FuzzyMeasurePairing

See also
ResetFuzzyMeasure

Alternatives
TransformFunct1D, SetFuzzyMeasure

Module

---

Sub-pixel operators

void HMeasureX.TranslateMeasure ([in] VARIANT Row, [in] VARIANT Column)

Translate a measure object.

TranslateMeasure translates the reference point of the measure object given by MeasureHandle to the point (Row, Column). If the measure object and the translated measure object lie completely within the image, the measure object is shifted to the new reference point in an efficient manner. Otherwise, the measure object is generated anew with GenMeasureRectangle2 or GenMeasureArc using the parameters that were specified when the measure object was created and the new reference point.

---

Parameter

MeasureHandle (input control) ......................... measure_id ~ HMeasureX / VARIANT Measure object handle.

Row (input control) ................................................ point_y ~ VARIANT ( integer, real )
Row coordinate of the new reference point.

Default Value : 50.0
Suggested values : Row ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typical range of values : 0.0 ≤ Row ≤ 0.0(lin)
Minimum Increment : 1.0
Recommended Increment : 10.0
Column (input control) ........................................ point.x  \(\sim\) VARIANT (integer, real)
Column coordinate of the new reference point.
Default Value: 100.0
Suggested values: Column \(\in\) \{10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0\}
Typical range of values: \(0.0 \leq \text{Column} \leq 0.0\) (lin)
Minimum Increment: 1.0
Recommended Increment: 10.0

Result
If the parameter values are correct the operator TranslateMeasure returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
TranslateMeasure is reentrant and processed without parallelization.

Possible Predecessors
GenMeasureRectangle2, GenMeasureArc

Possible Successors
MeasurePos, MeasurePairs, FuzzyMeasurePos, FuzzyMeasurePairs, FuzzyMeasurePairing, MeasureThresh

See also
CloseMeasure

Alternatives
GenMeasureRectangle2, GenMeasureArc

Module
Sub-pixel operators

13.12 OCR

Add characters to a training file.
The operator AppendOcrTrainf serves to prepare the training with the operator TrainfOcrClassBox. Hereby regions, representing characters, including their gray values (region and pixel) and the corresponding class name will be written into a file. An arbitrary number of regions within one image is supported. For each character (region) in Character the corresponding class name must be specified in Class. The gray values are passed via the parameter Image. In contrast to the operator WriteOcrTrainf the characters are appended to an existing file using the same training file format as this file. If the file does not exist, a new file is generated. In this case, the file format can be chosen by the parameter 'ocr_trainf_version' of the operator SetSystem.

Parameter

Character (input iconic) .................................. region(-array)  \(\sim\) HRegionX / IObjectX
Characters to be trained.

Image (input iconic) .................................. image  \(\sim\) HImageX / IObjectX (byte)
Gray values of the characters.

Class (input control) .................................. string(-array)  \(\sim\) VARIANT (string)
Class (name) of the characters.

FileName (input control) .................................... filename  \(\sim\) String / VARIANT
Name of the training file.
Default Value: 'train_ocr'

Result
If the parameters are correct, the operator AppendOcrTrainf returns the value TRUE. Otherwise an exception will be raised.
Parallelization Information

AppendOcrTrainf is processed under *mutual exclusion* against itself and without parallelization.

Possible Predecessors

Threshold, Connection, CreateOcrClassBox, ReadOcr

Possible Successors

TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, DoOcrMulti, DoOcrSingle

Alternatives

WriteOcrTrainf, WriteOcrTrainfImage

Module

Optical character recognition

| void HMiscX.CloseAllOcrs ( ) |
| void HOperatorSetX.CloseAllOcrs ( ) |

Destroy all OCR classifiers.

CloseAllOcrs deletes all OCR classifiers and frees the used memory space. All the trained data will be lost.

Attention

Since all classifiers are closed by CloseAllOcrs all handles become invalid.

Result

If it is possible to close the OCR classifiers the operator CloseAllOcrs returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information

CloseAllOcrs is processed *completely exclusively* without parallelization.

Alternatives

CloseOcr

Module

Optical character recognition

| void HOperatorSetX.CloseOcr ([in] VARIANT OcrHandle ) |

Deallocation of the memory of an OCR classifier.

The operator CloseOcr deallocates the memory of the classifier having the number OcrHandle. Hereby all corresponding data will be deleted. However, if necessary, they can be saved in advance using the operator WriteOcr. The number OcrHandle will be invalid after the call; but later the system can use it again for new classifiers.

Attention

All data of the classifier will be deleted in main memory (not on the hard disk).

Parameter

- **OcrHandle** (input control) .................. ocr ~ HOcrX / VARIANT
  ID of the OCR classifier to be deleted.

Example

```

```

Result

If the parameter OcrHandle is valid, the operator CloseOcr returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

CloseOcr is *reentrant* and processed *without* parallelization.
Concat training files.

The operator `ConcatOcrTrainf` stores all characters which are contained in the files `SingleFiles` into a new file with the name `ComposedFile`. The file format can be defined by the parameter 'ocr_trainf_version' of the operator `SetSystem`. The file name can be chosen at will.

**Parameter**

- **SingleFiles** (input control) Name of the single training files.
  - Default Value: ”
- **ComposedFile** (input control) Name of the composed training file.
  - Default Value: 'all_characters'

**Result**

If the parameters are correct, the operator `ConcatOcrTrainf` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**

`ConcatOcrTrainf` is processed under mutual exclusion against itself and without parallelization.

Create a new OCR-classifier.

The operator `CreateOcrClassBox` creates a new OCR classifier. For a description of this classifier see operator `LearnClassBox`. This classifier must then be trained with the help of the operators `TrainOcrClassBox` or `TrainfOcrClassBox`.

The parameters `WidthPattern` and `HeightPattern` indicate the size of the input-layer of the network. This size is used for the features 'projection_horizontal', 'projection_vertical', 'pixel', 'pixel_invar', and 'pixel_binary' to transform the character to a standard size. The bigger the standard size is, the more characters can be distinguished. Hereby the amount of time necessary for the training (as well as the number of training random samples)
and the time necessary for the recognition, however, will increase as well. The parameter \texttt{Interpolation} indicates the interpolation mode concerning the adaptation of characters in the image to the network. For more detailed information on this parameter see also \texttt{AffineTransImage}. The value \texttt{0} results in the same interpolation as \texttt{"none"} in \texttt{AffineTransImage}, i.e., no interpolation is performed. For \texttt{1}, the same behavior as \texttt{"constant"} in \texttt{AffineTransImage} is obtained, i.e., equally weighted interpolation between adjacent pixels is used. Finally, \texttt{2} results in the same interpolation as \texttt{"weighted"}, i.e., Gaussian interpolation between adjacent pixels is used. The parameter \texttt{Interpolation} must be chosen such that no aliasing occurs when the character is scaled to the standard size. Typically, this means that \texttt{Interpolation} should be set to \texttt{1}, except in cases where the characters are scaled down by a large amount, in which case \texttt{Interpolation} = \texttt{2} should be chosen. \texttt{Interpolation} = \texttt{0} should only be chosen if the characters will not be scaled.

The parameter \texttt{Character} determines all the characters which have to be recognized. Normally the transmitted strings consist of one character (e.g. alphabet). But also strings of any length can be learned. The number of distinguishable characters (number of strings in \texttt{Character}) is limited to \texttt{2048}.

The parameter \texttt{Features} helps to choose additional features besides gray values in order to recognize characters. By using \texttt{"default"} the features \texttt{\textquote{ratio}} and \texttt{\textquote{pixel\_invar}} will be set.

The following features are available:

- \texttt{\textquote{ratio}}’ Ratio of the character.
- \texttt{\textquote{width}}’ Width of the character (not invariant to scaling).
- \texttt{\textquote{height}}’ Height of the character (not invariant to scaling).
- \texttt{\textquote{zoom\_factor}}’ Difference in size between the current character and the values of \texttt{WidthPattern} and \texttt{HeightPattern} (not invariant to scaling).
- \texttt{\textquote{foreground}}’ Relative number of pixels in the foreground.
- \texttt{\textquote{foreground\_grid\_9}}’ Relative number of foreground pixels in a $3 \times 3$ grid within the surrounding rectangle of the character.
- \texttt{\textquote{foreground\_grid\_16}}’ Relative number of foreground pixels in a $4 \times 4$ grid within the surrounding rectangle of the character.
- \texttt{\textquote{anisometry}}’ Form feature anisometry.
- \texttt{\textquote{compactness}}’ Form feature compactness.
- \texttt{\textquote{convexity}}’ Form feature convexity.
- \texttt{\textquote{moments\_region\_2nd\_invar}}’ Normed 2nd geometric moments of the region. See also \texttt{MomentsRegion2NdInvar}.
- \texttt{\textquote{moments\_region\_2nd\_rel\_invar}}’ Normed 2nd relativ geometric moments of the region. See also \texttt{MomentsRegion2NdRelInvar}.
- \texttt{\textquote{moments\_region\_3rd\_invar}}’ Normed 3rd geometric moments of the region. See also \texttt{MomentsRegion3RdInvar}.
- \texttt{\textquote{moments\_central}}’ Normed central geometric moments of the region. See also \texttt{MomentsRegionCentral}.
- \texttt{\textquote{phi}}’ Orientation (angle) of the character.
- \texttt{\textquote{num\_connect}}’ Number of connecting components.
- \texttt{\textquote{num\_holes}}’ Number of holes.
- \texttt{\textquote{projection\_horizontal}}’ Horizontal projection of the gray values.
- \texttt{\textquote{projection\_horizontal\_invar}}’ Horizontal projection of the gray values with are automatically scaled to maximum range.
- \texttt{\textquote{projection\_vertical}}’ Vertical projection of the gray values.
- \texttt{\textquote{projection\_vertical\_invar}}’ Vertical projection of the gray values with are automatically scaled to maximum range.
- \texttt{\textquote{cooc}}’ Values of the binary cooccurrence matrix.
- \texttt{\textquote{moments\_gray\_plane}}’ Normed gray value moments and the angles of the gray value level.
- \texttt{\textquote{num\_runs}}’ Number of chords in the region normed to the area.
- \texttt{\textquote{chord\_histro}}’ Frequency of the chords per row.
- \texttt{\textquote{pixel}}’ Gray value of the character.
Gray values of the character with automatic maximal scaling of the gray values.

Region of the character as a binary image zoomed to a size of \( \text{WidthPattern} \times \text{HeightPattern} \).

---

**Attention**

---

**Parameter**

- **WidthPattern** (input control) .......................... integer \( \rightarrow \) long / VARIANT
  - Width of the input layer of the network.
  - Default Value: 8
  - Suggested values: \( \text{WidthPattern} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20\} \)
  - Typical range of values: \( 1 \leq \text{WidthPattern} \leq 1 \)

- **HeightPattern** (input control) .......................... integer \( \rightarrow \) long / VARIANT
  - Height of the input layer of the network.
  - Default Value: 10
  - Suggested values: \( \text{HeightPattern} \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20\} \)
  - Typical range of values: \( 1 \leq \text{HeightPattern} \leq 1 \)

- **Interpolation** (input control) .......................... integer \( \rightarrow \) long / VARIANT
  - Interpolation mode concerning scaling of characters.
  - Default Value: 1
  - List of values: \( \text{Interpolation} \in \{0, 1, 2\} \)

- **Features** (input control) ......................... string(-array) \( \rightarrow \) VARIANT ( string )
  - Additional features.
  - Default Value: ‘default’

- **Character** (input control) .......................... string \( \rightarrow \) VARIANT ( string )
  - All characters of a set.
  - Default Value: [‘a’, ‘b’, ‘c’]

- **OcrHandle** (output control) .......................... ocr \( \rightarrow \) HOcrX / VARIANT
  - ID of the created OCR classifier.

---

**Example**

---

**Result**

If the parameters are correct, the operator **CreateOcrClassBox** returns the value TRUE. Otherwise an exception will be raised.

---

**Parallelization Information**

**CreateOcrClassBox** is processed completely exclusively without parallelization.

---

**Possible Predecessors**

- ResetObjDb

---

**Possible Successors**

- TraindOcrClassBox, TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, OcrChangeChar

---

**See also**

- AffineTransImage, OcrChangeChar, MomentsRegion2NdInvar, MomentsRegion2NdRelInvar, MomentsRegion3RdInvar, MomentsRegionCentral

---

**Module**

- Optical character recognition

---

**HALCON 6.1.4**
Classify characters.

The operator DoOcrMulti assigns a class to every Character (character). For gray value features the gray values from the surrounding rectangles of the regions are used. The gray values will be taken from the parameter Image. For each character the corresponding class will be returned in Class and a confidence value will be returned in Confidence. The confidence value indicates the similarity between the input pattern and the assigned character.

---

**Attention**

---

**Parameter**

- **Character** (input iconic) ........................................ region(-array) ⇒ HRegionX / IHObjectX Characters to be recognized.
- **Image** (input iconic) ............................................ image ⇒ HImageX / IHObjectX (byte) Gray values for the characters.
- **OcrHandle** (input control) ....................................... ocr ⇒ HOcrX / VARIANT ID of the OCR classifier.
- **Class** (output control) ........................................... string(-array) ⇒ VARIANT (string) Class (name) of the characters.
  **Number of elements** : \( Class = Character \)
- **Confidence** (output control) .................................... real(-array) ⇒ VARIANT (real) Confidence values of the characters.
  **Number of elements** : \( Confidence = Character \)

---

**Example**

---

**Parallelization Information**

DoOcrMulti is reentrant and automatically parallelized (on tuple level).

---

**Possible Predecessors**

TrainOcrClassBox, TrainfOcrClassBox, ReadOcr, Connection, SortRegion

---

**See also**

WriteOcr

---

**Alternatives**

DoOcrSingle

---

**Module**

Optical character recognition

---

Classify one character.
The operator \texttt{DoOcrSingle} assigns classes to the \texttt{Character} (characters). For gray value features gray values of the surrounding rectangles of the regions will be used. The gray values will be taken from the parameter \texttt{Image}. For each character the two classes with the highest confidences will be returned in \texttt{Classes}. The corresponding confidences will be returned in \texttt{Confidences}. The confidence value indicates the similarity between the input pattern and the assigned character.

\begin{verbatim}
\textbf{Attention}
\end{verbatim}

\begin{itemize}
\item \texttt{Character} (input iconic) \texttt{region} \texttt{HRegionX / IHObjectX}
  Character to be recognized.
\item \texttt{Image} (input iconic) \texttt{image} \texttt{HImageX / IHObjectX (byte)}
  Gray values of the characters.
\item \texttt{OcrHandle} (input control) \texttt{ocr} \texttt{HOcrX / VARIANT}
  ID of the OCR classifier.
\item \texttt{Classes} (output control) \texttt{string} \texttt{VARIANT(string)}
  Classes (names) of the characters.
  \textbf{Number of elements : 2}
\item \texttt{Confidences} (output control) \texttt{real} \texttt{VARIANT(real)}
  Confidence values of the characters.
  \textbf{Number of elements : 2}
\end{itemize}

\begin{verbatim}
\textbf{Example}
\end{verbatim}

If the input parameters are correct, the operator \texttt{DoOcrSingle} returns the value TRUE. Otherwise an exception will be raised.

\begin{verbatim}
\textbf{Parallelization Information}
\end{verbatim}

\texttt{DoOcrSingle} is \textit{reentrant} and processed \textit{without} parallelization.

\begin{verbatim}
\textbf{Possible Predecessors}
\end{verbatim}

\texttt{TraindOcrClassBox, TrainfOcrClassBox, ReadOcr, Connection, SortRegion}

\begin{verbatim}
\textbf{See also}
\end{verbatim}

\texttt{WriteOcr}

\begin{verbatim}
\textbf{Alternatives}
\end{verbatim}

\texttt{DoOcrMulti}

\begin{verbatim}
\textbf{Module}
\end{verbatim}


\textbf{Optical character recognition}

\begin{verbatim}
[out] long WidthPattern \texttt{HOcrX.InfoOcrClassBox} ([out] long HeightPattern,
[out] long Interpolation, [out] long WidthMaxChar, [out] long HeightMaxChar,
[out] VARIANT Features, [out] VARIANT Characters )

void \texttt{HOperatorSetX.InfoOcrClassBox} ([in] VARIANT OcrHandle,
[out] VARIANT WidthPattern, [out] VARIANT HeightPattern,
[out] VARIANT Interpolation, [out] VARIANT WidthMaxChar,
[out] VARIANT HeightMaxChar, [out] VARIANT Features,
[out] VARIANT Characters )
\end{verbatim}

\textit{Get information about an OCR classifier.}

The operator \texttt{InfoOcrClassBox} returns some information about an OCR classifier. The parameters are equivalent to those of \texttt{CreateOcrClassBox}. The parameters \texttt{WidthMaxChar} and \texttt{HeightMaxChar} indicate the extension of the largest trained character. These values can be used to control the segmentation.

\begin{verbatim}
\textbf{Attention}
\end{verbatim}

HALCON 6.1.4
CHAPTER 13. TOOLS

Parameter

- **OcrHandle** (input control) ........................................... ocr  \(\propto\) HOcrX / VARIANT
  ID of the OCR classifier.
- **WidthPattern** (output control) ...................................... integer  \(\propto\) long / VARIANT
  Width of the scaled characters.
- **HeightPattern** (output control) ..................................... integer  \(\propto\) long / VARIANT
  Height of the scaled characters.
- **Interpolation** (output control) ................................... integer  \(\propto\) long / VARIANT
  Interpolation mode for scaling the characters.
- **WidthMaxChar** (output control) .................................... integer  \(\propto\) long / VARIANT
  Width of the largest trained character.
- **HeightMaxChar** (output control) .................................... integer  \(\propto\) long / VARIANT
  Height of the largest trained character.
- **Features** (output control) ......................................... string  \(\propto\) VARIANT (string)
  Used features.
- **Characters** (output control) ....................................... string  \(\propto\) VARIANT (string)
  All characters of the set.

Example

The operator *InfoOcrClassBox* always returns TRUE.

Parallelization Information

*InfoOcrClassBox* is reentrant and processed without parallelization.

Possible Predecessors

ReadOcr, CreateOcrClassBox

Possible Successors

WriteOcr

Module

Optical character recognition

```
void HOcrX.OcrChangeChar ([in] VARIANT Character )
void HOperatorSetX.OcrChangeChar ([in] VARIANT OcrHandle, [in] VARIANT Character )
```

Define a new conversion table for the characters.

The operator *OcrChangeChar* establishes a new look-up table for the characters. Hereby the number of strings of *Character* must be the same as of the classifier *OcrHandle*. In order to enlarge the font, the operator *OcrChangeChar* may be used as follows: More characters than actually needed will be indicated when creating a network using (*CreateOcrClassBox*). The last *n* characters will not be used so far. If more characters are needed at a later stage, these unused characters will be allocated and then trained with the help of the operator *OcrChangeChar*.

Attention

Parameter

- **OcrHandle** (input control) ........................................... ocr  \(\propto\) HOcrX / VARIANT
  ID of the OCR-network to be changed.
- **Character** (input control) ........................................... string  \(\propto\) VARIANT (string)
  New assign of characters.
  Default Value: ['a','b','c']
Example

Result

If the number of characters in `Character` is identical with the number of the characters of the network, the operator `OcrChangeChar` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**

`OcrChangeChar` is processed *completely exclusively* without parallelization.

**Possible Predecessors**

ReadOcr

**Possible Successors**

DoOcrMulti, DoOcrSingle, WriteOcr

Optical character recognition

---

**Module**

Optical character recognition

### Example

**Result**

If the number of characters in `Character` is identical with the number of the characters of the network, the operator `OcrChangeChar` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**

`OcrChangeChar` is processed *completely exclusively* without parallelization.

**Possible Predecessors**

ReadOcr

**Possible Successors**

DoOcrMulti, DoOcrSingle, WriteOcr

Optical character recognition

---

**Module**

Optical character recognition

### Example

**Result**

If the number of characters in `Character` is identical with the number of the characters of the network, the operator `OcrChangeChar` returns the value TRUE. Otherwise an exception will be raised.

**Parallelization Information**

`OcrChangeChar` is processed *completely exclusively* without parallelization.

**Possible Predecessors**

ReadOcr

**Possible Successors**

DoOcrMulti, DoOcrSingle, WriteOcr

Optical character recognition

---

**Module**

Optical character recognition
Read OCR classifier from a file.

The operator \texttt{ReadOcr} reads an OCR classifier from a file \texttt{FileName}. This file will hereby be searched in the directory ($HALCONROOT/ocr/) as well as in the currently used directory. If too many classifiers have been loaded, an error message will be displayed.

\begin{verbatim}
void HOcrX.ReadOcr ([in] String FileName )
void HOperatorSetX.ReadOcr ([in] VARIANT FileName, [out] VARIANT OcrHandle )
\end{verbatim}

\textit{Read an OCR classifier from a file.}

\textbf{Attention}

Parameter

\begin{itemize}
\item File\texttt{Name} (input control) \hspace{1cm} filename \textasciitilde String / VARIANT
  Name of the OCR classifier file.
  \textbf{Default Value :} 'testnet'
\item Ocr\texttt{Handle} (output control) \hspace{1cm} ocr \textasciitilde HOcrX / VARIANT
  ID of the read OCR classifier.
\end{itemize}

\textbf{Example}

If the indicated file is available and the format is correct, the operator \texttt{ReadOcr} returns the value TRUE. Otherwise an exception will be raised.

\begin{verbatim}
ResetObjDb
\end{verbatim}

\textbf{Parallelization Information}

\begin{verbatim}
ReadOcr is processed completely exclusively without parallelization.
\end{verbatim}

\textbf{Possible Predecessors}

\begin{verbatim}
ResetObjDb
\end{verbatim}

\textbf{Possible Successors}

\begin{verbatim}
DoOcrMulti, DoOcrSingle, TrainOcrClassBox, TrainfOcrClassBox
\end{verbatim}

\textbf{See also}

\begin{verbatim}
WriteOcr, DoOcrMulti, TrainOcrClassBox, TrainfOcrClassBox
\end{verbatim}

\textbf{Module}

Optical character recognition

\begin{verbatim}
void HImageX.ReadOcrTrainf ([in] VARIANT TrainFileNames, [out] VARIANT CharacterNames )
void HOperatorSetX.ReadOcrTrainf ([in] HUntypedObjectX Characters, [in] VARIANT TrainFileNames, [out] VARIANT CharacterNames )
\end{verbatim}

\textit{Read training characters from files and convert to images.}

\texttt{ReadOcrTrainf} reads all characters from the specified file names and converts them into images. The domain is defined according to the foreground of the characters (as specified in \texttt{WriteOcrTrainf}). The names of the characters are returned in \texttt{CharacterNames}. If more than one file name is given the files are processed in the order the file names.

\begin{verbatim}
Characters (output iconic) \hspace{1cm} image \textasciitilde HImageX / HUntypedObjectX ( byte )
Images read from file.
\item Train\texttt{FileNames} (input control) \hspace{1cm} filename.named(-array) \textasciitilde VARIANT ( string )
Names of the training files.
\textbf{Default Value} : ”
\end{verbatim}
CharacterNames (output control) .................................................. string ~> VARIANT( string )
Names of the read characters.

Result
If the parameter values are correct the operator ReadOcrTrainf returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
ReadOcrTrainf is reentrant and processed without parallelization.

Possible Predecessors
WriteOcrTrainf

Possible Successors
DispImage, SelectObj, ZoomImageSize

See also
TrainfOcrClassBox

Alternatives
ReadOcrTrainfSelect

Module
Optical character recognition

Query which characters are stored in a training file.
ReadOcrTrainfNames extracts the names and frequency of all characters in the specified training files.

Parameter

TrainFileNames (input control) ......................... filename.named(-array) ~> VARIANT( string )
Names of the training files.
Default Value: ”

CharacterNames (output control) ......................... string(-array) ~> VARIANT( string )
Names of the read characters.

CharacterCount (output control) ......................... integer(-array) ~> VARIANT( integer )
Number of the characters.

Result
If the parameter values are correct the operator ReadOcrTrainfNames returns the value TRUE. Otherwise an exception handling is raised.

Parallelization Information
ReadOcrTrainfNames is reentrant and processed without parallelization.

Possible Predecessors
WriteOcrTrainf

See also
TrainfOcrClassBox

Module
Optical character recognition
Read training specific characters from files and convert to images.

_readOcrTrainfSelect_ reads the characters given in _SearchNames_ from the specified files and converts them into images. It works similar to _ReadOcrTrain_ but here the characters which are extracted can be specified.

---

**Parameter**

- **Characters** (output iconic) ......................... image \( \sim \) HImageX / HUntypedObjectX (byte)
  Images read from file.
- **TrainFileNames** (input control) ..................... filename.named(-array) \( \sim \) VARIANT (string)
  Names of the training files.
  Default Value: ”
- **SearchNames** (input control) ......................... string(-array) \( \sim \) VARIANT (string)
  Names of the characters to be extracted.
  Default Value: ’0’
- **FoundNames** (output control) ......................... string(-array) \( \sim \) VARIANT (string)
  Names of the read characters.

---

**Result**

If the parameter values are correct the operator _ReadOcrTrainfSelect_ returns the value TRUE. Otherwise an exception handling is raised.

---

**Parallelization Information**

_ReadOcrTrainfSelect_ is _reentrant_ and processed without parallelization.

---

**Possible Predecessors**

WriteOcrTrainf

---

**Possible Successors**

DispImage, SelectObj, ZoomImageSize

---

**See also**

TrainOcrClassBox

---

**Alternatives**

ReadOcrTrainf

---

**Module**

Optical character recognition

---

Train an OCR classifier by the input of regions.

_TestdOcrClassBox_ tests the confidence with which a character belongs to a given class. Any number of regions of an image can be passed. For each character (region) in _Character_ the corresponding name (class) _Class_ must be specified. The gray values are passed in _Image_. When the operator has finished the parameter _Confidence_ provides information about how sure a character belongs to the (arbitrary chosen) class.
Parameter

- **Character** (input iconic) region(-array) \( \sim HRegionX / IObjectX \)
  Characters to be trained.
- **Image** (input iconic) image \( \sim HImageX / IObjectX \) (byte)
  Gray values for the characters.
- **OcrHandle** (input control) ocr \( \sim HOcrX / Variant \)
  ID of the desired OCR-classifier.
- **Class** (input control) string(-array) \( \sim Variant \) (string)
  Class (name) of the characters.
  Default Value: 'a'
- **Confidence** (output control) real(-array) \( \sim Variant \) (real)
  Confidence for the character to belong to the class.

Result

If the parameters are correct, the operator **TestdOcrClassBox** returns the value TRUE. Otherwise an exception will be raised.

Parallelization Information

**TestdOcrClassBox** is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

ReadOcr, TrainfOcrClassBox, TraindOcrClassBox

Module

Optical character recognition

---

**Train an OCR classifier by the input of regions.**

The operator **TraindOcrClassBox** trains the classifier directly via the input of regions in an image. Any number of regions of an image can be passed. For each character (region) in **Character** the corresponding name (class) **Class** must be specified. The gray values are passed in **Image**. When the procedure has finished the parameter **AvgConfidence** provides information about the success of the training: It contains the average confidence of the trained characters measured by a re-classification. The confidence of mismatched characters is set to 0 (thus, the average confidence will be decreased significantly).

Attention

Parameter

- **Character** (input iconic) region(-array) \( \sim HRegionX / IObjectX \)
  Characters to be trained.
- **Image** (input iconic) image \( \sim HImageX / IObjectX \) (byte)
  Gray values for the characters.
- **OcrHandle** (input control) ocr \( \sim HOcrX / Variant \)
  ID of the desired OCR-classifier.
- **Class** (input control) string(-array) \( \sim Variant \) (string)
  Class (name) of the characters.
  Default Value: 'a'
- **AvgConfidence** (output control) real \( \sim double / Variant \)
  Average confidence during a re-classification of the trained characters.
If the parameters are correct, the operator \texttt{Train\textsc{dOcrClassBox}} returns the value TRUE. Otherwise an exception will be raised.

\textbf{Parallelization Information}

\texttt{Train\textsc{dOcrClassBox}} is processed \textit{completely exclusively} without parallelization.

\textbf{Possible Predecessors}

\texttt{CreateOcrClassBox, ReadOcr}

\textbf{Possible Successors}

\texttt{Train\textsc{dOcrClassBox}, WriteOcr, DoOcrMulti, DoOcrSingle}

\textbf{Alternatives}

\texttt{Train\textsc{fOcrClassBox}}

\textbf{Module}

Optical character recognition

\begin{verbatim}
void \texttt{HOperatorSetX.Train\textsc{fOcrClassBox}} ([in] VARIANT OcrHandle, [in] VARIANT FileName, [out] VARIANT AvgConfidence )

Train an OCR classifier with the help of a training file.

The operator \texttt{Train\textsc{fOcrClassBox}} trains the classifier \texttt{OcrHandle} via the indicated training files. Any number of files can be indicated. The parameter \texttt{AvgConfidence} provides information about the success of the training: It contains the average confidence of the trained characters measured by a re-classification. The confidence of mismatched characters is set to 0 (thus, the average confidence will be decreased significantly).

\textbf{Attention}

The names of the characters in the file must fit the network.

\textbf{Parameter}

\begin{itemize}
  \item \texttt{OcrHandle} (input control) \hspace{1cm} ocr \sim HOcrX / VARIANT ID of the desired OCR-network.
  \item \texttt{FileName} (input control) \hspace{1cm} filename(-array) \sim VARIANT ( string ) Name(s) of the training file(s).
    \textbf{Default Value :} "train\_ocr"
  \item \texttt{AvgConfidence} (output control) \hspace{1cm} real \sim double / VARIANT Average confidence during a re-classification of the trained characters.
\end{itemize}

\textbf{Example}

If the file name is correct and the data fit the network, the operator \texttt{Train\textsc{fOcrClassBox}} returns the value TRUE. Otherwise an exception will be raised.

\textbf{Parallelization Information}

\texttt{Train\textsc{fOcrClassBox}} is processed \textit{completely exclusively} without parallelization.

\textbf{Possible Predecessors}

\texttt{CreateOcrClassBox, ReadOcr}

\textbf{Possible Successors}

\texttt{Train\textsc{fOcrClassBox}, WriteOcr, DoOcrMulti, DoOcrSingle}

\end{verbatim}
Alternatives

TrainOcrClassBox

Module

Optical character recognition

```c
void HOcrX.WriteOcr ([in] String FileName )
void HOperatorSetX.WriteOcr ([in] VARIANT OcrHandle,
[in] VARIANT FileName )
```

Writing an OCR classifier into a file.

The operator WriteOcr writes the OCR classifier OcrHandle into the file FileName. Since the data of the classifier will be lost when the program is finished, they have to be stored after the training if the user wants to use them again at a later execution of the program. The data can then be read with the help of the operator ReadOcr. The extension will be added automatically to the parameter FileName.

Attention

The output file FileName must be given without extension.

Parameter

- **OcrHandle** (input control) ............................... ocr ➔ HOcrX / VARIANT ID of the OCR classifier.
- **FileName** (input control) ............................... filename ➔ String / VARIANT Name of the file for the OCR classifier (without extension).

Default Value: ’my_ocr’

Example

```
If the parameter OcrHandle is valid and the indicated file can be written, the operator WriteOcr returns the value TRUE. Otherwise an exception will be raised.
```

Parallelization Information

WriteOcr is reentrant and processed without parallelization.

Possible Predecessors

TrainOcrClassBox, TrainfOcrClassBox

Possible Successors

DoOcrMulti, DoOcrSingle

See also

ReadOcr, DoOcrMulti, TrainOcrClassBox, TrainfOcrClassBox

Module

Optical character recognition

```c
void HRegionX.WriteOcrTrainf ([in] HImageX Image, [in] VARIANT Class,
[in] String FileName )
void HOperatorSetX.WriteOcrTrainf ([in] IObjectX Character,
```

Storing of trained characters into a file.

The operator WriteOcrTrainf serves to prepare the training with the operator TrainfOcrClassBox. Hereby regions, representing characters, including their gray values (region and pixel) and the corresponding class name will be written into a file. An arbitrary number of regions within one image is supported. For each character (region) in Character the corresponding class name must be specified in Class. The gray values are passed
CHAPTER 13. TOOLS

via the parameter Image. The file name can be chosen at will. The version of the file format used for writing data can be defined by the parameter 'ocr_trainf_version' of the operator SetSystem.

\[
\text{Attention}\]

\[
\text{Parameter}\]

- **Character** (input iconic) \( \text{region}(-\text{array}) \sim \text{HRegionX / IHOBJECTX} \)
  Characters to be trained.
- **Image** (input iconic) \( \text{image} \sim \text{HImageX / IHOBJECTX (byte)} \)
  Gray values of the characters.
- **Class** (input control) \( \text{string}(-\text{array}) \sim \text{VARIANT (string)} \)
  Class (name) of the characters.
- **FileName** (input control) \( \text{filename} \sim \text{String / VARIANT} \)
  Name of the training file.

**Default Value:** 'train_ocr'

\[
\text{Example}\]

\[
\text{Result}\]

If the parameters are correct, the operator WriteOcrTrainf returns the value TRUE. Otherwise an exception will be raised.

\[
\text{Parallelization Information}\]

WriteOcrTrainf is reentrant and processed without parallelization.

\[
\text{Possible Predecessors}\]

Threshold, Connection, CreateOcrClassBox, ReadOcr

\[
\text{Possible Successors}\]

TrainfOcrClassBox, InfoOcrClassBox, WriteOcr, DoOcrMulti, DoOcrSingle

Optical character recognition

\[
\text{Module}\]

```
void HIImageX.WriteOcrTrainfImage ([in] VARIANT Class,
[in] String FileName )
```

```
void HOperatorSetX.WriteOcrTrainfImage ([in] IObjectX Character,
[in] VARIANT Class, [in] VARIANT FileName )
```

Write characters into a training file.

The operator WriteOcrTrainfImage is used to prepare the training with the operator TrainfOcrClassBox. Hereby regions, representing characters, including their gray values (region and pixel) and the corresponding class name will be written into a file. An arbitrary number of regions within one image is supported. For each character (region) in Character the corresponding class name must be specified in Class. The file name can be chosen at will. In contrast to WriteOcrTrainf one image per character is passed. The domain of this image defines the pixels which belong to the character. The file format can be defined by the parameter 'ocr_trainf_version' of the operator SetSystem.

\[
\text{Parameter}\]

- **Character** (input iconic) \( \text{image}(-\text{array}) \sim \text{HImageX / IHOBJECTX (byte)} \)
  Characters to be trained.
- **Class** (input control) \( \text{string}(-\text{array}) \sim \text{VARIANT (string)} \)
  Class (name) of the characters.
- **FileName** (input control) \( \text{filename} \sim \text{String / VARIANT} \)
  Name of the training file.

**Default Value:** 'train_ocr'

HALCON/COM Reference Manual, 2005-2-1
If the parameters are correct, the operator \texttt{WriteOcrTrainfImage} returns the value TRUE. Otherwise an exception will be raised.

\textbf{Parallelization Information}

\texttt{WriteOcrTrainfImage} is \textit{reentrant} and processed without parallelization.

\textbf{Possible Predecessors}

\textit{Threshold}, \textit{Connection}, \textit{CreateOcrClassBox}, \textit{ReadOcr}

\textbf{Possible Successors}

\textit{TrainfOcrClassBox}, \textit{InfoOcrClassBox}, \textit{WriteOcr}, \textit{DoOcrMulti}, \textit{DoOcrSingle}

\textbf{Alternatives}

\textit{WriteOcrTrainf}, \textit{AppendOcrTrainf}

\textbf{Module}

Optical character recognition

\subsection{13.13 OCV}

\begin{verbatim}
void HMiscX.CloseAllOcvs ( )
void HOperatorSetX.CloseAllOcvs ( )
\end{verbatim}

Clear all OCV tools.

\texttt{CloseAllOcvs} closes all OCV tools which have been opened using \texttt{CreateOcvProj} or \texttt{ReadOcv}. All handles are invalid after this call.

\textbf{Attention}

This operator is only for internal use (like e.g. a reset program in HDevelop).

\textbf{Result}

\texttt{CloseAllOcvs} returns always TRUE.

\textbf{Parallelization Information}

\texttt{CloseAllOcvs} is processed \textit{completely exclusively} without parallelization.

\textbf{Possible Predecessors}

\textit{ReadOcv}, \textit{CreateOcvProj}

\textbf{Alternatives}

\texttt{CloseOcv}

\textbf{Module}

Optical character verification

\begin{verbatim}
void HOperatorSetX.CloseOcv ([in] VARIANT OCVHandle )
\end{verbatim}

Clear an OCV tool.

\texttt{CloseOcv} closes an open OCV tool and frees the memory. The OCV tool has been created using \texttt{CreateOcvProj} or \texttt{ReadOcv}. The handle is after this call no longer valid.

\textbf{Parameter}

\texttt{OCVHandle} (input control) \hspace{1cm} \texttt{oce \sim HOcvX / VARIANT}

Handle of the OCV tool which has to be freed.

\textbf{Result}

\texttt{CloseOcv} returns TRUE, if the handle is valid. Otherwise, an exception handling is raised.

\textbf{Parallelization Information}

\texttt{CloseOcv} is processed \textit{completely exclusively} without parallelization.

\textbf{Possible Predecessors}

\textit{ReadOcv}, \textit{CreateOcvProj}
Create a new OCV tool based on gray value projections.

CreateOcvProj creates a new OCV tool. This tool will be used to train good-patterns for the optical character verification. The training is done using the operator TrainOcvProj. Thus TrainOcvProj is normally called after CreateOcvProj.

The pattern comparison is based on the gray projections: For every training pattern the horizontal and vertical gray projections are calculated by summing up the gray values along the rows and columns inside the region of the pattern. This operation is applied to the training patterns and the test patterns. For the training patterns the result is stored inside the OCV tool to save runtime while comparing patterns. The OCV is done by comparing the corresponding projections. The Quality is the similarity of the projections.

Input for CreateOcvProj are the names of the patterns (PatternNames) which have to be trained. The number and the names can be chosen arbitrary. In most cases only one pattern will be trained, thus only one name has to be specified. The names will be used when doing the OCV (DoOcvSimple). It is possible to specify more names than actually used. These might be trained later.

To close the OCV tool, i.e. to free the memory, the operator CloseOcv is called.

**Parameter**

- **PatternNames** (input control) ...................... string(-array) -> VARIANT ( string )
  List of names for patterns to be trained.
  Default Value: 'a'
- **OCVHandle** (output control) ......................... ocv -> HOcvX / VARIANT
  Handle of the created OCV tool.

**Result**

CreateOcvProj returns TRUE, if the parameters are correct. Otherwise, an exception handling is raised.

**Parallelization Information**

CreateOcvProj is processed completely exclusively without parallelization.

**Possible Successors**

TrainOcvProj, WriteOcv, CloseOcv

See also

- CreateOcrClassBox
- Alternatives
- ReadOcv

Module

Optical character verification
13.13. OCV

### OCVHandle

The operator **HOperatorSetX.DoOcvSimple** allows the verification of a pattern using an OCV tool.

#### Definition

**HOperatorSetX.DoOcvSimple**

```c
void HOperatorSetX.DoOcvSimple ([in] IObjectX Pattern,
```

#### Parameters

- **Pattern** (input iconic) 
  Characters to be verified.
- **OCVHandle** (input control) 
  Handle of the OCV tool.
- **PatternName** (input control) 
  Name of the character.
  Default Value: `a`
- **AdaptPos** (input control) 
  Adaption to vertical and horizontal translation.
  Default Value: `true`
  List of values: `AdaptPos ∈ {'true', 'false'}`
- **AdaptSize** (input control) 
  Adaption to vertical and horizontal scaling of the size.
  Default Value: `true`
  List of values: `AdaptSize ∈ {'true', 'false'}`
- **AdaptAngle** (input control) 
  Adaption to changes of the orientation (not yet implemented).
  Default Value: `false`
  List of values: `AdaptAngle ∈ {'false'}`
- **AdaptGray** (input control) 
  Adaption to additive and scaling gray value changes.
  Default Value: `true`
  List of values: `AdaptGray ∈ {'true', 'false'}`
- **Threshold** (input control) 
  Minimum difference between objects.
  Default Value: `10`
  Suggested values: `Threshold ∈ {-1, 0, 1, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100, 150}`
- **Quality** (output control) 
  Evaluation of the character.
  Typical range of values: `0.0 ≤ Quality ≤ 0.0`

#### Description

**DoOcvSimple** evaluates the pattern in **(Pattern)**. Before the evaluation the good-pattern has be trained by using the operator **TrainDoOcvProj**. Both patterns should have roughly the same (relative) extent and shape. To specify which of the trained patterns is used as reference its name is specified in **PatternName**. The next four parameters influence the automatic adaption: **AdaptPos** and **AdaptSize** refer to the geometry of the pattern. **AdaptPos** specifies whether a shift of the position will be automatically. **AdaptAngle** is not yet implemented. The parameter **AdaptGray** controls the adaption to changes of the gray values. This comprises additive and multiplicative changes of the intensity.

The parameter **Threshold** specifies the minimum difference of the gray values to be treated as an error. In this case the percentage of wrong pixels is returned. If the value is below 0 the sum of all errors normalized with respect to the size is returned.

The result of the operator is the **Quality** of the pattern with a value between 0 and 1. The value 1 corresponds to a pattern with no faults. The value 0 corresponds to a very big fault.

### Examples

```c
[ out ] VARIANT Quality HImageX.DoOcvSimple ([ in ] HOcvX OCVHandle,
[ in ] VARIANT PatternName, [ in ] String AdaptPos, [ in ] String AdaptSize,
```

```c
[ out ] VARIANT Quality HOcvX.DoOcvSimple ([ in ] HImageX Pattern,
[ in ] VARIANT PatternName, [ in ] String AdaptPos, [ in ] String AdaptSize,
```
**DoOcvSimple** returns TRUE, if the handle and the characters are correct. Otherwise, an exception handling is raised.

**Parallelization Information**

**DoOcvSimple** is reentrant and processed without parallelization.

**Possible Predecessors**

TrainOcrClassBox, TrainfOcrClassBox, ReadOcv, Threshold, Connection, SelectShape

**Possible Successors**

CloseOcv

**See also**

CreateOcvProj

**Module**

Optical character verification

```c
void HOcvX.ReadOcv ([in] String FileName )

void HOperatorSetX.ReadOcv ([in] VARIANT FileName, [out] VARIANT OCVHandle )
```

**Reading an OCV tool from file.**

**ReadOcv** reads an OCV tool from file. The tool will contain the same information that it contained when saving it with **WriteOcv**. After reading the tool the training can be completed for those patterns which have not been trained so far. Otherwise a pattern comparison can be applied directly by calling **DoOcvSimple**.

As extension `.ocv` is used. If this extension is not given with the file name it will be added automatically.

**Parameter**

- **FileName** (input control) : Name of the file which has to be read.
  
  Default Value : 'test.ocv'

- **OCVHandle** (output control) : Handle of read OCV tool.

**Result**

**ReadOcv** returns TRUE, if the file is correct. If the file could not be opened **ReadOcv** returns FAIL. Otherwise, an exception handling is raised.

**Parallelization Information**

**ReadOcv** is processed completely exclusively without parallelization.

**Possible Predecessors**

**WriteOcv**

**Possible Successors**

**DoOcvSimple, CloseOcv**

**See also**

**ReadOcr**

**Module**

Optical character verification
Training of an OCV tool.

TraindOcvProj trains patterns for an OCV tool that has been created using the operators CreateOcvProj or ReadOcv. For this training one or multiple patterns are offered the system. Such a pattern consists of an image with a reduced domain (ROI) for the area of the pattern. Note that the pattern should not only contain foreground pixels (e.g. dark pixels of a character) but also background pixels. This can be implemented e.g. by the smallest surrounding rectangle of the pattern. Without this context an evaluation of the pattern is not possible.

If more than one pattern has to be trained this can be achieved by multiple calls (one for each pattern) or by calling TraindOcvProj once with all patterns and a tuple of the corresponding names. The result will be in both cases the same. However using multiple calls will normally result in a longer execution time than using one call with all patterns.

### Parameter

- **Pattern** (input iconic) ................................................. image(-array) \(\sim\) HImageX / IHObjectX (byte)
  - Pattern to be trained.
- **OCVHandle** (input control) ........................................... ocv \(\sim\) HOcvX / VARIANT
  - Handle of the OCV tool to be trained.
- **Name** (input control) ............................................... string(-array) \(\sim\) VARIANT (string)
  - Name(s) of the object(s) to analyse.
  - **Default Value**: 'a'
- **Mode** (input control) ................................................. string \(\sim\) String / VARIANT
  - Mode for training (only one mode implemented).
  - **Default Value**: 'single'
  - **List of values**: Mode \(\in\) { 'single' }

### Result

TraindOcvProj returns TRUE, if the handle and the training pattern(s) are correct. Otherwise, an exception handling is raised.

### Parallelization Information

TraindOcvProj is processed completely exclusively without parallelization.

### Possible Predecessors

WriteOcrTrainf, CreateOcvProj, ReadOcv, Threshold, Connection, SelectShape

### Possible Successors

CloseOcv

### See also

TraindOcrClassBox

### Module

Optical character verification

---

**Saving an OCV tool to file.**

WriteOcv writes an OCV tool to file. This can be used to save the result of a training (TraindOcvProj). The whole information contained in the OCV tool is stored in the file. The file can be reloaded afterwards using the operator ReadOcv.
As file extension `.ocv` is used. If this extension is not given with the file name, it will be added automatically.

Parameter

- **OCVHandle** (input control) .................. ocv \(\Rightarrow\) HObjectX / VARIANT
  Handle of the OCV tool to be written.
- **FileName** (input control) .................. filename \(\Rightarrow\) String / VARIANT
  Name of the file where the tool has to be saved.

Default Value: ‘test.ocr’

Result

WriteOcv returns TRUE, if the data is correct and the file can be written. Otherwise, an exception handling is raised.

Parallelization Information

WriteOcv is reentrant and processed without parallelization.

Possible Predecessors

TrainOcvProj

Possible Successors

CloseOcv

See also

WriteOcr

Module

Optical character verification

### 13.14 Shape-from

```
[out] HImageX Depth
```

```
```

Extract depth using multiple focus levels.

The operator **DepthFromFocus** extracts the depth using a focus sequence. The images of the focus sequence have to passed as a multi channel image (MultiFocusImage). The depth for each pixel will be returned in **Depth** as the channel number. The parameter **Confidence** returns a confidence value for each depth estimation: The larger this value, the higher the confidence of the depth estimation is.

**DepthFromFocus** selects the pixels with the best focus of all focus levels. The method used to extract these pixels is specified by the parameters **Filter** and **Selection**:

- **highpass** The value of the focus is estimated by a highpass filter.
- **bandpass** The value of the focus is estimated by a bandpass filter.
- **next_maximum** To decide which focus level has be selected, the pixel in the neighborhood with the best confidence is used to determine this value.
- **local** The decision for a focus level is based only on the locally calculated focus values.

Parameter

- **MultiFocusImage** (input iconic) ........... multichannel-image-array \(\Rightarrow\) HImageX / HObjectX ( byte )
  Multichannel gray image consisting of multiple focus levels.
- **Depth** (output iconic) ....................... image(-array) \(\Rightarrow\) HImageX / HUntypedObjectX ( byte )
  Depth image.
- **Confidence** (output iconic) ................... image(-array) \(\Rightarrow\) HImageX / HUntypedObjectX ( byte )
  Confidence of depth estimation.
Filter (input control) ……………………………………… string(-array) ～ VARIANT ( string )
Filter used to find sharp pixels.
Default Value: ’highpass’
List of values: Filter ∈ {’highpass’, ’bandpass’}

Selection (input control) ……………………………………… string(-array) ～ VARIANT ( string )
Method used to find sharp pixels.
Default Value: ’next_maximum’
List of values: Selection ∈ {’next_maximum’, ’local’}

DepthFromFocus is reentrant and automatically parallelized (on tuple level).

Compose2, Compose3, Compose4, AddChannels, ReadImage, ReadSequence

SelectGrayvaluesFromChannels, MeanImage, GaussImage, Threshold

CountChannels

Estimate the albedo of a surface and the amount of ambient light.

EstimateAlAm estimates the Albedo of a surface, i.e. the percentage of light reflected by the surface, and the amount of ambient light Ambient by using the maximum and minimum gray values of the image.

Attention
It is assumed that the image contains at least one point for which the reflection function assumes its minimum, e.g., points in shadows. Furthermore, it is assumed that the image contains at least one point for which the reflection function assumes its maximum. If this is not the case, wrong values will be estimated.

Parameter

Image (input iconic) ……………………………………… image(-array) ～ HImageX / IObjectX ( byte )
Image for which albedo and ambient are to be estimated.

Albedo (output control) ……………………………………… real(-array) ～ VARIANT ( real )
Amount of light reflected by the surface.

Ambient (output control) ……………………………………… real(-array) ～ VARIANT ( real )
Amount of ambient light.

Result
EstimateAlAm always returns the value TRUE.

EstimateAlAm is reentrant and automatically parallelized (on tuple level).

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools

Estimate the slant of a light source and the albedo of a surface.

EstimateSlAlLr estimates the Slant of a light source and the Albedo of a surface by using the maximum and minimum gray values of the image.

Attention
It is assumed that the image contains at least one point for which the reflection function assumes its minimum, e.g., points in shadows. Furthermore, it is assumed that the image contains at least one point for which the reflection function assumes its maximum. If this is not the case, wrong values will be estimated.

Parameter

Image (input iconic) ……………………………………… image(-array) ～ HImageX / IObjectX ( byte )

Slant (output control) ……………………………………… real(-array) ～ VARIANT ( real )
Amount of light reflected by the surface.

Albedo (output control) ……………………………………… real(-array) ～ VARIANT ( real )
Amount of ambient light.

Result
EstimateSlAlLr always returns the value TRUE.

EstimateSlAlLr is reentrant and automatically parallelized (on tuple level).

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools
EstimateSlAlLr estimates the slant of a light source, i.e., the angle between the light source and the positive z-axis, and the albedo of the surface in the input image Image, i.e. the percentage of light reflected by the surface, using the algorithm of Lee and Rosenfeld.

Attention
The Albedo is assumed constant for the entire surface depicted in the image.

Parameter

- **Image** (input iconic) ........................................ image(-array) ~ HImageX / IHObjectX (byte)
  Image for which slant and albedo are to be estimated.
- **Slant** (output control) ........................................... angle.deg(-array) ~ VARIANT (real)
  Angle between the light sources and the positive z-axis (in degrees).
- **Albedo** (output control) ........................................ real(-array) ~ VARIANT (real)
  Amount of light reflected by the surface.

Result
EstimateSlAlLr always returns the value TRUE.

Parallelization Information
EstimateSlAlLr is reentrant and automatically parallelized (on tuple level).

Possible Successors
SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools

```
[out] VARIANT Slant HImageX.EstimateSlAlZc ([out] VARIANT Albedo )
```

Estimate the slant of a light source and the albedo of a surface.

EstimateSlAlZc estimates the slant of a light source, i.e. the angle between the light source and the positive z-axis, and the albedo of the surface in the input image Image, i.e. the percentage of light reflected by the surface, using the algorithm of Zheng and Chellappa.

Attention
The Albedo is assumed constant for the entire surface depicted in the image.

Parameter

- **Image** (input iconic) ........................................ image(-array) ~ HImageX / IHObjectX (byte)
  Image for which slant and albedo are to be estimated.
- **Slant** (output control) ........................................... angle.deg(-array) ~ VARIANT (real)
  Angle of the light sources and the positive z-axis (in degrees).
- **Albedo** (output control) ........................................ real(-array) ~ VARIANT (real)
  Amount of light reflected by the surface.

Result
EstimateSlAlZc always returns the value TRUE.

Parallelization Information
EstimateSlAlZc is reentrant and automatically parallelized (on tuple level).

Possible Successors
SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

Tools
OutVARIANT Tilt HImageX.EstimateTiltLr ( )

void HOperatorSetX.EstimateTiltLr ([in] IHObjectX Image, [out] VARIANT Tilt )

Estimate the tilt of a light source.

**HImageX.EstimateTiltLr** estimates the tilt of a light source, i.e. the angle between the light source and the x-axis after projection into the xy-plane, from the image **Image** using the algorithm of Lee and Rosenfeld.

**Parameter**

- **Image** (input iconic) .......................................... image(-array) $\rightarrow$ HImageX / IHObjectX ( byte )
  Image for which the tilt is to be estimated.
- **Tilt** (output control) .............................................. angle.deg(-array) $\rightarrow$ VARIANT ( real )
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

**Result**

**HImageX.EstimateTiltLr** always returns the value TRUE.

**Parallelization Information**

**HImageX.EstimateTiltLr** is reentrant and automatically parallelized (on tuple level).

**Possible Successors**

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

**Tools**

OutVARIANT Tilt HImageX.EstimateTiltZc ( )

void HOperatorSetX.EstimateTiltZc ([in] IHObjectX Image, [out] VARIANT Tilt )

Estimate the tilt of a light source.

**HImageX.EstimateTiltZc** estimates the tilt of a light source, i.e. the angle between the light source and the x-axis after projection into the xy-plane, from the image **Image** using the algorithm of Zheng and Chellappa.

**Parameter**

- **Image** (input iconic) .......................................... image(-array) $\rightarrow$ HImageX / IHObjectX ( byte )
  Image for which the tilt is to be estimated.
- **Tilt** (output control) .............................................. angle.deg(-array) $\rightarrow$ VARIANT ( real )
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

**Result**

**HImageX.EstimateTiltZc** always returns the value TRUE.

**Parallelization Information**

**HImageX.EstimateTiltZc** is reentrant and automatically parallelized (on tuple level).

**Possible Successors**

SfsModLr, SfsOrigLr, SfsPentland, PhotStereo, ShadeHeightField

**Tools**

OutHImageX Height HImageX.PhotStereo ([in] VARIANT Slants, [in] VARIANT Tilts )


Reconstruct a surface from at least three gray value images.
**PhotStereo** reconstructs a surface (i.e., the relative height of each image point) using the algorithm of Woodham from at least three gray value images given by the multi-channel image **Images**. The light sources corresponding to the individual images are given by the parameters **Slants** and **Tilts** and are assumed to lie infinitely far away.

**Attention**

**PhotStereo** assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied by the step width after the call to **PhotStereo**. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with an edge length being a power of 2 are accepted. All given images must be byte-images. At least three images must be given in a multi-channel image. **Slants** and **Tilts** must contain exactly as many light sources as the number of channels in **Images**. At least three of the light source directions must be linearly independent.

**Parameter**

- **Images** (input iconic) .................................. (multichannel-)image  \( \leadsto \text{HImageX} / \text{IHObjectX} \) (byte)
  Shaded input image with at least three channels.

- **Height** (output iconic) .................................. image  \( \leadsto \text{HImageX} / \text{HUntypedObjectX} \) (real)
  Reconstructed height field.

- **Slants** (input control) .................................. angle.deg  \( \leadsto \text{VARIANT} \) (real, integer)
  Angle between the light sources and the positive z-axis (in degrees).
  - Default Value : 45.0
  - Suggested values : \( \text{Slants} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\} \)
  - Typical range of values : \( 0.0 \leq \text{Slants} \leq 0.0 \) (lin)
  - Minimum Increment : 0.01
  - Recommended Increment : 10.0

- **Tilts** (input control) .................................. angle.deg  \( \leadsto \text{VARIANT} \) (real, integer)
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  - Default Value : 45.0
  - Suggested values : \( \text{Tilts} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\} \)
  - Typical range of values : \( 0.0 \leq \text{Tilts} \leq 0.0 \) (lin)
  - Minimum Increment : 0.01
  - Recommended Increment : 10.0

**Result**

If all parameters are correct **PhotStereo** returns the value TRUE. Otherwise, an exception is raised.

**Parallelization Information**

**PhotStereo** is reentrant and processed without parallelization.

**Possible Predecessors**

EstimateSlAllr, EstimateSlAllzc, EstimateTiltLr, EstimateTiltZc

**Possible Successors**

ShadeHeightField

**Tools**

Selection of gray values of a multi channel image using an index image.

The operator **SelectGrayvaluesFromChannels** selects gray values from a **MultichannelImage**. The channel number for each pixel is determined by using **IndexImage**: The gray value in **IndexImage** is used as the channel number in **MultichannelImage**.

---

HALCON/COM Reference Manual, 2005-2-1
13.14. SHAPE-FROM

Parameter

- **MultichannelImage** (input iconic) \(\ldots \), multichannel-image-array \(\sim\) \(\text{HImageX} / \text{IObjectX} \) (byte)
  Multichannel gray image.
- **IndexImage** (input iconic) \(\ldots\), image(-array) \(\sim\) \(\text{HImageX} / \text{IObjectX} \) (byte)
  Image, where gray values are interpreted as indexes.
- **Selected** (output iconic) \(\ldots\), image(-array) \(\sim\) \(\text{HImageX} / \text{HUntypedObjectX} \) (byte)
  Depth image.

Parallelization Information

SelectGrayvaluesFromChannels is reentrant and automatically parallelized (on tuple level, domain level).

Possible Predecessors

DepthFromFocus, MeanImage

Possible Successors

DispImage

See also

CountChannels

Module

Tools

```
[out] \text{HImageX} \text{Height} \text{HImageX.SfsModLr} \ ([in] \text{VARIANT Slant,}
[in] \text{VARIANT Tilt,} [in] \text{VARIANT Albedo,} [in] \text{VARIANT Ambient})

void \text{HOperatorSetX.SfsModLr} \ ([in] \text{IObjectX Image,}
[out] \text{HUntypedObjectX Height,} [in] \text{VARIANT Slant,} [in] \text{VARIANT Tilt,}
[in] \text{VARIANT Albedo,} [in] \text{VARIANT Ambient})
```

Reconstruct a surface from a gray value image.

**SfsModLr** reconstructs a surface (i.e. the relative height of each image point) using the modified algorithm of Lee and Rosenfeld. The surface is reconstructed from the input image **Image**, and the light source given by the parameters **Slant**, **Tilt**, **Albedo** and **Ambient**, and is assumed to lie infinitely far away in the direction given by **Slant** and **Tilt**. The parameter **Albedo** determines the albedo of the surface, i.e. the percentage of light reflected in all directions. **Ambient** determines the amount of ambient light falling onto the surface. It can be set to values greater than zero if, for example, the white balance of the camera was badly adjusted at the moment the image was taken.

Attention

**SfsModLr** assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied with the step width after the call to **SfsModLr**. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with the edge length being a power of 2 are accepted. **SfsModLr** can only handle byte-images.

Parameter

- **Image** (input iconic) \(\ldots\), image(-array) \(\sim\) \(\text{HImageX} / \text{IObjectX} \) (byte)
  Shaded input image.
- **Height** (output iconic) \(\ldots\), image(-array) \(\sim\) \(\text{HImageX} / \text{HUntypedObjectX} \) (real)
  Reconstructed height field.
- **Slant** (input control) \(\ldots\), angle.deg \(\sim\) \text{VARIANT} (real, integer)
  Angle between the light source and the positive z-axis (in degrees).

Default Value : 45.0

Suggested values : \(\text{Slant} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)

Typical range of values : \(0.0 \leq \text{Slant} \leq 0.0\text{(lin)}\)

Minimum Increment : 0.01

Recommended Increment : 10.0
> **Tilt** (input control) .......................... angle.deg  \(\sim\) VARIANT (real, integer)

Angle between the light source and the x-axis after projection into the xy-plane (in degrees).

- **Default Value**: 45.0
- **Suggested values**: \(\text{Tilt} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
- **Typical range of values**: \(0.0 \leq \text{Tilt} \leq 0.0\) (lin)
- **Minimum Increment**: 0.01
- **Recommended Increment**: 10.0

> **Albedo** (input control) .......................... number  \(\sim\) VARIANT (real, integer)

Amount of light reflected by the surface.

- **Default Value**: 1.0
- **Suggested values**: \(\text{Albedo} \in \{0.1, 0.5, 1.0, 5.0\}\)
- **Typical range of values**: \(0.0 \leq \text{Albedo} \leq 0.0\) (lin)
- **Minimum Increment**: 0.01
- **Recommended Increment**: 0.1
- **Restriction**: \((\text{Albedo} \geq 0.0)\)

> **Ambient** (input control) .......................... number  \(\sim\) VARIANT (real, integer)

Amount of ambient light.

- **Default Value**: 0.0
- **Suggested values**: \(\text{Ambient} \in \{0.1, 0.5, 1.0\}\)
- **Typical range of values**: \(0.0 \leq \text{Ambient} \leq 0.0\) (lin)
- **Minimum Increment**: 0.01
- **Recommended Increment**: 0.1
- **Restriction**: \((\text{Ambient} \geq 0.0)\)

**Result**

If all parameters are correct **SfsModLr** returns the value TRUE. Otherwise, an exception is raised.

**Parallelization Information**

**SfsModLr** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

EstimateAlAm, EstimateSlAlLr, EstimateSlAlZc, EstimateTiltLr, EstimateTiltZc

**Possible Successors**

ShadeHeightField

**Tools**

```plaintext
```


Reconstruct a surface from a gray value image.

**SfsOrigLr** reconstructs a surface (i.e. the relative height of each image point) using the original algorithm of Lee and Rosenfeld. The surface is reconstructed from the input image **Image**. The light source is to be given by the parameters **Slant**, **Tilt**, **Albedo** and **Ambient**, and is assumed to lie infinitely far away in the direction given by **Slant** and **Tilt**. The parameter **Albedo** determines the albedo of the surface, i.e. the percentage of light reflected in all directions. **Ambient** determines the amount of ambient light falling onto the surface. It can be set to values greater than zero if, for example, the white balance of the camera was badly adjusted at the moment the image was taken.

**Attention**

**SfsOrigLr** assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied with the step width after the call to **SfsOrigLr**. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with the edge length being a power of 2 are accepted. **SfsOrigLr** can only handle byte-images.
13.14. SHAPE-FROM

- **Image** (input iconic) ......................... image(-array) $\sim HImageX / IOBJECTX$ (byte)
  Shaded input image.

- **Height** (output iconic) ...................... image(-array) $\sim HImageX / HUntypedObjectX$ (real)
  Reconstructed height field.

- **Slant** (input control) ....................... angle.deg $\sim$ VARIANT (real, integer)
  Angle between the light source and the positive z-axis (in degrees).
  Default Value: 45.0
  Suggested values: Slant $\in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}$
  Typical range of values: $0.0 \leq$ Slant $\leq 0.0$ (lin)
  Minimum Increment: 0.01
  Recommended Increment: 10.0

- **Tilt** (input control) ....................... angle.deg $\sim$ VARIANT (real, integer)
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  Default Value: 45.0
  Suggested values: Tilt $\in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}$
  Typical range of values: $0.0 \leq$ Tilt $\leq 0.0$ (lin)
  Minimum Increment: 0.01
  Recommended Increment: 10.0

- **Albedo** (input control) ..................... number $\sim$ VARIANT (real, integer)
  Amount of light reflected by the surface.
  Default Value: 1.0
  Suggested values: Albedo $\in \{0.1, 0.5, 1.0, 5.0\}$
  Typical range of values: $0.0 \leq$ Albedo $\leq 0.0$ (lin)
  Minimum Increment: 0.01
  Recommended Increment: 0.1
  Restriction: ($Albedo \geq 0.0$)

- **Ambient** (input control) .................... number $\sim$ VARIANT (real, integer)
  Amount of ambient light.
  Default Value: 0.0
  Suggested values: Ambient $\in \{0.1, 0.5, 1.0\}$
  Typical range of values: $0.0 \leq$ Ambient $\leq 0.0$ (lin)
  Minimum Increment: 0.01
  Recommended Increment: 0.1
  Restriction: ($Ambient \geq 0.0$)

---

If all parameters are correct $SfsOrigLr$ returns the value TRUE. Otherwise, an exception is raised.

---

Parallelization Information

$SfsOrigLr$ is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

EstimateAlAm, EstimateSlAlLr, EstimateSlAlZc, EstimateTiltLr, EstimateTiltZc

Possible Successors

ShadeHeightField

---

Module

---

Tools

```c
[out] HImageX Height HImageX.SfsPentland ([in] VARIANT Slant,
```

```c
void HOperatorSetX.SfsPentland ([in] IOBJECTX Image,
[out] HUntypedObjectX Height, [in] VARIANT Slant, [in] VARIANT Tilt,
```

Reconstruct a surface from a gray value image.

SfsPentland reconstructs a surface (i.e. the relative height of each image point) using the algorithm of Pentland.

The surface is reconstructed from the input image Image. The light source must be given by the parameters

HALCON 6.1.4
Slant, Tilt, Albedo, and Ambient, and is assumed to lie infinitely far away in the direction given by Slant and Tilt. The parameter Albedo determines the albedo of the surface, i.e. the percentage of light reflected in all directions. Ambient determines the amount of ambient light falling onto the surface. It can be set to values greater than zero if, for example, the white balance of the camera was badly adjusted at the moment the image was taken.

Attention

SfsPentland assumes that the heights are to be extracted on a lattice with step width 1. If this is not the case, the calculated heights must be multiplied with the step width after the call to SfsPentland. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally. Since the operator is based on the Fast Fourier Transform, only square images with the edge length being a power of 2 are accepted. SfsPentland can only handle byte-images.

## Parameter

- **Image** (input iconic) .......................... image(-array) \(\sim HImageX / HObjectX\) (byte)  
Shaded input image.

- **Height** (output iconic) ......................... image(-array) \(\sim HImageX / HUntypedObjectX\) (real)  
Reconstructed height field.

- **Slant** (input control) ......................... angle.deg \(\sim\) VARIANT (real, integer)  
Angle between the light source and the positive z-axis (in degrees).
  - **Default Value**: 45.0
  - **Suggested values**: \(\text{Slant} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
  - **Typical range of values**: \(0.0 \leq \text{Slant} \leq 0.0\) (lin)
  - **Minimum Increment**: 1.0
  - **Recommended Increment**: 10.0

- **Tilt** (input control) .......................... angle.deg \(\sim\) VARIANT (real, integer)  
Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  - **Default Value**: 45.0
  - **Suggested values**: \(\text{Tilt} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
  - **Typical range of values**: \(0.0 \leq \text{Tilt} \leq 0.0\) (lin)
  - **Minimum Increment**: 1.0
  - **Recommended Increment**: 10.0

- **Albedo** (input control) ....................... number \(\sim\) VARIANT (real, integer)  
Amount of light reflected by the surface.
  - **Default Value**: 1.0
  - **Suggested values**: \(\text{Albedo} \in \{0.1, 0.5, 1.0, 5.0\}\)
  - **Typical range of values**: \(0.0 \leq \text{Albedo} \leq 0.0\) (lin)
  - **Minimum Increment**: 0.01
  - **Recommended Increment**: 0.1
  - **Restriction**: \((\text{Albedo} \geq 0.0)\)

- **Ambient** (input control) ..................... number \(\sim\) VARIANT (real, integer)  
Amount of ambient light.
  - **Default Value**: 0.0
  - **Suggested values**: \(\text{Ambient} \in \{0.1, 0.5, 1.0\}\)
  - **Typical range of values**: \(0.0 \leq \text{Ambient} \leq 0.0\) (lin)
  - **Minimum Increment**: 0.01
  - **Recommended Increment**: 0.1
  - **Restriction**: \((\text{Ambient} \geq 0.0)\)

## Result

If all parameters are correct SfsPentland returns the value TRUE. Otherwise, an exception is raised.

Parallelization Information

SfsPentland is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- EstimateAlAm, EstimateSlAlLr, EstimateSlAlZc, EstimateTiltLr, EstimateTiltZc

Possible Successors

ShadeHeightField

Tools

HALCON/COM Reference Manual, 2005-2-1
Shade a height field.

ShadeHeightField computes a shaded image from the height field ImageHeight as if the image were illuminated by an infinitely far away light source. It is assumed that the surface described by the height field has Lambertian reflection properties determined by Albedo and Ambient. The parameter Shadows determines whether shadows are to be calculated.

Attention

ShadeHeightField assumes that the heights are given on a lattice with step width 1. If this is not the case, the heights must be divided by the step width before the call to ShadeHeightField. Otherwise, the derivatives used internally to compute the orientation of the surface will be estimated to steep or too flat. Example: The height field is given on 100*100 points on the square [0,1]*[0,1]. Then the heights must be divided by 1/100 first. A Cartesian coordinate system with the origin in the lower left corner of the image is used internally.

Parameter

- **ImageHeight** (input iconic) .................. image(-array)  \(\sim\) HImageX / IHObjectX (byte, int4, real)
  Height field to be shaded.
- **ImageShade** (output iconic) .................. image(-array)  \(\sim\) HImageX / HUntypedObjectX (byte)
  Shaded image.
- **Slant** (input control) ...................... angle.deg  \(\sim\) VARIANT (real, integer)
  Angle between the light source and the positive z-axis (in degrees).
  Default Value: 0.0
  Suggested values: \(\text{Slant} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
  Typical range of values: \(0.0 \leq \text{Slant} \leq 0.0\) (lin)
  Minimum Increment: 0.01
  Recommended Increment: 10.0
- **Tilt** (input control) ...................... angle.deg  \(\sim\) VARIANT (real, integer)
  Angle between the light source and the x-axis after projection into the xy-plane (in degrees).
  Default Value: 0.0
  Suggested values: \(\text{Tilt} \in \{1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0\}\)
  Typical range of values: \(0.0 \leq \text{Tilt} \leq 0.0\) (lin)
  Minimum Increment: 0.01
  Recommended Increment: 10.0
- **Albedo** (input control) ...................... number  \(\sim\) VARIANT (real, integer)
  Amount of light reflected by the surface.
  Default Value: 1.0
  Suggested values: \(\text{Albedo} \in \{0.1, 0.5, 1.0, 5.0\}\)
  Typical range of values: \(0.0 \leq \text{Albedo} \leq 0.0\) (lin)
  Minimum Increment: 0.01
  Recommended Increment: 0.1
  Restriction: \((\text{Albedo} \geq 0.0)\)
- **Ambient** (input control) ...................... number  \(\sim\) VARIANT (real, integer)
  Amount of ambient light.
  Default Value: 0.0
  Suggested values: \(\text{Ambient} \in \{0.1, 0.5, 1.0\}\)
  Typical range of values: \(0.0 \leq \text{Ambient} \leq 0.0\) (lin)
  Minimum Increment: 0.01
  Recommended Increment: 0.1
  Restriction: \((\text{Ambient} \geq 0.0)\)
章 13. 工具

Shadows (input control) ................................. string ～ String / VARIANT
Should shadows be calculated?
Default Value: 'false'
Suggested values: Shadows ∈ {'true', 'false'}

If all parameters are correct ShadeHeightField returns the value TRUE. Otherwise, an exception is raised.

ShadeHeightField is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
SfsModLr, SfsOrigLr, SfsPentland, PhotStereo

Tools
Chapter 14

Tuple

14.1 Arithmetic

\[
\begin{align*}
\text{[out]} \ \text{VARIANT Abs} & \ 	ext{HTupleX.TupleAbs} \ ([\text{in}] \ \text{VARIANT T}) \\
\text{void} & \ 	ext{HOperatorSetX.TupleAbs} \ ([\text{in}] \ \text{VARIANT T}, \ [\text{out}] \ \text{VARIANT Abs})
\end{align*}
\]

Compute the absolute value of a tuple.

\text{TupleAbs} computes the absolute value of the input tuple \text{T}. The absolute value of an integer number is again an integer number. The absolute value of a floating point number is a floating point number. The absolute value of a string is not allowed.

\textbf{Parameter}

\begin{itemize}
\item \text{T} (input control) \text{number(-array)} \leadsto \text{VARIANT} (\text{real, integer}) \ 
Input tuple.
\item \text{Abs} (output control) \text{number(-array)} \leadsto \text{VARIANT} (\text{real, integer}) \ 
Absolute value of the input tuple.
\end{itemize}

\textbf{Parallelization Information}

\text{TupleAbs} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Alternatives}

\textbf{Module}

Operators not requiring licensing

\[
\begin{align*}
\text{[out]} \ \text{VARIANT ACos} & \ 	ext{HTupleX.TupleAcos} \ ([\text{in}] \ \text{VARIANT T}) \\
\text{void} & \ 	ext{HOperatorSetX.TupleAcos} \ ([\text{in}] \ \text{VARIANT T}, \ [\text{out}] \ \text{VARIANT ACos})
\end{align*}
\]

Compute the arccosine of a tuple.

\text{TupleAcos} computes the arccosine of the input tuple \text{T}. The arccosine is always returned as a floating point number. The arccosine of a string is not allowed.

\textbf{Parameter}

\begin{itemize}
\item \text{T} (input control) \text{number(-array)} \leadsto \text{VARIANT} (\text{real, integer}) \ 
Input tuple.
\item \text{ACos} (output control) \text{number(-array)} \leadsto \text{VARIANT} (\text{real}) \ 
Arccosine of the input tuple.
\end{itemize}

\textbf{Parallelization Information}

\text{TupleAcos} is \textit{reentrant} and processed \textit{without} parallelization.
Add two tuples.

**TupleAdd** computes the sum of the input tuples \( S1 \) and \( S2 \). If both tuples have the same length the corresponding elements of both tuples are added. Otherwise, either \( S1 \) or \( S2 \) must have length 1. In this case, the addition is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are added, the result is again an integer number. If a floating point number is added to another number, the result is a floating point number. If two strings are added, the addition corresponds to a string concatenation. If a number and a string are added, the number is converted to a string first. Thus, the addition also corresponds to a string concatenation in this case.

**Parameter**

- \( S1 \) (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \( \sim \) VARIANT (real, integer, string)
  Input tuple 1.
- \( S2 \) (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \( \sim \) VARIANT (real, integer, string)
  Input tuple 2.
- \( Sum \) (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \( \sim \) VARIANT (real, integer, string)
  Sum of the input tuples.

**Parallelization Information**

**TupleAdd** is reentrant and processed without parallelization.

**TupleSub**

Operators not requiring licensing

---

**See also**

**TupleSin**

Compute the arcsine of a tuple.

**TupleAsin** computes the arcsine of the input tuple \( T \). The arcsine is always returned as a floating point number. The arcsine of a string is not allowed.

**Parameter**

- \( T \) (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \( \sim \) VARIANT (real, integer)
  Input tuple.
  **Restriction**: \((-1 \leq T \leq 1)\)
- \( Asin \) (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \( \sim \) VARIANT (real)
  Arcsine of the input tuple.

**Parallelization Information**

**TupleAsin** is reentrant and processed without parallelization.

**See also**

**TupleSin**
14.1. ARITHMETIC

Alternatives

TupleAcos, TupleAtan, TupleAtan2

Module

Operators not requiring licensing

[out] VARIANT ATan HTupleX.TupleAtan ([in] VARIANT T )

void HOperatorSetX.TupleAtan ([in] VARIANT T, [out] VARIANT ATan )

Compute the arctangent of a tuple.

TupleAtan computes the arctangent of the input tuple T. The arctangent is always returned as a floating point number. The arctangent of a string is not allowed.

Parameter

▷ T (input control) ........................................... number(-array)  VARIANT ( real, integer )
Input tuple.

▷ ATan (output control) ...........................................number(-array)  VARIANT ( real )
Arctangent of the input tuple.

Parallelization Information

TupleAtan is reentrant and processed without parallelization.

See also

TupleTan

Alternatives

TupleAtan2, TupleAsin, TupleAcos

Module

Operators not requiring licensing


void HOperatorSetX.TupleAtan2 ([in] VARIANT Y, [in] VARIANT X, 
[out] VARIANT ATan )

Compute the arctangent of a tuple for all four quadrants.

TupleAtan2 computes the arctangent of the input tuples Y/X while treating all four quadrants correctly. The arctangent is always returned as a floating point number. The arctangent of a string is not allowed.

Parameter

▷ Y (input control) ........................................... number(-array)  VARIANT ( real, integer )
Input tuple of the y-values.

▷ X (input control) ........................................... number(-array)  VARIANT ( real, integer )
Input tuple of the x-values.

▷ ATan (output control) ...........................................number(-array)  VARIANT ( real )
Arctangent of the input tuple.

Parallelization Information

TupleAtan2 is reentrant and processed without parallelization.

See also

TupleTan

Alternatives

TupleAtan, TupleAsin, TupleAcos

Module

Operators not requiring licensing
Compute the ceiling function of a tuple.

`TupleCeil` computes the ceiling function of the input tuple \( T \), i.e., the smallest integer greater than or equal to \( T \). The ceiling function is always returned as a floating point number. The ceiling function of a string is not allowed.

**Parameter**

- \( T \) (input control) ................. number(-array) \( \rightarrow \) VARIANT ( real, integer )
  Input tuple.
- `Ceil` (output control) .................. number(-array) \( \rightarrow \) VARIANT ( real )
  Ceiling function of the input tuple.

**Parallelization Information**

`TupleCeil` is reentrant and processed without parallelization.

**Alternatives**

Operators not requiring licensing

---

Compute the cosine of a tuple.

`TupleCos` computes the cosine of the input tuple \( T \). The cosine is always returned as a floating point number. The cosine of a string is not allowed.

**Parameter**

- \( T \) (input control) ................. number(-array) \( \rightarrow \) VARIANT ( real, integer )
  Input tuple.
- `Cos` (output control) .................. number(-array) \( \rightarrow \) VARIANT ( real )
  Cosine of the input tuple.

**Parallelization Information**

`TupleCos` is reentrant and processed without parallelization.

**See also**

`TupleAcos`

**Alternatives**

`TupleSin, TupleTan`

Operators not requiring licensing

---

Compute the hyperbolic cosine of a tuple.

`TupleCosh` computes the hyperbolic cosine of the input tuple \( T \). The hyperbolic cosine is always returned as a floating point number. The hyperbolic cosine of a string is not allowed.
14.1. ARITHMETIC

Parameter

- \( T \) (input control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real, integer)
  Input tuple.
- \( \text{Cosh} \) (output control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real)
  Hyperbolic cosine of the input tuple.

Parallelization Information

TupleCosh is reentrant and processed without parallelization.

Alternatives

TupleSinh, TupleTanh

Module

Operators not requiring licensing

\begin{verbatim}
[out] VARIANT Quot HTupleX.TupleDiv ([in] VARIANT Q1, [in] VARIANT Q2 )
void HOperatorSetX.TupleDiv ([in] VARIANT Q1, [in] VARIANT Q2, 
[out] VARIANT Quot )
\end{verbatim}

Divide two tuples.

TupleDiv computes the quotient of the input tuples \( Q1 \) and \( Q2 \). If both tuples have the same length the corresponding elements of both tuples are divided. Otherwise, either \( Q1 \) or \( Q2 \) must have length 1. In this case, the division is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are divided, the result is again an integer number. If one of the operands is a floating point number, the result is a floating point number. The division of strings is not allowed.

Parameter

- \( Q1 \) (input control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real, integer)
  Input tuple 1.
- \( Q2 \) (input control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real, integer)
  Input tuple 2.
  Restriction: \( (Q2 \neq 0) \)
- \( \text{Quot} \) (output control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real, integer)
  Quotient of the input tuples.

Parallelization Information

TupleDiv is reentrant and processed without parallelization.

Alternatives

TupleMult

Module

Operators not requiring licensing

\begin{verbatim}
[out] VARIANT Exp HTupleX.TupleExp ([in] VARIANT T )
void HOperatorSetX.TupleExp ([in] VARIANT T, [out] VARIANT Exp )
\end{verbatim}

Compute the exponential of a tuple.

TupleExp computes the exponential of the input tuple \( T \). The exponential is always returned as a floating point number. The exponential of a string is not allowed.

Parameter

- \( T \) (input control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real, integer)
  Input tuple.
- \( \text{Exp} \) (output control) \( \Rightarrow \) number(-array) \( \sim \) VARIANT (real)
  Exponential of the input tuple.
TupleExp is reentrant and processed without parallelization.

See also

TupleLog, TupleLog10

Alternatives

TuplePow

Module

Operators not requiring licensing

---

[out] VARIANT Abs HTupleX.TupleFabs ([in] VARIANT T )
void HOperatorSetX.TupleFabs ([in] VARIANT T, [out] VARIANT Abs )

Compute the absolute value of a tuple (as floating point numbers).

TupleFabs computes the absolute value of the input tuple T. In contrast to TupleAbs, the absolute value is always returned as a floating point number by TupleFabs. The absolute value of a string is not allowed.

Parameter

▶ T (input control) .................. number(-array) ⊲ VARIANT (real, integer) 
  Input tuple.
▶ Abs (output control) ................. number(-array) ⊲ VARIANT (real) 
  Absolute value of the input tuple.

TupleFabs is reentrant and processed without parallelization.

---

TupleFabs is reentrant and processed without parallelization.

---

[out] VARIANT Floor HTupleX.TupleFloor ([in] VARIANT T )
void HOperatorSetX.TupleFloor ([in] VARIANT T, [out] VARIANT Floor )

Compute the floor function of a tuple.

TupleFloor computes the floor function of the input tuple T, i.e., the largest integer less than or equal to T. The floor function is always returned as a floating point number. The floor function of a string is not allowed.

Parameter

▶ T (input control) .................. number(-array) ⊲ VARIANT (real, integer) 
  Input tuple.
▶ Floor (output control) ................. number(-array) ⊲ VARIANT (real) 
  Floor function of the input tuple.

TupleFloor is reentrant and processed without parallelization.

---

TupleFloor is reentrant and processed without parallelization.

---

Operators not requiring licensing
Calculate the remainder of the floating point division of two tuples.

TupleFmod computes the remainder of the floating point division of the input tuples T1/T2. If both tuples have the same length the division is performed for the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the division is performed for each element of the longer tuple with the single element of the other tuple. The result is always a floating point number. The division of strings is not allowed.

Parameter

- T1 (input control) number(-array) VARIANT ( real, integer )
  Input tuple 1.
- T2 (input control) number(-array) VARIANT ( real, integer )
  Input tuple 2.
- Fmod (output control) number(-array) VARIANT ( real )
  Remainder of the division of the input tuples.

Parallelization Information

TupleFmod is reentrant and processed without parallelization.

See also

TupleFloor, TupleCeil

Module

Operators not requiring licensing

Calculate the ldexp function of two tuples.

TupleLdexp computes the ldexp function of the input tuples, i.e., T1*2^T2. If both tuples have the same length the operation is performed for the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The result is always a floating point number. The ldexp function of strings is not allowed.

Parameter

- T1 (input control) number(-array) VARIANT ( real, integer )
  Input tuple 1.
- T2 (input control) number(-array) VARIANT ( real, integer )
  Input tuple 2.
- Ldexp (output control) number(-array) VARIANT ( real )
  Ldexp function of the input tuples.

Parallelization Information

TupleLdexp is reentrant and processed without parallelization.

See also

TupleExp

Module

Operators not requiring licensing
Compute the natural logarithm of a tuple.

TupleLog computes the natural logarithm of the input tuple \(T\). The natural logarithm is always returned as a floating point number. The natural logarithm of a string is not allowed.

Parameter

- \(T\) (input control) ........................................ number(-array) \(\sim\) VARIANT ( real, integer )
  Input tuple.
- \(Log\) (output control) ........................................ number(-array) \(\sim\) VARIANT ( real )
  Natural logarithm of the input tuple.

Parallelization Information

TupleLog is reentrant and processed without parallelization.

See also

TupleExp, TuplePow

Alternatives

TupleLog10

Module

Operators not requiring licensing

Compute the base 10 logarithm of a tuple.

TupleLog10 computes the base 10 logarithm of the input tuple \(T\). The logarithm is always returned as a floating point number. The logarithm of a string is not allowed.

Parameter

- \(T\) (input control) ........................................ number(-array) \(\sim\) VARIANT ( real, integer )
  Input tuple.
- \(Log\) (output control) ........................................ number(-array) \(\sim\) VARIANT ( real )
  Base 10 logarithm of the input tuple.

Parallelization Information

TupleLog10 is reentrant and processed without parallelization.

See also

TupleExp, TuplePow

Alternatives

TupleLog

Module

Operators not requiring licensing

Multiply two tuples.

TupleMult computes the product of the input tuples \(P1\) and \(P2\). If both tuples have the same length the corresponding elements of both tuples are subtracted. Otherwise, either \(P1\) or \(P2\) must have length 1. In this case,
the multiplication is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are multiplied, the result is again an integer number. If one of the operands is a floating point number, the result is a floating point number. The multiplication of strings is not allowed.

**Parameter**

- **P1** (input control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Input tuple 1.
- **P2** (input control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Input tuple 2.
- **Prod** (output control) ........................................... number(-array) ↦ VARIANT ( real, integer )
  Product of the input tuples.

**Parallelization Information**

**TupleMult** is reentrant and processed without parallelization.

**Alternatives**

**TupleDiv**

**Module**

Operators not requiring licensing

```
[out] VARIANT Neg HTupleX.TupleNeg ([in] VARIANT T )

void HOperatorSetX.TupleNeg ([in] VARIANT T, [out] VARIANT Neg )
```

Negate a tuple.

**TupleNeg** computes the negation of the input tuple T, i.e., Neg = −T. The negation of an integer number is again an integer number. The negation of a floating point number is a floating point number. The negation of a string is not allowed. The negation of an empty input tuple results in an empty output tuple.

**Parameter**

- **T** (input control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Input tuple.
- **Neg** (output control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Negation of the input tuple.

**Parallelization Information**

**TupleNeg** is reentrant and processed without parallelization.

**Module**

Operators not requiring licensing

```
[out] VARIANT Pow HTupleX.TuplePow ([in] VARIANT T1, [in] VARIANT T2 )

void HOperatorSetX.TuplePow ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Pow )
```

Calculate the power function two tuples.

**TuplePow** computes the power function of the input tuples T1 T2. If both tuples have the same length the power function is applied to the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the power function is performed for each element of the longer tuple with the single element of the other tuple. The result is always a floating point number. The power function of strings is not allowed.

**Parameter**

- **T1** (input control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Input tuple 1.
- **T2** (input control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Input tuple 2.
- **Pow** (output control) ............................................. number(-array) ↦ VARIANT ( real, integer )
  Power function of the input tuples.
Parallelization Information

TuplePow is reentrant and processed without parallelization.

See also

 TupleLog, TupleLog10

TupleExp

Alternatives

Module

Operators not requiring licensing

[out] VARIANT Rad HTupleX.TupleRad ([in] VARIANT Deg )

void HOperatorSetX.TupleRad ([in] VARIANT Deg, [out] VARIANT Rad )

Convert a tuple from degrees to radians.

TupleRad converts the input tuple Deg from degrees to radians. The result is always returned as a floating point number. The conversion of a string is not allowed.

Parameter

▷ Deg (input control) ............................................. number(-array) ⇐ VARIANT ( real, integer )
  Input tuple.
▷ Rad (output control) ............................................. number(-array) ⇐ VARIANT ( real )
  Input tuple in radians.

Parallelization Information

TupleRad is reentrant and processed without parallelization.

See also

TupleDeg

Module

Operators not requiring licensing

[out] VARIANT Sin HTupleX.TupleSin ([in] VARIANT T )

void HOperatorSetX.TupleSin ([in] VARIANT T, [out] VARIANT Sin )

Compute the sine of a tuple.

TupleSin computes the sine of the input tuple T. The sine is always returned as a floating point number. The sine of a string is not allowed.

Parameter

▷ T (input control) ............................................. number(-array) ⇐ VARIANT ( real, integer )
  Input tuple.
▷ Sin (output control) ............................................. number(-array) ⇐ VARIANT ( real )
  Sine of the input tuple.

Parallelization Information

TupleSin is reentrant and processed without parallelization.

See also

TupleAsin

Alternatives

TupleCos, TupleTan

Module

Operators not requiring licensing
14.1. ARITHMETIC

Compute the hyperbolic sine of a tuple.

TupleSinh computes the hyperbolic sine of the input tuple \( T \). The hyperbolic sine of a string is not allowed.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T ) (input control)</td>
</tr>
<tr>
<td>( \text{Sinh} ) (output control)</td>
</tr>
</tbody>
</table>

TupleSinh is reentrant and processed without parallelization.

Alternatives

TupleCosh, TupleTanh

Module

Operators not requiring licensing

Compute the square root of a tuple.

TupleSqrt computes the square root of the input tuple \( T \). The square root of a string is not allowed.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T ) (input control)</td>
</tr>
<tr>
<td>( \text{Sqrt} ) (output control)</td>
</tr>
</tbody>
</table>

Restriction: \( T \geq 0 \)

TupleSqrt is reentrant and processed without parallelization.

Module

Operators not requiring licensing

Subtract two tuples.

TupleSub computes the difference of the input tuples \( D_1 \) and \( D_2 \). If both tuples have the same length the corresponding elements of both tuples are subtracted. Otherwise, either \( D_1 \) or \( D_2 \) must have length 1. In this case, the subtraction is performed for each element of the longer tuple with the single element of the other tuple. If two integer numbers are subtracted, the result is again an integer number. If one of the operands is a floating point number, the result is a floating point number. The subtraction of strings is not allowed.
CHAPTER 14. TUPLE

Parameter

- $D_1$ (input control) number-array $\Rightarrow$ VARIANT (real, integer)
  Input tuple 1.
- $D_2$ (input control) number-array $\Rightarrow$ VARIANT (real, integer)
  Input tuple 2.
- $\text{Diff}$ (output control) number-array $\Rightarrow$ VARIANT (real, integer)
  Difference of the input tuples.

Parallelization Information

TupleSub is reentrant and processed without parallelization.

Alternatives

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>[out] VARIANT Tan</th>
<th>HTupleX.TupleTan ([in] VARIANT T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.TupleTan ([in] VARIANT T, [out] VARIANT Tan )</td>
<td></td>
</tr>
</tbody>
</table>

Compute the tangent of a tuple.

TupleTan computes the tangent of the input tuple $T$. The tangent is always returned as a floating point number. The tangent of a string is not allowed.

Parameter

- $T$ (input control) number-array $\Rightarrow$ VARIANT (real, integer)
  Input tuple.
- $\text{Tan}$ (output control) number-array $\Rightarrow$ VARIANT (real)
  Tangent of the input tuple.

Parallelization Information

TupleTan is reentrant and processed without parallelization.

See also

TupleAtan, TupleAtan2

TupleSin, TupleCos

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>[out] VARIANT Tanh</th>
<th>HTupleX.TupleTanh ([in] VARIANT T )</th>
</tr>
</thead>
<tbody>
<tr>
<td>void HOperatorSetX.TupleTanh ([in] VARIANT T, [out] VARIANT Tanh )</td>
<td></td>
</tr>
</tbody>
</table>

Compute the hyperbolic tangent of a tuple.

TupleTanh computes the hyperbolic tangent of the input tuple $T$. The hyperbolic tangent is always returned as a floating point number. The hyperbolic tangent of a string is not allowed.

Parameter

- $T$ (input control) number-array $\Rightarrow$ VARIANT (real, integer)
  Input tuple.
- $\text{Tanh}$ (output control) number-array $\Rightarrow$ VARIANT (real)
  Hyperbolic tangent of the input tuple.

Parallelization Information

TupleTanh is reentrant, local, and processed without parallelization.

TupleSinh, TupleCosh

HALCON/COM Reference Manual, 2005-2-1
Operators not requiring licensing

14.2 Bit-Operations

Compute the bitwise and of two tuples.

**TupleBand** computes the bitwise and of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

```haskell
void HOperatorSetX.TupleBand ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Band )
```

**Parameter**

- **T1** (input control) integer(-array) -> VARIANT(integer)
  Input tuple 1.
- **T2** (input control) integer(-array) -> VARIANT(integer)
  Input tuple 2.
- **Band** (output control) integer(-array) -> VARIANT(integer)
  Binary and of the input tuples.

**Parallelization Information**

TupleBand is reentrant and processed without parallelization.

See also

TupleAnd, TupleOr, TupleXor, TupleNot

Alternatives

TupleBor, TupleBxor, TupleBnot

Operators not requiring licensing

Compute the bitwise not of two tuples.

**TupleBnot** computes the bitwise not of the input tuple T. The input tuple must contain only integer numbers.

```haskell
void HOperatorSetX.TupleBnot ([in] VARIANT T, [out] VARIANT BNot )
```

**Parameter**

- **T** (input control) integer(-array) -> VARIANT(integer)
  Input tuple.
- **BNot** (output control) integer(-array) -> VARIANT(integer)
  Binary not of the input tuple.

**Parallelization Information**

TupleBnot is reentrant and processed without parallelization.

See also

TupleAnd, TupleOr, TupleXor, TupleNot

Alternatives

TupleBand, TupleBor, TupleBxor

Operators not requiring licensing
CHAPTER 14. TUPLE

```
[out] VARIANT Bor HTupleX.TupleBor ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleBor ([in] VARIANT T1, [in] VARIANT T2, 
[out] VARIANT Bor )
```

Compute the bitwise or of two tuples.

**TupleBor** computes the bitwise or of the input tuples **T1** and **T2**. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either **T1** or **T2** must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

```
Parameter

▷ **T1** (input control) ............................................ integer(-array) ↦ VARIANT ( integer )
  Input tuple 1.
▷ **T2** (input control) ............................................ integer(-array) ↦ VARIANT ( integer )
  Input tuple 2.
▷ **Bor** (output control) .......................................... integer(-array) ↦ VARIANT ( integer )
  Binary or of the input tuples.
```

**Parallelization Information**

**TupleBor** is reentrant and processed without parallelization.

See also

**TupleAnd, TupleOr, TupleXor, TupleNot**

Alternatives

**TupleBand, TupleBxor, TupleBnot**

Module

Operators not requiring licensing

```
[out] VARIANT Bxor HTupleX.TupleBxor ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleBxor ([in] VARIANT T1, [in] VARIANT T2, 
[out] VARIANT Bxor )
```

Compute the bitwise exclusive or of two tuples.

**TupleBxor** computes the bitwise exclusive or of the input tuples **T1** and **T2**. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either **T1** or **T2** must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

```
Parameter

▷ **T1** (input control) ............................................ integer(-array) ↦ VARIANT ( integer )
  Input tuple 1.
▷ **T2** (input control) ............................................ integer(-array) ↦ VARIANT ( integer )
  Input tuple 2.
▷ **Bxor** (output control) ........................................ integer(-array) ↦ VARIANT ( integer )
  Binary exclusive or of the input tuples.
```

**Parallelization Information**

**TupleBxor** is reentrant and processed without parallelization.

See also

**TupleAnd, TupleOr, TupleXor, TupleNot**

Alternatives

**TupleBand, TupleBor, TupleBnot**

Module

Operators not requiring licensing
14.2. BIT-OPERATIONS

---


Shift a tuple bitwise to the left.

TupleLsh shifts the tuple T bitwise to the left by Shift places. If no overflow occurs, this operation is equivalent to a multiplication by $2^{\text{Shift}}$. If T is negative, the result depends on the hardware. If both tuples have the same length the corresponding elements of both tuples are shifted. Otherwise, either T or Shift must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- T (input control) ..........integer(-array) $\Rightarrow$ VARIANT (integer)  
  Input tuple.
- Shift (input control) ..............integer(-array) $\Rightarrow$ VARIANT (integer)  
  Number of places to shift the input tuple.
- Lsh (output control) ........integer(-array) $\Rightarrow$ VARIANT (integer)  
  Shifted input tuple.

Parallelization Information

TupleLsh is reentrant and processed without parallelization.

See also

TupleRsh

Alternatives

TupleMult

Module

Operators not requiring licensing

---


Shift a tuple bitwise to the right.

TupleRsh shifts the tuple T bitwise to the right by Shift places. This operation is equivalent to a division by $2^{\text{Shift}}$. If T is negative, the result depends on the hardware. If both tuples have the same length the corresponding elements of both tuples are shifted. Otherwise, either T or Shift must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- T (input control) ..........integer(-array) $\Rightarrow$ VARIANT (integer)  
  Input tuple.
- Shift (input control) ..............integer(-array) $\Rightarrow$ VARIANT (integer)  
  Number of places to shift the input tuple.
- Rsh (output control) ........integer(-array) $\Rightarrow$ VARIANT (integer)  
  Shifted input tuple.

Parallelization Information

TupleRsh is reentrant and processed without parallelization.

See also

TupleLsh

Alternatives

TupleDiv

Module

Operators not requiring licensing
CHAPTER 14. TUPLE

14.3 Comparison

```c
[out] long Equal HTupleX.TupleEqual ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleEqual ([in] VARIANT T1, [in] VARIANT T2,
[out] VARIANT Equal )
```

Test whether two tuples are equal.

TupleEqual tests whether the two input tuples $T_1$ and $T_2$ are equal by comparing the tuples elementwise. Two tuples are equal, if they have got the same number of elements and if their elements are equal. Two tuple elements are equal, if they are both (integer or floating point) numbers or both are strings and contain the same value.

Parameter

- $T_1$ (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) $\rightsquigarrow$ VARIANT ( integer, real, string )
  Input tuple 1.
- $T_2$ (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) $\rightsquigarrow$ VARIANT ( integer, real, string )
  Input tuple 2.
- $Equal$ (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer $\rightsquigarrow$ long / VARIANT
  Result of the comparison of the input tuples.

Parallelization Information

TupleEqual is reentrant and processed without parallelization.

Alternatives

TupleNotEqual, TupleLess, TupleGreater, TupleLessEqual, TupleGreaterEqual

Module

Operators not requiring licensing

```c
[out] long Greater HTupleX.TupleGreater ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleGreater ([in] VARIANT T1, [in] VARIANT T2,
[out] VARIANT Greater )
```

Test whether a tuple is greater than another tuple.

TupleGreater tests whether the input tuple $T_1$ is greater than $T_2$. A tuple $T_1$ is said to be greater than a tuple $T_2$, if $T_1$ has been found to be greater when comparing it elementwise to $T_2$ or (for the case that the elementwise comparison did not show that $T_1$ is greater than $T_2$) if $T_1$ has got more elements than $T_2$. With the elementwise comparison the single elements of $T_1$ and $T_2$ are compared with each other one after another (i.e., the first element of $T_1$ is compared to the first element of $T_2$ and the second element of $T_1$ is compared to the second element of $T_2$ etc.). If an element of $T_1$ is greater than its counterpart of $T_2$, $T_1$ is said to be greater than $T_2$. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleGreater returns an error.

Parameter

- $T_1$ (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) $\rightsquigarrow$ VARIANT ( integer, real, string )
  Input tuple 1.
- $T_2$ (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) $\rightsquigarrow$ VARIANT ( integer, real, string )
  Input tuple 2.
- $Greater$ (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer $\rightsquigarrow$ long / VARIANT
  Result of the comparison of the input tuples.

Parallelization Information

TupleGreater is reentrant and processed without parallelization.

Alternatives

TupleGreaterEqual, TupleLess, TupleLessEqual, TupleEqual, TupleNotEqual

Module

Operators not requiring licensing
Test, whether a tuple is greater or equal to another tuple.

TupleGreaterEqual tests whether the input tuple T1 is greater or equal to T2. A tuple T1 is said to be greater or equal to a tuple T2, if T1 is not less than T2. A tuple T1 is said to be less than a tuple T2, if T1 has been found to be less when comparing it elementwise to T2 or (for the case that the elementwise comparison did not show that T1 is less than T2) if T1 has got less elements than T2. With the elementwise comparison the single elements of T1 and T2 are compared with each other one after another (i.e., the first element of T1 is compared to the first element of T2 and the second element of T1 is compared to the second element of T2 etc.). If an element of T1 is less than its counterpart of T2, T1 is said to be less than T2. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleGreaterEqual returns an error.

```
[out] long Greatereq HTupleX.TupleGreaterEqual ([in] VARIANT T1, [in] VARIANT T2 )
```

```
void HOperatorSetX.TupleGreaterEqual ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Greatereq )
```

Test, whether a tuple is less than another tuple.

TupleLess tests whether the input tuple T1 is less than T2. A tuple T1 is said to be less than a tuple T2, if T1 has been found to be less when comparing it elementwise to T2 or (for the case that the elementwise comparison did not show that T1 is less than T2) if T1 has got less elements than T2. With the elementwise comparison the single elements of T1 and T2 are compared with each other one after another (i.e., the first element of T1 is compared to the first element of T2 and the second element of T1 is compared to the second element of T2 etc.). If an element of T1 is less than its counterpart of T2, T1 is said to be less than T2. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleLess returns an error.

```
[out] long Less HTupleX.TupleLess ([in] VARIANT T1, [in] VARIANT T2 )
```

```
void HOperatorSetX.TupleLess ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Less )
```
CHAPTER 14. TUPLE

Alternatives

TupleLessEqual, TupleGreater, TupleGreaterEqual, TupleEqual, TupleNotEqual

Operators not requiring licensing

[out] long Lesseq HTupleX.TupleLessEqual ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleLessEqual ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Lesseq )

Test, whether a tuple is less or equal to another tuple.
TupleLessEqual tests whether the input tuple T1 is less or equal to T2. A tuple T1 is said to be less or equal to a tuple T2, if T1 is not greater than T2. A tuple T1 is said to be greater than a tuple T2, if T1 has been found to be greater when comparing it elementwise to T2 or (for the case that the elementwise comparison did not show that T1 is greater than T2) if T1 has got more elements than T2. With the elementwise comparison the single elements of T1 and T2 are compared with each other one after another (i.e., the first element of T1 is compared to the first element of T2 and the second element of T1 is compared to the second element of T2 etc.). If an element of T1 is greater than its counterpart of T2, T1 is said to be greater than T2. As a precondition for comparing the tuples elementwise two corresponding elements must either both be (integer or floating point) numbers or both be strings. Otherwise TupleLessEqual returns an error.

Parameter

▷ T1 (input control) .......................... number(-array) ~ VARIANT ( integer, real, string )
Input tuple 1.
▷ T2 (input control) .......................... number(-array) ~ VARIANT ( integer, real, string )
Input tuple 2.
▷ Lesseq (output control) ......................... integer ~ long / VARIANT
Result of the comparison of the input tuples.

Parallelization Information

TupleLessEqual is reentrant and processed without parallelization.

Alternatives

Operators not requiring licensing

[out] long Nequal HTupleX.TupleNotEqual ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleNotEqual ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Nequal )

Test, whether two tuples are not equal.
TupleNotEqual tests whether the two input tuples T1 and T2 are not equal. Two tuples are not equal, if they consist of a different number of elements or if they differ when compared elementwise. Two tuple elements differ, if they are of types that may not be compared (e.g., one string and one integer) or if they differ in their values.

Parameter

▷ T1 (input control) .......................... number(-array) ~ VARIANT ( integer, real, string )
Input tuple 1.
▷ T2 (input control) .......................... number(-array) ~ VARIANT ( integer, real, string )
Input tuple 2.
▷ Nequal (output control) ......................... integer ~ long / VARIANT
Result of the comparison of the input tuples.
14.4 Conversion

**Parallelization Information**

TupleNotEqual is reentrant and processed without parallelization.

**Alternatives**

TupleEqual, TupleLess, TupleGreater, TupleLessEqual, TupleGreaterEqual

Operators not requiring licensing

### 14.4 Conversion

**HOperatorSetX.TupleChr**

```c
[out] VARIANT Chr HTupleX.TupleChr ([in] VARIANT T )
void HOperatorSetX.TupleChr ([in] VARIANT T, [out] VARIANT Chr )
```

Convert a tuple of integers into strings with the corresponding ASCII codes.

**TupleChr** converts the input tuple \( T \), consisting of integer numbers, into a tuple of strings of length 1, the characters of which have the ASCII code of the corresponding input number.

**Parameter**

- \( T \) (input control) ........................................... integer(-array) \( \sim \) VARIANT ( integer )
  
  Input tuple.

- \( Chr \) (output control) ........................................ string(-array) \( \sim \) VARIANT ( string )
  
  Strings corresponding to the ASCII code of the input tuple.

**Parallelization Information**

TupleChr is reentrant and processed without parallelization.

**See also**

TupleOrd, TupleOrds

**Alternatives**

TupleChrt

**Module**

Operators not requiring licensing

**HOperatorSetX.TupleChrt**

```c
[out] VARIANT Chr HTupleX.TupleChrt ([in] VARIANT T )
void HOperatorSetX.TupleChrt ([in] VARIANT T, [out] VARIANT Chr )
```

Convert a tuple of integers into strings with the corresponding ASCII codes.

**TupleChrt** converts the input tuple \( T \), consisting of integer numbers, into a tuple of strings and integer numbers (where only the number 0 can occur in the output), the characters of which have the ASCII code of the corresponding input number. The operator tries to pack as many of the input numbers into one string as possible. Only if the value 0 occurs in \( T \) the current string is terminated at this point, the integer number 0 is inserted into the output, and a new string with the remaining input values is started. This operator can be used to convert data read with ReadSerial into strings. With this mechanism it is possible to read bytes with the value 0.

**Parameter**

- \( T \) (input control) ........................................... integer(-array) \( \sim \) VARIANT ( integer )
  
  Input tuple.

- \( Chrt \) (output control) ........................................ string(-array) \( \sim \) VARIANT ( string, integer )
  
  Strings corresponding to the ASCII code of the input tuple.

**Example**

read_serial (SerialHandle, 100, Data)
tuple_chrt (Data, Strings)
### TupleChrt

TupleChrt is reentrant and processed without parallelization.

### TupleOrd, TupleOrds, ReadSerial

See also Alternatives.

### TupleChr

Operators not requiring licensing

---

#### Parallelization Information

**TupleChrt** is reentrant and processed without parallelization.

**See also** TupleOrd, TupleOrds, ReadSerial

**Alternatives**

**TupleChr**

Operators not requiring licensing

---

#### TupleDeg

**Convert a tuple from radians to degrees.**

**TupleDeg** converts the input tuple **Rad** from radians to degrees. The result is always returned as a floating point number. The conversion of a string is not allowed.

**Parameter**

- **Rad** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \(\leadsto\) VARIANT (real, integer) .
  
  Input tuple.

- **Deg** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \(\leadsto\) VARIANT (real) .
  
  Input tuple in degrees.

---

#### Parallelization Information

**TupleDeg** is reentrant and processed without parallelization.

**See also** TupleRad

---

#### Module

Operators not requiring licensing

---

#### TupleIsNumber

**Check a tuple (of strings) whether it represents numbers.**

**TupleIsNumber** checks each element of the input tuple **T** whether it represents a number. If the element is a number (real or integer), 1 is returned for that element. If the element is a string, it is checked whether the string represents a number. If so, 1 is returned, otherwise 0.

**Parameter**

- **T** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . number(-array) \(\leadsto\) VARIANT (real, integer, string) .
  
  Input tuple.

- **IsNumber** (output control) . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \(\leadsto\) VARIANT (integer) .
  
  Tuple with boolean numbers.

---

#### Parallelization Information

**TupleIsNumber** is reentrant and processed without parallelization.

**See also** TupleNumber

---

#### Module

Operators not requiring licensing
14.4. CONVERSION

```plaintext
[out] VARIANT Number HTupleX.TupleNumber ([in] VARIANT T )
void HOperatorSetX.TupleNumber ([in] VARIANT T, [out] VARIANT Number )
```

Convert a tuple (of strings) into a tuple of numbers.

**TupleNumber** converts the input tuple `T` into a tuple of numbers. If the input tuple contains numbers, they are simply copied into the output tuple. Strings are converted into the appropriate type of number (integer or floating point numbers) or are copied as strings if they do not represent a number. Note that strings starting with `0x` are interpreted as hexadecimal numbers, and strings starting with `0` as octal numbers. For example, the string `"20"` is converted to the integer 20, `"020"` to 16, and `"0x20"` to 32.

- **Parameter**
  - `T` (input control) ............... number(-array) ⇐ VARIANT ( real, integer, string )
    Input tuple.
  - `Number` (output control) ............ number(-array) ⇐ VARIANT ( real, integer, string )
    Input tuple as numbers.

**Parallelization Information**

**TupleNumber** is reentrant and processed without parallelization.

---

```plaintext
[out] VARIANT Ord HTupleX.TupleOrd ([in] VARIANT T )
void HOperatorSetX.TupleOrd ([in] VARIANT T, [out] VARIANT Ord )
```

Convert a tuple of strings of length 1 into a tuple of their ASCII codes.

**TupleOrd** converts the input tuple `T`, which may only contain strings of length 1, into a tuple of integer numbers that represent the ASCII code of the characters of the strings.

- **Parameter**
  - `T` (input control) ...................... string(-array) ⇐ VARIANT ( string )
    Input tuple.
  - `Ord` (output control) .................. integer(-array) ⇐ VARIANT ( integer )
    ASCII code of the input tuple.

**Parallelization Information**

**TupleOrd** is reentrant and processed without parallelization.

---

```plaintext
[out] VARIANT Ords HTupleX.TupleOrds ([in] VARIANT T )
void HOperatorSetX.TupleOrds ([in] VARIANT T, [out] VARIANT Ords )
```

Convert a tuple of strings into a tuple of their ASCII codes.

**TupleOrds** converts the input tuple `T`, which may only contain strings and integer numbers, into a tuple of integer numbers that represent the ASCII code of the characters of the strings. The characters of the individual strings are output according to their order within the string and within the tuple. Integer numbers are simply copied.
to an appropriate position in the output string. This operator can be used to prepare outputs with \texttt{WriteSerial}. In particular, a byte with value 0 can be written by inserting the integer number 0 into \texttt{T}.

\begin{itemize}
  \item \texttt{T} (input control) \hspace{0.5cm} \texttt{string(-array)} \rightarrow \texttt{VARIANT} (\texttt{string, integer})
  \text{Input tuple.}
  \item \texttt{Ords} (output control) \hspace{0.5cm} \texttt{integer(-array)} \rightarrow \texttt{VARIANT} (\texttt{integer})
  \text{ASCII code of the input tuple.}
\end{itemize}

\begin{verbatim}
tuple_ords (["String 1", 0, "String 2", 0], Data)
write_serial (SerialHandle, Data)
\end{verbatim}

\textbf{Parallelization Information}
\texttt{TupleOrds} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{See also} \texttt{TupleChr, TupleChrt, WriteSerial}

\textbf{Alternatives}
\texttt{TupleOrd}

\textbf{Module}
Operators not requiring licensing

\begin{verbatim}
[out] VARIANT Real HTupleX.TupleReal ([in] VARIANT T )
void HOperatorSetX.TupleReal ([in] VARIANT T, [out] VARIANT Real )
\end{verbatim}

\textit{Convert a tuple into a tuple of floating point numbers.} \texttt{TupleReal} converts the input tuple \texttt{T} into a tuple of floating point numbers. The conversion of a string is not allowed.

\begin{itemize}
  \item \texttt{T} (input control) \hspace{0.5cm} \texttt{number(-array)} \rightarrow \texttt{VARIANT} (\texttt{real, integer})
  \text{Input tuple.}
  \item \texttt{Real} (output control) \hspace{0.5cm} \texttt{number(-array)} \rightarrow \texttt{VARIANT} (\texttt{real})
  \text{Input tuple as floating point numbers.}
\end{itemize}

\textbf{Parallelization Information}
\texttt{TupleReal} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Module}
Operators not requiring licensing

\begin{verbatim}
[out] VARIANT Round HTupleX.TupleRound ([in] VARIANT T )
void HOperatorSetX.TupleRound ([in] VARIANT T, [out] VARIANT Round )
\end{verbatim}

\textit{Convert a tuple into a tuple of integer numbers.} \texttt{TupleRound} converts the input tuple \texttt{T} into a tuple of integer numbers by rounding \texttt{T} to the nearest integer. The conversion of a string is not allowed.

\begin{itemize}
  \item \texttt{T} (input control) \hspace{0.5cm} \texttt{number(-array)} \rightarrow \texttt{VARIANT} (\texttt{real, integer})
  \text{Input tuple.}
  \item \texttt{Round} (output control) \hspace{0.5cm} \texttt{number(-array)} \rightarrow \texttt{VARIANT} (\texttt{integer})
  \text{Input tuple as integer numbers.}
\end{itemize}
**14.4. CONVERSION**

**Parallelization Information**

*TupleRound* is reentrant and processed without parallelization.

**Module**

Operators not requiring licensing

```c
[out] VARIANT String HTupleX.TupleString ( [in] VARIANT T,
[in] String Format )

void HOperatorSetX.TupleString ( [in] VARIANT T, [in] VARIANT Format,
[out] VARIANT String )
```

Convert a tuple into a tuple of strings.

*TupleString* converts numbers to strings or modifies strings. The operator has two parameters: *T* is the number or string that has to be converted. *Format* specifies the conversion. This format string consists of the following four parts

```markdown
<flags><field width><precision><conversion characters>
```

So a conversion might look like

```markdown
tuple\_string(1332.4554, \'.6e\', String)
```

**flags** Zero or more flags, in any order, which modify the meaning of the conversion specification. Flags may consist of the following characters:
- The result of the conversion is left justified within the field.
- The result of a signed conversion always begins with a sign, + or -.
- If the first character of a signed conversion is not a sign, a space character is prefixed to the result. This means that if the space flag and + flag both appear, the space flag is ignored.
- The value is to be converted to an “alternate form”. For *d* and *s* conversions, this flag has no effect. For *o* conversion (see below), it increases the precision to force the first digit of the result to be a zero. For *x* or *X* conversion (see below), a non-zero result has 0x or 0X prefixed to it. For *e*, *E*, *f*, *g*, and *G* conversions, the result always contains a radix character, even if no digits follow the radix character. For *g* and *G* conversions, trailing zeros are not removed from the result, contrary to usual behavior.

**field width** An optional string of decimal digits to specify a minimum field width. For an output field, if the converted value has fewer characters than the field width, it is padded on the left (or right, if the left-adjustment flag, - has been given) to the field width.

**precision** The precision specifies the minimum number of digits to appear for the *d*, *o*, *x*, or *X* conversions (the field is padded with leading zeros), the number of digits to appear after the radix character for the *e* and *f* conversions, the maximum number of significant digits for the *g* conversion, or the maximum number of characters to be printed from a string in *s* conversion. The precision takes the form of a period . followed by a decimal digit string. A null digit string is treated as a zero.

**conversion characters** A conversion character indicates the type of conversion to be applied:
- *d*, *o*, *x*, *X* The integer argument is printed in signed decimal (*d*), unsigned octal (*o*), or unsigned hexadecimal notation (*x* and *X*). The *x* conversion uses the numbers and letters 0123456789abcdef, and the *X* conversion uses the numbers and letters 0123456789ABCDEF. The precision component of the argument specifies the minimum number of digits to appear. If the value being converted can be represented in fewer digits than the specified minimum, it is expanded with leading zeroes. The default precision is 1. The result of converting a zero value with a precision of 0 is no characters.
- *f* The floating-point number argument is printed in decimal notation in the style \([-\)] dddrddd, where the number of digits after the radix character, r, is equal to the precision specification. If the precision is omitted from the argument, six digits are output; if the precision is explicitly 0, no radix appears.
- *e*, *E* The floating-point-number argument is printed in the style \([-\)] drdddde±dd, where there is one digit before the radix character, and the number of digits after it is equal to the precision. When the precision is missing, six digits are produced; if the precision is 0, no radix character appears. The *E* conversion character produces a number with *E* introducing the exponent instead of *e*. The exponent always contains at least two digits. However, if the value to be printed requires an exponent greater than two digits, additional exponent digits are printed as necessary.

HALCON 6.1.4
The floating-point-number argument is printed in style \( f \) or \( e \) (or in style \( E \) in the case of a \( G \) conversion character), with the precision specifying the number of significant digits. The style used depends on the value converted; style \( e \) is used only if the exponent resulting from the conversion is less than -4 or greater than or equal to the precision. Trailing zeros are removed from the result. A radix character appears only if it is followed by a digit.

The argument is taken to be a string, and characters from the string are printed until the end of the string or the number of characters indicated by the precision specification of the argument is reached. If the precision is omitted from the argument, it is interpreted as infinite and all characters up to the end of the string are printed.

Similar to the \( s \) conversion specifier, except that the string can contain backslash-escape sequences which are then converted to the characters they represent.

In no case does a nonexistent or insufficient field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result.

---

### Parameter

- **T** (input control) 
 .number(-array) \( \rightarrow \) VARIANT (real, integer, string)
  Input tuple.

- **Format** (input control) 
 .string \( \rightarrow \) String / VARIANT
  Format string.

- **String** (output control) 
 .string(-array) \( \rightarrow \) VARIANT (string)
  Input tuple converted to strings.

---

**Parallelization Information**

TupleString is reentrant and processed without parallelization.

---

**Alternatives**

---

**Module**

---

### Operators not requiring licensing

---

### 14.5 Creation

 TupleConcat concatenates the input tuples \( T1 \) and \( T2 \) to a new tuple \( Concat \). The first elements of \( Concat \) conform to the elements of \( T1 \) and the remaining elements of \( Concat \) conform to those of \( T2 \).

---

### Parameter

- **T1** (input control) 
  .number(-array) \( \rightarrow \) VARIANT (real, integer, string)
  Input tuple 1.

- **T2** (input control) 
  .number(-array) \( \rightarrow \) VARIANT (real, integer, string)
  Input tuple 2.

- **Concat** (output control) 
  .number(-array) \( \rightarrow \) VARIANT (real, integer, string)
  Concatenation of input tuples.

---

**Parallelization Information**

TupleConcat is reentrant and processed without parallelization.

---

**Alternatives**

---

**Module**

---

Operators not requiring licensing
14.6. ELEMENT-ORDER

Generate a tuple of a specific length and initialize its elements. 

TupleGenConst generates a new tuple in Newtuple. The input parameter Length determines the number of elements for the new tuple. Thus Length may only consist of a single number. If Length contains a floating point number, this may only represent an integer value (without fraction). The data type and value of each element of the new generated tuple is determined by the input parameter Const that may only consist of a single element. All elements in Newtuple have got the same data type and value as the single element in Const.

Parameter

> Length (input control) ......................... number ~ VARIANT (real, integer) 
  Length of tuple to generate.

> Const (input control) ......................... number ~ VARIANT (real, integer, string) 
  Constant for initializing the tuple elements.

> Newtuple (output control) ................. number(-array) ~ VARIANT (real, integer, string) 
  New Tuple.

Parallelization Information

TupleGenConst is reentrant and processed without parallelization.

Alternatives

TupleStrBitSelect, TupleSelect, TupleStrFirstN, TupleStrLastN, TupleConcat

Module

Operators not requiring licensing

14.6 Element-Order

Invert a tuple.

TupleInverse inverts the input tuple Tuple. Thus Inverted contains the same elements as Tuple but with the reverse order.

Parameter

> Tuple (input control) ....................... number(-array) ~ VARIANT (real, integer, string) 
  Input tuple.

> Inverted (output control) ................. number(-array) ~ VARIANT (real, integer, string) 
  Inverted input tuple.

Parallelization Information

TupleInverse is reentrant and processed without parallelization.

Alternatives

TupleSort, TupleSortIndex

Module

Operators not requiring licensing

Sorts the elements of a tuple in ascending order.

HALCON 6.1.4
TupleSort sorts all elements of Tuple in ascending order and returns the result with Sorted. As a precondition the single elements of Tuple must be comparable. Thus Tuple must either exclusively consist of strings or it must only contain (integer or floating point) numbers. In the latter case integers and floating point numbers may be mixed.

Parameter

- **Tuple** (input control) : number(-array) \(\sim\) VARIANT (real, integer, string)
  
  Input tuple.

- **Sorted** (output control) : number(-array) \(\sim\) VARIANT (real, integer, string)
  
  Sorted tuple.

Parallelization Information

TupleSort is reentrant and processed without parallelization.

Alternatives

TupleSortIndex, TupleInverse

Module

Operators not requiring licensing

```
[out] VARIANT Indices HTupleX.TupleSortIndex ([in] VARIANT Tuple )

void HOperatorSetX.TupleSortIndex ([in] VARIANT Tuple, [out] VARIANT Indices )
```

Sorts the elements of a tuple and returns the indices of the sorted tuple.

TupleSortIndex sorts all elements of Tuple in ascending order and returns the indices of the elements of the sorted tuple (in relation to the input tuple Tuple) with Indices. As a precondition the single elements of Tuple must be comparable. Thus Tuple must either exclusively consist of strings or it must only contain (integer or floating point) numbers. In the latter case integers and floating point numbers may be mixed.

Parameter

- **Tuple** (input control) : number(-array) \(\sim\) VARIANT (real, integer, string)
  
  Input tuple.

- **Indices** (output control) : number(-array) \(\sim\) VARIANT (integer)
  
  Sorted tuple.

Parallelization Information

TupleSortIndex is reentrant and processed without parallelization.

Alternatives

TupleSort, TupleInverse

Module

Operators not requiring licensing

14.7 Features

```
[out] VARIANT Deviation HTupleX.TupleDeviation ([in] VARIANT Tuple )

void HOperatorSetX.TupleDeviation ([in] VARIANT Tuple, [out] VARIANT Deviation )
```

Return the standard deviation of the elements of a tuple.

TupleDeviation calculates the standard derivation of all elements of the input tuple Tuple. It returns the derivation as a floating point number in the output parameter Deviation. The input tuple may only consist of numbers (integer or floating point numbers).
14.7. FEATURES

Parameter

- **Tuple** (input control) .................. number(-array) ~ VARIANT (real, integer, string)
  Input tuple.

- **Deviation** (output control) .................. number(-array) ~ VARIANT (real)
  Standard deviation of tuple elements.

Parallelization Information

**TupleDeviation** is reentrant and processed without parallelization.

Alternatives

**TupleMean, TupleSum, TupleMin, TupleMax, TupleLength**

Module

Operators not requiring licensing

---

```c
[out] VARIANT Length HTupleX.TupleLength ([in] VARIANT Tuple )
void HOperatorSetX.TupleLength ([in] VARIANT Tuple, [out] VARIANT Length )
```

Returns the number of elements of a tuple.

**TupleLength** returns the number of elements of the input tuple **Tuple**.

Parameter

- **Tuple** (input control) .................. number(-array) ~ VARIANT (real, integer, string)
  Input tuple.

- **Length** (output control) .................. number(-array) ~ VARIANT (real)
  Number of elements of input tuple.

Parallelization Information

**TupleLength** is reentrant and processed without parallelization.

Alternatives

**TupleMin, TupleMax, TupleMean, TupleDeviation, TupleSum**

Module

Operators not requiring licensing

---

```c
[out] VARIANT Max HTupleX.TupleMax ([in] VARIANT Tuple )
void HOperatorSetX.TupleMax ([in] VARIANT Tuple, [out] VARIANT Max )
```

Returns the maximal element of a tuple.

**TupleMax** returns the maximal element of all elements of the input tuple **Tuple**. All elements of **Tuple** either have to be strings or numbers (integer or floating point numbers). It is not allowed to mix strings with numerical values. The result parameter **Max** will contain a floating point number, if at least one element of **Tuple** is a floating point number. If all elements of **Tuple** are integer numbers the resulting sum will also be an integer number.

Parameter

- **Tuple** (input control) .................. number(-array) ~ VARIANT (real, integer, string)
  Input tuple.

- **Max** (output control) .................. number(-array) ~ VARIANT (real)
  Maximal element of the input tuple elements.

Parallelization Information

**TupleMax** is reentrant and processed without parallelization.

Alternatives

**TupleMin, TupleMean, TupleDeviation, TupleSum, TupleLength**

Module

Operators not requiring licensing

---

HALCON 6.1.4
Return the mean value of a tuple of numbers.

TupleMean returns the mean value of all elements of the input tuple Tuple as a floating point number in the output parameter Mean. The input tuple may only consist of numbers (integer or floating point numbers).

Parameter

- Tuple (input control) number(-array) ~ VARIANT (real, integer, string) Input tuple.
- Mean (output control) number(-array) ~ VARIANT (real) Mean value of tuple elements.

Parallelization Information

TupleMean is reentrant and processed without parallelization.

Alternatives

TupleDeviation, TupleSum, TupleMin, TupleMax, TupleLength

Module

Operators not requiring licensing

Returns the minimal element of a tuple.

TupleMin returns the minimal element of all elements of the input tuple Tuple. All elements of Tuple either have to be strings or numbers (integer or floating point numbers). It is not allowed to mix strings with numerical values. The result parameter Min will contain a floating point number, if at least one element of Tuple is a floating point number. If all elements of Tuple are integer numbers the resulting sum will also be an integer number.

Parameter

- Tuple (input control) number(-array) ~ VARIANT (real, integer, string) Input tuple.
- Min (output control) number(-array) ~ VARIANT (real) Minimal element of the input tuple elements.

Parallelization Information

TupleMin is reentrant and processed without parallelization.

Alternatives

TupleMax, TupleMean, TupleDeviation, TupleSum, TupleLength

Module

Operators not requiring licensing

Return the sum of all elements of a tuple.

TupleSum returns the sum of all elements of the input tuple Tuple. All elements of Tuple either have to be strings or numbers (integer or floating point numbers). It is not allowed to mix strings with numerical values. The result parameter Sum will contain a floating point number, if at least one element of Tuple is a floating point number. If all elements of Tuple are integer numbers the resulting sum will also be an integer number. If Tuple contains strings, the concatenation will be used for building the sum.
14.8 Logical-Operations

Parameter

- **Tuple** (input control) .............................. number(-array) \(\sim\) VARIANT (real, integer, string)
  Input tuple.
- **Sum** (output control) .............................. number(-array) \(\sim\) VARIANT (real)
  Sum of tuple elements.

Parallelization Information

TupleSum is reentrant and processed without parallelization.

Alternatives

TupleMean, TupleDeviation, TupleMin, TupleMax, TupleLength

Module

Operators not requiring licensing

14.8 Logical-Operations

```c
[out] VARIANT And HTupleX.TupleAnd ([in] VARIANT T1, [in] VARIANT T2 )
void HOperatorSetX.TupleAnd ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT And )
```

Compute the logical and of two tuples.

**TupleAnd** computes the logical and of the input tuples \(T_1\) and \(T_2\). If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either \(T_1\) or \(T_2\) must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- **T1** (input control) .............................. integer(-array) \(\sim\) VARIANT (integer)
  Input tuple 1.
- **T2** (input control) .............................. integer(-array) \(\sim\) VARIANT (integer)
  Input tuple 2.
- **And** (output control) .............................. integer(-array) \(\sim\) VARIANT (integer)
  Logical and of the input tuples.

Parallelization Information

TupleAnd is reentrant and processed without parallelization.

See also

TupleBand, TupleBor, TupleBxor, TupleBnot

TupleOr, TupleXor, TupleNot

Module

Operators not requiring licensing

```c
[out] VARIANT Not HTupleX.TupleNot ([in] VARIANT T )
void HOperatorSetX.TupleNot ([in] VARIANT T, [out] VARIANT Not )
```

Compute the logical not of two tuples.

**TupleNot** computes the logical not of the input tuple \(T\). The input tuple must contain only integer numbers.

Parameter

- **T** (input control) .............................. integer(-array) \(\sim\) VARIANT (integer)
  Input tuple.
- **Not** (output control) .............................. integer(-array) \(\sim\) VARIANT (integer)
  Binary not of the input tuple.
**Parallelization Information**

TupleNot is reentrant, local, and processed without parallelization.

See also TupleBand, TupleBor, TupleBxor, TupleBnot

Alternatives TupleAnd, TupleOr, TupleXor

Operators not requiring licensing

```c
// Function declaration

HOperatorSetX.TupleOr ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Or );
```

Compute the logical or of two tuples. TupleOr computes the logical or of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- **T1** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \sim VARIANT ( integer )
- **T2** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \sim VARIANT ( integer )
- **Or** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \sim VARIANT ( integer )

**Parallelization Information**

TupleOr is reentrant and processed without parallelization.

See also TupleBand, TupleBor, TupleBxor, TupleBnot

Alternatives TupleAnd, TupleXor, TupleNot

Module

Operators not requiring licensing

```c
// Function declaration

HOperatorSetX.TupleXor ([in] VARIANT T1, [in] VARIANT T2, [out] VARIANT Xor );
```

Compute the logical exclusive or of two tuples. TupleXor computes the logical exclusive or of the input tuples T1 and T2. If both tuples have the same length the operation is performed on the corresponding elements of both tuples. Otherwise, either T1 or T2 must have length 1. In this case, the operation is performed for each element of the longer tuple with the single element of the other tuple. The input tuples must contain only integer numbers.

Parameter

- **T1** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \sim VARIANT ( integer )
- **T2** (input control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \sim VARIANT ( integer )
- **Xor** (output control) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . integer(-array) \sim VARIANT ( integer )

HALCON/COM Reference Manual, 2005-2-1
14.9 Selection

 TupleFirstN selects the first elements of a tuple. Thus Selected contains all elements of Tuple from the first element up to the “n-th” element of Tuple (including the “n-th” element). The index “n” is determined by the input parameter Index. Thus Index must contain a single integer value (if Index consists of a floating point number, this must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0.

 Operators not requiring licensing

 TupleLastN is reentrant and processed without parallelization.

 Alternatives

 TupleConcat

 Operators not requiring licensing

 Select all elements from index “n” to the end of a tuple.

 Starting with the “n-th” element of the tuple Tuple, TupleLastN selects every element of Tuple and returns it with Selected. Thus Selected contains all elements of Tuple from index “n” up to the last element of Tuple (including the element at position “n”). The index “n” is determined by the input parameter Index. Thus Index must contain a single integer value (if Index consists of a floating point number, this must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0.
Parameter

- **Tuple** (input control) .......................... number(-array)  \( \sim \) VARIANT (real, integer, string)  
  Input tuple.
- **Index** (input control) .......................... number  \( \sim \) VARIANT (real, integer)  
  Index of the first element to select.
- **Selected** (output control) ......................... number(-array)  \( \sim \) VARIANT (real, integer, string)  
  Selected tuple elements.

Parallelization Information

**TupleLastN** is reentrant and processed without parallelization.

Alternatives

- **TupleFirstN**, **TupleSelect**, **TupleStrBitSelect**, **TupleConcat**

Module

Select single elements of a tuple.

**TupleSelect** selects one or more single elements of the tuple **Tuple** and returns them with **Selected**. At this, **Index** determines the indices of the elements to select. Thus **Index** may only contain integer values (any floating point number within **Index** must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0.

Parameter

- **Tuple** (input control) .......................... number(-array)  \( \sim \) VARIANT (real, integer, string)  
  Input tuple.
- **Index** (input control) .......................... number(-array)  \( \sim \) VARIANT (real, integer)  
  Indices of the elements to select.
- **Selected** (output control) ......................... number(-array)  \( \sim \) VARIANT (real, integer, string)  
  Selected tuple element.

Parallelization Information

**TupleSelect** is reentrant and processed without parallelization.

Alternatives

- **TupleFirstN**, **TupleLastN**, **TupleStrBitSelect**, **TupleConcat**

Module

Select several elements of a tuple.

**TupleSelectRange** selects several consecutive elements of the input tuple **Tuple** and returns them with **Selected**. At this, **Leftindex** determines the index of the first element and **Rightindex** determines the index of the last element to select. Thus both parameters **Leftindex** and **Rightindex** must contain a single integer value (any floating point number must represent an integer value without fraction). Indices of tuple elements start at 0, that means, the first tuple element has got the index 0. The result tuple **Selected** contains every element from the tuple **Tuple** that has got an index between **Leftindex** and **Rightindex** (including the elements at
position \textit{Leftindex} and \textit{Rightindex}). The index \textit{Rightindex} must be greater or equal to \textit{Leftindex}. If the parameters contain equal values, only one element is selected.

\begin{itemize}
  \item \textbf{Tuple} \textit{(input control)} \textit{number(-array)} $\rightsquigarrow$ \textbf{VARIANT} \textit{(real, integer, string)}
  \textit{Input tuple}.
  \item \textbf{Leftindex} \textit{(input control)} \textit{number(-array)} $\rightsquigarrow$ \textbf{VARIANT} \textit{(real, integer)}
  \textit{Index of first element to select}.
  \item \textbf{Rightindex} \textit{(input control)} \textit{number(-array)} $\rightsquigarrow$ \textbf{VARIANT} \textit{(real, integer)}
  \textit{Index of last element to select}.
  \item \textbf{Selected} \textit{(output control)} \textit{number(-array)} $\rightsquigarrow$ \textbf{VARIANT} \textit{(real, integer, string)}
  \textit{Selected tuple elements}.
\end{itemize}

\textbf{Parallelization Information}

\textit{TupleSelectRange} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Alternatives}

\textit{TupleSelect, TupleFirstN, TupleLastN, TupleStrBitSelect, TupleConcat}

\textbf{Module}

\textbf{Operators not requiring licensing}

\begin{verbatim}
[out] VARIANT Selected HTupleX.TupleStrBitSelect ([in] VARIANT Tuple,
[in] VARIANT Index )

void HOperatorSetX.TupleStrBitSelect ([in] VARIANT Tuple,
[in] VARIANT Index, [out] VARIANT Selected )
\end{verbatim}

Select single character or bit from a tuple.
\textbf{TupleStrBitSelect} selects a single character or bit from a tuple \textit{Tuple} of integer numbers and/or strings. The input parameter \textit{Index} determines the character or bit position to select. \textit{Index} must contain a single number. If \textit{Index} contains a floating point number, this may only represent an integer value (without fraction). The result tuple \textit{Selected} contains a new element for each element of \textit{Tuple}. Let \textit{Index} contain the number “\textit{n}” then each element of \textit{Selected} consists of the “\textit{n}-th” character (for strings) or “\textit{n}-th” bit (for integers) of the corresponding element of \textit{Tuple}.

\begin{itemize}
  \item \textbf{Tuple} \textit{(input control)} \textit{number(-array)} $\rightsquigarrow$ \textbf{VARIANT} \textit{(string, integer)}
  \textit{Input tuple}.
  \item \textbf{Index} \textit{(input control)} \textit{number} $\rightsquigarrow$ \textbf{VARIANT} \textit{(real, integer)}
  \textit{Position of character or bit to select}.
  \item \textbf{Selected} \textit{(output control)} \textit{number(-array)} $\rightsquigarrow$ \textbf{VARIANT} \textit{(string, integer)}
  \textit{Tuple containing the selected characters and bits}.
\end{itemize}

\textbf{Parallelization Information}

\textit{TupleStrBitSelect} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Alternatives}

\textit{TupleStrBitSelect, TupleSelect, TupleFirstN, TupleLastN, TupleConcat, TupleStrchr, TupleStrrchr, TupleStrFirstN, TupleStrLastN, TupleAnd, TupleOr, TupleXor, TupleNot}

\textbf{Module}

\textbf{Operators not requiring licensing}

\section{14.10 String-Operators}

\begin{verbatim}
[out] VARIANT Values HTupleX.TupleEnvironment ([in] VARIANT Names )

void HOperatorSetX.TupleEnvironment ([in] VARIANT Names, 
[out] VARIANT Values )
\end{verbatim}

\textit{Read one or more environment variables}.
TupleEnvironment reads the content of all environment variables that are referenced by their names in the input tuple Names and returns the content with the output tuple Values. The input tuple may only contain strings. An empty string is returned for every name within Names that does not denote a valid environment variable.

Parameter

- Names (input control) string(-array) \( \sim \) VARIANT (string)
  Tuple containing name(s) of the environment variable(s).
- Values (output control) string(-array) \( \sim \) VARIANT (string)
  Content of the environment variable(s).

Parallelization Information

TupleEnvironment is reentrant and processed without parallelization.

Alternatives

TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleSplit

Module

Operators not requiring licensing

\[
\begin{align*}
\text{out} & \quad \text{VARIANT S\textit{plitted}} \\
\text{HTupleX.TupleSplit} & \quad ([\text{in}] \text{ VARIANT } T_1, \\
& \quad \text{[in]} \text{ VARIANT } T_2 ) \\
\text{void} & \quad \text{HOperatorSetX.TupleSplit} \quad ([\text{in}] \text{ VARIANT } T_1, \text{[in]} \text{ VARIANT } T_2, \\
& \quad \text{[out]} \text{ VARIANT S\textit{plitted}} )
\end{align*}
\]

Split strings into substrings between predefined separator symbol(s).

TupleSplit searches within the strings of the input tuple T1 for one or more separators defined in the input tuple T2. TupleSplit then splits the examined strings into the substrings between the separators. Both input tuples may only consist of strings. Otherwise TupleSplit returns an error. If the elements of T2 contain more than one character, each character defines a separator. If T1 contains only one string, this is split up several times according to the elements of T2. For example: If T1 consists of the string “data1;data2:7;data3” and T2 contains the strings “;” and “:;”, the output tuple Splitted will comprise the strings “data1”, “data2:7”, “data3” as the result of splitting the string of T1 according to the first element of T2 and “data1”, “data2”, “7” und “data3” as the result of splitting according to the second element of T2. If both input tuples show the same number of elements, the search is done elementwise. I.e., TupleSplit will split the first string of T1 according to the separators in the first element of T2, the second string of T1 according to the separators in the second element of T2 and so on. If T2 only contains one string, the separators defined in this string will be used to split up all the strings of T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleSplit returns an error.

Parameter

- T1 (input control) string(-array) \( \sim \) VARIANT (string)
  Input tuple 1.
- T2 (input control) string(-array) \( \sim \) VARIANT (string)
  Input tuple 2.
- Splitted (output control) string(-array) \( \sim \) VARIANT (string)
  Substrings after splitting the input strings.

Parallelization Information

TupleSplit is reentrant and processed without parallelization.

Alternatives

TupleStrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleEnvironment

Module

Operators not requiring licensing
14.10. STRING-OPERATORS

[out] VARIANT Substring HTupleX.TupleStrFirstN ([in] VARIANT T1,
[in] VARIANT T2 )
void HOperatorSetX.TupleStrFirstN ([in] VARIANT T1, [in] VARIANT T2,
[out] VARIANT Substring )

Cut the first characters up to position “n” out of string tuple.

_tupleStrFirstN_ cuts the first characters up to position “n” out of each string of the input tuple _T1_ and returns them as new strings in the output tuple _Substring_ (remark: the position within strings starts with 0 for the first character of a string). The number “n” is determined by the second input tuple _T2_. If _T2_ only contains one element, this element contains the number “n”. If _T1_ and _T2_ have got the same number of elements, the first element of _T2_ determines the number “n” of characters to cut out of the first string of _T1_, the second element of _T2_ does so for the second string of _T1_ and so on. If _T2_ contains more than one element and _T1_ contains only one string, _TupleStrFirstN_ cuts more than one substrings out of this string. The elements of _T2_ then determine the lengths of these substrings. If both input tuples contain more than one element but differ in the number of elements, _TupleStrFirstN_ returns an error.

Parameter

▷ _T1_ (input control) ........................................ string(-array) \(\sim\) VARIANT ( string )
  Input tuple 1.
▷ _T2_ (input control) ........................................ number(-array) \(\sim\) VARIANT ( real, integer )
  Input tuple 2.
▷ _Substring_ (output control) ................................ string(-array) \(\sim\) VARIANT ( string )
  The first characters up to position “n” of each string.

Parallelization Information

_tupleStrFirstN_ is reentrant and processed without parallelization.

Alternatives

_tupleStrLastN_, _tupleStrrstr_, _tupleStrrstrrstr_, _tupleStrlen_, _tupleStrchr_, _tupleStrrchr_, _tupleSplit_, _tupleEnvironment_

Module

Operators not requiring licensing

[out] VARIANT Substring HTupleX.TupleStrLastN ([in] VARIANT T1,
[in] VARIANT T2 )
void HOperatorSetX.TupleStrLastN ([in] VARIANT T1, [in] VARIANT T2,
[out] VARIANT Substring )

Cut all characters starting at position “n” out of string tuple.

_tupleStrLastN_ cuts all characters from position “n” to the end of the string out of each string of the input tuple _T1_ and returns them as new strings in the output tuple _Substring_. The position “n” is determined by the second input tuple _T2_. If _T2_ only contains one element, this element contains “n”. If _T1_ and _T2_ have got the same number of elements, the first element of _T2_ determines the start position for the first string of _T1_, the second element of _T2_ does so for the second string of _T1_ and so on. If _T2_ contains more than one element and _T1_ contains only one string, _TupleStrLastN_ cuts more than one substrings out of this string. The elements of _T2_ then determine the start positions for these substrings. If both input tuples contain more than one element but differ in the number of elements, _TupleStrLastN_ returns an error.

Parameter

▷ _T1_ (input control) ........................................ string(-array) \(\sim\) VARIANT ( string )
  Input tuple 1.
▷ _T2_ (input control) ........................................ number(-array) \(\sim\) VARIANT ( real, integer )
  Input tuple 2.
▷ _Substring_ (output control) ................................ string(-array) \(\sim\) VARIANT ( string )
  The last characters starting at position “n”.

HALCON 6.1.4
CHAPTER 14. TUPLE

Parallelization Information

TupleStrLastN is reentrant and processed without parallelization.

Alternatives

TupleStrLastN, TupleStrStr, TupleStrRstr, TupleStrLen, TupleStrchr, TupleStrrchr, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ T1 (input control) ... string(-array) \sim \text{VARIANT} (\text{string})</td>
</tr>
<tr>
<td>▶ T2 (input control) ... string(-array) \sim \text{VARIANT} (\text{string})</td>
</tr>
<tr>
<td>▶ Position (output control) ... integer(-array) \sim \text{VARIANT} (\text{integer})</td>
</tr>
</tbody>
</table>

Parallelization Information

TupleStrStrchr is reentrant and processed without parallelization.

Alternatives

TupleStrrchr, TupleStrStr, TupleStrRstr, TupleStrLen, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ T1 (input control) ... string(-array) \sim \text{VARIANT} (\text{string})</td>
</tr>
</tbody>
</table>

Parallelization Information

TupleStrLen is reentrant and processed without parallelization.

Alternatives

TupleStrrchr, TupleStrStr, TupleStrRstr, TupleStrLen, TupleStrchr, TupleStrrchr, TupleSplit, TupleEnvironment

Module

Operators not requiring licensing

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ T1 (input control) ... string(-array) \sim \text{VARIANT} (\text{string})</td>
</tr>
</tbody>
</table>

Length of every string within a tuple of strings.

TupleStrLen checks the length of every string within the input tuple T1 and returns the length of each string with the output tuple Length. All elements of T1 may only consist of strings. Otherwise TupleStrLen returns an error.
14.10. STRING-OPERATORS

Parameter

- **T1** (input control) ... string(-array) => VARIANT ( string )
  Input tuple.
- **Length** (output control) ... integer(-array) => VARIANT ( integer )
  Length of the single strings of the input tuple.

Parallelization Information

**TupleStrlen** is reentrant and processed without parallelization.

Alternatives

TupleStrrstr, TupleStrrstr, TupleStrchr, TupleStrrchr, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Operators not requiring licensing

---

**TupleStrrchr** searches within the strings of the input tuple T1 for the characters of the input tuple T2. Both input tuples may only consist of strings. Otherwise **TupleStrrchr** returns an error. In any case backward search is used, i.e., every string is examined from its last to its first character. If the elements of T2 contain more than one character, only the first character of each element is considered for searching. If T1 contains only one string, all the characters defined in T2 are searched in this string. Thus the output tuple consists of as many elements as T2. Whenever a searched character has been found, the position of its first occurrence gets stored in the output tuple **Position**. If a character can not be found, -1 will be returned instead of its position (remark: the position starts at 0 for the first character of a string). If both input tuples show the same number of elements, the search is done element-wise. I.e., the first character of the first element of T2 is searched within the first string of T1, the first character of the second element of T2 is searched within the second string of T1 and so on. The results of the elementwise searches are returned with **Position** that contains as many elements as T1 and T2. If T2 only contains one string, its first character is searched within all strings of T1. Thus in this case **Position** consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, **TupleStrrchr** returns an error.

Parameter

- **T1** (input control) ... string(-array) => VARIANT ( string )
  Input tuple 1.
- **T2** (input control) ... string(-array) => VARIANT ( string )
  Input tuple 2.
- **Position** (output control) ... integer(-array) => VARIANT ( integer )
  Position of searched character within the string.

Parallelization Information

**TupleStrrchr** is reentrant and processed without parallelization.

Alternatives

TupleStrchr, TupleStrstr, TupleStrrstr, TupleStrlen, TupleStrFirstN, TupleStrLastN, TupleSplit, TupleEnvironment

Operators not requiring licensing
Backward search for string(s) within a string tuple.

TupleStrrstr searches within the strings of the input tuple T1 for the strings of the input tuple T2. Both input tuples may only consist of strings. Otherwise TupleStrrstr returns an error. In any case backward search is used, i.e., every string is examined from its last to its first character. If T1 contains only one string, all strings of T2 are searched in it. Thus the output tuple consists of as many elements as T2. Whenever a searched string has been found, the position of its first occurrence gets stored in the output tuple Position (positions in strings are counted starting with 0). If a string can not be found, -1 will be returned instead of its position. If both input tuples show the same number of elements, the strings are searched elementwise. I.e., the first string of T2 is searched within the first string of T1, the second string of T2 is searched within the second string of T1 and so on. The results of the elementwise searches are returned with Position that contains as many elements as T1 and T2. If T2 only contains one string, this is searched within all strings of T1. Thus in this case Position consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleStrrstr returns an error.

Forward search for string(s) within a string tuple.

TupleStrstr searches within the strings of the input tuple T1 for the strings of the input tuple T2. Both input tuples may only consist of strings. Otherwise TupleStrstr returns an error. If T1 contains only one string, all strings of T2 are searched in it. Thus the output tuple consists of as many elements as T2. Whenever a searched string has been found, the position of its first occurrence gets stored in the output tuple Position (positions in strings are counted starting with 0). If a string can not be found, -1 will be returned instead of its position. If both input tuples show the same number of elements, the strings are searched elementwise. I.e., the first string of T2 is searched within the first string of T1, the second string of T2 is searched within the second string of T1 and so on. The results of the elementwise searches are returned with Position that contains as many elements as T1 and T2. If T2 only contains one string, this is searched within all strings of T1. Thus in this case Position consists of as many elements as T1. If both input tuples contain more than one element and the number of elements differs for the input tuples, TupleStrstr returns an error.
14.10. STRING-OPERATORS

Parameter

- **T1** (input control) .................................................. string(-array) \(\rightarrow\) VARIANT (string)
  - Input tuple 1.

- **T2** (input control) .................................................. string(-array) \(\rightarrow\) VARIANT (string)
  - Input tuple 2.

- **Position** (output control) ...................................... integer(-array) \(\rightarrow\) VARIANT (integer)
  - Position of searched string(s) within the examined string(s).

Parallelization Information

**TupleStrstr** is reentrant and processed without parallelization.

Alternatives

- **TupleStrrstr**, **TupleStrlen**, **TupleStrchr**, **TupleStrrchr**, **TupleStrFirstN**, **TupleStrLastN**, **TupleSplit**, **TupleEnvironment**

Module

Operators not requiring licensing
Chapter 15

XLD

15.1 Access

```
[out] VARIANT Row HXLDContX.GetContourXld ([out] VARIANT Col )
void HOperatorSetX.GetContourXld ([in] IObjectX Contour,
[out] VARIANT Row, [out] VARIANT Col )
```

Return the coordinates of an XLD contour.

GetContourXld returns the following values of the XLD contour Contour:

- **Row**  Row coordinate of the contour’s points
- **Col**  Column coordinate of the contour’s points

Parameter

- Contour (input iconic) xld_cont ~ HXLDContX / IObjectX
  Input XLD contour.
- Row (output control) point.y ~ VARIANT ( real )
  Row coordinate of the contour’s points.
- Col (output control) point.x ~ VARIANT ( real )
  Column coordinate of the contour’s points.

Parallelization Information

GetContourXld is reentrant and processed without parallelization.

Possible Predecessors

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

See also

GetContourAttribXld, QueryContourAttribsXld, GetContourGlobalAttribXld,
QueryContourGlobalAttribsXld

Module

Sub-pixel operators

```
[out] VARIANT BeginRow HXLDPolyX.GetLinesXld ([out] VARIANT BeginCol,
[out] VARIANT EndRow, [out] VARIANT EndCol, [out] VARIANT Length,
[out] VARIANT Phi )
void HOperatorSetX.GetLinesXld ([in] IObjectX Polygon,
[out] VARIANT BeginRow, [out] VARIANT BeginCol, [out] VARIANT EndRow,
[out] VARIANT EndCol, [out] VARIANT Length, [out] VARIANT Phi )
```

Return an XLD polygon’s data (as lines).

GetLinesXld returns the XLD polygon Polygon as a set of lines. The following values are returned:
CHAPTER 15. XLD

**BeginRow:** Rows coordinates of the lines’ start points

**BeginCol:** Columns coordinates of the lines’ start points

**EndRow:** Row coordinates of the lines’ end points

**EndCol:** Column coordinates of the lines’ end points

**Length:** Lengths of the line segments

**Phi:** Angles of the line segments

---

**Polygon** (input iconic) .................. \textit{xld\_poly(-array)} \sim HXLDPolyX / IHObjectX
Input XLD polygons.

\begin{itemize}
  \item **BeginRow** (output control) .................. line.begin.y \sim VARIANT(\text{real})
  Row coordinates of the lines’ start points.
  \item **BeginCol** (output control) .................. line.begin.x \sim VARIANT(\text{real})
  Column coordinates of the lines’ start points.
  \item **EndRow** (output control) .................. line.end.y \sim VARIANT(\text{real})
  Column coordinates of the lines’ end points.
  \item **EndCol** (output control) .................. line.end.x \sim VARIANT(\text{real})
  Column coordinates of the lines’ end points.
  \item **Length** (output control) .................. real \sim VARIANT(\text{real})
  Lengths of the line segments.
  \item **Phi** (output control) .................. angle.rad \sim VARIANT(\text{real})
  Angles of the line segments.
\end{itemize}

---

GetLinesXld is \textit{reentrant} and automatically \textit{parallelized} (on tuple level).

---

**GetLinesXld**

**GenPolygonsXld**

**GetPolygonXld**

---

**Sub-pixel operators**

---

\begin{verbatim}
[out] VARIANT Row1 HXLDX.GetParallelsXld ([out] VARIANT Col1,
[out] VARIANT Length1, [out] VARIANT Phi1, [out] VARIANT Row2,
[out] VARIANT Col2, [out] VARIANT Length2, [out] VARIANT Phi2 )

void HOperatorSetX.GetParallelsXld ([in] IHObjectX Parallels,
[out] VARIANT Row1, [out] VARIANT Col1, [out] VARIANT Length1,
[out] VARIANT Phi1, [out] VARIANT Row2, [out] VARIANT Col2,
[out] VARIANT Length2, [out] VARIANT Phi2 )
\end{verbatim}

---

Return an XLD parallel’s data (as lines).

GetParallelsXld returns the following values of the XLD parallels \textit{Parallels}:

\begin{itemize}
  \item **Row1:** Row coordinates of the points on polygon P1
  \item **Col1:** Column coordinates of the points on polygon P1
  \item **Length1:** Lengths of the line segments on polygon P1
  \item **Phi1:** Angles of the line segments on polygon P1
  \item **Row2:** Row coordinates of the points on polygon P2
  \item **Col2:** Column coordinates of the points on polygon P2
  \item **Length2:** Lengths of the line segments on polygon P2
  \item **Phi2:** Angles of the line segments on polygon P2
\end{itemize}

---

\begin{itemize}
  \item **Parallels** (input iconic) .................. \textit{xld} \sim HXLDX / IHObjectX
  Input XLD parallels.
  \item **Row1** (output control) .................. polygon.y \sim VARIANT(\text{real})
  Row coordinates of the points on polygon P1.
\end{itemize

HALCON/COM Reference Manual, 2005-2-1
Col1 (output control) .......................... polygon.x ~ VARIANT ( real )
  Column coordinates of the points on polygon P1.

Length1 (output control) .......................... real ~ VARIANT ( real )
  Lengths of the line segments on polygon P1.

Phi1 (output control) .............................. angle.rad ~ VARIANT ( real )
  Angles of the line segments on polygon P1.

Row2 (output control) .............................. polygon.y ~ VARIANT ( integer )
  Row coordinates of the points on polygon P2.

Col2 (output control) .............................. polygon.x ~ VARIANT ( integer )
  Column coordinates of the points on polygon P2.

Length2 (output control) .......................... real ~ VARIANT ( real )
  Lengths of the line segments on polygon P2.

Phi2 (output control) .............................. angle.rad ~ VARIANT ( real )
  Angles of the line segments on polygon P2.

--- Parallelization Information
GetParallelsXld is reentrant and processed without parallelization.
--- Possible Predecessors
GenParallelsXld
--- See also
GetPolygonXld, GetLinesXld
--- Module

Sub-pixel operators

[out] VARIANT Row HXLDPolyX.GetPolygonXld ([out] VARIANT Col,
[out] VARIANT Length, [out] VARIANT Phi )

void HOperatorSetX.GetPolygonXld ([in] IHObjectX Polygon,
[out] VARIANT Row, [out] VARIANT Col, [out] VARIANT Length,
[out] VARIANT Phi )

Return an XLD polygon’s data.
GetPolygonXld returns the following values of the XLD polygon Polygon:

Row     Row coordinates of the polygons’ points
Col     Column coordinates of the polygons’ points
Length  Lengths of the line segments between points i and i + 1, respectively
Phi     Angles of the line segments between points i and i + 1, respectively

--- Parallelization Information
GetPolygonXld is reentrant and processed without parallelization.
--- Possible Predecessors
GenPolygonsXld
--- Alternatives
GetLinesXld
Generate a XLD contour with rounded corners from a polygon (given as tuples).

\texttt{GenContourPolygonRoundedXld} generates an XLD contour \textit{Contour} with rounded corners from a polygon given in the tuples \textit{Row} and \textit{Col}. The rounded corners are created in form of arcs defined by \textit{Radius}. For every specified point of the polygon there must be a corresponding rounding radius defined. In case of a closed polygon the first and the last point have to be defined identical with the radii of these points being equal. In contrast, the radii of the first and last point of open polygons are ignored. Finally, the \textit{SamplingInterval} parameter defines the distance of the control points of the formed contour \textit{Contour}.

\begin{itemize}
  \item \textbf{Contour} (output iconic) \texttt{xld\_cont} \rightsquigarrow \texttt{HXLDContX / HUntypedObjectX}
    \begin{itemize}
      \item Resulting contour.
    \end{itemize}
  \item \textbf{Row} (input control) \texttt{number} \rightsquigarrow \texttt{VARIANT( real, integer )}
    \begin{itemize}
      \item Row coordinates of the polygon.
      \hspace{1cm} \textbf{Default Value} : \{20,80,80,20,20\}
      \hspace{1cm} \textbf{Suggested values} : \texttt{Row} \in \{0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500\}
    \end{itemize}
  \item \textbf{Col} (input control) \texttt{number} \rightsquigarrow \texttt{VARIANT( real, integer )}
    \begin{itemize}
      \item Column coordinates of the polygon.
      \hspace{1cm} \textbf{Default Value} : \{20,20,80,80,20\}
      \hspace{1cm} \textbf{Suggested values} : \texttt{Col} \in \{0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500\}
    \end{itemize}
  \item \textbf{Radius} (input control) \texttt{number} \rightsquigarrow \texttt{VARIANT( real, integer )}
    \begin{itemize}
      \item Radii of the rounded corners.
      \hspace{1cm} \textbf{Default Value} : \{20,20,20,20\}
      \hspace{1cm} \textbf{Suggested values} : \texttt{Radius} \in \{0, 1, 2, 5, 10, 20, 50\}
    \end{itemize}
  \item \textbf{SamplingInterval} (input control) \texttt{number} \rightsquigarrow \texttt{VARIANT( real, integer )}
    \begin{itemize}
      \item Distance of the samples.
      \hspace{1cm} \textbf{Default Value} : 1.0
      \hspace{1cm} \textbf{Suggested values} : \texttt{SamplingInterval} \in \{0.5, 1.0, 2.0, 5.0\}
    \end{itemize}
\end{itemize}

\textit{GenContourPolygonRoundedXld} is \textit{reentrant} and processed \textit{without} parallelization.

Parallelization Information

Possible Predecessors

\texttt{GetRegionContour}

Possible Successors

\texttt{SmoothContoursXld, GenPolygonsXld}

See also

\texttt{GenContourPolygonXld, GenContoursSkeletonXld}

Sub-pixel operators
### Generate an XLD contour from a polygon (given as tuples).

**GenContourPolygonXld** generates an XLD contour **Contour** from a polygon given in the tuples **Row** and **Col**. This operator is useful if contours have been obtained from routines outside the core library, but higher level operators, e.g., polygon approximation and extraction of parallels, are to be performed on the contours.

**Parameter**

- **Contour** (output iconic) .................. **xld_cont** \(\leadsto\) **HXLDContX / HUntypedObjectX**
  - Resulting contour.
- **Row** (input control) .................... **number** \(\leadsto\) **VARIANT (real, integer)**
  - Row coordinates of the polygon.
  - Default Value: \([0,1,2,2,2]\)
  - Suggested values: Row \(\in\) \([0,1,2,3,4,5,10,20,50,100,200,500]\)
- **Col** (input control) .................... **number** \(\leadsto\) **VARIANT (real, integer)**
  - Column coordinates of the polygon.
  - Default Value: \([0,0,0,1,2]\)
  - Suggested values: Col \(\in\) \([0,1,2,3,4,5,10,20,50,100,200,500]\)

**Parallelization Information**

**GenContourPolygonXld** is **reentrant** and processed **without** parallelization.

**Possible Predecessors**

- GetRegionContour
- SmoothContoursXld, GenPolygonsXld
- GenContoursSkeletonXld

**Possible Successors**

- See also

**Module**

**Sub-pixel operators**

### Generate XLD contours from regions.

**GenContourRegionXld** generates XLD contours **Contours** from the regions given in **Regions**. This operator is useful if regions have been obtained from segmentation operations, but higher level operators, e.g., polygon approximation and extraction of parallels, are to be performed on their boundaries. For each connected component of the input regions a closed contour of the boundary is generated. The parameter **Mode** can take on the following values:

- 'center': The centers of the border pixels are used as contour points.
- 'border': The outer border of the border pixels is used as contour points.
- 'border_holes': In addition to the outer border of the input region you get the contours of all holes.

The difference between the two modes 'border' and 'center' can be seen by considering the following region:

```
  □ □ □ □ □
  □ □ □ □ □
  □ □ □ □ □
  □ □ □ □ □
  □ □ □ □ □
```

where □ symbolizes a single pixel.
Then, computing the contour with ‘border’ and ‘center’ results in the following two contours, respectively:

![Two contours](image)

This means, for example, that contours generated with ‘border’ will in general have a much larger Euclidean length (see LengthXld) than contours generated with ‘center’. This results from the fact that for diagonal border elements ‘border’ uses two contour segments of length 1 each, whereas ‘center’ uses a single segment of length \( \sqrt{2} \). Other features, e.g., the area (see AreaCenterXld), will obviously also have different values.

Parameter

- Regions (input iconic) .............................. region(-array) \( \sim \) HRegionX / IObjectX
  Input regions.
- Contours (output iconic) ......................... xld_cont(-array) \( \sim \) HXLDContX / HUntypedObjectX
  Resulting contours.
- Mode (input control) .............................. string \( \sim \) String / VARIANT
  Mode of contour generation.
  Default Value : ‘border’
  List of values : Mode \( \in \) \{‘border’, ‘border_holes’, ‘center’\}

Parallelization Information

GenContourRegionXld is reentrant and processed without parallelization.

Possible Successors

SmoothContoursXld, GenPolygonsXld
GenContoursSkeletonXld
See also
GenContourPolygonXld, GetRegionContour
Alternatives

Module

Sub-pixel operators

```h
```

Convert a skeleton into XLD contours.

GenContoursSkeletonXld converts the input skeleton (e.g., edges) Skeleton, which is assumed to contain mostly one pixel wide regions (see Skeleton), into XLD contours. The regions are first transformed to contain only line segments in 8-neighborhood. In the process 12 special configurations are taken into account: Points for which there is a junction of three or more lines in 8-neighborhood are preserved (in all four rotations):

```
0 0 1 0 1 0 0 1 0
1 1 0 0 1 1 1 1
0 1 0 0 1 0 0 1 0
```

In a second step, all junction points are labelled, taking six different characteristic configurations of all four possible rotations into account:

```
1 0 1 1 0 0 1 0 0
0 2 0 0 2 1 0 2 1 0
0 0 1 1 0 1 0 1 0 0
```

HALCON/COM Reference Manual, 2005-2-1
15.2. Creation

where 0 = background, 1 = foreground, and 2 = junction point.

After this, all contours having at least \texttt{Length} points are returned. The contours generated by \texttt{GenContoursSkeletonXld} always end in junction or end points. For closed contours the first point lies in the 8-neighborhood of the last point of the contour. Therefore, in order to determine the adjacency of contours it is sufficient to just take the end points into account.

Since contours are split at junction points, possibly long contours may be split into several short segments because of short adjacent lines, even if they are longer than \texttt{Length} points, if \texttt{Mode} = 'filter' was selected. This can be avoided by setting \texttt{Mode} = 'generalize1'. In this case, the contours are generated as if the segments shorter than \texttt{Length} were not contained in the input region. In order to preserve line segments, which are split into very short segments by the crossing of short lines, \texttt{Mode} = 'generalize2' can be selected. In this case, line segments are preserved if they end in two junction points, even if they are shorter than \texttt{Length}.

---

**Parameter**

- **Skeleton** (input iconic) \texttt{region} \leadsto \texttt{HRegionX / IHObjectX}  
  Skeleton of which the contours are to be determined.
- **Contours** (output iconic) \texttt{xld_cont} \leadsto \texttt{HXLDContX / HUntypedObjectX}  
  Resulting contours.
- **Length** (input control) \texttt{integer} \leadsto \texttt{long / VARIANT}  
  Minimum number of points a contour has to have.
  \textbf{Default Value :} 1  
  \textbf{Suggested values :} \texttt{Length} \in \{1, 2, 3, 5, 10, 20\}
- **Mode** (input control) \texttt{string} \leadsto \texttt{String / VARIANT}  
  Contour filter mode.
  \textbf{Default Value :} 'filter'  
  \textbf{List of values :} \texttt{Mode} \in \{'filter', 'generalize1', 'generalize2'\}

---

**Parallelization Information**

\texttt{GenContoursSkeletonXld} is reentrant and processed without parallelization.

---

**Possible Predecessors**

- \texttt{Skeleton}
- \texttt{SmoothContoursXld, GetContourXld, GenPolygonsXld}

---

**Possible Successors**

- \texttt{EdgesImage, Threshold, GetRegionContour}

---

**See also**

- Module

---

**Sub-pixel operators**

```c
void HXLDContX.GenEllipseContourXld ([in] VARIANT Row,  
[in] VARIANT Column, [in] VARIANT Phi, [in] VARIANT Radius1,  
in] VARIANT PointOrder, [in] double Resolution )

void HOperatorSetX.GenEllipseContourXld  
([out] HUntypedObjectX ContEllipse, [in] VARIANT Row, [in] VARIANT Column,  
in] VARIANT Phi, [in] VARIANT Radius1, [in] VARIANT Radius2,  
in] VARIANT StartPhi, [in] VARIANT EndPhi, [in] VARIANT PointOrder,  
in] VARIANT Resolution )
```

**Creation of an XLD contour corresponding to an elliptic arc.**

\texttt{GenEllipseContourXld} creates one or more elliptic arcs or closed ellipses. Ellipses are specified by their center (\texttt{Row, Column}), the orientation of the main axis \texttt{Phi}, the length of the larger half axis \texttt{Radius1}, and the length of the smaller half axis \texttt{Radius2}. In addition to that, elliptic arcs are characterized by the angle of the start point \texttt{StartPhi}, the angle of the end point \texttt{EndPhi}, and the point order \texttt{PointOrder} along the boundary. The angles in the interval [0, 2\pi] are measured in the coordinate system of the ellipse relative to the main axis. Thus, the two main poles correspond to the angles 0 and \pi, the two other poles to the angles \pi/2 and 3\pi/2. To create a closed ellipse the values 0 and 2\pi (with positive point order) have to be passed to the operator. The resolution of the
resulting contours ContEllipse is controlled via Resolution containing the maximum Euclidean distance between neighboring contour points.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContEllipse</td>
<td>xld_cont(-array)</td>
<td>Resulting contour.</td>
</tr>
<tr>
<td>Row</td>
<td>ellipse.center.y(-array)</td>
<td>Row coordinate of the center of the ellipse.</td>
</tr>
<tr>
<td>Column</td>
<td>ellipse.center.x(-array)</td>
<td>Column coordinate of the center of the ellipse.</td>
</tr>
<tr>
<td>Phi</td>
<td>ellipse.angle.rad(-array)</td>
<td>Orientation of the main axis [rad].</td>
</tr>
<tr>
<td>Radius1</td>
<td>ellipse.radius1(-array)</td>
<td>Length of the larger half axis.</td>
</tr>
<tr>
<td>Radius2</td>
<td>ellipse.radius2(-array)</td>
<td>Length of the smaller half axis.</td>
</tr>
<tr>
<td>StartPhi</td>
<td>real(-array)</td>
<td>Angle of the start point [rad].</td>
</tr>
<tr>
<td>EndPhi</td>
<td>real(-array)</td>
<td>Angle of the end point [rad].</td>
</tr>
<tr>
<td>PointOrder</td>
<td>string(-array)</td>
<td>Point order along the boundary.</td>
</tr>
<tr>
<td>Resolution</td>
<td>double / VARIANT</td>
<td>Resolution: Maximum distance between neighboring contour points.</td>
</tr>
</tbody>
</table>

Example
draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
gen_ellipse_contour_xld(Ellipse, Row, Column, Phi, Radius1, Radius2, 0, 6.28319, 'positive', 1.5)
length_xld(Ellipse, Length).

GenEllipseContourXld returns TRUE if all parameter values are correct. If necessary, an exception is raised.
15.2. CREATION

--- Parallelization Information ---

*GenEllipseContourXld* is *reentrant* and processed *without* parallelization.

--- Possible Predecessors ---

*DrawEllipse*

--- Possible Successors ---

*DispXld, GetPointsEllipse*

--- Module ---

Sub-pixel operators

---

```
```

```
```

---

**Extract parallel XLD polygons.**

*GenParallelsXld* examines the XLD polygons passed in *Polygons* for parallelism. The resulting parallel polygons are returned in *Parallels*. If the parameter *Merge* is set to ‘true’, adjacent parallel polygons are returned in a single parallel relation. Otherwise, one parallel relation is returned for each pair of parallel line segments. Whether two polygon segments are parallel depends on their distance (smaller than *Dist*), a maximum allowed angle difference (*Alpha*, in radians), and a minimum length of the two polygon segments. Furthermore, the two segments have to overlap. As a side effect, a quality factor is calculated for each pair of parallels. It is based on the normalized angular difference and the normalized length of the overlapping area:

\[
\text{quality} = \frac{\pi - \delta \alpha}{\pi/2} \frac{2 \text{overlap}}{\text{len}_1 + \text{len}_2}
\]

with \(0 \leq q \leq 1\)

Here, \(\delta \alpha\) is the angle difference of the polygon segments, \(\text{overlap}\) is the length of the overlap area, \(\text{len}_1\) and \(\text{len}_2\) the length of the polygon segments, and \(\text{quality}\) the resulting quality factor.

The quality factor is a measure of parallelism (the larger its value, the “more parallel” the polygons). Finally, the quality factors of all parallel polygon segments contained in a single polygon are added, weighted with their length of the overlapping area.

--- Parameter ---

- **Polygons** (input iconic) .................xld_poly ~ HXLDPolyX / IHObjectX
  Input polygons.

- **Parallels** (output iconic) ..................xld_para ~ HXLDParaX / HUntypedObjectX
  Parallel polygons.

- **Len** (input control) ..........................number ~ VARIANT (real, integer)
  Minimum length of the individual polygon segments.
  Default Value: 10.0
  Suggested values: Len ∈ \{5.0, 10.0, 15.0, 20.0\}
  Restriction: (Len > 0.0)

- **Dist** (input control) ..........................number ~ VARIANT (real, integer)
  Maximum distance between the polygon segments.
  Default Value: 30.0
  Suggested values: Dist ∈ \{20.0, 25.0, 30.0, 40.0, 50.0, 75.0\}
  Restriction: (Dist > 0.0)

- **Alpha** (input control) ......................number ~ VARIANT (real, integer)
  Maximum angle difference of the polygon segments.
  Default Value: 0.15
  Suggested values: Alpha ∈ \{0.05, 0.10, 0.15, 0.20, 0.30\}
  Restriction: \(0 \leq \alpha \leq (\pi/2)\)
Approximate XLD contours by polygons.

**GenPolygonsXld** approximates XLD contours (**Contours**) by polygons. The type of the approximation can be set by **Type**. The threshold for the approximation is set via **Alpha**. The procedure is able to process open as well as closed contours. The resulting approximating XLD polygons are returned in **Polygons**.

Contours can be approximated by the algorithms of Ramer, Ray, and Sato. The algorithm of Ramer approximates contours such that the Euclidean distance of the approximating polygon to the contour is at most **Alpha** pixel units. The algorithm of Ray does not need a threshold, and hence **Alpha** is ignored. Here, the contour is approximated by maximizing the line length, while minimizing the sum of the distances of the line segment from the contour. The algorithm of Sato produces a polygon point at the point of the contour in which the distance to the end points of the contour is maximal. The total approximation error for each iteration is then given by \( \frac{(L - L')}{L} \), where \( L \) is the Euclidean contour length, and \( L' \) is the length of the approximating polygon.

### Parameter

- **Contours** (input iconic) \( \bowtie \) xld_cont \( \rightarrow \) **HXLDContX / IHObjectX**
  Contours to be approximated.
- **Polygons** (output iconic) \( \bowtie \) xld_poly \( \rightarrow \) **HXLDPolyX / HUntypedObjectX**
  Approximating polygons.
- **Type** (input control) \( \bowtie \) string \( \rightarrow \) **String / VARIANT**
  Type of approximation.
  Default Value: ‘ramer’
  List of values: **Type** \( \in \) {‘ramer’, ‘ray’, ‘sato’}
- **Alpha** (input control) \( \bowtie \) number \( \rightarrow \) **VARIANT** (real, integer)
  Threshold for the approximation.
  Default Value: 2.0
  Suggested values: **Alpha** \( \in \) \{1.0, 1.5, 2.0, 3.0, 4.0\}
  Restriction: \( (\text{Alpha} > 0.0) \)

**GenPolygonsXld** is reentrant and processed without parallelization.

---

**Sub-pixel operators**

HALCON/COM Reference Manual, 2005-2-1
### Extract parallel XLD polygons enclosing a homogeneous area.

**ModParallelsXld** selects XLD parallels enclosing homogeneous areas from the input parallels **Parallels**. The parameter **Image** contains the corresponding gray value image.

Only parallels having a quality factor larger than **Quality** are examined. The algorithm performs parallel cross sections one pixel apart, and parallel to the line segments in the area of overlap between two parallel line segments.

In the first iteration, the mean gray value for each of the lines in the cross section is calculated. In the second iteration, the standard deviations of the gray values along each line are computed.

If the mean gray value of an area between parallels lies in the interval [MinGray, MaxGray], and if the mean of all standard deviations is smaller than the upper threshold **MaxStandard**, the corresponding parallels are returned as modified parallels in **ModParallels**.

In a second step, all polygon segments adjacent to parallels bordering homogeneous areas are checked for homogeneity. To do so, a rectangle having the width of the last area enclosed by a modified parallel is constructed, and checked for homogeneity using the algorithm described above. This process is continued as long as there are adjacent polygon segments. The polygons thus found are returned as extended parallels in **ExtParallels**.

---

**Parameter**

- **Parallels** (input iconic) .......................... xld | Para \( \sim \) HXLDParaX / IObjectX
  - Input XLD parallels.
- **Image** (input iconic) .......................... image \( \sim \) HImageX / IObjectX (byte)
  - Corresponding gray value image.
- **ModParallels** (output iconic) ................. xld | mod | para \( \sim \) HXLDMdParaX / HUntypedObjectX
  - Modified XLD parallels.
- **ExtParallels** (output iconic) ................. xld | ext | para \( \sim \) HXLDExtParaX / HUntypedObjectX
  - Modified XLD parallels.
- **Quality** (input control) .......................... number \( \sim \) VARIANT (real, integer)
  - Minimum quality factor (measure of parallelism).
    - Default Value: 0.4
    - Suggested values: Quality \( \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}\)
    - Restriction: \((0.0 \leq \text{Quality}) \land (\text{Quality} \leq 1.0)\)
- **MinGray** (input control) .......................... integer \( \sim \) long / VARIANT
  - Minimum mean gray value.
    - Default Value: 160
    - Suggested values: MinGray \( \in \{80, 100, 120, 140, 160, 180\}\)
    - Restriction: \((0 \leq \text{MinGray}) \land (\text{MinGray} \leq 255)\)
- **MaxGray** (input control) .......................... integer \( \sim \) long / VARIANT
  - Maximum mean gray value.
    - Default Value: 220
    - Suggested values: MaxGray \( \in \{140, 160, 180, 200, 220, 240\}\)
    - Restriction: \((0 \leq \text{MaxGray}) \land (\text{MaxGray} \leq 255)\) \land (MaxGray \geq \text{MinGray})
- **MaxStandard** (input control) ...................... number \( \sim \) VARIANT (real, integer)
  - Maximum allowed standard deviation.
    - Default Value: 10.0
    - Suggested values: MaxStandard \( \in \{5.0, 10.0, 15.0, 20.0\}\)
    - Restriction: \((\text{MaxStandard} \geq 0.0)\)

---

**Parallelization Information**

**ModParallelsXld** is reentrant and processed without parallelization.

---

**Possible Predecessors**

- **GenParallelsXld**

---

HALCON 6.1.4
CHAPTER 15. XLD

15.3 Features

Area and center of gravity (centroid) of contours and polygons.

**AreaCenterXld** calculates the area and center of gravity (centroid) of the region enclosed by the contours or polygons XLD as well as the order of the points along the boundary. The area and centroid are computed by applying Green's theorem using only the points on the contour or polygon, i.e., no region is generated explicitly for the purpose of calculating the features. If the points are arranged counterclockwise (i.e., in a positive mathematical sense) in the contour, **PointOrder** will be "positive". It is assumed that the contours or polygons are closed. If this is not the case **AreaCenterXld** will artificially close the contour. If more than one contour or polygon is passed the results are stored as tuples in which the index of a value corresponds to the index of the respective contour or polygon in XLD.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>XLD</strong> (input iconic) xld(-array) \sim IHXLDX / IHObjectX Contours or polygons to be examined.</td>
</tr>
<tr>
<td><strong>Area</strong> (output control) real(-array) \sim VARIANT (real) Area enclosed by the contour or polygon.</td>
</tr>
<tr>
<td><strong>Row</strong> (output control) point.y(-array) \sim VARIANT (real) Row coordinate of the centroid.</td>
</tr>
<tr>
<td><strong>Column</strong> (output control) point.x(-array) \sim VARIANT (real) Column coordinate of the centroid.</td>
</tr>
<tr>
<td><strong>PointOrder</strong> (output control) string(-array) \sim VARIANT (string) point order along the boundary (&quot;positive&quot;/&quot;negative&quot;).</td>
</tr>
</tbody>
</table>

**Complexity**

Let \( n \) be the number of points of the contour or polygon. Then the run time is \( O(n) \).

**Result**

**AreaCenterXld** returns TRUE if the input is not empty. If the input is empty the behaviour can be set via **SetSystem**(::’noObjectResult’,<Result>:). If necessary, an exception is raised.

**Parallelization Information**

**AreaCenterXld** is reentrant and automatically parallelized (on tuple level).

---

Possible Predecessors

**GenContoursSkeletonXld**, **SmoothContoursXld**, **GenPolygonsXld**

---

See also

**MomentsXld**, **MomentsAnyXld**, **AreaCenter**, **MomentsRegion2Nd**

---

Module

Sub-pixel operators
### 15.3. FEATURES

**VARIANT** `CircleXld`  
`void HOperatorSetX.CircularityXld ([in] IHObjectX XLD, [out] VARIANT Circularity)`  

**Shape factor for the circularity (similarity to a circle) of a contour.**

The operator `CircularityXld` calculates the similarity of the input contour with a circle. The input contour must not intersect itself, otherwise the resulting parameter is not meaningful. If the input contour is not closed it will be closed automatically.

**Calculation:** If $F$ is the enclosing area of the contour and $\text{max}$ is the maximum distance from the center to all contour pixels, the shape factor $\text{Circularity}$ is defined as:

$$\text{Circularity} = \frac{F}{(\text{max}^2 + \pi)}$$

The shape factor $\text{Circularity}$ of a circle is 1. If the contour encloses an elongated area $\text{Circularity}$ is smaller than 1. The operator `CircularityXld` especially responds to large bulges.

If more than one contour is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the contour in the input tuple.

**Parameter**

- `XLD` (input iconic)  
  Contour(s) to be examined.

- `Circularity` (output control)  
  Roundness of the input contour(s).

**Restriction:** $((0 \leq \text{Circularity}) \land (\text{Circularity} \leq 1.0))$

The operator `CircularityXld` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input contour available) is set via the operator `SetSystem('noObjectResult',<Result>)`. If necessary an exception is raised.

**Parallelization Information**

`CircularityXld` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

`GenContoursSkeletonXld`, `EdgesSubPix`, `ThresholdSubPix`, `GenContourPolygonXld`

**See also**

`AreaCenterXld`, `SelectShapeXld`

**Alternatives**

`CompactnessXld`, `ConvexityXld`, `EccentricityXld`

**Module**

Sub-pixel operators

---

**VARIANT** `Compactness XLD`  
`void HOperatorSetX.CompactnessXld ([in] IHObjectX XLD, [out] VARIANT Compactness)`  

**Shape factor for the compactness of a contour.**

The operator `CompactnessXld` calculates the compactness of the input contour `XLD`. The input contour must not intersect itself, otherwise the resulting parameter is not meaningful. If the input contour is not closed it will be closed automatically.

**Calculation:** If $L$ is the length of the contour and $F$ the enclosing area of the contour the shape factor `Compactness` is defined as:

$$\text{Compactness} = \frac{L^2}{4F\pi}$$

HALCON 6.1.4
The shape factor \textit{Compactness} of a circle is 1. If the contour encloses an elongated area \textit{Compactness} is larger than 1. The operator \textit{CompactnessXld} responds to the course of the contour (roughness). If more than one contour is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the contour in the input tuple.

\[ \text{CompactnessXld} \]

**Parameter**

- **\( \text{XLD} \) (input iconic)** \( \text{xld(-array)} \sim \text{IHLDX / IObjectX} \)
- **\textit{Compactness} (output control)** \( \text{real(-array)} \sim \text{VARIANT ( real )} \)

\textit{Restriction}: \((\text{Compactness} \geq 1.0) \lor (\text{Compactness} = 0)\)

**Result**

The operator \textit{CompactnessXld} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input contours available) is set via the operator \textit{SetSystem(‘noObjectResult’,<Result>).} If necessary an exception is raised.

**Parallelization Information**

\textit{CompactnessXld} is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

\textit{GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld}

**See also**

\textit{AreaCenterXld, SelectShapeXld}

**Alternatives**

\textit{CircularityXld, ConvexityXld, EccentricityXld}

**Module**

Sub-pixel operators

\[ \text{HXLDContX.ContourPointNumXld} \]

Return the number of points in an XLD contour.

\textit{ContourPointNumXld} returns the length of the input contour \textit{Contour}, i.e., its number of points, in \textit{Length}.

**Parameter**

- **\textit{Contour} (input iconic)** \( \text{xld(-cont)} \sim \text{HXLDContX / IObjectX} \)
- **\textit{Length} (output control)** \( \text{integer} \sim \text{long / VARIANT} \)

**Parallelization Information**

\textit{ContourPointNumXld} is reentrant and processed without parallelization.

**Possible Predecessors**

\textit{GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix}

**Possible Successors**

\textit{GetContourXld, GetContourAttribXld}

**See also**

\textit{QueryContourAttribsXld}

**Alternatives**

\textit{GetRegressParamsXld}

**Module**

Sub-pixel operators
### 15.3. FEATURES

<table>
<thead>
<tr>
<th>out</th>
<th>VARIANT Convexity</th>
<th>HXLDX.ConvexityXld</th>
<th>()</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>HOperatorSetX.ConvexityXld</td>
<td>([in] IHObjectX XLD, [out] VARIANT Convexity)</td>
<td></td>
</tr>
</tbody>
</table>

Shape factor for the convexity of a contour.

The operator `ConvexityXld` calculates the convexity of the input contour. The input contour must not intersect itself, otherwise the resulting parameter is not meaningful. If the input contour is not closed it will be closed automatically.

**Calculation:** If $F_c$ is the area of the convex hull and $F_o$ the area enclosed by the original contour the shape factor `Convexity` is defined as:

$$\text{Convexity} = \frac{F_o}{F_c}$$

The shape factor `Convexity` is 1 if the contour is convex (e.g., a rectangle, circle, etc.). If there are indentations `Convexity` is smaller than 1.

If more than one contour is passed the numerical values of the shape factor are stored in a tuple, the position of a value in the tuple corresponding to the position of the contour in the input tuple.

<table>
<thead>
<tr>
<th>XLD (input iconic)</th>
<th>xld(-array)</th>
<th>IHXLDX / IHObjectX Contour(s) to be examined.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convexity (output control)</td>
<td>real(-array)</td>
<td>VARIANT ( real ) Convexity of the input contour(s).</td>
</tr>
</tbody>
</table>

**Restriction:** $(\text{Convexity} \leq 1)$

The operator `ConvexityXld` returns the value TRUE if the input is not empty. The behavior in case of empty input (no input contour available) is set via the operator `SetSystem('noObjectResult',<Result>). If necessary an exception is raised.

**Parallelization Information**

`ConvexityXld` is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld

**See also**

AreaCenterXld, SelectShapeXld, ShapeTransXld

**Module**

Sub-pixel operators

| out | VARIANT Row1 | HXLDX.DiameterXld | ([out] VARIANT Column1, [out] VARIANT Column2, [out] VARIANT Diameter) |
|-----|---------------|-------------------|-----------------|-----------------|-----------------|
| void | HOperatorSetX.DiameterXld | ([in] IHObjectX XLD, [out] VARIANT Row1, [out] VARIANT Column1, [out] VARIANT Column2, [out] VARIANT Diameter) |

Maximum distance between two contour points.

The operator `DiameterXld` calculates the maximum distance between two contour points. The coordinates of these two extremes and the distance between them will be returned. The input contour must not intersect itself, otherwise the resulting parameters are not meaningful. If the input contour is not closed it will be closed automatically.

**Attention**

If the contour is empty, the results of `Row1`, `Column1`, `Row2` and `Column2` (all of them = 0) may lead to confusion.
XLD (input iconic) .......................... xld(-array) \sim IHXLDX / IXObjectX
Contour(s) to be examined.
Row1 (output control) .......................... line.begin.y(-array) \sim VARIANT (real)
Row coordinate of the first extreme point of the contour(s).
Column1 (output control) .......................... line.begin.x(-array) \sim VARIANT (real)
Column coordinate of the first extreme point of the contour(s).
Row2 (output control) .......................... line.end.y(-array) \sim VARIANT (real)
Row coordinate of the second extreme point of the contour(s).
Column2 (output control) .......................... line.end.x(-array) \sim VARIANT (real)
Column coordinate of the second extreme point of the contour(s).
Diameter (output control) .......................... number(-array) \sim VARIANT (real)
Distance of the two extreme points of the contour(s).

DiameterXld returns the value TRUE, if the input is not empty. The reaction to empty input (no input contours are available) may be determined with the help of the operator SetSystem ('noObjectResult',<Result>). If necessary an exception is raised.

DiameterXld is reentrant and automatically parallelized (on tuple level).

Distance of contours to an ellipse.

DistEllipseContourXld determines the distance between the contours in Contours and an ellipse specified by the center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2. Different measures for the distance of a contour point \( X = (x_i, y_i) \) to the ellipse are available (Mode):

'geometric' The distance is measured by the deviation \( X F_1 + X F_2 - 2a \), where \( F_1, F_2 \) are the focal points and \( a \) corresponds to Radius1. This measure shows a low curvature bias: Near points of high curvature on the ellipse (like the poles on the main axis) the distance is larger than near points with low curvature.

'bisec' The distance is measured by the distance between \( X \) and the intersection of the angular bisector of the two lines through \( X \) and the focal points with the ellipse. This is a very good approximation of the orthogonal distance from \( X \) to the ellipse.
The operator returns the minimum absolute distance \( \text{MinDist} \), the maximum absolute distance \( \text{MaxDist} \), the average absolute distance \( \text{AvgDist} \), and the standard deviation of the absolute distances \( \text{SigmaDist} \) of all contour points.

To reduce the computational load, the computation of the distances can be restricted to a subset of the contour points: If a value other than -1 is assigned to \( \text{MaxNumPoints} \), only up to \( \text{MaxNumPoints} \) points - uniformly distributed over the contour - are used. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude \( \text{ClippingEndPoints} \) points at the beginning and at the end of a contour from the computation.

### Parameters

- **Contours** (input iconic) \( \text{xld} \)  
  Input contours.
- **Mode** (input control) \( \text{string} \)  
  Method for the determination of the distances.  
  Default Value: ‘bisec’  
  List of values: Mode \( \in \{ \text{geometric}, \text{algebraic}, \text{bisec} \} \)
- **MaxNumPoints** (input control) \( \text{integer} \)  
  Maximum number of contour points used for the computation (-1 for all points).  
  Default Value: -1  
  Restriction: \((\text{MaxNumPoints} \geq 3)\)
- **ClippingEndPoints** (input control) \( \text{integer} \)  
  Number of points at the beginning and the end of the contours to be ignored for the computation of distances.  
  Default Value: 0  
  Restriction: \((\text{ClippingEndPoints} \geq 0)\)
- **Row** (input control) \( \text{ellipse.center.y} \)  
  Row coordinate of the center of the ellipse of the ellipse.
- **Column** (input control) \( \text{ellipse.center.x} \)  
  Column coordinate of the center of the ellipse.
- **Phi** (input control) \( \text{ellipse.angle.rad} \)  
  Orientation of the main axis \( [\text{rad}] \).  
  Restriction: \(( (\text{Phi} \geq 0) \land (\text{Phi} \leq 6.283185307))\)
- **Radius1** (input control) \( \text{ellipse.radius1} \)  
  Length of the larger half axis.  
  Restriction: \((\text{Radius1} > 0)\)
- **Radius2** (input control) \( \text{ellipse.radius2} \)  
  Length of the smaller half axis.  
  Restriction: \(( (\text{Radius2} \geq 0) \land (\text{Radius2} \leq \text{Radius1}))\)
- **MinDist** (output control) \( \text{real} \)  
  Minimum distance.
- **MaxDist** (output control) \( \text{real} \)  
  Maximum distance.
- **AvgDist** (output control) \( \text{real} \)  
  Mean distance.
- **SigmaDist** (output control) \( \text{real} \)  
  Standard deviation of the distance.

### Example

```
read_image (Image, 'caltab')
find_caltab (Image, Caltab, 'caltab_big.descr', 3, 112, 5)
reduce_domain (Image, Caltab, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
select_contours_xld (Edges, EdgesClosed, 'closed', 0, 2.0, 0, 0)
select_contours_xld (EdgesClosed, EdgesMarks, 'contour_length', 20, 100, 0, 0)
fit_ellipse_contour_xld (EdgesMarks, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0, 
                          Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder)
```
for $i := 0$ to $|\text{Row}| - 1$ by 1
  \text{Object}\_\text{Selected} := \text{EdgesMarks}[i+1]
  \text{dist\_ellipse\_contour\_xld} (\text{Object\_Selected}, \text{‘bisec’}, -1, 0, \text{Row}[i], \text{Column}[i], \text{Phi}[i], \text{Radius1}[i], \text{Radius2}[i], \text{MinDist}, \text{MaxDist}, \text{AvgDist}, \text{SigmaDist})
endfor.

\text{Result}
\text{DistEllipseContourXld} \text{ returns TRUE if all parameter values are correct. If necessary, an exception is raised.}

\text{Parallelization Information}
\text{DistEllipseContourXld} \text{ is local and processed completely exclusively without parallelization.}

\text{Possible Predecessors}
\text{FitEllipseContourXld}

\text{Module}
\text{Sub-pixel operators}

\begin{verbatim}
[out] VARIANT \text{Anisometry HXLDX.EccentricityXld} ([out] VARIANT \text{Bulkiness, [out] VARIANT \text{StructureFactor })
void \text{HOperatorSetX.EccentricityXld} ([in] \text{IHOBJECTX XLD, [out] VARIANT Anisometry, [out] VARIANT Bulkiness, [out] VARIANT StructureFactor })
\end{verbatim}

\text{Shape features derived from the ellipse parameters.}
The operator \text{EccentricityXld} calculates the three shape features \text{Anisometry}, \text{Bulkiness}, and \text{StructureFactor} derived from the geometric moments. The input contour must not intersect itself, otherwise the resulting parameters are not meaningful. If the input contour is not closed it will be closed automatically.

\text{Definition:} If the ellipse radii $Ra, Rb$ (\text{EllipticAxisXld}) and the enclosing area $A$ of the contour are given, the following applies:

\begin{align*}
\text{Anisometry} & = \frac{Ra}{Rb} \\
\text{Bulkiness} & = \pi \cdot \frac{Ra \cdot Rb}{A} \\
\text{StructureFactor} & = \text{Anisometry} \cdot \text{Bulkiness} - 1
\end{align*}

The anisometry of a circle is 1.0.

If more than one contour is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a contour in the input.

\text{Parameter}

\begin{itemize}
  \item \text{XLD} (input iconic) \text{xld(-array)} \sim \text{IHLDX / IHOBJECTX Contour(s) to be examined.}
  \item \text{Anisometry} (output control) \text{real(-array)} \sim \text{VARIANT ( real ) Anisometry of the contour(s).}
    \text{Restriction:} (\text{Anisometry} \geq 1.0)
  \item \text{Bulkiness} (output control) \text{real(-array)} \sim \text{VARIANT ( real ) Bulkiness of the contour(s).}
  \item \text{StructureFactor} (output control) \text{real(-array)} \sim \text{VARIANT ( real ) Structure factor of the contour(s).}
\end{itemize}

\text{Result}
\text{The operator \text{EccentricityXld} returns the value TRUE if the input is not empty. The behavior in case of empty input (no input contours available) is set via the operator \text{SetSystem (‘noObjectResult’,<Result>). If necessary an exception is raised.}}
EccentricityXld is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld

See also
EllipticAxisXld, MomentsRegion2Nd, SelectShapeXld, AreaCenterXld

Sub-pixel operators

```c
[out] VARIANT Ra HXLDX.EllipticAxisXld ([out] VARIANT Rb,
[out] VARIANT Phi )
```

void HOperatorSetX.EllipticAxisXld ([in] IHObjectX XLD,
[out] VARIANT Ra, [out] VARIANT Rb, [out] VARIANT Phi )

Parameters of the equivalent ellipse.
The operator EllipticAxisXld calculates the radii and the orientation of the ellipse having the “same orientation” and the “same aspect ratio” as the input contour. The closed input contour must not intersect itself, otherwise the resulting parameters are not meaningful. Several input contours can be passed in XLD as tuples. The length of the major radius Ra and the minor radius Rb as well as the orientation of the main axis with regard to the horizontal (Phi) are determined. The angle is indicated in radians.

Calculation:
If the moments $M_{20}$, $M_{02}$ and $M_{11}$ are normalized and passed to the area (see MomentsXld), the radii Ra and Rb are calculated as:

$$
Ra = \sqrt{ \frac{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}{2} }
$$

$$
Rb = \sqrt{ \frac{8(M_{20} + M_{02} - \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}{2} }
$$

The orientation Phi is defined by:

$$
Phi = -0.5\tan(2M_{11}, M_{02} - M_{20})
$$

If more than one contour is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a contour in the input.

Parameter

- **XLD** (input iconic) xld(-array) ~ IHXLDX / IHObjectX Contour(s) to be examined.
- **Ra** (output control) real(-array) ~ VARIANT (real) Major radius.
  **Restriction:** $Ra \geq 0.0$
- **Rb** (output control) real(-array) ~ VARIANT (real) Minor radius.
  **Restriction:** $(Rb \geq 0.0) \land (Rb \leq Ra)$
- **Phi** (output control) real(-array) ~ VARIANT (real) Angle between the major axis and the x axis (radians).
  **Restriction:** $(-(\pi)/2 < Phi) \land (Phi \leq (\pi)/2))$

Complexity
If $N$ is the number of contour points, the runtime complexity is $O(N)$.

Result
EllipticAxisXld returns TRUE if the input is not empty. If the input is empty the behavior can be set via SetSystem(::’noObjectResult’,<Result>);. If necessary, an exception is raised.

Parallelization Information
EllipticAxisXld is reentrant and automatically parallelized (on tuple level).
Approximation of XLD contours by circles.

`FitCircleContourXld` approximates the XLD contours `Contours` by circles. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one circle. The operator returns for each contour the center (`Row`, `Column`), and the `Radius`.

The algorithm used for the fitting of circles can be selected via `Algorithm`:

- 'algebraic'  This approach minimizes the algebraic distance between the contour points and the resulting circle.
- 'ahuber'     Similar to 'algebraic'. Here the contour points are weighted to decrease the impact of outliers based on the approach of Huber.
- 'atukey'     Similar to 'algebraic'. Here the contour points are weighted to decrease the impact of outliers based on the approach of Tukey.

For 'ahuber' and 'atukey' a robust error statistics is used to estimate the standard deviation of the distances from the contour points without outliers from the approximating circle. The parameter `ClippingFactor` (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for `ClippingFactor` the more outliers are detected. The detection of outliers and the least squares fitting is repeated. The parameter `Iterations` specifies the number of iterations.

To reduce the computational load, the fitting of circles can be restricted to a subset of the contour points: If a value other than -1 is assigned to `MaxNumPoints`, only up to `MaxNumPoints` points - uniformly distributed over the contour - are used.

For circular arcs, the points on the circle closest to the start points and end points of the original contours are chosen as start and end points. The corresponding angles referring to the X-axis are returned in `StartPhi` and `EndPhi`, see also `GenEllipseContourXld`. Contours, for which the distance between their start points and their end points is less or equal `MaxClosureDist` are considered to be closed. Thus, they are approximated by circles instead of circular arcs. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude `ClippingEndPoints` points at the beginning and at the end of a contour from the fitting of circles. However, they are still used for the determination of `StartPhi` and `EndPhi`. 

```plaintext
[out] VARIANT Row HXLDContX.FitCircleContourXld
([in] String Algorithm, [in] long MaxNumPoints, [in] double MaxClosureDist,
[in] long ClippingEndPoints, [in] long Iterations,
[in] double ClippingFactor, [out] VARIANT Column, [out] VARIANT Radius,
[out] VARIANT StartPhi, [out] VARIANT EndPhi, [out] VARIANT PointOrder )
```

```
void HOperatorSetX.FitCircleContourXld ([in] IObjectX Contours,
[in] VARIANT Algorithm, [in] VARIANT MaxNumPoints,
[in] VARIANT MaxClosureDist, [in] VARIANT ClippingEndPoints,
[in] VARIANT Iterations, [in] VARIANT ClippingFactor, [out] VARIANT Row,
[out] VARIANT Column, [out] VARIANT Radius, [out] VARIANT StartPhi,
[out] VARIANT EndPhi, [out] VARIANT PointOrder )
```
15.3. FEATURES

Parameter

Contours (input iconic) ... xld_cont(-array)  \(\sim\) HXLDContX / IOBJECTX
Input contours.

Algorithm (input control) ... string  \(\sim\) String / VARIANT
Algorithm for the fitting of circles.
Default Value: 'algebraic'
List of values: \{ 'algebraic', 'ahuber', 'atukey' \}

MaxNumPoints (input control) ... integer  \(\sim\) long / VARIANT
Maximum number of contour points used for the computation (-1 for all points).
Default Value: -1
Restriction: \(\text{MaxNumPoints} \geq 3\)

MaxClosureDist (input control) ... real  \(\sim\) double / VARIANT
Maximum distance between the end points of a contour to be considered as 'closed'.
Default Value: 0.0
Restriction: \(\text{MaxClosureDist} \geq 0.0\)

ClippingEndPoints (input control) ... integer  \(\sim\) long / VARIANT
Number of points at the beginning and at the end of the contours to be ignored for the fitting.
Default Value: 0
Restriction: \(\text{ClippingEndPoints} \geq 0\)

Iterations (input control) ... integer  \(\sim\) long / VARIANT
Maximum number of iterations.
Default Value: 3
Restriction: \(\text{Iterations} \geq 0\)

ClippingFactor (input control) ... real  \(\sim\) double / VARIANT
Clipping factor for the elimination of outliers (typical: 1.0 for Huber and 2.0 for Tukey).
Default Value: 2.0
List of values: \{ 1.0, 1.5, 2.0, 2.5, 3.0 \}
Restriction: \(\text{ClippingFactor} > 0\)

Row (output control) ... circle.center.y(-array)  \(\sim\) VARIANT ( real )
Row coordinate of the center of the circle.

Column (output control) ... circle.center.x(-array)  \(\sim\) VARIANT ( real )
Column coordinate of the center of the circle.

Radius (output control) ... circle.radius(-array)  \(\sim\) VARIANT ( real )
Radius of circle.

StartPhi (output control) ... real(-array)  \(\sim\) VARIANT ( real )
Angle of the start point [rad].

EndPhi (output control) ... real(-array)  \(\sim\) VARIANT ( real )
Angle of the end point [rad].

PointOrder (output control) ... string(-array)  \(\sim\) VARIANT ( string )
Point order along the boundary.
List of values: \{ 'positive', 'negative' \}

FitCircleContourXld returns TRUE if all parameter values are correct, and circles could be fitted to the input contours. If the input is empty the behaviour can be set via SetSystem('noObjectResult',<Result>).
If necessary, an exception is raised. If the parameter ClippingFactor is chosen too small, i.e., all points are classified as outliers, the error 3264 is raised.

Parallelization Information

FitCircleContourXld is local and processed completely exclusively without parallelization.

Possible Predecessors

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, SmoothContoursXld

Possible Successors

GenEllipseContourXld, DispCircle, GetPointsEllipse

See also

FitEllipseContourXld, FitLineContourXld

Module

Sub-pixel operators

HALCON 6.1.4
Approximation of XLD contours by ellipses or elliptic arcs.

FitEllipseContourXld approximates the XLD contours Contours by elliptic arcs or closed ellipses. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one elliptic structure. The operator returns for each contour the center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2 of the underlying ellipse. In addition to that, the angle corresponding to the start point and the end points - uniformly distributed Iterations, only up to FitEllipseContourXld by elliptic arcs or closed ellipses. It standard circular segments are used for this computation. To speed up the ClippingFactor, the Phi, and the length of the smaller half, and the point order along the boundary PointOrder, see also. Contours, for which the distance between their :.

For elliptic arcs, the points on the ellipse closest to the start points and end points of the original contours are chosen as start and end points. The corresponding angles referring to the main axis of the ellipse are returned in StartPhi, EndPhi, and the point order along the boundary PointOrder is returned for elliptic arcs. These parameters are set to 0, 2\(\pi\), and 'positive' for closed ellipses. The algorithm used for the fitting of ellipses can be selected via Algorithm:

'fitzgibbon’ This approach approximates the XLD contours Contours by elliptic arcs or closed ellipses. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one elliptic structure. The operator returns for each contour the center (Row, Column), the orientation of the main axis Phi, the length of the larger half axis Radius1, and the length of the smaller half axis Radius2 of the underlying ellipse. In addition to that, the angle corresponding to the start point and the end point StartPhi, EndPhi, and the point order along the boundary PointOrder is returned for elliptic arcs. These parameters are set to 0, 2\(\pi\), and 'positive' for closed ellipses. The algorithm used for the fitting of ellipses can be selected via Algorithm:

'fitzgibbon’ This approach minimizes the algebraic distance \(ax_i^2 + bx_iy_i + cy_i^2 + dx_i + ey_i + f\) between the contour points \((x_i, y_i)\) and the resulting ellipse. The constraint \(4ac - b^2 = 1\) guarantees that the resulting polygon characterizes an ellipse (instead of a hyperbola or a parabola).

'fhuber’ Similar to 'fitzgibbon’. Here the contour points are weighted to decrease the impact of outliers based on the approach of Huber.

'ftukey’ Similar to 'fitzgibbon’. Here the contour points are weighted to decrease the impact of outliers based on the approach of Tukey.

'voss' Each input contour is transformed in an affine standard position. Based on the moments of the transformed contour (that is of the enclosed image region) the standard circular segment is choosen whose standard position matches best with the standard position of the contour. The ellipse corresponding to the standard position of the selected circular segment is re-transformed based on the affine transformation which produced the standard position of the contour resulting in the ellipse matching the original contour. VossTabSize standard circular segments are used for this computation. To speed up the process the corresponding moments and other data is stored in a table which is created during the first call (with a specific value for VossTabSize) to FitEllipseContourXld.

'focpoints’ Each point \(P\) on an ellipse satisfies the constraint that the sum of distances to the focal points \(F_1, F_2\) equals twice the length of the larger half axis \(a\). In this approach, the deviation \(PF_1 + PF_2 - 2a\) is minimized for all contour points by a least squares optimization.

'fphuber’ Similar to 'focpoints’. Here a weighted least squares optimization is done to decrease the impact of outliers based on the approach of Huber.

'fptukey’ Similar to 'focpoints’. Here a weighted least squares optimization is done to decrease the impact of outliers based on the approach of Tukey.

For '*Huber’ and ‘*Tukey’ a robust error statistics is used to estimate the standard deviation of the distances from the contour points without outliers from the approximating ellipse. The parameter ClippingFactor (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for ClippingFactor the more outliers are detected. The detection of outliers and the least squares fitting in the mode 'focpoints’ is repeated. The parameter Iterations specifies the number of iterations.

To reduce the computational load, the fitting of ellipses can be restricted to a subset of the contour points: If a value other than -1 is assigned to MaxNumPoints, only up to MaxNumPoints points - uniformly distributed over the contour - are used.

For elliptic arcs, the points on the ellipse closest to the start points and end points of the original contours are chosen as start and end points. The corresponding angles referring to the main axis of the ellipse are returned in StartPhi and EndPhi, see also GenEllipseContourXld. Contours, for which the distance between their
start points and their end points is less or equal \( \text{MaxClosureDist} \) are considered to be closed. Thus, they are approximated by ellipses instead of elliptic arcs. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude ClippingEndPoints points at the beginning and at the end of a contour from the fitting of ellipses. However, they are still used for the determination of StartPhi and EndPhi.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Contours</strong></td>
<td>xld_cont(-array)</td>
<td>Input contours.</td>
</tr>
<tr>
<td><strong>Algorithm</strong></td>
<td>string</td>
<td>Algorithm for the fitting of ellipses.</td>
</tr>
<tr>
<td><strong>MaxNumPoints</strong></td>
<td>integer</td>
<td>Maximum number of contour points used for the computation (-1 for all points).</td>
</tr>
<tr>
<td><strong>MaxClosureDist</strong></td>
<td>real</td>
<td>Maximum distance between the end points of a contour to be considered as 'closed'.</td>
</tr>
<tr>
<td><strong>ClippingEndPoints</strong></td>
<td>integer</td>
<td>Number of points at the beginning and at the end of the contours to be ignored for the fitting.</td>
</tr>
<tr>
<td><strong>VossTabSize</strong></td>
<td>integer</td>
<td>Number of circular segments used for the Voss approach.</td>
</tr>
<tr>
<td><strong>Iterations</strong></td>
<td>integer</td>
<td>Maximum number of iterations (unused for 'fitzgibbon' and 'voss').</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>List of values</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm</strong></td>
<td>{'fitzgibbon', 'fhuber', 'ftukey', 'voss', 'focpoints', 'fphuber', 'fptukey'}</td>
<td></td>
</tr>
<tr>
<td><strong>MaxNumPoints</strong></td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td><strong>MaxClosureDist</strong></td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td><strong>ClippingEndPoints</strong></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>VossTabSize</strong></td>
<td>200</td>
<td></td>
</tr>
<tr>
<td><strong>Iterations</strong></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td><strong>ClippingFactor</strong></td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td><strong>PointOrder</strong></td>
<td>{'positive', 'negative'}</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

HALCON 6.1.4
read_image (Image, 'caltab')
find_caltab (Image, Caltab, 'caltabBig.descr', 3, 112, 5)
reduce_domain (Image, Caltab, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
select_contours_xld (Edges, EdgesClosed, 'closed', 0, 2.0, 0, 0)
select_contours_xld (EdgesClosed, EdgesMarks, 'length', 20, 80, 0, 0)
fit_ellipse_contour_xld (EdgesMarks, 'fitzgibbon', -1, 2, 0, 200, 3, 2.0, Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder)
gen_ellipse_contour_xld (EllMarks, Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder, 1.5)
length_xld(EllMarks,Length).

Result

FitEllipseContourXld returns TRUE if all parameter values are correct, and ellipses could be fitted to the input contours. If the input is empty the behaviour can be set via SetSystem ('noObjectResult',<Result>). If necessary, an exception is raised. If the parameter ClippingFactor is chosen too small, i.e., all points are classified as outliers, the error 3264 is raised.

Parallelization Information

FitEllipseContourXld is local and processed completely exclusively without parallelization.

Possible Predecessors

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, SmoothContoursXld

Possible Successors

GenEllipseContourXld, DispEllipse, GetPointsEllipse

See also

FitLineContourXld

Module

Sub-pixel operators

| [out] VARIANT RowBegin | HXLDContX.FitLineContourXld |


Approximation of XLD contours by line segments.

FitLineContourXld approximates the XLD contours Contours by line segments. It does not perform a segmentation of the input contours. Thus, one has to make sure that each contour corresponds to one and only one line segment. The operator returns for each contour the start point (RowBegin, ColBegin), the end point (RowEnd, ColEnd), and the regression line to the contour given by the normal vector (Nr, Nc) of the line and its distance Dist from the origin, i.e., the line equation is given by \( r \cdot Nr + c \cdot Nc - Dist = 0 \).

The algorithm used for the fitting of lines can be selected via Algorithm:

- 'regression' Standard 'least squares' line fitting.
- 'huber' Weighted 'least squares' line fitting, where the impact of outliers is decreased based on the approach of Huber.
- 'tukey' Weighted 'least squares' line fitting, where the impact of outliers is decreased based on the approach of Tukey.
- 'drop' Weighted 'least squares' line fitting, where outliers are eliminated.
15.3. FEATURES

'gauss' Weighted 'least squares' line fitting, where the impact of outliers is decreased based on the mean value and the standard deviation of the distances of all contour points from the approximating line.

For 'huber', 'tukey', and 'drop' a robust error statistics is used to estimate the standard deviation of the distances from the contour points without outliers from the approximating line. The parameter ClippingFactor (a scaling factor for the standard deviation) controls the amount of damping outliers: The smaller the value chosen for ClippingFactor the more outliers are detected. The detection of outliers is repeated. The parameter Iterations specifies the number of iterations. In the modus 'regression' this value is ignored.

To reduce the computational load, the fitting of lines can be restricted to a subset of the contour points: If a value other than -1 is assigned to MaxNumPoints, only up to MaxNumPoints points - uniformly distributed over the contour - are used.

The start point and the end point of a line segment is determined by projecting the first and the last point of the corresponding contour to the approximating line. Due to artefacts in the pre-processing the start and end points of a contour might be faulty. Therefore, it is possible to exclude ClippingEndPoints points at the beginning and at the end of a contour from the line fitting. However, they are still used for the determination of the start point and the end point of the line segment.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contours</td>
<td>xld_cont(-array)</td>
<td>Input contours.</td>
</tr>
<tr>
<td>Algorithm</td>
<td>string</td>
<td>Algorithm for the fitting of lines.</td>
</tr>
<tr>
<td></td>
<td>String / VARIANT</td>
<td>Default Value: 'tukey'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>List of values: Algorithm ∈ {'regression', 'huber', 'tukey', 'gauss', 'drop'}</td>
</tr>
<tr>
<td>MaxNumPoints</td>
<td>integer</td>
<td>Maximum number of contour points used for the computation (-1 for all points).</td>
</tr>
<tr>
<td></td>
<td>long / VARIANT</td>
<td>Default Value: -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restriction: (MaxNumPoints ≥ 3)</td>
</tr>
<tr>
<td>ClippingEndPoints</td>
<td>integer</td>
<td>Number of points at the beginning and at the end of the contours to be ignored for the fitting.</td>
</tr>
<tr>
<td></td>
<td>long / VARIANT</td>
<td>Default Value: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restriction: (ClippingEndPoints ≥ 3)</td>
</tr>
<tr>
<td>Iterations</td>
<td>integer</td>
<td>Maximum number of iterations (unused for 'regression').</td>
</tr>
<tr>
<td></td>
<td>long / VARIANT</td>
<td>Default Value: 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restriction: (Iterations ≥ 0)</td>
</tr>
<tr>
<td>ClippingFactor</td>
<td>real</td>
<td>Clipping factor for the elimination of outliers (typical: 1.0 for 'huber' and 'drop' and 2.0 for 'tukey').</td>
</tr>
<tr>
<td></td>
<td>double / VARIANT</td>
<td>Default Value: 2.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>List of values: ClippingFactor ∈ {1.0, 1.5, 2.0, 2.5, 3.0}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Restriction: (ClippingFactor &gt; 0)</td>
</tr>
<tr>
<td>RowBegin</td>
<td>line.begin.y(-array)</td>
<td>Row coordinates of the starting points of the line segments.</td>
</tr>
<tr>
<td>ColBegin</td>
<td>line.begin.x(-array)</td>
<td>Column coordinates of the starting points of the line segments.</td>
</tr>
<tr>
<td>RowEnd</td>
<td>line.end.y(-array)</td>
<td>Row coordinates of the end points of the line segments.</td>
</tr>
<tr>
<td>ColEnd</td>
<td>line.end.x(-array)</td>
<td>Column coordinates of the end points of the line segments.</td>
</tr>
<tr>
<td>Nr</td>
<td>number(-array)</td>
<td>Line parameter: Row coordinate of the normal vector</td>
</tr>
<tr>
<td>Nc</td>
<td>number(-array)</td>
<td>Line parameter: Column coordinate of the normal vector</td>
</tr>
<tr>
<td>Dist</td>
<td>number(-array)</td>
<td>Line parameter: Distance of the line from the origin</td>
</tr>
</tbody>
</table>

Example

HALCON 6.1.4
read_image (Image, 'mreut')
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 20, 40)
gen_polygons_xld (Edges, Polygons, 'ramer', 2)
split_contours_xld (Polygons, Contours, 'polygon', 1, 5)
fit_line_contour_xld (Contours, 'regression', -1, 0, 5, 2, RowBegin, ColBegin, RowEnd, ColEnd, Nr, Nc).

RESULT

FitLineContourXld returns TRUE if all parameter values are correct, and line segments could be fitted to the input contours. If the input is empty the behaviour can be set via SetSystem ('noObjectResult',<Result>). If necessary, an exception is raised. If the parameter ClippingFactor is chosen too small, i.e., all points are classified as outliers, the error 3264 is raised.

Parallelization Information

FitLineContourXld is reentrant and processed without parallelization.

Possible Predecessors

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, SmoothContoursXld

Possible Successors

DispLine, SelectLines, LineOrientation

See also

RegressContoursXld, GetRegressParamsXld

Module

Sub-pixel operators

[out] VARIANT Angles HXLDContX.GetContourAngleXld
void HOperatorSetX.GetContourAngleXld ([in] IObjectX Contour,
[out] VARIANT Angles )

Calculate the direction of an XLD contour for each contour point.

GetContourAngleXld calculates for each point of the XLD contour Contour the direction of its tangent. Two modes for the output values can be chosen: By passing 'abs' for AngleMode, the resulting angles are returned relative to the horizontal axis as angles between 0 and $2\pi$ (counter-clockwise). By passing 'rel', the absolute value of the difference of angles to the previous contour point is returned. In this case the range of values is between 0 and $\pi$.

There are three different ways of calculating the tangent direction (CalcMode) of the contour point $i$ using the contour points in the interval from $i - \text{Lookaround}$ to $i + \text{Lookaround}$. By using 'range', the angle of the line segment between the first and last point of the interval is used. For 'mean', the average of all angles between consecutive points of the contour is used. Finally, for 'regress', the direction of the regression line (the least squares fit of a line to the contour points in the interval) is used. Lookaround is a measure of how strongly the contour is smoothed. The angles are returned in Angles in radians.

Parameter

- **Contour** (input iconic) ... xld_cont $\rightarrow$ HXLDContX / IObjectX
  - Input contour.
- **AngleMode** (input control) ... string $\rightarrow$ String / VARIANT
  - Return type of the angles.
    - Default Value : 'abs'
    - List of values : AngleMode $\in$ {'abs', 'rel'}
- **CalcMode** (input control) ... string $\rightarrow$ String / VARIANT
  - Method for computing the angles.
    - Default Value : 'range'
    - List of values : CalcMode $\in$ {'range', 'mean', 'regress'}
15.3. FEATURES

- **Lookaround** (input control) .......................... integer \(\rightarrow\) long / VARIANT
  
  Number of points to take into account.

  **Default Value:** 3
  
  **Restriction:** \((\text{Lookaround} > 0)\)

- **Angles** (output control) .......................... real \(\rightarrow\) VARIANT (real)
  
  Direction of the tangent to the contour points.

---

GetContourAngleXld is reentrant and processed without parallelization.

---

Possible Predecessors

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

---

See also

GetContourXld, GetContourAttribXld

---

Module

Sub-pixel operators

[out] VARIANT Attrib HXLDContX.GetContourAttribXld ([in] String Name )

void HOperatorSetX.GetContourAttribXld ([in] IHObjectX Contour, [in] VARIANT Name, [out] VARIANT Attrib )

Return point attribute values of an XLD contour.

GetContourAttribXld returns the values of the attribute \(\text{Name}\) of the XLD contour \(\text{Contour}\) in \(\text{Attrib}\). Attributes are additional values defined for each contour point, e.g., the direction perpendicular to the contour (‘angle’). Operators which define this kind of attributes, e.g., LinesGauss and EdgesSubPix, contain a description of the name and semantics of the defined values. QueryContourAttribsXld can be used to query which attributes are defined for a particular contour.

---

Parameter

- **Contour** (input iconic) .......................... xld_cont \(\rightarrow\) HXLDContX / IHObjectX
  
  Input XLD contour.

- **Name** (input control) .......................... string \(\rightarrow\) String / VARIANT
  
  Name of the attribute.

  **Default Value:** ‘angle’

  **Suggested values:** \(\text{Name} \in \{ \text{‘angle’}, \text{‘edge\_direction’}, \text{‘width\_right’}, \text{‘width\_left’}, \text{‘response’}, \text{‘contrast’}, \text{‘asymmetry’} \}\)

- **Attrib** (output control) .......................... real \(\rightarrow\) VARIANT (real)
  
  Attribute values.

---

Parallelization Information

GetContourAttribXld is reentrant and processed without parallelization.

---

Possible Predecessors

LinesGauss, LinesFacet, EdgesSubPix

---

See also

QueryContourAttribsXld, GetContourGlobalAttribXld

---

Module

Sub-pixel operators

[out] VARIANT Attrib HXLDContX.GetContourGlobalAttribXld ([in] VARIANT Name )


Return global attributes values of an XLD contour.
GetContourGlobalAttribXld returns the values of the global attribute Name of the XLD contour Contour in Attrib. Global attributes are additional values defined for each contour, e.g., the normal vector of the regression line of a contour (‘regr_norm_row’ and ‘regr_norm_col’). Operators which define this kind of attributes contain a description of the name and semantics of the defined values. QueryContourGlobalAttribsXld can be used to query which attributes are defined for a particular contour.

Parameter

- **Contour** (input iconic) xld_cont ~ HXLDContX / IObjectX
  - Input XLD contour.
- **Name** (input control) string(-array) ~ VARIANT ( string )
  - Name of the attribute.
  - **Default Value:** ‘regr_norm_row’
  - **Suggested values:** Name ∈ {‘regr_norm_row’, ‘regr_norm_col’, ‘regr_mean_dist’, ‘regr_dev_dist’, ‘cont_approx’}
- **Attrib** (output control) real ~ VARIANT ( real )
  - Attribute values.

Parallelization Information

GetContourGlobalAttribXld is reentrant and processed without parallelization.

Possible Predecessors

LinesGauss, LinesFacet, EdgesSubPix

See also

QueryContourGlobalAttribsXld, GetContourAttribXld, QueryContourAttribsXld

Module

Sub-pixel operators

| out | VARIANT Length HXLDContX.GetRegressParamsXld ( [out] VARIANT Nx, |
| out | VARIANT Ny, [out] VARIANT Dist, [out] VARIANT Fpx, [out] VARIANT Fpy, |
| out | VARIANT Lpx, [out] VARIANT Lpy, [out] VARIANT Mean, |
| out | VARIANT Deviation ) |
| void HOperatorSetX.GetRegressParamsXld ( [in] IObjectX Contours, |
| [out] VARIANT Length, [out] VARIANT Nx, [out] VARIANT Ny, [out] VARIANT Dist, |
| [out] VARIANT Fpx, [out] VARIANT Fpy, [out] VARIANT Lpx, [out] VARIANT Lpy, |
| [out] VARIANT Mean, [out] VARIANT Deviation ) |

Return XLD contour parameters.

GetRegressParamsXld returns the following parameters for all XLD contours given in Contours:

- the number of contour points Length,
- the coordinates Nx and Ny of the normal vector of the regression line (i.e., least-squares approximating line),
- the distance Dist of the regression line to the origin
- the sub-pixel precise coordinates Fpx and Fpy of the perpendicular projection of the start point of the contour onto the regression line,
- the sub-pixel precise coordinates Lpx and Lpy of the perpendicular projection of the end point of the contour onto the regression line,
- the mean of the Euclidian distance of the contour points from the regression line,
- the standard deviation of these distances.

Attention

Before the contour parameters can be returned by GetRegressParamsXld, the parameters of the regression line to the contour must be calculated by calling RegressContoursXld.
Return information about the gray values of the area enclosed by XLD parallels.

`InfoParallelsXld` calculates various gray value features of the area enclosed by the XLD parallels. The input image `Image` is used to get the gray values needed for this. The algorithm used in this operator is very similar to the one used in `ModParallelsXld`. The operator returns ranges for the quality factor (`QualityMin` and `QualityMax`), the mean gray value (`GrayMin` and `GrayMax`), and the standard deviation with respect to the mean gray value (`StandardMin` and `StandardMax`).

This operator serves to determine appropriate thresholds for `ModParallelsXld`.
CHAPTER 15. XLD

Parameter

- **Parallels** (input iconic) ............................. \( xld\text{\_}para \sim HDMI/\text{IHObjectX} \\
  \text{Input XLD Parallels.}

- **Image** (input iconic) ............................. \( \text{image} \sim HDMI/\text{IHObjectX (byte)} \)
  \text{Corresponding gray value image.}

- **QualityMin** (output control) .................................. \( \text{real} \sim \text{double / VARIANT} \)
  Minimum quality factor.

- **QualityMax** (output control) ................................... \( \text{real} \sim \text{double / VARIANT} \)
  Maximum quality factor.

- **GrayMin** (output control) .................................. \( \text{integer} \sim \text{long / VARIANT} \)
  Minimum mean gray value.

- **GrayMax** (output control) .................................. \( \text{integer} \sim \text{long / VARIANT} \)
  Maximum mean gray value.

- **StandardMin** (output control) ............................. \( \text{real} \sim \text{double / VARIANT} \)
  Minimum standard deviation.

- **StandardMax** (output control) ............................. \( \text{real} \sim \text{double / VARIANT} \)
  Maximum standard deviation.

Parallelization Information

- **InfoParallelsXld** is *reentrant* and processed *without* parallelization.

Possible Predecessors

- **GenParallelsXld**

Possible Successors

- **ModParallelsXld**

See also

- **Intensity, MinMaxGray**

Sub-pixel operators

```plaintext
[out] \text{VARIANT Length} \text{HXLDX}\_\text{LengthXld} ( )

void \text{HOperatorSetX}\_\text{LengthXld} ([\text{in}] \text{IHObjectX XLD},
[out] \text{VARIANT Length} )
```

Length of contours or polygons.

**LengthXld** calculates the length of the contours or polygons XLD. The length is calculated as the sum of the euclidian distances of successive points on the contour or polygon. If more than one contour or polygon is passed the results are stored as tuples in which the index of a value corresponds to the index of the respective contour or polygon in XLD.

Parameter

- **XLD** (input iconic) ............................. \( xld\text{-array} \sim HDMI/\text{IHObjectX} \\
  \text{Contours or polygons to be examined.}

- **Length** (output control) ......................... \( \text{real\text{-array} \sim \text{VARIANT (real)}} \)
  Length of the contour or polygon.

Restriction: \((\text{Length} \geq 0)\)

Complexity

Let \( n \) be the number of points of the contour or polygon. Then the run time is \( O(n) \).

Result

**LengthXld** returns TRUE if the input is not empty. If the input is empty the behaviour can be set via \text{SetSystem(\text{::<noObjectResult>,<Result>})}. If necessary, an exception is raised.

Parallelization Information

**LengthXld** is *reentrant* and automatically *parallelized* (on tuple level).

Possible Predecessors

- **GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld**
Select XLD contours with a local maximum of gray values.

LocalMaxContoursXld selects XLD contours from the contours passed in Contours, which have a local maximum in gray values across the direction of the contour. In order to be selected, at least \( \text{MinPercent} \) of the contour points must have a local maximum perpendicular to the direction of the contour. The contours’ direction is determined by fitting a regression line through five neighboring points of the contour. In order to decide whether there is a local maximum for a contour point, a gray value profile that is \( \text{Distance} \) points wide and perpendicular to the contour is computed on both sides of the contour. If the gray value at the countour point is at least \( \text{MinDiff} \) larger than the gray value at at least one point on each side of the profile, the contour point is labeled as a local maximum. The selected contours are returned in LocalMaxContours.

**Parameter**

- **Contours** (input iconic) 
  - XLD contours to be examined.
- **Image** (input iconic) 
  - Corresponding gray value image.
- **LocalMaxContours** (output iconic) 
  - Selected contours.
- **MinPercent** (input control) 
  - Minimum percentage of maximum points.
  - Default Value: 70
  - Suggested values: \( \text{MinPercent} \in \{60, 70, 75, 80, 85, 90, 95\} \)
  - Restriction: \( (0.0 \leq \text{MinPercent}) \land (\text{MinPercent} \leq 100.0) \)
- **MinDiff** (input control) 
  - Minimum amount by which the gray value at the maximum must be larger than in the profile.
  - Default Value: 15
  - Suggested values: \( \text{MinDiff} \in \{5, 8, 10, 12, 15, 20\} \)
  - Restriction: \( (0 \leq \text{MinDiff}) \land (\text{MinDiff} \leq 255) \)
- **Distance** (input control) 
  - Maximum width of profile used to check for maxima.
  - Default Value: 4
  - Suggested values: \( \text{Distance} \in \{2, 3, 4, 5, 6\} \)
  - Restriction: \( (\text{Distance} \geq 1) \)

**Parallelization Information**

LocalMaxContoursXld is reentrant and processed without parallelization.
### CHAPTER 15. XLD

Join modified XLD parallels lying on the same polygon.

MaxParallelsXld joins all modified XLD parallels in ExtParallels into a polygon if they lie on the same original polygon segment. This means that polygons exhibiting parallelism and enclosing homogeneous areas in several places are joined into one long polygon (from the first parallel line to the last parallel line). The resulting polygons are returned in MaxPolygons.

```c
void HOperatorSetX.MaxParallelsXld ([in] IHObjectX ExtParallels, [out] HUntypedObjectX MaxPolygons )
```

Join modified XLD parallels lying on the same polygon.

MaxParallelsXld joins all modified XLD parallels in ExtParallels into a polygon if they lie on the same original polygon segment. This means that polygons exhibiting parallelism and enclosing homogeneous areas in several places are joined into one long polygon (from the first parallel line to the last parallel line). The resulting polygons are returned in MaxPolygons.

Parameter

- **ExtParallels** (input iconic) ................. xld_ext_para ჌ HXLDExtParaX / IHObjectX Extended XLD parallels.
- **MaxPolygons** (output iconic) ................. xld_poly ჌ HXLDPolyX / HUntypedObjectX Maximally extended parallels.

Parallelization Information

MaxParallelsXld is reentrant and processed without parallelization.

Possible Predecessors

- ModParallelsXld
- GetPolygonXld, GetLinesXld

Possible Successors

Module

- Sub-pixel operators

### Arbitrary geometric moments of contours or polygons.

MomentsAnyXld calculates arbitrary moments $M$ of the region enclosed by the contours or polygons XLD. The moments are computed by applying Green’s theorem using only the points on the contour or polygon, i.e., no region is generated explicitly for the purpose of calculating the features. It is assumed that the contours or polygons are closed. If this is not the case MomentsAnyXld will artificially close the contours or polygons. The computed moments are normalized depending on the desired mode Mode:

- 'unnormalized': No normalization. Let $R$ be the enclosed image region. Then the computed moment $M_{p,q}$ is equivalent to
  \[ M_{p,q} = \int_{R} r^p c^q \, dr \, dc \]

- 'unnormalized_central': Shift the region by its centroid $[\bar{r}, \bar{c}] = [CenterRow, CenterCol]$:
  \[ M_{p,q} = \int_{R} (r - \bar{r})^p (c - \bar{c})^q \, dr \, dc \]

- 'normalized': Normalization by the area $A = Area$ of the enclosed image region:
  \[ M_{p,q} = \frac{1}{A} \int_{R} r^p c^q \, dr \, dc \]
'central': Normalization by the area $A = \text{Area}$ of the enclosed image region and a shift of the region by its centroid $[\bar{r}, \bar{c}] = [\text{CenterRow}, \text{CenterCol}]$:

$$\begin{align*}
M_{p,q} &= \frac{1}{A} \int_{R} (r - \bar{r})^p (c - \bar{c})^q \, dr \, dc
\end{align*}$$

For the normalization of the moments three specific moments are used: The area $\text{Area}$ of the enclosed image region and the coordinates $\text{CenterRow}, \text{CenterCol}$ of its centroid, see $\text{AreaCenterXld}$. In addition to that $\text{MomentsAnyXld}$ expects information about the point order of the input contours/polylines in $\text{PointOrder}$, see $\text{AreaCenterXld}$ again. If more than one contour or polygon is passed, $M$ contains all desired moments of the first contour/polygon followed by all the moments of the second contour/polygon and so forth.

### Parameter

- **XLD** (input iconic) : xld(-array) $\sim$ IHLDX / IHObjectX
  - Contours or polygons to be examined.

- **Mode** (input control) : string $\sim$ String / VARIANT
  - Default Value: 'unnormalized'
  - Suggested values: Mode $\in \{ \text{"unnormalized"}, \text{"unnormalized central"}, \text{"normalized"}, \text{"central"} \}$

- **PointOrder** (input control) : string(-array) $\sim$ VARIANT ( string )
  - point order along the boundary.
  - Default Value: 'positive'
  - Suggested values: PointOrder $\in \{ \text{"positive"}, \text{"negative"} \}$

- **Area** (input control) : real(-array) $\sim$ VARIANT ( real )
  - Area enclosed by the contour or polygon.

- **CenterRow** (input control) : point.y(-array) $\sim$ VARIANT ( real )
  - Row coordinate of the centroid.

- **CenterCol** (input control) : point.x(-array) $\sim$ VARIANT ( real )
  - Column coordinate of the centroid.

- **P** (input control) : point.x(-array) $\sim$ VARIANT ( integer )
  - First index of the desired moments $M[P,Q]$.
  - Default Value: 1

- **Q** (input control) : point.x(-array) $\sim$ VARIANT ( integer )
  - Second index of the desired moments $M[P,Q]$.
  - Default Value: 1

- **M** (output control) : real(-array) $\sim$ VARIANT ( real )
  - The computed moments.

### Complexity

Let $n$ be the number of points of the contour or polygon. Then the run time is $O(n)$.

### Result

$\text{MomentsAnyXld}$ returns TRUE if the input is not empty. If the input is empty the behaviour can be set via $\text{SetSystem}(:'noObjectResult',<Result>):$. If necessary, an exception is raised.

### Parallelization Information

$\text{MomentsAnyXld}$ is local and processed completely exclusively without parallelization.

### Possible Predecessors

$\text{AreaCenterXld}, \text{GenContoursSkeletonXld}, \text{SmoothContoursXld}, \text{GenPolygonsXld}$

### See also

$\text{MomentsXld}, \text{AreaCenterXld}, \text{MomentsRegion2Nd}, \text{AreaCenter}$

### Alternatives

$\text{MomentsXld}$

### Module

Sub-pixel operators
CHAPTER 15. XLD

void HOperatorSetX.MomentsXld ([in] IHobjectX XLD, [out] VARIANT M11, [out] VARIANT M20, [out] VARIANT M02)

Geometric moments M20, M02, and M11 of contours or polygons.

MomentsXld calculates the moments (M20, M02, and M11) of the region enclosed by the contours or polygons XLD. See MomentsRegion2Nd for the definition of these features. The moments are computed by applying Green’s theorem using only the points on the contour or polygon, i.e., no region is generated explicitly for the purpose of calculating the features. It is assumed that the contours or polygons are closed. If this is not the case MomentsXld will artificially close the contours or polygons. If more than one contour or polygon is passed the results are stored as tuples in which the index of a value corresponds to the index of the respective contour or polygon in XLD.

Parameter

- **XLD** (input iconic) xld(-array) IHXLDX / IHObjectX
  Contours or polygons to be examined.
- **M11** (output control) real(-array) VARIANT(real)
  Mixed second order moment.
- **M20** (output control) real(-array) VARIANT(real)
  Second order moment along the row axis.
- **M02** (output control) real(-array) VARIANT(real)
  Second order moment along the column axis.

Complexity

Let \( n \) be the number of points of the contour or polygon. Then the run time is \( O(n) \).

Result

MomentsXld returns TRUE if the input is not empty. If the input is empty the behaviour can be set via SetSystem(::’noObjectResult’,<Result>:). If necessary, an exception is raised.

Parallelization Information

MomentsXld is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

GenContoursSkeletonXld, SmoothContoursXld, GenPolygonsXld

See also

MomentsAnyXld, AreaCenterXld, MomentsRegion2Nd, AreaCenter

Alternatives

MomentsAnyXld

Module

Sub-pixel operators

void HOperatorSetX.OrientationXld ([in] IHobjectX XLD, [out] VARIANT Phi)

Orientation of a contour.

The operator OrientationXld calculates the orientation of the input contour. The operator is based on EllipticAxisXld. In addition, the contour point with maximum distance to the center of gravity is calculated. If the column coordinate of this point is less than the column coordinate of the center of gravity the value of \( \pi \) is added to the angle. The input contour must not intersect itself, otherwise the resulting parameter is not meaningful. If the input contour is not closed it will be closed automatically.

If more than one contour is passed the results are stored in tuples, the index of a value in the tuple corresponding to the index of a contour in the input.
15.3. FEATURES

Parameter

- **XLD** (input iconic) .............................. xld(-array)  \( \sim \) IHXLDX / IHObjectX
  Contour(s) to be examined.
- **Phi** (output control) ............................. real(-array)  \( \sim \) VARIANT ( real )
  Orientation of the contour(s) (radians).
  **Restriction**: \(( ((-\pi)/2) < \Phi ) \land ( \Phi \leq ((3 \cdot \pi)/2)) \)

Result

The operator **OrientationXld** returns the value TRUE if the input is not empty. The behavior in case of empty input (no input contours available) is set via the operator **SetSystem('noObjectResult',<Result>)**. If necessary an exception is raised.

**Parallelization Information**

**OrientationXld** is reentrant and automatically parallelized (on tuple level).

**Possible Predecessors**

GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld

**Alternatives**

EllipticAxisXld, SmallestRectangle2Xld

**Module**

Sub-pixel operators

[out] VARIANT Attribs HXLDContX.QueryContourAttribsXld ( )

void HOperatorSetX.QueryContourAttribsXld ([in] IHObjectX Contour,
[out] VARIANT Attribs )

Return the names of the defined attributes of an XLD contour.

**QueryContourAttribsXld** returns the names of the defined attributes of an XLD contour **Contour** in **Attribs**. Attributes are additional values defined for each contour point, e.g., the direction perpendicular to the contour (‘angle’). Operators which define this kind of attributes contain a description of the name and semantics of the defined values. **GetContourAttribXld** can be used to access the values of a particular attribute.

Parameter

- **Contour** (input iconic) .............................. xldcont  \( \sim \) HXLDContX / IHObjectX
  Input contour.
- **Attribs** (output control) ............................. string  \( \sim \) VARIANT ( string )
  List of the defined contour attributes.

**Parallelization Information**

**QueryContourAttribsXld** is reentrant and processed without parallelization.

**Possible Predecessors**

LinesGauss, LinesFacet, EdgesSubPix

**See also**

GetContourAttribXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

**Module**

Sub-pixel operators

[out] VARIANT Attribs HXLDContX.QueryContourGlobalAttribsXld ( )

void HOperatorSetX.QueryContourGlobalAttribsXld ([in] IHObjectX Contour, [out] VARIANT Attribs )

Return the names of the defined global attributes of an XLD contour.

HALCON 6.1.4
QueryContourGlobalAttribsXld returns the names of the globally defined attributes of an XLD contour Contour in Attribs. Global attributes are additional values defined for each contour, e.g., the normal vector of the regression line of a contour (‘regr_norm_row’ and ‘regr_norm_col’). Operators which define this kind of attributes contain a description of the name and semantics of the defined values. GetContourGlobalAttribXld can be used to access the value of a particular attribute.

Parameter

- **Contour** (input iconic) ................................. xld_cont ː= HXLDContX / IHObjectX
  Input contour.
- **Attribs** (output control) ............................... string ː= VARIANT( string )
  List of the defined global contour attributes.

Parallelization Information

QueryContourGlobalAttribsXld is reentrant and processed without parallelization.

Possible Predecessors

LinesGauss, LinesFacet, EdgesSubPix

See also

GetContourGlobalAttribXld, GetContourAttribXld, QueryContourAttribsXld

Module

Sub-pixel operators

Select XLD contours according to several features.

SelectContoursXld selects XLD contours from the input contours Contours according to the following features (parameter Feature):

- **contour_length** all contours whose length is smaller than Min1 or larger than Max1 are not returned (Min2 and Max2 have no influence here).
- **maximum_extent** all contours whose maximum extent (as measured by their eight extremal points in row and column direction, according to Haralick und Shapiro: Computer and Robot Vision, Addison-Wesley 1992, chapter 3.2) is smaller than Min1 or larger than Max1 are not returned (Min2 and Max2 have no influence here).
- **direction** only contours for which the direction of the regression line is between Min1 and Max1 (in radians, counter-clockwise) are returned. Min1 and Max1 are mapped to the range of [0,2*PI]. (Min2 and Max2 have no influence here).
- **curvature** only contours for which the mean distance from the regression line lies between Min1 and Max1, and for which the standard deviation of the distances is between Min2 and Max2 are returned.
- **closed** only contours for which the distance between their start point and their end point is less or equal Max1 pixels are returned.
- **open** only contours for which the distance between their start point and their end point is greater than Max1 pixels are returned.

If Min1 = Max1 = 0 or Min2 = Max2 = 0 is used for the selection according to curvature, the respective feature has no influence on the selection.

Attention

Before contour can be filtered by SelectContoursXld according to ‘direction’ or ‘curvature’, the parameters of the regression lines to the contours must be calculated by calling RegressContoursXld.
### 15.3. FEATURES

**Parameter**

- **Contours** (input iconic) ................................. `xld_cont` ~ `HXLDContX / IHOObjectX`
  
  Input XLD contours.

- **SelectedContours** (output iconic) .......................... `xld_cont` ~ `HXLDContX / HUntypedObjectX`
  
  Output XLD contours.

- **Feature** (input control) ................................. `string` ~ `String / VARIANT`
  
  Feature to select contours with.

  **Default Value:** 'contour_length'

  **List of values:** `Feature ∈ { 'contour_length', 'maximum_extent', 'direction', 'curvature', 'closed', 'open' }`

- **Min1** (input control) ................................. `real` ~ `double / VARIANT`
  
  Lower threshold.

  **Default Value:** 0.5

- **Max1** (input control) ................................. `real` ~ `double / VARIANT`
  
  Upper threshold.

  **Default Value:** 200.0

- **Min2** (input control) ................................. `real` ~ `double / VARIANT`
  
  Lower threshold.

  **Default Value:** -0.5

- **Max2** (input control) ................................. `real` ~ `double / VARIANT`
  
  Upper threshold.

  **Default Value:** 0.5

**Parallelization Information**

**SelectContoursXld** is reentrant and processed without parallelization.

**Possible Predecessors**

- **RegressContoursXld**

**See also**


**References**


**Module**

Sub-pixel operators

---

```plaintext
[out] HUntypedObjectX SelectedXLD HXLDX.SelectShapeXld
([in] VARIANT Features, [in] String Operation, [in] VARIANT Min,
[in] VARIANT Max )

void HOperatorSetX.SelectShapeXld ([in] IHOObjectX XLD,
[out] HUntypedObjectX SelectedXLD, [in] VARIANT Features,
```

Select contours using shape features.

The operator **SelectShapeXld** chooses regions according to their shape. For each input region in XLD the indicated features (Features) are calculated. If each (Operation = 'and') or at least one (Operation = 'or') of the calculated features is within the limits (Min, Max) the contour is copied into the output.

**Condition:** \( \text{Min}, \leq \text{Features}(\text{Object}) \leq \text{Max} \),

Possible values for Features:

- **'area'**: Area of the contour
- **'row'**: Row index of the center of gravity
- **'column'**: Column index of the center of gravity
- **'width'**: Width of the contour
'height': Height of the contour
'row1': Row coordinate of upper left corner
'column1': Column coordinate of upper left corner
'row2': Row coordinate of lower right corner
'column2': Column coordinate of lower right corner
'circularity': Circularity (see CircularXld)
'compactness': Compactness (see CompactnessXld)
'contlength': Total length of contour (see LengthXld)
'convexity': Convexity (see ConvexityXld)
'ra': Major radius of the equivalent ellipse (see EllipticAxisXld)
'rb': Minor radius of the equivalent ellipse (see EllipticAxisXld)
'phi': Orientation of the equivalent ellipse (see EllipticAxisXld)
'anisometry': Anisometry (see EccentricityXld)
'bulkiness': Bulkiness (see EccentricityXld)
'struct_factor': Structure factor (see EccentricityXld)
'outer_radius': Radius of smallest enclosing circle (see SmallestCircleXld)
'max_diameter': Maximum diameter of the contour (see DiameterXld)
'orientation': Orientation of the contour (see OrientationXld)
'rect2_phi': Orientation of the smallest surrounding rectangle (see SmallestRectangle2Xld)
'rect2_len1': Half the length of the smallest surrounding rectangle (see SmallestRectangle2Xld)
'rect2_len2': Half the width of the smallest surrounding rectangle (see SmallestRectangle2Xld)
'moments_m11', 'moments_m20', 'moments_m02': Geometric moments of the contour (see MomentsRegion2Nd)

If only one feature (Features) is used the value of Operation is meaningless. Several features are processed in the sequence in which they are entered. The use of some features requires that the input contour must not intersect itself, otherwise the results are not meaningful.

Attention

Parameter

- **XLD** (input iconic) ................................. xld ~ IHXLDX / IHObjectX
  Contours to be examined.
- **SelectedXLD** (output iconic) .................. xld ~ HUntypedObjectX
  Contours fulfilling the condition(s).
- **Features** (input control) ......................... string(-array) ~ VARIANT ( string )
  Shape features to be checked.
  Default Value: 'area'
  List of values: Features ∈ { 'area', 'row', 'column', 'width', 'height', 'row1', 'column1', 'row2',
  'column2', 'circularity', 'compactness', 'contlength', 'convexity', 'ra', 'rb', 'phi', 'anisometry', 'bulkiness',
  'struct_factor', 'outer_radius', 'max_diameter', 'orientation', 'rect2_phi', 'rect2_len1', 'rect2_len2',
  'moments_m11', 'moments_m20', 'moments_m02' }
- **Operation** (input control) ...................... string ~ String / VARIANT
  Operation type between the individual features.
  Default Value: 'and'
  List of values: Operation ∈ { 'and', 'or' }
- **Min** (input control) .............................. real(-array) ~ VARIANT ( integer, real, string )
  Lower limits of the features or 'min'.
  Default Value: 150.0
  Typical range of values: 0.0 ≤ Min ≤ 0.0
  Minimum Increment: 0.001
  Recommended Increment: 1.0
Max (input control) .................................................. real(-array) ～ VARIANT ( integer, real, string )
Upper limits of the features or ’max’.
Default Value : 99999.0
Typical range of values : $0.0 \leq \text{Max} \leq 0.0$
Minimum Increment : 0.001
Recommended Increment : 1.0
Restriction : $(\text{Max} \geq \text{Min})$

Result

The operator SelectShapeXld returns the value TRUE if the input is not empty. The behavior in case of empty input (no input objects available) is set via the operator SetSystem(’noObjectResult’,<Result>). If necessary an exception is raised.

Parallelization Information
SelectShapeXld is reentrant and processed without parallelization.

Possible Predecessors
GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld

Possible Successors
ShapeTransXld, CountObj

See also
AreaCenterXld, CircularityXld, CompactnessXld, ConvexityXld, EllipticAxisXld, EccentricityXld, SmallestCircleXld, SmallestRectangle1Xld, SmallestRectangle2Xld, DiameterXld, OrientationXld, MomentsXld, SelectObj

Module

Sub-pixel operators

[out] VARIANT Row HXLDX.SmallestCircleXld ([out] VARIANT Column,
[out] VARIANT Radius )

void HOperatorSetX.SmallestCircleXld ([in] IHObjectX XLD,
[out] VARIANT Row, [out] VARIANT Column, [out] VARIANT Radius )

Smallest enclosing circle of a contour.
The operator SmallestCircleXld determines the smallest enclosing circle of a contour, i.e., the circle with the smallest area of all circles containing the contour. For this circle the center $(\text{Row,Column})$ and the radius $(\text{Radius})$ are calculated.

If several contours are passed in XLD corresponding tuples are returned as output parameter. In case of empty contours all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Parameter

- **XLD** (input iconic) .............................................. xld(-array) ～ IHXLDX / IHObjectX
  Contour(s) to be examined.
- **Row** (output control) ........................................... circle.center.y(-array) ～ VARIANT ( real )
  Row coordinate of the center of the enclosing circle.
- **Column** (output control) ....................................... circle.center.x(-array) ～ VARIANT ( real )
  Column coordinate of the center of the enclosing circle.
- **Radius** (output control) ........................................ circle.radius(-array) ～ VARIANT ( real )
  Radius of the enclosing circle.
  Restriction : $(\text{Radius} \geq 0)$

Complexity
If $N$ is the number of contour points then the runtime complexity is $O(N \times \ln(N))$.

Result

The operator SmallestCircleXld returns the value TRUE if the input is not empty. If the input is empty the behavior can be set via SetSystem(;; ’noObjectResult’,<Result>). If necessary, an exception is raised.

Parallelization Information
SmallestCircleXld is reentrant and automatically parallelized (on tuple level).
Enclosing rectangle parallel to the coordinate axes.

The operator SmallestRectangle1Xld calculates the enclosing rectangle for each input contour (parallel to the coordinate axes). The enclosing rectangle is described by the coordinates of the corner pixels (Row1, Column1, Row2, Column2).

If more than one contour is passed in XLD, the results are stored in tuples, the index of a value in the tuple corresponding to the index of a contour in the input. In case of an empty contour all parameters have the value 0 if no other behavior was set (see SetSystem).

Attention
In case of empty contours the result of Row1, Column1, Row2 and Column2 (all are 0) can lead to confusion.

Parameter

- **XLD** (input iconic) ................. xld(-array) → HXLDX / IHOBJECTX Contour(s) to be examined.
- **Row1** (output control) .............. rectangle.origin.y(-array) → VARIANT (real) Row coordinate of upper left corner point of the enclosing rectangle.
- **Column1** (output control) ......... rectangle.origin.x(-array) → VARIANT (real) Column coordinate of upper left corner point of the enclosing rectangle.
- **Row2** (output control) .............. rectangle.corner.y(-array) → VARIANT (real) Row coordinate of lower right corner point of the enclosing rectangle.
- **Column2** (output control) ......... rectangle.corner.x(-array) → VARIANT (real) Column coordinate of lower right corner point of the enclosing rectangle.

Complexity
If \( N \) is the number of contour points, the runtime complexity is \( O(N) \).

Result
SmallestRectangle1Xld returns TRUE if the input is not empty. If the input is empty the behavior can be set via SetSystem::'noObjectResult',<Result>:). If necessary, an exception is raised.

Parallelization Information
SmallestRectangle1Xld is reentrant and automatically parallelized (on tuple level).

Possible Predecessors
GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld

Possible Successors
GenPolygonsXld

See also
ShapeTransXld, SmallestRectangle2Xld, SmallestCircleXld, EllipticAxisXld, AreaCenterXld

HALCON/COM Reference Manual, 2005-2-1
Smallest Rectangle 2Xld

The operator SmallestRectangle2Xld determines the smallest enclosing rectangle of a contour, i.e., the rectangle with the smallest area of all rectangles containing the contour. For this rectangle the center, the inclination, and the two radii are calculated.

If more than one contour is passed, the results are stored in tuples, the index of a value in the tuple corresponds to the index of a contour in the input. In case of an empty contour all parameters have the value 0.0 if no other behavior was set (see SetSystem).

Parameter

- **XLD** (input iconic) ... \( xld(-array) \) \( \sim \) IHXLD / IHObjectX 
  - Contour(s) to be examined.

- **Row** (output control) ... \( \sim \) VARIANT (real) 
  - Row coordinate of the center point of the enclosing rectangle.

- **Column** (output control) ... \( \sim \) VARIANT (real) 
  - Column coordinate of the center point of the enclosing rectangle.

- **Phi** (output control) ... \( \sim \) VARIANT (real) 
  - Orientation of the enclosing rectangle (arc measure)
  
  **Restriction**: \( (\left(-\frac{\pi}{2}\right) < Phi \leq \left(\frac{\pi}{2}\right)) \)

- **Length1** (output control) ... \( \sim \) VARIANT (real) 
  - First radius (half length) of the enclosing rectangle.
  
  **Restriction**: \( (Length1 \geq 0.0) \)

- **Length2** (output control) ... \( \sim \) VARIANT (real) 
  - Second radius (half width) of the enclosing rectangle.
  
  **Restriction**: \( ((Length2 \geq 0.0) \land (Length2 \leq Length1)) \)

Complexity

If \( N \) is the number of contour points and \( C \) is the number of points in the convex hull, the runtime complexity is \( O(N * \ln(N) + C^2) \).

Result

SmallestRectangle2Xld returns TRUE if the input is not empty. If the input is empty the behavior can be set via SetSystem({::’noObjectResult’,<Result>:}). If necessary, an exception is raised.

Parallelization Information

SmallestRectangle2Xld is reentrant and automatically parallelized (on tuple level).

Possible Predecessors

- GenContoursSkeletonXld
- EdgesSubPix
- ThresholdSubPix
- GenContourPolygonXld

Possible Successors

- GenPolygonsXld

See also

- SmallestRectangle1
- SmallestCircle
- EllipticAxisXld

Alternatives

- SmallestRectangle1
- ShapeTransXld
15.4 Transformation

Add noise to XLD contours.

AddNoiseWhiteContourXld adds noise to the input XLD contours Contours and returns the noisy contours in NoisyContours. For each point along the original contours the local regression line (i.e., a least-squares approximating line) based on NumRegrPoints neighboring points is determined. Then the point is shifted perpendicular to this line. The length of the shifts corresponds to white noise, equally distributed in the interval \([-Amp,Amp]\) generated by using the C function “drand48” with an initial time dependent seed.

- **Contours** (input iconic) ...............................\(\text{xld\_cont(-array)}\)  \(\sim\) HXLDCntX / IHObjectX
  - Original contours.
- **NoisyContours** (output iconic) ......................\(\text{xld\_cont(-array)}\)  \(\sim\) HXLDCntX / HUntypedObjectX
  - Noisy contours.
- **NumRegrPoints** (input control) .......................integer  \(\sim\) long / VARIANT
  - Number of points used to calculate the regression line.
    - Default Value : 5
    - Suggested values : \(\text{NumRegrPoints} \in \{3, 5, 7, 9\}\)
    - Restriction : \((\text{NumRegrPoints} \land \text{odd})\)
- **Amp** (input control) ........................................real  \(\sim\) double / VARIANT
  - Maximum amplitude of the added noise (equally distributed in \([-Amp,Amp]\)).
    - Default Value : 1.0
    - Suggested values : \(\text{Amp} \in \{0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 10.0\}\)
    - Restriction : \((\text{Amp} > 0)\)

**Example**

draw_ellipse(WindowHandle, Row, Column, Phi, Radius1, Radius2)
gen_ellipse_contour_xld(Ellipse, Row, Column, Phi, Radius1, Radius2, 0, 6.28319, ’positive’)
add_noise_white_contour_xld(Ellipse, NoisyEllipse, 5, 0.5)
fit_ellipse_contour_xld (NoisyEllipse, ’fitzgibbon’, -1, 2, 0, 200, 3, 2.0, ERow, EColumn, EPhi, ERadiuss1, ERadiuss2, EStartPhi, EEndPhi, ’EPointOrder’).

**Result**

AddNoiseWhiteContourXld returns TRUE if all parameter values are correct. If the input is empty the behaviour can be set via SetSystem(’noObjectResult’,<Result>). If necessary, an exception is raised.

**Parallelization Information**

AddNoiseWhiteContourXld is reentrant, local, and processed without parallelization.

**Possible Predecessors**

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, GenEllipseContourXld

**Possible Successors**

SmoothContoursXld

**See also**

SmoothContoursXld, AddNoiseWhite

Sub-pixel operators
Apply an arbitrary affine 2D transformation to XLD contours.

**AffineTransContourXld** applies an arbitrary affine 2D transformation, i.e., scaling, rotation, translation, and slant (skewing), to the XLD contours given in `Contours` and returns the transformed contours in `ContoursAffinTrans`. The affine transformation is described by the homogeneous transformation matrix given in `HomMat2D`, which can be created using the operators `HomMat2dIdentity`, `HomMat2dScale`, `HomMat2dRotate`, `HomMat2dTranslate`, etc., or be the result of operators like `VectorAngleToRigid`.

The components of the homogeneous transformation matrix are interpreted as follows: The row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system for the image. In particular, this assures that rotations are performed in the correct direction. Note that the (x,y) order of the matrices quite naturally corresponds to the usual (row,column) order for coordinates in the image.

---

### Parameter

- **Contours** (input iconic) ... xld_cont(-array) \(\Rightarrow\) HXLDContX / IHObjectX
  - Input XLD contours.

- **ContoursAffinTrans** (output iconic) ... xld_cont(-array) \(\Rightarrow\) HXLDContX / HUntypedObjectX
  - Transformed XLD contours.

- **HomMat2D** (input control) ... affine2d \(\Rightarrow\) VARIANT / HHomMat2dX (real)
  - Input transformation matrix.

  **Number of elements**: 6

---

### Result

**AffineTransContourXld** returns TRUE if all parameter values are correct. If necessary, an exception is raised.

---

### Parallelization Information

**AffineTransContourXld** is reentrant and automatically parallelized (on tuple level).

---

### Possible Predecessors

- `HomMat2dIdentity`, `HomMat2dTranslate`, `HomMat2dRotate`, `HomMat2dScale`

---

### See also

- **AffineTransImage**, **AffineTransRegion**

---

### Module

- **Sub-pixel operators**

---

Apply an arbitrary affine transformation to XLD polygons.

**AffineTransPolygonXld** applies an arbitrary affine transformation, i.e., scaling, rotation, translation, and slant (skewing), to the XLD polygons given in `Polygons` and returns the transformed polygons in `PolygonsAffinTrans`. The affine transformation is described by the homogeneous transformation matrix given in `HomMat2D`. This matrix can be created using the operators `HomMat2dIdentity`, `HomMat2dScale`, `HomMat2dRotate`, `HomMat2dTranslate`, etc., or be the result of operators like `VectorAngleToRigid`.

---

HALCON 6.1.4
The components of the homogeneous transformation matrix are interpreted as follows: The row coordinate of the image corresponds to the x coordinate of the matrix, while the column coordinate of the image corresponds to the y coordinate of the matrix. This is necessary to obtain a right-handed coordinate system for the image. In particular, this assures that rotations are performed in the correct direction. Note that the \((x, y)\) order of the matrices quite naturally corresponds to the usual \((\text{row}, \text{column})\) order for coordinates in the image.

---

**Attention**

The XLD contours that are possibly referenced by **Polygons** are neither transformed nor stored with the output polygons, since this is generally impossible without creating inconsistencies for the attributes of the XLD contours. Hence, operators that access the contours associated with a polygon, e.g., **SplitContoursXld** will not work correctly.

---

**Parameter**

- **Polygons** (input iconic) \(\text{xld}\_\text{poly}(-\text{array})\) \(\leadsto\) **HXLDPolyX / IHObjectX**
  
  Input XLD polygons.

- **PolygonsAffinTrans** (output iconic) \(\text{xld}\_\text{poly}(-\text{array})\) \(\leadsto\) **HXLDPolyX / HUntypedObjectX**
  
  Transformed XLD polygons.

- **HomMat2D** (input control) \(\text{affine2d} \leadsto\) **VARIANT / HHomMat2dX (real)**
  
  Input transformation matrix.

**Number of elements**: 6

---

**Result**

**AffineTransPolygonXld** returns TRUE if all parameter values are correct. If necessary, an exception is raised.

---

**Parallelization Information**

**AffineTransPolygonXld** is **reentrant** and automatically **parallelized** (on tuple level).

---

**Possible Predecessors**

**HomMat2dIdentity, HomMat2dTranslate, HomMat2dRotate, HomMat2dScale**

---

**See also**

**AffineTransImage, AffineTransRegion, AffineTransContourXld**

---

**Module**

---

**Sub-pixel operators**

---

Clip an XLD contour.

**ClipContoursXld** clips all XLD contours given in **Contours**, i.e., only contour points contained in the rectangle given by **Row1**, **Column1**, **Row2**, and **Column2** are returned on output. If necessary, contours are split, and several new contours are produced. The resulting contours are returned in **ClippedContours**.

---

**Parameter**

- **Contours** (input iconic) \(\text{xld}\_\text{cont} \leadsto\) **HXLDContX / IHObjectX**
  
  Contours to be clipped.

- **ClippedContours** (output iconic) \(\text{xld}\_\text{cont}(-\text{array})\) \(\leadsto\) **HXLDContX / HUntypedObjectX**
  
  Clipped contours.

- **Row1** (input control) \(\text{rectangle.origin.y} \leadsto\) **long / VARIANT**
  
  Row coordinate of the upper left corner of the clip rectangle.

  **Default Value**: 0

  **Suggested values**: \(\text{Row1} \in \{0, 500, 1000, 1500, 2000\}\)

- **Column1** (input control) \(\text{rectangle.origin.x} \leadsto\) **long / VARIANT**
  
  Column coordinate of the upper left corner of the clip rectangle.

  **Default Value**: 0

  **Suggested values**: \(\text{Column1} \in \{0, 500, 1000, 1500, 2000\}\)
15.4. TRANSFORMATION

Row2 (input control) .................................. rectangle.corner.y → long / VARIANT
Row coordinate of the lower right corner of the clip rectangle.
Default Value : 512
Suggested values : Row2 ∈ {512, 1024, 1536, 2048}

Column2 (input control) ................................ rectangle.corner.x → long / VARIANT
Column coordinate of the lower right corner of the clip rectangle.
Default Value : 512
Suggested values : Column2 ∈ {512, 1024, 1536, 2048}

Parallelization Information
ClipContoursXld is reentrant and processed without parallelization.

Possible Predecessors
GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

Possible Successors
GenPolygonsXld

See also
ClipRegion, CropPart

Module

Sub-pixel operators

Combine road hypotheses from two resolution levels.

CombineRoadsXld combines road hypotheses obtained from two different resolution levels. The algorithm selects only those hypotheses which mutually support each other in both resolution levels. The parameters EdgePolygons, ModParallels and ExtParallels correspond to the road hypotheses from the highest resolution level. The parameter CenterLines is the result of road extraction in a lower resolution level. Those of the polygons EdgePolygons are returned for which evidence of being roadsides is found in both resolution levels. The parameters MaxAngleParallel and MaxAngleColinear determine the angle, that may be formed by two parallel collinear line segments, respectively. The parameters MaxDistanceParallel and MaxDistanceColinear determine the maximum allowed distance of two parallel or collinear line segments, respectively. The combination is done using a number of rules.

Parameter

EdgePolygons (input iconic) .......................... xld_poly → HXLDPolyX / IHOBJECTX
XLD polygons to be examined.

ModParallels (input iconic) .......................... xld_mod_para → HXLDMODParaX / IHOBJECTX
Modified parallels obtained from EdgePolygons.

ExtParallels (input iconic) .......................... xld_ext_para → HXLDEXTParaX / IHOBJECTX
Extended parallels obtained from EdgePolygons.

CenterLines (input iconic) .......................... xld_poly → HXLDPolyX / IHOBJECTX
Road-center-line polygons to be examined.

HALCON 6.1.4
CHAPTER 15. XLD

RoadSides (output iconic) RoadSides found.

MaxAngleParallel (input control) Maximum angle between two parallel line segments.
Default Value: 0.523598775598
Suggested values: MaxAngleParallel ∈ \{0.349065850399, 0.523598775598, 0.6981317008\}
Restriction: \((0 \leq \text{MaxAngleParallel} \leq \frac{\pi}{2})\)

MaxAngleColinear (input control) Maximum angle between two collinear line segments.
Default Value: 0.261799387799
Suggested values: MaxAngleColinear ∈ \{0.174532925199, 0.261799387799, 0.349065850399\}
Restriction: \((0 \leq \text{MaxAngleColinear} \leq \frac{\pi}{2})\)

MaxDistanceParallel (input control) Maximum distance between two parallel line segments.
Default Value: 40
Suggested values: MaxDistanceParallel ∈ \{20, 30, 40, 50, 60\}
Restriction: \((\text{MaxDistanceParallel} > 0)\)

MaxDistanceColinear (input control) Maximum distance between two collinear line segments.
Default Value: 40
Suggested values: MaxDistanceColinear ∈ \{20, 30, 40, 50, 60\}
Restriction: \((\text{MaxDistanceColinear} > 0)\)

Parallelization Information

CombineRoadsXld is processed under mutual exclusion against itself and without parallelization.

Possible Predecessors
ModParallelsXld, GenPolygonsXld, AffineTransContourXld

Possible Successors
GetPolygonXld, GetLinesXld

See also
LinesGauss, LinesFacet, GetChannelInfo, EdgesSubPix

References


Module
Sub-pixel operators

[out] HXLDContX ParallelContours HXLDContX.GenParallelContourXld
([in] String Mode, [in] VARIANT Distance )

void HOperatorSetX.GenParallelContourXld ([in] IObjectX Contours,
[out] HUntypedObjectX ParallelContours, [in] VARIANT Mode,
[in] VARIANT Distance )

Compute the parallel contour of an XLD contour.
GenParallelContourXld computes for each of the input contours Contours a parallel contour with distance Distance. The resulting contours are returned in ParallelContours. To calculate the parallel contour, the normal vector of the input contour is needed in every contour point. The parameter Mode determines how these normal vectors are computed. If Mode = ’gradient’, it is assumed that the input contours are edges, and the normal information is obtained from the gradient direction of the edge (see EdgesSubPix). For this, the attribute ’edge.direction’ must exist for the input contour. If Mode = ’contour.normal’, a possibly existing normal information is used to determine the normals. For this, the contour attribute ’angle’ must exist (see LinesGauss
15.4. TRANSFORMATION

or EdgesSubPix). Finally, if \texttt{Mode} = '\textit{regression\_normal}', the normal vectors are determined from a local line fit to each contour point. Here, the normal vectors are oriented such that they point to the right side of the contour when the contour is traversed from start to end. In contrast to the first two modes, this mode can be used for all XLD contours, no matter how they were generated.

\textbf{Parameter}

\begin{itemize}
\item \textbf{Contours} (input iconic) \texttt{xld\_cont} \rightarrow \texttt{HXLDContX / IObjectX}
\item \textbf{ParallelContours} (output iconic) \texttt{xld\_cont} \rightarrow \texttt{HXLDContX / HUntypedObjectX}
\item \textbf{Mode} (input control) \texttt{string} \rightarrow \texttt{String / VARIANT}
\end{itemize}

\textbf{Default Value :} '\textit{regression\_normal}'

\textbf{Suggested values :} \texttt{Mode} \in \{ 'gradient', 'contour\_normal', 'regression\_normal' \}

\begin{itemize}
\item \textbf{Distance} (input control) \texttt{number} \rightarrow \texttt{VARIANT ( real, integer )}
\end{itemize}

\textbf{Parallelization Information}

\texttt{GenParallelContourXld} is reentrant and processed without parallelization.

\textbf{Possible Predecessors}

\texttt{GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, ThresholdSubPix}

\textbf{Possible Successors}

\texttt{GenPolygonsXld}

\textbf{See also}

\texttt{GetContourXld}

\textbf{Module}

\textbf{Sub-pixel operators}

\begin{verbatim}
[out] HXLDContX CurrMergedConts HXLDContX.MergeContLineScanXld
([in] HXLDContX PrevConts, [out] HXLDContX PrevMergedConts,
[in] long ImageHeight, [in] double Margin, [in] String MergeBorder,
[in] long MaxImagesCont )

void HOperatorSetX.MergeContLineScanXld ([in] IObjectX CurrConts,
[in] IObjectX PrevConts, [out] HUntypedObjectX CurrMergedConts,
[out] HUntypedObjectX PrevMergedConts, [in] VARIANT ImageHeight,
\end{verbatim}

\textbf{Merge XLD contours from successive line scan images.}

The operator \texttt{MergeContLineScanXld} connects adjacent XLD contours, which were extracted from adjacent images with the height \texttt{ImageHeight}. This operator was especially designed to connect contours that were extracted from images grabbed by a line scan camera. \texttt{CurrConts} contains the contours from the current image and \texttt{PrevConts} the contours from the previous one.

With the help of the parameter \texttt{MergeBorder} two cases can be distinguished: If the top (first) line of the current image touches the bottom (last) line of the previous image, \texttt{MergeBorder} must be set to 'top', otherwise set \texttt{MergeBorder} to 'bottom'. \texttt{MergeBorder} defines a margin to the border. Only those end points of the contours which are inside this margin are considered for the following merging process.

If the operator \texttt{MergeContLineScanXld} is used recursively, the parameter \texttt{MaxImagesCont} determines the maximum number of images which are covered by a merged contour. All points of the merged contour from an older image are removed.

The operator \texttt{MergeContLineScanXld} returns two contour arrays. \texttt{PrevMergedConts} contains all those contours from the previous input contours \texttt{PrevConts}, which could not be merged with a current contour. \texttt{CurrMergedConts} collects all current contours together with the merged parts from the previous
images. Merged contours will exceed the original image, because the previous contours are moved upward (\(\text{MergeBorder} = \text{'top'}\)) or downward (\(\text{MergeBorder} = \text{'bottom'}\)) according to the image height.

\[\text{Parameter} \quad \text{currConts} \quad \text{input iconic} \quad \text{xld}_-\text{cont}(-\text{array}) \quad \text{hxldcontx} / \text{ihObjectx}\]

\[\text{Parameter} \quad \text{prevConts} \quad \text{input iconic} \quad \text{xld}_-\text{cont}(-\text{array}) \quad \text{hxldcontx} / \text{ihObjectx}\]

\[\text{Parameter} \quad \text{currMergedConts} \quad \text{output iconic} \quad \text{xld}_-\text{cont}(-\text{array}) \quad \text{hxldcontx} / \text{hUntypedObjectx}\]

\[\text{Parameter} \quad \text{prevMergedConts} \quad \text{output iconic} \quad \text{xld}_-\text{cont}(-\text{array}) \quad \text{hxldcontx} / \text{hUntypedObjectx}\]

\[\text{Parameter} \quad \text{imageheight} \quad \text{input control} \quad \text{integer} \quad \text{long} / \text{VARIANT}\]

\[\text{Default Value:} \quad 512\]

\[\text{List of values:} \quad \text{ImageHeight} \in \{240, 480, 512\}\]

\[\text{Parameter} \quad \text{margin} \quad \text{input control} \quad \text{real} \quad \text{double} / \text{VARIANT}\]

\[\text{Default Value:} \quad 0\]

\[\text{List of values:} \quad \text{Margin} \in \{0, 1, 2, 3, 4, 5\}\]

\[\text{Parameter} \quad \text{mergeborder} \quad \text{input control} \quad \text{string} \quad \text{String} / \text{VARIANT}\]

\[\text{Default Value:} \quad \text{'top'}\]

\[\text{List of values:} \quad \text{MergeBorder} \in \{\text{'top'}, \text{'bottom'}\}\]

\[\text{Parameter} \quad \text{maxImagesCont} \quad \text{input control} \quad \text{integer} \quad \text{long} / \text{VARIANT}\]

\[\text{Default Value:} \quad 3\]

\[\text{Suggested values:} \quad \text{MaxImagesCont} \in \{1, 2, 3, 4, 5\}\]

The operator \(\text{mergeContLineScanXld}\) returns the value TRUE if the given parameters are correct. Otherwise, an exception will be raised.

\[\text{Parallelization Information}\]

\(\text{mergeContLineScanXld}\) is \textit{reentrant} and processed without parallelization.

\[\text{Module}\]

Sub-pixel operators

```plaintext
[out] HXLDContX RegressContours HXLDContX.RegressContoursXld
([in] String Mode, [in] long Iterations )
```

```plaintext
```

Calculate the parameters of a regression line to an XLD contour. \(\text{RegressContoursXld}\) calculates the following parameters for the input XLD contours \text{Contours}, and stores them with the resulting contours as global attributes:

- the coordinates of the normal vector of the regression line, i.e., the least-squares approximating line, of all contour points; the normal vector always points from the origin to the line (attributes: \text{'regr_norm_row'}, \text{'regr_norm_col'}),
- the mean of the Euclidian distance of the contour points from the regression line (attribute: \text{'regr_mean_dist'}),
- the standard deviation of these distances to the regression line (attribute: \text{'regr_dev_dist'}).

For \text{Mode} = \text{'no'}, the parameters of the regression line are calculated for all points of the contour. In addition, three different kinds of outlier treatment can be applied. Outliers are contour points which do not lie on the general contour direction in an “obvious” manner, and thus “distort” the resulting regression line. 

\text{Mode} =
• 'drop': All contour points further away from the contour than the mean distance to the regression line are ignored for the calculation of the undistorted regression line.
• 'gauss': The distances of the contour points are weighted according to their probability of occurrence in a Gaussian distribution around the normal regression line.
• 'median': Here, also a normal distribution is assumed for the distances to the normal regression line, however with the outlier-independent standard deviation \( \text{median}(\text{alldist}) \). Again, the distances are weighted, and points further away than a certain distance are ignored for the undistorted regression line.

The calculation of the undistorted regression line can be iterated several times (Iterations).

---

**Parameter**

- Contours (input iconic) \( \text{xld\_cont} \) \( \rightarrow \text{HXLDContX} / \text{HHObjectX} \)
  Input XLD contours.
- RegressContours (output iconic) \( \text{xld\_cont} \) \( \rightarrow \text{HXLDContX} / \text{HUntypedObjectX} \)
  Resulting XLD contours.
- Mode (input control) \( \rightarrow \text{String} / \text{VARIANT} \)
  Type of outlier treatment.
  Default Value: 'no'
  List of values: Mode \( \in \{ \text{no}, \text{drop}, \text{gauss}, \text{median}\} \)
- Iterations (input control) \( \rightarrow \text{long} / \text{VARIANT} \)
  Number of iterations for the outlier treatment.
  Default Value: 1
  Suggested values: Iterations \( \in \{1, 2, 3, 5, 10, 20\} \)
---

**Parallelization Information**

RegressContoursXld is reentrant and processed without parallelization.

---

**Possible Predecessors**

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

**Possible Successors**

GetRegressParamsXld

---

See also

SmoothContoursXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

---

**References**


---

**Module**

Sub-pixel operators

---

```c
[out] HXLDContX ContoursXld = HXLDContX::SegmentContoursXld([
[in] String Mode, [in] long SmoothCont, [in] double MaxLineDist1,
[in] double MaxLineDist2 ])
```

```c
void HOperatorSetX::SegmentContoursXld ([in] HHObjectX Contours,
[out] HUntypedObjectX ContoursSplit, [in] VARIANT Mode,
[in] VARIANT SmoothCont, [in] VARIANT MaxLineDist1,
[in] VARIANT MaxLineDist2 )
```

---

**Segment contours into line segments and circular or elliptic arcs.**

SegmentContoursXld segments the input contours Contours into lines if Mode='lines', into lines and circular arcs if Mode='lines_circles', or into lines and elliptic arcs if Mode='lines_ellipses'. The segmented contours are returned in ContoursSplit. The information on whether an output contour represents a line or a circular of elliptic arc is done via the global contour attribute 'cont_approx' (see GetContourGlobalAttribXld). If 'cont_approx'=-1, the contour is best approximated by a line segment, if 'cont_approx'=0, by an elliptic arc, and if 'cont_approx'=1, by a circular arc.
SegmentContoursXld first approximates the input contours by polygons. With this, the contours are over-segmented in curved areas. After this, neighboring line segments are substituted by circular or elliptic arcs, respectively, if the contour can be approximated better by an arc. If SmoothCont is set to a value $> 0$, the input contours are first smoothed (see SmoothContoursXld). This can be necessary to prevent very short segments in the polygonal approximation and to achieve a more robust fit with circular or elliptic arcs, because the smoothing suppresses outliers on the contours.

The initial polygonal approximation is done by using the Ramer algorithm (see GenPolygonsXld) with a maximum distance of MaxLineDist1. After this, circular or elliptic arcs are fit into neighboring line segments. If the maximum distance of the resulting arc to the contour is smaller than the maximum distance of the two line segments, the two line segments are replaced with the arc. This procedure is iterated until no more changes occur.

After this, the parts of the contour that are still approximated by line segments are again segmented with a polygonal approximation with maximum distance MaxLineDist2, and the newly created line segments are merged to circular or elliptical arcs where possible. Obviously, this changes the output only if MaxLineDist2 < MaxLineDist1. This two-step approach is more efficient than a one-step approach with MaxLineDist2, since fewer line segments are generated in the first step, and hence the circle or ellipse fit has to be performed less often. Therefore, parts of the input contours that can be approximated by long arcs can be found more efficiently.

In the second step, parts of the input contours that can be approximated by short arcs are found and the end parts of long arcs are refined.

---

**Parameter**

- **Contours** (input iconic) .................................................. xld_cont(-array) $\rightarrow$ HXLDContX / HObjectX
  Contours to be segmented.

- **ContoursSplit** (output iconic) ........................................... xld_cont $\rightarrow$ HXLDContX / HUntypedObjectX
  Segmented contours.

- **Mode** (input control) ......................................................... string $\rightarrow$ String / VARIANT
  Mode for the segmentation of the contours.
  Default Value: 'lines_circles'
  List of values: Mode $\in \{ 'lines', 'lines_circles', 'lines_ellipses' \}$

- **SmoothCont** (input control) .............................................. integer $\rightarrow$ long / VARIANT
  Number of points used for smoothing the contours.
  Default Value: 5
  Suggested values: SmoothCont $\in \{ 0, 3, 5, 7, 9 \}$
  Restriction: \((\text{SmoothCont} \geq 3) \lor (\text{SmoothCont} \land \text{odd})\)

- **MaxLineDist1** (input control) ............................................ real $\rightarrow$ double / VARIANT
  Maximum distance between a contour and the approximating line (first iteration).
  Default Value: 4.0
  Suggested values: MaxLineDist1 $\in \{ 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 \}$
  Restriction: \((\text{MaxLineDist1} \geq 0.0)\)

- **MaxLineDist2** (input control) ............................................ real $\rightarrow$ double / VARIANT
  Maximum distance between a contour and the approximating line (second iteration).
  Default Value: 2.0
  Suggested values: MaxLineDist2 $\in \{ 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 \}$
  Restriction: \((\text{MaxLineDist2} \geq 0.0)\)

---

Example

```plaintext
read_image (Image, 'pumpe')
edges_sub_pix (Image, Edges, 'canny', 1.5, 15, 40)
segment_contours_xld (Edges, ContoursSplit, 'lines_circles', 5, 4, 2)
count_obj (ContoursSplit, Number)
gen_empty_obj (Lines)
gen_empty_obj (Circles)
for I := 1 to Number by 1
    select_obj (ContoursSplit, Contour, I)
    get_contour_global_attrib_xld (Contour, 'cont_approx', Type)
    if (Type = -1)
        concat_obj (Lines, Contour, Lines)
    else
```
concat_obj (Circles, Contour, Circles)
endif
endfor
fit_line_contour_xld (Lines, 'tukey', -1, 0, 5, 2, RowBegin, ColBegin, RowEnd, ColEnd, Nr, Nc)
fit_circle_contour_xld (Circles, 'atukey', -1, 2, 0, 3, 2, Row, Column, Radius, StartPhi, EndPhi, PointOrder)

--- Parallelization Information ---
SegmentContoursXld is local and processed completely exclusively without parallelization.

--- Possible Predecessors ---
GenContoursSkeletonXld, LinesGauss, EdgesSubPix

--- Possible Successors ---
FitLineContourXld, FitEllipseContourXld, FitCircleContourXld

--- See also ---
SplitContoursXld, GetContourGlobalAttribXld, SmoothContoursXld, GenPolygonsXld

--- Module ---
Sub-pixel operators

[out] HUntypedObjectX XLDTrans HXLDX.ShapeTransXld ([in] String Type )
void HOperatorSetX.ShapeTransXld ([in] IObjectX XLD, [out] HUntypedObjectX XLDTrans, [in] VARIANT Type )

Transform the shape of a contour.
ShapeTrans transforms the input contour depending on the parameter Type:

'convex' Convex hull.
'ellipse' Ellipse with the same moments and area as the input contour. The closed input contour must not intersect itself, otherwise the resulting ellipse is not meaningful.
'outer_circle' Smallest enclosing circle.
'rectangle1' Smallest enclosing rectangle parallel to the coordinate axes.
'rectangle2' Smallest enclosing rectangle.

--- Parameter ---

XLD (input iconic) ................................. xld(-array) ~ IHXLDX / IObjectX Contour(s) to be transformed.
XLDTrans (output iconic) ................................. xld(-array) ~ HUntypedObjectX Transformed contour(s).
Type (input control) ................................. string ~ String / VARIANT Type of transformation.
Default Value : 'convex'
List of values : Type ∈ {'convex', 'ellipse', 'outer_circle', 'rectangle1', 'rectangle2'}

--- Result ---
ShapeTransXld returns TRUE if all parameters are correct. The behavior in case of empty input (no contours given) can be set via SetSystem('noObjectResult',<Result>). If necessary, an exception is raised.

--- Parallelization Information ---
ShapeTransXld is reentrant and automatically parallelized (on tuple level).

--- Possible Predecessors ---
GenContoursSkeletonXld, EdgesSubPix, ThresholdSubPix, GenContourPolygonXld

--- See also ---
SmallestCircleXld, SmallestRectangle1Xld, SmallestRectangle2Xld, EllipticAxisXld
SmallestCircleXld, SmallestRectangle1Xld, SmallestRectangle2Xld, EllipticAxisXld

Sub-pixel operators

Smooth an XLD contour.

SmoothContoursXld smooths the input XLD contours Contours and returns the smoothed contours in SmoothedContours. The smoothing is done by projecting the contours’ points onto a local regression line (i.e., a least-squares approximating line), which is computed from NumRegrPoints on each side of the current contour point. This operator should be called, for example, before contours are scaled.

Parameter

- Contours (input iconic) .................................................. xld_cont ⇒ HXLDContX / IHObjectX
  Contour to be smoothed.
- SmoothedContours (output iconic) ..................................... xld_cont ⇒ HXLDContX / HUntypedObjectX
  Smoothed contour.
- NumRegrPoints (input control) ........................................ integer ⇒ long / VARIANT
  Number of points used to calculate the regression line.
  Default Value: 5
  Suggested values: NumRegrPoints ∈ {3, 5, 7, 9}
  Restriction: (NumRegrPoints ∧ odd)

Parallelization Information

SmoothContoursXld is reentrant and processed without parallelization.

Possible Predecessors

GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix

Possible Successors

AffineTransContourXld, GenPolygonsXld, LocalMaxContoursXld

See also

GetContourXld

Module

Split XLD contours at dominant points.

SplitContoursXld splits the contours which were used to generate the polygons Polygons at prominent points. If the mode ‘polygon’ is selected, the contours are split at the polygons’ control points. In mode ‘dominant’, they are split at dominant points, i.e., at points for which the calculated change of direction is larger than the (empirically determined) threshold \(2\pi \frac{\text{Weight}}{\text{contour length}}\), and for which in the (empirically determined) neighborhood of \(\sqrt{\text{Smooth} + \log(\text{contour length})}\) points no larger change of direction occurs. The contour direction is determined from the direction of the regression line (i.e., the least-squares approximating line) through all points in a neighborhood of Smooth points around a contour point. The directions thus determined are smoothed.
with a Gaussian of width $\text{Smooth}$. $\text{Weight}$ is a weighting factor for the sensitiveness of the operator. The larger $\text{Weight}$ is selected, the less dominant points are found.

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\triangleright$ <strong>Polygons</strong> (input iconic) ( \text{xld}_\text{poly(-array)} ) $\sim$ \text{HXLDPolyX} / \text{IHOBJECTX}</td>
</tr>
<tr>
<td>$\triangleright$ <strong>Contours</strong> (output iconic) ( \text{xld}_\text{cont(-array)} ) $\sim$ \text{HXLDContX} / \text{HUntypedObjectX}</td>
</tr>
<tr>
<td>$\triangleright$ <strong>Mode</strong> (input control) ( \text{string} ) $\sim$ \text{String} / \text{VARIANT}</td>
</tr>
<tr>
<td>$\triangleright$ <strong>Weight</strong> (input control) ( \text{integer} ) $\sim$ \text{long} / \text{VARIANT}</td>
</tr>
<tr>
<td>$\triangleright$ <strong>Smooth</strong> (input control) ( \text{integer} ) $\sim$ \text{long} / \text{VARIANT}</td>
</tr>
</tbody>
</table>

**Parallelization Information**

$\text{SplitContoursXld}$ is reentrant and processed without parallelization.

**Possible Predecessors**

- $\text{GenPolygonsXld}$
- $\text{RegressContoursXld}$

**Possible Successors**

- $\text{GenContoursSkeletonXld}$
- $\text{LinesGauss}$
- $\text{LinesFacet}$
- $\text{EdgesSubPix}$

**See also**

- $\text{Module}$

Sub-pixel operators

```plaintext
[out] \text{HXLDContX} \text{UnionContours}
\text{HXLDContX.UnionStraightContoursHistoXld}
(out) \text{HXLDContX} \text{SelectedContours}, [in] \text{long} \text{RefLineStartRow},
[in] \text{long} \text{RefLineStartColumn}, [in] \text{long} \text{RefLineEndRow},
[in] \text{long} \text{RefLineEndColumn}, [in] \text{long} \text{Width}, [in] \text{long} \text{MaxWidth},
[in] \text{long} \text{FilterSize}, [out] \text{VARIANT} \text{HistoValues} )

void \text{HOperatorSetX.UnionStraightContoursHistoXld}
(in) \text{IHOBJECTX} \text{Contours}, [out] \text{HUntypedObjectX} \text{UnionContours},
(out) \text{HUntypedObjectX} \text{SelectedContours}, [in] \text{VARIANT} \text{RefLineStartRow},
[in] \text{VARIANT} \text{RefLineStartColumn}, [in] \text{VARIANT} \text{RefLineEndRow},
[in] \text{VARIANT} \text{RefLineEndColumn}, [in] \text{VARIANT} \text{Width}, [in] \text{VARIANT} \text{MaxWidth},
[in] \text{VARIANT} \text{FilterSize}, [out] \text{VARIANT} \text{HistoValues} )
```

Merge neighboring straight contours having a similar distance from a given line.

$\text{UnionStraightContoursHistoXld}$ merges neighboring XLD contours $\text{Contours}$ if certain criteria are fulfilled.

The maximum and minimum distance of the contours to an given reference line are calculated. From this distances a histogram is created. If the histogram should be smoothed, $\text{FilterSize}$ must be greater then one. Afterwards the resulting histogram is divided into ranges (from minima to minima). Contours which lie in the same range are concatenated to a new contour. If the width of Range is greater than $\text{MaxWidth}$, all contours in this range are ignored (removed). Lies an contour in two ranges or more it is also ignored. In case there are parallel contours, there is a danger of merging neighboring contours.

The parameters of the regression line are calculated new for merged contours.

The resulting contours cannot be displayed.
Before the contour parameters can be returned by \texttt{UnionStraightContoursHistoXld}, the parameters of the regression line to the contour must be calculated by calling \texttt{RegressContoursXld}.

\begin{itemize}
\item \texttt{Contours} (input iconic) \xrightarrow{} \texttt{HXLDContX / IHObjectX}
\hspace{1cm} Input XLD contours.
\item \texttt{UnionContours} (output iconic) \xrightarrow{} \texttt{HXLDContX / HUntypedObjectX}
\hspace{1cm} Output XLD contours.
\item \texttt{SelectedContours} (output iconic) \xrightarrow{} \texttt{HXLDContX / HUntypedObjectX}
\hspace{1cm} Output XLD contours.
\item \texttt{RefLineStartRow} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} y coordinate of the starting point of the reference line.
\hspace{1cm} Default Value : 0
\item \texttt{RefLineStartColumn} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} x coordinate of the starting point of the reference line.
\hspace{1cm} Default Value : 0
\item \texttt{RefLineEndRow} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} y coordinate of the endpoint of the reference line.
\hspace{1cm} Default Value : 0
\item \texttt{RefLineEndColumn} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} x coordinate of the endpoint of the reference line.
\hspace{1cm} Default Value : 0
\item \texttt{Width} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} Maximum distance.
\hspace{1cm} Default Value : 1
\item \texttt{MaxWidth} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} Maximum Width between two minimas.
\hspace{1cm} Default Value : 1
\item \texttt{FilterSize} (input control) \xrightarrow{} \texttt{long / VARIANT}
\hspace{1cm} Size of Smoothfilter
\hspace{1cm} Default Value : 1
\hspace{1cm} Typical range of values : \(1 \leq \text{FilterSize} \leq 1\)
\item \texttt{HistoValues} (output control) \xrightarrow{} \texttt{VARIANT ( integer )}
\hspace{1cm} Output Values of Histogram.
\end{itemize}

\textbf{Parallelization Information}

\texttt{UnionStraightContoursHistoXld} is \textit{reentrant} and processed \textit{without} parallelization.

\textbf{Possible Predecessors}

\texttt{RegressContoursXld}

\textbf{See also}

\texttt{FitLineContourXld, GetContourXld, GetContourAttribXld, GenContoursSkeletonXld, LinesGauss, LinesFacet, EdgesSubPix, GetRegressParamsXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld}

\textbf{Module}

\textbf{Sub-pixel operators}

\begin{verbatim}
[out] HXLDContX UnionContours HXLDContX.UnionStraightContoursXld
([in] double MaxDist, [in] double MaxDiff, [in] double Percent,
[in] String Mode, [in] VARIANT Iterations )

void HOperatorSetX.UnionStraightContoursXld
([in] IHObjectX Contours, [out] HUntypedObjectX UnionContours,
[in] VARIANT Mode, [in] VARIANT Iterations )
\end{verbatim}

Merge neighboring straight contours having a similar direction.

\hspace{1cm} Attention

\hspace{1cm} Before the contour parameters can be returned by \texttt{UnionStraightContoursHistoXld}, the parameters of the regression line to the contour must be calculated by calling \texttt{RegressContoursXld}.
UnionStraightContoursXld merges neighboring XLD contours Contours if certain criteria are fulfilled. The merging is not done recursively, and only between two contours. The parameter Iterations controls how often this union step is executed.

Two contours are merged if the shortest distance between their end points (end points are the projections of the contours’ first and last points onto its regression line) is smaller than MaxDist, and if the difference in direction (i.e., the regression lines’ direction) is smaller than MaxDiff (radians).

If only one of the criteria is fulfilled, the decision on merging can be influenced by the weighting factor Percent, which allows the exceeding of one limit to be balanced by the other value remaining below the limit by the same amount. The end point distance is weighted by Percent, while the directional difference is weighted by 100 − Percent.

If, for example, two contours have an end point distance of 5.0 and a directional difference of 0.5 (with the limits chosen being MaxDist = 4.0 and MaxDiff = 0.625), both values differ from the limits by 25%. By choosing Percent = 60%, the larger end point distance is weighted more than the smaller directional difference, and thus the contours are not merged. Contrary, if Percent = 40% is chosen, the contours are merged.

For Percent = 100%, only the end point distance is taken into account, while for Percent = 0% only the difference of direction is used. If Percent = 50% both criteria are equally valid.

In case there are parallel contours, there is a danger of merging neighboring contours. If this is to be avoided, Mode = ‘noparallel’ has to be chosen, while otherwise Mode = ‘paralleltoo’ suffices. For Mode = ‘every’, contours are merged unconditionally. All other parameters have no influence in this case.

The parameters of the regression line are calculated anew for merged contours.

Attention

Before the contour parameters can be returned by GetRegressParamsXld, the parameters of the regression line to the contour must be calculated by calling RegressContoursXld.

Parameter

- Contours (input iconic) xld_cont ~ HXLDContX / IHObjectX
  - Input XLD contours.
- UnionContours (output iconic) xld_cont ~ HXLDContX / HUntypedObjectX
  - Output XLD contours.
- MaxDist (input control) real ~ double / VARIANT
  - Maximum distance of the contours’ endpoints.
    Default Value: 5.0
- MaxDiff (input control) angle.rad ~ double / VARIANT
  - Maximum difference in direction.
    Default Value: 0.5
- Percent (input control) real ~ double / VARIANT
  - Weighting factor for the two selection criteria.
    Default Value: 50.0
- Mode (input control) string ~ String / VARIANT
  - Should parallel contours be taken into account?
    Default Value: ’noparallel’
    List of values: Mode ∈ {’noparallel’, ’paralleltoo’, ’every’}
- Iterations (input control) string ~ VARIANT (integer, string)
  - Number of iterations or ‘maximum’.
    Default Value: ’maximum’
    Suggested values: Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ’maximum’}
    Typical range of values: 1 ≤ Iterations ≤ 10
    Minimum Increment: 1
    Recommended Increment: 1

Parallelization Information

UnionStraightContoursXld is reentrant and processed without parallelization.

Possible Predecessors

RegressContoursXld

See also

FitLineContourXld, GetContourXld, GetContourAttribXld, GenContoursSkeletonXld.
LinesGauss, LinesFacet, EdgesSubPix, GetRegressParamsXld, GetContourGlobalAttribXld, QueryContourGlobalAttribsXld

Sub-pixel operators
Chapter 16
Class Hierarchy

16.1 HBarcodeX

Grouping class for barcode related functionality.

<table>
<thead>
<tr>
<th></th>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>12</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>no</td>
</tr>
<tr>
<td>Object Identity</td>
<td>no</td>
</tr>
<tr>
<td>(through internal handle)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode1DBarCode</td>
<td>P.715</td>
</tr>
<tr>
<td>Decode2DBarCode</td>
<td>P.716</td>
</tr>
<tr>
<td>Discrete1DBarCode</td>
<td>P.717</td>
</tr>
<tr>
<td>Find1DBarCode</td>
<td>P.718</td>
</tr>
<tr>
<td>Find1DBarCodeRegion</td>
<td>P.721</td>
</tr>
<tr>
<td>Find2DBarCode</td>
<td>P.722</td>
</tr>
<tr>
<td>Gen1DBarCodeDescr</td>
<td>P.725</td>
</tr>
<tr>
<td>Gen1DBarCodeDescrGen</td>
<td>P.726</td>
</tr>
<tr>
<td>Gen2DBarCodeDescr</td>
<td>P.727</td>
</tr>
<tr>
<td>Get1DBarCode</td>
<td>P.729</td>
</tr>
<tr>
<td>Get2DBarCode</td>
<td>P.730</td>
</tr>
<tr>
<td>Get2DBarCodePos</td>
<td>P.734</td>
</tr>
</tbody>
</table>

Decoding of a sequence of elements of a bar code.

Decode 2D bar code data.

Generate a discrete bar code from the elements widths.

Look for one bar code in an image.

Look for multiple bar code regions in an image.

Find regions that might contain a 2D bar code.

Generate a description of a 1D bar code.

Generate a generic description of a 1D bar code.

Generate a generic description of a 2D bar code class.

Extract the widths of the elements inside a bar code region.

Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).

Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.
16.2 HBgEstiX

Represents an instance of a background estimator.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Constructor(s)

CreateBgEsti

Generate and initialize a data set for the background estimation.

Methods

GetBgEstiParams

Return the parameters of the data set.

GiveBgEsti

Return the estimated background image.

RunBgEsti

Estimate the background and return the foreground region.

SetBgEstiParams

Change the parameters of the data set.

UpdateBgEsti

Change the estimated background image.

16.3 HClassBoxX

Represents an instance of a classificator.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

Constructor(s)

CreateClassBox

Create a new classificator.

Methods

ClassNdimBox

Classify pixels using hyper-cuboids.

DescribeClassBox

Description of the classificator.

EnquireClassBox

Classify a tuple of attributes.
16.4 HFeatureSetX

Represents an instance of a training used for the classifier.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
<tr>
<td>Constructor(s)</td>
</tr>
</tbody>
</table>

16.5 HFileX

Represents an instance of a file.
CHAPTER 16. CLASS HIERARCHY

16.6 HFramegrabberX

Represents an instance of a frame grabber.

Class Description

Number of Methods ................................................. 10
Number of Properties ............................................. 14
Abstract Class ...................................................... no
Base Class(es) ...................................................... <none>
Automatic Destructor .......................................... yes, through CloseFramegrabber (not member of this class!)
Object Identity (through internal handle) .................. yes

Constructor(s)

OpenFramegrabber ................................................ P.361
Open and configure a frame grabber.

Methods

GetFramegrabberLut ................................................. P.353
Query frame grabber lut.

GetFramegrabberParam ........................................... P.353
Get specific parameters for a frame grabber.

GrabImage ......................................................... P.354
Grab an image from the specified frame grabber.

GrabImageAsync .................................................. P.355
Grab of an image from the specified frame grabber and start of the asynchronous grab of the next image.

GrabImageStart .................................................. P.356
Start an asynchronous grab of an image from the specified frame grabber.

GrabRegion ....................................................... P.357
Grab and segment an image from the specified frame grabber.

HALCON/COM Reference Manual, 2005-2-1
16.6. HFRAMEGRABBERX

- **GrabRegionAsync** ................................................................. P.358
  Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.

- **SetFramegrabberLut** ........................................................... P.363
  Set frame grabber lut.

- **SetFramegrabberParam** ....................................................... P.364
  Set specific parameters for a frame grabber.

---

Properties

- **BitsPerChannel** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ColorSpace** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ExternalTrigger** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Field** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Gain** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **HorizontalResolution** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ImageHeight** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **ImageWidth** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **LineIn** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Name** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **Port** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **StartColumn** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **StartRow** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.

- **VerticalResolution** (read-only)
  mapped to: GetFramegrabberParam
  Get specific parameters for a frame grabber.
16.7 HFunction1dX

Represents an instance of a 1d function.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>19</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>no</td>
</tr>
<tr>
<td>Object Identity</td>
<td>yes</td>
</tr>
<tr>
<td>(through internal handle)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constructor(s)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateFunct1DArray</td>
<td>P.791</td>
</tr>
<tr>
<td>CreateFunct1DPairs</td>
<td>P.791</td>
</tr>
<tr>
<td>ReadFunct1D</td>
<td>P.796</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AbsFunct1D</td>
<td>P.790</td>
</tr>
<tr>
<td>DistanceFunct1D</td>
<td>P.792</td>
</tr>
<tr>
<td>Funct1DToPairs</td>
<td>P.793</td>
</tr>
<tr>
<td>GetPairFunct1D</td>
<td>P.793</td>
</tr>
<tr>
<td>IntegrateFunct1D</td>
<td>P.794</td>
</tr>
<tr>
<td>MatchFunct1DTrans</td>
<td>P.794</td>
</tr>
<tr>
<td>NegateFunct1D</td>
<td>P.795</td>
</tr>
<tr>
<td>NumPointsFunct1D</td>
<td>P.796</td>
</tr>
<tr>
<td>SampleFunct1D</td>
<td>P.797</td>
</tr>
<tr>
<td>ScaleYFunct1D</td>
<td>P.797</td>
</tr>
<tr>
<td>SmoothFunct1DGauss</td>
<td>P.798</td>
</tr>
<tr>
<td>SmoothFunct1DMean</td>
<td>P.798</td>
</tr>
<tr>
<td>TransformFunct1D</td>
<td>P.799</td>
</tr>
<tr>
<td>WriteFunct1D</td>
<td>P.800</td>
</tr>
<tr>
<td>XRangeFunct1D</td>
<td>P.800</td>
</tr>
<tr>
<td>YRangeFunct1D</td>
<td>P.801</td>
</tr>
</tbody>
</table>

Smallest and largest y value of the function.

Smallest and largest x value of the function.
16.8  HGNUPLOTX

Represents an instance of a connection to a gnuplot process.

- **Class Description**
  - Number of Methods: 5
  - Number of Properties: 0
  - Abstract Class: no
  - Base Class(es): <none>
  - Automatic Destructor: yes, through GnuplotClose (not member of this class!)
  - Object Identity: (through internal handle) yes

- **Constructor(s)**
  - GnuplotOpenFile
    - Open a gnuplot file for visualization of images and control values.
  - GnuplotOpenPipe
    - Open a pipe to a gnuplot process for visualization of images and control values.

- **Methods**
  - GnuplotPlotCtrl
    - Plot control values using gnuplot.
  - GnuplotPlotFunct1D
    - Plot a function using gnuplot.
  - GnuplotPlotImage
    - Visualize images using gnuplot.

16.9  HHistogramX

Group class for histogram functionality.

- **Class Description**
  - Number of Methods: 2
  - Number of Properties: 0
  - Abstract Class: no
  - Base Class(es): <none>
  - Automatic Destructor: no
  - Object Identity: (through internal handle) no

- **Methods**
  - GenRegionHisto
    - Convert a histogram into a region.
  - HistoToThresh
    - Determine gray value thresholds from a histogram.

16.10  HHomMat2dX

Represents a homogeneous 2D transformation matrix.
### Class Description

| Number of Methods | 20 |
| Number of Properties | 0 |
| Abstract Class | no |
| Base Class(es) | <none> |
| Automatic Destructor | no |
| Object Identity (through internal handle) | yes |

#### Constructor(s)

- **DvfToHomMat2d**
  - Approximate an affine map from a displacement vector field.

- **HomMat2dIdentity**
  - Generate the homogeneous transformation matrix of the identical 2D transformation.

- **ReadWorldFile**
  - Read the geo coding from an ARC/INFO world file.

- **VectorAngleToRigid**
  - Compute a rigid affine transformation from points and angles.

- **VectorToHomMat2d**
  - Approximate an affine transformation from point correspondences.

- **VectorToRigid**
  - Approximate a rigid affine transformation from point correspondences.

- **VectorToSimilarity**
  - Approximate an similarity transformation from point correspondences.

#### Methods

- **AffineTransContourXld**
  - Apply an arbitrary affine 2D transformation to XLD contours.

- **AffineTransImage**
  - Apply an arbitrary affine 2D transformation to images.

- **AffineTransImageSize**
  - Apply an arbitrary affine 2D transformation to an image and specify the output image size.

- **AffineTransPoint2D**
  - Apply an arbitrary affine 2D transformation to points.

- **AffineTransPolygonXld**
  - Apply an arbitrary affine transformation to XLD polygons.

- **AffineTransRegion**
  - Apply an arbitrary affine 2D transformation to regions.

- **HomMat2dCompose**
  - Multiply two homogeneous 2D transformation matrices.

- **HomMat2dInvert**
  - Invert a homogeneous 2D transformation matrix.

- **HomMat2dRotate**
  - Add a rotation to a homogeneous 2D transformation matrix.

- **HomMat2dScale**
  - Add a scaling to a homogeneous 2D transformation matrix.

- **HomMat2dSlant**
  - Add a slant to a homogeneous 2D transformation matrix.

- **HomMat2dToAffinePar**
  - Compute the affine transformation parameters from a homogeneous 2D transformation matrix.

- **HomMat2dTranslate**
  - Add a translation to a homogeneous 2D transformation matrix.
16.11 HHomMat3dX

Represents a homogeneous 3D transformation matrix.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- **HomMat3dIdentity**
  
  Generate the homogeneous transformation matrix of the identical 3D transformation.

Methods

- **AffineTransPoint3D**
  
  Apply an arbitrary affine 3D transformation to points.

- **HomMat3dCompose**
  
  Multiply two homogeneous 3D transformation matrices.

- **HomMat3dInvert**
  
  Invert a homogeneous 3D transformation matrix.

- **HomMat3dRotate**
  
  Add a rotation to a homogeneous 3D transformation matrix.

- **HomMat3dScale**
  
  Add a scaling to a homogeneous 3D transformation matrix.

- **HomMat3dToPose**
  
  Convert a homogeneous transformation matrix into a 3D pose.

- **HomMat3dTranslate**
  
  Add a translation to a homogeneous 3D transformation matrix.

16.12 HImageX

Represents an instance of an image object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- **GenBandfilter**
  
  Generate an ideal band filter.

- **GenBandpass**
  
  Generate an ideal bandpass filter.

- **GenDiscSe**
  
  Generate ellipsoidal structuring elements for gray morphology.
- **GenFilterMask** ................................................. P.100  
  Store a filter mask in the spatial domain as a real-image.

- **GenGabor** .................................................. P.101  
  Generate a Gabor filter.

- **GenHighpass** ............................................... P.103  
  Generate an ideal highpass filter.

- **GenImage1** ................................................ P.302  
  Create an image from a pointer to the pixels.

- **GenImageExtern** ........................................... P.303  
  Create an image from a pointer on the pixels with storage management.

- **GenImageRect** ............................................. P.305  
  Create an image with a rectangular domain from a pointer on the pixels (with storage management).

- **GenImage3** ................................................. P.306  
  Create an image from three pointers to the pixels (red/green/blue).

- **GenImageConst** ............................................ P.307  
  Create an image with constant gray value.

- **GenImageGrayRamp** ...................................... P.308  
  Create a gray value ramp.

- **GenImageSurfaceFirstOrder** .......................... P.311  
  Create a curved gray surface with first order polynomial.

- **GenImageSurfaceSecondOrder** ......................... P.312  
  Create a curved gray surface with second order polynomial.

- **GenImageToWorldPlaneMap** ............................ P.757  
  Generate a projection map that describes the mapping between the image plane and a the plane z=0 of a world coordinate system.

- **GenLowpass** .............................................. P.103  
  Generate an ideal lowpass filter.

- **GenPsfDefocus** ........................................... P.159  
  Generate an impulse response of an uniform out-of-focus blurring.

- **GenPsfMotion** ............................................ P.160  
  Generate an impulse response of a (linearly) motion blurring.

- **GenRadialDistortionMap** ............................... P.760  
  Generate a projection map that describes the mapping of images corresponding to a changing radial distortion.

- **GenSinBandpass** ......................................... P.104  
  Generate a bandpass filter with sinusoidal shape.

- **GenStdBandpass** ......................................... P.105  
  Generate a bandpass filter with Gaussian or sinusoidal shape.

- **GiveBqEsti** ............................................... P.709  
  Return the estimated background image.

- **GrabImage** ................................................ P.354  
  Grab an image from the specified frame grabber.

- **GrabImageAsync** ......................................... P.355  
  Grab an image from the specified frame grabber and start of the asynchronous grab of the next image.

- **ReadGraySe** .............................................. P.440  
  Load a structuring element for gray morphology.

- **ReadImage** ............................................... P.11  
  Read an image with different file formats.

- **ReadOcrTrainf** .......................................... P.874  
  Read training characters from files and convert to images.

- **ReadOcrTrainfSelect** ................................. P.876  
  Read training specific characters from files and convert to images.
AnisotropeDiff
⊲
BitLshift
⊲
BitMask
⊲
BestMatchRot
⊲
BestMatchPreMg
⊲
BestMatch
⊲
BandpassImage

AbsImage ................................................................. P.37
Calculate the absolute value (modulus) of an image.

AccessChannel ......................................................... P.290
Access a channel of a multichannel image.

AdaptTemplate ......................................................... P.385
Adapting a template to the size of an image.

AddImage ............................................................... P.38
Add two images.

AddNoiseDistribution ................................................... P.130
Add noise to an image.

AddNoiseWhite ......................................................... P.131
Add noise to an image.

AffineTransImage ..................................................... P.29
Apply an arbitrary affine 2D transformation to images.

AffineTransImageSize .................................................. P.31
Apply an arbitrary affine 2D transformation to an image and specify the output image size.

AnisotropeDiff ........................................................ P.135
Smooth an image by edge-preserving anisotropic diffusion.

AppendChannel ........................................................ P.290
Append additional matrices (channels) to the image.

AutoThreshold ......................................................... P.597
Segment an image using thresholds determined from its histogram.

BandpassImage ......................................................... P.109
Edge extraction using bandpass filters.

BestMatch .............................................................. P.386
Searching the best matching of a template and an image.

BestMatchMg .......................................................... P.387
Searching the best grayvalue matches in a pyramid.

BestMatchPreMg ....................................................... P.388
Searching the best grayvalue matches in a pre generated pyramid.

BestMatchRot ........................................................ P.390
Searching the best matching of a template and an image with rotation.

BestMatchRotMg ...................................................... P.391
Searching the best matching of a template and a pyramid with rotation.

BinThreshold ........................................................ P.598
Segment a black-and-white image using an automatically determined threshold.

BitAnd ............................................................... P.46
Bit-by-bit AND of all pixels of the input images.

BitLshift ............................................................ P.47
Left shift of all pixels of the image.

BitMask .............................................................. P.48
Logical “AND” of each pixel using a bit mask.

BitNot ............................................................. P.49
Complement all bits of the pixels.
ConvolFft
Convolve an image with a Gabor filter in the frequency domain.

BitOr
Bit-by-bit OR of all pixels of the input images.

BitShift
Right shift of all pixels of the image.

BitSlice
Extract a bit from the pixels.

BitXor
Bit-by-bit XOR of all pixels of the input images.

CfaToRgb
Convert a single-channel color filter array image into an RGB image.

ChangeDomain
Change definition domain of an image.

ChangeFormat
Change image size.

ChangeRadialDistortionImage
Change the radial distortion of an image.

ChannelsToImage
Convert one-channel images into a multichannel image

CharThreshold
Perform a threshold segmentation for extracting characters.

CheckDifference
Compare two images pixel by pixel.

Class2DimSup
Segment an image using two-dimensional pixel classification.

Class2DimUnsup
Segment two images by clustering.

ClassNdBox
Classify pixels using hyper-cuboids.

ClassNdNorm
Classify pixels using hyper-spheres or hyper-cubes.

CompareVariationModel
Compare an image to a variation model.

ComplexToReal
Convert a complex image into two real images.

Compose2
Convert two images into a two-channel image.

Compose3
Convert 3 images into a three-channel image.

Compose4
Convert 4 images into a four-channel image.

Compose5
Convert 5 images into a five-channel image.

Compose6
Convert 6 images into a six-channel image.

Compose7
Convert 7 images into a seven-channel image.

ConvertImageType
Convert the type of an image.

Convolve
Convolve an image with a byte-mask in the frequency domain.

ConvolveGabor
Convolve an image with a Gabor filter in the frequency domain.
Convolution Image
Convolve an image with an arbitrary filter mask.

ConvolImage

Cooccurrence Matrix
Calculate gray value features from a co-occurrence matrix.

ConvolImage

Copy Image
Copy an image and allocate new memory for it.

Copy Image

Corner Response
Searching corners in images.

Corner Response

Count Channels
Count channels of image.

Count Channels

Create BgEsti
Generate and initialize a data set for the background estimation.

Create BgEsti

Create Scaled Shape Model
Prepare a shape model for scale invariant matching.

Create Scaled Shape Model

Create Shape Model
Prepare a shape model for matching.

Create Shape Model

Create Template
Preparing a pattern for template matching.

Create Template

Create Template Rot
Preparing a pattern for template matching with rotation.

Create Template Rot

Crop Domain
Cut out of defined gray values.

Crop Domain

Crop Domain Rel
Cut out an image area relative to the domain.

Crop Domain Rel

Crop Part
Cut out a rectangular image area.

Crop Part

Crop Rectangle
Cut out a rectangular image area.

Crop Rectangle

Decompose 2
Convert a two-channel image into two images.

Decompose 2

Decompose 3
Convert a three-channel image into three images.

Decompose 3

Decompose 4
Convert a four-channel image into four images.

Decompose 4

Decompose 5
Convert a five-channel image into five images.

Decompose 5

Decompose 6
Convert a six-channel image into six images.

Decompose 6

Decompose 7
Convert a seven-channel image into seven images.

Decompose 7

Depth From Focus
Extract depth using multiple focus levels.

Depth From Focus

Derivative Gauss
Convolve an image with derivatives of the Gaussian.

Derivative Gauss

Detect Edge Segments
Detect straight edge segments.

Detect Edge Segments

Deviation Image
Calculate the standard deviation of gray values within rectangular windows.

Deviation Image

DiffOf Gauss
Approximate the LoG operator (Laplace of Gaussian).

DiffOf Gauss

Disp Channel
Displays images with several channels.

Disp Channel

HALCON 6.1.4
Expand a region starting at a given line.

Expand the domain of an image and set the gray values in the expanded domain.

Divide two images.

Verification of a pattern using an OCV tool.

Opening, Median and Closing with circle or rectangle mask.

Threshold operator for signed images.

Approximate an affine map from a displacement vector field.

Convert a displacement vector field into two int1-images.

Segment an image using a local threshold.

Extract edges using Deriche, Lanser, Shen, or Canny filters.

Extract sub-pixel precise edges using Deriche, Lanser, Shen, or Canny filters.

Smooth an image in the spatial domain to suppress noise.

Replace values outside of thresholds with average value.

Enhance contrast of the image.

Calculate the energy of a two-channel image.

Calculate the entropy of gray values within a rectangular window.

Histogram linearisation of images

Estimate the albedo of a surface and the amount of ambient light.

Estimate the slant of a light source and the albedo of a surface.

Estimate the slant of a light source and the albedo of a surface.

Estimate the tilt of a light source.

Estimate the tilt of a light source.

Matching of a template and an image.

Matching a template and an image in a resolution pyramid.

Expand the domain of an image and set the gray values in the expanded domain.

Expand a region starting at a given line.
FastMatch
Searching all good matches of a template and an image.

FastMatchMg
Searching all good grayvalue matches in a pyramid.

FastThreshold
Fast selection of grayvalues within a given gray interval.

FuzzyMeasurePairing
Extract straight edge pairs perpendicular to a rectangle or an annular arc.

FuzzyMeasurePairs
Extract straight edge pairs perpendicular to a rectangle or an annular arc.

FuzzyMeasurePos
Extract straight edges perpendicular to a rectangle or an annular arc.

GaussImage
Smooth using discrete gauss functions.

GenGaussPyramid
Calculating a Gauss pyramid.
CHAPTER 16. CLASS HIERARCHY

- **GetImageProto** .......................................................... P.310
  Set the gray values of an image to a specified value.

- **Get1DBarCode** .......................................................... P.729
  Extract the widths of the elements inside a bar code region.

- **Get2DBarCode** .......................................................... P.730
  Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).

- **Get2DBarCodePos** ...................................................... P.734
  Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.

- **GetDomain** .............................................................. P.319
  Get the domain of an image.

- **GetGrayVal** .............................................................. P.285
  Access the gray values of an image object.

- **GetImagePointer1** ..................................................... P.286
  Access the pointer of a channel.

- **GetImagePointer1Rect** ............................................... P.287
  Access to the image data pointer and the image data inside the smallest rectangle of the domain of the input image.

- **GetImagePointer3** ..................................................... P.288
  Access the pointers of a colored image.

- **GetImageTime** .......................................................... P.289
  Request time at which the image was created.

- **GnuplotPlotImage** ..................................................... P.190
  Visualize images using gnuplot.

- **GrayBothat** ............................................................. P.429
  Perform a gray value bottom hat transformation on an image.

- **GrayClosing** ............................................................ P.430
  Perform a gray value closing on an image.

- **GrayClosingShape** .................................................... P.431
  Perform a gray value closing with a selected mask.

- **GrayDilation** ............................................................ P.432
  Perform a gray value dilation on an image.

- **GrayDilationRect** ..................................................... P.433
  Determine the maximum gray value within a rectangle.

- **GrayDilationShape** .................................................... P.433
  Determine the maximum gray value within a selected mask.

- **GrayErosion** ............................................................ P.434
  Perform a gray value erosion on an image.

- **GrayErosionRect** ..................................................... P.435
  Determine the minimum gray value within a rectangle.

- **GrayErosionShape** .................................................... P.436
  Determine the minimum gray value within a selected mask.

- **GrayInside** ............................................................. P.124
  Calculate the lowest possible gray value on an arbitrary path to the image border for each point in the image.

- **GrayOpening** .......................................................... P.437
  Perform a gray value opening on an image.

- **GrayOpeningShape** ................................................... P.438
  Perform a gray value opening with a selected mask.

- **GrayProjections** ..................................................... P.334
  Calculate horizontal and vertical gray-value projections.
GrayRangeRect
Determine the gray value range within a rectangle.

GraySkeleton
Thinning of gray value images.

GrayTophat
Perform a gray value top hat transformation on an image.

HighpassImage
Extract high frequency components from an image.

HoughLineTransDir
Compute the Hough transform for lines using local gradient direction.

HoughLinesDir
Detect lines in edge images with the help of the Hough transform using local gradient direction and return them in normal form.

HysteresisThreshold
Perform a hysteresis threshold operation on an image.

Illuminate
Illuminate image.

ImageToChannels
Convert a multichannel image into One-channel images

ImageToWorldPlane
Rectify an image by transforming it into the plane z=0 of a world coordinate system.

InspectShapeModel
Create the representation of a shape model.

Int1ToDvf
Convert two int1-images into a displacement vector field.

InvertImage
Invert an image.

KirschAmp
Detect edges (amplitude) using the Kirsch operator.

KirschDir
Detect edges (amplitude and direction) using the Kirsch operator.

LabelToRegion
Extract regions with equal gray values from an image.

Laplace
Calculate the Laplace operator by using finite differences.

LaplaceOfGauss
LoG-Operator (Laplace of Gaussian).

LinesFacet
Detection of lines using the facet model.

LinesGauss
Detect lines and their width.

LocalMax
Detect all local maxima in an image.

LutTrans
Transform an image with a gray-value look-up-table

MapImage
Apply a general transformation to an image.

MaxImage
Calculate the maximum of two images pixel by pixel.

MeanImage
Smooth by averaging.
Transform an image to polar coordinates
Detect the centers of all gray value plateaus.

Reconstruct a surface from at least three gray value images.
Return the phase of a complex image in radians.
Return the phase of a complex image in degrees.

Calculating the Displacement vector field by correlation methods.
Access the features which correspond to a character.
Suppress non-maximum points on an edge.
Suppress non-maximum points on an edge.
Multiply two images.

Suppress non-maximum points on an edge.

Calculate the average of maximum and minimum inside any mask.
Calculate the minimum of two images pixel by pixel.
Mirror an image.
Calculating the monotony operation.

Median filtering with different rank masks.
Separated median filtering with rectangle masks.
Weighted median filtering with different rank masks.

Calculate the average of maximum and minimum inside any mask.
Calculate the minimum of two images pixel by pixel.
Mirror an image.
Calculating the monotony operation.

Multiply two images.

Suppress non-maximum points on an edge.

Suppress non-maximum points on an edge.

Access the features which correspond to a character.
Calculating the Displacement vector field by correlation methods.

Paint the gray values of an image into another image.
Return the phase of a complex image in degrees.
Return the phase of a complex image in radians.
Reconstruct a surface from at least three gray value images.
Detect all gray value plateaus.
Detect the centers of all gray value plateaus.
Transform an image to polar coordinates
Segment an image by "pouring water" over it.
- **PowerByte**
  Return the power spectrum of a complex image.
- **PowerLn**
  Return the power spectrum of a complex image.
- **PowerReal**
  Return the power spectrum of a complex image.
- **PrewittAmp**
  Detect edges (amplitude) using the Prewitt operator.
- **PrewittDir**
  Detect edges (amplitude and direction) using the Prewitt operator.
- **PrincipalComp**
  Compute the principal components of multi-channel images.
- **RankImage**
  Smooth an image with an arbitrary rank mask.
- **RealToComplex**
  Convert two real images into a complex image.
- **RectangleDomain**
  Reduce the domain of an image to a rectangle.
- **ReduceDomain**
  Reduce the domain of an image.
- **Regiongrowing**
  Segment an image using regiongrowing.
- **RegiongrowingMean**
  Regiongrowing using mean gray values.
- **RegiongrowingN**
  Regiongrowing for multi-channel images.
- **Rgb1ToGray**
  Transform an RGB image into a gray scale image.
- **Rgb3ToGray**
  Transform an RGB image to a gray scale image.
- **Roberts**
  Detect edges using the Roberts filter.
- **RobinsonAmp**
  Detect edges (amplitude) using the Robinson operator.
- **RobinsonDir**
  Detect edges (amplitude and direction) using the Robinson operator.
- **RotateImage**
  Rotate an image about its center.
- **RunBgEsti**
  Estimate the background and return the foreground region.
- **ScaleImage**
  Scale the gray values of an image.
- **ScaleImageMax**
  Maximum gray value spreading in the value range 0 bis 255.
- **SelectGrayvaluesFromChannels**
  Selection of gray values of a multi-channel image using an index image.
- **SendImage**
  Send an image over a socket connection.
- **SetGrayval**
  Set single gray values in an image.
- **SfsModLr**
  Reconstruct a surface from a gray value image.
Image restoration by Wiener filtering.

Extract watersheds and basins from an image.

Transform an image from an arbitrary color space to the RGB color space.

Transform an image from the RGB color space to an arbitrary color space.

Train a variation model.

Training of an OCV tool.

Transform an image from the RGB color space to an arbitrary color space.

Transform an image from an arbitrary color space to the RGB color space.

Smooth an image with an arbitrary rank mask.

Change the estimated background image.

Extract watersheds and basins from an image.

Image restoration by Wiener filtering.
16.13 HInfoX

Group class for system information related functionality.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Methods

- **DispInfo**
  - Display the manual information of a procedure on the screen.

- **GetChapterInfo**
  - Get information concerning the chapters on procedures.

- **GetComprise**
  - Get the output treatment of an image matrix.

- **GetKeywords**
  - Get keywords which are assigned to procedures.

- **GetOperatorInfo**
  - Get information concerning a HALCON-procedure.

- **GetOperatorName**
  - Get procedures with the given string as a substring of their name.

- **GetParamInfo**
  - Get information concerning the procedure parameters.

- **GetParamNames**
  - Get the names of the parameters of a HALCON-procedure.

- **GetParamNum**
  - Get number of the different parameter classes of a HALCON-procedure.

- **GetParamTypes**
  - Get default data type for the control parameters of a HALCON-procedure.

- **InfoFramegrabber**
  - Information about the specified frame grabber.
16.14 HMeasureX

Represents an instance of a tool to measure distances.

---

Class Description

Number of Methods ................................................................. 13
Number of Properties ............................................................ 0
Abstract Class ................................................................. no
Base Class(es) ................................................................... <none>
Automatic Destructor .................................................. yes, through CloseMeasure (not member of this class!)
Object Identity (through internal handle) ......................... yes

---

Constructor(s)

- GenMeasureArc .......................................................... P.850
  Prepare the extraction of straight edges perpendicular to an annular arc.
- GenMeasureRectangle2 ................................................ P.852
  Prepare the extraction of straight edges perpendicular to a rectangle.

---

Methods

- FuzzyMeasurePairing .................................................. P.843
  Extract straight edge pairs perpendicular to a rectangle or an annular arc.
- FuzzyMeasurePairs .................................................... P.846
  Extract straight edge pairs perpendicular to a rectangle or an annular arc.
- FuzzyMeasurePos .......................................................... P.848
  Extract straight edges perpendicular to a rectangle or an annular arc.
- MeasurePairs ............................................................. P.854
  Extract straight edge pairs perpendicular to a rectangle or annular arc.
- MeasurePos ................................................................. P.856
  Extract straight edges perpendicular to a rectangle or annular arc.
- MeasureProjection ......................................................... P.858
  Extract a gray value profile perpendicular to a rectangle or annular arc.
- MeasureThresh .............................................................. P.859
  Extracting points with a particular grey value along a rectangle or an annular arc.
- ResetFuzzyMeasure ......................................................... P.860
  Reset a fuzzy member function.
16.15  HMiscX

Group class for methods belonging to no other HALCON class.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ AbsInvarFourierCoeff</td>
</tr>
<tr>
<td>Normalizing of the Fourier coefficients with respect to the displacement of the starting point.</td>
</tr>
<tr>
<td>▶ AngleLl</td>
</tr>
<tr>
<td>Calculate the angle between two lines.</td>
</tr>
<tr>
<td>▶ AngleLx</td>
</tr>
<tr>
<td>Calculate the angle between one line and the vertical axis.</td>
</tr>
<tr>
<td>▶ ApproxChain</td>
</tr>
<tr>
<td>Approximate a contour by arcs and lines.</td>
</tr>
<tr>
<td>▶ ApproxChainSimple</td>
</tr>
<tr>
<td>Approximate a contour by arcs and lines.</td>
</tr>
<tr>
<td>▶ CaltabPoints</td>
</tr>
<tr>
<td>Read the mark center points from the calibration plate description file.</td>
</tr>
<tr>
<td>▶ ChangeRadialDistortionCamPar</td>
</tr>
<tr>
<td>Determine new camera parameters in accordance to the specified radial distortion.</td>
</tr>
<tr>
<td>▶ ClearAllShapeModels</td>
</tr>
<tr>
<td>Free the memory of all shape models.</td>
</tr>
<tr>
<td>▶ ClearAllTemplates</td>
</tr>
<tr>
<td>Deallocation of the memory of all templates.</td>
</tr>
<tr>
<td>▶ ClearAllVariationModels</td>
</tr>
<tr>
<td>Free the memory of all variation models.</td>
</tr>
<tr>
<td>▶ CloseAllBgEsti</td>
</tr>
<tr>
<td>Delete all background estimation data sets.</td>
</tr>
<tr>
<td>▶ CloseAllClassBox</td>
</tr>
<tr>
<td>Destroy all classifiers.</td>
</tr>
<tr>
<td>▶ CloseAllFiles</td>
</tr>
<tr>
<td>Close all open files.</td>
</tr>
<tr>
<td>▶ CloseAllFramegrabbers</td>
</tr>
<tr>
<td>Close all frame grabbers.</td>
</tr>
<tr>
<td>▶ CloseAllMeasures</td>
</tr>
<tr>
<td>Delete all measure objects.</td>
</tr>
</tbody>
</table>
CHAPTER 16. CLASS HIERARCHY

- **CloseAllOcrs**
  Destroy all OCR classifiers.

- **CloseAllOcvs**
  Clear all OCV tools.

- **CloseAllSerials**
  Close all serial devices.

- **ConcatOcrTrainf**
  Concat training files.

- **CreateCaltab**
  Generate a calibration plate description file and a corresponding PostScript file.

- **DistancePl**
  Calculate the distance between one point and one line.

- **DistanceEp**
  Calculate the distance between two points.

- **DistancePs**
  Calculate the distances between a point and a line segment.

- **DistanceSl**
  Calculate the distances between one line segment and one line.

- **DistanceSs**
  Calculate the distances between two line segments.

- **FileExists**
  Check whether file exists.

- **FilterKalman**
  Estimate the current state of a system with the help of the Kalman filtering.

- **Fourier1Dim**
  Calculate the Fourier coefficients of a parameterized contour.

- **Fourier1DimInv**
  One dimensional Fourier synthesis (inverse Fourier transform).

- **GaussDistribution**
  Generate a Gaussian noise distribution.

- **GetLineOfSight**
  Compute the line of sight corresponding to a point in the image.

- **GetPointsEllipse**
  Points of an ellipse corresponding to specific angles.

- **ImagePointsToWorldPlane**
  Transform image points into the plane z=0 of a world coordinate system.

- **InfoEdges**
  Information on smoothing filter SmoothImage.

- **InfoSmooth**
  Information on smoothing filter SmoothImage.

- **IntersectionLl**
  Calculate the intersection point of two lines.

- **InvarFourierCoeff**
  Normalize the Fourier coefficients.

- **LineOrientation**
  Calculate the orientation of lines.

- **LinePosition**
  Calculate the center of gravity, length, and orientation of a line.

- **MatchFourierCoeff**
  Similarity of two contours.

- **MoveContourOrig**
  Transformation of the origin into the center of gravity.
16.16  HObjectX

Instance of an iconic object. Base class for HImage, HRegion,...

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity</td>
</tr>
<tr>
<td>Constructor(s)</td>
</tr>
</tbody>
</table>

- **PartitionLines**
  Partition lines according to various criteria.

- **PrepContourFourier**
  Parameterize the passed contour.

- **Project3DPoint**
  Project 3D points into (sub-)pixel image coordinates.

- **ProjectionP1**
  Calculate the projection of a point onto a line.

- **ReadCamPar**
  Read the interior camera parameters from text file.

- **ReadKalman**
  Read the description file of a Kalman filter.

- **ReadOcrTrainfNames**
  Query which characters are stored in a training file.

- **ReadTuple**
  Read a tuple from a file.

- **SelectLines**
  Select lines according to various criteria.

- **SelectLinesLongest**
  Select the longest input lines.

- **SensorKalman**
  Interactive input of measurement values for a Kalman filtering.

- **SpDistribution**
  Generate a salt-and-pepper noise distribution.

- **UpdateKalman**
  Read an update file of a Kalman filter.

- **WriteCamPar**
  Write the interior camera parameters to text file.

- **WriteTuple**
  Write a tuple to a file.
**CHAPTER 16. CLASS HIERARCHY**

Methods

- **ConcatObj**
  Concatenate two iconic object tuples.

- **CopyObj**
  Copy an iconic object in the HALCON database.

- **CountObj**
  Number of objects in a tuple.

- **DispObj**
  Displays image objects (image, region, XLD).

- **GetChannelInfo**
  Informations about the components of an image object.

- **GetObjClass**
  Name of the class of an image object.

- **ObjToInteger**
  Convert an iconic object into an “integer number.”

- **SelectObj**
  Select an object from an object tuple.

- **TestEqualObj**
  Compare image objects regarding equality.

- **TestObjDef**
  Test whether an object is already deleted.

16.17 **HOcrX**

*Instance of an OCR classifier.*

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>11</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through CloseOcr (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>yes</td>
</tr>
</tbody>
</table>

Constructor(s)

- **CreateOcrClassBox**
  Create a new OCR-classifier.

- **ReadOcr**
  Read an OCR classifier from a file.

Methods

- **DoOcrMulti**
  Classify characters.

- **DoOcrSingle**
  Classify one character.

- **InfoOcrClassBox**
  Get information about an OCR classifier.

- **OcrChangeChar**
  Define a new conversion table for the characters.

- **OcrGetFeatures**
  Access the features which correspond to a character.
16.18 HOCvX

Represents an instance of a tool for optical character verification.

Class Description

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>Number of Properties</th>
<th>Abstract Class</th>
<th>Base Class(es)</th>
<th>Automatic Destructor</th>
<th>Object Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>no</td>
<td>&lt;none&gt;</td>
<td>yes through CloseOcv</td>
<td>yes</td>
</tr>
</tbody>
</table>

Constructor(s)

- **CreateOcvProj**
  Create a new OCV tool based on gray value projections.

- **ReadOcv**
  Reading an OCV tool from file.

Methods

- **DoOcvSimple**
  Verification of a pattern using an OCV tool.

- **TrainOcvProj**
  Training of an OCV tool.

- **WriteOcv**
  Saving an OCV tool to file.

16.19 HOperatorSetX

Group class for the handling of operators.

Class Description

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>Number of Properties</th>
<th>Abstract Class</th>
<th>Base Class(es)</th>
<th>Automatic Destructor</th>
<th>Object Identity</th>
</tr>
</thead>
<tbody>
<tr>
<td>984</td>
<td>0</td>
<td>no</td>
<td>&lt;none&gt;</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

HALCON 6.1.4
## Methods

- **AbsFunct1D** .......................................................... P.790
  
  Absolute value of the y values.

- **AbsImage** .......................................................... P.37
  
  Calculate the absolute value (modulus) of an image.

- **AbsInvarFourierCoeff** ........................................... P.785
  
  Normalizing of the Fourier coefficients with respect to the displacement of the starting point.

- **AccessChannel** .................................................. P.290
  
  Access a channel of a multichannel image.

- **AdaptTemplate** .................................................. P.385
  
  Adapting a template to the size of an image.

- **AddChannels** ................................................... P.317
  
  Add gray values to regions.

- **AddImage** .......................................................... P.38
  
  Add two images.

- **AddNoiseDistribution** .......................................... P.130
  
  Add noise to an image.

- **AddNoiseWhite** .................................................. P.131
  
  Add noise to an image.

- **AddNoiseWhiteContourXld** ..................................... P.978
  
  Add noise to XLD contours.

- **AffineTransContourXld** ........................................ P.979
  
  Apply an arbitrary affine 2D transformation to XLD contours.

- **AffineTransImage** ............................................... P.29
  
  Apply an arbitrary affine 2D transformation to images.

- **AffineTransImageSize** ......................................... P.31
  
  Apply an arbitrary affine 2D transformation to an image and specify the output image size.

- **AffineTransPoint2D** ............................................. P.683
  
  Apply an arbitrary affine 2D transformation to points.

- **AffineTransPoint3D** ............................................. P.684
  
  Apply an arbitrary affine 3D transformation to points.

- **AffineTransPolygonXld** ....................................... P.979
  
  Apply an arbitrary affine transformation to XLD polygons.

- **AffineTransRegion** ............................................. P.508
  
  Apply an arbitrary affine 2D transformation to regions.

- **AngleLl** .......................................................... P.801
  
  Calculate the angle between two lines.

- **AngleLx** .......................................................... P.802
  
  Calculate the angle between one line and the vertical axis.

- **AnisotropeDiff** ................................................ P.135
  
  Smooth an image by edge-preserving anisotropic diffusion.

- **AppendChannel** .................................................. P.290
  
  Append additional matrices (channels) to the image.

- **AppendOcrTrainf** ............................................... P.865
  
  Add characters to a training file.

- **ApproxChain** ................................................... P.373
  
  Approximate a contour by arcs and lines.

- **ApproxChainSimple** ............................................ P.376
  
  Approximate a contour by arcs and lines.

- **AreaCenter** ..................................................... P.532
  
  Area and center of regions.
AreaCenterGray ................................................................. P.321
  Compute the area and center of gravity of a region in a gray value image.

AreaCenterXld ............................................................... P.948
  Area and center of gravity (centroid) of contours and polygons.

AutoThreshold ............................................................. P.597
  Segment an image using thresholds determined from its histogram.

BackgroundSeg ............................................................ P.574
  Determine the connected components of the background of given regions.

BandpassImage ............................................................ P.109
  Edge extraction using bandpass filters.

BestMatch ................................................................. P.386
  Searching the best matching of a template and an image.

BestMatchMg .............................................................. P.387
  Searching the best grayvalue matches in a pyramid.

BestMatchPreMg .......................................................... P.388
  Searching the best grayvalue matches in a pre generated pyramid.

BestMatchRot ............................................................ P.390
  Searching the best matching of a template and an image with rotation.

BestMatchRotMg .......................................................... P.391
  Searching the best matching of a template and a pyramid with rotation.

BinThreshold ............................................................. P.598
  Segment a black-and-white image using an automatically determined threshold.

BitAnd ................................................................. P.46
  Bit-by-bit AND of all pixels of the input images.

BitLshift ............................................................... P.47
  Left shift of all pixels of the image.

BitMask ................................................................. P.48
  Logical “AND” of each pixel using a bit mask.

BitNot ................................................................. P.49
  Complement all bits of the pixels.

BitOr ................................................................. P.49
  Bit-by-bit OR of all pixels of the input images.

BitRshift ............................................................. P.50
  Right shift of all pixels of the image.

BitSlice ............................................................. P.51
  Extract a bit from the pixels.

BitXor ............................................................. P.52
  Bit-by-bit XOR of all pixels of the input images.

BottomHat ............................................................ P.441
  Compute the bottom hat of regions.

Boundary ............................................................. P.442
  Reduce a region to its boundary.

CaltabPoints .......................................................... P.735
  Read the mark center points from the calibration plate description file.

CameraCalibration ........................................................ P.737
  Determine all camera parameters by a simultaneous minimization process.

CfaToRgb ............................................................... P.53
  Convert a single-channel color filter array image into an RGB image.

ChangeDomain .......................................................... P.318
  Change definition domain of an image.

ChangeFormat .......................................................... P.344
  Change image size.
ChangeRadialDistortionCamPar

Determine new camera parameters in accordance to the specified radial distortion.

ChangeRadialDistortionContoursXld

Change the radial distortion of contours.

ChangeRadialDistortionImage

Change the radial distortion of an image.

ChannelsToImage

Convert one-channel images into a multichannel image.

CharThreshold

Perform a threshold segmentation for extracting characters.

CheckDifference

Compare two images pixel by pixel.

CheckParHwPotential

Check hardware regarding its potential for parallel processing.

Circularity

Shape factor for the circularity (similarity to a circle) of a region.

CircularityXld

Shape factor for the circularity (similarity to a circle) of a contour.

Class2DimSup

Segment an image using two-dimensional pixel classification.

Class2DimUnsup

Segment two images by clustering.

ClassNdimBox

Classify pixels using hyper-cuboids.

ClassNdimNorm

Classify pixels using hyper-spheres or hyper-cubes.

ClearAllShapeModels

Free the memory of all shape models.

ClearAllTemplates

Deallocation of the memory of all templates.

ClearAllVariationModels

Free the memory of all variation models.

ClearObj

Delete an iconic object from the HALCON database.

ClearRectangle

Delete a rectangle on the output window.

ClearSampset

Free memory of a data set.

ClearSerial

Clear the buffer of a serial connection.

ClearShapeModel

Free the memory of a shape model.

ClearTemplate

Deallocation of the memory of a template.

ClearVariationModel

Free the memory of a variation model.

ClearWindow

Delete an output window.

ClipContoursXld

Clip an XLD contour.

ClipRegion

Clip a region to a rectangle.
ClipRegionRel ................................................................. P.576
Clip a region relative to its size.

CloseAllBgEsti .............................................................. P.704
Delete all background estimation data sets.

CloseAllClassBox .......................................................... P.1
Destroy all classifiers.

CloseAllFiles ............................................................... P.18
Close all open files.

CloseAllFramegrabbers .................................................... P.352
Close all frame grabbers.

CloseAllMeasures .......................................................... P.842
Delete all measure objects.

CloseAllOcrs ............................................................ P.866
Destroy all OCR classificators.

CloseAllOcvs ............................................................. P.881
Clear all OCV tools.

CloseAllSerials ............................................................. P.668
Close all serial devices.

CloseBgEsti ............................................................. P.704
Delete the background estimation data set.

CloseClassBox ............................................................ P.2
Destroy the classificator.

CloseEdges ............................................................... P.61
Close edge gaps using the edge amplitude image.

CloseEdgesLength ........................................................ P.62
Close edge gaps using the edge amplitude image.

CloseFile ................................................................. P.18
Closing a text file.

CloseFramegrabber ......................................................... P.352
Close specified frame grabber.

CloseMeasure ............................................................ P.843
Delete a measure object.

CloseOcr ................................................................. P.866
Deallocation of the memory of an OCR classifier.

CloseOcv ................................................................. P.881
Clear an OCV tool.

CloseSerial .............................................................. P.668
Close a serial device.

CloseSocket .............................................................. P.673
Close a socket.

CloseWindow ............................................................. P.261
Close an output window.

Closing ................................................................. P.443
Close a region.

ClosingCircle ............................................................ P.444
Close a region with a circular structuring element.

ClosingGolay ............................................................ P.445
Close a region with an element from the Golay alphabet.

ClosingRectangle ......................................................... P.446
Close a region with a rectangular structuring element.

CombineRoadsXld ......................................................... P.981
Combine road hypotheses from two resolution levels.
- **Compactness**
  Shape factor for the compactness of a region.

- **CompactnessXld**
  Shape factor for the compactness of a contour.

- **CompareVariationModel**
  Compare an image to a variation model.

- **Complement**
  Return the complement of a region.

- **ComplexToReal**
  Convert a complex image into two real images.

- **Compose2**
  Convert two images into a two-channel image.

- **Compose3**
  Convert 3 images into a three-channel image.

- **Compose4**
  Convert 4 images into a four-channel image.

- **Compose5**
  Convert 5 images into a five-channel image.

- **Compose6**
  Convert 6 images into a six-channel image.

- **Compose7**
  Convert 7 images into a seven-channel image.

- **ConcatObj**
  Concatenate two iconic object tuples.

- **ConcatOcrTrainf**
  Concat training files.

- **ConnectAndHoles**
  Number of connection components and holes

- **Connection**
  Compute connected components of a region.

- **Contlength**
  Contour length of a region.

- **ContourPointNumXld**
  Return the number of points in an XLD contour.

- **ContourToWorldPlaneXld**
  Transform an XLD contour into the plane z=0 of a world coordinate system.

- **ConvertImageType**
  Convert the type of an image.

- **ConvertPoseType**
  Change the representation type of a 3D pose.

- **Convexity**
  Shape factor for the convexity of a region.

- **ConvexityXld**
  Shape factor for the convexity of a contour.

- **ConvolFft**
  Convolve an image with a byte-mask in the frequency domain.

- **ConvolGabor**
  Convolve an image with a Gabor filter in the frequency domain.

- **ConvolImage**
  Convolve an image with an arbitrary filter mask.

- **CocFeatureImage**
  Calculate a co-occurrence matrix and derive gray value features thereof.
Cut out a rectangular image area.
Cut out of defined gray values.
Create a variation model for image comparison.
Preparing a pattern for template matching.
Prepare a shape model for scale invariant matching.
Create a shape model for matching.
Preparing a pattern for template matching with rotation.
Create a variation model for image comparison.
Cut out of defined gray values.
Cut out an image area relative to the domain.
Cut out a rectangular image area.
Cut out a rectangular image area.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decode1DBarCode</td>
<td>Decoding of a sequence of elements of a bar code.</td>
</tr>
<tr>
<td>Decode2DBarCode</td>
<td>Decode 2D bar code data.</td>
</tr>
<tr>
<td>Decompose2</td>
<td>Convert a two-channel image into two images.</td>
</tr>
<tr>
<td>Decompose3</td>
<td>Convert a three-channel image into three images.</td>
</tr>
<tr>
<td>Decompose4</td>
<td>Convert a four-channel image into four images.</td>
</tr>
<tr>
<td>Decompose5</td>
<td>Convert a five-channel image into five images.</td>
</tr>
<tr>
<td>Decompose6</td>
<td>Convert a six-channel image into six images.</td>
</tr>
<tr>
<td>Decompose7</td>
<td>Convert a seven-channel image into seven images.</td>
</tr>
<tr>
<td>DepthFromFocus</td>
<td>Extract depth using multiple focus levels.</td>
</tr>
<tr>
<td>DerivateGauss</td>
<td>Convolve an image with derivatives of the Gaussian.</td>
</tr>
<tr>
<td>DescriptClassBox</td>
<td>Description of the classifier.</td>
</tr>
<tr>
<td>DetectEdgeSegments</td>
<td>Detect straight edge segments.</td>
</tr>
<tr>
<td>DeviationImage</td>
<td>Calculate the standard deviation of gray values within rectangular windows.</td>
</tr>
<tr>
<td>DiameterRegion</td>
<td>Maximal distance between two boundary points of a region.</td>
</tr>
<tr>
<td>DiameterXld</td>
<td>Maximum distance between two contour points.</td>
</tr>
<tr>
<td>DiffOfGauss</td>
<td>Approximate the LoG operator (Laplace of Gaussian).</td>
</tr>
<tr>
<td>Difference</td>
<td>Calculate the difference of two regions.</td>
</tr>
<tr>
<td>Dilation1</td>
<td>Dilate a region.</td>
</tr>
<tr>
<td>Dilation2</td>
<td>Dilate a region (using a reference point).</td>
</tr>
<tr>
<td>DilationCircle</td>
<td>Dilate a region with a circular structuring element.</td>
</tr>
<tr>
<td>DilationGolay</td>
<td>Dilate a region with an element from the Golay alphabet.</td>
</tr>
<tr>
<td>DilationRectangle1</td>
<td>Dilate a region with a rectangular structuring element.</td>
</tr>
<tr>
<td>DilationSeq</td>
<td>Dilate a region sequentially.</td>
</tr>
<tr>
<td>Discretel1DBarCode</td>
<td>Generate a discrete bar code from the elements widths.</td>
</tr>
<tr>
<td>DispArc</td>
<td>Displays circular arcs in a window.</td>
</tr>
<tr>
<td>DispArrow</td>
<td>Displays arrows in a window.</td>
</tr>
</tbody>
</table>
Distance between one point and one region.
Distance between two points.
Distance between one point and one line.
Distance between one line and one region.
Distance of two functions.
Distance of contours to an ellipse.
Display an XLD object.
Display the manual information of a procedure on the screen.
Draws lines in a window.
Graphical view of the look-up-table (lut).
Displays image objects (image, region, XLD).
Displays a polyline.
Display of rectangles aligned to the coordinate axes.
Displays arbitrarily oriented rectangles.
Displays regions in a window.
Display of rectangles aligned to the coordinate axes.
Displays regions in a window.
Distance between one line and one region.
Distance between one point and one line.
Distance between two points.
Distance between one point and one region.
Distance between a point and a line segment.
Minimum distance between the contour pixels of two regions each.
Minimum distance between two regions with the help of dilatation.
Calculate the distances between one line segment and one line.
Interactive drawing of a closed region.

Interactive drawing of any orientated rectangle.

Interactive drawing of any orientated rectangle.

Draw a rectangle parallel to the coordinate axis.

Interactive drawing of a polygon row.

Draw a point.

Interactive drawing of a polygon row.

Interactive drawing of a circle.

Interactive movement of a region with restriction of positions.

Interactive movement of a region with fixpoint specification.

Interactive movement of a region.

Divide two images.

Classify characters.

Classify one character.

Verification of a pattern using an OCV tool.

Interactive moving of a region.

Interactive movement of a region with restriction of positions.

Interactive movement of a region with fixpoint specification.

Interactive drawing of a circle.

Interactive drawing of a circle.

Interactive drawing of an ellipse.

Interactive drawing of an ellipse.

Interactive drawing of an ellipse.

Draw a line.

Draw a line.

Manipulate look-up-table (lut) interactively.

Draw a point.

Draw a point.

Interactive drawing of a polygon row.

Draw a rectangle parallel to the coordinate axis.

Draw a rectangle parallel to the coordinate axis.

Interactive drawing of any orientated rectangle.

Interactive drawing of any orientated rectangle.

Interactive drawing of a closed region.

Opening, Median and Closing with circle or rectangle mask.
- DualThreshold ........................................ P.606
  Threshold operator for signed images.
- DumpWindow ......................................... P.263
  Write the window content to a file.
- DvfToHomMat2d ...................................... P.685
  Approximate an affine map from a displacement vector field.
- DvfToInt1 ........................................... P.370
  Convert a displacement vector field into two int1-images.
- DynThreshold ....................................... P.608
  Segment an image using a local threshold.
- Eccentricity ........................................ P.538
  Shape features derived from the ellipse parameters.
- EccentricityXld ..................................... P.954
  Shape features derived from the ellipse parameters.
- EdgesImage .......................................... P.67
  Extract edges using Deriche, Lanser, Shen, or Canny filters.
- EdgesSubPix ......................................... P.70
  Extract sub-pixel precise edges using Deriche, Lanser, Shen, or Canny filters.
- EliminateMinMax ..................................... P.136
  Smooth an image in the spatial domain to suppress noise.
- EliminateRuns ....................................... P.580
  Eliminate runs of a given length.
- EliminateSp ......................................... P.138
  Replace values outside of thresholds with average value.
- EllipticAxis ......................................... P.539
  Parameters of the equivalent ellipse.
- EllipticAxisGray .................................... P.324
  Compute the orientation and major axes of a region in a gray value image.
- EllipticAxisXld ...................................... P.955
  Parameters of the equivalent ellipse.
- Emphasize .......................................... P.90
  Enhance contrast of the image.
- EnergyGabor ......................................... P.95
  Calculate the energy of a two-channel image.
- EnquireClassBox ..................................... P.3
  Classify a tuple of attributes.
- EnquireRejectClassBox .............................. P.4
  Classify a tuple of attributes with rejection class.
- EntropyGray ......................................... P.325
  Determine the entropy and anisotropy of images.
- EntropyImage ........................................ P.156
  Calculate the entropy of gray values within a rectangular window.
- EquHistImage ....................................... P.91
  Histogram linearisation of images
- Erosion1 ............................................ P.454
  Erode a region.
- Erosion2 ............................................ P.456
  Erode a region (using a reference point).
- ErosionCircle ....................................... P.457
  Erode a region with a circular structuring element.
- ErosionGolay ....................................... P.458
  Erode a region with an element from the Golay alphabet.
 Fill up holes in regions having given shape features.

ErosionRectangle1
Erode a region with a rectangular structuring element.

ErosionSeq
Erode a region sequentially.

EstimateAlAm
Estimate the albedo of a surface and the amount of ambient light.

EstimateSlAlZc
Estimate the slant of a light source and the albedo of a surface.

FastMatchMg
Searching all good matches of a template and an image.

FastThreshold
Fast selection of grayvalues within a given gray interval.

FftGeneric
Compute the fast Fourier transform of an image.

FftImage
Compute the fast Fourier transform of an image.

FftImageInv
Compute the inverse fast Fourier transform of an image.

FileExists
Check whether file exists.

FillDvf
Expansion of a displacement vector field onto undefined areas.

FillInterlace
Interpolate 2 video half images.

FillUp
Fill up holes in regions.

FillUpShape
Fill up holes in regions having given shape features.
Detect edges (amplitude and direction) using the Frei-Chen operator.

Detect edges (amplitude) using the Frei-Chen operator.

One dimensional Fourier synthesis (inverse Fourier transform).

Calculate the Fourier coefficients of a parameterized contour.

Approximation of XLD contours by circles.

Approximation of XLD contours by ellipses or elliptic arcs.

Approximation of XLD contours by line segments.

Calculate gray value moments and approximation by a first order surface (plane).

Calculate gray value moments and approximation by a second order surface.

Perform a closing after an opening with multiple structuring elements.

Create a line feed.

Calculate the Fourier coefficients of a parameterized contour.

One dimensional Fourier synthesis (inverse Fourier transform).

Read a character from a text file.

Read strings from a text file.

Detect edges (amplitude) using the Frei-Chen operator.

Detect edges (amplitude and direction) using the Frei-Chen operator.

Expand the domain of an image to maximum.
CHAPTER 16. CLASS HIERARCHY

- **Funct1DToPairs** ............................................... P.793
  Access to the x/y values of a function.

- **FuzzyEntropy** ................................................. P.328
  Determine the fuzzy entropy of regions.

- **FuzzyMeasurePos** ........................................... P.843
  Extract straight edge pairs perpendicular to a rectangle or an annular arc.

- **FuzzyMeasurePairs** ......................................... P.846
  Extract straight edge pairs perpendicular to a rectangle or an annular arc.

- **FuzzyMeasurePos** ........................................... P.848
  Extract straight edges perpendicular to a rectangle or an annular arc.

- **FuzzyPerimeter** ............................................. P.329
  Calculate the fuzzy perimeter of a region.

- **FwriteString** ................................................ P.21
  Write values in a text file.

- **GaussDistribution** ......................................... P.132
  Generate a Gaussian noise distribution.

- **GaussImage** .................................................. P.140
  Smooth using discrete gauss functions.

- **Gen1DBarCodeDescr** ....................................... P.725
  Generate a description of a 1D bar code.

- **Gen1DBarCodeDescrGen** .................................. P.726
  Generate a generic description of a 1D bar code.

- **Gen2DBarCodeDescr** ....................................... P.727
  Generate a generic description of a 2D bar code class.

- **GenBandfilter** ............................................ P.99
  Generate an ideal band filter.

- **GenBandpass** ............................................... P.100
  Generate an ideal bandpass filter.

- **GenCheckerRegion** ......................................... P.512
  Create a checkered region.

- **GenCircle** .................................................. P.514
  Create a circle.

- **GenContourPolygonRoundedXld** ......................... P.940
  Generate a XLD contour with rounded corners from a polygon (given as tuples).

- **GenContourPolygonXld** ................................ P.941
  Generate an XLD contour from a polygon (given as tuples).

- **GenContourRegionXld** ................................... P.941
  Generate XLD contours from regions.

- **GenContoursSkeletonXld** ................................ P.942
  Convert a skeleton into XLD contours.

- **GenCoocMatrix** ............................................ P.330
  Calculate the co-occurrence matrix of a region in an image.

- **GenDiscSe** .................................................. P.429
  Generate ellipsoidal structuring elements for gray morphology.

- **GenEllipse** ............................................... P.515
  Create an ellipse.

- **GenEllipseContourXld** ................................ P.943
  Creation of an XLD contour corresponding to an elliptic arc.

- **GenEmptyObj** .............................................. P.499
  Create an empty object tuple.

- **GenEmptyRegion** .......................................... P.517
  Create an empty region.
16.19. HOPERATOREX

- GenFilterMask: Store a filter mask in the spatial domain as a real-image.
- GenGabor: Generate a Gabor filter.
- GenGaussPyramid: Calculating a Gauss pyramid.
- GenGridRegion: Create a region from lines or pixels.
- GenHighpass: Generate an ideal highpass filter.
- GenImage1: Create an image from a pointer to the pixels.
- GenImage1Extern: Create an image from a pointer on the pixels with storage management.
- GenImageRect: Create an image with a rectangular domain from a pointer on the pixels (with storage management).
- GenImage3: Create an image from three pointers to the pixels (red/green/blue).
- GenImageConst: Create an image with constant gray value.
- GenImageGrayRamp: Create a gray value ramp.
- GenImageProto: Set the gray values of an image to a specified value.
- GenImageSurfaceFirstOrder: Create a curved gray surface with first order polynomial.
- GenImageSurfaceSecondOrder: Create a curved gray surface with second order polynomial.
- GenImageToWorldPlaneMap: Generate a projection map that describes the mapping between the image plane and a the plane z=0 of a world coordinate system.
- GenLowpass: Generate an ideal lowpass filter.
- GenMeasureArc: Prepare the extraction of straight edges perpendicular to an annular arc.
- GenMeasureRectangle2: Prepare the extraction of straight edges perpendicular to a rectangle.
- GenParallelContourXld: Compute the parallel contour of an XLD contour.
- GenParallelsXld: Extract parallel XLD polygons.
- GenPolygonsXld: Approximate XLD contours by polygons.
- GenPsfDefocus: Generate an impulse response of a uniform out-of-focus blurring.
- GenPsfMotion: Generate an impulse response of a (linearly) motion blurring.
- GenRadialDistortionMap: Generate a projection map that describes the mapping of images corresponding to a changing radial distortion.
- GenRandomRegion: Create a random region.
CHAPTER 16. CLASS HIERARCHY

- GenRandomRegions ........................................ P.519
  Create random regions like circles, rectangles and ellipses.
- GenRectangle1 ............................................ P.521
  Create a rectangle parallel to the coordinate axes.
- GenRectangle2 ............................................ P.522
  Create a rectangle of any orientation.
- GenRegionContourXld ................................... P.524
  Create a region from an XLD contour.
- GenRegionHisto .......................................... P.524
  Convert a histogram into a region.
- GenRegionHline .......................................... P.525
  Store input lines described in Hesse normal shape as regions.
- GenRegionLine ............................................. P.526
  Store input lines as regions.
- GenRegionPoints ......................................... P.527
  Store individual pixels as image region.
- GenRegionPolygon ....................................... P.528
  Store a polygon as an image object.
- GenRegionPolygonFilled ............................... P.529
  Store a polygon as a “filled” region.
- GenRegionPolygonXld .................................. P.530
  Create a region from an XLD polygon.
- GenRegionRuns ........................................... P.530
  Create an image region from a runlength coding.
- GenSinBandpass .......................................... P.104
  Generate a bandpass filter with sinusoidal shape.
- GenStdBandpass .......................................... P.105
  Generate a bandpass filter with Gaussian or sinusoidal shape.
- GenStructElements ...................................... P.462
  Generate standard structuring elements.
- Get1DBarCode ............................................ P.729
  Extract the widths of the elements inside a bar code region.
- Get2DBarCode ............................................ P.730
  Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).
- Get2DBarCodePos ........................................ P.734
  Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.
- GetBgEstiParams ........................................ P.708
  Return the parameters of the data set.
- GetChannelInfo ......................................... P.494
  Information about the components of an image object.
- GetChapterInfo .......................................... P.646
  Get information concerning the chapters on procedures.
- GetCheck .................................................. P.640
  State of the HALCON control modes.
- GetClassBoxParam ........................................ P.4
  Get information about the current parameter.
- GetComprise .............................................. P.218
  Get the output treatment of an image matrix.
- GetContourAngleXld .................................... P.962
  Calculate the direction of an XLD contour for each contour point.
GetContourXld
Return the coordinates of an XLD contour.

GetContourGlobalAttribXld
Return global attributes values of an XLD contour.

GetContourAttribXld
Return point attribute values of an XLD contour.

GetDomain
Get the domain of an image.

GetDraw
Get the current region fill mode.

GetErrorText
Inquiry after the error text of a HALCON error number.

GetFix
Get mode of fixing of current look-up-table (lut).

GetFixedLut
Get fixing of "look-up-table" (lut) for "real color images"

GetFont
Get the current font.

GetFramegrabberLut
Query frame grabber lut.

GetFramegrabberParam
Get specific parameters for a frame grabber.

GetGrayval
Access the gray values of an image object.

GetHsi
Get the HSI coding of the current color.

GetIcon
Query the icon for region output

GetImagePointer1
Access the pointer of a channel.

GetImagePointer1Rect
Access to the image data pointer and the image data inside the smallest rectangle of the domain of the input image.

GetImagePointer3
Access the pointers of a colored image.

GetImageTime
Request time at which the image was created.

GetInsert
Get the current display mode.

GetKeywords
Get keywords which are assigned to procedures.

GetLineApprox
Get the current approximation error for contour display.

GetLineOfSight
Compute the line of sight corresponding to a point in the image.

GetLineStyle
Get the current graphic mode for contours.

GetLineWidth
Get the current line width for contour display.

GetLinesXld
Return an XLD polygon’s data (as lines).
CHAPTER 16. CLASS HIERARCHY

- GetLut: Get current look-up-table (lut).
  - Page: 193
- GetLutStyle: Get modification parameters of look-up-table (lut).
  - Page: 194
- GetModuleName: Wait until a mouse button is pressed.
  - Page: 199
- GetModules: Query of used modules and the module key.
  - Page: 638
- GetMposition: Query the mouse position.
  - Page: 200
- GetMshape: Query the current mouse pointer shape.
  - Page: 201
- GetNextSocketDataType: Determine the HALCON data type of the next socket data.
  - Page: 673
- GetObjClass: Name of the class of an image object.
  - Page: 494
- GetOperatorInfo: Get information concerning a HALCON-procedure.
  - Page: 648
- GetOperatorName: Get procedures with the given string as a substring of their name.
  - Page: 649
- GetOsWindowHandle: Get the operating system window handle.
  - Page: 265
- GetPaint: Get the current display mode for grayvalues.
  - Page: 223
- GetPairFunct1D: Access a function value using the index of the control points.
  - Page: 793
- GetParallelsXld: Return an XLD parallel's data (as lines).
  - Page: 938
- GetParamInfo: Get information concerning the procedure parameters.
  - Page: 650
- GetParamNames: Get the names of the parameters of a HALCON-procedure.
  - Page: 651
- GetParamNum: Get number of the different parameter classes of a HALCON-procedure.
  - Page: 652
- GetParamTypes: Get default data types for the control parameters of a HALCON-procedure.
  - Page: 653
- GetPart: Get the image part.
  - Page: 224
- GetPartStyle: Get the current interpolation mode for grayvalue display.
  - Page: 224
- GetPixel: Get the current color lookup table index.
  - Page: 225
- GetPointsEllipse: Points of an ellipse corresponding to specific angles.
  - Page: 815
- GetPolygonXld: Return an XLD polygon's data.
  - Page: 939
- GetPoseType: Get the representation type of a 3D pose.
  - Page: 762
- GetRegionChain: Contour of an object as chain code.
  - Page: 503
- GetRegionContour: Access the contour of an object.
  - Page: 504
- **GetRegionConvex**
  Access convex hull as contour.

- **GetRegionIndex**
  Index of all regions containing a given pixel.

- **GetRegionPoints**
  Access the pixels of a region.

- **GetRegionPolygon**
  Polygon approximation of a region.

- **GetRegionRuns**
  Access the runlength coding of a region.

- **GetRegionThickness**
  Access the thickness of a region along the main axis.

- **GetRegressParamsXld**
  Return XLD contour parameters.

- **GetRgb**
  Get the current color in RGB-coding.

- **GetSerialParam**
  Get the parameters of a serial device.

- **GetShape**
  Get the current region output shape.

- **GetShapeModelOrigin**
  Return the origin (reference point) of a shape model.

- **GetShapeModelParams**
  Return the parameters of a shape model.

- **GetSocketTimeout**
  Get the timeout of a socket.

- **GetSpy**
  Current configuration of the HALCON debugging-tool.

- **GetStringExtents**
  Get the spatial size of a string.

- **GetSystem**
  Information concerning the currently used HALCON system parameter.

- **GetTposition**
  Get cursor position.

- **GetTshape**
  Get the shape of the text cursor.

- **GetVariationModel**
  Return the images used for image comparison by a variation model.

- **GetWindowAttr**
  Get window characteristics.

- **GetWindowExtents**
  Information about a window’s size and position.

- **GetWindowPointer3**
  Access to a window’s pixel data.

- **GetWindowType**
  Get the window type.

- **GiveBgEsti**
  Return the estimated background image.

- **GnuplotClose**
  Close all open gnuplot files or terminate an active gnuplot sub-process.

- **GnuplotOpenFile**
  Open a gnuplot file for visualization of images and control values.
Calculate horizontal and vertical gray-value projections.

Perform a gray value opening with a selected mask.

Perform a gray value opening on an image.

Perform a gray value closing with a selected mask.

Perform a gray value closing on an image.

Determine the minimum gray value within a selected mask.

Determine the minimum gray value within a rectangle.

Determine the minimum gray value within a rectangle.

Determine the minimum gray value within a selected mask.

Calculate the gray value distribution.

Calculate the gray value distribution.

Calculate the lowest possible gray value on an arbitrary path to the image border for each point in the image.

Perform a gray value opening on an image.

Perform a gray value opening with a selected mask.

Calculate horizontal and vertical gray-value projections.
16.19. HOPERATORSENX

- **GrayRangeRect**
  - Determine the gray value range within a rectangle.
  
- **GraySkeleton**
  - Thinning of gray value images.
  
- **GrayTophat**
  - Perform a gray value top hat transformation on an image.
  
- **HammingChangeRegion**
  - Generate a region having a given Hamming distance.
  
- **HammingDistance**
  - Hamming distance between two regions.
  
- **HammingDistanceNorm**
  - Hamming distance between two regions using normalization.
  
- **HandEyeCalibration**
  - Perform a hand-eye calibration.
  
- **HighpassImage**
  - Extract high frequency components from an image.
  
- **Histo2Dim**
  - Calculate the histogram of two-channel gray value images.
  
- **HistoToThresh**
  - Determine gray value thresholds from a histogram.
  
- **HitOrMiss**
  - Hit-or-miss operation for regions.
  
- **HitOrMissGolay**
  - Hit-or-miss operation for regions using the Golay alphabet.
  
- **HitOrMissSeq**
  - Hit-or-miss operation for regions using the Golay alphabet (sequential).
  
- **HomMat2dCompose**
  - Multiply two homogeneous 2D transformation matrices.
  
- **HomMat2dIdentity**
  - Generate the homogeneous transformation matrix of the identical 2D transformation.
  
- **HomMat2dInvert**
  - Invert a homogeneous 2D transformation matrix.
  
- **HomMat2dRotate**
  - Add a rotation to a homogeneous 2D transformation matrix.
  
- **HomMat2dScale**
  - Add a scaling to a homogeneous 2D transformation matrix.
  
- **HomMat2dSlant**
  - Add a slant to a homogeneous 2D transformation matrix.
  
- **HomMat2dToAffinePar**
  - Compute the affine transformation parameters from a homogeneous 2D transformation matrix.
  
- **HomMat2dTranslate**
  - Add a translation to a homogeneous 2D transformation matrix.
  
- **HomMat3dCompose**
  - Multiply two homogeneous 3D transformation matrices.
  
- **HomMat3dIdentity**
  - Generate the homogeneous transformation matrix of the identical 3D transformation.
  
- **HomMat3dInvert**
  - Invert a homogeneous 3D transformation matrix.
  
- **HomMat3dRotate**
  - Add a rotation to a homogeneous 3D transformation matrix.
  
- **HomMat3dScale**
  - Add a scaling to a homogeneous 3D transformation matrix.
- **HomMat3dToPose**
  Convert a homogeneous transformation matrix into a 3D pose.

- **HomMat3dTranslate**
  Add a translation to a homogeneous 3D transformation matrix.

- **HoughCircleTrans**
  Return the Hough-Transform for circles with a given radius.

- **HoughCircles**
  Centres of circles for a specific radius.

- **HoughLineTrans**
  Produce the Hough transform for lines within regions.

- **HoughLineTransDir**
  Compute the Hough transform for lines using local gradient direction.

- **HoughLines**
  Detect lines in edge images with the help of the Hough transform and returns it in HNF.

- **HoughLinesDir**
  Detect lines in edge images with the help of the Hough transform using local gradient direction and return them in normal form.

- **HysteresisThreshold**
  Perform a hysteresis threshold operation on an image.

- **Illuminate**
  Illuminate image.

- **ImagePointsToWorldPlane**
  Transform image points into the plane \( z=0 \) of a world coordinate system.

- **ImageToChannels**
  Convert a multichannel image into One-channel images.

- **ImageToWorldPlane**
  Rectify an image by transforming it into the plane \( z=0 \) of a world coordinate system.

- **InfoEdges**
  Estimate the width of a filter in **EdgesImage**.

- **InfoFramegrabber**
  Information about the specified frame grabber.

- **InfoOcrClassBox**
  Get information about an OCR classifier.

- **InfoParallelsXld**
  Return information about the gray values of the area enclosed by XLD parallels.

- **InfoSmooth**
  Information on smoothing filter **SmoothImage**.

- **InnerCircle**
  Largest inner circle of a region.

- **InspectShapeModel**
  Create the representation of a shape model.

- **IntToDvf**
  Convert two int1-images into a displacement vector field.

- **IntegerToObj**
  Convert an “integer number” into an iconic object.

- **IntegrateFunct1D**
  Compute the positive and negative areas of a function.

- **Intensity**
  Calculate the mean and deviation of gray values.

- **Interjacent**
  Partition the image plane using given regions.
- **Intersection**
  Calculate the intersection of two regions.

- **Intersection**
  Calculate the intersection point of two lines.

- **InvarFourierCoeff**
  Normalize the Fourier coefficients.

- **InvertImage**
  Invert an image.

- **JunctionsSkeleton**
  Find junctions and end points in a skeleton.

- **KirschAmp**
  Detect edges (amplitude) using the Kirsch operator.

- **KirschDir**
  Detect edges (amplitude and direction) using the Kirsch operator.

- **LabelToRegion**
  Extract regions with equal gray values from an image.

- **Laplace**
  Calculate the Laplace operator by using finite differences.

- **LaplaceOfGauss**
  LoG-Operator (Laplace of Gaussian).

- **LearnClassBox**
  Train the classificator.

- **LearnNdimBox**
  Train the current classificator using a multi-channel image.

- **LearnNdimNorm**
  Construct clusters for **ClassNdimNorm**.

- **LearnSampsetBox**
  Train the classificator with one data set.

- **LengthXld**
  Length of contours or polygons.

- **LineOrientation**
  Calculate the orientation of lines.

- **LinePosition**
  Calculate the center of gravity, length, and orientation of a line.

- **LinesFacet**
  Detection of lines using the facet model.

- **LinesGauss**
  Detect lines and their width.

- **LoadParKnowledge**
  Load knowledge about automatic parallelization from file.

- **LocalMax**
  Detect all local maxima in an image.

- **LocalMaxContoursXld**
  Select XLD contours with a local maximum of gray values.

- **LutTrans**
  Transform an image with a gray-value look-up-table.

- **MapImage**
  Apply a general transformation to an image.

- **MatchFourierCoeff**
  Similarity of two contours.

- **MatchFunct1DTrans**
  Calculate transformation parameters between two functions.
Calculate gray value moments and approximation by a plane.

Arbitrary geometric moments of contours or polygons.

Extract parallel XLD polygons enclosing a homogeneous area.

Reflect a region about the x- or y-axis.

Mirror an image.

Erode a region (using a reference point).

Dilate a region (using a reference point).

Perform a Minkowski addition on a region.

Dilate a region (using a reference point).

Erode a region.

Erode a region (using a reference point).

Mirror an image.

Reflect a region about the x- or y-axis.

Extract parallel XLD polygons enclosing a homogeneous area.

Arbitrary geometric moments of contours or polygons.

Calculate gray value moments and approximation by a plane.
Access the features which correspond to a character.

Define a new conversion table for the characters.

Convert an iconic object into an “integer number.”

Number of control points of the function.

Suppress non-maximum points on an edge.

Suppress non-maximum points on an edge.

Determine the noise distribution of an image.

Set the position of the text cursor to the beginning of the next line.

Create a virtual graphics window under Windows NT.

Set the position of the text cursor to the beginning of the next line.

Determine the noise distribution of an image.

Suppress non-maximum points on an edge.

Suppress non-maximum points on an edge.

Number of control points of the function.

Convert an iconic object into an “integer number.”

Define a new conversion table for the characters.

Access the features which correspond to a character.
Detect all gray value plateaus.

Calculate the deviation of the gray values from the approximating image plane.

Reconstruct a surface from at least three gray value images.

Return the phase of a complex image in radians.

Return the phase of a complex image in degrees.

Reconstruct a surface from at least three gray value images.

Calculate the deviation of the gray values from the approximating image plane.

Detect all gray value plateaus.
PlateausCenter .................................................. P.622
Detect the centers of all gray value plateaus.

PolarTransImage ............................................. P.33
Transform an image to polar coordinates

PoseToHomMat3d .............................................. P.775
Convert a 3D pose into a homogeneous transformation matrix.

Pouring .......................................................... P.623
Segment an image by "pouring water" over it.

PowerByte ...................................................... P.107
Return the power spectrum of a complex image.

PowerLn ........................................................ P.108
Return the power spectrum of a complex image.

PowerReal ...................................................... P.108
Return the power spectrum of a complex image.

PrepContourFourier ........................................ P.790
Parameterize the passed contour

PrepareVariationModel ..................................... P.829
Prepare a variation model for comparison with an image.

PrewittAmp .................................................... P.81
Detect edges (amplitude) using the Prewitt operator.

PrewittDir ..................................................... P.82
Detect edges (amplitude and direction) using the Prewitt operator.

PrincipalComp ............................................... P.128
Compute the principal components of multi-channel images.

Project3DPoint ............................................... P.776
Project 3D points into (sub-)pixel image coordinates.

ProjectionPl ................................................ P.817
Calculate the projection of a point onto a line.

Pruning .......................................................... P.484
Prune the branches of a region.

QueryAllColors ............................................... P.227
Query all color names.

QueryColor .................................................... P.228
Query all color names displayable in the window.

QueryColored ................................................ P.228
Query the number of colors for color output.

QueryContourAttrsXld ..................................... P.971
Return the names of the defined attributes of an XLD contour.

QueryContourGlobalAttrsXld ............................ P.971
Return the names of the defined global attributes of an XLD contour.

QueryFont ..................................................... P.253
Query the available fonts.

QueryGray ..................................................... P.229
Query the displayable grayvalues.

QueryInsert .................................................. P.230
Query the possible graphic modes.

QueryLineWidth ............................................. P.230
Query the possible line widths.

QueryLut ...................................................... P.194
Query all available look-up-tables (lut).

QueryMshape ................................................. P.202
Query all available mouse pointer shapes.
- **QueryOperatorInfo**: Query slots concerning information with relation to the operator `GetOperatorInfo`.
- **QueryPaint**: Query the grayvalue display modes.
- **QueryParamInfo**: Query slots of the online-information concerning the operator `GetParamInfo`.
- **QueryShape**: Query the region display modes.
- **QuerySpy**: Inquiring for possible settings of the HALCON debugging tool.
- **QueryTshape**: Query all shapes available for text cursors.
- **QueryWindowType**: Query all available window types.
- **RankImage**: Smooth an image with an arbitrary rank mask.
- **RankRegion**: Rank operator for regions.
- **ReadCamPar**: Read the interior camera parameters from text file.
- **ReadChar**: Read a character from a text window.
- **ReadClassBox**: Read the classifier from a file.
- **ReadContourXldArcInfo**: Read XLD contours to a file in ARC/INFO generate format.
- **ReadFunct1D**: Read a function from a file.
- **ReadGraySe**: Load a structuring element for gray morphology.
- **ReadImage**: Read an image with different file formats.
- **ReadKalman**: Read the description file of a Kalman filter.
- **ReadOcr**: Read an OCR classifier from a file.
- **ReadOcrTrainf**: Read training characters from files and convert to images.
- **ReadOcrTrainfSelect**: Read training specific characters from files and convert to images.
- **ReadOcv**: Reading an OCV tool from file.
- **ReadPolygonXldArcInfo**: Read XLD polygons from a file in ARC/INFO generate format.
- **ReadPose**: Read a 3D pose from a text file.
- **ReadRegion**: Read binary images or HALCON regions.
- **ReadSampset**: Read a training data set from a file.
<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReadSequence</td>
<td>Read images.</td>
<td>P.12</td>
</tr>
<tr>
<td>ReadSerial</td>
<td>Read from a serial device.</td>
<td>P.670</td>
</tr>
<tr>
<td>ReadShapeModel</td>
<td>Read a shape model from a file.</td>
<td>P.424</td>
</tr>
<tr>
<td>ReadString</td>
<td>Read a string in a text window.</td>
<td>P.255</td>
</tr>
<tr>
<td>ReadTemplate</td>
<td>Reading a template from file.</td>
<td>P.399</td>
</tr>
<tr>
<td>ReadTuple</td>
<td>Read a tuple from a file.</td>
<td>P.23</td>
</tr>
<tr>
<td>ReadVariationModel</td>
<td>Read a variation model from a file.</td>
<td>P.830</td>
</tr>
<tr>
<td>ReadWorldFile</td>
<td>Read the geo coding from an ARC/INFO world file.</td>
<td>P.16</td>
</tr>
<tr>
<td>RealToComplex</td>
<td>Convert two real images into a complex image.</td>
<td>P.371</td>
</tr>
<tr>
<td>ReceiveImage</td>
<td>Receive an image over a socket connection.</td>
<td>P.676</td>
</tr>
<tr>
<td>ReceiveRegion</td>
<td>Receive regions over a socket connection.</td>
<td>P.677</td>
</tr>
<tr>
<td>ReceiveTuple</td>
<td>Receive a tuple over a socket connection.</td>
<td>P.677</td>
</tr>
<tr>
<td>ReceiveXld</td>
<td>Receive an XLD object over a socket connection.</td>
<td>P.678</td>
</tr>
<tr>
<td>RectangleDomain</td>
<td>Reduce the domain of an image to a rectangle.</td>
<td>P.319</td>
</tr>
<tr>
<td>ReduceDomain</td>
<td>Reduce the domain of an image.</td>
<td>P.320</td>
</tr>
<tr>
<td>RegionToBin</td>
<td>Convert a region into a binary byte-image.</td>
<td>P.314</td>
</tr>
<tr>
<td>RegionToLabel</td>
<td>Convert regions to a label image.</td>
<td>P.315</td>
</tr>
<tr>
<td>RegionToMean</td>
<td>Paint regions with their average gray value.</td>
<td>P.316</td>
</tr>
<tr>
<td>RegionGrowing</td>
<td>Segment an image using regiongrowing.</td>
<td>P.625</td>
</tr>
<tr>
<td>RegionGrowingMean</td>
<td>Regiongrowing using mean gray values.</td>
<td>P.626</td>
</tr>
<tr>
<td>RegionGrowingN</td>
<td>Regiongrowing for multi-channel images.</td>
<td>P.628</td>
</tr>
<tr>
<td>RegressContoursXld</td>
<td>Calculate the parameters of a regression line to an XLD contour.</td>
<td>P.984</td>
</tr>
<tr>
<td>RemoveNoiseRegion</td>
<td>Remove noise from a region.</td>
<td>P.591</td>
</tr>
<tr>
<td>ResetFuzzyMeasure</td>
<td>Reset a fuzzy member function.</td>
<td>P.860</td>
</tr>
<tr>
<td>ResetObjDb</td>
<td>Initialization of the HALCON system.</td>
<td>P.639</td>
</tr>
<tr>
<td>Rgb1ToGray</td>
<td>Transform an RGB image into a gray scale image.</td>
<td>P.54</td>
</tr>
</tbody>
</table>
CHAPTER 16. CLASS HIERARCHY

- **Rgb3ToGray** .......................................................... P.54
  Transform an RGB image to a gray scale image.

- **Roberts** ................................................................. P.83
  Detect edges using the Roberts filter.

- **RobinsonAmp** ......................................................... P.84
  Detect edges (amplitude) using the Robinson operator.

- **RobinsonDir** ............................................................ P.85
  Detect edges (amplitude and direction) using the Robinson operator.

- **RotateImage** ............................................................ P.34
  Rotate an image about its center.

- **Roundness** ............................................................... P.554
  Shape factors from contour.

- **RunBgEsti** ............................................................... P.710
  Estimate the background and return the foreground region.

- **RunlengthDistribution** ............................................. P.555
  Distribution of runs needed for runlength encoding of a region.

- **RunlengthFeatures** .................................................. P.556
  Characteristic values for runlength coding of regions.

- **SampleFunct1D** ....................................................... P.797
  Sample a function equidistantly in an interval.

- **ScaleImage** .............................................................. P.44
  Scale the gray values of an image.

- **ScaleImageMax** ....................................................... P.93
  Maximum gray value spreading in the value range 0 bis 255.

- **ScaleYFunct1D** ......................................................... P.797
  Multiplication and addition of the y values.

- **SearchOperator** ....................................................... P.655
  Search names of all procedures assigned to one keyword.

- **SegmentContoursXld** ................................................. P.985
  Segment XLD contours into line segments and circular or elliptic arcs.

- **SelectContoursXld** ................................................. P.972
  Select XLD contours according to several features.

- **SelectGray** .............................................................. P.340
  Select regions based on gray value features.

- **SelectGrayvaluesFromChannels** ................................ P.890
  Selection of gray values of a multi channel image using an index image.

- **SelectLines** ............................................................. P.380
  Select lines according to various criteria.

- **SelectLinesLongest** ................................................ P.382
  Select the longest input lines.

- **SelectMatchingLines** .............................................. P.824
  Select those lines from a set of lines (in HNF) which fit best into a region.

- **SelectObj** .............................................................. P.501
  Select an object from an object tuple.

- **SelectRegionPoint** ................................................ P.557
  Choose all regions containing a given pixel.

- **SelectRegionSpatial** .............................................. P.558
  Pose relation of regions.

- **SelectShape** .......................................................... P.560
  Choose regions with the aid of shape features.

- **SelectShapeProto** ................................................ P.562
  Choose regions having a certain relation to each other.
Select Shape Std .......................... P.564
   Select regions of a given shape.
Select Shape Xld .......................... P.973
   Select contours using shape features.
Send Image ................................ P.678
   Send an image over a socket connection.
Send Region ................................. P.679
   Send regions over a socket connection.
Send Tuple ................................ P.679
   Send a tuple over a socket connection.
Send Xld .................................... P.680
   Send an XLD object over a socket connection.
Sensor Kalman ................................ P.839
   Interactive input of measurement values for a Kalman filtering.
Set Bg Esti Params .......................... P.712
   Change the parameters of the data set.
Set Check ................................... P.642
   Activating and deactivating of HALCON control modes.
Set Class Box Param ........................ P.8
   Set system parameters for classification.
SetColor .................................... P.232
   Set output color.
Set Colored ................................. P.233
   Set multiple output colors.
Set Comprise ................................ P.233
   Define the image matrix output clipping.
Set Draw .................................... P.234
   Define the region fill mode.
Set Fix ..................................... P.235
   Set fixing of “look-up-table” (lut)
Set Fixed Lut ................................ P.195
   Fix “look-up-table” (lut) for “real color images”.
SetFont .................................... P.256
   Set the font used for text output.
Set Framegrabber Lut ......................... P.363
   Set frame grabber lut.
Set Framegrabber Parm ....................... P.364
   Set specific parameters for a frame grabber.
Set Fuzzy Measure ........................... P.861
   Specify a fuzzy member function.
Set Fuzzy Measure Norm Pair ................ P.862
   Specify a normalized fuzzy member function for edge pairs.
Set Gray .................................... P.236
   Define grayvalues for region output.
Set Grayval .................................. P.368
   Set single gray values in an image.
Set Hsi ..................................... P.237
   Define output colors (HSI-coded).
Set Icon .................................... P.238
   Icon definition for region output.
Set Insert ................................... P.238
   Define the pixel output function.
SetLineApprox .................................................. P.239
  Define the approximation error for contour display.
SetLineStyle ............................................... P.240
  Define a contour output pattern.
SetLineWidth ............................................. P.241
  Define the line width for region contour output.
SetLut .......................................................... P.195
  Set "look-up-table" (lut).
SetLutStyle .................................................. P.197
  Changing the look-up-table (lut).
SetMShape ................................................... P.202
  Set the current mouse pointer shape.
SetOffsetTemplate .......................................... P.399
  Gray value offset for template.
SetOriginPose ............................................... P.779
  Translate the origin of a 3D pose.
SetPaint ...................................................... P.241
  Define the grayvalue output mode.
SetPart ........................................................ P.245
  Modify the displayed image part.
SetPartStyle ................................................ P.246
  Define an interpolation method for grayvalue output.
SetPixel ...................................................... P.247
  Define a color lookup table index.
SetReferenceTemplate ...................................... P.400
  Define reference position for a matching template.
SetRgb ........................................................ P.248
  Set the color definition via RGB values.
SetSerialParam ............................................. P.671
  Set the parameters of a serial device.
SetShape ...................................................... P.249
  Define the region output shape.
SetShapeModelOrigin ..................................... P.425
  Set the origin (reference point) of a shape model.
SetSocketTimeout ........................................... P.681
  Set the timeout of a socket.
SetSpy ........................................................ P.643
  Control of the HALCON Debugging Tools.
SetSystem .................................................... P.663
  Setting of HALCON system parameters.
SetTposition ................................................ P.257
  Set the position of the text cursor.
SetTshape .................................................... P.258
  Set the shape of the text cursor.
SetWindowAttr ............................................. P.279
  Set window characteristics.
SetWindowDc ................................................ P.279
  Set the device context of a virtual graphics window (Windows NT).
SetWindowExtents ......................................... P.280
  Modify position and size of a window.
SetWindowType ............................................. P.281
  Specify a window type.
Detect edges (amplitude and direction) using the Sobel operator.

Smooth an image using recursive filters.

Smooth a 1D function by averaging its values.

Smooth an equidistant function with a Gaussian function.

Smooth an XLD contour.

Smallest enclosing rectangle with arbitrary orientation.

Smallest surrounding rectangle with any orientation.

Smallest enclosing rectangle with arbitrary orientation.

Smooth an XLD contour.

Smooth an equidistant function with a Gaussian function.

Smooth a 1D function by averaging its values.

Smooth an image using recursive filters.

Detect edges (amplitude) using the Sobel operator.

Detect edges (amplitude and direction) using the Sobel operator.
Extract level crossings from an image with subpixel accuracy.

Select gray values lying within an interval.

Remove the result of a hit-or-miss operation from a region (sequential).

Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).

Remove the result of a hit-or-miss operation from a region.

Add the result of a hit-or-miss operation to a region (sequential).

Add the result of a hit-or-miss operation to a region (using a Golay structuring element).

Remove the result of a hit-or-miss operation from a region.

Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).

Remove the result of a hit-or-miss operation from a region (sequential).

Select gray values lying within an interval.

Extract level crossings from an image with subpixel accuracy.
- **TileChannels** ................................................................. P.348
  Tile multiple images into a large image.
- **TileImages** ................................................................. P.349
  Tile multiple image objects into a large image.
- **TileImagesOffset** ......................................................... P.350
  Tile multiple image objects into a large image with explicit positioning information.
- **TopHat** ................................................................. P.492
  Compute the top hat of regions.
- **TopographicSketch** .................................................. P.129
  Compute the topographic primal sketch of an image.
- **TrainVariationModel** ............................................... P.831
  Train a variation model.
- **TrainOcrClassBox** .................................................. P.877
  Train an OCR classifier by the input of regions.
- **TrainOcvProj** ........................................................ P.885
  Training of an OCV tool.
- **TrainOcrClassBox** .................................................. P.878
  Train an OCR classifier with the help of a training file.
- **TransFromRgb** ........................................................ P.55
  Transform an image from the RGB color space to an arbitrary color space.
- **TransToRgb** ........................................................ P.58
  Transform an image from an arbitrary color space to the RGB color space.
- **TransformFunct1D** ................................................ P.799
  Transform a function using given transformation parameters.
- **TranslateMeasure** ................................................ P.864
  Translate a measure object.
- **TransposeRegion** ...................................................... P.510
  Reflect a region about a point.
- **TrimmedMean** ........................................................ P.154
  Smooth an image with an arbitrary rank mask.
- **TupleAbs** ............................................................ P.897
  Compute the absolute value of a tuple.
- **TupleAcos** ............................................................ P.897
  Compute the arc cosine of a tuple.
- **TupleAdd** ............................................................... P.898
  Add two tuples.
- **TupleAnd** ............................................................... P.925
  Compute the logical and of two tuples.
- **TupleAsin** ............................................................. P.898
  Compute the arc sine of a tuple.
- **TupleAtan** ............................................................. P.899
  Compute the arc tangent of a tuple.
- **TupleAtan2** ............................................................ P.899
  Compute the arc tangent of a tuple for all four quadrants.
- **TupleBand** ............................................................. P.909
  Compute the bitwise and of two tuples.
- **TupleBnot** ............................................................. P.909
  Compute the bitwise not of two tuples.
- **TupleBor** ............................................................... P.910
  Compute the bitwise or of two tuples.
- **TupleBxor** ............................................................. P.910
  Compute the bitwise exclusive or of two tuples.
TupleCeil
Compute the ceiling function of a tuple.

TupleChr
Convert a tuple of integers into strings with the corresponding ASCII codes.

TupleChrt
Convert a tuple of integers into strings with the corresponding ASCII codes.

TupleConcat
Concatenate two tuple to a new one.

TupleCos
Compute the cosine of a tuple.

TupleCosh
Compute the hyperbolic cosine of a tuple.

TupleDeg
Convert a tuple from radians to degrees.

TupleDeviation
Return the standard deviation of the elements of a tuple.

TupleDiv
Divide two tuples.

TupleEnvironment
Read one or more environment variables.

TupleEqual
Test, whether two tuples are equal.

TupleExp
Compute the exponential of a tuple.

TupleFabs
Compute the absolute value of a tuple (as floating point numbers).

TupleFirstN
Select the first elements of a tuple.

TupleFloor
Compute the floor function of a tuple.

TupleFmod
Calculate the remainder of the floating point division of two tuples.

TupleGenConst
Generate a tuple of a specific length and initialize its elements.

TupleGreater
Test, whether a tuple is greater than another tuple.

TupleGreaterEqual
Test, whether a tuple is greater or equal to another tuple.

TupleInverse
Invert a tuple.

TupleIsNumber
Check a tuple (of strings) whether it represents numbers.

TupleLastN
Select all elements from index “n” to the end of a tuple.

TupleLdexp
Calculate the ldexp function of two tuples.

TupleLength
Returns the number of elements of a tuple.

TupleLess
Test, whether a tuple is less than another tuple.

TupleLessEqual
Test, whether a tuple is less or equal to another tuple.
16.19. HOPERATOE TX

- **TupleLog**
  - Compute the natural logarithm of a tuple.

- **TupleLog10**
  - Compute the base 10 logarithm of a tuple.

- **TupleLsh**
  - Shift a tuple bitwise to the left.

- **TupleMax**
  - Returns the maximal element of a tuple.

- **TupleMean**
  - Return the mean value of a tuple of numbers.

- **TupleMin**
  - Returns the minimal element of a tuple.

- **TupleMult**
  - Multiply two tuples.

- **TupleNeg**
  - Negate a tuple.

- **TupleNot**
  - Compute the logical not of two tuples.

- **TupleNotEqual**
  - Test, whether two tuples are not equal.

- **TupleNumber**
  - Convert a tuple (of strings) into a tuple of numbers.

- **TupleOr**
  - Compute the logical or of two tuples.

- **TupleOrd**
  - Convert a tuple of strings into a tuple of their ASCII codes.

- **TupleOrds**
  - Convert a tuple of strings of length 1 into a tuple of their ASCII codes.

- **TuplePow**
  - Calculate the power function two tuples.

- **TupleRad**
  - Convert a tuple from degrees to radians.

- **TupleReal**
  - Convert a tuple into a tuple of floating point numbers.

- **TupleRound**
  - Convert a tuple into a tuple of integer numbers.

- **TupleRsh**
  - Shift a tuple bitwise to the right.

- **TupleSelect**
  - Select single elements of a tuple.

- **TupleSelectRange**
  - Select several elements of a tuple.

- **TupleSin**
  - Compute the sine of a tuple.

- **TupleSinh**
  - Compute the hyperbolic sine of a tuple.

- **TupleSort**
  - Sorts the elements of a tuple in ascending order.

- **TupleSortIndex**
  - Sorts the elements of a tuple and returns the indices of the sorted tuple.

- **TupleSplit**
  - Split strings into substrings between predefined separator symbol(s).
TupleSqrt
Compute the square root of a tuple.

TupleStrBitSelect
Select single character or bit from a tuple.

TupleStrFirstN
Cut the first characters up to position “n” out of string tuple.

TupleStrLastN
Cut all characters starting at position “n” out of string tuple.

TupleStrchr
Forward search for a character within a string tuple.

TupleStr
Convert a tuple into a tuple of strings.

TupleStrlen
Length of every string within a tuple of strings.

TupleStrrchr
Backward search for a character within a string tuple.

TupleStrrstr
Backward search for string(s) within a string tuple.

TupleStrstr
Forward search for string(s) within a string tuple.

TupleSub
Subtract two tuples.

TupleSum
Return the sum of all elements of a tuple.

TupleTan
Compute the tangent of a tuple.

TupleTanh
Compute the hyperbolic tangent of a tuple.

TupleXor
Compute the logical exclusive or of two tuples.

Union1
Return the union of all input regions.

Union2
Return the union of two regions.

UnionStraightContoursHistoxld
Merge neighboring straight contours having a similar distance from a given line.

UnionStraightContoursXld
Merge neighboring straight contours having a similar direction.

UpdateBgEsti
Change the estimated background image.

UpdateKalman
Read an update file of a Kalman filter.

VectorAngleToRigid
Compute a rigid affine transformation from points and angles.

VectorToHomMat2d
Approximate an affine transformation from point correspondences.

VectorToRigid
Approximate a rigid affine transformation from point correspondences.

VectorToSimilarity
Approximate an similarity transformation from point correspondences.

WaitSeconds
Delaying the execution of the program.
- **Watersheds**
  Extract watersheds and basins from an image.

- **WienerFilter**
  Image restoration by Wiener filtering.

- **WienerFilterNi**
  Image restoration by Wiener filtering.

- **WriteCamPar**
  Write the interior camera parameters to text file.

- **WriteClassBox**
  Save the classifier in a file.

- **WriteContourXldArcInfo**
  Write XLD contours to a file in ARC/INFO generate format.

- **WriteFunct1D**
  Write a function to a file.

- **WriteImage**
  Write images in graphic formats.

- **WriteLut**
  Write look-up-table (lut) as file.

- **WriteOcr**
  Writing an OCR classifier into a file.

- **WriteOcrTrainf**
  Storing of trained characters into a file.

- **WriteOcrTrainfImage**
  Write characters into a training file.

- **WriteOcv**
  Saving an OCV tool to file.

- **WritePolygonXldArcInfo**
  Write XLD polygons to a file in ARC/INFO generate format.

- **WritePose**
  Write a 3D pose to a text file.

- **WriteRegion**
  Write regions on file.

- **WriteSerial**
  Write to a serial connection.

- **WriteShapeModel**
  Write a shape model to a file.

- **WriteString**
  Print text in a window.

- **WriteTemplate**
  Writing a template to file.

- **WriteTuple**
  Write a tuple to a file.

- **WriteVariationModel**
  Write a variation model to a file.

- **XRangeFunct1D**
  Smallest and largest x value of the function.

- **YRangeFunct1D**
  Smallest and largest y value of the function.

- **ZeroCrossing**
  Detect zero crossings in an image.

- **ZeroCrossingSubPix**
  Extract zero crossings from an image with subpixel accuracy.
16.20 HPoseX

Describes a rigid 3D transformation (translation and rotation) with 7 parameters (3 for the rotation, 3 for the translation, 1 for the representation type).

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Methods

- **CameraCalibration**
  
  Determine all camera parameters by a simultaneous minimization process.

- **ContourToWorldPlaneXld**
  
  Transform an XLD contour into the plane z=0 of a world coordinate system.

- **ConvertPoseType**
  
  Change the representation type of a 3D pose.

- **CreatePose**
  
  Create a 3D pose.

- **GenImageToWorldPlaneMap**
  
  Generate a projection map that describes the mapping between the image plane and the plane z=0 of a world coordinate system.

- **GenRadialDistortionMap**
  
  Generate a projection map that describes the mapping of images corresponding to a changing radial distortion.

- **GetPoseType**
  
  Get the representation type of a 3D pose.

- **HandEyeCalibration**
  
  Perform a hand-eye calibration.

- **PoseToHomMat3d**
  
  Convert a 3D pose into a homogeneous transformation matrix.

- **ReadPose**
  
  Read a 3D pose from a text file.

- **SetOriginPose**
  
  Translate the origin of a 3D pose.

- **WritePose**
  
  Write a 3D pose to a text file.
16.21 HRegionX

Represents an instance of a region object.

Class Description

| Number of Methods | 202 |
| Number of Properties | 0 |
| Abstract Class | no |
| Base Class(es) | HObject |
| Automatic Destructor | yes, through ClearObj (not member of this class!) |
| Object Identity | yes (through internal handle) |

Constructor(s)

- DrawPolygon
  Interactive drawing of a polygon row.
- DrawRegion
  Interactive drawing of a closed region.
- GenCheckerRegion
  Create a checkered region.
- GenCircle
  Create a circle.
- GenEllipse
  Create an ellipse.
- GenEmptyRegion
  Create an empty region.
- GenGridRegion
  Create a region from lines or pixels.
- GenRandomRegion
  Create a random region.
- GenRandomRegions
  Create random regions like circles, rectangles and ellipses.
- GenRectangle1
  Create a rectangle parallel to the coordinate axes.
- GenRectangle2
  Create a rectangle of any orientation.
- GenRegionHisto
  Convert a histogram into a region.
- GenRegionHline
  Store input lines described in Hesse normal shape as regions.
- GenRegionLine
  Store input lines as regions.
- GenRegionPoints
  Store individual pixels as image region.
- GenRegionPolygon
  Store a polygon as an image object.
- GenRegionPolygonFilled
  Store a polygon as a “filled” region.
- GenRegionRuns
  Create an image region from a runlength coding.
- GenStructElements
  Generate standard structuring elements.
Methods

GetIcon
Query the complement of a region.

GolayElements
Generate the structuring elements of the Golay alphabet.

GrabRegion
Grab and segment an image from the specified frame grabber.

GrabRegionAsync
Grab and segment an image from the specified frame grabber and start the asynchronous grab of the next image.

ReadRegion
Read binary images or HALCON regions.

ReceiveRegion
Receive regions over a socket connection.

AddChannels
Add gray values to regions.

AffineTransRegion
Apply an arbitrary affine 2D transformation to regions.

AppendOcrTrainf
Add characters to a training file.

AreaCenter
Area and center of regions.

AreaCenterGray
Compute the area and center of gravity of a region in a gray value image.

BackgroundSeg
Determine the connected components of the background of given regions.

BottomHat
Compute the bottom hat of regions.

Boundary
Reduce a region to its boundary.

Circularity
Shape factor for the circularity (similarity to a circle) of a region.

ClipRegion
Clip a region to a rectangle.

ClipRegionRel
Clip a region relative to its size.

CloseEdges
Close edge gaps using the edge amplitude image.

CloseEdgesLength
Close edge gaps using the edge amplitude image.

Closing
Close a region.

ClosingCircle
Close a region with a circular structuring element.

ClosingGolay
Close a region with an element from the Golay alphabet.

ClosingRectangle1
Close a region with a rectangular structuring element.

Compactness
Shape factor for the compactness of a region.

Complement
Return the complement of a region.
ConnectAndHoles
Number of connection components and holes

Connection
Compute connected components of a region.

Contlength
Contour length of a region.

Convexity
Shape factor for the convexity of a region.

CoocFeatureImage
Calculate a co-occurrence matrix and derive gray value features thereof.

DiameterRegion
Maximal distance between two boundary points of a region.

Difference
Calculate the difference of two regions.

Dilation
Dilate a region.

Dilation2
Dilate a region (using a reference point).

DilationCircle
Dilate a region with a circular structuring element.

DilationGolay
Dilate a region with an element from the Golay alphabet.

DilationRectangle1
Dilate a region with a rectangular structuring element.

DilationSeq
Dilate a region sequentially.

DispRegion
Displays regions in a window.

DistanceLr
Calculate the distance between one line and one region.

DistancePr
Calculate the distance between one point and one region.

DistanceRrMin
Minimum distance between the contour pixels of two regions each.

DistanceRrMinDil
Minimum distance between two regions with the help of dilatation.

DistanceSr
Calculate the distance between one line segment and one region.

DistanceTransform
Compute the distance transformation of a region.

DoOcrMulti
Classify characters.

DoOcrSingle
Classify one character.

DragRegion1
Interactive moving of a region.

DragRegion2
Interactive movement of a region with fixpoint specification.

DragRegion3
Interactive movement of a region with restriction of positions.

Eccentricity
Shape features derived from the ellipse parameters.
CHAPTER 16. CLASS HIERARCHY

- EliminateRuns.......................................................... P.580
  Eliminate runs of a given length.
- EllipticAxis............................................................ P.539
  Parameters of the equivalent ellipse.
- EllipticAxisGray...................................................... P.324
  Compute the orientation and major axes of a region in a gray value image.
- EntropyGray............................................................ P.325
  Determine the entropy and anisotropy of images.
- Erosion1................................................................. P.454
  Erode a region.
- Erosion2................................................................. P.456
  Erode a region (using a reference point).
- ErosionCircle.......................................................... P.457
  Erode a region with a circular structuring element.
- ErosionGolay........................................................... P.458
  Erode a region with an element from the Golay alphabet.
- ErosionRectangle1.................................................... P.459
  Erode a region with a rectangular structuring element.
- ErosionSeq............................................................. P.460
  Erode a region sequentially.
- EulerNumber........................................................... P.540
  Calculate the Euler number.
- ExpandGray............................................................ P.610
  Fill gaps between regions (depending on gray value or color) or split overlapping regions.
- ExpandGrayRef......................................................... P.611
  Fill gaps between regions (depending on gray value or color) or split overlapping regions.
- ExpandRegion........................................................ P.580
  Fill gaps between regions or split overlapping regions.
- FillUp................................................................. P.582
  Fill up holes in regions.
- FillUpShape........................................................... P.582
  Fill up holes in regions having given shape features.
- FindNeighbors........................................................ P.541
  Search direct neighbors.
- FitSurfaceFirstOrder................................................ P.326
  Calculate gray value moments and approximation by a first order surface (plane).
- FitSurfaceSecondOrder............................................... P.327
  Calculate gray value moments and approximation by a second order surface.
- Fitting................................................................. P.461
  Perform a closing after an opening with multiple structuring elements.
- FuzzyEntropy........................................................ P.328
  Determine the fuzzy entropy of regions.
- FuzzyPerimeter....................................................... P.329
  Calculate the fuzzy perimeter of a region.
- GenContourRegionXld................................................ P.941
  Generate XLD contours from regions.
- GenContoursSkeletonXld............................................ P.942
  Convert a skeleton into XLD contours.
- GenCoocMatrix........................................................ P.330
  Calculate the co-occurrence matrix of a region in an image.

Calculate the co-occurrence matrix of a region in an image.
Get2DBarCode ........................................................................................................ P.730
Extract the values of the data elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”).

Get2DBarCodePos .................................................................................................. P.734
Extract the data values of the elements (in ECC 200: “modules”) inside a bar code region (“Data Matrix symbol”) and their positions in the image.

GetRegionChain ................................................................................................. P.503
Contour of an object as chain code.

GetRegionContour .............................................................................................. P.504
Access the contour of an object.

GetRegionConvex .............................................................................................. P.504
Access convex hull as contour.

GetRegionIndex .................................................................................................. P.542
Index of all regions containing a given pixel.

GetRegionPoints ................................................................................................. P.505
Access the pixels of a region.

GetRegionPolygon ............................................................................................. P.506
Polygon approximation of a region.

GetRegionRuns ................................................................................................... P.507
Access the runlength coding of a region.

GetRegionThickness ......................................................................................... P.543
Access the thickness of a region along the main axis.

GrayHisto ........................................................................................................... P.332
Calculate the gray value distribution.

GrayHistoAbs ..................................................................................................... P.333
Calculate the gray value distribution.

GrayProjections ................................................................................................. P.334
Calculate horizontal and vertical gray-value projections.

HammingChangeRegion ................................................................................... P.583
Generate a region having a given Hamming distance.

HammingDistance .............................................................................................. P.543
Hamming distance between two regions.

HammingDistanceNorm ...................................................................................... P.544
Hamming distance between two regions using normalization.

Histo2Dim ........................................................................................................... P.334
Calculate the histogram of two-channel gray value images.

HitOrMiss ............................................................................................................ P.466
Hit-or-miss operation for regions.

HitOrMissGolay ................................................................................................. P.468
Hit-or-miss operation for regions using the Golay alphabet.

HitOrMissSeq ...................................................................................................... P.469
Hit-or-miss operation for regions using the Golay alphabet (sequential).

HoughCircleTrans ............................................................................................... P.818
Return the Hough-Transform for circles with a given radius.

HoughCircles ...................................................................................................... P.819
Centres of circles for a specific radius.

HoughLineTrans ................................................................................................. P.820
Produce the Hough transform for lines within regions.

HoughLines ......................................................................................................... P.822
Detect lines in edge images with the help of the Hough transform and returns it in HNF.

InnerCircle .......................................................................................................... P.545
Largest inner circle of a region.
- **Intensity** ........................................... P.335
  Calculate the mean and deviation of gray values.
- **Interjacent** ...................................... P.584
  Partition the image plane using given regions.
- **Intersection** ..................................... P.572
  Calculate the intersection of two regions.
- **JunctionsSkeleton** ................................ P.586
  Find junctions and end points in a skeleton.
- **LearnNdimBox** ................................... P.617
  Train the current classifier using a multi-channel image.
- **LearnNdimNorm** .................................. P.618
  Construct clusters for `ClassNdimNorm`.
- **MergeRegionsLineScan** .......................... P.587
  Merge regions from line scan images.
- **MinMaxGray** ....................................... P.336
  Determine the minimum and maximum gray values within regions.
- **MinkowskiAdd1** .................................. P.470
  Perform a Minkowski addition on a region.
- **MinkowskiAdd2** .................................. P.471
  Dilate a region (using a reference point).
- **MinkowskiSub1** .................................. P.473
  Erode a region.
- **MinkowskiSub2** .................................. P.474
  Erode a region (using a reference point).
- **MirrorRegion** .................................... P.509
  Reflect a region about the x- or y-axis.
- **MomentsGrayPlane** .............................. P.338
  Calculate gray value moments and approximation by a plane.
- **MomentsRegion2Nd** .............................. P.547
  Geometric moments of regions.
- **MomentsRegion2NdInvar** ....................... P.548
  Geometric moments of regions.
- **MomentsRegion2NdRelInvar** ................... P.549
  Geometric moments of regions.
- **MomentsRegion3Rd** ............................. P.549
  Geometric moments of regions.
- **MomentsRegion3RdInvar** ...................... P.550
  Geometric moments of regions.
- **MomentsRegionCentral** ....................... P.551
  Geometric moments of regions.
- **MomentsRegionCentralInvar** ................. P.552
  Geometric moments of regions.
- **MorphHat** ...................................... P.475
  Compute the union of `BottomHat` and `TopHat`.
- **MorphSkeleton** ................................ P.476
  Compute the morphological skeleton of a region.
- **MorphSkiz** ...................................... P.477
  Thinning of a region.
- **MoveRegion** .................................... P.509
  Translate a region.
- **NoiseDistributionMean** ...................... P.133
  Determine the noise distribution of an image.
16.21. HREGIONX

- **Opening**
  - Open a region.

- **OpeningCircle**
  - Open a region with a circular structuring element.

- **OpeningGolay**
  - Open a region with an element from the Golay alphabet.

- **OpeningRectangle**
  - Open a region with a rectangular structuring element.

- **OpeningSeg**
  - Separate overlapping regions.

- **OrientationRegion**
  - Orientation of a region.

- **PaintRegion**
  - Paint regions into an image.

- **PartitionDynamic**
  - Partition a region horizontally into rectangles.

- **PartitionRectangle**
  - Partition a region into rectangles of equal size.

- **PlaneDeviation**
  - Calculate the deviation of the gray values from the approximating image plane.

- **Pruning**
  - Prune the branches of a region.

- **RankRegion**
  - Rank operator for regions.

- **RegionToBin**
  - Convert a region into a binary byte-image.

- **RegionToLabel**
  - Convert regions to a label image.

- **RegionToMean**
  - Paint regions with their average gray value.

- **RemoveNoiseRegion**
  - Remove noise from a region.

- **Roundness**
  - Shape factors from contour.

- **RunlengthDistribution**
  - Distribution of runs needed for runlength encoding of a region.

- **RunlengthFeatures**
  - Characteristic values for runlength coding of regions.

- **SelectGray**
  - Select regions based on gray value features.

- **SelectMatchingLines**
  - Select those lines from a set of lines (in HNF) which fit best into a region.

- **SelectRegionPoint**
  - Choose all regions containing a given pixel.

- **SelectRegionSpatial**
  - Pose relation of regions.

- **SelectShape**
  - Choose regions with the aid of shape features.

- **SelectShapeProto**
  - Choose regions having a certain relation to each other.

- **SelectShapeStd**
  - Select regions of a given shape.
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SendRegion</code></td>
<td>Send regions over a socket connection.</td>
</tr>
<tr>
<td><code>SetIcon</code></td>
<td>Icon definition for region output.</td>
</tr>
<tr>
<td><code>ShapeHist0All</code></td>
<td>Determine a histogram of features along all threshold values.</td>
</tr>
<tr>
<td><code>ShapeHist0Point</code></td>
<td>Determine a histogram of features along all threshold values.</td>
</tr>
<tr>
<td><code>ShapeTrans</code></td>
<td>Transform the shape of a region.</td>
</tr>
<tr>
<td><code>Skeleton</code></td>
<td>Compute the skeleton of a region.</td>
</tr>
<tr>
<td><code>SmallestCircle</code></td>
<td>Smallest surrounding circle of a region.</td>
</tr>
<tr>
<td><code>SmallestRectangle1</code></td>
<td>Surrounding rectangle parallel to the coordinate axes.</td>
</tr>
<tr>
<td><code>SmallestRectangle2</code></td>
<td>Smallest surrounding rectangle with any orientation.</td>
</tr>
<tr>
<td><code>SortRegion</code></td>
<td>Sorting of regions with respect to their relative position.</td>
</tr>
<tr>
<td><code>SpatialRelation</code></td>
<td>Pose relation of regions with regard to the coordinate axes.</td>
</tr>
<tr>
<td><code>SplitSkeletonLines</code></td>
<td>Split lines represented by one pixel wide, non-branching lines.</td>
</tr>
<tr>
<td><code>SplitSkeletonRegion</code></td>
<td>Split lines represented by one pixel wide, non-branching regions.</td>
</tr>
<tr>
<td><code>TestEqualRegion</code></td>
<td>Test whether the regions of two objects are identical.</td>
</tr>
<tr>
<td><code>TestRegionPoint</code></td>
<td>Test if the region consists of the given point.</td>
</tr>
<tr>
<td><code>TrainOcrClassBox</code></td>
<td>Train an OCR classifier by the input of regions.</td>
</tr>
<tr>
<td><code>Thickening</code></td>
<td>Add the result of a hit-or-miss operation to a region.</td>
</tr>
<tr>
<td><code>ThickeningGolay</code></td>
<td>Add the result of a hit-or-miss operation to a region (using a Golay structuring element).</td>
</tr>
<tr>
<td><code>ThickeningSeq</code></td>
<td>Add the result of a hit-or-miss operation to a region (sequential).</td>
</tr>
<tr>
<td><code>Thinning</code></td>
<td>Remove the result of a hit-or-miss operation from a region.</td>
</tr>
<tr>
<td><code>ThinningGolay</code></td>
<td>Remove the result of a hit-or-miss operation from a region (using a Golay structuring element).</td>
</tr>
<tr>
<td><code>ThinningSeq</code></td>
<td>Remove the result of a hit-or-miss operation from a region (sequential).</td>
</tr>
<tr>
<td><code>TopHat</code></td>
<td>Compute the top hat of regions.</td>
</tr>
<tr>
<td><code>TrainOcrClassBox</code></td>
<td>Train an OCR classifier by the input of regions.</td>
</tr>
<tr>
<td><code>TransposeRegion</code></td>
<td>Reflect a region about a point.</td>
</tr>
<tr>
<td><code>Union1</code></td>
<td>Return the union of all input regions.</td>
</tr>
</tbody>
</table>
16.22 HSerialX

Represents an instance of a connection via a serial port.

- **Union2**: Return the union of two regions.
- **WriteOcrTrainf**: Storing of trained characters into a file.
- **WriteRegion**: Write regions on file.
- **ZoomRegion**: Zoom a region.

### Class Description

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through CloseSerial (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>yes</td>
</tr>
</tbody>
</table>

### Constructor(s)

- **OpenSerial**: Open a serial device.

### Methods

- **ClearSerial**: Clear the buffer of a serial connection.
- **GetSerialParam**: Get the parameters of a serial device.
- **ReadSerial**: Read from a serial device.
- **SetSerialParam**: Set the parameters of a serial device.
- **WriteSerial**: Write to a serial connection.

16.23 HShapeModelIX

Represents an instance of a shape model for matching.

### Class Description

<table>
<thead>
<tr>
<th>Number of Methods</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearShapeModel (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>yes</td>
</tr>
</tbody>
</table>
CHAPTER 16. CLASS HIERARCHY

Constructor(s)

- CreateScaledShapeModel
  Prepare a shape model for scale invariant matching.

- CreateShapeModel
  Prepare a shape model for matching.

- ReadShapeModel
  Read a shape model from a file.

Methods

- FindScaledShapeModel
  Find the best matches of a scale invariant shape model in an image.

- FindScaledShapeModels
  Find the best matches of multiple scale invariant shape models.

- FindShapeModel
  Find the best matches of a shape model in an image.

- FindShapeModels
  Find the best matches of multiple shape models.

- GetShapeModelOrigin
  Return the origin (reference point) of a shape model.

- GetShapeModelParams
  Return the parameters of a shape model.

- SetShapeModelOrigin
  Set the origin (reference point) of a shape model.

- WriteShapeModel
  Write a shape model to a file.

16.24 HSocketX

Represents an instance of a communication via a POSIX socket.

Class Description

- Number of Methods: 14
- Number of Properties: 0
- Abstract Class: No
- Base Class(es): None
- Automatic Destructor: Yes, through CloseSocket (not part of this class!)
- Object Identity: Yes, through internal handle

Constructor(s)

- OpenSocketAccept
  Open a socket that accepts connection requests.

- OpenSocketConnect
  Open a socket to an existing socket.

Methods

- GetNextSocketDataType
  Determine the HALCON data type of the next socket data.

- GetSocketTimeout
  Get the timeout of a socket.

- ReceiveImage
  Receive an image over a socket connection.
16.25  HSystemX

Group class for system information and manipulation related functionality.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Methods

- **ReceiveRegion**: Receive regions over a socket connection.
- **ReceiveTuple**: Receive a tuple over a socket connection.
- **ReceiveXld**: Receive an XLD object over a socket connection.
- **SendImage**: Send an image over a socket connection.
- **SendRegion**: Send regions over a socket connection.
- **SendTuple**: Send a tuple over a socket connection.
- **SendXld**: Send an XLD object over a socket connection.
- **SetSocketTimeout**: Set the timeout of a socket.
- **SocketAcceptConnect**: Accept a connection request on a listening socket.
- **CheckParHwPotential**: Check hardware regarding its potential for parallel processing.
- **CountRelation**: Number of entries in the HALCON database.
- **CountSeconds**: Elapsed processing time since the last call of CountSeconds.
- **GetCheck**: State of the HALCON control modes.
- **GetErrorText**: Inquiry after the error text of a HALCON error number.
- **GetModules**: Query of used modules and the module key.
- **GetSpy**: Current configuration of the HALCON debugging-tool.
- **GetSystem**: Information concerning the currently used HALCON system parameter.
- **GetWindowAttr**: Get window characteristics.
CHAPTER 16. CLASS HIERARCHY

- **LoadParKnowledge** ................................................. P.658
  Load knowledge about automatic parallelization from file.

- **QuerySpy** ....................................................... P.642
  Inquiring for possible settings of the HALCON debugging tool.

- **QueryWindowType** ................................................. P.278
  Query all available window types.

- **ResetObjDb** ....................................................... P.639
  Initialization of the HALCON system.

- **SetCheck** .......................................................... P.642
  Activating and deactivating of HALCON control modes.

- **SetSpy** ............................................................. P.643
  Control of the HALCON Debugging Tools.

- **SetSystem** ........................................................ P.663
  Setting of HALCON system parameters.

- **SetWindowAttr** .................................................. P.279
  Set window characteristics.

- **SetWindowType** ................................................ P.281
  Specify a window type.

- **StoreParKnowledge** ............................................. P.659
  Store knowledge about automatic parallelization in file.

- **SystemCall** ...................................................... P.656
  Executes a system command.

- **WaitSeconds** .................................................... P.656
  Delaying the execution of the program.

---

### 16.26 HTemplateX

*Represents an instance of a template for gray value matching.*

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

---

### Constructor(s)

- **CreateTemplate** ............................................ P.393
  Preparing a pattern for template matching.

- **CreateTemplateRot** ....................................... P.395
  Preparing a pattern for template matching with rotation.

- **ReadTemplate** ................................................. P.399
  Reading a template from file.

---

### Methods

- **AdaptTemplate** ............................................. P.385
  Adapting a template to the size of an image.

- **BestMatch** .................................................. P.386
  Searching the best matching of a template and an image.
16.27 HTupleX

Grouping class for tuple-related functionality.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>78</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>no</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>no</td>
</tr>
</tbody>
</table>

### Methods

- **TupleAbs**
  Compute the absolute value of a tuple.

- **TupleAcos**
  Compute the arccosine of a tuple.

- **TupleAdd**
  Add two tuples.

- **TupleAnd**
  Compute the logical and of two tuples.

- **TupleAsin**
  Compute the arcsine of a tuple.

- **TupleAtan**
  Compute the arctangent of a tuple.

- **TupleAtan2**
  Compute the arctangent of a tuple for all four quadrants.

- **TupleBand**
  Compute the bitwise and of two tuples.

- **TupleBnot**
  Compute the bitwise not of two tuples.

- **BestMatchMg**
  Searching the best grayvalue matches in a pyramid.

- **BestMatchPreMg**
  Searching the best grayvalue matches in a pre generated pyramid.

- **BestMatchRot**
  Searching the best matching of a template and an image with rotation.

- **BestMatchRotMg**
  Searching the best matching of a template and a pyramid with rotation.

- **FastMatch**
  Searching all good matches of a template and an image.

- **FastMatchMg**
  Searching all good grayvalue matches in a pyramid.

- **SetOffsetTemplate**
  Gray value offset for template.

- **SetReferenceTemplate**
  Define reference position for a matching template.

- **WriteTemplate**
  Writing a template to file.
Calculate the ldexp function of two tuples.

Check a tuple (of strings) whether it represents numbers.

Invert a tuple.

Test, whether a tuple is greater than another tuple.

Generate a tuple of a specific length and initialize its elements.

Calculate the remainder of the floating point division of two tuples.

Compute the floor function of a tuple.

Select the first elements of a tuple.

Compute the absolute value of a tuple (as floating point numbers).

Select the first elements of a tuple.

Compute the floor function of a tuple.

Calculate the remainder of the floating point division of two tuples.

Generate a tuple of a specific length and initialize its elements.

Test, whether a tuple is greater than another tuple.

Test, whether a tuple is greater or equal to another tuple.

Invert a tuple.

Check a tuple (of strings) whether it represents numbers.

Select all elements from index “n” to the end of a tuple.

Calculate the lexp function of two tuples.

Returns the number of elements of a tuple.
TupleLess ......................................................... P913
Test, whether a tuple is less than another tuple.

TupleLessEqual ............................................... P914
Test, whether a tuple is less or equal to another tuple.

TupleLog ....................................................... P904
Compute the natural logarithm of a tuple.

TupleLog10 ..................................................... P904
Compute the base 10 logarithm of a tuple.

TupleLsh ....................................................... P911
Shift a tuple bitwise to the left.

TupleMax ....................................................... P923
Returns the maximal element of a tuple.

TupleMean ..................................................... P924
Return the mean value of a tuple of numbers.

TupleMin ....................................................... P924
Returns the minimal element of a tuple.

TupleMult ...................................................... P904
Multiply two tuples.

TupleNeg ....................................................... P905
Negate a tuple.

TupleNot ....................................................... P925
Compute the logical not of two tuples.

TupleNotEqual ............................................... P914
Test, whether two tuples are not equal.

TupleNumber .................................................. P917
Convert a tuple (of strings) into a tuple of numbers.

TupleOr ......................................................... P926
Compute the logical or of two tuples.

TupleOrds ..................................................... P917
Convert a tuple of strings of length 1 into a tuple of their ASCII codes.

TuplePow ....................................................... P905
Calculate the power function two tuples.

TupleRad ....................................................... P906
Convert a tuple from degrees to radians.

TupleReal ...................................................... P918
Convert a tuple into a tuple of floating point numbers.

TupleRound ................................................... P918
Convert a tuple into a tuple of integer numbers.

TupleRsh ....................................................... P911
Shift a tuple bitwise to the right.

TupleSelect ................................................... P928
Select single elements of a tuple.

 TupleSelectRange ............................................ P928
Select several elements of a tuple.

TupleSin ....................................................... P906
Compute the sine of a tuple.

TupleSinh .................................................... P907
Compute the hyperbolic sine of a tuple.

TupleSort ..................................................... P921
Sorts the elements of a tuple in ascending order.
16.28  HUnTypedObjectX

weak typed data type for object parameters.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>0</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>HObject</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearObj (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity</td>
<td>yes</td>
</tr>
</tbody>
</table>
16.29 HVariationModelX

Represents an instance of a variation model.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- CreateVariationModel
  Create a variation model for image comparison.

- ReadVariationModel
  Read a variation model from a file.

Methods

- CompareVariationModel
  Compare an image to a variation model.

- GetVariationModel
  Return the images used for image comparison by a variation model.

- PrepareVariationModel
  Prepare a variation model for comparison with an image.

- TrainVariationModel
  Train a variation model.

- WriteVariationModel
  Write a variation model to a file.

16.30 HWindowX

Represents an instance of a HALCON window.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- NewExternWindow
  Create a virtual graphics window under Windows NT.

- OpenTextWindow
  Open a textual window.

- OpenWindow
  Open a graphics window.
CHAPTER 16. CLASS HIERARCHY

Methods

- **ClearRectangle**
  Delete a rectangle on the output window.

- **ClearWindow**
  Delete an output window.

- **CopyRectangle**
  Copy all pixels within rectangles between output windows.

- **DispArc**
  Displays circular arcs in a window.

- **DispArrow**
  Displays arrows in a window.

- **DispCalib**
  Project and visualize the 3D model of the calibration plate in the image.

- **DispChannel**
  Displays images with several channels.

- **DispCircle**
  Displays circles in a window.

- **DispColor**
  Displays a color (RGB) image.

- **DispDistribution**
  Displays a noise distribution.

- **DispEllipse**
  Displays ellipses.

- **DispImage**
  Displays gray value images.

- **DispLine**
  Draws lines in a window.

- **DispLut**
  Graphical view of the look-up-table (lut).

- **DispObj**
  Displays image objects (image, region, XLD).

- **DispPolygon**
  Displays a polyline.

- **DispRectangle1**
  Display of rectangles aligned to the coordinate axes.

- **DispRectangle2**
  Displays arbitrarily oriented rectangles.

- **DispRegion**
  Displays regions in a window.

- **DispXld**
  Display an XLD object.

- **DragRegion1**
  Interactive moving of a region.

- **DragRegion2**
  Interactive movement of a region with fixpoint specification.

- **DragRegion3**
  Interactive movement of a region with restriction of positions.

- **DrawCircle**
  Interactive drawing of a circle.

- **DrawCircleMod**
  Interactive drawing of a circle.
Get modification parameters of look-up-table (lut).

Get current look-up-table (lut).

Get the current line width for contour display.

Get the current graphic mode for contours.

Get the current approximation error for contour display.

Get the current graphic mode for contours.

Get the current line width for contour display.

Get current look-up-table (lut).

Get modification parameters of look-up-table (lut).
199

CHAPTER 16. CLASS HIERARCHY

- GetMButton ................................................................. P.199
  Wait until a mouse button is pressed.
- GetMPosition .............................................................. P.200
  Query the mouse position.
- GetMShape ................................................................. P.201
  Query the current mouse pointer shape.
- GetOsWindowHandle ......................................................... P.265
  Get the operating system window handle.
- GetPaint ................................................................. P.223
  Get the current display mode for grayvalues.
- GetPart ................................................................. P.224
  Get the image part.
- GetPartStyle ............................................................. P.224
  Get the current interpolation mode for grayvalue display.
- GetPixel ................................................................. P.225
  Get the current color lookup table index.
- GetRgb ................................................................. P.226
  Get the current color in RGB-coding.
- GetShape ................................................................. P.226
  Get the current region output shape.
- GetStringExtents .......................................................... P.251
  Get the spatial size of a string.
- GetTPosition ............................................................. P.251
  Get cursor position.
- GetTShape ................................................................. P.252
  Get the shape of the text cursor.
- GetWindowExtents ........................................................ P.266
  Information about a window’s size and position.
- GetWindowPointer3 ......................................................... P.267
  Access to a window’s pixel data.
- GetWindowType ........................................................... P.267
  Get the window type.
- MoveRectangle .......................................................... P.268
  Copy inside an output window.
- NewLine ................................................................. P.253
  Set the position of the text cursor to the beginning of the next line.
- QueryAllColors .......................................................... P.227
  Query all color names.
- QueryColor ............................................................... P.228
  Query all color names displayable in the window.
- QueryFont ............................................................... P.253
  Query the available fonts.
- QueryGray ............................................................... P.229
  Query the displayable grayvalues.
- QueryInsert ............................................................. P.230
  Query the possible graphic modes.
- QueryLut ................................................................. P.194
  Query all available look-up-tables (lut).
- QueryMShape ............................................................ P.202
  Query all available mouse pointer shapes.
- QueryPaint .............................................................. P.231
  Query the grayvalue display modes.
define the region output shape.

set the color definition via RGB values.

define a color lookup table index.

define an interpolation method for grayvalue output.

modify the displayed image part.

define the grayvalue output mode.

icon definition for region output.

define output colors (HSI-coded).

define grayvalues for region output.

define output colors (HSI-coded).

define the pixel output function.

define the region fill mode.

set fixing of “look-up-table” (lut)

fix “look-up-table” (lut) for “real color images”.

set the font used for text output.

define grayvalues for region output.

set the font used for text output.

define fixing of “look-up-table” (lut)

define the image matrix output clipping.

set multiple output colors.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.

set the color definition via RGB values.
**SetCursorPosition**
Set the position of the text cursor.

**SetCurrentShape**
Set the shape of the text cursor.

**SetWindowDeviceContext**
Set the device context of a virtual graphics window (Windows NT).

**SetWindowExtents**
Modify position and size of a window.

**SlideImage**
Interactive output from two window buffers.

**WriteLut**
Write look-up-table (lut) as file.

**WriteString**
Print text in a window.

---

**Properties**

- **AllColors** (read-only)
  mapped to: QueryAllColors
  Query all color names.

- **Blue**
  mapped to: GetRgb
  Get the current color in RGB-coding.

- **Color**
  mapped to: SetColor
  Set output color.

- **Colored**
  mapped to: SetColored
  Set multiple output colors.

- **Colors** (read-only)
  mapped to: QueryColor
  Query all color names displayable in the window.

- **Comprise**
  mapped to: SetComprise
  Define the image matrix output clipping.

- **Draw**
  mapped to: GetDraw
  Get the current region fill mode.

- **Fix**
  mapped to: GetFix
  Get mode of fixing of current look-up-table (lut).

- **Font**
  mapped to: GetFont
  Get the current font.

- **Fonts** (read-only)
  mapped to: QueryFont
  Query the available fonts.

- **GraphicModes** (read-only)
  mapped to: QueryInsert
  Query the possible graphic modes.

- **Gray**
  mapped to: SetGray
  Define grayvalues for region output.

- **GrayValues** (read-only)
  mapped to: QueryGray
  Query the displayable grayvalues.
**Green**
- mapped to: GetRgb
  *Get the current color in RGB-coding.*

**Hue**
- mapped to: GetHsi
  *Get the HSI coding of the current color.*

**Insert**
- mapped to: GetInsert
  *Get the current display mode.*

**Intensity**
- mapped to: GetHsi
  *Get the HSI coding of the current color.*

**LineApprox**
- mapped to: GetLineApprox
  *Get the current approximation error for contour display.*

**LineWidth**
- mapped to: GetLineWidth
  *Get the current line width for contour display.*

**Lut**
- mapped to: GetLut
  *Get current look-up-table (lut).*

**Luts (read-only)**
- mapped to: QueryLut
  *Query all available look-up-tables (lut).*

**OSDisplayHandle (read-only)**
- mapped to: GetOsWindowHandle
  *Get the operating system window handle.*

**OSWindowHandle (read-only)**
- mapped to: GetOsWindowHandle
  *Get the operating system window handle.*

**Paint**
- mapped to: GetPaint
  *Get the current display mode for grayvalues.*

**PaintModes (read-only)**
- mapped to: QueryPaint
  *Query the grayvalue display modes.*

**PartColumn1**
- mapped to: GetPart
  *Get the image part.*

**PartColumn2**
- mapped to: GetPart
  *Get the image part.*

**PartRow1**
- mapped to: GetPart
  *Get the image part.*

**PartRow2**
- mapped to: GetPart
  *Get the image part.*

**PartStyle**
- mapped to: GetPartStyle
  *Get the current interpolation mode for grayvalue display.*

**Pixel**
- mapped to: GetPixel
  *Get the current color lookup table index.*
Red
mapped to: GetRgb
Get the current color in RGB-coding.

Saturation
mapped to: GetHsi
Get the HSI coding of the current color.

Shape
mapped to: GetShape
Get the current region output shape.

Type (read-only)
mapped to: GetWindowType
Get the window type.

16.31 HXLDX

Instance of a XLD object.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>21</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>yes</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>HObject</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearObj (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity</td>
<td>yes</td>
</tr>
<tr>
<td>(through internal handle)</td>
<td></td>
</tr>
</tbody>
</table>

Methods

ReceiveXld
Receive an XLD object over a socket connection.

AreaCenterXld
Area and center of gravity (centroid) of contours and polygons.

CircularityXld
Shape factor for the circularity (similarity to a circle) of a contour.

CompactnessXld
Shape factor for the compactness of a contour.

ConvexityXld
Shape factor for the convexity of a contour.

DiameterXld
Maximum distance between two contour points.

DispXld
Display an XLD object.

EccentricityXld
Shape features derived from the ellipse parameters.

EllipticAxisXld
Parameters of the equivalent ellipse.

GetParallelsXld
Return an XLD parallel’s data (as lines).

LengthXld
Length of contours or polygons.

MomentsAnyXld
Arbitrary geometric moments of contours or polygons.
16.32. HXLDContX

Represents an instance of an XLD contour object.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>32</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>HXLD</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearObj (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
<td>yes</td>
</tr>
</tbody>
</table>

Constructor(s)

- **GenContourPolygonRoundedXld**
  *Generate a XLD contour with rounded corners from a polygon (given as tuples).*
  [P.940](#)

- **GenContourPolygonXld**
  *Generate an XLD contour from a polygon (given as tuples).*
  [P.941](#)

- **GenEllipseContourXld**
  *Creation of an XLD contour corresponding to an elliptic arc.*
  [P.943](#)

- **ReadContourXldArcInfo**
  *Read XLD contours to a file in ARC/INFO generate format.*
  [P.24](#)

Methods

- **AddNoiseWhiteContourXld**
  *Add noise to XLD contours.*
  [P.978](#)

- **AffineTransContourXld**
  *Apply an arbitrary affine 2D transformation to XLD contours.*
  [P.979](#)

- **ChangeRadialDistortionContoursXld**
  *Change the radial distortion of contours.*
  [P.743](#)

- **ClipContoursXld**
  *Clip an XLD contour.*
  [P.980](#)
CHAPTER 16. CLASS HIERARCHY

- **ContourPointNumXld**
  
  Return the number of points in an XLD contour.

- **DistEllipseContourXld**
  
  Distance of contours to an ellipse.

- **FitCircleContourXld**
  
  Approximation of XLD contours by circles.

- **FitEllipseContourXld**
  
  Approximation of XLD contours by ellipses or elliptic arcs.

- **FitLineContourXld**
  
  Approximation of XLD contours by line segments.

- **GenParallellContourXld**
  
  Compute the parallel contour of an XLD contour.

- **GenPolygonsXld**
  
  Approximate XLD contours by polygons.

- **GenRegionContourXld**
  
  Create a region from an XLD contour.

- **GetContourAngleXld**
  
  Calculate the direction of an XLD contour for each contour point.

- **GetContourAttribXld**
  
  Return point attribute values of an XLD contour.

- **GetContourGlobalAttribXld**
  
  Return global attributes values of an XLD contour.

- **GetContourXld**
  
  Return the coordinates of an XLD contour.

- **GetRegressParamsXld**
  
  Return XLD contour parameters.

- **LocalMaxContoursXld**
  
  Select XLD contours with a local maximum of gray values.

- **MergeContLineScanXld**
  
  Merge XLD contours from successive line scan images.

- **QueryContourAttribsXld**
  
  Return the names of the defined attributes of an XLD contour.

- **QueryContourGlobalAttribsXld**
  
  Return the names of the defined global attributes of an XLD contour.

- **RegressContoursXld**
  
  Calculate the parameters of a regression line to an XLD contour.

- **SegmentContoursXld**
  
  Segment XLD contours into line segments and circular or elliptic arcs.

- **SelectContoursXld**
  
  Select XLD contours according to several features.

- **SmoothContoursXld**
  
  Smooth an XLD contour.

- **UnionStraightContoursHistoXld**
  
  Merge neighboring straight contours having a similar distance from a given line.

- **UnionStraightContoursXld**
  
  Merge neighboring straight contours having a similar direction.

- **WriteContourXldArcInfo**
  
  Write XLD contours to a file in ARC/INFO generate format.
### 16.33 HXLDExtParaX

Represents an instance of an XLD extended parallel object.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>1</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>HXLD</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearObj (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity</td>
<td>yes (through internal handle)</td>
</tr>
</tbody>
</table>

#### Methods

- **MaxParallelsXld**
  
  Join modified XLD parallels lying on the same polygon.

### 16.34 HXLDModParaX

Represents an instance of an XLD modified parallel object.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>1</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>HXLD</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearObj (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity</td>
<td>yes (through internal handle)</td>
</tr>
</tbody>
</table>

#### Methods

- **CombineRoadsXld**
  
  Combine road hypotheses from two resolution levels.

### 16.35 HXLDParaX

Represents an instance of an XLD parallel object.

<table>
<thead>
<tr>
<th>Class Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
<td>2</td>
</tr>
<tr>
<td>Number of Properties</td>
<td>0</td>
</tr>
<tr>
<td>Abstract Class</td>
<td>no</td>
</tr>
<tr>
<td>Base Class(es)</td>
<td>HXLD</td>
</tr>
<tr>
<td>Automatic Destructor</td>
<td>yes, through ClearObj (not member of this class!)</td>
</tr>
<tr>
<td>Object Identity</td>
<td>yes (through internal handle)</td>
</tr>
</tbody>
</table>

#### Methods

- **InfoParallelsXld**
  
  Return information about the gray values of the area enclosed by XLD parallels.

- **ModParallelsXld**
  
  Extract parallel XLD polygons enclosing a homogeneous area.
16.36 HXLDPolyX

Represents an instance of an XLD polygon object.

<table>
<thead>
<tr>
<th>Class Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Methods</td>
</tr>
<tr>
<td>Number of Properties</td>
</tr>
<tr>
<td>Abstract Class</td>
</tr>
<tr>
<td>Base Class(es)</td>
</tr>
<tr>
<td>Automatic Destructor</td>
</tr>
<tr>
<td>Object Identity (through internal handle)</td>
</tr>
</tbody>
</table>

Constructor(s)

- **ReadPolygonXldArcInfo**
  - Read XLD polygons from a file in ARC/INFO generate format.

Methods

- **AffineTransPolygonXld**
  - Apply an arbitrary affine transformation to XLD polygons.
- **CombineRoadsXld**
  - Combine road hypotheses from two resolution levels.
- **GenParallelsXld**
  - Extract parallel XLD polygons.
- **GenRegionPolygonXld**
  - Create a region from an XLD polygon.
- **GetLinesXld**
  - Return an XLD polygon’s data (as lines).
- **GetPolygonXld**
  - Return an XLD polygon’s data.
- **SplitContoursXld**
  - Split XLD contours at dominant points.
- **WritePolygonXldArcInfo**
  - Write XLD polygons to a file in ARC/INFO generate format.
Index

3D-Projection, 761, 776
3D-Transformation, 684, 693–695, 697, 698, 745, 749, 762, 770, 775, 778, 779, 784
AbsFunct1D, 790
AbsImage, 37
AbsInvarFourierCoeff, 785
Absolute-Value, 37, 790, 897, 902
abstract, 648
Access, 793
AccessChannel, 290
AdaptTemplate, 385
AddChannels, 317
AddImage, 38
Addition, 38, 797, 898
AddNoiseDistribution, 130
AddNoiseWhite, 131
Affine-Transformation, 29, 31, 508, 685, 701, 979
AreaCenterXld, 948
Arithmetic, 37–45
assurance, 648, 650
Attributes, 963, 971
AutoThreshold, 597
Average, 924, 947, 962, 964, 965, 972
Background, 574, 578
background, 704, 705, 708–710, 712, 714
background-estimation, 704, 705, 708–710, 712, 714
BackgroundSeg, 574
Band-Filter, 99
Bandpass-Filter, 100, 104, 105, 109
BandpassImage, 109
Barcode, 715, 717, 718, 721, 725, 726, 729
Barcode-description, 725, 726
Best-Match, 386–388, 390, 391, 396, 398, 408, 411, 415, 418
BestMatch, 386
BestMatchMg, 387
BestMatchPreMg, 388
BestMatchRot, 390
BestMatchRotMg, 391
Binary-image, 17
BinThreshold, 598
Bit, 46–52
Bit-Operations, 909–911, 929
BitAnd, 46
BitLshift, 47
BitMask, 48
BitNot, 49
BitOr, 49
BitRshift, 50
BitSlice, 51
BitsPerChannel (property), 997
BitXor, 52
Blue (property), 1080
Blurring, 159–162
BMP, 11, 14, 17, 263
Border-Treatment, 122
Bottom-Hat, 429, 441, 475
BottomHat, 441
Boundary, 442
Boundary, 442
Branches, 484
Buffer, 272, 275, 659, 663
Bulkiness, 538, 954
C-procedure, 652
Calibration, 735, 737, 742–744, 746, 753–755, 757, 760, 761, 771, 773, 776, 777, 780, 782
Index

CaltabPoints, 735
CameraCalibration, 737
Canny-Filter, 67, 70
Ceiling-Function, 900
Center, 321, 514, 532, 560, 566, 973, 976
Center-of-Gravity, 379, 380
Centroid, 948
CfaToRgb, 53
Chain-code, 503
Chaincode, 373
Chaincodes, 376
ChangeDomain, 318
ChangeFormat, 344
ChangeRadialDistortionCamPar, 742
ChangeRadialDistortionContoursXld, 743
ChangeRadialDistortionImage, 743
Channels, 639
ChannelsToImage, 291
Chapter, 646
chapter, 648
Character, 881–885
Character-sequence, 20
Characters, 874–876
CharThreshold, 598
CheckDifference, 599
Checkerboard, 512
CheckParHwPotential, 657
Choice-of-regions, 560, 973
Chord, 555, 556
Chord-representation, 507
Circle, 206, 444, 450, 457, 479, 514, 519, 545, 554, 565, 975, 985
circle, 172, 173, 249, 956
Circularity, 533
CircularityXld, 949
Class, 494
Class2DimSup, 601
Class2DimUnsup, 602
Classification, 129, 601–604, 617, 618
ClassNdimBox, 603
ClassNdimNorm, 604
ClearAllShapeModels, 401
ClearAllTemplates, 392
ClearAllVariationModels, 825
Clearing, 310, 497
ClearObj, 497
ClearRectangle, 259
ClearSampset, 1
ClearSerial, 667
ClearShapeModel, 401
ClearTemplate, 393
ClearVariationModel, 826
ClearWindow, 260
ClipContoursXld, 980
Clipping, 345, 575, 576, 588, 980
ClipRegion, 575
ClipRegionRel, 576
Close, 881

CloseAllBgEsti, 704
CloseAllClassBox, 1
CloseAllFiles, 18
CloseAllFramegrabbers, 352
CloseAllMeasures, 842
CloseAllOcrs, 866
CloseAllOcvs, 881
CloseAllSerials, 668
CloseBgEsti, 704
CloseClassBox, 2
CloseEdges, 61
CloseEdgesLength, 62
CloseFile, 18
CloseFramegrabber, 352
CloseMeasure, 843
CloseOcr, 866
CloseOcv, 881
CloseSerial, 668
CloseSocket, 673
CloseWindow, 261
Closing, 261, 427, 429, 430, 441, 443–445, 461, 462, 475
Closing, 443
ClosingCircle, 444
ClosingGolay, 445
ClosingRectangle, 446
Clustering, 602–604, 617, 618
Co-occurrence-Matrix, 322, 323, 325, 330
Coding, 556
Coincidence, 519
Color, 53, 192–195, 197
color, 55, 58, 220, 225–228, 232, 233, 237, 247
Color (property), 1080
color-coding, 220, 237
Color-Image, 208
color-image, 54
color-lookup-table, 225, 226
Color-Space, 53
color-space, 55, 58
Color-Space-Transformation, 53
color-space-transformation, 54, 55, 58
Color-Table, 659, 663
Colored (property), 1080
Colors (property), 1080
ColorSpace (property), 997
CombineRoadsXld, 981
Compactness, 533, 534, 560, 949, 973
Compactness, 534
CompactnessXld, 949
CompareVariationModel, 826
Comparison, 495, 883
Comparison-Operators, 912–914
Complement, 570
Complement, 570
ComplexToReal, 368
Components, 494
Compose2, 291
Compose3, 292
Compose4, 293

HALCON/COM Reference Manual, 2005-2-1
Compose5, 293
Compose6, 294
Compose7, 295
Comprise (property), 1080
Concatenate, 298
ConcatObject, 498
ConcatOCRTrainf, 867
Configuration, 278
configuration, 227–231, 253
ConnectAndHoles, 535
Connected-Components, 341, 343, 574, 577
Connection, 577
Connection-component, 535, 540
ContLength, 536
Contour, 503, 504, 506, 536, 809
contour, 222, 223, 234, 240
Contour-Length, 577
Contour-point, 535, 540
ContourPointNumXld, 950
ContourToWorldPlaneXld, 744
Contrast, 95
contrast, 90
Contrast-enhancement, 91, 92
Control, 641, 657–659, 663
Control-Modes, 642
control-parameter, 651
Conversion, 915–919
CopyImage, 302
Copying, 302, 310, 364–366, 498, 501
CopyObject, 498
CopyRectangle, 261
CornerResponse, 114
Corners, 114
Correlation, 115, 117, 386–388, 390, 391, 396, 398
Correspondences, 755
Cosine, 900
CountChannels, 296
CountObject, 493
CountRelation, 637
CountSeconds, 655
Create, 882
CreateBgEsti, 705
CreateCaltab, 746
CreateClassBox, 2
CreateFunct1DArray, 791
CreateFunct1DPairs, 791
CreateOCRClassBox, 867
CreateOcvProj, 882
CreatePose, 749
CreateScaledShapeModel, 402
CreateShapeModel, 405
CreateTemplate, 393
CreateTemplateRot, 395
CreateVariationModel, 827
Creation, 920, 921
CropDomain, 344
CropDomainRel, 345
CropPart, 346
CropRectangle, 347
Cross-reference, 647, 655
Cursor, 201, 202
cursor-position, 251
Database, 302, 497, 498, 500, 501, 637, 639
Debugging, 641, 642
Decode, 715, 717
Decode1DBarCode, 715
Decode2DBarCode, 716
Decompose2, 296
Decompose3, 297
Decompose4, 298
Decompose5, 299
Decompose6, 299
Decompose7, 300
default-type, 650
Degrees, 906, 916
Delete, 496
Deletion, 259, 260
DepthFromFocus, 886
Deriche-Filter, 67, 70, 152
DerivativeGauss, 63
describeClassBox, 2
description-file, 11
description-file, 836
detectEdgeSegments, 605
Deviation, 335, 339, 340
deviationImage, 155
devicecontext, 279
Diameter, 537, 951
diameterRegion, 537
diameterXld, 951
Difference, 571
difference, 571
differential-geometry, 63
diffOfGauss, 66
diffusion-Filter, 135
dilatation, 810
dilation1, 447
Dilation2, 448
DilationCircle, 450
DilationGolay, 451
DilationRectangle1, 452
DilationSeq, 453
Direction, 937–939, 962, 964, 972, 978, 982, 988–990
Direction-code, 503
Discrete, 717
Discrete1DBarCode, 717
DispArc, 203
DispArrow, 204
DispCaltab, 753
DispChannel, 205
DispCircle, 206
DispColor, 208
DispDistribution, 208
DispEllipse, 209
DispImage, 211
DispInfo, 646
Displacement-Vector-Field, 118, 120, 370
DISPLAY, 272, 275
Display, 218
display, 221, 230
DispLine, 212
DispLut, 191
DispObj, 213
DispPolygon, 214
DispRectangle1, 214
DispRectangle2, 216
DispRegion, 217
DispXld, 218
Distance, 537, 578, 792, 809, 810, 951, 952
DistanceFunct1D, 792
DistanceLr, 803
DistancePl, 805
DistancePp, 806
DistancePr, 807
DistancePs, 808
DistanceRmRm, 809
DistanceRmRmD1, 810
DistanceS1, 811
DistanceSr, 812
DistanceSt, 813
DistanceTransform, 578
DistEllipseContourXld, 952
Distribution, 325, 332–334, 336, 555
DivImage, 39
Division, 39, 901, 903
Documentation, 659, 663
Domain, 317–320
DoOcrMulti, 870
DoOcrSingle, 870
DoOcvSimple, 883
DragRegion1, 169
DragRegion2, 170
DragRegion3, 171
Draw (property), 1080
DrawCircle, 172
DrawCircleMod, 173
DrawEllipse, 174
DrawEllipseMod, 175
drawing, 172, 173, 176, 177, 179, 181, 182, 184–186
DrawLine, 176
DrawLineMod, 177
DrawLut, 192
DrawPoint, 179
DrawPointMod, 179
DrawPolygon, 180
DrawRectangle1, 181
DrawRectangle1Mod, 182
DrawRectangle2, 184
DrawRectangle2Mod, 185
DrawRegion, 186
DualRank, 427
DualThreshold, 606
DumpWindow, 263
DvfToHomMat2d, 685
DvfToInt1, 370
Dynamic-Threshold, 599, 608
DynThreshold, 608
Eccentricity, 538
EccentricityXld, 954
Edge-Amplitude, 67, 70, 73, 77, 78, 81, 82, 84, 85, 87, 88
Edge-Closing, 61, 62
Edge-Detection, 61–63, 66, 67, 73, 75–79, 81, 82, 84, 85, 87, 88, 605, 620, 621
Edge-Direction, 67, 70, 73, 78, 82, 85, 88, 982
Edge-Preservation, 135, 151
Edge-Thinning, 67, 620, 621
Edges, 61, 62, 605, 606, 620, 621, 634, 635, 842, 843, 846, 848, 850, 852, 854, 856, 858, 859, 864
EdgesImage, 67
EdgesSubPix, 70
Elements, 718
EliminateMinMax, 136
EliminateRuns, 580
EliminateSp, 138
Ellipse, 209, 324, 515, 519, 538, 539, 815, 952, 954, 955, 958
ellipse, 174, 175, 249, 943, 958
EllipticAxis, 539
EllipticAxisGray, 324
EllipticAxisXld, 955
Emphasize, 90
End-Points, 586
Energy, 95
EnergyGabor, 95
EnquireClassBox, 3
EnquireRejectClassBox, 4
Entropy, 156, 325, 328, 340
EntropyGray, 325
EntropyImage, 156
EquHistoImage, 91
Gabor-Filter, 95, 101
Gain (property), 997
Gaps, 580, 610, 611
Gauss-Filter, 140, 152
Gauss-Pyramid, 119
GaussDistribution, 132
Gaussian, 63, 66, 79, 81, 87, 88, 105, 140, 152, 597, 598, 615
Gaussian-Function, 798
Gen1DBarCodeDescr, 725
Gen1DBarCodeDescrGen, 726
Gen2DBarCodeDescr, 727
GenBandfilter, 99
GenBandpass, 100
GenCheckerRegion, 512
GenCircle, 514
GenContourPolygonRoundedXld, 940
GenContourPolygonXld, 941
GenContourRegionXld, 941
GenContoursSkeletonXld, 942
GenCoocMatrix, 330
GenDiscSe, 429
GenEllipse, 515
GenEllipseContourXld, 943
GenEmptyObj, 499
GenEmptyRegion, 517
generic, 726
GenFilterMask, 100
GenGabor, 101
GenGaussPyramid, 119
GenGridRegion, 517
GenHighpass, 103
GenImage1, 302
GenImage1Extern, 303
GenImage1Rect, 305
GenImage3, 306
GenImageConst, 307
GenImageGrayRamp, 308
GenImageProto, 310
GenImageSurfaceFirstOrder, 311
GenImageSurfaceSecondOrder, 312
GenImageToWorldPlaneMap, 757
GenLowpass, 103
GenMeasureArc, 850
GenMeasureRectangle2, 852
GenParallelContourXld, 982
GenParallelsXld, 945
GenPolygonsXld, 946
GenPsfDefocus, 159
GenPsfMotion, 160
GenRadialDistortionMap, 760
GenRandomRegion, 518
GenRandomRegions, 519
GenRectangle1, 521
GenRectangle2, 522
GenRegionContourXld, 524
GenRegionHisto, 524
GenRegionHline, 525
GenRegionLine, 526
GenRegionPoints, 527
GenRegionPolygon, 528
GenRegionPolygonFilled, 529
GenRegionPolygonXld, 530
GenRegionRuns, 530
GenSinBandpass, 104
GenStdBandpass, 105
GenStructElements, 462
Geocoding, 16, 24–26
Get1DBarCode, 729
Get2DBarCode, 730
Get2DBarCodePos, 734
GetBgEstiParams, 708
GetChannelInfo, 494
GetChapterInfo, 646
GetCheck, 640
GetClassBoxParam, 4
GetComprise, 218
GetContourAngleXld, 962
GetContourAttribXld, 963
GetContourGlobalAttribXld, 963
GetContourXld, 937
GetDomain, 319
GetDraw, 219
GetErrorText, 640
GetFix, 219
GetFixedLut, 193
GetFont, 250
GetFramegrabberLut, 353
GetFramegrabberParam, 353
GetGrayval, 285
GetHsi, 220
GetIcon, 221
GetImagePointer1, 286
GetImagePointer1Rect, 287
GetImagePointer3, 288
GetImageTime, 289
GetInsert, 221
GetKeywords, 647
GetLineApprox, 222
GetLineOfSight, 761
GetLineStyle, 222
GetLinesXld, 937
GetLineWidth, 223
GetLut, 193
GetLutStyle, 194
GetMbutton, 199
GetModules, 638
GetMposition, 200
GetPaint, 223
GetPairFunct1D, 793

HALCON/COM Reference Manual, 2005-2-1
HFramegrabberX, 996
HFunction1dX, 998
HGraphPlotX, 999
HHistogramX, 999
HHomMat2dX, 999
HHomMat3dX, 1001
High-Pass-Filter, 75
Highpass-Filter, 103
HighpassImage, 75
Hilbert-Transform, 95, 101
 HIMageX, 1001
HInfoX, 1013
Histo2Dim, 334
Histogram, 332–334, 336, 341, 343, 524, 597, 598, 601, 602, 615
histogram, 241
Histogram-linearisation, 91
Histogram-spreading, 91
HistoToThresh, 615
Hit-Or-Miss-Transformation, 463, 466, 468, 469, 485–490
HitOrMiss, 466
HitOrMissGolay, 468
HitOrMissSeq, 469
HMeasureX, 1014
HMiscX, 1015
HNF, 820, 822–824
HObjectX, 1017
HOcrX, 1018
HOcvX, 1019
hole, 535, 540
Holes, 582
HomMat2dCompose, 686
HomMat2dIdentity, 686
HomMat2dInvert, 687
HomMat2dRotate, 687
HomMat2dScale, 689
HomMat2dSlant, 690
HomMat2dToAffinePar, 691
HomMat2dTranslate, 692
HomMat3dCompose, 693
HomMat3dIdentity, 694
HomMat3dInvert, 695
HomMat3dRotate, 695
HomMat3dScale, 697
HomMat3dToPose, 770
HomMat3dTranslate, 698
HOperatorSetX, 1019
Hopfield, 543, 544
HorizontalResolution (property), 997
Hough-Transform, 818–820, 822–824
HoughCircles, 819
HoughCircleTrans, 818
HoughLines, 822
HoughLinesDir, 823
HoughLineTrans, 820
HoughLineTransDir, 820
HPoseX, 1058
HRegionX, 1059
HSeriesX, 1067
HShapeModelX, 1067
HSocketX, 1068
HSystemX, 1069
HTemplateX, 1070
HTupleX, 1071
Hue (property), 1081
Hull, 504, 545, 565, 567, 975, 977
hull, 249
HUntypedObjectX, 1074
HVariationModelX, 1075
HWindowX, 1075
HXLDContX, 1083
HXLDExtParaX, 1085
HXLDModParaX, 1085
HXLDParaX, 1085
HXLDPolyX, 1086
HXLDX, 1082
Hyper-Cube, 604, 618
Hyper-Cuboid, 603, 617
Hyper-Sphere, 604, 618
Hyperbolic-Cosine, 900
Hyperbolic-Sine, 907
Hyperbolic-Tangent, 908
Hysteresis, 615
Hysteresis-Thresholding, 67, 70
HysteresisThreshold, 615
Iconic-Object, 302, 497, 498, 500, 501, 637
Illuminate, 92
Illumination, 92
Image, 213, 302, 303, 305–308, 311, 312, 659, 663, 676, 678
image-clipping, 224, 245, 246
Image-Comparison, 825–827, 829–832
Image-directory, 11
Image-Generation, 314, 315
Image-grabbing, 353–358, 364
image-matrix, 233
Image-Object, 211, 659, 663
Image-object, 11, 302, 303, 305–308, 311, 312, 493–496
image-object, 233
Image-Part, 344, 346, 347
image-part, 224, 245, 246
Image-Plane, 326, 327, 338, 339
Image-plane, 308
Image-Restoration, 164, 165
image-sequences, 704, 705, 708–710, 712, 714
Image-Size, 344, 346–350
Image_data, 12
ImageHeight (property), 997
ImagePointsToWorldPlane, 771
Images, 639
ImageToWorldPlane, 773
Maximum, 41, 120, 125, 336, 340, 433, 800, 801, 923
MaxParallelsXld, 968
Mean, 335, 340, 798
Mean-Filter, 136, 138, 142, 143
MeanImage, 142
MeanN, 143
MeanSp, 143
Measurement, 842, 843, 846, 848, 850, 852, 854, 856, 858, 859, 864
measurements, 832, 836, 839, 840
MeasurePairs, 854
MeasurePos, 856
MeasureProjection, 858
MeasureThresh, 859
Medial-Axis, 592
Median, 589
Median-Filter, 136, 145–149, 154, 427
MedianImage, 145
MedianSeparate, 146
MedianWeighted, 147
Memory, 659
MergeContLineScanXld, 983
MergeRegionsLineScan, 587
Metric, 578
Mexican-Hat-Operator, 63, 66, 81
Midrange-Filter, 148
MidrangeImage, 148
Minima, 597, 598, 615
MinImage, 42
Minimum, 42, 120, 336, 340, 431, 435, 436, 438, 800, 801, 809, 810, 924
Minkowski-Addition, 470, 471, 478–480, 482, 483
Minkowski-Subtraction, 443–445, 473, 474
MinkowskiAdd1, 470
MinkowskiAdd2, 471
MinkowskiSub1, 473
MinkowskiSub2, 474
MinMaxGray, 336
MirrorImage, 32
MirrorRegion, 509
ModParallelsXld, 947
Modules, 638, 717
Moments, 324, 326, 327, 338, 538, 539, 547–552, 954, 955, 968, 970
MomentsAnyXld, 968
MomentsGrayPlane, 338
MomentsRegion2Nd, 547
MomentsRegion2NdInvar, 548
MomentsRegion2NdRelInvar, 549
MomentsRegion3Rd, 549
MomentsRegion3RdInvar, 550
MomentsRegionCentral, 551
MomentsRegionCentralInvar, 552
MomentsXld, 970
Monotony, 120
Monotony, 120
MorphHat, 475
MorphSkeleton, 476
MorphSkiz, 477
Motion-Blur, 160, 162, 164, 165
Mouse, 199–202
MoveContourOrig, 789
movement, 169–171
MoveRegion, 268
MoveRegion, 509
multichannel, 205
MultiImage, 43
Multiplication, 43, 797, 904
Name (property), 997
NegateFunct1D, 795
Negation, 795, 905
Neighborhood, 118, 541
neighborhood, 558, 568
NewExternWindow, 269
NewLine, 253
Noise, 130–134, 164, 165, 978
noise, 136
Noise-Distribution, 130–134, 978
Noise-Removal, 591
NoiseDistributionMean, 133
Non-linear-Filter, 135
Non-Maximum-Suppression, 67, 620, 621
NonmaxSuppressionAmp, 620
NonmaxSuppressionDir, 621
not, 49
NumPointsFunct1D, 796
Object, 213, 496, 499
object, 12
Object-choice, 542, 557
Object-comparison, 495, 496
Object-Key, 500
Object-Selection, 340
ObjToInteger, 509
OCR, 874–876
OcrChangeChar, 872
OcrGetFeatures, 873
online-text, 647, 654, 655
OpenFile, 22
OpenFramegrabber, 361
Opening, 427, 437, 439, 461, 462, 475, 478–480, 482, 483, 492
Opening, 478
OpeningCircle, 479
OpeningGolay, 480
OpeningRectangle1, 482
OpeningSeg, 483
OpenSerial, 670
OpenSocketAccept, 674
OpenSocketConnect, 676
OpenTextwindow, 272
OpenWindow, 275
Operating-System-Window-Handle, 265
Optical-Flow, 120, 370
OpticalFlowMatch, 120
Orientatio, 560
Orientation, 324, 377–380, 515, 522, 539, 553, 567, 718, 955, 970, 973, 977
OrientationRegion, 553
OrientationXld, 970
OSDisplayHandle (property), 1081
OSWindowHandle (property), 1081
Out-Of-Focus-Blur, 159, 161, 164, 165
Outer-circle, 560
Outliers, 984
Output, 203, 205, 206, 208, 209, 211, 213, 214, 216, 253
output, 218, 222, 223, 258
output-parameter, 651
parameter, 223, 648, 650, 651, 653, 654
parameter-number, 652
PartColumn1 (property), 1081
PartColumn2 (property), 1081
Partition, 588
PartitionDynamic, 588
PartitionLines, 379
PartitionRectangle, 588
PartRow1 (property), 1081
PartRow2 (property), 1081
PartStyle (property), 1081
Pattern, 512, 517, 881–885
Perimeter, 329
Phase, 106
PhaseDeg, 106
PhaseRad, 106
Photometric-Stereo, 889
PhotStereo, 889
Pixel, 505, 527
Pixel (property), 1081
Pixel-Relations, 659, 663
PixelTypes, 12
Pixels, 517, 519
PlaneDeviation, 339
Plateaus, 622
Plateaus, 622
PlateausCenter, 622
plot, 241
PNG, 11
point, 179
Polar-Transformation, 33
PolarTransImage, 33
Polygon, 214, 528, 529, 554
polygon, 180
Polygon-Approximation, 946
Polygon-approximation, 506
Polygon-approximation, 239
Polygon-Length, 966
Polygons, 937–941, 946, 968, 979, 981, 988
Polyline, 214
Port (property), 997
Pose, 745, 749, 762, 770, 775, 777, 779, 784
Pose-relation, 558, 568
PoseToHomMat3d, 775
PostScript, 263
Pouring, 623
Pouring, 623
Power-Function, 905
Power-Spectrum, 106–108
PowerByte, 107
PowerLn, 108
PowerReal, 108
prediction, 832, 836, 840
PrepareVariationModel, 829
PrepContourFourier, 790
Prewitt-Filter, 81, 82
PrewittAmp, 81
PrewittDir, 82
Principal-Components, 128
PrincipalComp, 128
procedure, 652, 654
Procedure-Chapters, 646
procedure-name, 649
Product-of-inertia, 547–552
Project3DPoint, 776
projection, 241
ProjectionPl, 817
Pruning, 484
Pruning, 484
Pyramid, 117
QueryAllColors, 227
QueryColor, 228
QueryColored, 228
QueryContourAttribsXld, 971
QueryContourGlobalAttribsXld, 971
QueryFont, 253
QueryGray, 229
QueryInsert, 230
QueryLineWidth, 230
QueryLut, 194
QueryMshape, 202
QueryOperatorInfo, 654
QueryPaint, 231
QueryParamInfo, 654
QueryShape, 231
QuerySpy, 642
QueryTshape, 254
QueryWindowType, 278
Radians, 906, 916
Selection, 498, 500, 501, 927–929
SelectLines, 380
SelectLinesLongest, 382
SelectMatchingLines, 824
SelectObj, 501
SelectRegionPoint, 557
SelectRegionSpatial, 558
SelectShape, 560
SelectShapeProto, 562
SelectShapeStd, 564
SelectShapeXld, 973
SendImage, 678
SendRegion, 679
SendTuple, 679
SendXld, 680
SensorKalman, 839
Separating-Lines, 584
Serial-Interface, 667–672
SetBgEstiParam, 712
SetCheck, 642
SetClassBoxParam, 8
SetColor, 232
SetColored, 233
SetComprise, 233
SetDraw, 234
SetFix, 235
SetFixedLut, 195
SetFont, 256
SetFramegrabberLut, 363
SetFramegrabberParam, 364
SetFuzzyMeasure, 861
SetFuzzyMeasureNormPair, 862
SetGray, 236
SetGrayval, 368
SetHsi, 237
SetIcon, 238
SetInsert, 238
SetLineApprox, 239
SetLineStyle, 240
SetLineWidth, 241
SetLut, 195
SetLutStyle, 197
SetMshape, 202
SetOffsetTemplate, 399
SetOriginPose, 779
SetPaint, 241
SetPart, 245
SetPartStyle, 246
SetPixel, 247
SetReferenceTemplate, 400
SetRgb, 248
SetSerialParam, 671
SetShape, 249
SetShapeModelOrigin, 425
SetSocketTimeout, 681
SetSpy, 643
SetSystem, 663
SetTposition, 257
SetTshape, 258
SetWindowAttr, 279
SetWindowDC, 279
SetWindowExtents, 280
SetWindowType, 281
SfsModLr, 891
SfsOrigLr, 892
SfsPentland, 893
ShadeHeightField, 895
Shading, 889, 891–893, 895
Shadows, 895
Shape, 562
Shape (property), 1082
Shape-Feature, 966, 968, 970
Shape-feature, 533, 534, 536, 538, 539, 543, 544, 547, 553, 554, 564, 949, 951, 954, 955, 970
Shape-Features, 582
Shape-from-Shading, 889, 891–893
Shape-Transformation, 591, 987
ShapeHistAll, 341
ShapeHistPoint, 343
ShapeTrans, 591
ShapeTransXld, 987
Sharpness, 90
Shen-Filter, 67, 70, 152
Shift, 47, 50
Sigma-Filter, 151
SigmaImage, 151
SimCaltab, 780
Similarity, 564, 792
Similarity-Transformation, 703
SimulateDefocus, 161
SimulateMotion, 162
Sine, 104, 105, 906
Skeleton, 476, 477, 484, 488–490, 592, 942
Skeleton, 592
Slant, 690, 691, 887, 888
SlideImage, 282
Slope, 120
SmallestCircle, 565
SmallestCircleXld, 975
SmallestRectangle1, 566
SmallestRectangle1Xld, 976
SmallestRectangle2, 567
SmallestRectangle2Xld, 977
SmoothContoursXld, 988
SmoothFunct1D Gauss, 798
SmoothFunct1D Mean, 798
SmoothImage, 152
Smoothing, 63, 135, 136, 138–143, 145–149, 151, 152, 154, 427, 798, 988
Sobel-Filter, 87, 88
SobelAmp, 87
SobelDir, 88
SocketAcceptConnect, 681
Sockets, 673, 674, 676–681
SortRegion, 593
Spatial-Domain, 96, 98, 100
SpatialRelation, 568
SpDistribution, 134
Split-And-Merge, 594, 595
SplitContoursXld, 988
SplitSkeletonLines, 594
SplitSkeletonRegion, 595
spreading, 93
Square-Root, 907
Standard-Deviation, 155, 922, 947, 964, 965, 972
StartColumn (property), 997
StartRow (property), 997
Store, 884, 885
StoreParKnowledge, 659
Storing-Capacity, 663
String, 20, 21
String-Operators, 929 – 934
Structure-factor, 538, 954
Structuring-Elements, 429, 440
SubImage, 45
Subtraction, 45, 907
Sum, 924
Surrogate, 496, 500
Surrounding-circle, 560, 565, 973, 975
surrounding-circle, 249
Surrounding-rectangle, 560, 566, 567, 973, 976, 977
Symmetry, 128
Symmetry, 128
System-Call., 656
System-Parameter, 659, 663
SystemCall, 656
systems, 832, 836, 840
Tangent, 908
Test-pixel, 542, 557, 569
TestdOcrClassBox, 876
TestEqualObj, 496
TestEqualRegion, 496
TestObjDef, 496
TestRegionPoint, 569
TestSampsetBox, 9
Text, 272
text, 250–256, 258
text-cursor, 252, 254, 258
text-file, 18–22
text-position, 257
text-size, 251
Texture, 146, 155–157, 322, 323, 325, 330
Texture-Transformation, 157
TextureLaws, 157
Thickening, 485–487
Thickening, 485
ThickeningGolay, 486
ThickeningSeq, 487
Thinning, 125, 476, 477, 488–490, 592
Thinning, 488
ThinningGolay, 489
ThinningSeq, 490
Threshold, 341, 343, 597–599, 606, 608, 614, 615, 632, 859
Threshold, 632
Thresholding, 597, 598, 615
ThresholdSubPix, 633
TIFF, 11, 263
Tiff, 14, 17
TileChannels, 348
TileImages, 349
TileImagesOffset, 350
Tiling, 348–350
Tilt, 889
Time-Measure, 655
Top-Hat, 439, 475, 492
TopHat, 492
Topographic-Primal-Sketch, 129
TopographicSketch, 129
Trace, 641
TraindOcrClassBox, 877
TraindOcvProj, 885
TrainfOcrClassBox, 878
Training, 617, 618, 874–876, 885
TrainVariationModel, 831
Transformation, 126, 127, 578, 757, 760, 771, 773, 794, 797, 799
TransformFunct1D, 799
TransFromRgb, 55
TranslateMeasure, 864
Translation, 509, 691, 692, 698, 779
TransposeRegion, 510
Transposition, 510
TransToRgb, 58
TrimmedMean, 154
truecolor, 241
Tuple, 499, 517, 897–934
Tuple-Length, 923
Tuple-length, 493
TupleAbs, 897
TupleAcos, 897
TupleAdd, 898
TupleAnd, 925
TupleAsin, 898
TupleAtan, 899
TupleAtan2, 899
TupleBand, 909
TupleBnot, 909
TupleBor, 910
TupleBxor, 910
TupleCeil, 900
TupleChir, 915
TupleChrt, 915
TupleConcat, 920
TupleCos, 900
TupleCosh, 900
TupleDeg, 916
TupleDeviation, 922
TupleDiv, 901
TupleEnvironment, 929
TupleEqual, 912
TupleExp, 901
TupleFabs, 902
TupleFirstN, 927
TupleFloor, 902
TupleFmod, 903
TupleGenConst, 921
TupleGreater, 912
TupleGreaterEqual, 913
TupleInverse, 921
TupleIsNumber, 916
TupleLastN, 927
TupleLdexp, 903
TupleLength, 923
TupleLess, 913
TupleLessEqual, 914
TupleLog, 904
TupleLog10, 904
TupleLsh, 911
TupleMax, 923
TupleMean, 924
TupleMin, 924
TupleMult, 904
TupleNeg, 905
TupleNot, 925
TupleNotEqual, 914
TupleNumber, 917
TupleOr, 926
TupleOrd, 917
TupleOrds, 917
TuplePow, 905
TupleRad, 906
TupleReal, 918
TupleRound, 918
TupleRsh, 911
TupleSelect, 928
TupleSelectRange, 928
TupleSin, 906
TupleSinh, 907
TupleSort, 921
TupleSortIndex, 922
TupleSort, 907
TupleSqrt, 907
TupleStrBitSelect, 929
TupleStrChar, 932
TupleStrFirstN, 931
TupleString, 919
TupleStrLastN, 931
TupleStrLen, 922
TupleStrrChar, 933
TupleStrrStr, 934
TupleStrStr, 934
TupleSub, 907
TupleSum, 924
TupleTan, 908
TupleTanh, 908
TupleXor, 926
type, 650
Type (property), 1082
Type-Conversion, 368, 370, 371

types, 653

Union, 572, 573

Union1, 572
Union2, 573
UnionStraightContoursHistoXld, 989
UnionStraightContoursXld, 990
update-file, 840
UpdateBgEsti, 714
UpdateKalman, 840

Variation-Model, 825–827, 829–832
VectorAngleToRigid, 700
VectorToHomMat2d, 701
VectorToRigid, 702
VectorToSimilarity, 703
Verification, 881–885
VerticalResolution (property), 997

WaitSeconds, 656
Watersheds, 124, 634
Watersheds, 634
Weighting, 147
Wiener-Filter, 159, 160, 164, 165
WienerFilter, 164
WienerFilterNi, 165
Window, 261, 269, 272, 275, 278, 279, 659, 663
Window-Dump, 263
Window-Position, 280
Window-Size, 266, 280
Window-Types, 278
World-File, 16, 24–26
WriteCamPar, 782
WriteClassBox, 10
WriteContourXldArcInfo, 26
WriteFunct1D, 800
WriteImage, 14
WriteLut, 199
WriteOcr, 879
WriteOcrTrainf, 879
WriteOcrTrainfImage, 880
WriteOcv, 885
WritePolygonXldArcInfo, 26
WritePose, 784
WriteRegions, 17
WriteSerial, 672
WriteShapeModel, 425
WriteString, 258
WriteTemplate, 400
WriteTuple, 23
WriteVariationModel, 832

X-Window, 659, 663
XLD-Parallels, 218, 678, 680, 938, 945, 947, 965, 968, 981

HALCON 6.1.4
XLD-Polygons, 25, 26, 218, 678, 680, 937–939, 946, 968, 979, 981, 988
xor, 52
XRangeFunct1D, 800
YRangeFunct1D, 801
Zero-Crossings, 606, 634, 635
ZeroCrossing, 634
ZeroCrossingSubPix, 635
ZoomImageFactor, 35
ZoomImageSize, 36
Zooming, 511
ZoomRegion, 511