HALCON Version 6.0.4

HALCON/HDevelop

Referenzhandbuch

Dieses Handbuch beschreibt die Operatoren des Bildverarbeitungssystems HALCON, Version 6.0.4, in HDevelop-Syntax. Es wurde generiert am 21. Juli 2003.

Mehr Information über HALCON findet sich unter: `http://www.mvtec.com`

Inhaltsverzeichnis

Kapitel 1

Bild

1.1 Domain

add_channels (Regions, Image : GrayRegions : :)

Regionen mit Grauwerten versehen.

add_channels versieht die Regionen in Regions mit den Grauwerten aus Image. Es werden dabei alle Kanäle von Image übernommen. Der Definitionsbereich berechnet sich als der Durchschnitt des Definitionsbereiches des Bildes mit der Region. Der neue Definitionsbereich kann also eine Teilmenge der Eingaberegion sein. Die Größe der Matrix wird nicht verändert.

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Eingaberegionen (ohne Grauwerte).
- ▷ **Image** (input_object) . . . . . (multichannel-)image ⇝ *Hobject* : any
  Graubild für Regionen.
- ▷ **GrayRegions** (output_object) . . . . . image(-array) ⇝ *Hobject*
  Regionen mit Grauwerten (also Bilder).
  Parameteranzahl : Regions = GrayRegions

Parallelisierungsinformation

add_channels ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, gen_circle, draw_region

Mögliche Nachfolgerfunktionen

threshold, regiongrowing, get_domain

Alternativen

change_domain, reduce_domain

Siehe auch

full_domain, get_domain, intersection

Modul

Image / region / XLD management

change_domain (Image, NewDomain : ImageNew : :)

Verändert den Definitionsbereich eines Bildes.

change_domain verwendet die angegebene Region als neuen Definitionsbereich. Im Gegensatz zu reduce_domain wird hier nicht der Durchschnitt mit dem bisherigen Definitionsbereich gebildet. Dies kann

insbesondere dann zu Fehlern führen, wenn die Region größer als die Bildmatrix ist. Die Größe der Matrix wird nicht verändert.

Achtung

Aus Laufzeitgründen wird die übergebene Region nicht auf Konsistenz (d.h. ob sie zur Bildmatrix paßt) überprüft. Falsche Regionen führen bei nachfolgenden Operationen zu Systemabstürzen.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **NewDomain** (input_object) ..... region ⇝ *Hobject*
  Neuer Definitionsbereich.
- ▷ **ImageNew** (output_object) ..... image(-array) ⇝ *Hobject*
  Bild mit neuem Definitionsbereich.

Parallelisierungsinformation

change_domain ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

get_domain

Alternativen

reduce_domain

Siehe auch

full_domain, get_domain, intersection

Modul

Image / region / XLD management

full_domain (Image : ImageFull : :)

Definitionsbereich eines Bildes maximal erweitern.

full_domain trägt ein Rechteck mit der Kantenlänge des Bildes als neuen Defintionsbereich ein. Dies bedeutet, daß alle Pixel der Matrix bei weiteren Operationen mit einbezogen werden. Man erhält somit den gleichen Definitionsbereich, den man nach dem Einlesen oder Erzeugen eines Bildes hat. Die Größe der Matrix wird nicht verändert.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **ImageFull** (output_object) ..... image(-array) ⇝ *Hobject*
  Bild mit maximalem Definitionsbreich.

Parallelisierungsinformation

full_domain ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

get_domain

Alternativen

change_domain, reduce_domain

Siehe auch

get_domain, gen_rectangle1

Modul

Image / region / XLD management

get_domain (Image : Domain : :)

Zugriff auf den Definitionsbereich eines Bildes.

get_domain liefert zu allen Bildern in der Eingabe die Definitionsbereiche als Region zurück.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Eingabebilder.
- ▷ **Domain** (output_object) ..... region(-array) ⇝ *Hobject*
  Definitionsbereiche der Eingabebilder.

Parallelisierungsinformation

get_domain ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

change_domain, reduce_domain, full_domain

Siehe auch

get_domain, change_domain, reduce_domain, full_domain

Modul

Image / region / XLD management

rectangle1_domain (Image : ImageReduced : Row1, Column1, Row2, Column2 :)

Verkleinert den Definitionsbereich eines Bildes auf ein Rechteck.

rectangle1_domain verkleinert den Definitionsbereich der Eingabebilder auf das angegebene Rechteck. Der alte Definitionsbereich des Bildes wird dabei nicht beachtet. Die Größe der Matrix wird nicht verändert.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **ImageReduced** (output_object) ..... image(-array) ⇝ *Hobject*
  Bild mit reduziertem Definitionsbereich.
- ▷ **Row1** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilenindex des linken oberen Ecks des Bildauschnittes.
  Defaultwert : 100
  Wertevorschläge : Row1 ∈ {10, 20, 50, 100, 200, 300, 500}
  Typischer Wertebereich : $0 \leq$ Row1 ≤ 1024
- ▷ **Column1** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltenindex des linken oberen Ecks des Bildauschnittes.
  Defaultwert : 100
  Wertevorschläge : Column1 ∈ {10, 20, 50, 100, 200, 300, 500}
  Typischer Wertebereich : $0 \leq$ Column1 ≤ 1024
- ▷ **Row2** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilenindex des rechten unteren Ecks des Bildauschnittes.
  Defaultwert : 200
  Wertevorschläge : Row2 ∈ {10, 20, 50, 100, 200, 300, 500}
  Typischer Wertebereich : $0 \leq$ Row2 ≤ 1024
- ▷ **Column2** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltenindex des rechten unteren Ecks des Bildauschnittes.
  Defaultwert : 200
  Wertevorschläge : Column2 ∈ {10, 20, 50, 100, 200, 300, 500}
  Typischer Wertebereich : $0 \leq$ Column2 ≤ 1024

Parallelisierungsinformation

rectangle1_domain ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

get_domain

Alternativen

change_domain, reduce_domain, add_channels

Siehe auch

full_domain, get_domain, intersection

Modul

Image / region / XLD management

reduce_domain (Image, Region : ImageReduced : :)

Verkleinert den Definitionsbereich eines Bildes.

reduce_domain verkleinert den Definitionsbereich der Eingabebilder um die angegebene Region. Der neue Definitionsbereich berechnet sich als der Durchschnitt des alten Definitionsbereiches mit der Region. Der neue Definitionsbereich kann also eine Teilmenge der Region sein. Die Größe der Matrix wird nicht verändert.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Neuer Definitionsbereich.
- ▷ **ImageReduced** (output_object) ..... image(-array) ⇝ *Hobject*
  Bild mit reduziertem Definitionsbreich.

Parallelisierungsinformation

reduce_domain ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

get_domain

Alternativen

change_domain, rectangle1_domain, add_channels

Siehe auch

full_domain, get_domain, intersection

Modul

Image / region / XLD management

1.2 Format

change_format (Image : ImagePart : Width, Height :)

Ändern der Bildgröße.

change_format vergrößert bzw. verkleinert die Eingabebilder auf die angegebene Höhe bzw. Breite. Bei einer Verkleinerung werden am „rechten“ bzw. „unteren“ Bildrand Teile abgeschnitten. Bei einer Vergrößerung werden die zusätzlichen Bereiche mit 0 besetzt. Der Definitionsbereich des Ausgabebildes entspricht dem des Eingabebildes, eventuell beschnitten auf die neue Bildgröße. Es wird kein Zooming durchgeführt.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **ImagePart** (output_object) ..... image(-array) ⇝ *Hobject*
  Bild mit neuem Format.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des neuen Bildes.
  Defaultwert : 512
  Wertevorschläge : Width ∈ {32, 64, 128, 256, 512, 768, 1024}
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des neuen Bildes.
  Defaultwert : 512
  Wertevorschläge : Height ∈ {32, 64, 128, 256, 512, 525, 1024}

Parallelisierungsinformation

`change_format` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`crop_part`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Image / region / XLD management

`crop_domain ( Image : ImagePart : : )`

Ausschneiden der gültigen Pixel.

`crop_domain` schneidet aus den Eingabebildern einen rechteckigen Ausschnitt aus. Dieses Rechteck ist das kleinste umschließende Rechteck um den Definitionsbereich des Bildes. Der neue Definitionsbereich umfaßt alle Punkte des neuen Bildes. Die neue Bildmatrix hat die Größe des Rechtecks.

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **`ImagePart`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
  Bildausschnitt.

Parallelisierungsinformation

`crop_domain` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`crop_part`, `crop_rectangle1`, `change_format`, `reduce_domain`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Image / region / XLD management

`crop_domain_rel ( Image : ImagePart : Top, Left, Bottom, Right : )`

Ausschneiden eines Bildausschnittes um den Definitionsbereich.

`crop_domain_rel` schneidet aus den Eingabebildern einen rechteckigen Ausschnitt aus. Der Ausschitt wird durch das umschließende Rechteck des Definitionsbereiches bestimmt. Das Rechteck kann durch die Steuerparameter oben (`Top`), links (`Left`), unten (`Bottom`) oder rechts (`Right`) verändert werden. Positive Werte bewirken eine Verkleinerung, negative Zahlen bewirken eine Vergrößerung. Sind alle Parameter gleich Null, dann bleibt das Rechteck unverändert.

Parameter

- ▷ **`Image`** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
  Eingabebild.
- ▷ **`ImagePart`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
  Bildausschnitt.
- ▷ **`Top`** (input_control) . . . . . integer ⇝ *integer*
  Anzahl Zeilen die „oben“ abgeschnitten werden.
  Defaultwert : -1
  Wertevorschläge : `Top` ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

▷ **Left** (input_control) . . . . . integer ⇝ *integer*
Anzahl Spalten die „links“ abgeschnitten werden.
Defaultwert : -1
Wertevorschläge : Left ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

▷ **Bottom** (input_control) . . . . . integer ⇝ *integer*
Anzahl Zeilen die „unten“ abgeschnitten werden.
Defaultwert : -1
Wertevorschläge : Bottom ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

▷ **Right** (input_control) . . . . . integer ⇝ *integer*
Anzahl Zeilen die „rechts“ abgeschnitten werden.
Defaultwert : -1
Wertevorschläge : Right ∈ {-20, -10, -5, -3, -2, -1, 0, 1, 2, 3, 4, 5, 10, 20}

Ergebnis

crop_domain_rel liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system ('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

crop_domain_rel ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

reduce_domain, threshold, connection, regiongrowing, pouring

Alternativen

crop_domain, crop_rectangle1

Siehe auch

smallest_rectangle1, intersection, gen_rectangle1, clip_region

Modul

Image / region / XLD management

crop_part (Image : ImagePart : Row, Column, Width, Height :)

Ausschneiden eines rechteckigen Bildausschnittes.

crop_part schneidet aus den Eingabebildern einen rechteckigen Ausschnitt aus. Der Ausschitt wird durch ein Rechteck angegeben (linkes oberes Eck und Größe). Der Ausschnitt muß innerhalb des Bildes liegen. Der Definitionsbereich umfaßt alle Punkte des neuen Bildes. Die neue Bildmatrix hat die Größe des Rechtecks.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
Eingabebild.

▷ **ImagePart** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
Bildauschnitt.

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex des linken oberen Ecks des Bildauschnittes.
Defaultwert : 100
Wertevorschläge : Row ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich : 0 ≤ Row ≤ 1024

▷ **Column** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltenindex des linken oberen Ecks des Bildauschnittes.
Defaultwert : 100
Wertevorschläge : Column ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich : 0 ≤ Column ≤ 1024

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des neuen Bildes.
Defaultwert : 128
Wertevorschläge : Width ∈ {32, 64, 128, 256, 512, 768}
Typischer Wertebereich : 0 ≤ Width ≤ 1024

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des neuen Bildes.
Defaultwert : 128
Wertevorschläge : Height ∈ {32, 64, 128, 256, 512, 525}
Typischer Wertebereich : 0 ≤ Height ≤ 1024

Parallelisierungsinformation

crop_part ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

crop_rectangle1, crop_domain, change_format, reduce_domain

Siehe auch

zoom_image_size, zoom_image_factor

Modul

Image / region / XLD management

crop_rectangle1 (Image : ImagePart : Row1, Column1, Row2, Column2 :)

Ausschneiden eines rechteckigen Bildausschnittes.

crop_rectangle1 schneidet aus den Eingabebildern einen rechteckigen Ausschnitt aus. Der Ausschitt wird durch ein Rechteck angegeben (linkes oberes und rechtes unteres Eck). Der Ausschnitt muß innerhalb des Bildes liegen. Der Definitionsbereich umfaßt alle Punkte des neuen Bildes. Die neue Bildmatrix hat die Größe des Rechtecks.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
Eingabebild.

▷ **ImagePart** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject*
Bildauschnitt.

▷ **Row1** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex des linken oberen Ecks des Bildauschnittes.
Defaultwert : 100
Wertevorschläge : Row1 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich : 0 ≤ Row1 ≤ 1024

▷ **Column1** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltenindex des linken oberen Ecks des Bildauschnittes.
Defaultwert : 100
Wertevorschläge : Column1 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich : 0 ≤ Column1 ≤ 1024

▷ **Row2** (input_control) . . . . . rectangle.corner.y ⇝ *integer*
Zeilenindex des rechten unteren Ecks des Bildauschnittes.
Defaultwert : 200
Wertevorschläge : Row2 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich : 0 ≤ Row2 ≤ 1024

▷ **Column2** (input_control) . . . . . rectangle.corner.x ⇝ *integer*
Spaltenindex des rechten unteren Ecks des Bildauschnittes.
Defaultwert : 200
Wertevorschläge : Column2 ∈ {10, 20, 50, 100, 200, 300, 500}
Typischer Wertebereich : 0 ≤ Column2 ≤ 1024

Parallelisierungsinformation

`crop_rectangle1` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`crop_part`, `crop_domain`, `change_format`, `reduce_domain`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Image / region / XLD management

`tile_channels` (Image : TiledImage : NumColumns, TileOrder :)

Zusammenfügen von mehreren Bildern zu einem großen Bild.

`tile_channels` fügt ein aus mehreren Kanälen bestehendes Bild durch Kachelung zu einem großen einkanaligen Bild zusammen. Das Eingabebild `Image` besteht aus *Num* Bildern gleicher Größe, die in den einzelnen Kanälen gespeichert werden. Das Ausgabebild `TiledImage` besteht aus einem einzigen Kanal, in dem die *Num* Eingabekanäle in `NumColumns` Spalten gekachelt worden sind. Insbesondere kann `tile_channels` also keine Farbbilder zusammenfügen. Hierzu kann `tile_images` verwendet werden. Der Parameter `TileOrder` legt die Richtung fest, in der die Eingabekanäle in die Ausgabe kopiert werden, falls dies nicht schon durch `NumColumns` festgelegt wird (d.h., falls `NumColumns` != 1 und `NumColumns` != *Num*). Falls `TileOrder` = **’horizontal’**, werden die Bilder in horizontaler Richtung kopiert, d.h. der zweite Kanal des Eingabebildes liegt rechts neben dem ersten Kanal. Falls `TileOrder` = **’vertical’**, werden die Bilder in vertikaler Richtung kopiert, d.h. der zweite Kanal des Eingabebildes liegt unter dem ersten Kanal. Der Definitionsbereich (die Region) des Ausgabebildes `TiledImage` wird berechnet, indem die Region des Eingabebildes `Image` an die entsprechenden Stellen im Ausgabebild kopiert wird. Falls *Num* kein Vielfaches von `NumColumns` ist, hat das Ausgabebild undefinierte Grauwerte in der unteren linken Ecke des Bildes. Die Ausgaberegion ist entsprechend gesetzt.

Parameter

- ▷ **`Image`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte / int1 / cyclic / direction / int2 / int4 / real
  Eingabebild.
- ▷ **`TiledImage`** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Gekacheltes Ausgabegild.
- ▷ **`NumColumns`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Spalten, die für das Ausgabebild verwendet werden
  Defaultwert : 1
  Wertevorschläge : `NumColumns` ∈ {1, 2, 3, 4, 5, 6, 7}
  Restriktion : `NumColumns` ≥ `1`
- ▷ **`TileOrder`** (input_control) ..... string ⇝ *string*
  Reihenfolge der Eingabebilder im Ausgabebild.
  Defaultwert : ’vertical’
  Werteliste : `TileOrder` ∈ {’horizontal’, ’vertical’}

Beispiel

```
/* 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’)
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `tile_channels`

den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`tile_channels` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`append_channel`

Alternativen

`tile_images`, `tile_images_offset`

Siehe auch

`change_format`, `crop_part`, `crop_rectangle1`

Modul

Image / region / XLD management

tile_images (Images : TiledImage : NumColumns, TileOrder :)

Zusammenfügen von mehreren Bildobjekten zu einem großen Bild.

`tile_images` fügt mehrere Bildobjekte, die die gleiche Anzahl von Kanälen besitzen müssen, durch Kachelung zu einem großen Bild zusammen. Das Eingabeobjekt `Images` besteht aus *Num* Bildern, die unterschiedlich groß sein können. Das Ausgabebild `TiledImage` besteht aus einem Bildobjekt, das genauso viele Kanäle besitzt, wie die Eingabebilder. Im Ausgabebild sind die *Num* Eingabekanäle in `NumColumns` Spalten gekachelt. Jede Kachel besitzt dieselbe Größe, die sich aus der maximalen Breite und Höhe der Eingabebilder bestimmt. Falls ein Eingabebild kleiner als die Kachelgröße ist, wird es in die Mitte der entsprechenden Kachel kopiert. Der Parameter `TileOrder` legt die Richtung fest, in der die Eingabekanäle in die Ausgabe kopiert werden, falls dies nicht schon durch `NumColumns` festgelegt wird (d.h., falls `NumColumns` != 1 und `NumColumns` != *Num*). Falls `TileOrder` = **'horizontal'**, werden die Bilder in horizontaler Richtung kopiert, d.h. das zweite Eingabebild liegt rechts neben dem ersten Eingabebild. Falls `TileOrder` = **'vertical'**, werden die Bilder in vertikaler Richtung kopiert, d.h. das zweite Eingabebild liegt unter dem ersten Eingabebild. Der Definitionsbereich (die Region) des Ausgabebildes `TiledImage` wird berechnet, indem die Regionen des Eingabebildes `Images` an die entsprechenden Stellen im Ausgabebild kopiert werden. Falls *Num* kein Vielfaches von `NumColumns` ist, hat das Ausgabebild undefinierte Grauwerte in der unteren linken Ecke des Bildes. Die Ausgaberegion ist entsprechend gesetzt.

Parameter

- ▷ **Images** (input_object) ..... (multichannel-)image-array ⇝ *Hobject* : byte / int1 / cyclic / direction / int2 / int4 / real
  Eingabebilder.
- ▷ **TiledImage** (output_object) .....(multichannel-)image ⇝ *Hobject*
  Gekacheltes Ausgabegild.
- ▷ **NumColumns** (input_control) ..... integer ⇝ *integer*
  Anzahl der Spalten, die für das Ausgabebild verwendet werden
  Defaultwert : 1
  Wertevorschläge : `NumColumns` ∈ {1, 2, 3, 4, 5, 6, 7}
  Restriktion : `NumColumns` ≥ `1`
- ▷ **TileOrder** (input_control) .....string ⇝ *string*
  Reihenfolge der Eingabebilder im Ausgabebild.
  Defaultwert : 'vertical'
  Werteliste : `TileOrder` ∈ {'horizontal', 'vertical'}

Beispiel

```
/* 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')
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `tile_images` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`tile_images` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

`append_channel`

Alternativen

`tile_channels`, `tile_images_offset`

Siehe auch

`change_format`, `crop_part`, `crop_rectangle1`

Modul

Image / region / XLD management

`tile_images_offset` (Images : TiledImage : OffsetRow, OffsetCol, Row1, Col1, Row2, Col2, Width, Height :)

Zusammenfügen von mehreren Bildobjekten zu einem großen Bild mit expliziter Positionsangabe.

`tile_images_offset` fügt mehrere Bildobjekte, die die gleiche Anzahl von Kanälen besitzen müssen, durch Kachelung zu einem großen Bild zusammen. Das Eingabeobjekt `Images` besteht aus *Num* Bildern, die unterschiedlich groß sein können. Das Ausgabebild `TiledImage` besteht aus einem Bildobjekt, das genauso viele Kanäle besitzt, wie die Eingabebilder. Die Größe des Ausgabebildes wird durch die Parameter `Width` und `Height` bestimmt. Die Position der oberen linken Ecke der Eingabebilder im Ausgabebild wird durch die Parameter `OffsetRow` und `OffsetCol` bestimmt. Die beiden Parameter müssen genau *Num* Werte enthalten. Optional kann jedes Eingabebild auf ein beliebiges Rechteck, das kleiner als das Eingabebild ist, zugeschnitten werden. Hierzu sind die Parameter `Row1`, `Col1`, `Row2` und `Col2` entsprechend zu setzen. Wird irgendeiner dieser vier Parameter auf **-1** gesetzt, wird das entsprechende Eingabebild nicht beschnitten. In jedem Fall müssen alle vier Parameter *Num* Werte enthalten. Falls die Eingabebilder beschnitten werden, beziehen sich die Positionsparameter `OffsetRow` und `OffsetCol` auf die obere linke Ecke des beschnittenen Bildes. Falls sich die Eingabebilder im Ausgabebild überlappen (unter der Beachtung des jeweiligen Definitionsbereiches), so überschreibt ein Bild mit höherem Index in `Images` die Bilddaten eines vorhergehenden Bildes. Der Definitionsbereich (die Region) des Ausgabebildes `TiledImage` wird berechnet, indem die Regionen des Eingabebildes `Images` an die entsprechenden Stellen im Ausgabebild kopiert werden.

Achtung

Falls die Bilder alle gleich groß sind und sich genau aneinanderfügen, ist der Operator `tile_images` normalerweise etwas schneller.

Parameter

- ▷ **`Images`** (input_object) ..... (multichannel-)image-array ⇝ *Hobject* : byte / int1 / cyclic / direction / int2 / int4 / real
  Eingabebilder.
- ▷ **`TiledImage`** (output_object) ..... (multichannel-)image ⇝ *Hobject*
  Gekacheltes Ausgabegild.
- ▷ **`OffsetRow`** (input_control) ..... point.y-array ⇝ *integer*
  Zeilenkoordinate der linken oberen Ecke der Eingabebilder im Ausgabebild.
  Defaultwert : 0
  Wertevorschläge : `OffsetRow` ∈ {0, 50, 100, 150, 200, 250}

▷ **OffsetCol** (input_control) . . . . . point.x-array ⇝ *integer*
Spaltenkoordinate der linken oberen Ecke der Eingabebilder im Ausgabebild.
Defaultwert : 0
Wertevorschläge : OffsetCol ∈ {0, 50, 100, 150, 200, 250}

▷ **Row1** (input_control) . . . . . rectangle.origin.y-array ⇝ *integer*
Zeilenkoordinate der oberen linken Ecke des zu kopierenden Teils des jeweiligen Bildes.
Defaultwert : -1
Wertevorschläge : Row1 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}

▷ **Col1** (input_control) . . . . . rectangle.origin.x-array ⇝ *integer*
Spaltenkoordinate der oberen linken Ecke des zu kopierenden Teils des jeweiligen Bildes.
Defaultwert : -1
Wertevorschläge : Col1 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}

▷ **Row2** (input_control) . . . . . rectangle.corner.y-array ⇝ *integer*
Zeilenkoordinate der rechten unteren Ecke des zu kopierenden Teils des jeweiligen Bildes.
Defaultwert : -1
Wertevorschläge : Row2 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}

▷ **Col2** (input_control) . . . . . rectangle.corner.x-array ⇝ *integer*
Spaltenkoordinate der rechten unteren Ecke des zu kopierenden Teils des jeweiligen Bildes.
Defaultwert : -1
Wertevorschläge : Col2 ∈ {-1, 0, 10, 20, 50, 100, 200, 300, 500}

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Ausgabeildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {32, 64, 128, 256, 512, 768, 1024, 2048, 4096}

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Ausgabeildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {32, 64, 128, 256, 512, 525, 1024, 2048, 4096}

Beispiel

```
/* Grab 2 (multi-channel) NTSC images, crop the bottom 5 lines off  */
/* of each image, the right 5 columns off of the first image, and   */
/* the left five lines off 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, FGHandle, -1)
    concat_obj (Images, ImageGrabbed, Images)
endfor
tile_images_offset (Images, TiledImage, [0,635], [0,0], [0,0],
                    [0,5], [474,474], [634,639])
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert tile_images_offset den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

tile_images_offset ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

append_channel

Alternativen

tile_channels, tile_images

Siehe auch

change_format, crop_part, crop_rectangle1

Modul

Image / region / XLD management

1.3 Framegrabber

close_all_framegrabbers (: : :)

Schließen aller Framegrabber.

close_all_framegrabbers schließt alle derzeit geöffneten Framegrabber. Dieser Operator ist insbesondere bei Verklemmungen nützlich, wenn Framegrabber-Handles beschädigt wurden und daher ein Framegrabber nicht mehr zugänglich ist, also auch kein gezieltes close_framegrabber mehr möglich ist.

Achtung

Da alle Framegrabber geschlossen werden, sind nach Aufruf von close_all_framegrabbers alle Framegrabber-Handles ungültig.

Ergebnis

Gelingt es, die Framegrabber zu schließen, liefert close_all_framegrabbers den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_all_framegrabbers ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

grab_image

Siehe auch

open_framegrabber

Modul

Image / region / XLD management

close_framegrabber (: : FGHandle :)

Schließen des angegebenen Framegrabbers.

close_framegrabber schließt den durch FGHandle spezifizierten Framegrabber. Dabei wird insbesondere etwaiger Speicherplatz für Datenpuffer freigegeben und der Framegrabber wieder für andere Prozesse zugänglich gemacht.

Parameter

▷ **FGHandle** (input_control) ..... framegrabber ⇝ *integer*
Zu schließender Framegrabber.

Ergebnis

Ist der Framegrabber geöffnet, liefert close_framegrabber den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_framegrabber ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

grab_image

Siehe auch

open_framegrabber

Modul

Image / region / XLD management

get_framegrabber_lut (: : FGHandle : ImageRed, ImageGreen, ImageBlue)

Abfragen der Framegrabber Lut (lookuptable).

get_framegrabber_lut fragt die Lut des mit FGHandle spezifizierten Framegrabbers ab. Diese Operation ist nicht für alle Framegrabber sinnvoll.

Parameter

- ▷ **FGHandle** (input_control) ..... framegrabber ⇝ *integer*
  Zu bearbeitender Framegrabber.
- ▷ **ImageRed** (output_control) ..... integer-array ⇝ *integer*
  Rotanteil der Lut-Einträge.
- ▷ **ImageGreen** (output_control) ..... integer-array ⇝ *integer*
  Grünanteil der Lut-Einträge.
- ▷ **ImageBlue** (output_control) ..... integer-array ⇝ *integer*
  Blauanteil der Lut-Einträge.

Ergebnis

get_framegrabber_lut liefert den Wert 2 (H_MSG_TRUE), falls der Framegrabber geöffnet ist.

Parallelisierungsinformation

get_framegrabber_lut ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber

Mögliche Nachfolgerfunktionen

set_framegrabber_lut

Siehe auch

set_framegrabber_lut, open_framegrabber

Modul

Image / region / XLD management

get_framegrabber_param (: : FGHandle, Param : Value)

Abfrage spezieller Parameter fuer einen Framegrabber.

get_framegrabber_param liefert spezielle Parameterwerte für den durch FGHandle angegebenen Framegrabber. Die unten aufgelisteten Standardparameter sind dabei für jeden beliebigen Framegrabber abrufbar. Darüberhinaus werden unter Umständen noch weitere Parameter von einem spezifischen Framegrabber unterstützt. Eine Liste dieser zusätzlichen Parameter ist mit der Query **’parameter’** vermöge info_framegrabber abrufbar.

Standardwerte für Param, vgl. open_framegrabber:

’name’ Name des Framegrabbers.

’horizontal_resolution’ Horizontale Auflösung des Framegrabbers.

’vertical_resolution’ Vertikale Auflösung des Framegrabbers.

’image_width’ Breite des Bildausschnittes.

’image_height’ Höhe des Bildausschnittes.

’start_row’ Zeilennummer der oberen linken Ecke des Bildausschnittes.

’start_column’ Spaltennummer der oberen linken Ecke des Bildausschnittes.

’field’ Selektiertes Halbbild bzw. Vollbild.

’bits_per_channel’ Anzahl übertragener Bits pro Pixel und Bildkanal.

’color_space’ Farbraum des resultierenden Bildes.

’gain’ Verstärkungsfaktor für Video-Verstärker.

’external_trigger’ Triggerung durch externes Signal (’true’ / ’false’).

’camera_type’ Verwendeter Kameratyp (Framegrabber-spezifisch).

’device’ Device, an das der Framegrabber angeschlossen ist.

’port’ Port des Framegrabbers, an den das Videosignal angeschlossen ist.

’line_in’ Kameraeingang (falls Multiplexer verfügbar).

Parameter

- ▷ **FGHandle** (input_control) . . . . . framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.
- ▷ **Param** (input_control) . . . . . string(-array) ⇝ *string*
  Abzufragender Parameter.
  Defaultwert : 'revision'
  Wertevorschläge : Param ∈ {'name', 'horizontal_resolution', 'vertical_resolution', 'image_width', 'image_height', 'start_row', 'start_column', 'field', 'bits_per_channel', 'color_space', 'gain', 'external_trigger', 'camera_type', 'device', 'port', 'line_in', 'grab_timeout', 'volatile', 'revision'}
- ▷ **Value** (output_control) . . . . . string(-array) ⇝ *string* / real / integer
  Parameterwert.

Ergebnis

Ist der Framegrabber geöffnet und wird der angegebene Parameter vom Framegrabber unterstützt, liefert get_framegrabber_param den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_framegrabber_param ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber

Mögliche Nachfolgerfunktionen

grab_image, grab_region, grab_image_start, grab_image_async, grab_region_async, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Image / region / XLD management

grab_image (: Image : FGHandle :)

Einzug eines Bildes vom angegebenen Framegrabber.

grab_image zieht über den durch FGHandle spezifizierten Framegrabber ein Bild ein. Der gewünschte Betriebsmodus des Framegrabbers sowie ein passender Bildausschnitt kann mittels open_framegrabber eingestellt werden. Weitere (Framegrabber-spezifische) Einstellungen können gegebenenfalls mit set_framegrabber_param vorgenommen werden.

Parameter

- ▷ **Image** (output_object) . . . . . image ⇝ *Hobject* : byte / int2
  Eingezogenes Bild.
- ▷ **FGHandle** (input_control) . . . . . framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.

Beispiel

```
// 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_image(Img,FgHandle)
close_framegrabber(FgHandle).
```

Ergebnis

Ist der Framegrabber geöffnet, liefert grab_image den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`grab_image` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_framegrabber`, `grab_image_start`

Mögliche Nachfolgerfunktionen

`grab_image`, `grab_image_start`, `grab_image_async`, `close_framegrabber`

Siehe auch

`open_framegrabber`, `info_framegrabber`, `set_framegrabber_param`

Modul

Image / region / XLD management

`grab_image_async` (: Image : FGHandle, MaxDelay :)

Einzugs eines Bildes vom angegebenen Framegrabber und asynchroner Start des nächsten Einzuges.

`grab_image_async` liest ein Bild vom durch `FGHandle` spezifizierten Framegrabber ein und startet den asynchronen Einzug des nächsten Bildes. Der gewünschte Betriebsmodus des Framegrabbers sowie ein passender Bildausschnitt kann mittels `open_framegrabber` eingestellt werden. Weitere (Framegrabber-spezifische) Einstellungen können gegebenenfalls mit `set_framegrabber_param` vorgenommen werden. Der Abschluß des neu gestarteten Bildeinzuges erfolgt mittels `grab_image` oder neuerlich `grab_image_async`. Sind mehr als `MaxDelay` ms seit dem Start des Einzuges vergangen, wird ein neues Bild eingezogen. Ein negativer Wert für `MaxDelay` deaktiviert diesen Mechanismus.

Parameter

- ▷ **`Image`** (output_object) ..... image ⇝ *Hobject* : byte / int2
  Eingezogenes Bild.
- ▷ **`FGHandle`** (input_control) ..... framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.
- ▷ **`MaxDelay`** (input_control) ..... number ⇝ *real*
  Maximal erlaubte Verzoegerung zwischen Start des Einzugs und Auslesen des Bildes [ms].
  Defaultwert : -1.0
  Wertevorschläge : `MaxDelay` ∈ {-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9}

Beispiel

```
// 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 image + start next grab
grab_image_async(Img,FgHandle,-1.0)
// Process Img ...
// Finish asynchronous grab + start next grab
grab_image_async(Img,FgHandle,-1.0)
close_framegrabber(FgHandle).
```

Ergebnis

Ist der Framegrabber geöffnet und unterstützt er den asynchronen Bildeinzug, liefert `grab_image_async` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`grab_image_async` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_framegrabber`

Mögliche Nachfolgerfunktionen

`grab_image`, `grab_image_async`, `close_framegrabber`

Siehe auch

grab_image_start, open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Image / region / XLD management

grab_image_start (: : FGHandle, MaxDelay :)

Start des asynchronen Einzugs eines Bildes vom angegebenen Framegrabber.

grab_image_start startet den asynchronen Bildeinzug über den durch FGHandle spezifizierten Framegrabber. Der gewünschte Betriebsmodus des Framegrabbers sowie ein passender Bildausschnitt kann mittels open_framegrabber eingestellt werden. Weitere (Framegrabber-spezifische) Einstellungen können gegebenenfalls mit set_framegrabber_param vorgenommen werden. Der Abschluß des Bildeinzuges erfolgt dann mittels grab_image, grab_image_async, grab_region oder grab_region_async. Sind dabei mehr als MaxDelay ms seit dem Start des Einzuges vergangen, wird ein neues Bild eingezogen. Ein negativer Wert für MaxDelay deaktiviert diesen Mechanismus.

Parameter

- ▷ **FGHandle** (input_control) . . . . . framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.
- ▷ **MaxDelay** (input_control) . . . . . number ⇝ *real*
  Maximal erlaubte Verzoegerung zwischen Start des Einzugs und Auslesen des Bildes [ms].
  Defaultwert : -1.0
  Wertevorschläge : MaxDelay ∈ {20.0, 33.3, 40.0, 66.6, 80.0, 99.9}

Beispiel

```
// 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_image(Img,FgHandle)
// Start next grab
grab_image_start(FgHandle,-1.0)
// Process Img ...
// Finish asynchronous grab
grab_image(Img,FgHandle)
close_framegrabber(FgHandle).
```

Ergebnis

Ist der Framegrabber geöffnet und unterstützt er den asynchronen Bildeinzug, liefert grab_image_start den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

grab_image_start ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber

Mögliche Nachfolgerfunktionen

grab_image, grab_region, grab_image_async, grab_region_async, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Image / region / XLD management

grab_region (: Region : FGHandle :)

Einzug und Segmentation eines Bildes vom angegebenen Framegrabber.

grab_region zieht über den durch FGHandle spezifizierten Framegrabber ein Bild ein und segmentiert es. Der gewünschte Betriebsmodus des Framegrabbers sowie ein passender Bildausschnitt kann mittels open_framegrabber eingestellt werden. Weitere (Framegrabber-spezifische) Einstellungen können gegebenenfalls mit set_framegrabber_param vorgenommen werden. Die gefundenen Segmente werden in Region zurückgeliefert.

Parameter

- ▷ **Region** (output_object) ..... region(-array) ⇝ *Hobject*
  Eingezogenes, segmentiertes Bild: Region(en).
- ▷ **FGHandle** (input_control) ..... framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.

Beispiel

```
// 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).
```

Ergebnis

Ist der Framegrabber geöffnet, liefert grab_region den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

grab_region ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber, grab_image_start

Mögliche Nachfolgerfunktionen

grab_region, grab_region_async, grab_image_start, grab_image, grab_image_async, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Image / region / XLD management

grab_region_async (: Region : FGHandle, MaxDelay :)

Einzug und Segmentation eines Bildes vom angegebenen Framegrabber sowie asynchroner Start des nächsten Einzuges.

grab_region_async zieht über den durch FGHandle spezifizierten Framegrabber ein Bild ein, segmentiert es und startet den asynchronen Einzug des nächsten Bildes. Der gewünschte Betriebsmodus des Framegrabbers sowie ein passender Bildausschnitt kann mittels open_framegrabber eingestellt werden. Weitere (Framegrabber-spezifische) Einstellungen können gegebenenfalls mit set_framegrabber_param vorgenommen werden. Die gefundenen Segmente werden in Region zurückgeliefert. Der Abschluß des neu gestarteten Bildeinzuges erfolgt mittels grab_region oder neuerlich grab_region_async. Sind mehr als MaxDelay ms seit dem Start des Einzuges vergangen, wird ein neues Bild eingezogen. Ein negativer Wert für MaxDelay deaktiviert diesen Mechanismus.

Parameter

- ▷ **Region** (output_object) ..... region(-array) ⇝ *Hobject*
  Eingezogenes, segmentiertes Bild: Region(en).
- ▷ **FGHandle** (input_control) ..... framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.
- ▷ **MaxDelay** (input_control) ..... number ⇝ *real*
  Maximal erlaubte Verzoegerung zwischen Start des Einzugs und Auslesen des Bildes [ms].
  Defaultwert : -1.0
  Wertevorschläge : MaxDelay ∈ {-1.0, 20.0, 33.3, 40.0, 66.6, 80.0, 99.9}

Beispiel

```
// 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 image, segment it, and start next grab
grab_region_async(Region,FgHandle,-1.0)
// Process Region ...
// Finish asynchronous grab, segment this image, and start next grab
grab_region_async(Region,FgHandle,-1.0)
close_framegrabber(FgHandle).
```

Ergebnis

Ist der Framegrabber geöffnet, liefert grab_region_async den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

grab_region_async ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber, grab_image_start

Mögliche Nachfolgerfunktionen

grab_region, grab_region_async, grab_image_start, grab_image, grab_image_async, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, set_framegrabber_param

Modul

Image / region / XLD management

info_framegrabber (: : Name, Query : Information, ValueList)

Ausgabe von Informationen zum angegebenen Framegrabber.

info_framegrabber liefert Informationen zum angegebenen Framegrabber Name. Über den Parameter Query wird die gewünschte Information ausgewählt. In Information wird dann eine textuelle Beschreibung, in ValueList gegebenenfalls eine Liste von zulässigen Werten zurückgegeben. Derzeit werden folgende Abfragen unterstützt:

’camera_types’: Beschreibung des Framegrabber-spezifischen Parameters ’CameraType’, vgl. open_framegrabber.

’defaults’: Framegrabberspezifische Defaultwerte in ValueList, vgl. open_framegrabber.

’general’: Allgemeine Angaben in Information.

’info_boards’: Informationen über die tatsächlich im Rechner installierten Framegrabberkarten, angeschlossene Ports etc. Diese Daten werden vor allem für den Auto-detect-Mechanismus der ActivVisionTools verwendet.

'parameters': Auflistung aller Framegrabberspezifische Parameter, die über `set_framegrabber_param` und `get_framegrabber_param` zugänglich sind.

'ports': Beschreibung der Ports (Signal, Farbtiefe, Stecker etc.) in `Information` und die Portnummern in `ValueList`.

'revision': Versionsnummer des Framegrabber-Interfaces.

Weitere Infomationen zu ausgewählte Framegrabbern finden sich in Beschreibungsdateien im Verzeichnis **doc/html/manuals**.

Parameter

- ▷ **`Name`** (input_control) . . . . . string ⇝ *string*
  Interessierender Framegrabber.
  Defaultwert : 'File'
  Wertevorschläge : `Name` ∈ {'Barracuda', 'Bcam1394', 'BitFlow', 'CCi4', 'DFG-BW', 'DFG-LC', 'DirectShow', 'DqVll', 'DT315x', 'DT3162', 'File', 'Fire-i', 'FirePackage', 'FlashBus', 'FlashBusMX', 'Ginga', 'Ginga++', 'IDS', 'Inspecta', 'MatrixVision', 'Meteor1', 'MultiCam', 'Opteon', 'PCEye', 'PicPort', 'PicPortPro', 'PicProdigy', 'PX', 'PXC', 'PXD', 'PXR', 'Ramses1', 'TWAIN', 'VideoPort'}
- ▷ **`Query`** (input_control) . . . . . string ⇝ *string*
  Name der gewählten Anfrage.
  Defaultwert : 'info_boards'
  Werteliste : `Query` ∈ {'camera_types', 'defaults', 'general', 'info_boards', 'parameters', 'ports', 'revision'}
- ▷ **`Information`** (output_control) . . . . . string ⇝ *string*
  Textuelle Beschreibung (abhängig von `Query`).
- ▷ **`ValueList`** (output_control) . . . . . string(-array) ⇝ *string* / integer / real
  Gegebenenfalls eine Werteliste (abhängig von `Query`).

Beispiel

```
// 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_image(Img,FgHandle)
close_framegrabber(FgHandle).
```

Ergebnis

Sind die Parameterwerte korrekt und ist der gewünschte Framegrabber zum Aufrufzeitpunkt verfügbar, liefert `info_framegrabber` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`info_framegrabber` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`open_framegrabber`

Mögliche Nachfolgerfunktionen

`open_framegrabber`

Siehe auch

`open_framegrabber`

Modul

Image / region / XLD management

`open_framegrabber` `( : : Name, HorizontalResolution, VerticalResolution, ImageWidth, ImageHeight, StartRow, StartColumn, Field, BitsPerChannel, ColorSpace, Gain, ExternalTrigger, CameraType, Device, Port, LineIn : FGHandle )`

Öffnen und Konfigurieren eines Framegrabbers.

open_framegrabber öffnet und konfiguriert den gewählten Framegrabber. Dabei wird insbesondere auch die Verbindung zum Framegrabber getestet, der Framegrabber (normalerweise) für andere Prozesse gesperrt und gegebenenfalls Speicher als Datenpuffer reserviert. Der eigentliche Bildeinzug erfolgt dann mittels grab_image, grab_region, grab_image_async oder grab_region_async. Wird der Framegrabber nicht mehr benötigt, sollte er mittels close_framegrabber wieder geschlossen und so für andere Prozesse freigegeben werden. Dies geschieht bei den meisten Framegrabbern automatisch vor dem erneuten Öffnen dieses Framegrabbers. Einige Framegrabber erlauben das gleichzeitige Öffnen mehrerer Instanzen (bis zu 5).

Für alle Parameter kann explizit der Framegrabber-spezifische default-Wert verwendet werden (vgl. Parameterbeschreibung unten). Nähere Informationen zu einem bestimmten Framegrabber erhält man mit Hilfe der Routine info_framegrabber. Außerdem finden sich im Verzeichnis **doc/html/manuals** Beschreibungsdateien für einige ausgewählte Framegrabber.

Die Bedeutung der Parameter ist im einzelnen:

HorizontalResolution, VerticalResolution Gewünschte Auflösung des Framegrabbers.

ImageWidth, ImageHeight Größe des Bildausschnittes, der von grab_image etc. geliefert werden soll.

StartRow, StartColumn Linke obere Ecke des gewünschten Bildausschnittes.

Field Gewünschtes Halbbild (**'first'**, **'second'** oder **'next'**) bzw. Einstellung eines Vollbildes.

BitsPerChannel Die Anzahl an Bits, die der Framegrabber pro Pixel und Bildkanal überträgt (typischerweise 5, 8, 10, 12 oder 16 Bits).

ColorSpace Festlegung, ob Einzug von einkanaligen ('gray') bzw. dreikanaligen Bildern ('rgb', 'yuv', ...).

Gain Verstärkungsfaktor für den Video-Verstärker (soweit unterstützt).

ExternalTrigger Aktivierung der externen Triggerung des Framegrabbers (soweit unterstützt).

CameraType Ein generischer Parameter (vom Typ string) mit Framegrabber-spezifischer Bedeutung. Diese ist mittels info_framegrabber abfragbar.

Device Devicename der Framegrabberkarte.

Port Port des Framegrabbers, an dem das Videosignal anliegt.

LineIn Selektion des Kameraeingangs (falls ein Multiplexer für den Port verfügbar ist).

Der Operator open_framegrabber liefert mit FGHandle einen Schlüssel auf den geöffneten Framegrabber zurück.

Achtung

Bedingt durch die Vielzahl der unterstützten Framegrabber umfaßt open_framegrabber eine große Anzahl von Parametern. Für einen bestimmten Framegrabber werden deshalb nicht immer alle Parameter benötigt.

Parameter

▷ **Name** (input_control) . . . . . string ⇝ *string*
HALCON Framegrabber-Interface, d.h. Name der zugehörigen DLL (Windows) bzw. 'shared library' (UNIX).
Defaultwert : 'File'
Wertevorschläge : Name ∈ {'Barracuda', 'Bcam1394', 'BitFlow', 'CCi4', 'DFG-BW', 'DFG-LC', 'DirectShow', 'DqVll', 'DT315x', 'DT3162', 'File', 'Fire-i', 'FirePackage', 'FlashBus', 'FlashBusMX', 'Ginga', 'Ginga++', 'IDS', 'Inspecta', 'MatrixVision', 'Meteor1', 'MultiCam', 'Opteon', 'PCEye', 'PicPort', 'PicPortPro', 'PicProdigy', 'PX', 'PXC', 'PXD', 'PXR', 'Ramses1', 'TWAIN', 'VideoPort'}

▷ **HorizontalResolution** (input_control) . . . . . extent.x ⇝ *integer*
Gewünschte horizontale Auflösung des Framegrabbers (1: Vollauflösung, 2: Halbe Auflösung, 4: Viertel Auflösung).
Defaultwert : 1
Wertevorschläge : HorizontalResolution ∈ {1, 2, 4, 768, 720, 640, 384, 320, 192, 160}

▷ **VerticalResolution** (input_control) . . . . . extent.y ⇝ *integer*
Gewünschte vertikale Auflösung des Framegrabbers (1: Vollauflösung, 2: Halbe Auflösung, 4: Viertel Auflösung).
Defaultwert : 1
Wertevorschläge : VerticalResolution ∈ {1, 2, 4, 576, 480, 288, 240, 144, 120}

▷ **ImageWidth** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des gewünschten Bildausschnittes (0: HorizontalResolution - 2*StartColumn).
Defaultwert : 0
Wertevorschläge : ImageWidth ∈ {-1, 0}

▷ **ImageHeight** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des gewünschten Bildausschnittes (0: VerticalResolution - 2*StartRow).
Defaultwert : 0
Wertevorschläge : ImageHeight ∈ {-1, 0}

▷ **StartRow** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilennummer der oberen linken Ecke des gewünschten Bildausschnittes, bei ImageHeight = 0: Höhe eines Rahmens.
Defaultwert : 0
Wertevorschläge : StartRow ∈ {-1, 0}

▷ **StartColumn** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
Spaltennummer der oberen linken Ecke des gewünschten Bildausschnittes, bei ImageWidth = 0: Breite eines Rahmens.
Defaultwert : 0
Wertevorschläge : StartColumn ∈ {-1, 0}

▷ **Field** (input_control) . . . . . string ⇝ *string*
Gewünschtes Halbbild bzw. Vollbild.
Defaultwert : 'default'
Wertevorschläge : Field ∈ {'first', 'second', 'next', 'interlaced', 'progressive', 'default'}

▷ **BitsPerChannel** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl übertragener Bits pro Pixel und Bildkanal (-1: Framegrabber-spezifischer Defaultwert).
Defaultwert : -1
Wertevorschläge : BitsPerChannel ∈ {5, 8, 10, 12, 16, -1}

▷ **ColorSpace** (input_control) . . . . . string(-array) ⇝ *string*
Festlegung, ob Einzug von einkanaligen ('gray') bzw. dreikanaligen ('rgb', 'yuv', ...) Bildern ('default': Framegrabber-spezifischer Defaultwert, meist 'rgb').
Defaultwert : 'default'
Wertevorschläge : ColorSpace ∈ {'gray', 'rgb', 'yuv', 'default'}

▷ **Gain** (input_control) . . . . . real ⇝ *real*
Verstärkungsfaktor für Video-Verstärker (-1.0: Framegrabber-spezifischer Defaultwert).
Defaultwert : -1.0
Wertevorschläge : Gain ∈ {0.25, 0.5, 0.75, 1.0, -1.0}

▷ **ExternalTrigger** (input_control) . . . . . string ⇝ *string*
Triggerung durch externes Signal.
Defaultwert : 'default'
Werteliste : ExternalTrigger ∈ {'true', 'false', 'default'}

▷ **CameraType** (input_control) . . . . . string(-array) ⇝ *string*
Verwendeter Kameratyp (Framegrabber-spezifisch) ('default': Framegrabber-spezifischer Defaultwert).
Defaultwert : 'default'
Wertevorschläge : CameraType ∈ {'ntsc', 'pal', 'auto', 'default'}

▷ **Device** (input_control) . . . . . string(-array) ⇝ *string*
Device, an das der Framegrabber angeschlossen ist ('default': Framegrabber-spezifischer Defaultwert).
Defaultwert : 'default'
Wertevorschläge : Device ∈ {'-1', '0', '1', '2', '3', 'default'}

▷ **Port** (input_control) . . . . . integer(-array) ⇝ *integer*
Port des Framegrabbers, an den das Videosignal angeschlossen ist (-1: Framegrabber-spezifischer Defaultwert).
Defaultwert : -1
Wertevorschläge : Port ∈ {0, 1, 2, 3, -1}

▷ **LineIn** (input_control) . . . . . integer(-array) ⇝ *integer*
Kameraeingang (falls Multiplexer für den Port verfügbar.
Defaultwert : -1
Wertevorschläge : LineIn ∈ {1, 2, 3, 4, -1}

▷ **FGHandle** (output_control) . . . . . framegrabber ⇝ *integer*
Neu geöffneter Framegrabber.

Beispiel

```
// 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_image(Img,FgHandle)
close_framegrabber(FgHandle).
```

Ergebnis

Sind die Parameterwerte korrekt und ist der gewünschte Framegrabber zum Aufrufzeitpunkt verfügbar, liefert open_framegrabber den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

open_framegrabber ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

info_framegrabber

Mögliche Nachfolgerfunktionen

grab_image, grab_region, grab_image_start, grab_image_async, grab_region_async, set_framegrabber_param

Siehe auch

info_framegrabber, close_framegrabber, grab_image

Modul

Image / region / XLD management

set_framegrabber_lut (: : FGHandle, ImageRed, ImageGreen, ImageBlue :)

Setzen der Framegrabber Lut (lookuptable).

set_framegrabber_lut setzt die Lut des durch FGHandle spezifizierten Framegrabbers. Diese Operation ist nicht für alle Framegrabber sinnvoll.

Parameter

- ▷ **FGHandle** (input_control) ..... framegrabber ⇝ *integer*
  Zu bearbeitender Framegrabber.
- ▷ **ImageRed** (input_control) ..... integer-array ⇝ *integer*
  Rotanteil der Lut-Einträge.
- ▷ **ImageGreen** (input_control) ..... integer-array ⇝ *integer*
  Grünanteil der Lut-Einträge.
- ▷ **ImageBlue** (input_control) ..... integer-array ⇝ *integer*
  Blauanteil der Lut-Einträge.

Ergebnis

set_framegrabber_lut liefert den Wert 2 (H_MSG_TRUE), sofern die übergebene Lut korrekt ist und der Framegrabber geöffnet ist.

Parallelisierungsinformation

set_framegrabber_lut ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber, get_framegrabber_lut

Mögliche Nachfolgerfunktionen

grab_image, grab_region, grab_image_start, grab_image_async, grab_region_async

Siehe auch

get_framegrabber_lut, open_framegrabber

Modul

Image / region / XLD management

set_framegrabber_param (: : FGHandle, Param, Value :)

Setzen spezieller Parameter fuer einen Framegrabber.

set_framegrabber_param setzt spezifische Parameter für den durch FGHandle angegebenen Framegrabber.

Parameter

- ▷ **FGHandle** (input_control) . . . framegrabber ⇝ *integer*
  Zu benutzender Framegrabber.
- ▷ **Param** (input_control) . . . string(-array) ⇝ *string*
  Zu setzender Parameter.
  Wertevorschläge : Param ∈ {'continuous_grabbing', 'external_trigger', 'gain', 'image_height', 'image_width', 'port', 'start_column', 'start_row', 'trigger_signal', 'volatile'}
- ▷ **Value** (input_control) . . . string(-array) ⇝ *string* / real / integer
  Zu setzender Parameterwert.

Ergebnis

Ist der Framegrabber geöffnet und wird der angegebene Parameter bzw. Parameterwert vom Framegrabber unterstützt, liefert set_framegrabber_param den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_framegrabber_param ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_framegrabber

Mögliche Nachfolgerfunktionen

grab_image, grab_region, grab_image_start, grab_image_async, grab_region_async, close_framegrabber

Siehe auch

open_framegrabber, info_framegrabber, get_framegrabber_param

Modul

Image / region / XLD management

1.4 Generierung

copy_image (Image : DupImage : :)

Physikalisches Kopieren eines Bildes.

copy_image kopiert den ersten Kanal des Eingabebildes in eine neues (einkanaliges) Bild mit gleichem Definitionsbereich wie das Eingabebild. Im Gegensatz zu HALCON-Prozeduren wie copy_obj, wird hierbei eine physikalische Kopie der Matrix angelegt. Diese kann z.B. dazu verwendet werden, um die Grauwerte zu modifizieren (siehe get_image_pointer1).

Parameter

- ▷ **Image** (input_object) . . . image ⇝ *Hobject*
  Bild das kopiert werden soll.
- ▷ **DupImage** (output_object) . . . image ⇝ *Hobject*
  Kopiertes Bild.

Parallelisierungsinformation

copy_image ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image, gen_image_const

Mögliche Nachfolgerfunktionen

set_grayval, get_image_pointer1

Alternativen

set_grayval, paint_gray, gen_image_const, gen_image_proto

Siehe auch

`get_image_pointer1`

Modul

Basic operators

gen_image1 (: Image : Type, Width, Height, PixelPointer :)

Erzeugen eines Bildes aus einem Zeiger auf die Pixel.

`gen_image1` erzeugt ein Bild in der Größe `Width` × `Height`. Die Pixel in `PixelPointer` sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel (`PixelPointer`) muß mit (`Type`) übereinstimmen. Der Speicher für das neue Bild wird von HALCON neu angelegt. Der Speicherplatz auf den `PixelPointer` kann also nach dem Aufruf freigegeben werden. Da der Typ des Parameters `PixelPointer` generisch ist (long), muß ein Cast bei dem Aufruf verwendet werden.

Parameter

- ▷ **`Image`** (output_object) ..... image ⇝ *Hobject*
  Erzeugtes Bild mit neuer Bildmatrix.
- ▷ **`Type`** (input_control) ..... string ⇝ *string*
  Pixeltyp.
  Defaultwert : 'byte'
  Werteliste : `Type` ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic'}
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Breite des Bildes.
  Defaultwert : 512
  Wertevorschläge : `Width` ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ `Width` ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : `Width` ≥ `1`
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Bildes.
  Defaultwert : 512
  Wertevorschläge : `Height` ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ `Height` ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : `Height` ≥ `1`
- ▷ **`PixelPointer`** (input_control) ..... integer ⇝ *integer*
  Zeiger auf den ersten Grauwert.

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char  image[768*525];
  int            r,c;
  for (r=0; r<525; r++)
    for (c=0; c<768; c++)
      image[r*768+c] = c % 255;
  gen_image1(new,"byte",768,525,(long)image);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image1` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_image1` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_image_const, get_image_pointer1

Alternativen

gen_image3, gen_image_const, get_image_pointer1

Siehe auch

reduce_domain, paint_gray, paint_region, set_grayval

Modul

Image / region / XLD management

gen_image1_extern (: Image : Type, Width, Height, PixelPointer, ClearProc :)

Erzeugen eines Bildes aus einem Zeiger auf Pixel (mit Speicherverwaltung).

gen_image1_extern erzeugt ein Bild in der Größe Width × Height. Die Pixel in PixelPointer sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel (PixelPointer) muß mit (Type) übereinstimmen. Da der Typ des Parameters PixelPointer generisch ist (long), muß ein Cast bei dem Aufruf verwendet werden.

Der Speicher für das neue Bild wird von HALCON im Gegensatz zu gen_image1 nicht neu angelegt und damit auch nicht kopiert. Der Speicherplatz auf den PixelPointer zeigt, muß also beim Löschen des Objektes Image freigegeben werden. Dies geschieht durch die, vom Aufrufer zur Verfügung gestellte, Prozedur ClearProc. Diese Prozedur muß folgende Signatur haben:

void ClearProc(void* ptr);

Sie wird beim Löschen von Image aufgerufen. Falls der Speicher nicht freigegeben werden soll (im Fall von Framegrabbern oder statischem Speicher) kann eine Prozedur „ohne Rumpf“ oder der NULL-Pointer übergeben werden. Analog zum Parameter PixelPointer muß der Zeiger auf die Prozedur mit einem Cast auf den Typ (long) übergeben werden.

Parameter

- **Image** (output_object) ..... image ⇝ *Hobject*
  Erzeugtes HALCON-Bild.
- **Type** (input_control) ..... string ⇝ *string*
  Pixeltyp.
  Defaultwert : 'byte'
  Werteliste : Type ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic'}
- **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Bildes.
  Defaultwert : 512
  Wertevorschläge : Width ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ Width ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : Width ≥ 1
- **Height** (input_control) ..... extent.y ⇝ *integer*
  Height of image.
  Defaultwert : 512
  Wertevorschläge : Height ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ Height ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : Height ≥ 1
- **PixelPointer** (input_control) ..... integer ⇝ *integer*
  Zeiger auf den ersten Grauwert.
- **ClearProc** (input_control) ..... integer ⇝ *integer*
  Zeiger auf die Prozedur, die den Speicher des Bildes beim Löschen des Objektes wieder freigibt.
  Defaultwert : 0

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char  *image;
  int            r,c;
  image = malloc(640*480);
  for (r=0; r<480; r++)
    for (c=0; c<640; c++)
      image[r*640+c] = c % 255;
  gen_image1_extern(new,"byte",640,480,(long)image,(long)free);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image1_extern` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_image1_extern` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`gen_image1`, `gen_image_const`, `get_image_pointer1`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`

Modul

Image / region / XLD management

`gen_image1_rect` (: Image : PixelPointer, Width, Height, VerticalPitch, HorizontalBitPitch, BitsPerPixel, DoCopy, ClearProc :)

Erzeugen eines Bildes mit rechteckiger Domäne aus einem Zeiger auf Pixel (mit Speicherverwaltung).

Der Operator `gen_image1_rect` erzeugt ein Bild in der Größe (`VerticalPitch`/(`HorizontalBitPitch`/8)) * `Height`. Die Pixel in `PixelPointer` werden zeilensequentiell abgelegt. Da der Typ des Parameters `PixelPointer` generisch ist (long), muß ein Cast bei dem Aufruf verwendet werden. `VerticalPitch` entspricht dem Abstand (in Bytes) des m-ten Bildpunktes in Zeile n und des m-ten Bildpunktes in Zeile n+1 innerhalb des Speichers und ist für alle Zeilen des ’Eingabebildes’ identisch. Die Breite des Ausgabebildes ist `VerticalPitch` / (`HorizontalBitPitch` / 8). Höhe von Ein- und Ausgabebild sind identisch. Die Domäne des Ausgabebildes `Image` ist ein Rechteck der Größe `Width` * `Height`. Der Parameter `HorizontalBitPitch` ist der horizontale Abstand (in Bits) benachbarter Pixel. `BitsPerPixel` ist die Anzahl der verwendeten Bits pro Pixel. Falls `HorizontalBitPitch` = 16 und `BitsPerPixel` = 16, dann ist `Image` vom Typ int4 (`DoCopy` muss ’true’ sein).

Ist `DoCopy` auf ’true’ gesetzt, werden die von `PixelPointer` referenzierten Bilddaten kopiert und Speicher für das neue Bild wird von HALCON neu angelegt. Andernfalls muss der Speicherbereich auf den `PixelPointer` zeigt wieder freigegeben werden. Dies geschieht durch die vom Aufrufer zur Verfügung gestellte Prozedur `ClearProc`. Diese Prozedur muß folgende Signatur haben:

void ClearProc(void* ptr);

Sie wird beim Löschen von `Image` aufgerufen. Falls der Speicher nicht freigegeben werden soll (im Fall von Framegrabbern oder statischem Speicher) kann eine Prozedur ”ohne Rumpf” oder der NULL-Pointer übergeben werden. Analog zum Parameter `PixelPointer` muß der Zeiger auf die Prozedur mit einem Cast auf den Typ (long) übergeben werden. Ist `DoCopy` gleich ’true’ so ist `ClearProc` irrelevant. Der Operator `gen_image1_rect` ist symmetrisch zu `get_image_pointer1_rect`.

Parameter

- ▷ **`Image`** (output_object) ..... image ⇝ *Hobject* : byte / int2 / int4
  Erzeugtes HALCON-Bild.
- ▷ **`PixelPointer`** (input_control) ..... integer ⇝ *integer*
  Zeiger auf das erste Pixel.

▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert : 512
Wertevorschläge : `Width` ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ `Width` ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : `Width` ≥ `1`

▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert : 512
Wertevorschläge : `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : `Height` ≥ `1`

▷ **`VerticalPitch`** (input_control) ..... integer ⇝ *integer*
Abstand (in Bytes) zwischen Pixel m in Zeile n und Pixel m in Zeile n+1 des 'Eingabebildes'.
Restriktion : `VerticalPitch` ≥ (`Width` · (`HorizontalBitPitch`/8))

▷ **`HorizontalBitPitch`** (input_control) ..... integer ⇝ *integer*
Abstand benachbarter Pixel in Bits.
Defaultwert : 8
Werteliste : `HorizontalBitPitch` ∈ {8, 16, 32}

▷ **`BitsPerPixel`** (input_control) ..... integer ⇝ *integer*
Anzahl verwendeter Bits pro Pixel.
Defaultwert : 8
Werteliste : `BitsPerPixel` ∈ {8, 9, 10, 11, 12, 13, 14, 15, 16, 32}
Restriktion : `BitsPerPixel` ≤ `HorizontalBitPitch`

▷ **`DoCopy`** (input_control) ..... string ⇝ *string*
Bilddaten kopieren.
Defaultwert : 'false'
Wertevorschläge : `DoCopy` ∈ {'true', 'false'}

▷ **`ClearProc`** (input_control) ..... integer ⇝ *integer*
Zeiger auf die Prozedur, die den Speicher des Bildes beim Löschen des Objektes wieder freigibt.
Defaultwert : 0

Beispiel

```
void NewImage(Hobject *new)
{
  unsigned char       *image;
  int                 r,c;

  image = malloc(640*480);
  for (r=0; r<480; r++)
    for (c=0; c<640; c++)
      image[r*640+c] = c % 255;
  gen_image1_rect(new,(long)image,400,480,640,8,8,'false',(long)free);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image1_rect` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_image1_rect` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Alternativen

`gen_image1`, `gen_image1_extern`

Siehe auch

`get_image_pointer1_rect`

Modul

Image / region / XLD management

gen_image3 (: ImageRGB : Type, Width, Height, PixelPointerRed, PixelPointerGreen, PixelPointerBlue :)

Erzeugen eines Bildes aus drei Zeigern auf die Pixel (rot/grün/blau).

gen_image3 erzeugt ein dreikanaliges Bild in der Größe Width × Height. Die Pixel in PixelPointerRed, PixelPointerGreen und PixelPointerBlue sind zeilensequentiell abgelegt. Der Typ der übergebenen Pixel (PixelPointerRed etc.) muß mit dem Namen der Pixel (Type) übereinstimmen. Der Speicher für das neue Bild wird von HALCON neu angelegt. Er kann also nach dem Aufruf freigegeben werden. Da der Typ der Parameter (PixelPointerRed etc.) generisch ist (long), muß ein "Cast" bei dem Aufruf verwendet werden.

Parameter

- ▷ **ImageRGB** (output_object) ..... image ⇝ *Hobject*
  Erzeugtes Bild mit neuer Bildmatrix.
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Pixeltyp.
  Defaultwert : 'byte'
  Werteliste : Type ∈ {'int1', 'int2', 'int4', 'byte', 'real', 'direction', 'cyclic'}
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Bildes.
  Defaultwert : 512
  Wertevorschläge : Width ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ Width ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Bildes.
  Defaultwert : 512
  Wertevorschläge : Height ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ Height ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **PixelPointerRed** (input_control) ..... integer ⇝ *integer*
  Zeiger auf den ersten Rot-Wert (Kanal 1).
- ▷ **PixelPointerGreen** (input_control) ..... integer ⇝ *integer*
  Zeiger auf den ersten Grün-Wert (Kanal 2).
- ▷ **PixelPointerBlue** (input_control) ..... integer ⇝ *integer*
  Zeiger auf den ersten Blau-Wert (Kanal 3).

Beispiel

```
void NewRGBImage(Hobject *new)
{
  unsigned char  red[768*525];
  unsigned char  green[768*525];
  unsigned char  blue[768*525];
  int            r,c;
  for (r=0; r<525; r++)
    for (c=0; c<768; c++)
    {
      red[r*768+c]   = c % 255;
      green[r*768+c] = (767 - c) % 255;
      blue[r*768+c]  = r % 255;
    }
    gen_image3(new,"byte",768,525,(long)red,(long)green,(long)blue);
```

```
}

main()
{
  Hobject  rgb;
  open_window(0,0,768,525,0,"","",&WindowHandle);
  NewRGBImage(&rgb);
  disp_color(rgb,WindowHandle);
  clear_obj(rgb);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image3` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_image3` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_image_const`, `get_image_pointer1`

Mögliche Nachfolgerfunktionen

`disp_color`

Alternativen

`gen_image1`, `compose3`, `gen_image_const`

Siehe auch

`reduce_domain`, `paint_gray`, `paint_region`, `set_grayval`, `get_image_pointer1`, `decompose3`

Modul

Image / region / XLD management

`gen_image_const ( : Image : Type, Width, Height : )`

Erzeugen eines Bildes mit konstantem Grauwert.

`gen_image_const` erzeugt ein Bild in der angegebenen Größe. Die Höhe und Breite des Bildes wird mit `Height` und `Width` festgelegt. HALCON unterstützt folgende Bildtypen:

’byte’ 1 Byte pro Pixel (0..255)

’int1’ 1 Byte pro Pixel (-127..127)

’int2’ 2 Byte pro Pixel (-32767..32767)

’int4’ 4 Byte pro Pixel (-2147483647..2147483647)

’real’ 4 Byte pro Pixel, Gleitpunkt

’complex’ zwei Matrixen vom Typ **real**

’dvf’ zwei Matrixen vom Typ **int1**

’dir’ 1 Byte pro Pixel (0..180)

’cyclic’ 1 Byte pro Pixel; zyklische Arithmetik (0..255).

Die Vorbesetzung mit 0 wird durch `set_system('init_new_image',<'true'/'false'>)` festgelegt.

Parameter

- ▷ **`Image`** (output_object) ..... image ⇝ *Hobject*
  Erzeugtes Bild mit neuer Bildmatrix.
- ▷ **`Type`** (input_control) ..... string ⇝ *string*
  Pixeltyp.
  Defaultwert : ’byte’
  Werteliste : `Type` ∈ {’int1’, ’int2’, ’int4’, ’byte’, ’real’, ’direction’, ’cyclic’, ’complex’, ’dvf’, ’lut’}

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Width ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Width ≥ 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Height ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Height ≥ 1

Beispiel

```
gen_image_const(&New,"byte",width,height);
get_image_pointer1(New,(long*)&pointer,"byte",width,height);
for (row=0; row<height-1; row++)
  for (col=0; col<width-1; col++)
    pointer[row*width+col] = (row + col) % 256;
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_image_const den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_image_const ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain, get_image_pointer1, copy_obj

Alternativen

gen_image1, gen_image3

Siehe auch

reduce_domain, paint_gray, paint_region, set_grayval, get_image_pointer1

Modul

Image / region / XLD management

gen_image_gray_ramp (: ImageGrayRamp : Alpha, Beta, Mean, Row, Column, Width, Height :)

Erzeugen einer Grauwertrampe.

gen_image_gray_ramp erzeugt eine Grauwertrampe nach folgender Gleichung:

$$\texttt{ImageGrayRamp}'(r,c) = \texttt{Alpha}(r - \texttt{Row}) + \texttt{Beta}(c - \texttt{Column}) + \texttt{Mean}$$

Die Größe des Bildes wird mit Width und Height bestimmt. Die Grauwerte sind vom Type **byte**. Grauwerte außerhalb des gültigen Bereichs werden beschnitten.

Parameter

▷ **ImageGrayRamp** (output_object) ..... image ⇝ *Hobject* : byte
Erzeugtes Bild mit neuer Bildmatrix.

▷ **Alpha** (input_control) ..... number ⇝ *real*
Steigung in Zeilenrichtung.
Defaultwert : 1.0
Wertevorschläge : Alpha ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Beta** (input_control) ..... number ⇝ *real*
Steigung in Spaltenrichtung.
Defaultwert : 1.0
Wertevorschläge : Beta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Mean** (input_control) ..... number ⇝ *real*
Mittlerer Grauwert.
Defaultwert : 128
Wertevorschläge : Mean ∈ {0, 20, 40, 60, 80, 100, 120, 140, 160, 180, 200, 220, 255}
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert : 256
Wertevorschläge : Row ∈ {128, 256, 512, 1024}
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert : 256
Wertevorschläge : Column ∈ {128, 256, 512, 1024}
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Width ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Width ≥ 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Height ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Height ≥ 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_image_gray_ramp den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_image_gray_ramp ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

moments_gray_plane

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain, get_image_pointer1, copy_obj

Alternativen

gen_image1

Siehe auch

reduce_domain, paint_gray

Modul

Image / region / XLD management

gen_image_proto (Image : ImageCleared : Grayval :)

Löschen der Grauwerte eines Bildes mit einem Wert.

gen_image_proto löscht die Grauwerte des Eingabebildes mit dem Grauwert Grayval. Das Eingabebild wird dabei nicht modifiziert, es wird nur ein neues Bild mit dem gleichen Format wie das Eingabebild erzeugt.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Bild, das gelöscht werden soll.
- ▷ **ImageCleared** (output_object) ..... image ⇝ *Hobject*
  Mit konstantem Wert belegtes Bild.
- ▷ **Grayval** (input_control) ..... number ⇝ *real* / integer
  Mit diesem Grauwert wird das Ausgabebild gelöscht.
  Defaultwert : 0
  Wertevorschläge : Grayval ∈ {0, 1, 2, 5, 10, 16, 32, 64, 128, 253, 254, 255}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_image_proto den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung ausgelöst.

Parallelisierungsinformation

gen_image_proto ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

test_obj_def

Alternativen

set_grayval, paint_gray, gen_image_const, copy_image

Siehe auch

get_image_pointer1

Modul

Basic operators

gen_image_surface_second_order (: ImageSurface : Type, Alpha, Beta, Gamma, Delta, Epsilon, Zeta, Row, Col, Width, Height :)

Erzeugen einer Grauwertfläche mit einem Polynom zweiter Ordnung.

gen_image_surface_second_order erzeugt eine gekrümmte Grauwertfläche nach folgender Gleichung:

$$\texttt{ImageSurface}(r,c) = \texttt{Alpha}(r-\texttt{Row})**2+\texttt{Beta}(c-\texttt{Col})**2+\texttt{Gamma}(r-\texttt{Row})*(c-\texttt{Col})+\texttt{Delta}(r-\texttt{Row})+\texttt{Delta}(c-\texttt{C}$$

Die Größe des Bildes wird mit Width und Height bestimmt. Die Grauwerte sind vom Type Type. Grauwerte außerhalb des gültigen Bereichs werden beschnitten.

Parameter

- ▷ **ImageSurface** (output_object) ..... image ⇝ *Hobject* : byte / real
  Erzeugtes Bild mit neuer Bildmatrix.
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Pixeltyp.
  Defaultwert : 'byte'
  Werteliste : Type ∈ {'byte', 'real'}

▷ **Alpha** (input_control) . . . . . number ⇝ *real*
Koeffizient zweiter Ordnung in Zeilenrichtung.
Defaultwert : 1.0
Wertevorschläge : Alpha ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Beta** (input_control) . . . . . number ⇝ *real*
Koeffizient zweiter Ordnung in Spaltenrichtung.
Defaultwert : 1.0
Wertevorschläge : Beta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Gamma** (input_control) . . . . . number ⇝ *real*
Gemischter Koeffizient zweiter Ordnung.
Defaultwert : 1.0
Wertevorschläge : Gamma ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Delta** (input_control) . . . . . number ⇝ *real*
Koeffizient erster Ordnung in Zeilenrichtung.
Defaultwert : 1.0
Wertevorschläge : Delta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Epsilon** (input_control) . . . . . number ⇝ *real*
Koeffizient erster Ordnung in Spaltenrichtung.
Defaultwert : 1.0
Wertevorschläge : Epsilon ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Zeta** (input_control) . . . . . number ⇝ *real*
Koeffizient nullter Ordnung
Defaultwert : 1.0
Wertevorschläge : Zeta ∈ {-2.0, -1.0, -0.5, -0.0, 0.5, 1.0, 2.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Row** (input_control) . . . . . number ⇝ *real*
Zeilenkoordinate des Scheitelpunkts der Fläche
Defaultwert : 256.0
Wertevorschläge : Row ∈ {0.0, 128.0, 256.0, 512.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Col** (input_control) . . . . . number ⇝ *real*
Spaltenkoordinate des Scheitelpunkts der Fläche
Defaultwert : 256.0
Wertevorschläge : Col ∈ {0.0, 128.0, 256.0, 512.0}
Minimale Schrittweite : 0.000001
Empfohlene Schrittweite : -0.005

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Bildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Width ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Width ≥ 1

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Bildes.
Defaultwert : 512
Wertevorschläge : `Height` ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ `Height` ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : `Height` ≥ `1`

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_image_surface_second_order` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_image_surface_second_order` ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Siehe auch

`gen_image_gray_ramp`

Modul

Image / region / XLD management

`region_to_bin` (`Region` : `BinImage` : `ForegroundGray, BackgroundGray, Width, Height` :)

Umwandlung von Regionen in ein binäres byte-Bild.

`region_to_bin` erzeugt ein **byte**-Bild und stellt darin die Eingaberegionen mit dem durch den Eingabe-Steuerparameter `ForegroundGray` vorgegebenen Grauwert dar. Regionen, die über die Größe des angegebenen Bildes hinausgehen, werden entsprechend beschnitten. Der Hintergrund wird auf `BackgroundGray` gesetzt.

Parameter

▷ **Region** (input_object) . . . . . region(-array) ⇝ *Hobject*
Enthält die darzustellenden Regionen.

▷ **BinImage** (output_object) . . . . . image ⇝ *Hobject* : byte
Ergebnisbild der Größe Width × Height mit eingetragenen Regionen.

▷ **ForegroundGray** (input_control) . . . . . integer ⇝ *integer*
Grauwert für die Darstellung der Regionen.
Defaultwert : 255
Wertevorschläge : `ForegroundGray` ∈ {0, 1, 50, 100, 128, 150, 200, 254, 255}
Typischer Wertebereich : 0 ≤ `ForegroundGray` ≤ 255 (lin)
Empfohlene Schrittweite : 1

▷ **BackgroundGray** (input_control) . . . . . integer ⇝ *integer*
Grauwert für die Darstellung des Hintergrunds.
Defaultwert : 0
Wertevorschläge : `BackgroundGray` ∈ {0, 1, 50, 100, 128, 150, 200, 254, 255}
Typischer Wertebereich : 0 ≤ `BackgroundGray` ≤ 255 (lin)
Empfohlene Schrittweite : 1

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des zu erstellenden Bildes.
Defaultwert : 512
Wertevorschläge : `Width` ∈ {256, 512, 1024}
Typischer Wertebereich : 1 ≤ `Width` ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16
Restriktion : `Width` ≥ `1`

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des zu erstellenden Bildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {256, 512, 1024}
Typischer Wertebereich : 1 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16
Restriktion : Height ≥ 1

Komplexität

$O(2 * \texttt{Height} * \texttt{Width})$.

Ergebnis

region_to_bin liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

region_to_bin ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, connection, regiongrowing, pouring

Mögliche Nachfolgerfunktionen

get_grayval

Alternativen

region_to_label, paint_region, set_grayval

Siehe auch

gen_image_proto, paint_gray

Modul

Image / region / XLD management

region_to_label (Region : ImageLabel : Type, Width, Height :)

Eintragen von Regionen in ein (Label-)Bild.

region_to_label trägt die Regionen mit ihrem Index ($1..n$) in eine Bildmatrix ein. Die erste Region erhält den „Grauwert“ 1, die zweite den „Grauwert“ 2 usw. Es werden also nur positive Zahlen verwendet. Bei **byte**-Bildern wird der Index modulo 256 genommen.

Regionen, die über die Größe des angegebenen Bildes hinausgehen, werden entsprechend beschnitten. Falls sich Regionen überlappen, so gibt die Reihenfolge vor, welche Region in dem Bild eingetragen wird: Es wird in der Reihenfolge der Regionen in die Matrix gezeichnet. Die zuletzt gezeichnete Region bleibt folglich im Ergebnisbild markiert. Die Regionen können gegebenenfalls mit expand_region aufbereitet werden.

Der Hintergrund (keine Regionenpunkte) wird auf 0 gesetzt. Hiermit kann getestet werden, in welchen Bildbereichen keine Regionen vorhanden sind.

Parameter

▷ **Region** (input_object) . . . . . region(-array) ⇝ *Hobject*
Enthält die einzutragenden Regionen.

▷ **ImageLabel** (output_object) . . . . . image ⇝ *Hobject* : byte / int2 / int4
Ergebnisbild der Größe Width × Height mit eingetragenen Regionen.

▷ **Type** (input_control) . . . . . string ⇝ *string*
Pixeltyp
Defaultwert : ’int2’
Werteliste : Type ∈ {’byte’, ’int2’, ’int4’}

▷ **Width** (input_control) ..... extent.y ⇝ *integer*
Breite des zu erstellenden Bildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {64, 128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16
Restriktion : Width ≥ 1

▷ **Height** (input_control) ..... extent.x ⇝ *integer*
Höhe des zu erstellenden Bildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {64, 128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16
Restriktion : Height ≥ 1

Komplexität

$O(2 * \texttt{Height} * \texttt{Width})$.

Ergebnis

region_to_label liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

region_to_label ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, expand_region

Mögliche Nachfolgerfunktionen

get_grayval, get_image_pointer1

Alternativen

region_to_bin, paint_region

Siehe auch

label_to_region

Modul

Image / region / XLD management

region_to_mean (Regions, Image : ImageMean : :)

Einfärben von Regionen mit ihrem mittleren Grauwert.

region_to_mean liefert ein Ergebnisbild, in das die Eingaberegionen Regions mit ihrem mittleren Grauwert innerhalb des zugrundegelegten Bildes Image eingezeichnet wurden. Diese Routine ist insbesondere zur Visualisierung von Segmentationsergebnissen nützlich.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Eingaberegionen.

▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
Zugrundeliegendes Grauwertbild.

▷ **ImageMean** (output_object) ..... image ⇝ *Hobject* : byte
Ausgabebild mit eingefärbten Regionen.

Beispiel

```
read_image(Image,'fabrik')
```

```
region_growing(Image,Regions,3,3,6,100)
region_to_mean(Regions,Image,Disp)
disp_image(Disp,WindowHandle)
set_draw(WindowHandle,'margin')
set_color(WindowHandle,'black')
disp_region(Regions,WindowHandle).
```

Ergebnis

region_to_mean liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

region_to_mean ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

regiongrowing, connection

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

paint_region, intensity

Modul

Image filters

1.5 Kanal

access_channel (MultiChannelImage : Image : Channel :)

Zugriff auf einen Kanal eines mehrkanaligen Bildes.

access_channel greift auf einen Kanal des (mehrkanaligen) Eingabebildes zu. Das Ergebnis ist ein einkanaliges Bild. Der Definitionsbereich wird von der Eingabe übernommen. Die Kanäle sind von 1 bis n durchnummeriert. Die Anzahl der Kanäle läßt sich mit count_channels bestimmen.

Parameter

- ▷ **MultiChannelImage** (input_object) . . . . . multichannel-image ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image** (output_object) . . . . . singlechannel-image ⇝ *Hobject*
  Ein Kanal von MultiChannelImage.
- ▷ **Channel** (input_control) . . . . . channel ⇝ *integer*
  Index des Kanals der zugegriffen werden soll.
  Defaultwert : 1
  Wertevorschläge : Channel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
  Typischer Wertebereich : $1 \leq$ Channel

Beispiel

```
read_image(&Color,"patras");   /* Farbbild einlesen */
access_channel(Color,&Red,1); /* Rotkanal extrahieren */
disp_image(Red,WindowHandle);
```

Parallelisierungsinformation

access_channel ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

count_channels

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

decompose2, decompose3, decompose4, decompose5

Siehe auch

count_channels

Modul

Image / region / XLD management

append_channel (MultiChannelImage, Image : ImageExtended : :)

Erweitern des Bildes um zusätzliche Matrizen (Kanäle).

append_channel hängt die Matrizen des Bildes Image an die Matrizen von MultiChannelImage an. Hierdurch erhält man ein Bild das so viele Matrizen (Kanäle) enthält, wie MultiChannelImage und Image zusammen. Der Definitionsbereich des Ausgabebildes berechnet sich als der Durchschnitt aus den Definitionsbereichen der beiden Eingabebilder.

Parameter

- ▷ **MultiChannelImage** (input_object) ..... image ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Neu anzufügendes Bild.
- ▷ **ImageExtended** (output_object) ..... multichannel-image ⇝ *Hobject*
  Um Image erweitertes Bild.

Parallelisierungsinformation

append_channel ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

compose2, compose3, compose4, compose5

Modul

Image / region / XLD management

channels_to_image (Images : MultiChannelImage : :)

Umwandlung von Einkanalbildern in ein Mehrkanalbild

channels_to_image erzeugt aus mehreren einkanaligen Bildern ein Mehrkanalbild. Der neue Definitionsbereich ergibt sich als Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Images** (input_object) ..... singlechannel-image-array ⇝ *Hobject*
  Einkanalbilder die zu einem Mehrkanalbild zusammengefaßt werden sollen.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image ⇝ *Hobject*
  Mehrkanalbild.

Parallelisierungsinformation

channels_to_image ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

count_channels, disp_image

Modul

Image / region / XLD management

compose2 (Image1, Image2 : MultiChannelImage : :)

Umwandlung von 2 Bildern in ein zweikanaliges Bild.

compose2 erzeugt aus 2 einkanaligen Bildern ein 2-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Image1** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 2.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.

Parallelisierungsinformation

compose2 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

append_channel

Siehe auch

decompose2

Modul

Image / region / XLD management

compose3 (Image1, Image2, Image3 : MultiChannelImage : :)

Umwandlung von 3 Bildern in ein dreikanaliges Bild.

compose3 erzeugt aus 3 einkanaligen Bildern ein 3-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Image1** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 2.
- ▷ **Image3** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 3.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.

Parallelisierungsinformation

compose3 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

append_channel

Siehe auch

decompose3

Modul

Image / region / XLD management

compose4 (Image1, Image2, Image3, Image4 : MultiChannelImage : :)

Umwandlung von 4 Bildern in ein vierkanaliges Bild.

compose4 erzeugt aus 4 einkanaligen Bildern ein 4-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Image1** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 2.
- ▷ **Image3** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 3.
- ▷ **Image4** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 4.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.

Parallelisierungsinformation

compose4 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

append_channel

Siehe auch

decompose4

Modul

Image / region / XLD management

compose5 (Image1, Image2, Image3, Image4,
Image5 : MultiChannelImage : :)

Umwandlung von 5 Bildern in ein fünfkanaliges Bild.

compose5 erzeugt aus 5 einkanaligen Bildern ein 5-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Image1** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 2.
- ▷ **Image3** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 3.
- ▷ **Image4** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 4.
- ▷ **Image5** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 5.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.

Parallelisierungsinformation

compose5 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

append_channel

Siehe auch

decompose5

Modul

Image / region / XLD management

compose6 (Image1, Image2, Image3, Image4, Image5, Image6 : MultiChannelImage : :)

Umwandlung von 6 Bildern in ein sechskanaliges Bild.

compose6 erzeugt aus 6 einkanaligen Bildern ein 6-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Image1** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 2.
- ▷ **Image3** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 3.
- ▷ **Image4** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 4.
- ▷ **Image5** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 5.
- ▷ **Image6** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 6.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.

Parallelisierungsinformation

compose6 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

append_channel

Siehe auch

decompose6

Modul

Image / region / XLD management

compose7 (Image1, Image2, Image3, Image4, Image5, Image6, Image7 : MultiChannelImage : :)

Umwandlung von 7 Bildern in ein siebenkanaliges Bild.

compose7 erzeugt aus 7 einkanaligen Bildern ein 7-kanaliges Bild. Der Definitionsbereich berechnet sich als der Durchschnitt der Definitionsbereiche der Eingabebilder.

Parameter

- ▷ **Image1** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 1.
- ▷ **Image2** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 2.

- ▷ **Image3** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 3.
- ▷ **Image4** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 4.
- ▷ **Image5** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 5.
- ▷ **Image6** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 6.
- ▷ **Image7** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Eingabebild 7.
- ▷ **MultiChannelImage** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.

Parallelisierungsinformation

`compose7` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`append_channel`

Siehe auch

`decompose7`

Modul

Image / region / XLD management

count_channels (MultiChannelImage : : : Channels)

Anzahl Kanäle eine Bildes.

`count_channels` berechnet zu allen Eingabebildern die Anzahl der Kanäle.

Parameter

- ▷ **MultiChannelImage** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject*
  Ein- oder Mehrkanaliges Bild.
- ▷ **Channels** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl Kanäle.

Beispiel

```
read_image(&Color,"patras");
count_channels(Color,&num_channels);
for (i=1; i<=num_channels; i++)
{
  access_channel(Color,&Channel,i);
  disp_image(Channel,WindowHandle);
  clear_obj(Channel);
}
```

Parallelisierungsinformation

`count_channels` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`access_channel`, `append_channel`, `disp_image`

Siehe auch

`append_channel`, `access_channel`

Modul

Image / region / XLD management

decompose2 (MultiChannelImage : Image1, Image2 : :)

Umwandlung eines zweikanaligen Bildes in zwei Bilder.

decompose2 erzeugt aus einem 2-kanaligen Bild zwei einkanalige Bilder mit gleichem Definitionsbereich.

Parameter

- ▷ **MultiChannelImage** (input_object) . . . . . multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image1** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 1.
- ▷ **Image2** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 2.

Parallelisierungsinformation

decompose2 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

count_channels

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

access_channel, image_to_channels

Siehe auch

compose2

Modul

Image / region / XLD management

decompose3 (MultiChannelImage : Image1, Image2, Image3 : :)

Umwandlung eines dreikanaligen Bildes in drei Bilder.

decompose3 erzeugt aus einem 3-kanaligen Bild drei einkanalige Bilder mit gleichem Definitionsbereich.

Parameter

- ▷ **MultiChannelImage** (input_object) . . . . . multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image1** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 1.
- ▷ **Image2** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 2.
- ▷ **Image3** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 3.

Parallelisierungsinformation

decompose3 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

count_channels

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

access_channel, image_to_channels

Siehe auch

compose3

Modul

Image / region / XLD management

decompose4 (MultiChannelImage : Image1, Image2, Image3, Image4 : :)

Umwandlung eines vierkanaligen Bildes in vier Bilder.

decompose4 erzeugt aus einem 4-kanaligen Bild vier einkanalige Bilder mit gleichem Definitionsbereich.

Parameter

- ▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image1** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 1.
- ▷ **Image2** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 2.
- ▷ **Image3** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 3.
- ▷ **Image4** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 4.

Parallelisierungsinformation

decompose4 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

count_channels

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

access_channel, image_to_channels

Siehe auch

compose4

Modul

Image / region / XLD management

decompose5 (MultiChannelImage : Image1, Image2, Image3, Image4, Image5 : :)

Umwandlung eines fünfkanaligen Bildes in fünf Bilder.

decompose5 erzeugt aus einem 5-kanaligen Bild fünf einkanalige Bilder mit gleichem Definitionsbereich.

Parameter

- ▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image1** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 1.
- ▷ **Image2** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 2.
- ▷ **Image3** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 3.
- ▷ **Image4** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 4.
- ▷ **Image5** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 5.

Parallelisierungsinformation

decompose5 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

count_channels

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose5`

Modul

Image / region / XLD management

decompose6 (MultiChannelImage : Image1, Image2, Image3, Image4, Image5, Image6 : :)

Umwandlung eines sechskanaligen Bildes in sechs Bilder.

`decompose6` erzeugt aus einem 6-kanaligen Bild sechs einkanalige Bilder mit gleichem Definitionsbereich.

Parameter

- ▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image1** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 1.
- ▷ **Image2** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 2.
- ▷ **Image3** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 3.
- ▷ **Image4** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 4.
- ▷ **Image5** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 5.
- ▷ **Image6** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 6.

Parallelisierungsinformation

`decompose6` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`count_channels`

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose6`

Modul

Image / region / XLD management

decompose7 (MultiChannelImage : Image1, Image2, Image3, Image4, Image5, Image6, Image7 : :)

Umwandlung eines siebenkanaligen Bildes in sieben Bilder.

`decompose7` erzeugt aus einem 7-kanaligen Bild sieben einkanalige Bilder mit gleichem Definitionsbereich.

Parameter

- ▷ **MultiChannelImage** (input_object) . . . . . multichannel-image(-array) ⇝ *Hobject*
  Mehrkanaliges Bild.
- ▷ **Image1** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 1.
- ▷ **Image2** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 2.
- ▷ **Image3** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 3.
- ▷ **Image4** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 4.
- ▷ **Image5** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 5.
- ▷ **Image6** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 6.
- ▷ **Image7** (output_object) . . . . . singlechannel-image(-array) ⇝ *Hobject*
  Ausgabebild 7.

Parallelisierungsinformation

`decompose7` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`count_channels`

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`access_channel`, `image_to_channels`

Siehe auch

`compose7`

Modul

Image / region / XLD management

image_to_channels (MultiChannelImage : Images : :)

Umwandlung eines Mehrkanalbildes in Einkanalbilder

`image_to_channels` erzeugt aus dem Mehrkanalbild in `MultiChannelImage` für jeden Kanal ein einkanaliges Bild. Die Definitionsbereiche werden von dem Eingabebild übernommen. Es werden so viele Bilder erzeugt wie `MultiChannelImage` Kanäle hat.

Parameter

- ▷ **MultiChannelImage** (input_object) . . . . . multichannel-image ⇝ *Hobject*
  Mehrkanalbild das zerlegt werden soll.
- ▷ **Images** (output_object) . . . . . singlechannel-image-array ⇝ *Hobject*
  Erzeugte Einkanalbilder.

Parallelisierungsinformation

`image_to_channels` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`count_channels`

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`access_channel`, `decompose2`, `decompose3`, `decompose4`, `decompose5`

Modul

Image / region / XLD management

1.6 Manipulation

paint_gray (ImageSource, ImageDestination : MixedImage : :)

Eintragen der Grauwerte eines Bildes in ein anderes Bild.

paint_gray trägt die Grauwerte der Bilder aus ImageSource in das Bild in ImageDestination ein. Kopiert werden nur die Grauwerte aus dem Definitionsbereich von ImageSource (siehe reduce_domain).

Parameter

- ▷ **ImageSource** (input_object) ..... image ⇝ *Hobject*
  Eingabebild, das die zusätzlichen Grauwerte liefert.
- ▷ **ImageDestination** (input_object) ..... image ⇝ *Hobject*
  Eingabebild, in das eingezeichnet werden soll.
- ▷ **MixedImage** (output_object) ..... image ⇝ *Hobject*
  Ergebnisbild.

Beispiel

```
/* Copy of a segment (circle) of the image 'affe' into a new image (New):
*/

read_image(Image,'affe')
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 of a segment of the image 'affe' into New1 */
paint_gray(Mask,New1,New).
```

Ergebnis

Sind die Parameterwerte korrekt, liefert paint_gray den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

paint_gray ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image, gen_image_const, gen_image_proto

Alternativen

get_image_pointer1, set_grayval, copy_image

Siehe auch

paint_region

Modul

Basic operators

paint_region (Region, Image : ImageResult : Grayval, Type :)

Eintragen einer Region in ein Bild.

paint_region trägt die Regionen aus Region in das Bild Image mit dem vorgegebenen Grauwerten Grayval ein. Die Grauwerte können entweder einmal für jeden Bildkanal, geltend für alle Regionen, definiert werden, oder für jede Region und jeden Bildkanal einzeln. Um letzteres zu definieren, werden die Kanal-Grauwerte g zu jedem Objekt gruppiert und zu einem Tupel entsprechend der Reihenfolge der Objekte verknüpft, z.B. für ein dreikanaliges Bild:

$$[g(\mathit{channel1},\mathit{object1}), g(\mathit{channel2},\mathit{object1}), g(\mathit{channel3},\mathit{object1}), g(\mathit{channel1},\mathit{object2}), \ldots]$$

Sollen alle Regionen den gleichen Farbwert erhalten, reicht es aus, die Grauwerte für die erste Region (*object1*) zu definieren. Der Parameter `Type` gibt an, ob die Regionen ausgefüllt (**'fill'**) oder nur der Rand (**'margin'**) eingetragen werden soll. `ImageResult` liefert das Ergebnisbild in das die Regionen eingezeichnet wurden.

Achtung

`paint_region` sollte nur zum Eintragen in neu erzeugte Objekte (`gen_image_const`) verwendet werden, da sonst die Grauwerte von anderen Bildern überschrieben werden könnten.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die eingetragen werden soll.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Bild, in das die Regionen eingezeichnet werden.
- ▷ **`ImageResult`** (output_object) ..... image ⇝ *Hobject*
  Bild, in das die Regionen eingezeichnet wurden.
- ▷ **`Grayval`** (input_control) ..... number-array ⇝ *real* / integer
  Mit diesem Grauwerten werden die Regionen in Result eingetragen.
  Defaultwert : 255.0
  Wertevorschläge : `Grayval` ∈ {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 ⇝ *string*
  Regionen ausfüllen oder nur den Rand eintragen.
  Defaultwert : 'fill'
  Werteliste : `Type` ∈ {'fill', 'margin'}

Beispiel

```
/* Copy of a rectangle in a new image (New) */

read_image(Image,'affe')
gen_rectangle1(Rectangle,100.0,100.0,300.0,300.0)
reduce_domain(Image,Rectangle,Mask)
/* generate a black image */
gen_image_proto(Image,New1,0.0)
/* copy a white rectangle */
paint_region(Mask,New1,New,255.0,'fill',).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `paint_region` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`paint_region` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_image`, `gen_image_const`, `gen_image_proto`, `reduce_domain`

Alternativen

`set_grayval`, `paint_gray`

Siehe auch

`reduce_domain`, `set_draw`, `gen_image_const`

Modul

Basic operators

`set_grayval` (Image : : Row, Column, Grayval :)

Setzen von einzelnen Grauwerten in einem Bildobjekt.

set_grayval setzt die Grauwerte des Eingabebildes Image. Grayval ist dabei ein Tupel von Gleitpunkt- bzw. ganzen Zahlen, das die Grauwerte enthält, die einigen Bildpunkten von Image zugewiesen werden sollen. Die Zeilenkoordinaten der Bildpunkte stehen in Row, die Spalten in Column. Die Anzahl der Werte in Grayval muß mit der Anzahl der angegebenen Punkte übereinstimmen.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Dieses Bild wird modifiziert.
- ▷ **Row** (input_control) ..... point.y(-array) ⇝ *integer*
  Zeilenkoordinaten der zu modifizierenden Bildpunkte.
  Defaultwert : 0
  Wertevorschläge : Row ∈ {0, 10, 50, 127, 255, 511}
  Typischer Wertebereich : 0 ≤ Row
  Restriktion : (0 ≤ Row) ∧ (Row < height(Image))
- ▷ **Column** (input_control) ..... point.x(-array) ⇝ *integer*
  Spaltennummern der zu modifizierenden Bildpunkte.
  Defaultwert : 0
  Wertevorschläge : Column ∈ {0, 10, 50, 127, 255, 511}
  Typischer Wertebereich : 0 ≤ Column
  Restriktion : (0 ≤ Column) ∧ (Column < width(Image))
- ▷ **Grayval** (input_control) ..... grayval(-array) ⇝ *real* / integer
  Zu setzende Grauwerte.
  Defaultwert : 255.0
  Wertevorschläge : Grayval ∈ {0.0, 1.0, 10.0, 128.0, 255.0}

Ergebnis

Bei korrekter Besetzung der Parameter liefert set_grayval den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_grayval ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image, get_image_pointer1, gen_image_proto, gen_image1

Alternativen

get_image_pointer1, paint_gray, paint_region

Siehe auch

get_grayval, gen_image_const, gen_image1, gen_image_proto

Modul

Basic operators

1.7 Merkmale

area_center_gray (Regions, Image : : : Area, Row, Column)

Berechnung der Fläche und des Schwerpunktes von Regionen in Grauwertbildern.

area_center_gray berechnet die Fläche und den Schwerpunkt der Eingaberegionen Regions, die die durch das Eingabebild Image definierten Grauwerte besitzen. Dieser Operator verhält sich ähnlich wie der Operator area_center, nur daß hier die Grauwerte zur Berechnung der Fläche und des Schwerpunktes verwendet werden.

Die Fläche A einer Region R im Bild mit den Grauwerten $g(r, c)$ wird definiert durch

$$A = \sum_{(r,c) \in R} g(r, c).$$

Das bedeutet, daß die Fläche durch das Volumen der durch die Grauwerte definierten Funktion $g(r, c)$ definiert wird. Der Schwerpunkt wird durch die ersten zwei normalisierten Momente der Funktion $g(r, c)$ definiert, d.h.

durch $(m_{1,0}, m_{0,1})$, wobei

$$m_{p,q} = \frac{1}{A} \sum_{(r,c) \in R} r^p c^q g(r,c).$$

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte / cyclic / direction / int1 / int2 / int4 / real
  Grauwertdaten.
- ▷ **Area** (output_control) ..... real(-array) ⇝ *real*
  Grauwertvolumen der Region.
- ▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinate des Grauwert-Schwerpunktes.
- ▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenkoordinate des Grauwert-Schwerpunktes.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `area_center_gray` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`area_center_gray` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`area_center`

Siehe auch

`area_center_xld`, `elliptic_axis_gray`

Modul

Image filters

cooc_feature_image (Regions, Image : : LdGray, Direction : Energy, Correlation, Homogeneity, Contrast)

Berechnung der Co-Occurrence-Matrix und daraus abgeleiteter Grauwertmerkmale.

Der Aufruf von `cooc_feature_image` entspricht der Hintereinanderausführung von `gen_cooc_matrix` und `cooc_feature_matrix`. Wenn mehrere Richtungsmatrizen der Co-Occurrence-Matrix hintereinander ausgewertet werden sollen, ist es günstiger, die Matrix mit `gen_cooc_matrix` zu erstellen und danach `cooc_feature_matrix` für die Ergebnismatrix aufzurufen. Bei dem Parameter `Direction` wird die Richtung der Nachbarschaft in Winkel oder **’mean’** übergeben. Bei **’mean’** wird der Mittelwert aller vier Richtungen berechnet.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
  Zugehörige Grauwerte.
- ▷ **LdGray** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu unterscheidenden Grauwerte ($2^{\texttt{LdGray}}$).
  Defaultwert : 6
  Werteliste : `LdGray` ∈ {1, 2, 3, 4, 5, 6, 7, 8}
- ▷ **Direction** (input_control) ..... integer ⇝ *integer* / string
  Auszuwertende Richtung in der Matrix.
  Defaultwert : 0
  Werteliste : `Direction` ∈ {0, 45, 90, 135, ’mean’}

▷ **Energy** (output_control) ..... real(-array) ⇝ *real*
Energie der Grauwerte.

▷ **Correlation** (output_control) ..... real(-array) ⇝ *real*
Korrelation der Grauwerte.

▷ **Homogeneity** (output_control) ..... real(-array) ⇝ *real*
Lokale Homogenität der Grauwerte.

▷ **Contrast** (output_control) ..... real(-array) ⇝ *real*
Kontrast der Grauwerte.

Ergebnis

`cooc_feature_image` liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten (**byte**) eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`cooc_feature_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gen_cooc_matrix`

Alternativen

`cooc_feature_matrix`

Siehe auch

`intensity`, `min_max_gray`, `entropy_gray`, `select_gray`

Modul

Image filters

cooc_feature_matrix (CoocMatrix : : : Energy, Correlation, Homogeneity, Contrast)

Berechnung von Grauwertmerkmalen aus einer Co-Occurrence-Matrix.

Die Prozedur berechnet aus einer Co-Occurence-Matrix (`CoocMatrix`) die Energie (`Energy`), Korrelation (`Correlation`), lokale Homogenität (`Homogeneity`) und den Kontrast (`Contrast`).

`cooc_feature_matrix` berechnet aus dem Teil der mit `gen_cooc_matrix` generierten Eingabe-Matrix, der der mit dem dortigen Parametern $LdGray$ und $Direction$ angegebenen Richtungsmatrix entspricht, die Grauwertmerkmale nach folgenden Formeln:

Energie:

$$Energy = \sum_{i,j=0}^{width} c_{ij}^2$$

(Maß für Bildhomogenität)

Korrelation:

$$Correlation = \frac{\sum_{i,j=0}^{width} (i-u_x)(j-u_y)c_{ij}}{s_x s_y}$$

(Maß für die Grauwertabhängigkeiten)

lokale Homogenität:

$$Homogeneity = \sum_{i,j=0}^{width} \frac{1}{1+(i-j)^2} c_{ij}$$

Kontrast:

$$Contrast = \sum_{i,j=0}^{width} (i-j)^2 c_{ij}$$

(Maß für die Größe der Intensitätsunterschiede)

wobei

$$
\begin{array}{rcl}
width & = & \text{Breite von } \texttt{CoocMatrix} \\
c_{ij} & = & \text{Eintrag in der Co-Occurrence-Matrix} \\
u_x & = & \sum_{i,j=0}^{width} i * c_{ij} \\
u_y & = & \sum_{i,j=0}^{width} j * c_{ij} \\
s_x^2 & = & \sum_{i,j=0}^{width} (i - u_x)^2 * c_{ij} \\
s_y^2 & = & \sum_{i,j=0}^{width} (i - u_y)^2 * c_{ij}
\end{array}
$$

Achtung

Die Region des Eingabebildes wird nicht beachtet.

Parameter

- ▷ **CoocMatrix** (input_object) ..... image ⇝ *Hobject* : int4
  Co-Occurrence-Matrix.
- ▷ **Energy** (output_control) ..... real ⇝ *real*
  Homogenität der Grauwerte.
- ▷ **Correlation** (output_control) ..... real ⇝ *real*
  Korrelation der Grauwerte.
- ▷ **Homogeneity** (output_control) ..... real ⇝ *real*
  Lokale Homogenität der Grauwerte.
- ▷ **Contrast** (output_control) ..... real ⇝ *real*
  Kontrast der Grauwerte.

Ergebnis

`cooc_feature_matrix` liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`cooc_feature_matrix` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gen_cooc_matrix`

Alternativen

`cooc_feature_image`

Siehe auch

`intensity`, `min_max_gray`, `entropy_gray`, `select_gray`

Modul

Image filters

elliptic_axis_gray (Regions, Image : : Ra, Rb, Phi)

Berechnung der Orientierung und der Hauptachsen von Regionen in Grauwertbildern.

`elliptic_axis_gray` berechnet die Hauptachsen und die Orientierung der Ellipse, die die „gleiche Orientierung“ und das “gleiche Seitenverhältnis“ wie die Eingaberegion haben. Es können auch mehrere Eingaberegionen als Tupel in `Regions` übergeben werden. Es wird die Länge der großen Halbachse `Ra` und der kleinen Halbachse `Rb` sowie die Orientierung der großen Halbachse bezüglich der x-Achse (`Phi`) bestimmt. Der Winkel wird dabei im Bogenmaß angegeben. Die Berechnung erfolgt dabei analog zu `elliptic_axis`, nur daß hier Grauwertmomente verwendet werden, die durch das Eingabebild `Image` definiert werden. Zur Definition der Grauwertmomente siehe `area_center_gray`.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).

▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte / cyclic / direction / int1 / int2 / int4 / real
Grauwertdaten.
▷ **Ra** (output_control) ..... real(-array) ⇝ *real*
Große Halbachse der Region.
▷ **Rb** (output_control) ..... real(-array) ⇝ *real*
Kleine Halbachse der Region.
▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
Winkel zwischen der großen Halbachse und der x-Achse.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert `elliptic_axis_gray` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`elliptic_axis_gray` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Mögliche Nachfolgerfunktionen

`gen_ellipse`

Alternativen

`elliptic_axis`

Siehe auch

`area_center_gray`

Modul

Image filters

entropy_gray (Regions, Image : : : Entropy, Anisotropy)

Bestimmung der Entropie und der Anisotropie von Bildern.

`entropy_gray` erstellt das Histogramm der relativen Häufigkeiten der Grauwerte im Eingabebild und errechnet daraus nach folgenden Formeln im Eingabebild und errechnet daraus für jede Region aus `Regions` nach folgenden Formeln die Entropie und den Anisotropiekoeffizienten:

Entropie:

$$Entropy = -\sum_{0}^{255} rel[i] * \log_2(rel[i])$$

Anisotropiekoeffizient:

$$Anisotropy = \frac{\sum_0^k rel[i] * \log_2(rel[i])}{Entropy}$$

wobei:

$rel[i]$ = Histogramm der relativen Grauwerthäufigkeiten
i = Grauwert des Eingabebildes $(0 \ldots 255)$
k = Kleinstmöglicher Grauwert mit $\sum_0^k rel[i] \geq 0.5$

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen, in denen die Merkmale bestimmt werden sollen.
▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
Grauwertdaten.
▷ **Entropy** (output_control) ..... real(-array) ⇝ *real*
Informationsgehalt (Entropie) der Grauwerte.
Zusicherung : $(0 \leq \texttt{Entropy}) \wedge (\texttt{Entropy} \leq 8)$

▷ **Anisotropy** (output_control) ..... real(-array) ⇝ *real*
Bewertung der Symmetrie der Grauwertverteilung.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F + 255)$.

Ergebnis

`entropy_gray` liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`entropy_gray` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Alternativen

`select_gray`

Siehe auch

`entropy_image`, `gray_histo`, `fuzzy_entropy`, `fuzzy_perimeter`

Modul

Image filters

fit_surface_first_order (Regions, Image : : Algorithm, Iterations, ClippingFactor : Alpha, Beta, Gamma)

Berechnung der Grauwertmomente und der Approximation durch eine Fläche erster Ordnung (Ebene).

`fit_surface_first_order` berechnet die Grauwertmomente und die Approximation der Grauwerte durch eine Fläche erster Ordnung. Dazu wird der Abstand zwischen der Fläche und den Grauwerten minimiert. Eine Fläche erster Ordnung wird durch folgende Gleichung beschrieben:

$$\texttt{Image}'(r,c) = \texttt{Alpha}(r - r\_center) + \texttt{Beta}(c - c\_center) + \texttt{Gamma}$$

r_center und c_center sind die Koordinaten des Schwerpunkts der Eingaberegion. Bei der Minimierung des Abstands werden die Parameter `Alpha` bis `Gamma` berechnet.

Das gewünschete Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

’regression’ Standard ’least squares’ Anpassung.

’huber’ Gewichtete *least squares* Anpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

’tukey’ Gewichtete *least squares* Anpassung, bei der Ausreißer nach dem Ansatz von Tukey gedämpft werden.

Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen. Er wird im Modus ’regression’ ignoriert.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Zu untersuchende Regionen.

▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte / real
Zugehörige Grauwerte.

▷ **Algorithm** (input_control) ..... string ⇝ *string*
Algorithmus Anpassung.
Defaultwert : ’regression’
Werteliste : `Algorithm` ∈ {’regression’, ’huber’, ’tukey’}

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von Iterationen (unbenutzt bei ’regression’).
Defaultwert : 5
Restriktion : `Iterations` ≥ `0`

▷ **ClippingFactor** (input_control) ..... real ⇝ *real*
Clipping Faktor für die Ausreißerdämpfung.
Defaultwert : 2.0
Werteliste : ClippingFactor ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
Restriktion : ClippingFactor > 0

▷ **Alpha** (output_control) ..... real(-array) ⇝ *real*
Parameter Alpha der Fläche.

▷ **Beta** (output_control) ..... real(-array) ⇝ *real*
Parameter Beta der Fläche.

▷ **Gamma** (output_control) ..... real(-array) ⇝ *real*
Parameter Gamma der Fläche.

Ergebnis

fit_surface_second_order liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten (**byte**) eingegeben wird und die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fit_surface_first_order ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Siehe auch

moments_gray_plane, fit_surface_second_order

Modul

Image filters

fit_surface_second_order (Regions, Image : : Algorithm, Iterations, ClippingFactor : Alpha, Beta, Gamma, Delta, Epsilon, Zeta)

Berechnung der Grauwertmomente und der Approximation durch eine Fläche zweiter Ordnung.

fit_surface_second_order berechnet die Grauwertmomente und die Approximation der Grauwerte durch eine Fläche zweiter Ordnung. Dazu wird der Abstand zwischen der Fläche und den Grauwerten minimiert. Eine Fläche zweiter Ordnung wird durch folgende Gleichung beschrieben:

$$\texttt{Image}(r,c) = \texttt{Alpha}(r-r\_center)**2+\texttt{Beta}(c-c\_center)**2+\texttt{Gamma}(r-r\_center)*(c-c\_center)+\texttt{Delta}(r-r\_center)+$$

r_center und c_center sind die Koordinaten des Schwerpunkts der Eingaberegion. Bei der Minimierung des Abstands werden die Parameter Alpha bis Zeta berechnet.

Das gewünschete Approximationsverfahren wird über den Parameter Algorithm ausgewählt:

’regression’ Standard ’least squares’ Anpassung.

’huber’ Gewichtete *least squares* Geradenanpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

’tukey’ Gewichtete *least squares* Geradenanpassung, bei der Ausreißer nach dem Ansatz von Tukey gedämpft werden.

Der Parameter ClippingFactor (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wird iteriert. Der Parameter Iterations enthält die Anzahl durchzuführender Iterationen. Er wird im Modus ’regression’ ignoriert.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Zu untersuchende Regionen.

▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte / real
Zugehörige Grauwerte.

▷ **Algorithm** (input_control) ..... string ⇝ *string*
Algorithmus Anpassung.
Defaultwert : ’regression’
Werteliste : Algorithm ∈ {’regression’, ’tukey’, ’huber’}

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Maximale Anzahl von Iterationen (unbenutzt bei 'regression').
Defaultwert : 5
Restriktion : `Iterations` $\geq$ `0`

▷ **ClippingFactor** (input_control) ..... real ⇝ *real*
Clipping Faktor für die Ausreißerdämpfung.
Defaultwert : 2.0
Werteliste : `ClippingFactor` $\in \{1.0, 1.5, 2.0, 2.5, 3.0\}$
Restriktion : `ClippingFactor` $>$ `0`

▷ **Alpha** (output_control) ..... real(-array) ⇝ *real*
Parameter Alpha der Fläche.

▷ **Beta** (output_control) ..... real(-array) ⇝ *real*
Parameter Beta der Fläche.

▷ **Gamma** (output_control) ..... real(-array) ⇝ *real*
Parameter Gamma der Fläche.

▷ **Delta** (output_control) ..... real(-array) ⇝ *real*
Parameter Delta der Fläche.

▷ **Epsilon** (output_control) ..... real(-array) ⇝ *real*
Parameter Epsilon der Fläche.

▷ **Zeta** (output_control) ..... real(-array) ⇝ *real*
Parameter Zeta der Fläche.

Ergebnis

`fit_surface_second_order` liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten (**byte**) eingegeben wird und die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fit_surface_second_order` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Siehe auch

`moments_gray_plane`, `fit_surface_first_order`

Modul

Image filters

fuzzy_entropy (Regions, Image : : Apar, Cpar : Entropy)

Fuzzy-Entropie von Regionen.

`fuzzy_entropy` berechnet die Entropy einer Fuzzy Menge. Hierbei wird das Bild als eine Fuzzy Menge betrachtet. Die Entropy bewertet, wie gut das Bild ein schwarzes bzw. weißes Bild annähert. Sie wird durch folgende Gleichung definiert:

$$H(X) = \frac{1}{MNln2} \sum_l T_e(l) h(l)$$

wobei $M \times N$ die Dimension des Bildes und $h(l)$ das Histogramm des Bildes ist. Außerdem

$$T_e(l) = -\mu(l) \ln \mu(l) - (1 - \mu(l)) \ln(1 - \mu(l))$$

Hier ist $u(x(m,n))$ eine Zugehörigkeitsfunktion (siehe `fuzzy_perimeter`), die die Fuzzy-Menge definiert. Dabei gelten für die Parameter die gleiche Bedingungen wie in `fuzzy_perimeter`.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen in den das Merkmal bestimmt werden soll.

▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
Bild für die Grauwerte (Fuzzy-Zugehörigkeitswerte).

▷ **Apar** (input_control) ..... integer ⇝ *integer*
Anfang der Fuzzyfunktion
Defaultwert : 0
Wertevorschläge : Apar ∈ {0, 5, 10, 20, 50, 100}
Typischer Wertebereich : 0 ≤ Apar ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5

▷ **Cpar** (input_control) ..... integer ⇝ *integer*
Ende der Fuzzyfunktion.
Defaultwert : 255
Wertevorschläge : Cpar ∈ {50, 100, 150, 200, 220, 255}
Typischer Wertebereich : 0 ≤ Cpar ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : Apar ≤ Cpar

▷ **Entropy** (output_control) ..... real(-array) ⇝ *real*
Fuzzy Entropy einer Region.

Beispiel

```
/* To find a Fuzzy Entropy from an Image */
read_image(Image,'affe')
fuzzy_entropy(Trans,Trans,0,255,Entro).
```

Ergebnis

fuzzy_entropy liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fuzzy_entropy ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Siehe auch

fuzzy_perimeter

Literatur

M.K. Kundu, S.K. Pal: ‘Äutomatic selection of object enhancement operator with quantitative justification based on fuzzy set theoretic measures”; Pattern Recognition Letters 11; 1990; pp. 811-829.

Modul

Image filters

fuzzy_perimeter (Regions, Image : : Apar, Cpar : Perimeter)

Fuzzy-Umfang einer Region.

Mit Hilfe des Merkmals fuzzy_perimeter werden die Unterschiede in der Zugehörigkeitwerte zwischen einem Bildpunkt und seinem Nachbarbildpunkt gemessen. Dabei wird der rechte und der untere Nachbar berücksichtigt. Der Umfang ist wie folgt definiert:

$$p(X) = \sum_{m=1}^{M-1}\sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m,n+1})| + \sum_{m=1}^{M-1}\sum_{n=1}^{N-1} |\mu_X(x_{m,n}) - \mu_X(x_{m+1,n})|$$

wobei $M \times N$ die Dimension des Bildes ist und $u(x(m,n))$ die Zugehörigkeitsfunktion. Bei dieser Implementation wurde die Standard-S Funktion nach Zadeh verwendet und wird wie folgt definiert:

$$\mu_X(x) = \begin{cases} 0, & x \le a \\ 2\left(\frac{x-a}{c-a}^2\right), & a < x \le b \\ 1 - 2\left(\frac{x-a}{c-a}^2\right), & b < x \le c \\ 1, & c \le x \end{cases}$$

Für die Parameter a, b und c gelten nachstehende Bedingungen: $b = \frac{a+c}{2}$ ist der Wendepunkt der Funktion, $\Delta b = b - a = c - b$ ist die Bandbreite, für $x = b$ gilt $\mu(x) = 0.5$. In `fuzzy_perimeter` sind die Parameter `Apar` und `Cpar` folgendermaßen definiert: b ist $\frac{\texttt{Apar}+\texttt{Cpar}}{2}$.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen in den das Merkmal bestimmt werden soll.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte
  Bild für die Grauwerte (Fuzzy-Zugehörigkeitswerte).
- ▷ **`Apar`** (input_control) ..... integer ⇝ *integer*
  Anfang der Fuzzyfunktion
  Defaultwert : 0
  Wertevorschläge : `Apar` ∈ {0, 5, 10, 20, 50, 100}
  Typischer Wertebereich : 0 ≤ `Apar` ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5
- ▷ **`Cpar`** (input_control) ..... integer ⇝ *integer*
  Ende der Fuzzyfunktion.
  Defaultwert : 255
  Wertevorschläge : `Cpar` ∈ {50, 100, 150, 200, 220, 255}
  Typischer Wertebereich : 0 ≤ `Cpar` ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5
  Restriktion : `Apar` ≤ `Cpar`
- ▷ **`Perimeter`** (output_control) ..... real(-array) ⇝ *real*
  Fuzzy-Umfang einer Region.

Beispiel

```
/* To find a Fuzzy Entropy from an Image */
read_image(Image,'affe')
fuzzy_perimeter(Trans,Trans,0,255,Per).
```

Ergebnis

`fuzzy_perimeter` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fuzzy_perimeter` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Siehe auch

`fuzzy_entropy`

Literatur

M.K. Kundu, S.K. Pal: ‘Äutomatic selection of object enhancement operator with quantitative justification based on fuzzy set theoretic measures”; Pattern Recognition Letters 11; 1990; pp. 811-829.

Modul

Image filters

`gen_cooc_matrix` (Regions, Image : Matrix : LdGray, Direction :)

Berechnung der Co-Occurrence-Matrix einer Region in einem Bild.

`gen_cooc_matrix` ermittelt aus den Eingaberegionen, wie oft die Grauwerte i und j in einer bestimmten Richtung (**0**, **45**, **90**, **135** Grad) nebeneinander liegen, speichert diese Zahl in der Co-Occurrence-Matrix an den Stellen (i, j) und (j, i) ab (die Matrix ist also symmetrisch) und normiert am Schluß die Matrix mit der Zahl der Einträge. `LdGray` git die Anzahl der zu unterscheidenden Grauwerte (nämlich $2^{\texttt{LdGray}}$) an.

Beispiel (`LdGray` = 2, d.h. es werden 4 Grauwerte unterschieden):

Eingabebild mit Grauwerten:	Co-Occurrence-Matrix (nicht normiert)	
0 0 3 1 1 2 1 2 3	2 0 0 1 0 2 2 0 0 2 0 1 1 0 1 0	0 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0
	0 2 0 0 2 2 1 0 0 1 0 2 0 0 2 0	0 1 0 0 1 2 0 1 0 0 2 0 0 1 0 0

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Image** (input_object) . . . . . image ⇝ *Hobject* : byte
  Bild das die Grauwerte liefert.
- ▷ **Matrix** (output_object) . . . . . image(-array) ⇝ *Hobject* : int4
  Co-Occurrence-Matrix (Matrizen).
- ▷ **LdGray** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der zu unterscheidenden Grauwerte ($2^{\texttt{LdGray}}$).
  Defaultwert : 6
  Werteliste : LdGray $\in \{1, 2, 3, 4, 5, 6, 7, 8\}$
  Typischer Wertebereich : $1 \leq$ LdGray ≤ 256 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Direction** (input_control) . . . . . integer ⇝ *integer*
  Richtung der Nachbarschaftsbeziehung.
  Defaultwert : 0
  Werteliste : Direction $\in \{0, 45, 90, 135\}$

Ergebnis

gen_cooc_matrix liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Bilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_cooc_matrix ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

draw_region, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, threshold, erosion_circle, gauss_image, smooth_image, sub_image

Alternativen

cooc_feature_image

Siehe auch

cooc_feature_matrix

Modul

Image filters

gray_histo (Regions, Image : : : AbsoluteHisto, RelativeHisto)

Verteilung der Grauwerte.

gray_histo berechnet für das übergebene Bild (Image) innerhalb von Regions das absolute (AbsoluteHisto) und relative (RelativeHisto) Histogramm der Grauwerte.

Beide Histogramme sind Tupel von 256 Werten, die (beginnend mit 0) die Häufigkeiten der einzelnen Grauwerte im Bild enthalten.

`AbsoluteHisto` gibt dabei die absoluten Häufigkeiten der Grauwerte durch ganze Zahlen an, `RelativeHisto` die relativen, d.h. die absoluten Häufigkeiten dividiert durch die Fläche des Bildes, als Gleitpunktzahlen.

real- **int2**-, und **int4**-Bilder werden in **byte**-Bilder umgewandelt (zuerst wird der größte und der kleinste Grauwert im Bild bestimmt und anschließend werden die Originalgrauwerte linear in den Bereich 0..255 abgebildet) und dann wie oben verarbeitet. Mit `set_paint(::WindowHandle,'histogram':)` und `disp_image` kann das Histogramm auch direkt als Graphik ausgegeben werden.

Achtung

real-, int2 und int4-Bilder werden auf 256 Grauwerte reduziert.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Region, in der das Histogramm berechnet werden soll.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte / cyclic / direction / int1 / int2 / int4 / real
  Bild, dessen Grauwertverteilung berechnet werden soll.
- ▷ **`AbsoluteHisto`** (output_control) ..... histogram-array ⇝ *integer*
  Absolute Häufigkeiten der Grauwerte.
- ▷ **`RelativeHisto`** (output_control) ..... histogram-array ⇝ *real*
  Häufigkeiten, normiert auf die Fläche der Region.

Komplexität

If F is the area of the region the runtime complexity is $O(F + 255)$.

Ergebnis

`gray_histo` liefert den Wert 2 (H_MSG_TRUE), falls das Bild definierte Grauwerte hat und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gray_histo` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`histo_to_thresh`, `gen_region_histo`

Alternativen

`min_max_gray`, `intensity`

Siehe auch

`set_paint`, `disp_image`, `histo_2dim`, `scale_image_max`, `entropy_gray`

Modul

Image filters

`gray_projections` (Region, Image : : Mode : HorProjection, VertProjection)

Horizontale und vertikale Grauwertprojektionen.

`gray_projections` berechnet die horizontalen und vertikalen Grauwertprojektionen, d.h. die Mittelwerte der Grauwerte des Eingabebildes `Image` in horizontaler und vertikaler Richtung innerhalb der Eingaberegion `Region`.

Falls `Mode` = **'simple'** gewählt wird, erfolgt die Projektion entlang der Koordinatenachsen des Bildes, d.h.:

$$\texttt{HorProjection}(r) = \sum_{(r+r',c+c') \in \texttt{Region}} \texttt{Image}(r+r', c+c')$$

$$\texttt{VertProjection}(c) = \sum_{(r+r',c+c') \in \texttt{Region}} \texttt{Image}(r+r', c+c')$$

Hierbei ist (r', c') der linke obere Eckpunkt des kleinsten umschließenden achsenparallelen Rechtecks der Eingaberegion (siehe `smallest_rectangle1`). Die horizontale Projektion liefert also eine eindimensionale Funktion, die den vertikalen Grauwertverlauf wiedergibt. Entsprechend liefert die vertikale Projektion eine Funktion, die den horizontalen Grauwertverlauf wiedergibt.

Falls `Mode` = **'rectangle'** gewählt wird, erfolgt die Projektion entlang der Hauptachsen des kleinsten umschließenden nicht-achsenparallelen Rechtecks der Eingaberegion (siehe `smallest_rectangle2`). Hierbei entspeicht die horizontale Projektionsrichtung der kleinen Achse des Rechtecks, die vertikale Richtung der großen Achse. In diesem Modus werden alle Grauwerte innerhalb des kleinsten umschließenden nicht-achsenparallelen Rechtecks der Eingaberegion zur Berechnung der Projektionen verwendet.

Parameter

- ▷ **`Region`** (input_object) ..... region ⇝ *Hobject*
  Zu untersuchende Region.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte / int2
  Grauwerte für die Messung.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Art der Projektionsberechnung.
  Defaultwert : 'simple'
  Werteliste : `Mode` ∈ {'simple', 'rectangle'}
- ▷ **`HorProjection`** (output_control) ..... real-array ⇝ *real*
  Horizontale Projektion.
- ▷ **`VertProjection`** (output_control) ..... real-array ⇝ *real*
  Vertikale Projektion.

Parallelisierungsinformation

`gray_projections` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Modul

Image filters

`histo_2dim ( Regions, ImageCol, ImageRow : Histo2Dim : : )`

Histogramm von zweikanaligen Grauwertbildern.

`histo_2dim` berechnet das 2-dimensionale Histogramm von zwei Bildern innerhalb von `Regions`. Die Grauwerte von Kanal 1 (`ImageCol`) werden als Zeilenindex, diejenigen von Kanal 2 (`ImageRow`) als Spaltenindex interpretiert. Der Grauwert an einem Punkt $P(g1, g2)$ im Ergebnisbild `Histo2Dim` gibt die Häufigkeit der Grauwertkombination ($g1$,$g2$) an, wobei $g1$ den Zeilen- und $g2$ den Spaltenindex darstellt.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Region, in der das Histogramm berechnet werden soll.
- ▷ **`ImageCol`** (input_object) ..... image ⇝ *Hobject* : byte / direction / cyclic / int1
  Kanal 1.
- ▷ **`ImageRow`** (input_object) ..... image ⇝ *Hobject* : byte
  Kanal 2.
- ▷ **`Histo2Dim`** (output_object) ..... image ⇝ *Hobject* : int4
  Zu berechnendes Histogramm.

Beispiel

```
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).
```

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F + 256^2)$.

Ergebnis

histo_2dim liefert den Wert 2 (H_MSG_TRUE), falls die beiden Bilder definierte Grauwerte haben. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system (::'no_object_result',<Result>:), das bei leerer Region mit set_system (::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

histo_2dim ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

decompose3, decompose2, draw_region

Mögliche Nachfolgerfunktionen

threshold, class_2dim_sup, pouring, local_max, gray_skeleton

Alternativen

gray_histo

Siehe auch

get_grayval

Modul

Image filters

intensity (Regions, Image : : : Mean, Deviation)

Mittelwert und Abweichung der Grauwerte.

intensity berechnet den Mittelwert und die Abweichung der Grauwerte im Eingabebild innerhalb von Regions. Sei R eine Region, p ein Punkt aus R mit Grauwert $g(p)$ und F die Fläche ($F = |R|$), dann sind die Merkmale definiert durch:

$$\texttt{Mean} := \frac{\sum_{p \in R} g(p)}{F}$$

$$\texttt{Deviation} := \sqrt{\frac{\sum_{p \in R} (g(p) - \texttt{Mean})^2}{F}}$$

Achtung

Es ist zu beachten, daß die Berechnung von Deviation nicht der üblichen Definition folgt. Sie ist hier so festgelegt, daß sie für ein Bild mit nur einem Pixel den Wert 0.0 liefert.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, deren Merkmale berechnet werden sollen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Grauwertdaten.
- ▷ **Mean** (output_control) ..... real(-array) ⇝ *real*
  Mittlerer Grauwert einer Region.
- ▷ **Deviation** (output_control) ..... real(-array) ⇝ *real*
  Abweichung der Grauwerte innerhalb einer Region.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F)$.

Ergebnis

intensity liefert den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system (::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

intensity ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`threshold`

Alternativen

`select_gray`, `min_max_gray`

Siehe auch

`mean_image`, `mean_image`, `gray_histo`

Modul

Image filters

`min_max_gray` (Regions, Image : : Percent : Min, Max, Range)

Minimale und maximale Grauwerte innerhalb von Regionen.

`min_max_gray` erstellt das Histogramm der absoluten Häufigkeiten der Grauwerte innerhalb von `Regions` im Eingabebild `Image` (siehe `gray_histo`) und berechnet die Anzahl der Bildpunkte, die `Percent` der Fläche des Eingabebildes entspricht. Daraufhin wird auf beiden Seiten des Histogramms um diese Anzahl von Bildpunkten nach innen gegangen und der kleinste und größte Grauwert bestimmt:

z.B.:

Fläche = 60, Prozent = 5, d.h. 3 Bildpunkte
Histogramm = [2,8,0,7,13,0,0,...,0,10,10,5,3,1,1]
$\Rightarrow$ Maximum = 255, Minimum = 0, Spanne = 255

`min_max_gray` liefert: Maximum = 253, Minimum = 1, Spanne = 252

Wird `Percent` auf 50 gesetzt, dann ist `Min` = `Max` = Median. Wenn `Percent` gleich 0 ist, dann wird kein Histogramm berechnet, um die Laufzeit zu verbessern.

Achtung

Bei int2, int4- und real-Bildern muß `Percent` = 0 sein.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) $\rightsquigarrow$ *Hobject*
  Regionen, deren Merkmale berechnet werden sollen.
- ▷ **`Image`** (input_object) ..... image $\rightsquigarrow$ *Hobject* : byte / int1 / int2 / int4 / real
  Grauwertdaten.
- ▷ **`Percent`** (input_control) ..... number $\rightsquigarrow$ *real* / integer
  Prozentsatz unterhalb (oberhalb) des absoluten Maximums (Minimums).
  Defaultwert : 0
  Wertevorschläge : `Percent` $\in \{0, 1, 2, 5, 7, 10, 15, 20, 30, 40, 50\}$
  Restriktion : $(\texttt{0} \leq \texttt{Percent}) \wedge (\texttt{Percent} \leq \texttt{50})$
- ▷ **`Min`** (output_control) ..... real(-array) $\rightsquigarrow$ *real*
  „Minimaler“ Grauwert.
- ▷ **`Max`** (output_control) ..... real(-array) $\rightsquigarrow$ *real*
  „Maximaler“ Grauwert.
  Zusicherung : $\texttt{Max} \geq \texttt{Min}$
- ▷ **`Range`** (output_control) ..... real(-array) $\rightsquigarrow$ *real*
  Differenz aus Max und Min.
  Zusicherung : $\texttt{Range} \geq \texttt{0}$

Beispiel

```
/* Threshold segmentation with training region: */
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
min_max_gray(Region,Image,5,Min,Max,_)
threshold(Bild,Seg,Min,Max)
disp_region(Seg,WindowHandle).
```

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F)$ (falls `Percent` = 0), $O(F+255)$ sonst.

Ergebnis

`min_max_gray` liefert den Wert 2 (H_MSG_TRUE), falls das Eingabebild definierte Grauwerte besitzt und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`min_max_gray` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`draw_region`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `threshold`, `regiongrowing`

Mögliche Nachfolgerfunktionen

`threshold`

Alternativen

`select_gray`, `intensity`

Siehe auch

`gray_histo`, `scale_image`, `scale_image_max`, `learn_ndim_norm`

Modul

Image filters

moments_gray_plane (Regions, Image : : : MRow, MCol, Alpha, Beta, Mean)

Berechnung der Grauwertmomente und der Approximation durch eine Ebene.

`moments_gray_plane` berechnet die Grauwertmomente und die Approximation der Grauwerte durch eine Ebene. Die Berechnung erfolgt nach folgender Formel:

$$\texttt{MRow} = \frac{1}{F^2} \sum_{(r,c) \in \texttt{Regions}} (r-\overline{r})(\texttt{Image}(r,c) - \texttt{Mean}) \qquad \texttt{MCol} = \frac{1}{F^2} \sum_{(r,c) \in \texttt{Regions}} (c-\overline{c})(\texttt{Image}(r,c) - \texttt{Mean})$$

$$\texttt{Alpha} = \frac{\texttt{MRow}Fm_{02} - m_{11}\texttt{MCol}}{F} m_{20}m_{02} - m_{11}^2 \qquad \texttt{Beta} = \frac{m_{20}\texttt{MCol}F - \texttt{MRow}Fm_{11}}{m_{20}m_{02} - m_{11}^2}$$

wobei F die Fläche, $\overline{r}$, $\overline{c}$ der Schwerpunkt, m_{11}, m_{20} und m_{02} die normierten Momente von `Regions` sind.

Die Parameter `Alpha`, `Beta` und `Mean` beschreiben dabei eine Ebene über der Region:

$$\texttt{Image}'(r,c) = \texttt{Alpha}(r-\overline{r}) + \texttt{Beta}(c-\overline{c}) + \texttt{Mean}$$

`Alpha` gibt also die Steigung in Richtung der Zeilenachse (nach „unten“), `Beta` die Steigung in Richtung der Spaltenachse (nach „rechts“) an.

Achtung

Bei Regionen, die entweder aus nur einem Pixel bestehen oder aus mehreren Pixeln, die auf einer gemeinsamen Linie liegen, ist die Approximation der Ebene unterbestimmt und daher nicht eindeutig lösbar. In diesem Fall werden `Alpha` und `Beta` auf 0 gesetzt.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte / real
  Zugehörige Grauwerte.
- ▷ **MRow** (output_control) ..... real(-array) ⇝ *real*
  Gemischte Momente der Zeile.

▷ **MCol** (output_control) ..... real(-array) ⇝ *real*
Gemischte Momente der Spalte.

▷ **Alpha** (output_control) ..... real(-array) ⇝ *real*
Parameter Alpha der Ebenengleichung.

▷ **Beta** (output_control) ..... real(-array) ⇝ *real*
Parameter Beta der Ebenengleichung.

▷ **Mean** (output_control) ..... real(-array) ⇝ *real*
Mittlerer Grauwert.

Ergebnis

`moments_gray_plane` liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten (**byte**) eingegeben wird und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`moments_gray_plane` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`draw_region`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `threshold`, `regiongrowing`

Siehe auch

`intensity`, `moments_region_2nd`

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, pp 75-76

Modul

Image filters

plane_deviation (Regions, Image : : : Deviation)

Abweichung der Grauwerte von der approximierten Bildebene.

`plane_deviation` berechnet die Abweichung der Grauwerte in `Image` von der Approximation der Grauwerte durch eine Ebene. Im Gegensatz zur Standardabweichung bei `intensity` erhalten auch geneigte Grauwertflächen den Wert Null. Die Berechnung der Grauwertebene erfolgt gemäß `gen_image_gray_ramp`.

Sei F die Fläche, α, β, μ die Parameter der Bildebene und (r', c') der Schwerpunkt, dann ist `Deviation` definiert durch:

$$\texttt{Deviation} = \sqrt{\frac{sum_{(r,c)\in\texttt{Regions}}((\alpha(r-r')+\beta(c-c')+\mu)-\texttt{Image}(r,c))^2}{F}} \quad .$$

Achtung

Es ist zu beachten, daß die Berechnung von `Deviation` nicht der üblichen Definition folgt. Sie ist hier so festgelegt, daß sie für ein Bild mit nur einem Pixel den Wert 0.0 liefert.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen, deren Abweichung berechnet werden sollen.

▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
Grauwertdaten.

▷ **Deviation** (output_control) ..... real(-array) ⇝ *real*
Abweichung der Grauwerte innerhalb einer Region.

Komplexität

Sei F die Fläche der Region, dann beträgt die Laufzeitkomplexität $O(F)$.

Ergebnis

`plane_deviation` liefert den Wert 2 (H_MSG_TRUE), falls `Image` vom Typ **byte** ist. Das Verhalten bei leerer

Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`plane_deviation` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Alternativen

`intensity`, `gen_image_gray_ramp`, `sub_image`

Siehe auch

`moments_gray_plane`

Modul

Image filters

`select_gray` (Regions, Image : SelectedRegions : Features, Operation, Min, Max :)

Auswahl von Regionen aufgrund von Grauwertmerkmalen.

`select_gray` hat als Eingabe eine Anzahl von Regionen (`Regions`). Für jede dieser Regionen werden die Merkmale (`Features`) berechnet. Wenn jedes (`Operation` = 'and') oder mindestens eines (`Operation` = 'or') der berechneten Merkmale in den durch die Parameter festgelegten Grenzen liegt, wird die Region in die Ausgabe übernommen (dupliziert). Der Parameter `Image` enthält ein Bild, das die Grauwerte zur Berechnung der Merkmale liefert.

Bedingung:

$$\texttt{Min}_i \leq \texttt{Features}_i(\texttt{Regions}, \texttt{Image}) \leq \texttt{Max}_i$$

Mögliche Werte für `Features`:

'area'	Volumen des Grauwertgebirges (vgl. `area_center_gray`)
'row'	Zeilenindex des Schwerpunkts des Grauwertgebirges (vgl. `area_center_gray`)
'column'	Spaltenindex des Schwerpunkts des Grauwertgebirges (vgl. `area_center_gray`)
'ra'	Hauptradius der äquivalenten Ellipse (vgl. `elliptic_axis_gray`)
'rb'	Nebenradius der äquivalenten Ellipse (vgl. `elliptic_axis_gray`)
'phi'	Hauptradius der äquivalenten Ellipse (vgl. `elliptic_axis_gray`)
'min'	Minimaler Grauwert (vgl. `min_max_gray`)
'max'	Maximaler Grauwert (vgl. `min_max_gray`)
'mean'	Mittlerer Grauwert (vgl. `intensity`)
'deviation'	Standardabweichung der Grauwerte (vgl. `intensity`)
'plane_deviation'	Standardabweichung von der approximierten Grauwertebene (vgl. `plane_deviation`)
'anisotropy'	Anisotropie (vgl. `entropy_gray`)
'entropy'	Entropie (vgl. `entropy_gray`)
'fuzzy_entropy'	Fuzzy-Entropie der Region (vgl. `fuzzy_entropy`, mit einer Fuzzyfunktion von Apar=0 bis Cpar=255)
'fuzzy_perimeter'	Fuzzy-Umfang der Region (vgl. `fuzzy_perimeter`, mit einer Fuzzyfunktion von Apar=0 bis Cpar=255)
'moments_row'	Gemischte Grauwertmomente der Zeile (vgl. `moments_gray_plane`)
'moments_column'	Gemischte Grauwertmomente der Spalte (vgl. `moments_gray_plane`)
'alpha'	Ebenengleichung, Parameter Alpha (vgl. `moments_gray_plane`)
'beta'	Ebenengleichung, Parameter Beta (vgl. `moments_gray_plane`)

Achtung

Wird nur ein Merkmal verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie eingegeben werden.

Parameter

- ▷ **Regions** (input_object) ..... region-array ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte / int2 / int4 / real
  Grauwertdaten.
- ▷ **SelectedRegions** (output_object) ..... region-array ⇝ *Hobject*
  Regionen, deren Merkmale innerhalb der Grenzen liegen.
- ▷ **Features** (input_control) ..... string(-array) ⇝ *string*
  Namen der Merkmale.
  Defaultwert : 'mean'
  Werteliste : `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 ⇝ *string*
  Logische Verknüpfung der Merkmale.
  Defaultwert : 'and'
  Werteliste : `Operation` ∈ {'and', 'or'}
- ▷ **Min** (input_control) ..... number(-array) ⇝ *real* / integer
  Untergrenze(n) der Merkmale.
  Defaultwert : 128.0
  Wertevorschläge : `Min` ∈ {0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0}
- ▷ **Max** (input_control) ..... number(-array) ⇝ *real* / integer
  Obergrenze(n) der Merkmale.
  Defaultwert : 255.0
  Wertevorschläge : `Max` ∈ {0.5, 1.0, 10.0, 20.0, 50.0, 128.0, 255.0, 1000.0}

Komplexität

Sei F die Fläche der Region und N die Anzahl der Merkmale, dann beträgt die Laufzeitkomplexität $O(F * N)$.

Ergebnis

`select_gray` liefert den Wert 2 (H_MSG_TRUE), falls das Eingabebild definierte Grauwerten besitzt und die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_gray` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`connection`, `mean_image`, `entropy_image`, `sobel_amp`, `median_separate`

Mögliche Nachfolgerfunktionen

`select_shape`, `select_gray`, `shape_trans`, `reduce_domain`, `count_obj`

Siehe auch

`deviation_image`, `entropy_gray`, `intensity`, `mean_image`, `min_max_gray`, `select_obj`

Modul

Image filters

shape_histo_all (Region, Image : : Feature : AbsoluteHisto, RelativeHisto)

Ausprägung von Merkmalen entlang aller Schwellenwerte.

`shape_histo_all` führt 255 Schwellenwertoperationen innerhalb von `Region` mit den Grauwerten von `Image` durch. Der Eintrag i im Histogramm entspricht dann der Anzahl der Zusammenhangskomponenten/Löcher dieses mit der Schwelle i segmentierten Bildes (`Feature` = **'connected_components'**, **'holes'**) bzw. dem Mittelwert der Merkmalswerte der so segmentierten Regionen (`Feature` = **'convexity'**, **'compactness'**, **'ansisometry'**).

Mit `set_paint(::WindowHandle,'component_histogram':)` und `disp_image` kann das Histogramm auch direkt als Graphik ausgegeben werden.

Achtung

`shape_histo_all` erwartet als Eingabe eine Region und genau ein Grauwertbild. Wegen der Mächtigkeit dieser Prozedur ist die Laufzeit von `shape_histo_all` relativ groß!

Parameter

- ▷ **`Region`** (input_object) ..... region ⇝ *Hobject*
  Region, in der die Merkmale betrachtet werden.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte
  Grauwertdaten.
- ▷ **`Feature`** (input_control) ..... string ⇝ *string*
  Merkmal, das untersucht werden soll.
  Defaultwert : 'connected_components'
  Werteliste : `Feature` ∈ {'connected_components', 'convexity', 'compactness', 'anisometry', 'holes'}
- ▷ **`AbsoluteHisto`** (output_control) ..... histogram-array ⇝ *real* / integer
  Absolute Verteilung des Merkmals.
- ▷ **`RelativeHisto`** (output_control) ..... histogram-array ⇝ *real*
  Relative Verteilung des Merkmals.

Beispiel

```
/* Simulation von shape_histo_all mit Merkmal 'connected_components': */
my_shape_histo_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()
```

Komplexität

Sei F die Fläche der Eingaberegion und N die mittlere Anzahl der Zusammenhangskomponenten, dann beträgt die Laufzeitkomplexität $O(255(F + \sqrt{F}\sqrt{N}))$.

Ergebnis

`shape_histo_all` liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten eingegeben wird. Das Verhalten bei leerer Eingabe (keine Bilder) wird mit `set_system(::'no_object_result',<Result>:)`, das bei leerer Region mit `set_system(::'empty_region_result',<Result>:)` festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`shape_histo_all` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`histo_to_thresh`, `threshold`, `gen_region_histo`

Alternativen

`shape_histo_point`

Siehe auch

`connection`, `convexity`, `compactness`, `connect_and_holes`, `entropy_gray`, `gray_histo`, `set_paint`, `count_obj`

Modul

Image filters

shape_histo_point (Region, Image : : Feature, Row, Column : AbsoluteHisto, RelativeHisto)

Ausprägung von Merkmalen entlang aller Schwellenwerte.

shape_histo_point führt wie shape_histo_all 255 Schwellenwertoperationen innerhalb von Region mit den Grauwerten von Image durch. Im Gegensatz zu shape_histo_all wird hier nur diejenige segmentierte Region betrachtet, die den Punkt (Row, Column) enthält. Der Eintrag i im Histogramm entspricht dann der Anzahl der Löcher dieser mit der Schwelle i segmentierten Region (Feature = **'holes'**) bzw. dem Merkmalswert der Region (Feature = **'convexity'**, **'compactness'**, **'ansisometry'**).

Mit set_paint(::WindowHandle,'component_histogram':) und disp_image kann das Histogramm auch direkt als Graphik ausgegeben werden.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Region, in der die Merkmale betrachtet werden.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
  Grauwertdaten.
- ▷ **Feature** (input_control) ..... string ⇝ *string*
  Merkmal, das untersucht werden soll.
  Defaultwert : 'convexity'
  Werteliste : Feature ∈ {'convexity', 'compactness', 'anisometry', 'holes'}
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Punktes, den die Region enthalten muß.
  Defaultwert : 256
  Wertevorschläge : Row ∈ {10, 50, 100, 200, 300, 400}
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Punktes, den die Region enthalten muß.
  Defaultwert : 256
  Wertevorschläge : Column ∈ {10, 50, 100, 200, 300, 400}
- ▷ **AbsoluteHisto** (output_control) ..... histogram-array ⇝ *real* / integer
  Absolute Verteilung des Merkmals.
- ▷ **RelativeHisto** (output_control) ..... histogram-array ⇝ *real*
  Relative Verteilung des Merkmals.

Ergebnis

shape_histo_point liefert den Wert 2 (H_MSG_TRUE), falls ein Bild mit definierten Grauwerten eingegeben wird. Das Verhalten bei leerer Eingabe (keine Bilder) wird mit set_system(::'no_object_result',<Result>:), das bei leerer Region mit set_system(::'empty_region_result',<Result>:) festgelegt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

shape_histo_point ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_mbutton, area_center

Mögliche Nachfolgerfunktionen

histo_to_thresh, threshold, gen_region_histo

Alternativen

shape_histo_all

Siehe auch

connection, connect_and_holes, convexity, compactness, set_paint

Modul

Image filters

1.8 Typanpassung

complex_to_real (ImageComplex : ImageReal, ImageImaginary : :)

Umwandlung eines komplexwertigen Bildes in zwei real-Bilder.

complex_to_real wandelt das komplexwertige Bild ImageComplex in zwei reellwertige Bilder ImageReal und ImageImaginary um, die den Real- bzw. Imaginärteil des komplexen Bildes enthalten.

Parameter

- ▷ **ImageComplex** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Komplexes Bild.
- ▷ **ImageReal** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Realteil.
- ▷ **ImageImaginary** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Imaginärteil.

Parallelisierungsinformation

complex_to_real ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Siehe auch

real_to_complex

Modul

Image / region / XLD management

convert_image_type (Image : ImageConverted : NewType :)

Umwandeln des Bildtyps.

convert_image_type konvertiert Bilder eines beliebigen Bildtyps in einen beliebigen Bildtyp. Erfolgt die Konvertierung von einem größeren in einen kleineren Grauwertbereich (z.B. von ’int4’ nach ’byte’) werden zu große oder zu kleine Werte einfach „geclippt“. Es empfiehlt sich daher, vor Aufruf der Routine den Wertebereich der Grauwerte im Eingabebild mittels scale_image entsprechend anzupassen.

Achtung

Stimmen der Ziel- und der Quelltyp überein, dann wird keine neue Bildmatrix angelegt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : any
  Bilder, deren Grauwerttyp geändert werden sollen.
- ▷ **ImageConverted** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : any
  Ergebnisbild.
- ▷ **NewType** (input_control) ..... string ⇝ *string*
  Gewünschter Bildtyp (i.e. Typ der Grauwerte).
  Defaultwert : ’byte’
  Werteliste : NewType ∈ {’int1’, ’int2’, ’int4’, ’byte’, ’real’, ’direction’, ’cyclic’, ’complex’, ’dvf’, ’lut’}

Ergebnis

convert_image_type liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (’no_object_result’,<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

convert_image_type ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

scale_image

Siehe auch

scale_image, abs_image

Modul

Image / region / XLD management

dvf_to_int1 (VectorField : Row, Col : :)

Umwandlung eines Verschiebungsvektorfeldes in zwei int1-Bilder.

dvf_to_int1 wandelt das Verschiebungsvektorfeld-Bild (VectorField) in zwei Int1-Bilder Row und Col um. Die Ausgabe-Bilder enthalten die Verschiebungs-Komponenten in x- bzw. y-Richtung.

Parameter

- ▷ **VectorField** (input_object) ..... image(-array) ⇝ *Hobject* : dvf
  Verschiebungsvektorfeld.
- ▷ **Row** (output_object) ..... image(-array) ⇝ *Hobject* : int1
  Verschiebung in y.
- ▷ **Col** (output_object) ..... image(-array) ⇝ *Hobject* : int1
  Verschiebung in x.

Parallelisierungsinformation

dvf_to_int1 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

mean_image, optical_flow_match, fill_dvf

Siehe auch

optical_flow_match

Modul

Image / region / XLD management

int1_to_dvf (Row, Col : VectorField : :)

Umwandlung von zwei int1-Bildern in ein Verschiebungsvektorfeld.

int1_to_dvf wandelt zwei Int1-Bilder Row und Col in ein Verschiebungsvektorfeld-Bild VectorField um. Die Eingabe-Bilder enthalten dabei die Verschiebungs-Komponenten in x- bzw. y-Richtung.

Parameter

- ▷ **Row** (input_object) ..... image(-array) ⇝ *Hobject* : int1
  Verschiebung in y.
- ▷ **Col** (input_object) ..... image(-array) ⇝ *Hobject* : int1
  Verschiebung in x.
- ▷ **VectorField** (output_object) ..... image(-array) ⇝ *Hobject* : dvf
  Verschiebungsvektorfeld

Parallelisierungsinformation

int1_to_dvf ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

dvf_to_int1

Mögliche Nachfolgerfunktionen

optical_flow_match, fill_dvf

Modul

Image / region / XLD management

real_to_complex (ImageReal, ImageImaginary : ImageComplex : :)

Umwandlung von zwei real-Bildern in ein komplexes Bild.

`real_to_complex` wandelt zwei reellwertige Bilder `ImageReal` und `ImageImaginary` (Real- bzw. Imaginärteil eines komplexen Bildes) in ein komplexwertiges Bild `ImageComplex` um.

Parameter

- ▷ **`ImageReal`** (input_object) ..... image(-array) ⇝ *Hobject* : real
  Realteil.
- ▷ **`ImageImaginary`** (input_object) ..... image(-array) ⇝ *Hobject* : real
  Imaginärteil.
- ▷ **`ImageComplex`** (output_object) ..... image(-array) ⇝ *Hobject* : complex
  Komplexes Bild

Parallelisierungsinformation

`real_to_complex` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Siehe auch

`complex_to_real`

Modul

Image / region / XLD management

1.9 Zugriff

`get_grayval` (`Image` : : `Row, Column` : `Grayval`)

Zugriff auf Grauwerte eines Bildobjektes.

`Grayval` ist ein Tupel von Gleitpunkt- bzw. ganzen Zahlen, die die Grauwerte mehrerer Bildpunkte von `Image` liefert. Die Zeilenkoordinaten der Bildpunkte stehen im Tupel `Row`, die Spalten in `Column`.

Achtung

Der Typ der Werte von `Grayval` hängt von dem Typ der Grauwerte ab.
Es kann auch auf Grauwerte zugegriffen werden, die nicht zum Bild gehören. Hier ist nicht sichergestellt, wie diese belegt sind.
Der Aufwand von `get_grayval` ist relativ hoch. Für die Programmierung von Bildverarbeitungsoperationen wie Filter ist sie nicht geeignet. Hier ist es sinnvoller die Prozedur `get_image_pointer1` oder direkt die C-Schnittstelle zum Einbinden von eigenen Prozeduren zu verwenden.

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Bild, auf dessen Grauwerte zugegriffen werden soll.
- ▷ **`Row`** (input_control) ..... point.y(-array) ⇝ *integer*
  Zeilennummern der zu betrachtenden Bildpunkte.
  Defaultwert : 0
  Wertevorschläge : `Row` ∈ {0, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ `Row` ≤ 32768 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : (`0` ≤ `Row`) ∧ (`Row` < height(`Image`))
- ▷ **`Column`** (input_control) ..... point.x(-array) ⇝ *integer*
  Spaltennummern der zu betrachtenden Bildpunkte.
  Defaultwert : 0
  Wertevorschläge : `Column` ∈ {0, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ `Column` ≤ 32768 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Parameteranzahl : `Column` = `Row`
  Restriktion : (`0` ≤ `Column`) ∧ (`Column` < width(`Image`))
- ▷ **`Grayval`** (output_control) ..... grayval(-array) ⇝ *real* / integer
  Grauwerte an den angegebenen Bildpunkten.
  Parameteranzahl : `Grayval` = `Row`

Ergebnis

Bei korrekter Besetzung der Parameter liefert `get_grayval` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_grayval` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_image`

Alternativen

`get_image_pointer1`

Siehe auch

`set_grayval`

Modul

Image / region / XLD management

`get_image_pointer1` (Image : : : Pointer, Type, Width, Height)

Zugriff auf den Zeiger eines Kanals.

`get_image_pointer1` liefert einen C-Pointer auf den ersten Kanal des Bildes `Image`. Außerdem wird der Bildtyp (`Type` = ’byte’, ’integer’, ’float’ etc.) und die Bildgröße (Breite und Höhe) ausgegeben. In der Folge ist dann über den Zeiger ein direkter Zugriff auf die Bilddaten in der HALCON-Datenbank von der HALCON Wirtssprache aus möglich. Ein Bild ist in HALCON als ein Vektor von Bildzeilen abgelegt.

Achtung

`get_image_pointer1` sollte nur zum Eintragen in neu erzeugte Bilder verwendet werden, da sonst die Grauwerte von anderen Bildern überschrieben werden könnten (siehe Relationale Struktur).

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Eingabebild.
- ▷ **`Pointer`** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Bilddaten in der HALCON-Datenbank.
- ▷ **`Type`** (output_control) ..... string ⇝ *string*
  Typ des Bildes.
  Werteliste : `Type` ∈ {’int1’, ’int2’, ’int4’, ’byte’, ’real’, ’direction’, ’cyclic’, ’complex’, ’dvf’, ’lut’}
- ▷ **`Width`** (output_control) ..... extent.x ⇝ *integer*
  Breite des Bildes.
- ▷ **`Height`** (output_control) ..... extent.y ⇝ *integer*
  Höhe des Bildes.

Beispiel

```
Hobject  Bild;
char     typ[128];
long     width,height;
unsigned char *ptr;

read_image(&Bild,"fabrik");
get_image_pointer1(Bild,(long*)&ptr,typ,&width,&height);
```

Ergebnis

`get_image_pointer1` liefert den Wert 2 (H_MSG_TRUE), falls genau ein Bild übergeben wurde. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_image_pointer1` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_image`

Alternativen

`set_grayval`, `get_grayval`, `get_image_pointer3`

Siehe auch

`paint_region`, `paint_gray`

Modul

Image / region / XLD management

`get_image_pointer1_rect` (`Image : : : PixelPointer, Width, Height, VerticalPitch, HorizontalBitPitch, BitsPerPixel`)

Zugriff auf Bilddatenzeiger und Bildpunkte innerhalb des kleinsten umschließenden Rechtecks der Domäne des Eingabebildes.

Der Operator `get_image_pointer1_rect` liefert den Zeiger `PixelPointer` auf den Anfang der Bilddaten, die sich innerhalb des kleinsten umschließenden Rechtecks der Domäne von `Image` befinden. `VerticalPitch` entspricht der Breite des Eingabebildes `Image` mal der Anzahl der Bytes pro Pixel (`HorizontalBitPitch` / 8). `Width` und `Height` entsprechen der Größe des kleinsten umschließenden Rechtecks der Eingaberegion. `HorizontalBitPitch` ist der horizontale Abstand (in Bits) benachbarter Pixel. `BitsPerPixel` ist die Anzahl der verwendeten Bits pro Pixel. `get_image_pointer1_rect` ist symmetrisch zu `gen_image1_rect`.

Achtung

`get_image_pointer1_rect` sollte nur zum Eintragen in neu erzeugte Bilder verwendet werden, da sonst die Grauwerte von anderen Bildern überschrieben werden könnten (siehe relationale Struktur).

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte / int2 / int4
  Eingabebild (Himage).
- ▷ **`PixelPointer`** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Bilddaten.
- ▷ **`Width`** (output_control) ..... extent.x ⇝ *integer*
  Breite des Ausgabebildes.
- ▷ **`Height`** (output_control) ..... extent.y ⇝ *integer*
  Höhe des Ausgabebildes.
- ▷ **`VerticalPitch`** (output_control) ..... integer ⇝ *integer*
  Breite(Eingabebild)*(HorizontalBitPitch/8).
- ▷ **`HorizontalBitPitch`** (output_control) ..... integer ⇝ *integer*
  Abstand benachbarter Pixel in Bits.
  Defaultwert : 8
  Werteliste : `HorizontalBitPitch` ∈ {8, 16, 32}
- ▷ **`BitsPerPixel`** (output_control) ..... integer ⇝ *integer*
  Anzahl verwendeter Bits pro Pixel.
  Defaultwert : 8
  Werteliste : `BitsPerPixel` ∈ {8, 16, 32}

Beispiel

```
Hobject image,reg,imagereduced;
char typ[128];
long width,height,vert_pitch,hori_bit_pitch,bits_per_pix, winID;
unsigned char *ptr;

open_window(0,0,512,512,"black",winID);
read_image(&image,"monkey");
```

```
draw_region(&reg,winID);
reduce_domain(image,reg,&imagereduced);
get_image_pointer1_rect(imagereduced,(long*)&ptr,&width,&height,
                        &vert_pitch,&hori_bit_pitch,&bits_per_pix);
```

Ergebnis

get_image_pointer1_rect liefert den Wert 2 (H_MSG_TRUE), falls genau ein Bild übergeben wurde. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_image_pointer1_rect ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image

Alternativen

set_grayval, get_grayval, get_image_pointer3, get_image_pointer1

Siehe auch

paint_region, paint_gray, gen_image1_rect

Modul

Image / region / XLD management

get_image_pointer3 (ImageRGB : : : PointerRed, PointerGreen, PointerBlue, Type, Width, Height)

Zugriff auf die Zeiger eines Farbbildes.

get_image_pointer3 liefert einen C-Pointer auf die drei Kanäle eines Farbbildes (ImageRGB). Außerdem wird der Bildtyp (Type = ’byte’, ’int2’, ’float’ etc.) und die Bildgröße (Width und Height) ausgegeben. In der Folge ist dann über den Zeiger ein direkter Zugriff auf die Bilddaten in der HALCON-Datenbank von der HALCON Wirtssprache aus möglich. Ein Bild ist in HALCON als ein Vektor von Bildzeilen abgelegt. Die drei Kanäle müßen den geleichen Pixeltyp und die gleiche Größe haben.

Achtung

Es darf nur ein Bild übergeben werden. get_image_pointer3 sollte nur zum Eintragen in neu erzeugte Bilder verwendet werden, da sonst die Grauwerte von anderen Bildern überschrieben werden könnten (siehe Relationale Struktur).

Parameter

- **ImageRGB** (input_object) ..... image ⇝ *Hobject*
  Eingabebild.
- **PointerRed** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Pixel des ersten Kanals.
- **PointerGreen** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Pixel des zweiten Kanals.
- **PointerBlue** (output_control) ..... pointer ⇝ *integer*
  Zeiger auf die Pixel des dritten Kanals.
- **Type** (output_control) ..... string ⇝ *string*
  Typ des Bildes.
  Werteliste : Type ∈ {’int1’, ’int2’, ’int4’, ’byte’, ’real’, ’direction’, ’cyclic’, ’complex’, ’dvf’, ’lut’}
- **Width** (output_control) ..... extent.x ⇝ *integer*
  Breite des Bildes.
- **Height** (output_control) ..... extent.y ⇝ *integer*
  Höhe des Bildes.

Ergebnis

get_image_pointer3 liefert den Wert 2 (H_MSG_TRUE), falls genau ein Bild übergeben wird. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system

('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_image_pointer3 ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image

Alternativen

set_grayval, get_grayval, get_image_pointer1

Siehe auch

paint_region, paint_gray

Modul

Image / region / XLD management

get_image_time (Image : : : MSecond, Second, Minute, Hour, Day, YDay, Month, Year)

Abfragen der Zeit, zu der das Bild erzeugt wurde.

get_image_time liefert den Zeitpunkt, zu dem das Bild erzeugt wurde.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Eingabebild.
- ▷ **MSecond** (output_control) ..... integer ⇝ *integer*
  Milli-Sekunden (0..999).
- ▷ **Second** (output_control) ..... integer ⇝ *integer*
  Sekunden (0..59).
- ▷ **Minute** (output_control) ..... integer ⇝ *integer*
  Minuten (0..59).
- ▷ **Hour** (output_control) ..... integer ⇝ *integer*
  Stunden (0..11).
- ▷ **Day** (output_control) ..... integer ⇝ *integer*
  Tag innerhalb des Monats (1..31).
- ▷ **YDay** (output_control) ..... integer ⇝ *integer*
  Tag innerhalb des Jahres (1..365).
- ▷ **Month** (output_control) ..... integer ⇝ *integer*
  Monat (1..12).
- ▷ **Year** (output_control) ..... integer ⇝ *integer*
  Jahr (xxxx).

Ergebnis

get_image_time liefert den Wert 2 (H_MSG_TRUE), falls genau ein Bild übergeben wurde. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_image_time ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image, grab_image

Siehe auch

count_seconds

Modul

Image / region / XLD management

Kapitel 2

Datei

2.1 Bilddaten

read_image (: Image : FileName :)

Einlesen eines Bildes mit unterschiedlichen Dateiformaten.

read_image liest die angegebenen Bilddateien vom Hintergrundspeicher und erzeugt daraus ein Bild. Es können eine oder mehrere Dateien angegeben werden. Die Region des erzeugten Bildobjektes ist maximal (= alle Bildpunkte der Matrix) gewählt.

Alle Bilddateien, die mit write_image (Format 'ima') geschrieben wurden, haben die Extension **'.ima'**. Zu jedem Bild im HALCON-Format kann eine Beschreibungsdatei vorhanden sein (gleicher Dateiname mit Extension **'.exp'**). Der Typ der Pixeldaten (**byte**, **int4**, **real**) wird ebenfalls der Beschreibungsdatei entnommen. Ist diese Information nicht vorhanden, so wird der Typ **byte** als Voreinstellung verwendet.

Neben dem HALCON-Format können auch TIFF, GIF, BMP, JPEG, PNG, PCX, SUN-Raster, PGM, PPM, PBM und XWD Dateien eingelesen werden. Die Grauwerte bei PBM-Bildern werden auf Werte 0 und 255 gesetzt. Die Dateiformate werden entweder an der Extension (soweit angegeben) oder aufgrund ihrer inneren Struktur der Dateien erkannt. Bei Angabe der Extension wird das Bild schneller gefunden. Bei PGM, PPM und PBM kann die jeweilige Extension (z.B. **'pgm'**) oder auch allgemein **'pnm'** verwendet werden. Bei TIFF ist **'tiff'** und **'tif'** zulässig. Bei Farbbildern wird ein Bild mit drei Farbkanälen (Matrizen) erzeugt, wobei der Rotkanal in der ersten, der Blaukanal in der zweiten und der Grünkanal in der dritten Komponente (Kanalnummer) abgelegt wird.

Bilddateien werden im aktuellen Directory (vorgegeben durch die Betriebssystemumgebung) und im Bilderdirectory von HALCON gesucht. Das Bilderdirectory von HALCON ist in einer UNIX-Umgebung auf **'.'** und **'/usr/local/halcon/images'** voreingestellt und kann mit dem Befehl set_system gesetzt werden. Es kann mehr als ein Bilderdirectory angegeben werden. Dies geschieht, indem man die einzelnen Directories durch einen Doppelpunkt trennt.

Weiterhin kann der Suchpfad mit der Environment-Variablen HALCONIMAGES gesetzt werden (gleicher Aufbau wie bei **'image_dir'**). Beispiel:

```
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"\\*
```

Ebenso sucht HALCON die Bilder in dem Unter-Directory **„images“** (Bilder für die Programmbeispiele). Für das HALCON-Directory wird die Environment-Variable HALCONROOT verwendet.

Achtung

Falls CMYK oder YCCK JPEG-Dateien gelesen werden, nimmt HALCON an, daß diese Dateien der Konvention von Adobe Photoshop folgen, gemäß der die CMYK-Kanäle invertiert gespeichert sind, d.h. 0 repräsentiert 100% Druckfarbe und nicht 0%, wie man es eigentlich erwarten würde. Die Bilder werden gemäß dieser Konvention in RGB-Bilder umgewandelt. Falls die JPEG-Dateien nicht dieser Konvention folgen und die CMYK-Kanäle in der üblichen Weise abspeichern, muß invert_image nach dem Einlesen des Bildes aufgerufen werden.

Beim Lesen von PNG-Dateien mit Alphakanal wird der Alphakanal als zweiter bzw. vierter Kanal des Ausgabebildes zurückgegeben, außer der Alpha-Kanal enthält genau zwei verschiedene Grauwerte. In diesem Fall wird

ein ein- oder dreikanaliges Bild mit eingeschränktem Definitionsbereich zurückgegeben, in dem die Punkte des Definitionsbereiches den Pixeln des Aplha-Kanals mit dem höheren der zwei Grauwerte entspricht.

Parameter

▷ **Image** (output_object) ..... image ⇝ *Hobject* : byte / int4 / real
▷ **FileName** (input_control) ..... filename(-array) ⇝ *string*
Defaultwert : 'fabrik'
Wertevorschläge : FileName ∈ {'monkey', 'fabrik', 'mreut'}

Beispiel

```
/* 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').
```

Ergebnis

Sind die Parameter korrekt, dann liefert read_image den Wert 2 (H_MSG_TRUE). Werden die angegebenen Dateien nicht gefunden, dann liefert read_image den Wert 5 (H_MSG_FAIL). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_image wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive") ausgeführt.

Mögliche Nachfolgerfunktionen

disp_image, threshold, regiongrowing, count_channels, decompose3, class_ndim_norm, gauss_image, fill_interlace, zoom_image_size, zoom_image_factor, crop_part, write_image, rgb1_to_gray

Alternativen

read_sequence

Siehe auch

set_system, write_image

Modul

Image / region / XLD management

read_sequence (: Image : HeaderSize, SourceWidth, SourceHeight, StartRow, StartColumn, DestWidth, DestHeight, PixelType, BitOrder, ByteOrder, Pad, Index, FileName :)

Einlesen von Bildern.

read_sequence liest Bilddaten, die unformatiert vorliegen, aus einer Datei und liefert ein „passendes" Bild zurück. Die Bilddaten müßen dabei Pixel für Pixel zeilenweise hintereinander abgespeichert sein.

Ein etwaiger Datei-Header (mit Länge HeaderSize Bytes) wird übersprungen. Die Parameter SourceWidth und SourceHeight geben die Größe des Bildes auf Datei an. DestWidth und DestHeight geben an, wie groß das eingelesene Bild sein soll. Im einfachsten Fall stimmen diese Parameter überein. Es können aber auch Auschnitte eingelesen werden. Dazu kann mittels StartRow und StartColumn die obere linke Ecke des gewünschten Bildausschnitts festgelegt werden.

An Pixeltypen werden **'bit'**, **'byte'**, **'short'** (16 Bit, ohne Vorzeichen), **'signed_short'** (16 Bit, mit Vorzeichen), **'long'** (32 Bit, mit Vorzeichen), **'swapped_long'** (32 Bit, mit vertauschten Halbwörtern) und **'real'** (32 Bit Gleitkommazahlen) (unterstützt. Weiter erlaubt read_sequence die Extraktion von Komponenten eines RGB Bildes, sofern für jedes Pixel ein Tripel von drei Bytes (in der Reihenfolge „rot", „grün", „blau") in der Bilddatei abgelegt wurde. Für die Rotkomponente ist dann als Pixeltyp **'r_byte'** und entsprechend für die Grün- und Blaukomponente **'g_byte'** bzw. **'b_byte'** zu wählen.

Für die Bitreihenfolge (`BitOrder`) kann **'MSBFirst'** (most significant bit first) bzw. die Umkehrung davon (**'LSBFirst'**) gewählt werden. Analog dazu werden auch die Bytereihenfolgen (`ByteOrder`) **'MSBFirst'** (most significant byte first) bzw. **'LSBFirst'** verarbeitet. Schließlich läßt sich noch ein Alignement (`Pad`) am Zeilenende einstellen: **'byte'**, **'short'** oder **'long'**. Falls in der Datei eine ganze Bildsequenz abgelegt ist, erlaubt der Parameter `Index` die Wahl eines Einzelbildes (beginnend mit Index 1). Environment-Variable HALCONROOT verwendet.

Bilddateien werden im aktuellen Directory (vorgegeben durch die Betriebssystemumgebung) und im Bilderdirectory von HALCON gesucht. Das Bilderdirectory von HALCON ist in einer UNIX-Umgebung auf **'.'** und **'/usr/local/halcon/images'** voreingestellt und kann mit dem Befehl `set_system` gesetzt werden. Es kann mehr als ein Bilderdirectory angegeben werden. Dies geschieht, indem man die einzelnen Directories durch einen Doppelpunkt trennt.

Weiterhin kann der Suchpfad mit der Environment-Variablen HALCONIMAGES gesetzt werden (gleicher Aufbau wie bei **'image_dir'**). Beispiel:

```
setenv HALCONIMAGES "/usr/images:/usr/local/halcon/images"\\*
```

Ebenso sucht HALCON die Bilder in dem Unter-Directory **„images“** (Bilder für die Programmbeispiele). Für das HALCON-Directory wird die Environment-Variable HALCONROOT verwendet.

Achtung

Falls Dateien vom Pixeltyp **'real'** eingelesen werden und die Bytereihenfolge falsch (d.h. nicht wie die Daten in der Datei gespeichert sind) gewählt wird, kann es zu Programmfehlern oder sogar Abstürzen aufgrund von Gleitkomma-Ausnahmebehandlungen kommen.

Parameter

- ▷ **`Image`** (output_object) ..... image ⇝ *Hobject*
  Eingelesenes Bild.
- ▷ **`HeaderSize`** (input_control) ..... integer ⇝ *integer*
  Anzahl Bytes für Dateikopf.
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `HeaderSize`
- ▷ **`SourceWidth`** (input_control) ..... extent.x ⇝ *integer*
  Anzahl Bildspalten des Bildes auf Datei.
  Defaultwert : 512
  Typischer Wertebereich : 1 ≤ `SourceWidth`
- ▷ **`SourceHeight`** (input_control) ..... extent.y ⇝ *integer*
  Anzahl Bildzeilen des Bildes auf Datei.
  Defaultwert : 512
  Typischer Wertebereich : 1 ≤ `SourceHeight`
- ▷ **`StartRow`** (input_control) ..... point.y ⇝ *integer*
  Startpunkt Bildausschnitt (Zeile).
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `StartRow`
  Restriktion : `StartRow` < `SourceHeight`
- ▷ **`StartColumn`** (input_control) ..... point.x ⇝ *integer*
  Startpunkt Bildausschnitt (Spalte).
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `StartColumn`
  Restriktion : `StartColumn` < `SourceWidth`
- ▷ **`DestWidth`** (input_control) ..... extent.x ⇝ *integer*
  Anzahl Bildspalten des erzeugten Bildes.
  Defaultwert : 512
  Typischer Wertebereich : 1 ≤ `DestWidth`
  Restriktion : `DestWidth` ≤ `SourceWidth`
- ▷ **`DestHeight`** (input_control) ..... extent.y ⇝ *integer*
  Anzahl Bildzeilen des erzeugten Bildes.
  Defaultwert : 512
  Typischer Wertebereich : 1 ≤ `DestHeight`
  Restriktion : `DestHeight` ≤ `SourceHeight`

▷ **PixelType** (input_control) . . . . . string ⇝ *string*
Art der Pixelwerte.
Defaultwert : 'byte'
Werteliste : PixelType ∈ {'bit', 'byte', 'r_byte', 'g_byte', 'b_byte', 'short', 'signed_short', 'long', 'swapped_long'}

▷ **BitOrder** (input_control) . . . . . string ⇝ *string*
Reihenfolge der Bits innerhalb eines Bytes.
Defaultwert : 'MSBFirst'
Werteliste : BitOrder ∈ {'MSBFirst', 'LSBFirst'}

▷ **ByteOrder** (input_control) . . . . . string ⇝ *string*
Reihenfolge der Bytes innerhalb einer 'short'-Einheit.
Defaultwert : 'MSBFirst'
Werteliste : ByteOrder ∈ {'MSBFirst', 'LSBFirst'}

▷ **Pad** (input_control) . . . . . string ⇝ *string*
Dateneinheiten innerhalb einer Bildzeile (Alignement).
Defaultwert : 'byte'
Werteliste : Pad ∈ {'byte', 'short', 'long'}

▷ **Index** (input_control) . . . . . integer ⇝ *integer*
Nummer des Bildes innerhalb der Datei.
Defaultwert : 1
Typischer Wertebereich : 1 ≤ Index (lin)

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Name der einzulesenden Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_sequence den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, wird 5 (H_MSG_FAIL) zurückgeliefert. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_sequence wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive") ausgeführt.

Mögliche Nachfolgerfunktionen

disp_image, count_channels, decompose3, write_image, rgb1_to_gray

Alternativen

read_image

Siehe auch

read_image

Modul

Image / region / XLD management

write_image (Image : : Format, FillColor, FileName :)

Schreiben von Bildern in Graphikformaten.

write_image gibt das angegebene Bild (Image) in verschiedenen Bildformaten in Dateien aus. Pixel außerhalb der Region erhalten dabei die durch FillColor definierte Farbe. Für Grauwertbilder muß ein Wert zwischen 0 (Schwarz) und 255 (Weiß) angegeben werden, bei Farbbildern können direkt die RGB-Werte angegeben werden, z.B. hexadezimal: 0xffff00 fuer einen gelben Hintergrund (Rot=255, Grün=255, Blau=0).

Derzeit werden folgende Formate unterstützt:

'tiff' TIFF-Format, bei 3-kanaligen Bildern (Farbbilder) 3 Samples, ansonsten 1 Sample pro Pixel, 8 Bit pro Sample, unkomprimiert,72 dpi; Dateiendung: .tiff bzw. .tif (unter Windows)

'bmp' Windows-BMP-Format, bei 3-kanaligen Bildern (Farbbilder) 3 Byte, ansonsten 1 Byte pro Pixel; Dateiendung: *.bmp

'jpeg' JPEG-Format (verlustbehaftete Komprimierung); Zusammen mit dem Format kann ein Qualitätsmaß angeben werden, das die Komprimierungsrate und die Qualität des gespeicherten Bildes bestimmt. Große Werte (Maximum 100) erzeugen eine relativ große Datei mit hoher Qualität, bei kleine Werten nimmt die Qualität und die Dateigröße deutlich ab; Bsp.: 'jpeg 30'. Achtung: Bilder, die zu einem späteren Zeitpunkt noch ausgewertet werden müssen, sollten aufgrund der Verluste nicht im JPEG-Format abgespeichert werden!

'png' PNG-Format (verlustfreie Komprimierung); zusammen mit dem Format kann ein Komprimierungsgrad zwischen 0 und 9 angegeben werden, wobei 0 keiner Kompression und 9 der bestmöglichen Kompression entspricht. Alternativ kann die Kompression über folgende Konstanten ausgewählt werden: 'best', 'fastest' und 'none'. Korrekte Formatparameter sind also z.B. 'png', 'png 7' und 'png none'. In PNG-Dateien können Bilder vom Type byte abgespeichert werden. Falls ein Bild mit eingeschränktem Definitionsbereich übergeben wird, wird die Region als Alpha-Kanal abgespeichert, wobei die Punkte innerhalb des Definitionsbereichs mit dem maximalen Grauwert des Eingababildtyps abgespeichert werden und die Punkte außerhalb des Definitionsbereichs mit dem Grauwert 0. Falls ein Bild mit vollem Definitionsbereich übergeben wird, wird kein Alpha-Kanal abgespeichert.

'ima' Die Daten werden binär und zeilensequentiell (ohne Header oder Zeilentrenner) geschrieben. Die Größe des Bildes und der Pixeltyp wird in der Beschreibungsdatei "`FileName`.exp" abgelegt. Es können **byte**-, **int4**- und **real**-Bilder geschrieben werden. Die Dateiendung ist: *.ima

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject*
  Bild(er) das (die) ausgegeben wird (werden).
- ▷ **`Format`** (input_control) ..... string ⇝ *string*
  Graphikformat.
  Defaultwert : 'tiff'
  Werteliste : `Format` ∈ {'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 ⇝ *integer*
  Füllgrauwert für Bildpunkte, die nicht zur Region der Bilder gehören.
  Defaultwert : 0
  Wertevorschläge : `FillColor` ∈ {-1, 0, 255, '0xff0000', '0xff00'}
- ▷ **`FileName`** (input_control) ..... filename(-array) ⇝ *string*
  Name der Graphikdatei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_image` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Kann die Datei nicht geöffnet werden, liefert `write_image` 5 (H_MSG_FAIL).

Parallelisierungsinformation

`write_image` wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `read_image`

Modul

Image / region / XLD management

2.2 Region

`read_region` (: `Region` : `FileName` :)

Einlesen von Binärbildern und HALCON-Regionen.

`read_region` liest Regionen aus einer Binärdatei. Die Daten sind in gepackter Form abgespeichert. Aus den eingelesenen Daten wird je nach Format eine oder mehrere Regionen erzeugt. Folgende Binärformate werden unterstützt:

Tiff: Binäre Tiff-Bilder mit der Extension **'tiff'** oder **'tif'**. Man erhält immer $eine$ Region. Die Farbe Schwarz wird als Vordergrund interpretiert.

BMP: Windows Bitmap-Format mit der Extension **'bmp'**. Man erhält immer *eine* Region. Die Farbe Schwarz wird als Vordergrund interpretiert.

HALCON Regionen: Dateiformat von HALCON zum Speichern von Regionen. Hiermit können auch mehrere Regionen mit `write_region` und `read_region` gleichzeitig (in einer Datei) gespeichert bzw. gelesen werden. Alle Regionendateien haben die Extension '.reg'. Diese wird beim Lesen und Schreiben der Datei nicht angegeben.

Analog zu `read_image` kann ein Suchpfad (**'image_dir'**) definiert werden.

Parameter

▷ **`Region`** (output_object) ..... region(-array) ⇝ *Hobject*

▷ **`FileName`** (input_control) ..... filename ⇝ *string*

Beispiel

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

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `read_region` den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, wird 5 (H_MSG_FAIL) zurückgeliefert. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_region` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_image`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `disp_region`

Siehe auch

`write_region`, `read_image`

Modul

Image / region / XLD management

`write_region ( Region : : FileName : )`

Schreiben von Regionen auf Datei.

`write_region` gibt die Regionen der Eingabebilder (in Lauflängenkodierung) auf eine Binärdatei aus. Die Daten werden in gepackter Form abgespeichert. Die ausgegebenen Daten können mit `read_region` wieder eingelesen werden. Der Dateiname lautet <`FileName`>.reg. Die Extension wird nicht mit angegeben.

Parameter

▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
Bilder, deren Regionen ausgegeben werden.

▷ **`FileName`** (input_control) ..... filename ⇝ *string*
Name der Regionendatei ohne Extension.
Defaultwert : '/tmp/region'

Beispiel

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

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `write_region` den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, wird 5 (H_MSG_FAIL) geliefert. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`write_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_window`, `read_image`, `read_region`, `threshold`, `regiongrowing`

Siehe auch

`read_region`

Modul

Image / region / XLD management

2.3 Sonstiges

`file_exists` (: : `FileName` :)

Test ob Datei existiert.

`file_exists` überprüft, ob die angegebene Datei bereits existiert. Ist dies der Fall, liefert `file_exists` TRUE, anderenfalls FALSE.

Parameter

- ▷ **`FileName`** (input_control) . . . . . filename ⇝ *string*
  Name der zu prüfenden Datei.
  Defaultwert : ’/bin/cc’

Ergebnis

`file_exists` liefert den Wert 2 (H_MSG_TRUE) (Datei existiert) oder 3 (H_MSG_FALSE) (Datei existiert nicht).

Parallelisierungsinformation

`file_exists` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`open_file`

Alternativen

`open_file`

Modul

Basic operators

`read_world_file` (: : `FileName` : `WorldTransformation`)

Lesen der Geokodierung aus einem ARC/INFO World File.

`read_world_file` liest die Geokodierung aus dem ARC/INFO World File mit Namen `FileName` aus, und liefert sie als homogene 2D-Transformationsmatrix in `WorldTransformation` zurück. Dabei werden alle Verzeichnisse, die in der HALCON-Systemvariablen **’image_dir’** bzw. in der Umgebungsvariablen HALCONIMAGES enthalten sind, nach der Datei `FileName` durchsucht (siehe `read_image`). Die Transformationsmatrix kann verwendet werden, um XLD-Konturen vor dem Schreiben mit `write_contour_xld_arc_info` in Weltkoordinaten zu transformieren. Wenn die Matrix `WorldTransformation` mit `hom_mat2d_invert` invertiert wird, kann die so entstehende Matrix verwendet werden, um Konturen, die mit `read_contour_xld_arc_info` eingelesen worden sind, in das Bildkoordinatensystem zu transformieren.

Parameter

- ▷ **`FileName`** (input_control) . . . . . filename ⇝ *string*
  Name des ARC/INFO World Files.
- ▷ **`WorldTransformation`** (output_control) . . . . . affine2d-array ⇝ *real*
  Transformations-Matrix von Bild- in Weltkoordinaten.
  Parameteranzahl : 6

Ergebnis

Wenn die Parameter korrekt sind und das angegebene World File gelesen werden konnte, liefert `read_world_file` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_world_file` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`hom_mat2d_invert`, `affine_trans_contour_xld`, `affine_trans_polygon_xld`

Siehe auch

`write_contour_xld_arc_info`, `read_contour_xld_arc_info`, `write_polygon_xld_arc_info`, `read_polygon_xld_arc_info`

Modul

Sub-pixel operators

2.4 Text

`close_all_files` (: : :)

Schließt alle offenen Dateien.

`close_all_files` schließt alle offenen Dateien.

Achtung

Da alle Dateien geschlossen werden, sind alle vorhandenen Handle ungültig.

Ergebnis

Gelingt es, die Dateien zu schließen, liefert `close_all_files` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_all_files` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Alternativen

`close_file`

Modul

Basic operators

`close_file` (: : FileHandle :)

Schließen einer Textdatei.

`close_file` schließt eine Datei, die mit `open_file` geöffnet wurde.

Parameter

▷ **`FileHandle`** (input_control) . . . . . file ⇝ *integer*
Datei Handle.

Beispiel

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

Ergebnis

Ist das Datei Handle korrekt, dann liefert `close_file` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_file` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_file`

Siehe auch

`open_file`

Modul

Basic operators

`fnew_line ( : : FileHandle : )`

Erzeugen eines Zeilenvorschubs.

`fnew_line` gibt einen Zeilenvorschub in der Ausgabedatei aus. Gleichzeitig wird der Ausgabepuffer entleert.

Parameter

▷ **`FileHandle`** (input_control) . . . . . file ⇝ *integer*
Datei Handle.

Beispiel

```
fwrite_string(FileHandle,'Good Morning')
fnew_line(FileHandle)
```

Ergebnis

`fnew_line` liefert den Wert 2 (H_MSG_TRUE), falls eine Ausgabedatei geöffnet ist und geschrieben werden kann. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fnew_line` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`fwrite_string`

Siehe auch

`fwrite_string`

Modul

Basic operators

`fread_char ( : : FileHandle : Char )`

Einlesen eines Zeichens aus einer Textdatei.

`fread_char` liest aus der Eingabedatei ein Zeichen ein. Wird über das Ende der Datei hinausgelesen, dann liefert `fread_char` die Zeichenreihe **'eof'**. Am Ende einer Zeile wird **'nl'** zurückgegeben.

Parameter

▷ **`FileHandle`** (input_control) . . . . . file ⇝ *integer*
Datei Handle.

▷ **`Char`** (output_control) . . . . . string ⇝ *string*
Eingelesenes Zeichen oder Steuerstring ('nl','eof').

Beispiel

```
repeat >
  fread_char(FileHandle:Char)
  (if(Char = 'nl') > fnew_line(FileHandle)) |
  (if(Char != 'nl') > fwrite_string(FileHandle,Char))
until(Char = 'eof').
```

Ergebnis

fread_char liefert den Wert 2 (H_MSG_TRUE), falls eine Eingabedatei geöffnet ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fread_char` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_file`

Mögliche Nachfolgerfunktionen

`close_file`

Alternativen

`fread_string`, `read_string`

Siehe auch

`open_file`, `close_file`, `fread_string`

Modul

Basic operators

fread_string (: : FileHandle : OutString, IsEOF)

Einlesen von Strings aus einer Textdatei.

`fread_string` liest aus der aktuellen Eingabedatei einen String ein. Ein String beginnt mit dem ersten darstellbaren Zeichen: Buchstaben, Zahlen, Sonderzeichen (ohne Leerzeichen). Ein String endet, falls ein Leerzeichen oder ein Zeilensprung erreicht wird. Mehrere aufeinanderfolgende Zeilensprünge werden ignoriert. Falls das Ende der Datei erreicht ist liefert `IsEOF` den Wert **1**, ansonsten **0**.

Parameter

- ▷ **FileHandle** (input_control) . . . . . file ⇝ *integer*
  Datei Handle.
- ▷ **OutString** (output_control) . . . . . string ⇝ *string*
  Eingelesene Zeichenreihe.
- ▷ **IsEOF** (output_control) . . . . . integer ⇝ *integer*
  Ende der Datei erreicht.

Beispiel

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

Ergebnis

`fread_string` liefert den Wert 2 (H_MSG_TRUE), falls eine Datei geöffnet ist und ein geeigneter String eingelesen wird. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fread_string` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_file`

Mögliche Nachfolgerfunktionen

`close_file`

Alternativen

`fread_char`, `read_string`

Siehe auch

`open_file`, `close_file`, `fread_char`

Modul

Basic operators

fwrite_string (: : FileHandle, String :)

Ausgabe von Werten auf eine Textdatei.

fwrite_string gibt auf die Ausgabedatei einen String oder Zahlen aus. Eine Datei wird mit open_file geöffnet. Mit dem Aufruf set_system(::'flush_file', <Wahrheitswert>:) wird festgelegt, ob die ausgegebenen Zeichen direkt auf das Ausgabemedium ausgegeben werden. Ist **'flush_file'** auf **'false'** gesetzt, dann erscheinen die Zeichen (insbesondere bei Bildschirmausgabe) erst nach dem Aufruf von fnew_line.

Es können sowohl Zeichenreihen, als auch ganze Zahlen und Gleitpunktzahlen übergeben werden. Wird mehr als ein Wert übergeben, so werden diese ohne Leerzeichen hintereinander ausgegeben.

Parameter

- **FileHandle** (input_control) . . . . . file ⇝ *integer*
  Datei Handle.
- **String** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
  Werte, die auf die Textdatei ausgegeben werden sollen.
  Defaultwert : 'hallo'

Beispiel

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

Ergebnis

fwrite_string liefert den Wert 2 (H_MSG_TRUE), falls der Schreibvorgang erfolgreich abgeschlossen wird. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fwrite_string ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_file

Mögliche Nachfolgerfunktionen

close_file

Alternativen

write_string

Siehe auch

open_file, close_file, set_system

Modul

Basic operators

open_file (: : FileName, FileType : FileHandle)

Öffnen einer Textdatei.

open_file öffnet eine Datei, wobei durch FileType bestimmt wird, ob es sich um eine Ein- (**'input'**) oder Ausgabedatei (**'output'** oder **'append'**) handelt. Es werden hierbei Textdateien erzeugt, auf die entweder nur lesend (**'input'**) oder nur schreibend (**'output'** oder **'append'**) zugegriffen werden kann. Für Aus- und Eingaben auf dem Terminal sind die „Dateinamen" **'standard'** (**'input'** und **'output'**) und **'error'** (nur **'output'**) reserviert.

Parameter

- **FileName** (input_control) . . . . . filename ⇝ *string*
  Name der zu öffnenden Datei.
  Defaultwert : 'standard'
  Wertevorschläge : FileName ∈ {'standard', 'error', "/tmp/dat.dat"}

▷ **FileType** (input_control) ..... string ⇝ *string*
Art der Datei.
Defaultwert : 'output'
Werteliste : FileType ∈ {'input', 'output', 'append'}

▷ **FileHandle** (output_control) ..... file ⇝ *integer*
Datei Handle.

Beispiel

```
/* Creating of an outputfile 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).
```

Ergebnis

Sind die Parameter korrekt, dann liefert open_file den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

open_file ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Nachfolgerfunktionen

fwrite_string, fread_char, fread_string, close_file

Siehe auch

close_file, fwrite_string, fread_char, fread_string

Modul

Basic operators

2.5 Tupel

read_tuple (: : FileName : Tuple)

Ein Tupel von Datei lesen.

read_tuple liest den Inhalt der Datei FileName und wandelt diesen in das tupel Tuple um. Die Datei muß mit write_tuple erzeugt worden sein.

Parameter

▷ **FileName** (input_control) ..... filename ⇝ *string*
Name der zu lesenden Datei.

▷ **Tuple** (output_control) ..... number(-array) ⇝ *real* / integer / string
Tupel mit beliebigen Werten.

Ergebnis

Sind die Parameter korrekt, dann liefert read_tuple den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, dann liefert read_tuple den Wert 5 (H_MSG_FAIL). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_tuple ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Alternativen

fwrite_string

Siehe auch

write_tuple, gnuplot_plot_ctrl, write_image, write_region, open_file

Modul

Operators not requiring licensing

write_tuple (: : Tuple, FileName :)

Inhalt eines Tupels auf Datei schreiben.

write_tuple schreibt den Inhalt von Tuple auf Datei. Es handelt sich hierbei um ein ASCII Format, d.h. die Daten sind zwischen unterschiedlichen Rehnertypen austauschbar. Die Daten können mit dem Operator read_tuple wieder eingelesen werden. Für den Dateiname ist keine Extension festgelegt.

Parameter

- ▷ **Tuple** (input_control) ..... number(-array) ⇝ *real* / integer / string
  Tupel mit beliebigen Werten.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der zu schreibenden Datei.

Ergebnis

Sind die Parameter korrekt, dann liefert write_tuple den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, dann liefert write_tuple den Wert 5 (H_MSG_FAIL). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_tuple ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

fwrite_string

Siehe auch

read_tuple, write_image, write_region, open_file

Modul

Operators not requiring licensing

2.6 XLD

read_contour_xld_arc_info (: Contours : FileName :)

Lesen von XLD-Konturen im ARC/INFO-Generate-Format.

read_contour_xld_arc_info liest die in der Datei FileName im ARC/INFO-Generate-Format gespeicherten Linien ein und liefert sie als XLD-Konturen in Contours zurück. Dabei werden alle Verzeichnisse, die in der HALCON-Systemvariablen **'image_dir'** bzw. in der Umgebungsvariablen HALCONIMAGES enthalten sind, nach der Datei FileName durchsucht (siehe read_image). Die eingelesenen Konturen liegen im Weltkoordinatensystem vor. Der Operator affine_trans_contour_xld kann verwendet werden, um sie in das Bildkoordinatensystem zu transformieren. Die dazu notwendige Transformationsmatrix kann erzeugt werden, indem mit read_world_file aus einem ARC/INFO World File die Transformationsmatrix von Bild- nach Weltkoordinatensystem gelesen wird, und diese mit hom_mat2d_invert invertiert wird.

Parameter

- ▷ **Contours** (output_object) ..... xld_cont(-array) ⇝ *Hobject*
  Eingelesene XLD-Konturen.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der ARC/INFO-Datei.

Beispiel

```
/* 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)
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei gelesen werden konnte, liefert `read_contour_xld_arc_info` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_contour_xld_arc_info` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`hom_mat2d_invert`, `affine_trans_contour_xld`

Siehe auch

`read_world_file`, `write_contour_xld_arc_info`, `read_polygon_xld_arc_info`

Modul

Sub-pixel operators

`read_polygon_xld_arc_info` (: Polygons : FileName :)

Lesen von XLD-Polygonen im ARC/INFO-Generate-Format.

`read_polygon_xld_arc_info` liest die in der Datei `FileName` im ARC/INFO-Generate-Format gespeicherten Linien ein und liefert sie als XLD-Polygone in `Polygons` zurück. Dabei werden alle Verzeichnisse, die in der HALCON-Systemvariablen **’image_dir’** bzw. in der Umgebungsvariablen HALCONIMAGES enthalten sind, nach der Datei `FileName` durchsucht (siehe `read_image`). Die eingelesenen Polygone liegen im Weltkoordinatensystem vor. Der Operator `affine_trans_polygon_xld` kann verwendet werden, um sie in das Bildkoordinatensystem zu transformieren. Die dazu notwendige Transformationsmatrix kann erzeugt werden, indem mit `read_world_file` aus einem ARC/INFO World File die Transformationsmatrix von Bild- nach Weltkoordinatensystem gelesen wird, und diese mit `hom_mat2d_invert` invertiert wird.

Parameter

- ▷ **`Polygons`** (output_object) ..... xld_poly(-array) ⇝ *Hobject*
  Eingelesene XLD-Polygone.
- ▷ **`FileName`** (input_control) ..... filename ⇝ *string*
  Name der ARC/INFO-Datei.

Beispiel

```
/* 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)
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei gelesen werden konnte, liefert `read_polygon_xld_arc_info` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_polygon_xld_arc_info` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`hom_mat2d_invert`, `affine_trans_polygon_xld`

Siehe auch

`read_world_file`, `write_polygon_xld_arc_info`, `read_contour_xld_arc_info`

Modul

Sub-pixel operators

write_contour_xld_arc_info (Contours : : FileName :)

Schreiben von XLD-Konturen im ARC/INFO-Generate-Format.

write_contour_xld_arc_info schreibt die XLD-Konturen Contours im ARC/INFO-Generate-Format in die Datei FileName. Wenn kein absoluter Pfad in FileName angegeben wird, wird die Ausgabedatei im aktuellen Verzeichnis des HALCON-Prozesses geschrieben. Die Konturen müssen zuvor mit affine_trans_contour_xld in Weltkoordinaten transformiert worden sein. Die dazu notwendige Transformation kann mit read_world_file aus einem ARC/INFO World File eingelesen werden.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
  Zu schreibende XLD-Konturen.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der ARC/INFO-Datei.

Beispiel

```
/* 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')
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei geschrieben werden konnte, liefert write_contour_xld_arc_info den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_contour_xld_arc_info ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

affine_trans_contour_xld

Siehe auch

read_world_file, read_contour_xld_arc_info, write_polygon_xld_arc_info

Modul

Sub-pixel operators

write_polygon_xld_arc_info (Polygons : : FileName :)

Schreiben von XLD-Polygonen im ARC/INFO-Generate-Format.

write_polygon_xld_arc_info schreibt die XLD-Polygone Polygons im ARC/INFO-Generate-Format in die Datei FileName. Wenn kein absoluter Pfad in FileName angegeben wird, wird die Ausgabedatei im aktuellen Verzeichnis des HALCON-Prozesses geschrieben. Die Polygone müssen zuvor mit affine_trans_polygon_xld in Weltkoordinaten transformiert worden sein. Die dazu notwendige Transformation kann mit read_world_file aus einem ARC/INFO World File eingelesen werden.

Achtung

Die XLD-Konturen, die möglicherweise von Polygons referenziert werden, werden nicht in die ARC/INFO-Datei abgespeichert, da dies mit dem ARC/INFO-Generate-Format nicht möglich ist. Wenn die Polygone mit read_polygon_xld_arc_info wieder eingelesen werden, ist diese Information verlorengegangen, und deshalb können auch keine Referenzen auf Konturen für die Polygone generiert werden. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. split_contours_xld, zu Fehlermeldungen kommen.

Parameter

- ▷ **Polygons** (input_object) ..... xld_poly(-array) ⇝ *Hobject*
  Zu schreibende XLD-Polygone.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der ARC/INFO-Datei.

Beispiel

```
/* 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')
```

Ergebnis

Wenn die Parameter korrekt sind und die angegebene Datei geschrieben werden konnte, liefert `write_polygon_xld_arc_info` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`write_polygon_xld_arc_info` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`affine_trans_polygon_xld`

Siehe auch

`read_world_file`, `read_polygon_xld_arc_info`, `write_contour_xld_arc_info`

Modul

Sub-pixel operators

Kapitel 3

Filter

3.1 Affine-Abbildungen

```
affine_trans_image ( Image : ImageAffinTrans : HomMat2D,
Interpolation, AdaptImageSize : )
```

Ausführung einer affinen Abbildung.

affine_trans_image dient zur Vergrößerung, Verkleinerung, Drehung oder Verschiebung eines Bildes. Dazu verwendet die Prozedur eine Transformationsmatrix, die in HomMat2D übergeben wird. Diese kann mit den Routinen hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate und hom_mat2d_translate aufgebaut werden. Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Die Region des Eingabebildes wird ignoriert, also als das volle Rechteck der Eingabebildmatrix angenommen. Die Region des Ausgabebildes besteht aus dem transformierten Rechteck des Eingabebildes (gegebenenfalls wird das Ausgabebild mit Nullen, d.h. „schwarz“, aufgefüllt).

Im allgemeinen fallen transformierte Punkte zwischen die Rasterpunkte der Bildmatrix. Es ist also eine geeignete Interpolation zwischen den Grauwerten des Eingabebildes durchzuführen. Sie dient insbesondere auch dazu, die „Klötzchenbildung“ bei Vergrößerungen bzw. Aliasing-Effekte bei Verkleinerungen zu vermeiden. Die Qualität (bzw. Geschwindigkeit) der Interpolation wird über den Parameter Interpolation gesteuert:

none Keine Interpolation: Grauwerte werden vervielfacht bzw. der Grauwert des nächstgelegenen Bildpunktes verwendet (eventuell niedrigere Qualität, sehr schnell).

constant Es erfolgt eine ungewichtete Interpolation zwischen den Grauwerten (mittlere Qualität und Laufzeit).

weighted Es erfolgt eine gewichtete Interpolation zwischen den Grauwerten (höchste Qualität, langsam).

Zusätzlich beeinflußt der Systemparameter **'int_zooming'** (siehe set_system) die Genauigkeit der Transformation. Falls **'int_zooming'** auf **'true'** gesetzt wird, wird die Transformation intern bei Byte- und Int2-Bildern mit Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die Genauigkeit der berechneten Grauwerte geringer. Für Byte-Bilder sind die Differenzen zur genaueren Berechnung (mit **'int_zooming'** = **'false'**) normalerweise kleiner als zwei Grauwerte. Für Int2-Bilder gilt entsprechend, daß die Grauwertdifferenzen kleiner als 1/128 mal der dynamische Grauwertbereich des Bildes sind. Sie können also bis zu 512 Grauwerte betragen, wenn der volle Grauwertbereich von 16 Bit ausgenutzt wird. Für Real-Bilder hat der Parameter **'int_zooming'** keinen Einfluß, da intern immer mit Gleitkommazahlen gerechnet wird.

Die Größe des Zielbildes wird durch den Parameter AdaptImageSize gesteuert: Bei **'true'** wird die Größe des Zielbildes so gewählt, daß rechts und unten kein Clipping auftritt. Bei **'false'** hat das Zielbild die gleiche Größe wie das Eingabebild. Beachten Sie, dass unabhängig von AdaptImageSize das Zielbild grundsätzlich am linken und oberen Rand beschnitten wird, d.h. alle Bildbereiche, die nach der Transformation negative Koordinaten aufweisen, werden abgeschnitten.

Achtung

Die Region des Eingabebildes wird nicht beachtet.

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / real
  Eingabebild.
- ▷ **`ImageAffinTrans`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / real
  Transformiertes Ausgabebild.
- ▷ **`HomMat2D`** (input_control) ..... affine2d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : `6`
- ▷ **`Interpolation`** (input_control) ..... string ⇝ *string*
  Art der Interpolation.
  Defaultwert : 'constant'
  Werteliste : `Interpolation` ∈ {'none', 'constant', 'weighted'}
- ▷ **`AdaptImageSize`** (input_control) ..... string ⇝ *string*
  Anpassung der Zielbildgröße.
  Defaultwert : 'false'
  Werteliste : `AdaptImageSize` ∈ {'true', 'false'}

Beispiel

```
/* 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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `affine_trans_image` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`affine_trans_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

`hom_mat2d_identity`, `hom_mat2d_translate`, `hom_mat2d_rotate`, `hom_mat2d_scale`

Alternativen

`zoom_image_size`, `zoom_image_factor`, `mirror_image`, `rotate_image`, `affine_trans_region`

Siehe auch

`set_part_style`

Modul

Image filters

`mirror_image` (`Image` : `ImageMirror` : `Mode` :)

Spiegeln von Bildern.

`mirror_image` spiegelt das Bild `Image` an einer von drei Achsen. Wenn `Mode` 'row' ist, wird das Bild vertikal gespiegelt, wenn er 'column' ist, wird es horizontal gespiegelt, und wenn er 'main' ist, wird das Bild an der Winkelhalbierenden $x = y$ gespiegelt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **`ImageMirror`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Gespiegeltes Ausgabebild.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Art der Spiegelung.
  Defaultwert : 'row'
  Werteliste : `Mode` ∈ {'row', 'column', 'main'}

Beispiel

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

Parallelisierungsinformation

`mirror_image` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Alternativen

`hom_mat2d_rotate`, `affine_trans_image`, `rotate_image`

Siehe auch

`rotate_image`, `hom_mat2d_rotate`

Modul

Image filters

`polar_trans_image` (`ImageXY` : `ImagePolar` : `Row`, `Column`, `Width`, `Height` :)

Polartransformation

`polar_trans_image` wandelt ein Bild mit (x,y)-Koordinaten in ein Bild mit (Winkel,Radius)-Koordinaten um. Die Größe des Zielbildes wird mit `Width` und `Height` angegeben. `Width` gibt dabei die Auflösung des Winkels und `Height` die Auflösung des Radius an. `Row` und `Column` geben die Position im Eingabebild an, in der das Zentrum des Polarkoordinatensystems liegt. Dieser Punkt wird auf die erste (oberste) Zeile im Ergebnisbild abgebildet.

Ein Punkt (x',y') im Ergebnisbild ist der Punkt (x,y) im Eingabebild wie folgt zugeordnet:

$$x = y' \cos(2\pi(x'/resultwidth)) + \texttt{Column} \quad y = y' \sin(2\pi(x'/resultwidth)) + \texttt{Row} \ .$$

Parameter

- ▷ **ImageXY** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild in x/y-Koordinaten.
- ▷ **ImagePolar** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Ausgabebild in Polar-Koordinaten.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Zentrums.
  Defaultwert : 100
  Wertevorschläge : Row ∈ {0, 10, 100, 200}
  Typischer Wertebereich : 0 ≤ Row ≤ 512
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Zentrums.
  Defaultwert : 100
  Wertevorschläge : Column ∈ {0, 10, 100, 200}
  Typischer Wertebereich : 0 ≤ Column ≤ 512
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Zielbildes.
  Defaultwert : 314
  Wertevorschläge : Width ∈ {100, 200, 157, 314, 512}
  Typischer Wertebereich : 2 ≤ Width ≤ 512
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Zielbildes.
  Defaultwert : 200
  Wertevorschläge : Height ∈ {100, 128, 256, 512}
  Typischer Wertebereich : 2 ≤ Height ≤ 512
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

Beispiel

```
read_image(Image,'affe')
disp_image(Image,WindowHandle)
polar_trans_image(Image,PolarImage,100,100,314,200).
disp_image(PolarImage,WindowHandle)
```

Parallelisierungsinformation

polar_trans_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Alternativen

affine_trans_image

Modul

Image filters

rotate_image (Image : ImageRotate : Phi, Interpolation :)

Rotation von Bildern um das Zentrum.

rotate_image dreht das Bild Image um den Bildmittelpunkt in mathematisch positiver Drehrichtung um Phi Grad. Diese Prozedur ist für Vielfache von 90° schneller als die allgemeine Funktion affine_trans_image. Bei Rotationen um 90, 180 und 270 wird die Region passend mitgedreht. Bei allen anderen Winkeln wird die Region maximal gesetzt. Der Parameter Interpolation entspricht der Wirkung bei affine_trans_image. Bei Rotationen um 90, 180 und 270 wird der Parameter ignoriert. Die Größe des Ergebnisbildes ist mit Ausnahme von 90 und 270 Grad identisch mit dem Eingabebild. In den beiden Sonderfällen erhält das Ergebnisbild als Höhe die Breite der Eingabe und als Breite die Höhe der Eingabe.

Achtung

Es ist zu beachten, daß der Winkel Phi in Grad und nicht in Bogenmaß angegeben wird.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageRotate** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Rotiertes Ausgabebild.
- ▷ **Phi** (input_control) ..... angle.deg ⇝ *real* / integer
  Rotationswinkel.
  Defaultwert : 90
  Wertevorschläge : Phi ∈ {90, 180, 270}
  Typischer Wertebereich : 0 ≤ Phi ≤ 360
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.2
- ▷ **Interpolation** (input_control) ..... string ⇝ *string*
  Art der Interpolation.
  Defaultwert : 'constant'
  Werteliste : Interpolation ∈ {'none', 'constant', 'weighted'}

Beispiel

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

Parallelisierungsinformation

rotate_image ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Alternativen

hom_mat2d_rotate, affine_trans_image

Siehe auch

mirror_image

Modul

Image filters

zoom_image_factor (Image : ImageZoomed : ScaleWidth, ScaleHeight, Interpolation :)

Skalierung von Bildern um einen Vergrößerungsfaktor.

zoom_image_factor vergrößert das Bild Image um einen Faktor ScaleWidth in der Breite und einen Faktor ScaleHeight in der Höhe. Der Parameter Interpolation gibt dabei die Art der verwendeten Interpolationsmethode an (siehe affine_trans_image).

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / real
  Eingabebild.

▷ **ImageZoomed** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / real
Skaliertes Ausgabebild.

▷ **ScaleWidth** (input_control) . . . . . extent.x ⇝ *real*
Faktor für Breite des Zielbildes.
Defaultwert : 0.5
Wertevorschläge : ScaleWidth ∈ {0.25, 0.5, 1.5, 2.0}
Typischer Wertebereich : 0.001 ≤ ScaleWidth ≤ 10.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **ScaleHeight** (input_control) . . . . . extent.y ⇝ *real*
Faktor für Höhe des Zielbildes.
Defaultwert : 0.5
Wertevorschläge : ScaleHeight ∈ {0.25, 0.5, 1.5, 2.0}
Typischer Wertebereich : 0.001 ≤ ScaleHeight ≤ 10.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Art der Interpolation.
Defaultwert : 'constant'
Werteliste : Interpolation ∈ {'none', 'constant', 'weighted'}

Beispiel

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

Parallelisierungsinformation

zoom_image_factor ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Alternativen

zoom_image_factor, affine_trans_image, hom_mat2d_scale

Siehe auch

hom_mat2d_scale, affine_trans_image

Modul

Image filters

zoom_image_size (Image : ImageZoom : Width, Height, Interpolation :)

Skalierung von Bildern auf eine vorgegebene Größe.

zoom_image_size vergrößert das Bild Image auf die durch Width und Height vorgegebene Größe. Der Parameter Interpolation gibt dabei die Art der verwendeten Interpolationsmethode an (siehe affine_trans_image).

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / real
Eingabebild

▷ **ImageZoom** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / real
Skaliertes Ausgabebild.

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des Zielbildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 256, 512}
Typischer Wertebereich : 2 ≤ Width ≤ 512
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des Zielbildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {128, 256, 512}
Typischer Wertebereich : 2 ≤ Height ≤ 512
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Art der Interpolation.
Defaultwert : 'constant'
Werteliste : Interpolation ∈ {'none', 'constant', 'weighted'}

Beispiel

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

Parallelisierungsinformation

zoom_image_size ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Alternativen

zoom_image_factor, affine_trans_image, hom_mat2d_scale

Siehe auch

hom_mat2d_scale, affine_trans_image

Modul

Image filters

3.2 Arithmetik

abs_image (Image : ImageAbs : :)

Absolutbetrag eines Bildes.

Die Prozedur abs_image berechnet für Bilder eines beliebigen Typs den Absolutbetrag der Grauwerte und legt das Ergebnis in ImageAbs ab. Bei complexen Bildern wird das Powerspektrum als 'real'-Bild berechnet. abs_image erzeugt bei Bildern ohne Vorzeichen eine logische Kopie.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : any
Eingabebild(er).

▷ **ImageAbs** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : any
Ergebnisbild(er).

Beispiel

```
laws_int2(Image,&Texture,0,5);
abs_image(Texture,Abs);
```

Ergebnis

abs_image liefert den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen.

Parallelisierungsinformation

abs_image ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Siehe auch

convert_image_type, power_byte

Modul

Image filters

add_image (Image1, Image2 : ImageResult : Mult, Add :)

Addition von zwei Bildern.

add_image addiert zwei Bilder. Die Grauwerte ($g1, g2$) der Eingabebilder (Image1 und Image2) werden dabei wie folgt transformiert:

$$g' := (g1 + g2) * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten. Dies gilt im Fall von von int2-Bilder falls Mult den Wert 1 und Add den Wert 0 hat. Hier wird aus Laufzeitgründen auf den Test auf Überlauf verzichtet. Das Ergebnisbild wird in ImageResult abgelegt.
Es ist möglich byte-Bilder mit int2 oder int4 Bildern zu addieren. In diesem Fall werden int2 oder int4 als Ergebnis erzeugt.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.
Es ist zu beachten, daß die Laufzeit des Operators von der Wahl der Steuerparameter abhängt. Für häufig verwendete Parameterkombinationen sind spezielle Optimierungen implementiert.

Parameter

- ▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
  Bild(er) 1.
- ▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
  Bild(er) 2.
- ▷ **ImageResult** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
  Ergebnisbil(der) durch die Bildpunktaddition.
- ▷ **Mult** (input_control) ..... number ⇝ *real* / integer
  Faktor zur Grauwertanpassung.
  Defaultwert : 0.5
  Wertevorschläge : Mult ∈ {0.2, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 5.0}
  Typischer Wertebereich : -255.0 ≤ Mult ≤ 255.0
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.1
- ▷ **Add** (input_control) ..... number ⇝ *real* / integer
  Wert für Grauwertausgleich.
  Defaultwert : 0
  Wertevorschläge : Add ∈ {0, 64, 128, 255, 512}
  Typischer Wertebereich : -512.0 ≤ Add ≤ 512.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0

Beispiel

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

Ergebnis

add_image liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe

(keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`add_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

`sub_image`, `mult_image`

Siehe auch

`sub_image`, `mult_image`

Modul

Image filters

`div_image` (Image1, Image2 : ImageResult : Mult, Add :)

Division von zwei Bildern.

`div_image` dividiert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (`Image1`) werden dabei wie folgt transformiert:

$$g' := g1/g2 * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

- **`Image1`** (input_object) . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
  Bild(er) 1.
- **`Image2`** (input_object) . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
  Bild(er) 2.
- **`ImageResult`** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
  Ergebnisbild(er) durch die Division.
- **`Mult`** (input_control) . . . . . number ⇝ *real* / integer
  Faktor für Graubereichsanpassung.
  Defaultwert : 255
  Wertevorschläge : `Mult` ∈ {0.1, 0.2, 0.5, 1.0, 2.0, 3.0, 10, 100, 500, 1000}
  Typischer Wertebereich : -1000 ≤ `Mult` ≤ 1000
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 1
- **`Add`** (input_control) . . . . . number ⇝ *real* / integer
  Wert für Graubereichsanpassung.
  Defaultwert : 0
  Wertevorschläge : `Add` ∈ {0.0, 128.0, 256.0, 1025}
  Typischer Wertebereich : -1000 ≤ `Add` ≤ 1000
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0

Beispiel

```
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)
```

Ergebnis

`div_image` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`div_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

`add_image`, `sub_image`, `mult_image`

Siehe auch

`add_image`, `sub_image`, `mult_image`

Modul

Image filters

`invert_image ( Image : ImageInvert : : )`

Invertieren eines Bildes.

`invert_image` invertiert die Grauwerte eines Bildes. Bei Bildern vom Typ ’byte’ und ’cyclic’ berechnet sich das Ergebnis als:

$$g' = 255 - g$$

Bilder vom Typ ’direction’ werden durch

$$g' = (g + 90) \text{ modulo } 180$$

transformiert. Bei Typen mit Vorzeichen werden die Werte negiert. Das Ergebnisbild hat den gleichen Pixeltyp wie das Eingabebild.
Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / cyclic / direction
  Eingabebild(er).
- ▷ **`ImageInvert`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / cyclic / direction
  Bild(er) mit invertierten Grauwerten.

Beispiel

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

Parallelisierungsinformation

`invert_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`watersheds`

Alternativen

`scale_image`

Siehe auch

`scale_image`, `add_image`, `sub_image`

Modul

Image filters

max_image (Image1, Image2 : : ImageMax : :)

Pixelweises Maximum von zwei Bildern.

max_image ermittelt (pixelweise) das Maximum der Bilder Image1 und Image2. Das Ergebnis wird in dem Bild ImageMax abgelegt. Das Ergebnisbild hat den gleichen Pixeltyp wie das Eingabebild. Sollen mehrere Bilder(paare) bei einem Aufruf bearbeitet werden, so wird jeweils das i-te Bild aus Image1 mit dem i-ten Bild aus Image2 verglichen. Die Anzahl der Bilder in beiden Eingabeparametern muß also gleich sein. Zu jedem Eingabepaar wird ein Ausgabebild erzeugt.

Achtung

Die beiden Eingabebilder müssen den gleichen Typ und die gleiche Größe haben.

Parameter

- ▷ **Image1** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic

  Bild(er) 1.
- ▷ **Image2** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic

  Bild(er) 2.
- ▷ **ImageMax** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic

  Ergebnisbild(er) durch die Maximierung.

Beispiel

```
read_image(Bild1,"affe")
read_image(Bild2,"fabrik")
max_image(Bild1,Bild2,Max)
disp_image(Max,WindowHandle)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert max_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabbilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

max_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

max_image

Siehe auch

min_image

Modul

Image filters

min_image (Image1, Image2 : : ImageMin : :)

Pixelweises Minimum von zwei Bildern.

min_image ermittelt (pixelweise) das Minimum der Bilder Image1 und Image2. Das Ergebnis wird in dem Bild ImageMin abgelegt. Das Ergebnisbild hat den gleichen Pixeltyp wie das Eingabebild. Sollen mehrere Bilder(paare) bei einem Aufruf bearbeitet werden, so wird jeweils das i-te Bild aus Image1 mit dem i-ten Bild aus Image2 verglichen. Die Anzahl der Bilder in beiden Eingabeparametern muß also gleich sein. Zu jedem Eingabepaar wird ein Ausgabebild erzeugt.

Parameter

▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic

Bild(er) 1.

▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic

Bild(er) 2.

▷ **ImageMin** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic

Ergebnisbild(er) durch das Minimierung.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `min_image` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`min_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

`gray_erosion`

Siehe auch

`max_image`, `min_image`

Modul

Image filters

`mult_image ( Image1, Image2 : ImageResult : Mult, Add : )`

Multiplikation von zwei Bildern.

`mult_image` multipliziert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (`Image1`) werden dabei wie folgt transformiert:

$$g' := g1 * g2 * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex

Bild(er) 1.

▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex

Bild(er) 2.

▷ **ImageResult** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex

Ergebnisbild(er) durch das Produkt.

▷ **Mult** (input_control) ..... number ⇝ *real* / integer

Faktor für Graubereichsanpassung.
Defaultwert : 0.005
Wertevorschläge : `Mult` ∈ {0.001, 0.01, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0}
Typischer Wertebereich : -255.0 ≤ `Mult` ≤ 255.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **Add** (input_control) . . . . . number ⇝ *real* / integer
Wert für Graubereichsanpassung.
Defaultwert : 0
Wertevorschläge : Add ∈ {0.0, 128.0, 256.0}
Typischer Wertebereich : -512.0 ≤ Add ≤ 512.0
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 1.0

Beispiel

```
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)
```

Ergebnis

mult_image liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

mult_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

add_image, sub_image, div_image

Siehe auch

add_image, sub_image, div_image

Modul

Image filters

scale_image (Image : ImageScaled : Mult, Add :)

Skalierung von Grauwerten.

scale_image skaliert die Eingabebilder (Image) mittels folgender Transformation:

$$g' := g * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten.

Dieser Operator kann z.B. dafür verwendet werden, die Grauwerte eines Bildes, also das Intervall [GMin,GMax], auf das volle Intervall [0:255] abbzubilden, indem man die Parameter wie folgt wählt:

$$\texttt{Mult} = \frac{255}{\text{GMax} - \text{GMin}} \qquad \texttt{Add} = -\texttt{Mult} * \text{GMin}$$

Min und GMax können z.B. mit dem Operator min_max_gray bestimmt werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
Bild(er), dessen/deren Grauwerte skaliert werden sollen.

▷ **ImageScaled** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
Ergebnisbild(er) durch die Skalierung

▷ **Mult** (input_control) ..... number ⇝ *real* / integer
Grauwertskalierung.
Defaultwert : 0.01
Wertevorschläge : Mult ∈ {0.001, 0.003, 0.005, 0.008, 0.01, 0.02, 0.03, 0.05, 0.08, 0.1, 0.5, 1.0}
Typischer Wertebereich : -255.0 ≤ Mult ≤ 255.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **Add** (input_control) ..... number ⇝ *real* / integer
Grauwertverschiebung.
Defaultwert : 0
Wertevorschläge : Add ∈ {0, 10, 50, 100, 200, 500}
Typischer Wertebereich : -512.0 ≤ Add ≤ 512.0
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 1.0

Beispiel

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

Ergebnis

scale_image liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

scale_image ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

min_max_gray

Alternativen

mult_image, add_image, sub_image

Siehe auch

min_max_gray

Modul

Image filters

sub_image (ImageMinuend, ImageSubtrahend : ImageSub : Mult, Add :)

Subtraktion von zwei Bildern.

sub_image subtrahiert zwei Bilder. Die Grauwerte $(g1, g2)$ der Eingabebilder (ImageMinuend und ImageSubtrahend) werden dabei wie folgt transformiert:

$$g' := (g1 - g2) * \texttt{Mult} + \texttt{Add}$$

Tritt ein Überlauf oder ein Unterlauf ein, so werden die Werte beschnitten.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

▷ **ImageMinuend** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
Minuend(en).

▷ **ImageSubtrahend** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
Subtrahend(en).

▷ **ImageSub** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic / complex
Ergebnisbild(er) durch die Substraktion.

▷ **Mult** (input_control) ..... number ⇝ *real* / integer
Korrekturfaktor.
Defaultwert : 1.0
Wertevorschläge : `Mult` ∈ {0.0, 1.0, 2.0, 3.0, 4.0}
Typischer Wertebereich : -255.0 ≤ `Mult` ≤ 255.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **Add** (input_control) ..... number ⇝ *real* / integer
Korrekturwert.
Defaultwert : 128.0
Wertevorschläge : `Add` ∈ {0.0, 128.0, 256.0}
Typischer Wertebereich : -512.0 ≤ `Add` ≤ 512.0
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 1.0

Beispiel

```
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)
```

Ergebnis

`sub_image` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`sub_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`dual_threshold`

Alternativen

`mult_image`, `add_image`, `sub_image`

Siehe auch

`add_image`, `mult_image`, `dyn_threshold`, `check_difference`

Modul

Image filters

3.3 Bit

bit_and (Image1, Image2 : ImageAnd : :)

Bitweises AND aller Pixel der Eingabebilder.

`bit_and` berechnet das bitweise „and“ aller Pixel der Eingabebilder. Die Semantik der „and“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, int/long). Die Bilder müssen die gleiche Bildgröße und den gleichen Pixeltyp haben. Es werden die Bildpunkte innerhalb des Definitionsbereiches des Bildes im ersten Parameter bearbeitet.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

- ▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er) 1.
- ▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er) 2.
- ▷ **ImageAnd** (output_object) .....(multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der AND-Operation.

Beispiel

```
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).
```

Ergebnis

Sind die Bilder korrekt (Typ und Anzahl), dann liefert `bit_and` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`bit_and` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

`bit_mask`, `add_image`, `max_image`

Siehe auch

`bit_mask`, `add_image`, `max_image`

Modul

Image filters

bit_lshift (Image : ImageLShift : Shift :)

Links-Shift aller Pixel des Bildes.

`bit_lshift` berechnet einen bitweisen „links-Shift“ aller Pixel des Eingabebildes. Die Semantik der „links-Shift“ Operation entspricht der von C („<<“) für die jeweiligen Typen (signed char, unsigned char, short, int/long). Tritt ein Überlauf auf, so wird das Ergebnis auf den maximalen Wert des jeweiligen Pixeltyps begrenzt. Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

- ▷ **Image** (input_object) .....(multichannel-)image(-array) ⇝ *Hobject*
  Eingabebild(er).
- ▷ **ImageLShift** (output_object) .....(multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der Shift-Operation.
- ▷ **Shift** (input_control) .....integer ⇝ *integer*
  Shift-Wert.
  Defaultwert : 3
  Wertevorschläge : `Shift` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31}
  Typischer Wertebereich : 0 ≤ `Shift` ≤ 31
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : (`Shift` ≥ `1`) ∧ (`Shift` ≤ `31`)

Beispiel

```
read_image(&ByteImage,"fabrik");
convert_image_type(ByteImage,&Int2Image,"int2");
bit_lshift(Int2Image,&FullInt2Image,8);
```

Ergebnis

Sind die Bilder korrekt (Typ) und hat `Shift` eine gültigen Wert, dann liefert `bit_lshift` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`bit_lshift` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

`scale_image`

Siehe auch

`bit_rshift`

Modul

Image filters

bit_mask (Image : ImageMask : BitMask :)

Logisches „AND“ jedes Pixels mit einer Bitmaske.

`bit_mask` führt eine „and“ Operation von jedem Pixel mit einer festen Maske durch. Die Semantik der „and“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.
Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er).
- ▷ **`ImageMask`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der Kombination mit Mask.
- ▷ **`BitMask`** (input_control) ..... integer ⇝ *integer*
  Bitfeld.
  Defaultwert : 128
  Werteliste : `BitMask` ∈ {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}
  Wertevorschläge : `BitMask` ∈ {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096}

Ergebnis

Sind die Bilder korrekt (Typ), dann liefert `bit_mask` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`bit_mask` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`threshold`, `bit_or`

Alternativen

`bit_slice`

Siehe auch

`bit_and`, `bit_lshift`

Modul

Image filters

bit_not (Image : ImageNot : :)

Komplementieren aller Bits der Pixel.

bit_not berechnet das bitweise „complement“ aller Pixel des Eingabebildes. Die Semantik der „complement“ Operation entspricht der von C („∼“) für die jeweiligen Typen (signed char, unsigned char, short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.
Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er).
- ▷ **ImageNot** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der Komplement-Operation.

Beispiel

```
read_image(Image0,'affe')
disp_image(Image0,WindowHandle)
bit_not(Image0,ImageBitN)
disp_image(ImageBitN,WindowHandle).
```

Ergebnis

Sind die Bilder korrekt (Typ), dann liefert bit_not den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

bit_not ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

bit_or, bit_and, add_image

Siehe auch

bit_slice, bit_mask

Modul

Image filters

bit_or (Image1, Image2 : ImageOr : :)

Bitweises OR aller Pixel der Eingabebilder.

bit_or berechnet das bitweise „or“ aller Pixel der Eingabebilder. Die Semantik der „or“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, int/long). Die Bilder müssen die gleiche Bildgröße und den gleichen Pixeltyp haben. Es werden die Bildpunkte innerhalb des Definitionsbereiches des Bildes im ersten Parameter bearbeitet.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

- ▷ **Image1** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er) 1.
- ▷ **Image2** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er) 2.
- ▷ **ImageOr** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der OR-Operation.

Beispiel

```
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).
```

Ergebnis

Sind die Bilder korrekt (Typ und Anzahl), dann liefert `bit_or` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`bit_or` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

`bit_and`, `add_image`

Siehe auch

`bit_xor`, `bit_and`

Modul

Image filters

`bit_rshift ( Image : ImageRShift : Shift : )`

Rechts-Shift aller Pixel des Bildes.

`bit_rshift` berechnet einen bitweisen „rechts-Shift“ aller Pixel des Eingabebildes. Die Semantik der „rechts-Shift“ Operation entspricht der von C („>>“) für die jeweiligen Typen (signed char, unsigned char, short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.

Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

- **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er).
- **`ImageRShift`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der Shift-Operation.
- **`Shift`** (input_control) ..... integer ⇝ *integer*
  Shift-Wert.
  Defaultwert : 3
  Wertevorschläge : `Shift` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 31}
  Typischer Wertebereich : 0 ≤ `Shift` ≤ 31
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : (`Shift` ≥ `1`) ∧ (`Shift` ≤ `31`)

Beispiel

```
bit_rshift(Int2Image,&ReducedInt2Image,8);
convert_image_type(ReducedInt2Image,&ByteImage,"byte");
```

Ergebnis

Sind die Bilder korrekt (Typ) und hat `Shift` eine gültigen Wert, dann liefert `bit_rshift` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels

set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

bit_rshift ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

scale_image

Siehe auch

bit_lshift

Modul

Image filters

bit_slice (Image : ImageSlice : Bit :)

Extraktion eines Bits aus den Pixeln.

bit_slice extrahiert eine Bitebene aus dem Eingabebild. Die Semantik der „and“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, int/long). Es werden nur die Bildpunkte innerhalb des Definitionsbereiches des Bildes bearbeitet.
Es können mehrere Bilder pro Aufruf bearbeitet werden. Zu jedem Eingabebild wird ein Ausgabebild erzeugt.

Parameter

- **Image** (input_object) ..... (multichannel-)image(-array) $\rightsquigarrow$ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er).
- **ImageSlice** (output_object) ..... (multichannel-)image(-array) $\rightsquigarrow$ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der Extraktion.
- **Bit** (input_control) ..... integer $\rightsquigarrow$ *integer*
  Auszuwählendes Bit.
  Defaultwert : 8
  Wertevorschläge : Bit $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 24, 30, 32\}$
  Typischer Wertebereich : $1 \leq$ Bit ≤ 32
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : $(\text{Bit} \geq 1) \wedge (\text{Bit} \leq 32)$

Beispiel

```
read_image(&ByteImage,"fabrik");
for (bit=1; bit<=8; i++)
{
  bit_slice(ByteImage,&Slice,bit);
  threshold(Slice,&Region,0,255);
  disp_region(Region,WindowHandle);
  clear(bit_slice); clear(Slice); clear(Region);
}
```

Ergebnis

Sind die Bilder korrekt (Typ) und hat Bit eine gültigen Wert, dann liefert bit_slice den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

bit_slice ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

threshold, bit_or

Alternativen

bit_mask

Siehe auch

bit_and, bit_lshift

Modul

Image filters

bit_xor (Image1, Image2 : ImageXor : :)

Bitweise XOR aller Pixel der Eingabebilder.

bit_xor berechnet das bitweise „xor“ aller Pixel der Eingabebilder. Die Semantik der „xor“ Operation entspricht der von C für die jeweiligen Typen (signed char, unsigned char, short, int/long). Die Bilder müssen die gleiche Bildgröße und den gleichen Pixeltyp haben. Es werden die Bildpunkte innerhalb des Definitionsbereiches des Bildes im ersten Parameter bearbeitet.
Es können mehrere Bilder pro Aufruf bearbeitet werden. In diesem Fall enthalten beide Eingabeparameter gleich viele Bilder, die dann paarweise abgearbeitet werden. Zu jedem Paar wird ein Ausgabebild erzeugt.

Parameter

- ▷ **Image1** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er) 1.
- ▷ **Image2** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Eingabebild(er) 2.
- ▷ **ImageXor** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4
  Ergebnis(se) der XOR-Operation.

Beispiel

```
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).
```

Ergebnis

Sind die Bilder korrekt (Typ und Anzahl), dann liefert bit_xor den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

bit_xor ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

bit_or, bit_and, add_image

Siehe auch

bit_or, bit_and

Modul

Image filters

3.4 Color

rgb1_to_gray (RGBImage : GrayImage : :)

Berechnung einer Graubildes aus einem RGB-Bild.

rgb1_to_gray transformiert ein RGB-Bild in ein Graubild. Die drei Kanäle werden in den ersten drei Kanälen des Eingabebildes übergeben. Rot ist der erste, Grün der zweite und Blau der dritte Kanal. Die Transformation erfolgt nach folgender Formel:

$$k = 0.299r + 0.587g + 0.144b \ .$$

Parameter

- ▷ **RGBImage** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Dreikanaliges RGB-Bild.
- ▷ **GrayImage** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Graubild.

Beispiel

```
/* Tranformation from rgb to gray  */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
rgb1_to_gray(Image,GrayImage)
disp_image(GrayImage,WindowHandle).
```

Parallelisierungsinformation

rgb1_to_gray ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

compose3

Alternativen

trans_from_rgb, rgb3_to_gray

Modul

Image filters

rgb3_to_gray (ImageRed, ImageGreen, ImageBlue : ImageGray : :)

Berechnung einer Graubildes aus drei Bildern (RGB).

rgb3_to_gray transformiert ein RGB-Bild in ein Graubild. Die drei Kanäle werden als drei getrennte Bilder übergegeben. Die Transformation erfolgt nach folgender Formel:

$$k = 0.299r + 0.587g + 0.144b \ .$$

Parameter

- ▷ **ImageRed** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild (Rot-Kanal).
- ▷ **ImageGreen** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild (Grün-Kanal).
- ▷ **ImageBlue** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild (Blau-Kanal).
- ▷ **ImageGray** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Graubild.

Beispiel

```
/* Tranformation 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).
```

Parallelisierungsinformation

`rgb3_to_gray` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`decompose3`

Alternativen

`rgb1_to_gray`, `trans_from_rgb`

Modul

Image filters

trans_from_rgb (ImageRed, ImageGreen, ImageBlue : ImageResult1, ImageResult2, ImageResult3 : ColorSpace :)

Transformation vom RGB- in einen anderen Farbraum.

`trans_from_rgb` transformiert ein Bild von RGB in den angegebenen Farbraum (`ColorSpace`). Die drei Kanäle werden als drei getrennte Bilder sowohl ein- als auch ausgegeben.

Folgende Transformationen werden unterstützt:

’yiq’

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

’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’

```
min = min(R,G,B)
max = max(R,G,B)
L = (min + max) / 2
if (max == min)
   H = 0
   S = 0
else
   if (L > 0.5)
      S = (max - min) / (2 - max - min)
   else
      S = (max - min) / (max + min)
   fi
   if (R == max)
      H = ((G - B) / (max - min)) * 60
   elif (G == max)
      H = (2 + (B - R) / (max - min)) * 60
   elif (B == max)
      H = (4 + (R - G) / (max - min)) * 60
   fi
fi
```

'hsi'

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

$$\begin{pmatrix} H \\ S \\ I \end{pmatrix} = \begin{pmatrix} \arctan \frac{M2}{M1} \\ \sqrt{M1^2 + M2^2} \\ \frac{I1}{\sqrt{3}} \end{pmatrix}$$

'hsv'
```
min = min(R,G,B)
max = max(R,G,B)
V = max
if (max == min)
   S = 0
   H = 0
else
   S = (max - min) / max
   if (R == max)
      H = ((G - B) / (max - min)) * 60
   elif (G == max)
      H = (2 + (B - R) / (max - min)) * 60
   elif (B == max)
      H = (4 + (R - G) / (max - min)) * 60
   fi
fi
```

'ihs'
```
min = min(R,G,B)
max = max(R,G,B)
I = (R + G + B) / 3
if (I == 0)
   H = 0
   S = 1
else
   S = 1 - min / I
   if (S == 0)
      H = 0
   else
      A = (R + R - G - B) / 2
      B = (R - G) * (R - G) + (R - B) * (G - B)
      C = sqrt(B)
      if (C == 0)
         H = 0
      else
         H = acos(A / C)
      fi
      if (B > G)
         H = 2 * pi - H
      fi
   fi
fi
```

'isfeuklid'
```
min = min(R,G,B)
max = max(R,G,B)
I = sqrt(R * R + G * G + B * B) / sqrt(3)
if (I == 0)
   S = 0
   F = 0
else
   S = acos(max(1, (R + G + B) / (I * 3))) / (pi / 2)
   if (R == min)
```

```
            F = acos(max(1, B / sqrt(G * G + B * B)))
                / (pi * 3 / 2) + 1 / 3
         elif (G == min)
            F = acos(max(1, R / sqrt(B * B + R * R)))
                / (pi * 3 / 2) + 2 / 3
         else
            F = acos(max(1, G / sqrt(R * R + G * G)))
                / (pi * 3 / 2)
         fi
      fi
```

'isfdiff'

```
      if (R >= B && B >= G)
         I = R
         S = R - G
         F = (R - B) / 6
      elif (R >= G && G >= B)
         I = R
         S = R - B
         F = (2 - (R - G)) / 6
      elif (G >= R && R >= B)
         I = G
         S = G - B
         F = (2 + (G - R)) / 6
      elif (G >= B && B >= R)
         I = G
         S = G - R
         F = (4 - (G - B)) / 6
      elif (B >= G && G >= R)
         I = B
         S = B - R
         F = (4 + (B - G)) / 6
      else
         I = B
         S = B - G
         F = (6 + (B - R)) / 6
      fi
```

'cielab'

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

$$L = 116 * Y^{\frac{1}{3}} - 16$$

$$a = 500 * (\frac{X}{0.95}^{\frac{1}{3}} - Y^{\frac{1}{3}})$$

$$b = 200 * (Y^{\frac{1}{3}} - \frac{Z}{1.09}^{\frac{1}{3}})$$

'i1i2i3'

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

'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'

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

wobei zusätzlich entsprechend des Datentyps der beteiligten Bilder noch Skalierungen vorgenommen werden wenn eine der folgende Bedingungen zutrifft:

- Der Wertebereich der Farbraumgrößen wird bei *byte*-Bildern generell auf den vollen Bereich von [0..255] abgebidet. Bei vorzeichenbehafteten Farbraumgrößen wird der Nullpunkt auf 128 verschoben.
- Hue Werte werden im Winkelmaß von $[0..2\pi]$ angegeben, die für die jeweiligen Bildtypen unterschiedlich kodiert werden:
  - *byte*-Bilder bilden den Winkelbereich auf $[0..255]$ ab.
  - *int4*-Bilder werden in Winkelminuten $[0..21600]$ kodiert.
  - *real*-Bilder werden in Radiant $[0..2\pi]$ kodiert.
- Bei *int4*-Bildern wird ein Wertebereich von $[0..1]$ einer transformierten Farbgröße auf $[0..10000]$ abgebildet.

Parameter

▷ **ImageRed** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
Eingabebild (Rot-Kanal).

▷ **ImageGreen** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
Eingabebild (Grün-Kanal).

▷ **ImageBlue** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
Eingabebild (Blau-Kanal).

▷ **ImageResult1** (output_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
Farbtransformiertes Ausgabebild (Kanal 1).

▷ **ImageResult2** (output_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
Farbtransformiertes Ausgabebild (Kanal 2).

▷ **ImageResult3** (output_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / complex
Farbtransformiertes Ausgabebild (Kanal 3).

▷ **ColorSpace** (input_control) . . . . . string ⇝ *string*
Gewünschte Farbtransformation.
Defaultwert : 'hsv'
Werteliste : `ColorSpace` ∈ {'cielab', 'hsv', 'hsi', 'yiq', 'argyb', 'ciexyz', 'ciexyz2', 'ciexyz3', 'hls', 'ihs', 'isfeuklid', 'isfdiff', 'i1i2i3'}

Beispiel

```
/* Tranformation from rgb to hsv and conversely */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image,Rimage,Gimage,Bimage)
trans_from_rgb(Rimage,Gimage,Bimage,Image1,Image2,Image3,'hsv')
trans_to_rgb(Image1,Image2,Image3,ImageRed,ImageGreen,ImageBlue,'hsv')
compose3(ImageRed,ImageGreen,ImageBlue,Multichannel)
disp_color(Multichannel,WindowHandle).
```

Ergebnis

`trans_from_rgb` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`trans_from_rgb` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`decompose3`

Mögliche Nachfolgerfunktionen

compose3

Alternativen

rgb1_to_gray, rgb3_to_gray

Siehe auch

trans_to_rgb

Modul

Image filters

trans_to_rgb (ImageInput1, ImageInput2, ImageInput3 : ImageRed, ImageGreen, ImageBlue : ColorSpace :)

Transformation von einem Farbraum nach RGB.

trans_to_rgb transformiert ein Bild aus einem angegebenen Farbraum (ColorSpace) nach RGB. Die drei Kanäle werden als drei getrennte Bilder sowohl ein- als auch ausgegeben.

Folgende Transformationen werden unterstützt:

'yiq'

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

'argyb'

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

'ciexyz'

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

'hls'

```
Hi = integer(H * 6)
Hf = fraction(H * 6)
if (L <= 0.5)
   max = L * (S + 1)
else
   max = L + S - (L * S)
fi
min = 2 * L - max
if (S == 0)
   R = L
   G = L
   B = L
else
  if (Hi == 0)
     R = max
     G = min + Hf * (max - min)
     B = min
  elif (Hi == 1)
     R = min + (1 - Hf) * (max - min)
     G = max
     B = min
  elif (Hi == 2)
     R = min
```

```
            G = max
            B = min + Hf * (max - min)
         elif (Hi == 3)
            R = min
            G = min + (1 - Hf) * (max - min)
            B = max
         elif (Hi == 4)
            R = min + Hf * (max - min)
            G = min
            B = max
         elif (Hi == 5)
            R = max
            G = min
            B = min + (1 - Hf) * (max - min)
         fi
       fi
```

'hsi'

$$M1 = S * \cos H$$

$$M2 = S * \sin H$$

$$I1 = \frac{I}{\sqrt{3}}$$

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

'hsv'

```
   if (S == 0)
     if (H == 0)
        R = V
        G = V
        B = V
     else
        R = 0
        G = 0
        B = 0
     fi
   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
```

wobei entsprechend des Datentyps die Eingabe-Bilder entsprechend skaliert erwartet werden, wenn eine der folgende Bedingungen zutrifft:

- Der Wertebereich der Farbraumgrößen wird bei *byte*-Bildern generell in dem vollen Bereich von [0..255] erwartet. Bei vorzeichenbehafteten Farbraumgrößen muss der Nullpunkt auf 128 verschoben sein.
- Hue Werte werden im Winkelmaß von $[0..2\pi]$ angegeben, die für die jeweiligen Bildtypen unterschiedlich kodiert werden:
  - *byte*-Bilder bilden den Winkelbereich auf $[0..255]$ ab.
  - *int4*-Bilder werden in Winkelminuten $[0..21600]$ kodiert.
  - *real*-Bilder werden in Radiant $[0..2\pi]$ kodiert.
- Bei *int4*-Bildern muss ein Wertebereich von $[0..1]$ einer transformierten Farbgröße auf den Bereich von $[0..10000]$ gespreizt sein.

Parameter

▷ **ImageInput1** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real
Eingabebild (1. Kanal).

▷ **ImageInput2** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real
Eingabebild (2. Kanal).

▷ **ImageInput3** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real
Eingabebild (3. Kanal).

▷ **ImageRed** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real
Rot-Kanal.

▷ **ImageGreen** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real
Grün-Kanal.

▷ **ImageBlue** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real
Blau-Kanal.

▷ **ColorSpace** (input_control) ..... string ⇝ *string*
Gewünschte Farbtransformation.
Defaultwert : 'hsv'
Werteliste : ColorSpace ∈ {'hsi', 'yiq', 'argyb', 'ciexyz', 'hls', 'hsv'}

Beispiel

```
/* Tranformation from rgb to hsv and conversely */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
decompose3(Image,Rimage,Gimage,Bimage)
trans_from_rgb(Rimage,Gimage,Bimage,Image1,Image2,Image3,'hsv')
trans_to_rgb(Image1,Image2,Image3,ImageRed,ImageGreen,ImageBlue,'hsv')
compose3(ImageRed,ImageGreen,ImageBlue,Multichannel)
disp_color(Multichannel,WindowHandle).
```

Ergebnis

trans_to_rgb liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

trans_to_rgb ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`decompose3`

Mögliche Nachfolgerfunktionen

`compose3`, `disp_color`

Siehe auch

`decompose3`

Modul

Image filters

3.5 FFT

`convol_fft` (ImageFFT, ImageFilter : ImageConvol : :)

Faltung (Convolution) mit einer byte-Maske in Frequenzraum.

`convol_fft` führt eine Filterung der (fouriertransformierten) Eingabebilder im Frequenzraum durch. Eine Filterung im Frequenzraum bedeutet eine Multiplikation der Pixel des komplexen Bildes `ImageFFT` mit den zugehörigen Pixel des Filters `ImageFilter`. Der Filter ist ein byte-Bild, bei dem der Wert 0 vollständige Unterdrückung einer Frequenz und 255 keine Unterdrückung bedeuted. Das Ausgabebild ist komplex.

Achtung

Die Filterung erfolgt immer im Vollbild (Region wird ignoriert).

Parameter

- ▷ **`ImageFFT`** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Komplexes Eingabebild.
- ▷ **`ImageFilter`** (input_object) ..... image ⇝ *Hobject* : byte
  Filter im Frequenzraum.
- ▷ **`ImageConvol`** (output_object) ..... image(-array) ⇝ *Hobject* : complex
  Ergebnis der Filterung (Real- und Imaginärteile).

Beispiel

```
my_highpass(Hobject Image, Hobject *Result, int frequency, int size)
{
  Hobject  FFT, Highpass, FFTConvol;
  fft_image(Image,&FFT);
  gen_highpass(&Highpass,frequency,size);
  convol_fft(FFT,Highpass,&FFTConvol);
  clear_obj(Highpass); clear_obj(FFT);
  fft_image_inv(FFTConvol,Result);
  clear_obj(FFTConvol);
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `convol_fft` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`convol_fft` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`fft_image`, `fft_generic`, `gen_highpass`, `gen_lowpass`, `gen_bandpass`, `gen_bandfilter`

Mögliche Nachfolgerfunktionen

`power_byte`, `power_real`, `power_ln`, `fft_image_inv`, `fft_generic`

Alternativen

`convol_gabor`

Siehe auch

gen_gabor, gen_highpass, gen_lowpass, gen_bandpass, convol_gabor, fft_image_inv

Modul

Image filters

convol_gabor (ImageFFT, GaborFilter : ImageResultGabor, ImageResultHilbert : :)

Faltung mit einem Gaborfilter im Frequenzraum.

convol_gabor berechnet die Faltung des Gaborfilters GaborFilter (siehe gen_gabor) und seiner Hilbert-transformierten mit einem bereits fouriertransformierten Bild im Frequenzraum. Das Ausgabebild ist komplex.

Achtung

Die Filterung erfolgt immer im Vollbild (Region wird ignoriert).

Parameter

- ▷ **ImageFFT** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild (Real- und Imaginärteile).
- ▷ **GaborFilter** (input_object) ..... multichannel-image ⇝ *Hobject* : real
  Gabor/Hilbert-Filter.
- ▷ **ImageResultGabor** (output_object) ..... image(-array) ⇝ *Hobject* : complex
  Ergebnis Gaborfilterung.
- ▷ **ImageResultHilbert** (output_object) ..... image(-array) ⇝ *Hobject* : complex
  Ergebnis Hilbertfilterung.

Beispiel

```
fft_image(Image,&FFT);
gen_gabor(&Filter,1.4,0.4,1.0,1.5,512);
convol_gabor(FFT,Filter,&Gabor,&Hilbert);
fft_image_inv(Gabor,&GaborInv);
fft_image_inv(Hilbert,&HilbertInv);
energy_gabor(GaborInv,HilbertInv,&Energy);
```

Ergebnis

Sind die Bilder vom richtigen Typ, dann liefert convol_gabor den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

convol_gabor ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

fft_image, fft_generic, gen_gabor

Mögliche Nachfolgerfunktionen

power_byte, power_real, power_ln, fft_image_inv, fft_generic

Alternativen

convol_fft

Siehe auch

convol_image

Modul

Image filters

energy_gabor (ImageGabor, ImageHilbert : Energy : :)

Energie eines zweikanaligen Bildes.

energy_gabor berechnet den lokalen Kontrast (Energy) der zwei Eingabebilder. Die Energie des Bildes ergibt sich dann zu

$$\texttt{Energy} = \text{Kanal1}^2 + \text{Kanal2}^2 \ .$$

Häufig geht der Bestimmung der Energie die Faltung eines Bildes mit einem Gaborfilter und der Hilberttransformierten des Gaborfilters voraus (siehe convol_gabor). Als Kanal 1 wird energy_gabor dann das gaborgefilterte Bild, zurücktransformiert in den Ortsraum (siehe fft_image_inv) übergeben, als Kanal 2 das rücktransformierte Ergebnis der Faltung mit der Hilberttransformierten. Die lokale Energie ist ein Maß für den lokalen Kontrast von Strukturen (Kanten, Linien) in Bildern.

Parameter

- ▷ **ImageGabor** (input_object) ..... image(-array) ⇝ *Hobject* : byte / real
  Eingabe 1. Kanal (typisch: Gaborbild).
- ▷ **ImageHilbert** (input_object) ..... image(-array) ⇝ *Hobject* : byte / real
  Eingabe 2. Kanal (typisch: Hilbertbild).
- ▷ **Energy** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Bild mit der lokalen Energie.

Beispiel

```
fft_image(Image,&FFT);
gen_gabor(&Filter,1.4,0.4,1.0,1.5,512);
convol_gabor(FFT,Filter,&Gabor,&Hilbert);
fft_image_inv(Gabor,&GaborInv);
fft_image_inv(Hilbert,&HilbertInv);
energy_gabor(GaborInv,HilbertInv,&Energy);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert energy_gabor den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

energy_gabor ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gen_gabor, convol_gabor, fft_image_inv

Modul

Image filters

fft_generic (Image : ImageFFT : Direction, Exponent, Norm, Mode :)

Schnelle Fouriertransformation.

fft_generic berechnet die schnelle Fouriertransformation des Eingabebildes Image. Da in der Literatur mehrere Definitionen der Hin- und Rücktransformation der Fouriertransformation existieren, erlaubt diese Prozedur dem Benutzer die Auswahl der für seine Bedürfnisse geeigneten Version.

Die allgemeine Version einer Fouriertransformation sieht wie folgt aus:

$$F(m,n) = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} e^{s2\pi i(km/M+ln/N)} f(k,l)$$

In der Literatur herrscht keine Einigkeit, ob das Vorzeichen s im Exponenten für die Hintransformation mit 1 oder -1 zu besetzen ist. Ebenso wenig Klarheit herrscht über den Normierungsfaktor c. Dieser wird manchmal für die Hintransformation mit 1 angegeben, manchmal mit MN, manchmal wird er auch (für die unitäre FFT) zu

$\sqrt{MN}$ gesetzt. Insbesondere in der Bildverarbeitung wird außerdem manchmal der konstante Term der FFT (im englischen DC-Term genannt) in die Bildmitte verschoben.

`fft_generic` erlaubt es, diese Auswahlmöglichkeiten individuell anzupassen. Mit `Direction` kann die logische Richtung der FFT bestimmt werden. (Dieser Parameter ist nicht überflüssig; Er dient dazu, zu unterscheiden, wie das Bild verschoben werden muß, falls der DC-Term in der Mitte des Bildes liegen soll.) Mögliche Werte sind **'to_freq'** und **'from_freq'**. Mit dem Parameter `Exponent` kann der Exponent der Transformation festgelegt werden. Er kann die Werte 1 und -1 annehmen. Mit `Norm` kann die gewünschte Normierung angegeben werden. Zulässige Werte sind **'none'**, **'sqrt'** und **'n'**. Mit `Mode` kann angegeben werden, wo der DC-Term der FFT liegen soll. Dabei kann zwischen **'dc_center'** und **'dc_edge'** gewählt werden.

In jedem Fall ist auf die konsistente Verwendung der Parameter zu achten. D.h., die Normierungsfaktoren bei der Hin- und Rücktransformation müssen aufmultipliziert MN ergeben. Bei der Rücktransformation ist ein anderes Vorzeichen für den Exponenten zu wählen als bei der Hintransformation. `Mode` muß für beide Transformationen gleich gewählt werden.

Eine sinnvolle Kombination ist z.B. **('to_freq',-1,'n','dc_edge')** für die Hintransformation und **('from_freq',1,'none','dc_edge')** für die Rücktransformation. In diesem Fall kann die FFT als Interpolation mit trigonometrischen Basisfunktionen interpretiert werden. Eine andere mögliche Kombination ist **('to_freq',1,'sqrt','dc_center')** und **('from_freq',-1,'sqrt','dc_center')**.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : any
  Eingabebild im Ortsraum.
- ▷ **`ImageFFT`** (output_object) ..... image(-array) ⇝ *Hobject* : complex
  Fouriertransformiertes Bild.
- ▷ **`Direction`** (input_control) ..... string ⇝ *string*
  Hin- oder Rücktransformation.
  Defaultwert : 'to_freq'
  Werteliste : `Direction` ∈ {'to_freq', 'from_freq'}
- ▷ **`Exponent`** (input_control) ..... integer ⇝ *integer*
  Vorzeichen des Exponenten.
  Defaultwert : 1
  Werteliste : `Exponent` ∈ {-1, 1}
- ▷ **`Norm`** (input_control) ..... string ⇝ *string*
  Normierung des Ergebnisbildes.
  Defaultwert : 'sqrt'
  Werteliste : `Norm` ∈ {'none', 'sqrt', 'n'}
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Position der Nullfrequenz im Frequenzraum.
  Defaultwert : 'dc_center'
  Werteliste : `Mode` ∈ {'dc_center', 'dc_edge'}

Beispiel

```
/* simulation of fft */
my_fft(Hobject In, Hobject *Out)
{
  fft_generic(In,Out,'to_freq',1,'sqrt','dc_center');
}

/* simulation of fft_image_inv */
my_fft_image_inv(Hobject In, Hobject *Out)
{
  Hobject  Tmp;
  fft_generic(In,&Tmp,'from_freq',1,'sqrt','dc_center');
  convert_image_type(Tmp,Out,"byte");
  clear_obj(Tmp);
}
```

Ergebnis

Ist die Kantenlänge des Eingabebildes eine Zweierpotenz und der Pixeltyp korrekt, und sind alle Parameterwerte

korrekt, dann liefert `fft_generic` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fft_generic` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`convol_fft`, `convol_gabor`, `convert_image_type`, `power_byte`, `power_real`, `power_ln`, `phase_deg`, `phase_rad`, `energy_gabor`

Alternativen

`fft_image`, `fft_image_inv`

Modul

Image filters

`fft_image` (Image : ImageFFT : :)

Schnelle Fouriertransformation.

`fft_image` berechnet die Fouriertransformierte des Eingabebildes (`Image`), vollzieht also den Übergang vom Orts- in den Frequenzraum. Die Berechnung erfolgt mittels des Fast-Fourier-Algorithmus. Dies entspricht dem Aufruf von

```
fft_generic(Image,ImageFFT,'to_freq',1,'sqrt','dc_center':)
```

Das Ausgabebild ist komplex.

Achtung

Die Berechnung erfolgt immer im Vollbild (Region wird ignoriert). Die Bilder müssen quadratisch sein und die Kantenlänge muß eine Zweierpotenz sein.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / real
  Eingabebild im Ortsraum
- ▷ **`ImageFFT`** (output_object) ..... image(-array) ⇝ *Hobject* : complex
  Fouriertransformiertes Bild.

Beispiel

```
fft_image(Image,&FFT);
gen_gabor(&Filter,1.4,0.4,1.0,1.5,512);
convol_gabor(FFT,Filter,&Gabor,&Hilbert);
fft_image_inv(Gabor,&GaborInv);
fft_image_inv(Hilbert,&HilbertInv);
energy_gabor(GaborInv,HilbertInv,&Energy);
```

Ergebnis

Ist die Kantenlänge des Eingabebildes eine Zweierpotenz und der Pixeltyp korrekt, dann liefert `fft_image` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fft_image` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`convol_fft`, `convol_gabor`, `convert_image_type`, `power_byte`, `power_real`, `power_ln`, `phase_deg`, `phase_rad`

Alternativen

`fft_generic`

Siehe auch

fft_image_inv

Modul

Image filters

fft_image_inv (Image : ImageFFTInv : :)

Schnelle inverse Fouriertransformation.

fft_image_inv führt auf den Eingabebildern die inverse Fouriertransformation aus, vollzieht also den Übergang vom Frequenzraum zurück in den Ortsraum. Das Ausgabebild ist vom Type **'byte'**. Dies entspricht dem Aufruf von

```
fft_generic(Image,ImageFFT,'from_freq',-1,'sqrt','dc_center')
convert_image_type(ImageFFT,ImageFFTInv,'byte')
```

Das Ausgabebild ist vom Type **'byte'**.

Achtung

Die Berechnung erfolgt immer im Vollbild (Region wird ignoriert). Die Bilder müssen quadratisch sein und die Kantenlänge muß eine Zweierpotenz sein.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild (Frequenzraum).
- ▷ **ImageFFTInv** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Rücktransformierte Bilder (Ortsraum).

Beispiel

```
fft_image(Image,&FFT);
gen_gabor(&Filter,1.4,0.4,1.0,1.5,512);
convol_gabor(FFT,Filter,&Gabor,&Hilbert);
fft_image_inv(Gabor,&GaborInv);
fft_image_inv(Hilbert,&HilbertInv);
energy_gabor(GaborInv,HilbertInv,&Energy);
```

Ergebnis

Ist der Typ der Bildes korrekt, dann liefert fft_image_inv den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fft_image_inv ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

convol_fft, convol_gabor, fft_image

Mögliche Nachfolgerfunktionen

convert_image_type, energy_gabor

Alternativen

fft_generic

Siehe auch

fft_image, fft_generic, energy_gabor

Modul

Image filters

gen_bandfilter (: ImageFilter : MinFrequency, MaxFrequency, Size :)

Erzeugen eines idealen Bandfilters.

gen_bandfilter erzeugt einen idealen Bandfilter im Frequenzraum. Die Nullfrequenz wird im Bildmittelpunkt angenommen. Die Frequenzen (als Abstand vom Mittelpunkt in Pixel) geben die Parameter MinFrequency und MaxFrequency vor. Das Ergebnis ist ein Bild, bei dem ein Ring um das Zentrum auf 0 gesetzt ist. Alle anderen Pixel haben den Wert 255.

Parameter

- ▷ **ImageFilter** (output_object) ..... image ⇝ *Hobject* : byte
  Bandfilter im Frequenzraum.
- ▷ **MinFrequency** (input_control) ..... real ⇝ *real*
  Minimale Frequenz.
  Defaultwert : 20
  Wertevorschläge : MinFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : 1 ≤ MinFrequency
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : MinFrequency > 0
- ▷ **MaxFrequency** (input_control) ..... real ⇝ *real*
  Maximale Frequenz.
  Defaultwert : 40
  Wertevorschläge : MaxFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : 1 ≤ MaxFrequency
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : MaxFrequency > 0
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Beispiel

```
my_lowpass(Hobject Image, Hobject *Result)
{
  Hobject  FFT, Bandpass, FFTConvol;
  fft_image(Image,&FFT);
  gen_bandfilter(&Bandpass,20,40,512);
  convol_fft(FFT,Bandpass,&FFTConvol);
  clear_obj(Bandpass); clear_obj(FFT);
  fft_image_inv(FFTConvol,Result);
  clear_obj(FFTConvol);
}
```

Ergebnis

Sind die Parameter korrekt, dann liefert gen_bandfilter den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_bandfilter ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

convol_fft

Alternativen

gen_circle, paint_region

Siehe auch

gen_highpass, gen_lowpass, gen_bandpass

Modul

Image filters

gen_bandpass (: ImageBandpass : MinFrequency, MaxFrequency, Size :)

Erzeugen eines idealen Bandpaßfilters.

gen_bandpass erzeugt einen idealen Bandpaßfilter im Frequenzraum. Die Nullfrequenz wird im Bildmittelpunkt angenommen. Die Frequenzen (als Abstand vom Mittelpunkt in Pixel) geben die Parameter MinFrequency und MaxFrequency vor. Das Ergebnis ist ein Bild, das mit Außnahme eines Ringes auf Null gesetzt ist.

Parameter

- ▷ **ImageBandpass** (output_object) ..... image ⇝ *Hobject* : byte
  Bandpass im Frequenzraum.
- ▷ **MinFrequency** (input_control) ..... real ⇝ *real*
  Minimale Frequenz.
  Defaultwert : 20
  Wertevorschläge : MinFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : 1 ≤ MinFrequency ≤ 200
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : MinFrequency > 0
- ▷ **MaxFrequency** (input_control) ..... real ⇝ *real*
  Maximale Frequenz.
  Defaultwert : 40
  Wertevorschläge : MaxFrequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : 1 ≤ MaxFrequency ≤ 200
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : MaxFrequency > 0
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Beispiel

```
my_lowpass(Hobject Image, Hobject *Result)
{
  Hobject  FFT, Bandpass, FFTConvol;
  fft(Image,&FFT);
  gen_bandpass(&Bandpass,20,40,512);
  convol_fft(FFT,Bandpass,&FFTConvol);
  clear_obj(Bandpass); clear_obj(FFT);
  fft_image_inv(FFTConvol,Result);
  clear_obj(FFTConvol);
}
```

Ergebnis

Sind die Parameter korrekt, dann liefert gen_bandpass den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_bandpass ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

convol_fft

Alternativen

gen_circle, paint_region

Siehe auch

gen_highpass, gen_lowpass, gen_bandfilter

Modul

Image filters

gen_filter_mask (: ImageFilter : FilterMask, Scale, Size :)

Eintragen einer Filtermaske im Ortsraum in ein real-Bild.

gen_filter_mask trägt eine Filtermaske im Ortsraum in ein Bild ein. Das Zentrum des Filters kommt in der Mitte des Bildes zu liegen. Der Parameter Scale gibt an, um wieviel die Filterkoeffizienten zu multiplizieren sind (dies führt zu einer stärkeren Antwort bei der Fouriertransformation). Die Filtermatrix, die durch FilterMask definiert ist, kann dabei entweder aus einer Datei oder einem Tupel generiert werden. Das Format für die Filtermatrix ist bei convol_image beschrieben. Beispieldateien sind im Directory „filter“ unter dem HALCON-Homedirectory zu finden. Die Operation kann gut zur Visualisierung von Filter (durch anschließende Fouriertransformation) verwendet werden.

Parameter

- ▷ **ImageFilter** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Filter im Ortsraum.
- ▷ **FilterMask** (input_control) ..... string(-array) ⇝ *string* / integer
  Filtermaske als Dateiname oder Tupel.
  Defaultwert : 'sobel'
  Wertevorschläge : FilterMask ∈ {'laplace4', 'laplace8', 'lowpass_3_3'}
- ▷ **Scale** (input_control) ..... real ⇝ *real*
  Skalierungsfaktor.
  Defaultwert : 1.0
  Wertevorschläge : Scale ∈ {0.3, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0}
  Typischer Wertebereich : 0.001 ≤ Scale ≤ 10.0
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.1
  Restriktion : Scale > 0.0
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Beispiel

```
gen_filter_mask(&Filter,"lowpass_3_3",1.0,512);
fft_image(Filter,&FilterFFT);
set_paint(WindowHandle,"3D-plot_hidden");
disp_image(FilterFFT,WindowHandle);
```

Parallelisierungsinformation

gen_filter_mask ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

fft_image, fft_generic

Siehe auch

convol_image

Modul

Image filters

gen_gabor (: ImageFilter : Angle, Frequency, Bandwidth, Orientation, Size :)

Erzeugen eines Gaborfilters.

gen_gabor erzeugt einen Gaborfilter, dessen Bandpaßverhalten und die zugehörigen Signumwerte für die Berechnung der Hilberttransformierten durch die Eingabeparameter einstellbar ist. Dazu wird ein symmetrischer Filter im Frequenzraum errechnet, der sich durch die Parameter Angle, Frequency, Bandwidth und Orientation so einstellen läßt, daß ein bestimmter Frequenzbereich in einem bestimmten Richtungsbereich eines Bildes im Ortsraum durch die Faltung im Ortsfrequenzraum herausgefiltert wird.

Die Parameter `Frequency` (Mittenfrequenz = Abstand vom Mittelpunkt) und `Orientation` (Winkel) geben das Zentrum des Filters an. Größere Werte von `Frequency` ergeben höhere Frequenzen. Ein Wert von 0 bei `Orientation` erzeugt eine „Sichel“ die horizontal ausgerichtet ist (Öffnung nach „unten“). Höhere Winkelwerte lassen die Sichel in mathematisch positiver Richtung rotieren.

Mit den Parametern `Angle` und `Bandwidth` kann die Größe des Richtungsbereiches und des Frequenzbereiches geändert werden. Je größer der Parameter `Angle` ist, desto kleiner wird der Winkelbereich (kürzere Sichel), das heißt, desto kleiner wird der Richtungsbereich. Je größer der Parameter `Bandwidth`, desto kleiner wird der Frequenzbereich (dünnere Sichel).

Die Berechnung liefert ein zweikanaliges Real-Bild, welches im ersten Kanal den Gaborfilter und im zweiten Kanal den zugehörigen Hilbertfilter enthält.

Parameter

▷ **`ImageFilter`** (output_object) ..... multichannel-image(-array) ⇝ *Hobject* : real
Gabor- und Hilbertfilter.

▷ **`Angle`** (input_control) ..... real ⇝ *real*
Winkel, indirekt proportional zur Größe des Richtungsbereichs.
Defaultwert : 1.4
Wertevorschläge : `Angle` ∈ {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}
Typischer Wertebereich : 1.0 ≤ `Angle` ≤ 500.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **`Frequency`** (input_control) ..... real ⇝ *real*
Abstand des Filterschwerpunktes vom Mittelpunkt.
Defaultwert : 0.4
Wertevorschläge : `Frequency` ∈ {0.0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.50, 0.55, 0.60, 0.65, 0.699}
Typischer Wertebereich : 0.0 ≤ `Frequency` ≤ 0.7
Minimale Schrittweite : 0.00001
Empfohlene Schrittweite : 0.005

▷ **`Bandwidth`** (input_control) ..... real ⇝ *real*
Bandbreite, indirekt proportional zur Größe des Frequenzbereichs.
Defaultwert : 1.0
Wertevorschläge : `Bandwidth` ∈ {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}
Typischer Wertebereich : 0.05 ≤ `Bandwidth` ≤ 100.0
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1

▷ **`Orientation`** (input_control) ..... real ⇝ *real*
Richtung, legt zusammen mit Frequency die Lage des Filterschwerpunktes fest.
Defaultwert : 1.5
Wertevorschläge : `Orientation` ∈ {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}
Typischer Wertebereich : 0.0 ≤ `Orientation` ≤ 3.1416
Minimale Schrittweite : 0.0001
Empfohlene Schrittweite : 0.05

▷ **`Size`** (input_control) ..... integer ⇝ *integer*
Größe (Kantenlänge) des Bildes (Filters).
Defaultwert : 512
Werteliste : `Size` ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Beispiel

```
fft_image(Image,&FFT);
gen_gabor(&Filter,1.4,0.4,1.0,1.5,512);
convol_gabor(FFT,Filter,&Gabor,&Hilbert);
fft_image_inv(Gabor,&GaborInv);
fft_image_inv(Hilbert,&HilbertInv);
energy_gabor(GaborInv,HilbertInv,&Energy);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_gabor` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_gabor` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`fft_image`, `fft_generic`

Mögliche Nachfolgerfunktionen

`convol_gabor`

Alternativen

`gen_bandpass`, `gen_bandfilter`, `gen_highpass`, `gen_lowpass`

Siehe auch

`fft_image_inv`, `energy_gabor`

Modul

Image filters

gen_highpass (: ImageHighpass : Frequency, Size :)

Erzeugen eines idealen Hochpaßfilters.

`gen_highpass` erzeugt einen idealen Hochpaßfilter im Frequenzraum. Die Nullfrequenz wird im Bildmittelpunkt angenommen. Die Frequenz (als Abstand vom Mittelpunkt in Pixel) gibt der Parameter `Frequency` vor. Das Ergebnis ist ein Bild, das im inneren Bereich (Kreis mit Radius `Frequency`) den Wert Null hat und außerhalb dieses Bereiches auf 255 gesetzt ist.

Parameter

- ▷ **ImageHighpass** (output_object) ..... image ⇝ *Hobject* : byte
  Hochpaßfilter im Frequenzraum.
- ▷ **Frequency** (input_control) ..... real ⇝ *real*
  Trennfrequenz.
  Defaultwert : 20
  Wertevorschläge : `Frequency` ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : $1 \leq$ `Frequency` ≤ 200
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `Frequency` $>$ `0`
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : `Size` ∈ {16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Beispiel

```
my_highpass(Hobject Image, Hobject *Result, int frequency, int size)
{
  Hobject  FFT, Highpass, FFTConvol;
  fft_image(Image,&FFT);
  gen_highpass(&Highpass,frequency,size);
  convol_fft(FFT,Highpass,&FFTConvol);
  clear_obj(Highpass); clear_obj(FFT);
  fft_image_inv(FFTConvol,Result);
  clear_obj(FFTConvol);
}
```

Ergebnis

Sind die Parameter korrekt, dann liefert `gen_highpass` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_highpass ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

convol_fft

Alternativen

gen_circle, paint_region

Siehe auch

convol_fft, gen_lowpass, gen_bandpass, gen_bandfilter

Modul

Image filters

gen_lowpass (: ImageLowpass : Frequency, Size :)

Erzeugen eines idealen Tiefpaßfilters.

gen_lowpass erzeugt einen idealen Tiefpaßfilter im Frequenzraum. Die Nullfrequenz wird im Bildmittelpunkt angenommen. Die Frequenz (als Abstand vom Mittelpunkt in Pixel) gibt der Parameter Frequency vor. Das Ergebnis ist ein Bild, das im inneren Bereich (Kreis mit Radius Frequency) den Wert 255 hat und außerhalb dieses Bereiches auf 0 gesetzt ist.

Parameter

- ▷ **ImageLowpass** (output_object) ..... image ⇝ *Hobject* : byte
  Tiefpaß im Frequenzraum.
- ▷ **Frequency** (input_control) ..... real ⇝ *real*
  Trennfrequenz.
  Defaultwert : 20
  Wertevorschläge : Frequency ∈ {10, 20, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : $1 \leq$ Frequency ≤ 200
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : Size ∈ {16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Beispiel

```
my_lowpass(Hobject Image, Hobject *Result, int frequency, int size)
{
  Hobject  FFT, Lowpass, FFTConvol;
  fft(Image,&FFT);
  gen_lowpass(&Lowpass,frequency,size);
  convol_fft(FFT,Lowpass,&FFTConvol);
  clear_obj(Lowpass); clear_obj(FFT);
  fft_image_inv(FFTConvol,Result);
  clear_obj(FFTConvol);
}
```

Ergebnis

Sind die Parameter korrekt, dann liefert gen_lowpass den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_lowpass ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

convol_fft

Alternativen

gen_circle, paint_region

Siehe auch

gen_highpass, gen_bandpass, gen_bandfilter

Modul

Image filters

gen_sin_bandpass (: ImageFilter : Frequency, Size :)

Erzeugen eines Bandpaßfilter in der Form einer Sinusfunktion.

gen_sin_bandpass erzeugt einen rotationsinvarianten Bandpaßfilter, in der Form einer Sinusfunktion. Das Maximum der Sinusfunktion (255) wird durch Frequency (Abstand in Pixeln vom Zentrum) bestimmt. Im Ursprung ist der Filter immer Null und steigt mit der Sinusfunktion bis Frequency an und fällt dann analog ab. Es wird der Wertebereich von 0 bis π der Sinusfunktion genutzt. Alle Punkte außerhalb werden auf Null gesetzt.

Parameter

- ▷ **ImageFilter** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Bandpaßfilter als Bild in Frequenzraum.
- ▷ **Frequency** (input_control) ..... real ⇝ *real*
  Abstand des Filtermaximums vom Mittelpunkt.
  Defaultwert : 20
  Wertevorschläge : Frequency ∈ {0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200}
  Typischer Wertebereich : $0.0 \leq$ Frequency ≤ 1000
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : Size ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_sin_bandpass den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_sin_bandpass ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

fft_image, fft_generic

Mögliche Nachfolgerfunktionen

convol_fft

Alternativen

gen_std_bandpass, gen_bandpass, gen_bandfilter, gen_highpass, gen_lowpass

Siehe auch

fft_image_inv

Modul

Image filters

gen_std_bandpass (: ImageFilter : Frequency, Sigma, Size, Mode :)

Erzeugen eines Bandpaßfilter in der Form einer Gaußmaske oder Sinusfunktion.

gen_std_bandpass erzeugt einen rotationsinvarianten Bandpaßfilter, dessen Bandpaßverhalten und durch zugehörige Parameter Frequency und Sigma beschrieben werden: Frequency legt den Abstand vom Nullpunkt und Sigma die Ausdehnung fest. Bei Mode = **'gauss'** wird eine Gaußfunktion erzeugt. Sigma ist dabei die

Standardabweichung. Bei `Mode` = **'sin'** wird eine Sinusfunktion mit dem Maximum bei `Frequency` und der Ausdehnung `Sigma` berechnet.

Parameter

- ▷ **`ImageFilter`** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Bandpaßfilter als Bild in Frequenzraum.
- ▷ **`Frequency`** (input_control) ..... real ⇝ *real*
  Abstand des Filtermaximums vom Mittelpunkt.
  Defaultwert : 20
  Wertevorschläge : `Frequency` ∈ {0.0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 80, 100, 150, 200}
  Typischer Wertebereich : 0.0 ≤ `Frequency` ≤ 1000
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **`Sigma`** (input_control) ..... real ⇝ *real*
  Bandbreite der Filters (Standardabweichung).
  Defaultwert : 1.0
  Wertevorschläge : `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}
  Typischer Wertebereich : 0.05 ≤ `Sigma` ≤ 100.0
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.1
- ▷ **`Size`** (input_control) ..... integer ⇝ *integer*
  Größe (Kantenlänge) des Bildes (Filters).
  Defaultwert : 512
  Werteliste : `Size` ∈ {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192}
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Art des Filters.
  Defaultwert : 'sin'
  Werteliste : `Mode` ∈ {'sin', 'gauss'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_std_bandpass` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_std_bandpass` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`fft_image`, `fft_generic`

Mögliche Nachfolgerfunktionen

`convol_fft`

Alternativen

`gen_sin_bandpass`, `gen_bandpass`, `gen_bandfilter`, `gen_highpass`, `gen_lowpass`

Siehe auch

`fft_image_inv`

Modul

Image filters

`phase_deg ( ImageComplex : ImagePhase : : )`

Phase eines komplexen Bildes in Grad.

`phase_deg` berechnet die Phase eines komplexen Bildes im Winkelmaß. Die Berechnung erfolgt nach folgender Formel:

$$phase = \frac{180 tan^{-1}(Imaginärteil/Realteil)}{\pi} \ .$$

Parameter

- ▷ **ImageComplex** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild im Frequenzraum.
- ▷ **ImagePhase** (output_object) ..... image(-array) ⇝ *Hobject* : direction
  Phase des Bildes im Winkelmaß.

Beispiel

```
read_image(&Image,"affe");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
phase_deg(FFT,&Phase);
disp_image(Phase,WindowHandle);
```

Ergebnis

Ist der Pixeltyp korrekt, dann liefert `phase_deg` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`phase_deg` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`fft_image`, `fft_generic`

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`phase_rad`

Siehe auch

`fft_image_inv`

Modul

Image filters

phase_rad (ImageComplex : ImagePhase : :)

Phase eines komplexen Bildes im Bogenmaß.

`phase_rad` berechnet die Phase eines komplexen Bildes in Bogenmaß. Die Berechnung erfolgt nach folgender Formel:

$$phase = tan^{-1}(Imaginärteil/Realteil) \ .$$

Parameter

- ▷ **ImageComplex** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild im Frequenzraum.
- ▷ **ImagePhase** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Phase des Bildes in Bogenmaß.

Beispiel

```
read_image(&Image,"affe");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
phase_rad(FFT,&Phase);
disp_image(Phase,WindowHandle);
```

Ergebnis

Ist der Pixeltyp korrekt, dann liefert `phase_rad` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`phase_rad` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`fft_image`, `fft_generic`

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`phase_deg`

Siehe auch

`fft_image_inv`

Modul

Image filters

`power_byte ( Image : PowerByte : : )`

Powerspektrum eines komplexen Bildes.

`power_byte` berechnet aus dem Real- und Imaginärteil der Fouriertransformierten (vgl. `fft_image`) das Powerspektrum, d.h. die Beträge der Fouriertransformierten. Das Ergebnisbild ist vom Typ byte. Die Berechnung erfolgt nach folgender Formel:

$$\sqrt{Realteil^2 + Imaginärteil^2} \quad .$$

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild im Frequenzraum.
- ▷ **`PowerByte`** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Powerspektrum des Bildes.

Beispiel

```
read_image(&Image,"affe");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
power_byte(FFT,&Power);
disp_image(Power,WindowHandle);
```

Ergebnis

Ist der Typ des Bildes korrekt, dann liefert `power_byte` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`power_byte` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`fft_image`, `fft_generic`, `convol_fft`, `convol_gabor`

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

abs_image, convert_image_type, power_real, power_ln

Siehe auch

fft_image

Modul

Image filters

power_ln (Image : ImageResult : :)

Powerspektrum eines komplexen Bildes.

power_ln berechnet aus dem Real- und Imaginärteil der Fouriertransformierten (vgl. fft_image) das Powerspektrum, d.h. die Beträge der Fouriertransformierten. Zusätzlich wird von dem Ergebnis der natürliche Logarithmus berechnet. Das Ergebnisbild ist vom Typ real. Die Berechnung erfolgt nach folgender Formel:

$$\ln(\sqrt{Realteil^2 + Imaginärteil^2}) \ .$$

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild im Frequenzraum.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Powerspektrum des Bildes.

Beispiel

```
read_image(&Image,"monkey");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
power_ln(FFT,&Power);
disp_image(Power,WindowHandle);
```

Ergebnis

Ist der Typ des Bildes korrekt, dann liefert power_ln den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

power_ln ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

fft_image, fft_generic, convol_fft, convol_gabor

Mögliche Nachfolgerfunktionen

disp_image, convert_image_type, scale_image

Alternativen

abs_image, convert_image_type, power_real, power_byte

Siehe auch

fft_image, fft_generic

Modul

Image filters

power_real (Image : ImageResult : :)

Powerspektrum eines komplexen Bildes.

power_real berechnet aus dem Real- und Imaginärteil der Fouriertransformierten (vgl. fft_image) das Powerspektrum, d.h. die Beträge der Fouriertransformierten. Das Ergebnisbild ist vom Typ real. Die Berechnung erfolgt nach folgender Formel:

$$\sqrt{Realteil^2 + Imaginärteil^2} \ .$$

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : complex
  Eingabebild im Frequenzraum.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *Hobject* : real
  Powerspektrum des Bildes.

Beispiel

```
read_image(&Image,"affe");
disp_image(Image,WindowHandle);
fft_image(Image,&FFT);
power_real(FFT,&Power);
disp_image(Power,WindowHandle);
```

Ergebnis

Ist der Typ des Bildes korrekt, dann liefert power_real den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

power_real ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

fft_image, fft_generic, convol_fft, convol_gabor

Mögliche Nachfolgerfunktionen

disp_image, convert_image_type, scale_image

Alternativen

abs_image, convert_image_type, power_byte, power_ln

Siehe auch

fft_image

Modul

Image filters

3.6 Glättung

anisotrope_diff (Image : ImageAniso : Percent, Mode, Iteration, neighborhoodType :)

Kantenerhaltende Glättung durch anisotrope Diffusion.

anisotrope_diff realisiert ein iteratives, anisotropes Glättungsverfahren, das auf den mathematischen Grundlagen der physikalischen Diffusion beruht. In Analogie zum physikalischen Diffusionsprozeß, der den Konzentrationsausgleich zwischen Molekülen in Abhängigkeit vom Dichtegradienten beschreibt, führt der Diffusionsfilter eine Glättung der Grauwerte, abhängig von den lokalen Grauwertgradienten, durch.

Zur iterativen Berechnung des Grauwertes eines Pixels werden die Grauwertdifferenzen zu den vier bzw. acht Nachbarn herangezogen. Diese Grauwertdifferenzen gehen jedoch mit unterschiedlicher Gewichtung in die Berechnung ein, d.h. es handelt sich um einen nichtlinearen Diffusionsprozeß.

Die Gewichtung erfolgt mittels einer Diffusionsfunktion (implementiert wurden zwei verschiedene Funktionen, Mode = 1 bzw. 2), die — abhängig vom Gradienten — dafür sorgt, daß innerhalb homogener Regionen stärker

geglättet wird als über die Regionengrenzen hinweg. Die Kanten bleiben deswegen scharf erhalten. Die Diffusionsfunktion wird mittels einer Histogrammanalyse im Gradientenbild (nach Canny) auf das Rauschverhältnis des Bildes hin abgestimmt. Ein hoher Wert für `Percent` erhöht dabei die glättende Wirkung, verwischt aber auch Kanten etwas mehr (typisch sind Werte von 80 - 90 Prozent).

Der Parameter `Iteration` legt die Zahl der Iterationen fest (typisch 3–7).

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu glättendes Bild.
- ▷ **`ImageAniso`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Geglättetes Bild.
- ▷ **`Percent`** (input_control) ..... integer ⇝ *integer*
  Für Histogrammanalyse. Größere Werte erhöhen die glättende Wirkung, typisch: 80 - 90.
  Defaultwert : 80
  Wertevorschläge : `Percent` ∈ {65, 70, 75, 80, 85, 90}
  Typischer Wertebereich : 50 ≤ `Percent` ≤ 100
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5
- ▷ **`Mode`** (input_control) ..... integer ⇝ *integer*
  Selection of diffusion function.
  Defaultwert : 1
  Werteliste : `Mode` ∈ {1, 2}
- ▷ **`Iteration`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen, typische Werte: 3 - 7.
  Defaultwert : 5
  Wertevorschläge : `Iteration` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Typischer Wertebereich : 1 ≤ `Iteration` ≤ 30
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`neighborhoodType`** (input_control) ..... integer ⇝ *integer*
  Gewünschter Nachbarschaftstyp.
  Defaultwert : 8
  Werteliste : `neighborhoodType` ∈ {4, 8}

Beispiel

```
read_image(Image,'fabrik')
anisotrope_diff(Image,Aniso,80,1,5,8)
sub_image(Image,Aniso,Sub,2.0,127)
disp_image(Sub,WindowHandle).
```

Komplexität

Pro Bildpunkt: $O(Iterations * 18)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `anisotrope_diff` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`anisotrope_diff` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`read_image`, `grab_image`

Mögliche Nachfolgerfunktionen

`regiongrowing`, `threshold`, `sub_image`, `dyn_threshold`, `auto_threshold`

Alternativen

`sigma_image`, `rank_image`

Siehe auch

smooth_image, gauss_image, sigma_image, rank_image, eliminate_min_max

Literatur

P. Perona, J. Malik: “Scale-space and edge detection using anisotropic diffusion”, IEEE transaction on pattern analysis and machine intelligence, Vol. 12, No. 7, July 1990.

Modul

Image filters

eliminate_min_max (Image : filteredImage : MaskWidth, MaskHeight, Gap, Mode :)

Ortsraumglättung zur Entfernung von Rauschspitzen.

eliminate_min_max filtert Rauschspitzen aus einem verrauschten Bild. Hierfür wird das Bild punktweise im Ortsraum verarbeitet. Um Kanten und Linien nicht allzusehr zu glätten, werden nur diejenigen Bildpunkte ersetzt, die ein lokales Minimum bzw. Maximum darstellen. Der Gedanke hierbei ist, daß bei Kanten oder Linien noch mindestens ein weiterer Punkt innerhalb der lokalen Umgebung existiert, der ungefähr denselben Grauwert besitzt wie der betrachtete. Dadurch bleibt die Bedingung eines lokalen Minimums/Maximums unerfüllt, der Punkt wird nicht ersetzt und die Kante/Linie nicht geglättet. Um die Strenge der Ersetzungsbedingung beeinflussen zu können, kann mit Gap angegeben werden, um wieviel kleiner bzw. größer das lokale Minimum/Maximum gegenüber den anderen Grauwerten mindestens sein muß. eliminate_min_max arbeitet also nach folgendem Schema: Enthalte $U(x,y)$ alle Grauwerte der $N \times M$ großen, rechteckigen lokalen Umgebung des Punktes (x,y), außer dem Grauwert des Punktes (x,y) selbst;

- Falls $Grauwert(x,y) \geq$ Gap $+ maximum(U(x,y)) \rightarrow Ersetzung$;
- Falls $Grauwert(x,y) +$ Gap $\leq minimum(U(x,y)) \rightarrow Ersetzung$;
- sonst $Grauwert(x,y)$ gleichbelassen;

Mit welchem neuen Wert ein zu ersetzender Punkt belegt wird, ist durch Mode zu steuern:

Mode $= 1 \rightarrow$ Ersetzung eines lokalen Maximums durch das nächstkleinere lokale Maximum und Ersetzung eines lokalen Minimums durch das nächstgrößere lokale Minimum

Mode $= 2 \rightarrow$ Ersetzung durch das (ungewichtete) arithmetische Mittel aller Punkte innerhalb der lokalen Umgebung (inkl. des betrachteten Punktes)

Mode $= 3 \rightarrow$ Ersetzung durch den Median-Wert aller Punkte der lokalen Umgebung (inkl. des betrachteten Punktes) (auch bei allen anderen Werten von Mode außer 1,2)

Die Breite und Höhe der (rechteckigen) lokalen Punktumgebung wird durch MaskWidth und MaskHeight festgelegt. Bemerkung zur Randbehandlung: Punkte jenseits des Bildrands werden nicht in die Betrachtung einbezogen. Sei z.B. eine 3×3-Maske gegeben, so reduziert sich die lokale Umgebung des Punktes $(0,0)$ auf die Werte von $(1,0), (0,1)$ und $(1,1)$.

Achtung

Der Filter ist nur für Bilder vom Typ ’byte’ implementiert. Werden für MaskWidth und MaskHeight gerade statt ungerader Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt). Die Filtermaskenbreite bzw. -höhe darf die Bildbreite bzw. -höhe nicht überschreiten.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image ⇝ *Hobject* : byte
Bild, das gefiltert werden soll.

▷ **filteredImage** (output_object) ..... (multichannel-)image ⇝ *Hobject* : byte
Geglättetes Bild.

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der rechteckigen Filtermaske.
Defaultwert : 3
Wertevorschläge : MaskWidth ∈ {3, 5, 7, 9}
Typischer Wertebereich : 3 ≤ MaskWidth ≤ width(Image)
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskWidth)

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der rechteckigen Filtermaske.
Defaultwert : 3
Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9}
Typischer Wertebereich : 3 ≤ MaskHeight ≤ width(Image)
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskWidth)

▷ **Gap** (input_control) . . . . . number ⇝ *real*
Mindestabstand vom lokalen Minimum/Maximum zu allen anderen Grauwerten der lokalen Punktumgebung.
Defaultwert : 1.0
Wertevorschläge : Gap ∈ {1.0, 2.0, 5.0, 10.0}

▷ **Mode** (input_control) . . . . . integer ⇝ *integer*
Ersetzungsvorschrift für zu ersetzende Grauwerte (1=nächstes Minimum/Maximum, 2=Average, sonst Median).
Defaultwert : 3
Werteliste : Mode ∈ {1, 2, 3}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert eliminate_min_max den Wert 2 (H_MSG_TRUE). Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Parallelisierungsinformation

eliminate_min_max ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

wiener_filter, wiener_filter_ni

Siehe auch

mean_sp, mean_image, median_image, median_weighted, gauss_image, smooth_image

Literatur

M. Imme:“A Noise Peak Elimination Filter”; S. 204-211 in CVGIP Graphical Models and Image Processing, Vol. 53, No. 2, March 1991

M. Lückenhaus:“Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse”; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Modul

Image filters

eliminate_sp (Image : ImageFillSP : MaskWidth, MaskHeight, MinThresh, MaxThresh :)

Ersetzen der Werte außerhalb der Schwellen durch den Mittelwert.

eliminate_sp ersetzt alle Grauwerte außerhalb des angegebenen Grauwertintervalls (MinThresh bis MaxThresh) durch den Mittelwert der Nachbarn. Es werden für die Mittellung aber nur die Nachbarpunkte herangezogen, die ebenfalls innerhalb des Grauwertintervalls liegen. Ist in der Umgebung kein solcher Punkt enthalten, wird der Originalgrauwert verwendet. Die Grauwerte im Eingabebild, die innerhalb des Grauwertintervalls liegen, werden ebenfalls unverändert übernommen.

Achtung

Werden für MaskHeight oder MaskWidth gerade statt ungerade Werte übergeben, verwendet die Routine an

ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **ImageFillSP** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Geglättetes Bild.
- ▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 3
  Wertevorschläge : `MaskWidth` ∈ {3, 5, 7, 9, 11}
  Typischer Wertebereich : 3 ≤ `MaskWidth` ≤ 512 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(`MaskWidth`)
- ▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 3
  Wertevorschläge : `MaskHeight` ∈ {3, 5, 7, 9, 11}
  Typischer Wertebereich : 3 ≤ `MaskHeight` ≤ 512 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(`MaskHeight`)
- ▷ **MinThresh** (input_control) ..... integer ⇝ *integer*
  Mindestgrauwert.
  Defaultwert : 1
  Wertevorschläge : `MinThresh` ∈ {1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
  Typischer Wertebereich : 0 ≤ `MinThresh` ≤ 254 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **MaxThresh** (input_control) ..... integer ⇝ *integer*
  Maximalgrauwert.
  Defaultwert : 254
  Wertevorschläge : `MaxThresh` ∈ {5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254}
  Typischer Wertebereich : 1 ≤ `MaxThresh` ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `MinThresh` ≤ `MaxThresh`

Beispiel

```
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
eliminate_sp(Image,ImageMeansp,3,3,101,201)
disp_image(ImageMeansp,WindowHandle).
```

Parallelisierungsinformation

`eliminate_sp` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`mean_sp`, `mean_image`, `median_image`, `eliminate_min_max`

Siehe auch

`gauss_image`, `smooth_image`, `anisotrope_diff`, `sigma_image`, `eliminate_min_max`

Modul

Image filters

fill_interlace (ImageCamera : ImageFilled : Mode :)

Interpolation von 2 Video-Halbbildern.

fill_interlace berechnet aus einem Videobild, das aus zwei Halbbildern zusammengesetzt ist, ein interpoliertes Vollbild bzw. entfernt gerade oder ungerade Bildzeilen. Wird ein Bild mit einer Videokamera aufgenommen, so besteht dies aus zwei Halbbildern, die mit einem zeitlichen Abstand aufgenommen wurden, aber in der digitalen Form in einem gemeinsamen Bild abgelegt sind. Hierdurch können starke Fehler bei der weiteren Verarbeitung entstehen. Um diese Fehler etwas zu mildern, wird das Videobild modifiziert. Dabei wird entweder jede zweite Zeile neu berechnet oder entfernt. Der Parameter Mode gibt an, ob dies für gerade ('even' bzw. 'rmeven') oder ungerade ('odd' bzw. 'rmodd') Zeilennummern gemacht werden soll. In den Modi 'even' bzw. 'odd' werden die Grauwerte in den erzeugten Zeilen als Mittelwert aus dem direkten Nachbarn oberhalb bzw. unterhalb des aktuellen Punktes berechnet. In den Modi 'rmeven' bzw. 'rmodd' werden gerade bzw. ungerade Zeilen entfernt (dementsprechend hat dann das Ausgabebild nur noch die halbe Höhe des Eingabebildes). Hat Mode den Wert 'switch', dann werden gerade und ungerade Zeilen vertauscht.

Parameter

- ▷ **ImageCamera** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Graubild, bestehend aus zwei Halbbildern.
- ▷ **ImageFilled** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Vollbild mit interpolierten bzw. entfernten Zeilen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Angabe, ob die geraden oder ungeraden Zeilen ersetzt bzw. entfernt werden sollen.
  Defaultwert : 'odd'
  Werteliste : Mode ∈ {'odd', 'even', 'rmodd', 'rmeven', 'switch'}

Beispiel

```
read_image(Image,'video_bild')
fill_interlace(Image,New,'odd')
sobel_amp(New,Sobel,'sum_abs',3).
```

Komplexität

Pro Bildpunkt: $O(2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fill_interlace den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fill_interlace ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

read_image, grab_image

Mögliche Nachfolgerfunktionen

sobel_amp, edges_image, regiongrowing, diff_of_gauss, threshold, dyn_threshold, auto_threshold, mean_image, gauss_image, anisotrope_diff, sigma_image, median_image

Siehe auch

median_image, gauss_image, crop_part

Modul

Image filters

gauss_image (Image : ImageGauss : Size :)

Glättung mit diskreten Gaussfunktionen.

gauss_image glättet Bilder mittels der diskreten Gaussfunktion. Die glättende Wirkung erhöht sich dabei mit zunehmender Filtergröße. Es werden folgende Filtergrößen (Size) unterstützt (in Klammer steht der sigma-Wert der Gaussfunktion):

3	(0.81)
5	(0.93)
7	(1.50)
9	(2.00)
11	(2.45)

Zur Randbehandlung werden die Grauwerte der Bilder an den Bildrändern gespiegelt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int4 / int2
  Zu glättendes Bild.
- ▷ **ImageGauss** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int4 / int4
  Gefiltertes Bild.
- ▷ **Size** (input_control) ..... integer ⇝ *integer*
  Gewünschte Filtergröße.
  Defaultwert : 5
  Werteliste : Size ∈ {3, 5, 7, 9, 11}

Beispiel

```
gauss_image(Input,Gauss,7)
regiongrowing(Gauss,Segments,7,7,5,100).
```

Komplexität

Pro Bildpunkt: $O(Size * 2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gauss_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gauss_image ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

read_image, grab_image

Mögliche Nachfolgerfunktionen

regiongrowing, threshold, sub_image, dyn_threshold, auto_threshold

Alternativen

smooth_image, derivate_gauss

Siehe auch

mean_image, anisotrope_diff, sigma_image, gen_lowpass

Modul

Image filters

info_smooth (: : Filter, Alpha : Size, Coeffs)

Informationen zum Glättungsfilter smooth_image.

info_smooth liefert eine Abschätzung des Einzugsgebiets der in der Routine smooth_image verwendeten Glättungsfilter. Dazu werden die zugrundeliegenden kontinuierlichen Impulsantworten der Filter abgetastet, bis ein Filterkoeffizient kleiner als fünf Prozent des Maximalkoeffizienten (im Ursprung) ist. Alpha ist der Filterparameter (siehe smooth_image). Derzeit werden vier Filter unterstützt (Parameter Filter):

'deriche1', 'deriche2', 'shen' und 'gauss'.

Der Gaussfilter wurde dabei konventionell mittels Filtermasken implementiert (bei den drei anderen handelt es sich um rekursive Filter). Im Falle des Gaussfilters werden daher zusätzlich zur Filtergröße auch die Filterkoeffizienten (der eindimensionalen Impulsantwort $f(n)$ mit $n \geq 0$) in `Coeffs` zurückgeliefert.

Parameter

- ▷ **`Filter`** (input_control) ..... string ⇝ *string*
  Name des gewünschten Filters.
  Defaultwert : 'deriche2'
  Werteliste : `Filter` ∈ {'deriche1', 'deriche2', 'shen', 'gauss'}
- ▷ **`Alpha`** (input_control) ..... real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung (bei 'gauss' umgekehrt).
  Defaultwert : 0.5
  Wertevorschläge : `Alpha` ∈ {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich : 0.01 ≤ `Alpha` ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Alpha` > `0.0`
- ▷ **`Size`** (output_control) ..... integer ⇝ *integer*
  Einzugsgebiet des Filters ist etwa Size x Size Pixel groß.
- ▷ **`Coeffs`** (output_control) ..... integer-array ⇝ *integer*
  Falls Gaussfilter: Koeffizienten der „positiven" Hälfte der 1D-Impulsantwort.

Beispiel

```
info_smooth('deriche2',0.5,Size,Coeffs)
smooth_image(Input,Smooth,'deriche2',7).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `info_smooth` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`info_smooth` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_image`

Mögliche Nachfolgerfunktionen

`smooth_image`

Siehe auch

`smooth_image`

Modul

Image filters

`mean_image` (Image : ImageMean : MaskWidth, MaskHeight :)

Glättung durch Mittelwertbildung.

`mean_image` führt eine lineare Glättung mit den Grauwerten aller Eingabebilder (`Image`) durch. Die Filtermatrix besteht aus Einsen (gleich gewichtet) und hat die Größe `MaskHeight` × `MaskWidth`. Das Ergebnis der Konvolution wird durch `MaskHeight` × `MaskWidth` dividiert. Als Randbehandlung werden die Grauwerte an den Bildrändern gespiegelt.

Achtung

Werden für `MaskHeight` oder `MaskWidth` gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / int4 / real / dvf
  Zu glättendes Bild.
- ▷ **ImageMean** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / int4 / real / dvf
  Geglättetes Bild.
- ▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 9
  Wertevorschläge : MaskWidth ∈ {3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
  Typischer Wertebereich : 3 ≤ MaskWidth ≤ 501
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(MaskWidth)
- ▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 9
  Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
  Typischer Wertebereich : 3 ≤ MaskHeight ≤ 501
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(MaskHeight)

Beispiel

```
read_image(Image,'fabrik')
mean_image(Image,Mean,3,3)
disp_image(Mean,WindowHandle).
```

Komplexität

Pro Bildpunkt: O(15).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert mean_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

mean_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

dyn_threshold, regiongrowing

Alternativen

gauss_image, smooth_image

Siehe auch

anisotrope_diff, sigma_image, convol_image, gen_lowpass

Modul

Image filters

mean_n (Image : ImageMean : :)

Mittelung der Grauwerte zwischen mehreren Kanälen.

mean_n bildet den pixelweisen Mittelwert aller Kanäle. Es wird für jeden Koordinatenpunkt die Summe alle Grauwerte an diese Koordinate berechnet. Das Ergebnisbild wird mit dem Mittelwert der Grauwerte (Summe geteilt durch Anzahl der Kanäle) belegt. Das Ausgabebild hat einen Kanal.

Parameter

- ▷ **Image** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte / real
  Mehrkanaliges Graubild.
- ▷ **ImageMean** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject* : byte / real
  Ergebnis der Mittelung.

Beispiel

```
compose3(Channel1,Channel2,Channel3,&MultiChannel);
mean_n(MultiChannel,&Mean);
```

Parallelisierungsinformation

mean_n ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

compose2, compose3, compose4, add_channels

Mögliche Nachfolgerfunktionen

disp_image

Siehe auch

count_channels

Modul

Image filters

mean_sp (Image : ImageSPMean : MaskWidth, MaskHeight, MinThresh, MaxThresh :)

Unterdrückung von Salz- und Pfeffer-Rauschen.

mean_sp führt eine Glättung durch Mittelwertbildung durch. Dabei gehen nur die Grauwerte in die Mittellung mit ein, die in dem Intervall von MinThresh bis MaxThresh liegen. Zu helle oder zu dunkle Grauwerte werden bei der Summation ignoriert. Liegt kein Grauwert bei der Summation innerhalb des vorgegebenen Intervalls, so wird der Originalgrauwert übernommen. Setzt man die Schwellen auf **0** bzw. **255** so verhält sich mean_sp, abgesehen von der Laufzeit, wie mean_image.

mean_sp wird verwendet um extreme Grauwerte (Salz- und Pfeffer-Rauschen = weiße und schwarze Punkte) zu unterdrücken.

Achtung

Werden für MaskHeight oder MaskWidth gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **ImageSPMean** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Geglättetes Bild.
- ▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 3
  Wertevorschläge : MaskWidth ∈ {3, 5, 7, 9, 11}
  Typischer Wertebereich : 3 ≤ MaskWidth ≤ 512 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(MaskWidth)

▷ **MaskHeight** (input_control) . . . . . extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert : 3
Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9, 11}
Typischer Wertebereich : 3 ≤ MaskHeight ≤ 512 (lin)
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskHeight)

▷ **MinThresh** (input_control) . . . . . integer ⇝ *integer*
Mindestgrauwert.
Defaultwert : 1
Wertevorschläge : MinThresh ∈ {1, 5, 7, 9, 11, 15, 23, 31, 43, 61, 101}
Typischer Wertebereich : 0 ≤ MinThresh ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **MaxThresh** (input_control) . . . . . integer ⇝ *integer*
Maximalgrauwert.
Defaultwert : 254
Wertevorschläge : MaxThresh ∈ {5, 7, 9, 11, 15, 23, 31, 43, 61, 101, 200, 230, 250, 254}
Typischer Wertebereich : 0 ≤ MaxThresh ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : MinThresh ≤ MaxThresh

Beispiel

```
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
mean_sp(Image,ImageMeansp,3,3,101,201)
disp_image(ImageMeansp,WindowHandle).
```

Parallelisierungsinformation

mean_sp ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

mean_image, median_image, median_separate, eliminate_min_max

Siehe auch

anisotrope_diff, sigma_image, gauss_image, smooth_image, eliminate_min_max

Modul

Image filters

median_image (Image : ImageMedian : MaskType, Radius, Margin :)

Medianfilterung mit verschiedenen Rangmasken.

median_image führt eine nichtlineare Glättung der Grauwerte aller Eingabebilder (Image) durch. Die Verschiebungsmaske (MaskType) wird in Form eines Objektes (genauer: dessen Region) übergeben. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (Margin) gewählt werden:

0...255	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
-1	Fortsetzung der Randpunkte.
-2	zyklische Fortsetzung der Bildränder.
-3	Spiegelung der Bildpunkte an den Bildrändern.

Die angebene Maske (= Region des Maskenbildes) wird so über das zu filternde Bild geschoben, daß der Schwerpunkt der Maske alle Bildpunkte der Objekte einmal berührt. Für jeden solchen Bildpunkt werden alle Nachbarpunkte, die von der Maske überdeckt werden, bzgl. ihrer Grauwerte aufsteigend sortiert. Jede solche sortierte Sequenz von Grauwerten enthält somit genau so viele Grauwerte, wie die Maske Bildpunkte hat. Aus diesen Sequenzen wird der Median ausgewählt und als Ergebnisgrauwert bei dem jeweiligen Ausgabebild eingetragen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu filterndes Bild.
- ▷ **ImageMedian** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Mediangefiltertes Bild.
- ▷ **MaskType** (input_control) ..... string ⇝ *string*
  Art der Median-Maske.
  Defaultwert : 'circle'
  Werteliste : MaskType ∈ {'circle', 'rectangle'}
- ▷ **Radius** (input_control) ..... integer ⇝ *integer*
  Radius der Median-Maske.
  Defaultwert : 1
  Wertevorschläge : Radius ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59}
  Typischer Wertebereich : 1 ≤ Radius ≤ 101
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **Margin** (input_control) ..... integer ⇝ *integer*
  Randbehandlung: 0...255 (konstant), -1 (Randpunkte fortgesetzt), -2 (zyklische Fortsetzung), -3 (Spiegelung).
  Defaultwert : -3
  Typischer Wertebereich : -3 ≤ Margin ≤ 255

Beispiel

```
read_image(Image,'fabrik')
median_image(Image,Median,'circle',3,-1)
disp_image(MedianWeighted,WindowHandle).
```

Komplexität

Pro Bildpunkt: O($\sqrt{Fläche(\texttt{MaskType})} * 5$).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert median_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

median_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

read_image

Mögliche Nachfolgerfunktionen

threshold, dyn_threshold, regiongrowing

Alternativen

rank_image

Siehe auch

gen_circle, gen_rectangle1, gray_erosion_rect, gray_dilation_rect

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 318-319

Modul

Image filters

median_separate (Image : ImageSMedian : MaskWidth, MaskHeight, Margin :)

Separierte Medianfilterung mit Rechteckmasken.

median_separate führt eine abgewandelte Variante der Median-Filterung durch: Es werden zunächst zwei Hilfsbilder erzeugt. Das erste entsteht aus einer Medianfilterung mit einer horizontalen, ein Pixel hohen und MaskWidth breiten Maske und anschließender Filterung mit einer MaskHeight hohen Maske. Das zweite Hilfsbild entsteht durch Filterung mit den gleichen Masken, wobei aber die Reihenfolge der Anwendung umgekehrt wird. Zuerst wird die vertikale, dann die horizontale Maske angewendet. Das Ausgabebild ergibt sich durch pixelweise Mittelung der beiden Hilfsbilder.

median_separate ist deutlich schneller als der normale median_image, da beide Masken ein Pixel breit sind und somit eine sehr effiziente Verarbeitung möglich ist. Die Laufzeit ist praktisch $unabhängig$ von der Größe der Maske. median_separate läßt sich beispielweise gut nach Texturfiltern einsetzen, da dort große Masken benötigt werden.

Zur Verbesserung der Glättung kann der Filter auch mehrfach hintereinander angewandt werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu filterndes Bild.
- ▷ **ImageSMedian** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Mediangefiltertes Bild.
- ▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
  Breite der Rangmaske.
  Defaultwert : 25
  Wertevorschläge : MaskWidth ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151}
  Typischer Wertebereich : 1 ≤ MaskWidth ≤ 401
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
- ▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Rangmaske.
  Defaultwert : 25
  Wertevorschläge : MaskHeight ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 27, 43, 51, 67, 91, 121, 151}
  Typischer Wertebereich : 1 ≤ MaskHeight ≤ 401
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
- ▷ **Margin** (input_control) ..... integer ⇝ *integer*
  Margin control: 0...255 (constant), -1 (edge pixels continued), -2 (cyclic continuation), -3 (reflection).
  Defaultwert : -3
  Typischer Wertebereich : -3 ≤ Margin ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Beispiel

```
read_image(Image,'fabrik')
median_separate(Image,MedianSeparate,5,5,3)
disp_image(MedianSeparate,WindowHandle).
```

Komplexität

Pro Bildpunkt: O(40).

Parallelisierungsinformation

median_separate ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

texture_laws, sobel_amp, deviation_image

Mögliche Nachfolgerfunktionen

learn_ndim_norm, learn_ndim_box, median_separate, regiongrowing, auto_threshold

Alternativen

median_image

Siehe auch

rank_image

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 319

Modul

Image filters

median_weighted (Image : ImageWMedian : MaskType, MaskSize :)

Gewichtete Medianfilterung mit verschiedenen Rangmasken.

median_weighted berechnet den Median der Grauwerte innerhalb einer lokalen Umgebung. Im Gegensatz zu median_image, bei dem alle Grauwerte innerhalb der Umgebung genau einmal eingehen, werden bei median_weighted die Grauwerte, abhängig von ihrer Position mehrfach gewichtet. Dabei wird ein Grauwert, entsprechend seiner Gewichtung mehrfach in das zu sortierende Feld aufgenommen. Es stehen folgende Masken zur Verfügung:

’gauss’ (MaskSize = 3)

$$\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

’inner’ (MaskSize = 3)

$$\begin{matrix} 1 & 1 & 1 \\ 1 & 3 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Der median_weighted ist, daß im Gegensatz zu median_image Grauwertecken erhalten bleiben.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Zu filterndes Bild.
- ▷ **ImageWMedian** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Mediangefiltertes Bild.
- ▷ **MaskType** (input_control) ..... string ⇝ *string*
  Art der Median-Maske.
  Defaultwert : ’inner’
  Werteliste : MaskType ∈ {’inner’, ’gauss’}
- ▷ **MaskSize** (input_control) ..... integer ⇝ *integer*
  Maskengröße.
  Defaultwert : 3
  Werteliste : MaskSize ∈ {3}

Beispiel

```
read_image(Image,’fabrik’)
median_weighted(Image,MedianWeighted,’gauss’,3)
disp_image(MedianWeighted,WindowHandle).
```

Komplexität

Pro Bildpunkt: O(log($Fläche$(MaskType))$Fläche$(MaskType)).

Parallelisierungsinformation

median_weighted ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

read_image

Mögliche Nachfolgerfunktionen

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`median_image`, `trimmed_mean`, `sigma_image`

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 319

Modul

Image filters

`midrange_image ( Image, Mask : ImageMidrange : Margin : )`

Mittelwert aus Maximum und Minimum innerhalb einer beliebigen Maske.

`midrange_image` bildet den Mittelwert aus Maximum und Minimum innerhalb der angegebenen Maske im gesamten Bild. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (Margin) gewählt werden:

0...255	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
-1	Fortsetzung der Randpunkte.
-2	zyklische Fortsetzung der Bildränder.
-3	Spiegelung der Bildpunkte an den Bildrändern.

Die angegebene Maske (= Region des Maskenobjekts) wird so über die zu filternden Bilder geschoben, daß der Schwerpunkt der Maske alle Bildpunkte einmal berührt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu filterndes Bild.
- ▷ **`Mask`** (input_object) ..... region ⇝ *Hobject* : byte
  Region die als Filtermaske dient.
- ▷ **`ImageMidrange`** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Gefiltertes Ergebnisbild.
- ▷ **`Margin`** (input_control) ..... integer ⇝ *integer*
  Randbehandlung: 0...255 (konstant), -1 (Randpunkte fortgesetzt), -2 (zyklische Fortsetzung), -3 (Spiegelung).
  Defaultwert : -3
  Typischer Wertebereich : -3 ≤ `Margin` ≤ 255

Beispiel

```
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
midrange_image(Image,Region,Midrange,-3)
disp_image(Midrange,WindowHandle).
```

Komplexität

Pro Bildpunkt: O($\sqrt{Fläche(\texttt{Mask})} * 5$).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `midrange_image` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`midrange_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

`read_image`, `draw_region`, `gen_circle`, `gen_rectangle1`

Mögliche Nachfolgerfunktionen

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`sigma_image`

Siehe auch

`gen_circle`, `gen_rectangle1`, `gray_erosion_rect`, `gray_dilation_rect`, `gray_range_rect`

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 319

Modul

Image filters

`rank_image` (Image, Mask : ImageRank : Rank, Margin :)

Glättung mit einer beliebigen Rangmaske.

`rank_image` führt eine nichtlineare Glättung der Grauwerte aller Eingabebilder (`Image`) durch. Die Verschiebungsmaske `Mask` wird in Form einer Region übergeben. Im Gegensatz zu vielen anderen Filtern kann demnach eine beliebige Form für die Filtermaske gewählt werden; die entsprechende Region kann z.B. mit Operatoren wie `gen_circle` or `draw_region` erzeugt werden. Die Position der Maskenregion hat keinen Einfluss auf das Ergebnis; als Referenzpunkt der Maske wird der Schwerpunkt der Region verwendet.

Die angebene Maske wird so über das zu filternde Bild geschoben, daß der Referenzpunkt der Maske alle Bildpunkte einmal berührt. An jedem solchen Bildpunkt wird ein Histogramm der Grauwerte aller von der Maske überdeckten Bildpunkte berechnet. Durch die Wahl von `Rank` = **1** wird der niedrigste (= dunkelste) Grauwert, der im Histogramm auftritt, selektiert und als resultierender Grauwert im Ausgabebild `ImageRank` eingetragen; wenn `Rank` gleich der Anzahl der Pixel in der Maskenregion, also gleich deren Fläche ist, wird der hellste auftretende Grauwert selektiert. Für diese zwei Werte ist das Verhalten von `rank_image` identisch zu dem der Erosion/Dilatation der Grauwert-Morphologie (`gray_erosion`, `gray_dilation`). Wenn für `Rank` die Hälfte der Maskenpunkte gewählt wird, verhält sich `rank_image` wie der Median-Filter (`median_image`).

`rank_image` kann vielfältig eingesetzt werden, z.B. zur Rauschunterdrückung oder zum Eliminieren von Strukturen einer bestimmten Orientierung (Erzeugen der Maskenregion mittels `gen_rectangle2`). Außerdem ist der Operator unempfindlicher gegenüber Rauschen als die entsprechenden Operatoren der Grauwertmorpholgie; hier empfiehlt es sich, anstatt 1 bzw. der Maskenfläche etwas höhere bzw. tiefere Werte für `Rank` zu wählen.

Bei der Filterung kann zwischen verschiedenen Randbehandlungen (`Margin`) gewählt werden:

0...255	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
-1	Fortsetzung der Randpunkte.
-2	zyklische Fortsetzung der Bildränder.
-3	Spiegelung der Bildpunkte an den Bildrändern.

Parameter

- ▷ **`Image`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte
  Zu filterndes Bild.
- ▷ **`Mask`** (input_object) ..... region ⇝ *Hobject* : byte
  Region die als Filtermaske dient.
- ▷ **`ImageRank`** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Gefiltertes Bild.
- ▷ **`Rank`** (input_control) ..... integer ⇝ *integer*
  Rang des Ausgabegrauwerts in der sortierten Sequenz der Eingabegrauwerte innerhalb der Filtermaske.
  Typischer Wert (Median): Fläche(Mask) / 2.
  Defaultwert : 5
  Wertevorschläge : `Rank` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31}
  Typischer Wertebereich : 1 ≤ `Rank` ≤ 512
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2

▷ **Margin** (input_control) . . . . . integer ⇝ *integer*
Randbehandlung: 0...255 (konstant), -1 (Randpunkte fortgesetzt), -2 (zyklische Fortsetzung), -3 (Spiegelung).
Defaultwert : -3
Typischer Wertebereich : -3 ≤ Margin ≤ 255

Beispiel

```
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
rank_image(Image,Region,ImageRank,5,-3)
disp_image(ImageRank,WindowHandle).
```

Komplexität

Pro Bildpunkt: O($\sqrt{Fläche(\texttt{Mask})} * 5$).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert rank_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

rank_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

read_image, draw_region, gen_circle, gen_rectangle1

Mögliche Nachfolgerfunktionen

threshold, dyn_threshold, regiongrowing

Alternativen

sigma_image

Siehe auch

gen_circle, gen_rectangle1, gray_erosion_rect, gray_dilation_rect

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 318-320

Modul

Image filters

sigma_image (Image : ImageSigma : MaskHeight, MaskWidth, Sigma :)

Nichtlineare Glättung mit dem Sigmafilter.

sigma_image führt eine nichtlineare Glättung der Grauwerte aller Eingabebilder (Image) durch. Dabei werden alle Bildpunkte in einem rechteckigen Fenster (MaskHeight × MaskWidth) untersucht. Alle Punkte des Fensters, die um weniger als Sigma vom dem aktuellen Bildpunkt abweichen, werden zur Berechnung des neuen Bildpunktes verwendet. Dieser ergibt sich als Mittelwert dieser ausgewählten Bildpunkte. Falls alle Differenzen größer als Sigma sind, wird der Grauwert unverändert übernommen.

Achtung

Der Filter ist nur für Bilder vom Typ ’byte’ implementiert. Werden für MaskHeight oder MaskWidth gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
Zu glättendes Bild.

▷ **ImageSigma** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
Geglättetes Bild.

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
Fenstergröße für Mittelung (Anzahl Zeilen).
Defaultwert : 5
Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich : 3 ≤ MaskHeight ≤ 101
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskHeight)

▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
Fenstergröße für Mittelung (Anzahl Spalten).
Defaultwert : 5
Wertevorschläge : MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich : 3 ≤ MaskWidth ≤ 101
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskWidth)

▷ **Sigma** (input_control) ..... integer ⇝ *integer*
Maximale Differenz zum Mittelwert.
Defaultwert : 3
Wertevorschläge : Sigma ∈ {3, 5, 7, 9, 11, 20, 30, 50}
Typischer Wertebereich : 0 ≤ Sigma ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2

Beispiel

```
read_image(Image,'fabrik')
sigma_image(Image,ImageSigma,5,5,3)
disp_image(ImageSigma,WindowHandle).
```

Komplexität

Pro Bildpunkt: O(MaskHeight× MaskWidth).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sigma_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

sigma_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

read_image

Mögliche Nachfolgerfunktionen

threshold, dyn_threshold, regiongrowing

Alternativen

anisotrope_diff, rank_image

Siehe auch

smooth_image, gauss_image, mean_image

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 325

Modul

Image filters

smooth_image (Image : ImageSmooth : Filter, Alpha :)

Glättung mit rekursiven Filtern.

smooth_image dient zur Glättung von Graustufenbildern mittels rekursiver Filter (nach Deriche und Shen) bzw. mittels des konventionell implementierten Gaussfilters. Konkret lassen sich über den Parameter Filter folgende vier Filter auswählen:

'deriche1', 'deriche2', 'shen' und 'gauss'.

Die „Filterbreite“ (i.e. das Einzugsgebiet und damit die glättende Wirkung der Filter) ist frei wählbar. Sie nimmt bei den Deriche- und dem Shen-Filter mit wachsendem Filterparameter Alpha ab, beim Gaussfilter hingegen zu (Alpha ist in diesem Fall die Standardabweichung der Gaussfunktion). Eine Abschätzung des Einzugsgebiets der Filter für konkrete Werte von Alpha liefert die Routine info_smooth.

Nicht-rekursive Filter, wie hier der Gaussfilter, werden häufig mittels Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit natürlich mit wachsender Filterbreite. Die Laufzeit der rekursiven Filter ist hingegen praktisch konstant. Lediglich die Randbehandlung (Spiegelung der Randpunkte) ist bei „breiteren“ Filter etwas aufwendiger. Das Einzugsgebiet der Deriche- bzw. des Shen-Filters ist also fast ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierende Laufzeitvorteil gegenüber dem Gaussfilter nimmt naturgemäß mit wachsender „Filterbreite“ zu. Allerdings ist nur der Gaussfilter isotrop, alle anderen Filter sind anisotrop (wobei der 'deriche2'-Filter nur schwach richtungs-sensitiv ist). Eine vergleichbare glättende Wirkung der Filter erhält man bei folgender Wahl von Alpha:

$$\begin{aligned} \texttt{Alpha}('deriche2') &= \frac{\texttt{Alpha}('deriche1')}{2} \\ \texttt{Alpha}('shen') &= \frac{\texttt{Alpha}('deriche1')}{2} \\ \texttt{Alpha}('gauss') &= \frac{1.77}{\texttt{Alpha}('deriche1')} \end{aligned}$$

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu glättende Bilder.
- ▷ **ImageSmooth** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Geglättete Bilder.
- ▷ **Filter** (input_control) ..... string ⇝ *string*
  Gewünschter Filter.
  Defaultwert : 'deriche2'
  Werteliste : Filter ∈ {'deriche1', 'deriche2', 'shen', 'gauss'}
- ▷ **Alpha** (input_control) ..... real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung (bei **'gauss'** umgekehrt).
  Defaultwert : 0.5
  Wertevorschläge : Alpha ∈ {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}
  Typischer Wertebereich : 0.01 ≤ Alpha ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Alpha > 0

Beispiel

```
info_smooth('deriche2',0.5,Size,Coeffs)
smooth_image(Input,Smooth,'deriche2',7)
```

Ergebnis

Sind die Parameterwerte korrekt liefert smooth_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

smooth_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

`read_image`

Mögliche Nachfolgerfunktionen

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`gauss_image`, `mean_image`, `derivate_gauss`

Siehe auch

`info_smooth`, `median_image`, `sigma_image`, `anisotrope_diff`

Literatur

R.Deriche: “Fast Algorithms for Low-Level Vision”; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; S. 78-87; 1990.

Modul

Image filters

`trimmed_mean ( Image, Mask : ImageTMean : Number, Margin : )`

Glättung mit einer beliebigen Rangmaske.

`trimmed_mean` führt eine nichtlineare Glättung der Grauwerte aller Eingabebilder (`Image`) durch. Die Verschiebungsmaske (`Mask`) wird in Form einer Region übergeben. Berechnet wird der Mittelwert von `Number` Grauwert, die in der Nähe des Medians liegen. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (`Margin`) gewählt werden:

0...255	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
-1	Fortsetzung der Randpunkte.
-2	zyklische Fortsetzung der Bildränder.
-3	Spiegelung der Bildpunkte an den Bildrändern.

Die angebene Maske (= Region des Maskenbildes) wird so über die zu filternden Bilder geschoben, daß der Schwerpunkt der Maske alle Bildpunkte einmal berührt. Für jeden solchen Bildpunkt werden alle Nachbarpunkte, die von der Maske überdeckt werden, bzgl. ihrer Grauwerte aufsteigend sortiert. Jede solche sortierte Sequenz von Grauwerten enthält somit genau so viele Grauwerte, wie die Maske Bildpunkte hat. Sei F die Fläche der Maske, dann wird aus diesen Sequenzen der Mittelwert folgendermassen berechnet: Die ersten (F - Number)/2 Grauwerte werden ignoriert. Dann werden die folgenden `Number` Grauwerte aufsummiert und durch `Number` dividiert. Die verbleibenden (F - `Number`)/2 Grauwerte werden wiederum ignoriert.

Parameter

- ▷ **`Image`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte
  Zu filterndes Bild.
- ▷ **`Mask`** (input_object) ..... region ⇝ *Hobject*
  Bild, dessen Region als Filtermaske dient.
- ▷ **`ImageTMean`** (output_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte
  Gefiltertes Ergebnisbild.
- ▷ **`Number`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Punkte über die gemittelt wird. Typischer Wert: Fläche(Mask) / 2.
  Defaultwert : 5
  Wertevorschläge : `Number` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31}
  Typischer Wertebereich : 1 ≤ `Number` ≤ 401
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **`Margin`** (input_control) ..... integer ⇝ *integer*
  Randbehandlung: 0...255 (konstant), -1 (Randpunkte fortgesetzt), -2 (zyklische Fortsetzung), -3 (Spiegelung).
  Defaultwert : -3
  Typischer Wertebereich : -3 ≤ `Margin` ≤ 255

Beispiel

```
read_image(Image,'fabrik')
draw_region(Region,WindowHandle)
trimmed_mean(Image,Region,TrimmedMean,5,-3)
disp_image(TrimmedMean,WindowHandle).
```

Komplexität

Pro Bildpunkt: O($\sqrt{Fläche(\texttt{Mask})} * 5$).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `trimmed_mean` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`trimmed_mean` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

`read_image`, `draw_region`, `gen_circle`, `gen_rectangle1`

Mögliche Nachfolgerfunktionen

`threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`sigma_image`, `median_weighted`, `median_image`

Siehe auch

`gen_circle`, `gen_rectangle1`, `gray_erosion_rect`, `gray_dilation_rect`

Literatur

R. Haralick, L. Shapiro; “Computer and Robot Vision”; Addison-Wesley, 1992, Seite 320

Modul

Image filters

3.7 Kanten

`close_edges` (Edges, EdgeImage : RegionResult : MinAmplitude :)

Schließen von Kanten gemäß Amplitudenbild.

`close_edges` schließt die von einem Kantendetektor gelieferten Kanten zu Konturen. Dabei wird iterativ von jedem Kantenpunkt aus nach den beiden Nachbarpunkten mit stärkster Amplitude (i.e. größtem Gradienten) gesucht und diese beiden Punkte zur Kantenregion hinzugenommen, falls ihre Amplituden größer oder gleich der übergebenen Mindestamplitude sind. Als Eingabe erwartet die Routine also neben den Kanten (`Edges`) auch ein Amplitudenbild (`EdgeImage`), wie es beispielsweise die Routinen `edges_image` oder `sobel_amp` als Ergebnis liefern. Nicht berücksichtigt wird hingegen die Richtung der Konturen. Sie können daher in Gebieten mit Plateaus im Amplitudenbild zu „schwingen“ beginnen.

Parameter

- ▷ **`Edges`** (input_object) ..... region(-array) ⇝ *Hobject*
  Region mit ein Pixel breiten Kanten.
- ▷ **`EdgeImage`** (input_object) ..... image ⇝ *Hobject* : byte
  Kanten-Amplitudenbild.
- ▷ **`RegionResult`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ausgaberegionen (geschlossene Kanten).

▷ **MinAmplitude** (input_control) ..... integer ⇝ *integer*
Mindestamplitude für Kanten.
Defaultwert : 16
Wertevorschläge : MinAmplitude ∈ {5, 8, 10, 12, 16, 20, 25, 30, 40, 50}
Typischer Wertebereich : 1 ≤ MinAmplitude ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : MinAmplitude ≥ 0

Beispiel

```
sobel_amp(Image,&EdgeAmp,"sum_abs",5);
threshold(EdgeAmp,&EdgeRegion,40.0,255.0);
skeleton(EdgeRegion,&ThinEdge);
close_edges(ThinEdge,EdgeAmp,&CloseEdges,15);
skeleton(CloseEdges,&ThinCloseEdges);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert close_edges den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_edges ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

edges_image, sobel_amp, threshold, skeleton

Mögliche Nachfolgerfunktionen

skeleton

Alternativen

close_edges_length, dilation1, closing

Siehe auch

gray_skeleton

Modul

Image filters

close_edges_length (Edges, Gradient : ClosedEdges : MinAmplitude, MaxGapLength :)

Schließen von Kanten gemäß Amplitudenbild.

close_edges_length schließt die von einem Kantendetektor gelieferten Kanten zu Konturen. Als Eingabe erwartet die Routine also neben den Kanten (Edges) auch ein Amplitudenbild (Gradient), wie es beispielsweise die Routinen edges_image oder sobel_amp als Ergebnis liefern.

Der Konturschluß erfolgt in zwei Schritten: Zunächst werden ein Pixel große Lücken in den Eingabekonturstücken geschlossen und isolierte Kantenpunkte eliminiert. Im zweiten Schritt werden dann offene Konturen um maximal MaxGapLength verlängert, bis entweder die Kontur geschlossen ist oder kein Fortsetzungspunkt mit ausreichend signifikantem Gradienten mehr gefunden werden kann. Ein Gradientenwert gilt dabei als signifikant, wenn er größer oder gleich MinAmplitude ist. Kandidaten für den nächsten Fortsetzungspunkt sind der Nachbarpunkt in Richtung der Kontur (gemäß Gradient im aktuellen Endpunkt) sowie die beiden angrenzenden Nachbarpunkte (8er Nachbarschaft). Für jeden dieser drei potentiellen Fortsetzungspunkte wird die Summe aus dem eigenen Gradienten und dem maximalen Gradienten der nächsten drei möglichen Fortsetzungspunkte gebildet (*look ahead* der Länge 1). Der Punkt mit maximaler Summe wird dann als Fortsetzungspunkt ausgewählt.

Parameter

- ▷ **Edges** (input_object) ..... region(-array) ⇝ *Hobject*
  Region mit ein Pixel breiten Kanten.
- ▷ **Gradient** (input_object) ..... image ⇝ *Hobject* : byte
  Kanten-Amplitudenbild.
- ▷ **ClosedEdges** (output_object) ..... region(-array) ⇝ *Hobject*
  Ausgaberegion (geschlossene Kanten).
- ▷ **MinAmplitude** (input_control) ..... integer ⇝ *integer*
  Mindestamplitude für Kanten.
  Defaultwert : 16
  Wertevorschläge : `MinAmplitude` ∈ {5, 8, 10, 12, 16, 20, 25, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ `MinAmplitude` ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `MinAmplitude` ≥ `0`
- ▷ **MaxGapLength** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Punkten, um die die Kanten verlängert werden.
  Defaultwert : 3
  Wertevorschläge : `MaxGapLength` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50, 70, 100}
  Typischer Wertebereich : 1 ≤ `MaxGapLength` ≤ 127
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : (`MaxGapLength` > `0`) ∧ (`MaxGapLength` ≤ `127`)

Beispiel

```
sobel_amp(Image,&EdgeAmp,"sum_abs",5);
threshold(EdgeAmp,&EdgeRegion,40.0,255.0);
skeleton(EdgeRegion,&ThinEdge);
close_edges_length(ThinEdge,EdgeAmp,&CloseEdges,15,3);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `close_edges_length` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_edges_length` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`edges_image`, `sobel_amp`, `threshold`, `skeleton`

Alternativen

`close_edges`, `dilation1`, `closing`

Literatur

M. Üsbeck: “Untersuchungen zur echtzeitfähigen Segmentierung”; Studienarbeit, Bayerisches Forschungszentrum für Wissensbasierte Systeme (FORWISS), Erlangen, 1993.

Modul

Image filters

derivate_gauss (Image : DerivGauss : Sigma, Component :)

Ableitungen der Gaußfunktion.

`derivate_gauss` berechnet verschiedene Ableitungen der Gaußfunktion und daraus abgeleiteter Größen. Mögliche Werte für `Component` sind:

'none' Nur Glättung.

'x' 1. Ableitung nach x.

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

'y' 1. Ableitung nach y.

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

'gradient' Betrag des Gradienten.

$$g'(x,y) = \sqrt{\frac{\partial g(x,y)^2}{\partial x}\frac{\partial g(x,y)^2}{\partial y}}$$

'xx' 2. Ableitung nach x.

$$g'(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2}$$

'yy' 2. Ableitung nach y.

$$g'(x,y) = \frac{\partial^2 g(x,y)}{\partial y^2}$$

'xy' Ableitung nach x und y.

$$g'(x,y) = \frac{\partial^2 g(x,y)}{\partial x \partial y}$$

'xxx' 3. Ableitung nach x.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial x^3}$$

'yyy' 3. Ableitung nach y.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial y^3}$$

'xxy' Ableitung nach x, x und y.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial x^2 \partial y}$$

'xyy' Ableitung nach x, y und y.

$$g'(x,y) = \frac{\partial^3 g(x,y)}{\partial x \partial y^2}$$

'det' Determinante der Hessematrix:

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

'laplace' Laplace-Operator (Spur der Hessematrix)

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

'mean_curvatue' Mittlere Krümmung H

$$\begin{aligned}
a &= (1 + \frac{\partial g(x,y)^2}{\partial x})\frac{\partial^2 g(x,y)}{\partial y^2} \\
b &= 2\frac{\partial g(x,y)}{\partial x}\frac{\partial g(x,y)}{\partial y}\frac{\partial^2 g(x,y)}{\partial y \partial x} \\
c &= (1 + \frac{\partial g(x,y)^2}{\partial y})\frac{\partial^2 g(x,y)}{\partial x^2} \\
d &= (1 + \frac{\partial g(x,y)^2}{\partial x} + \frac{\partial g(x,y)^2}{\partial y})^{\frac{3}{2}} \\
H &= \frac{a - b + c}{d} \\
H &= \frac{1}{2}(\kappa_{min} + \kappa_{max})
\end{aligned}$$

'gauss_curvatue' Gaußkrümmung K

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

'eigenvalue1' Erster Eigenwert

$$\begin{aligned} a &= \frac{\frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}}{2} \\ \lambda_1 &= a + \sqrt{a^2 - (\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial^2 g(x,y)}{\partial y^2} - \frac{\partial^2 g(x,y)^2}{\partial y \partial x})} \end{aligned}$$

'eigenvalue2' Zweiter Eigenwert

$$\begin{aligned} a &= \frac{\frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}}{2} \\ \lambda_2 &= a - \sqrt{a^2 - (\frac{\partial^2 g(x,y)}{\partial x^2}\frac{\partial^2 g(x,y)}{\partial y^2} - \frac{\partial^2 g(x,y)^2}{\partial y \partial x})} \end{aligned}$$

'main1_curvature' Erste Hauptkrümmung

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

'main2_curvature' Zweite Hauptkrümmung

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

'kitchen_rosenfeld' 2. Ableitung in Richtung senkrecht zum Gradienten

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

'2nd_ddg' 2. Ableitung in Richtung des Gradienten

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

'de_saint_venant' 2. Ableitung in Richtung des Gradienten und senkrecht dazu

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

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Eingabebild.
- ▷ **`DerivGauss`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : real
  Gefiltertes Bild.
- ▷ **`Sigma`** (input_control) ..... real ⇝ *real*
  Sigma der Gaußfunktion.
  Defaultwert : 1.0
  Wertevorschläge : `Sigma` ∈ {0.7, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich : 0.2 ≤ `Sigma` ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Sigma` > `0.0`
- ▷ **`Component`** (input_control) ..... string ⇝ *string*
  Zu berechnende Ableitung oder Merkmal.
  Defaultwert : 'x'
  Werteliste : `Component` ∈ {'none', 'x', 'y', 'gradient', 'xx', 'yy', 'xy', 'xxx', 'yyy', 'xxy', 'xyy', 'det', 'mean_curvature', 'gauss_curvature', 'eigenvalue1', 'eigenvalue2', 'main1_curvature', 'main2_curvature', 'kitchen_rosenfeld', 'zuniga_haralick', "2nd_ddg", 'de_saint_venant', 'area', 'laplace', 'gradient_dir', 'eigenvec_dir'}

Beispiel

```
read_image(&Image,"mreut");
derivate_gauss(Image,&Gauss,3.0,"x");
zero_crossing(Gauss,&ZeroCrossings);
```

Parallelisierungsinformation

`derivate_gauss` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`zero_crossing`, `dual_threshold`

Alternativen

`laplace`, `laplace_of_gauss`, `gauss_image`, `smooth_image`

Siehe auch

`zero_crossing`, `dual_threshold`

Literatur

K. Voss H. Süse; “Praktische Bildverarbeitung” Hanser Verlag; München, Wien; 1991; Seite 88.

Modul

Image filters

`diff_of_gauss ( Image : DiffOfGauss : Sigma, SigFactor : )`

Näherung für den LoG-Operator (Laplace of Gaussian).

`diff_of_gauss` ist eine Näherung für den Laplace-Operator. Dieser wird hier als Differenz zweier Gaußfunktionen approximiert. Die Standardabweichungen dieser Gaußfunktionen ergeben sich nach Marr aus dem Parameter `Sigma` des Laplace-Operators und dem Verhältnis der beiden Standardabweichungen zueinander (`SigFactor`):

$$sigma1 = \frac{\texttt{Sigma}}{\sqrt{-2\frac{\log(\frac{1}{SigFactor})}{\texttt{SigFactor}^2-1}}}$$

$$sigma2 = \frac{sigma1}{\texttt{SigFactor}}$$

$$\texttt{DiffOfGauss} = (\texttt{Image} * gauss(sigma1)) - (\texttt{Image} * gauss(sigma2))$$

Für $\texttt{SigFactor} = 1.6$ ergibt sich nach Marr eine Näherung an den Mexican-Hat-Operator (LoG, Laplace of Gaussian). Das Ergebnisbild wird in `DiffOfGauss` abgelegt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebilder
- ▷ **`DiffOfGauss`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : int2
  Gefilterte Ausgabebilder.
- ▷ **`Sigma`** (input_control) ..... real ⇝ *real*
  Smoothing parameter of the Laplace operator to approximate.
  Defaultwert : 3.0
  Wertevorschläge : `Sigma` ∈ {2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich : $0.2 \leq$ `Sigma` ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Sigma` $>$ `0.0`

▷ **SigFactor** (input_control) ..... real ⇝ *real*
Verhältnis der Standardabweichungen der eingesetzten Gaußfunktionen (Marr empfiehlt 1.6).
Defaultwert : 1.6
Typischer Wertebereich : 0.1 ≤ SigFactor ≤ 10.0
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1
Restriktion : SigFactor > 0.0

Beispiel

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

Komplexität

Die Komplexität hängt linear von der Anzahl der Bildpunkte und der Größe von Sigma ab.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert diff_of_gauss den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

diff_of_gauss ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

zero_crossing, dual_threshold

Alternativen

laplace, derivate_gauss

Literatur

D. Marr: “Vision (A computational investigation into human representation and processing of visual information)”; New York, W.H. Freeman and Company; 1982.

Modul

Image filters

edges_image (Image : ImaAmp, ImaDir : Filter, Alpha, NMS, Low, High :)

Kantendetektion mit Deriche-, Lanser-, Shen- oder Canny-Filtern.

edges_image dient der Detektion von Stufenkanten mittels rekursiver Filter (nach Deriche, Lanser und Shen) bzw. mittels des konventionell über Filtermasken realisierten „Derivative of Gaussian“ Filter, den auch Canny verwendet. Konkret stehen somit die Kantenoperatoren

’deriche1’, ’lanser1’, ’deriche1_int4’, ’deriche2’, ’lanser2’, ’lanser2_int4’, ’shen’, ’mshen’ und ’canny’

zur Verfügung (Parameter Filter).

Es werden jeweils die Kantenamplituden (in ImaAmp) und -richtungen (in ImaDir) zurückgeliefert. Die Kantenoperatoren ’deriche1’ bzw. ’deriche2’ stehen auch für int4-Bilder zur Verfügung. In dem Fall liefert die Routine statt der Amplituden, also dem Betrag der Filterantwort, die eigentliche, vorzeichenbehaftete Filterantwort als int4-Bild. Dieses Verhalten läßt sich durch Verwendung von ’deriche1_int4’ bzw. ’deriche2_int4’ auch für Byte-Bilder erzwingen. Auf die vorzeichenbehafteten Filterantworten läßt sich edges_image (mit Parameter ’lanser2’) neuerlich anwenden, um die zweite Ableitung im Bild zu bestimmen. Die Kantenrichtungen werden in 2-Grad-Schritten kodiert, d.h. eine Kantenrichtung von x Grad (bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[E_x, E_y]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme	E_x/E_y	
Kantenrichtung r		
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$+/-$	$]0, 90[$
von rechts nach links	$+/0$	90
von rechts oben nach links unten	$+/+$	$]90, 180[$
von oben nach unten	$0/+$	180
von links oben nach rechts unten	$-/+$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$-/-$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Die „Filterbreite“ (i.e. das Einzugsgebiet der Filter) ist frei wählbar und mittels `info_edges` für konkrete Werte des Parameters `Alpha` abzuschätzen. Sie nimmt bei den Deriche-, Lanser- und Shen-Filtern mit `Alpha` ab bzw. beim Canny-Filter zu (`Alpha` ist in diesem Fall die Standardabweichung der zugrundeliegenden Gaußfunktion). „Breite“ Filter weisen eine höhere Rauschinvarianz, aber auch ein verringertes Auflösungsvermögen für Bilddetails auf. Nicht-rekursive Filter, wie hier die Canny-Filter, werden häufig mittels Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit natürlich mit wachsender Filterbreite. Die Laufzeit der rekursiver Filter ist hingegen konstant. Das Einzugsgebiet der Deriche-, Lanser bzw. Shen-Filter ist also ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierennde Laufzeitvorteil gegenüber dem Canny-Operator nimmt naturgemäß mit wachsender „Filterbreite“ zu. Zur Randbehandlung wird bei den rekursiven Filtern angenommen, das Bildsignal falle außerhalb des Bildes auf Null ab. Beim Canny-Operator wird hingegen das Bild durch „Vervielfachung“ der Randpunkte fortgesetzt. Vergleichbare Einzugsgebiete der Filter erhält man bei folgender Wahl von Alpha:

$$
\begin{aligned}
\texttt{Alpha}('lanser1') &= \texttt{Alpha}('deriche1') \\
\texttt{Alpha}('deriche2') &= \texttt{Alpha}('deriche1')/2 \\
\texttt{Alpha}('lanser2') &= \texttt{Alpha}('deriche2') \\
\texttt{Alpha}('shen') &= \texttt{Alpha}('deriche1')/2 \\
\texttt{Alpha}('mshen') &= \texttt{Alpha}('shen') \\
\texttt{Alpha}('canny') &= 1.77/\texttt{Alpha}('deriche1')
\end{aligned}
$$

Die hier eingesetzten rekursiven Filter verzerren in ihrer ursprünglichen Form (’deriche1’, ’deriche2’, ’shen’) die Amplituden diagonaler Kanten. Diese Fehler sind in den zugehörigen modifizierten Operatoren ’lanser1’, ’lanser2’ und ’mshen’ (bei unveränderter Laufzeit) beseitigt.

Bei relativ kleinem Einzugsgebiet der Filter (11×11) Bildpunkte, etwa für Alpha (’lanser2’ = 0.5) liefern alle (modifizierten) Operatoren praktisch identische Ergebnisse. Erst bei „breiteren“ Filtern zeigen sich Unterschiede: Insbesondere fallen dann die Shen-basierten Operatoren qualitativ ab. Allerdings sind sie die schnellsten der implementierten Operatoren — dicht gefolgt von den Deriche-Operatoren.

`edges_image` bietet (optional) die Weiterverarbeitung der Filterergebnisse mittels Non-Maximum-Suppression (`NMS` = ’nms’/’inms’/’hvnms’, sonst Wert ’none’) und Hysterese-Schwellenwertoperation (`Low`,`High`, falls nicht erwünscht: einer der Werte negativ). Dies entspricht im wesentlichen nachfolgenden Aufrufen der Routinen

```
nonmax_suppression_dir(...,NMS,...)
hysteresis_threshold(...,Low,High,999,...)
```

Letztere liefern als Ergebnis jedoch entsprechend modifizierte Regionen, während `edges_image` unterdrückte Kantenpunkte durch die Grauwerte 0 (Amplitudenbild) bzw. 255 (Richtungsbild) markiert, die Eingangsregion aber unverändert läßt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int4 / int2
  Eingabebilder.
- ▷ **`ImaAmp`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int4 / int2
  Amplitudenbilder.

▷ **ImaDir** (output_object) ..... image(-array) ⇝ *Hobject* : direction
Richtungsbilder.

▷ **Filter** (input_control) ..... string ⇝ *string*
Gewünschter Kanten-Operator.
Defaultwert : 'lanser2'
Werteliste : Filter ∈ {'deriche1', 'lanser1', 'deriche1_int4', 'deriche2', 'lanser2', 'lanser2_int4', 'shen', 'mshen', 'canny'}

▷ **Alpha** (input_control) ..... real ⇝ *real*
Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei 'canny' umgekehrt).
Defaultwert : 0.5
Wertevorschläge : Alpha ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1}
Typischer Wertebereich : 0.2 ≤ Alpha ≤ 50.0
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1
Restriktion : Alpha > 0.0

▷ **NMS** (input_control) ..... string ⇝ *string*
Non-Maximum Suppression ('none', falls nicht erwünscht).
Defaultwert : 'nms'
Werteliste : NMS ∈ {'nms', 'inms', 'hvnms', 'none'}

▷ **Low** (input_control) ..... integer ⇝ *integer*
Untere Schwelle für Hysterese-Schwellenwertoperation (negativ, falls keine Schwellenwertbildung erwünscht).
Defaultwert : 20
Wertevorschläge : Low ∈ {5, 10, 15, 20, 25, 30, 40}
Typischer Wertebereich : 1 ≤ Low ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : (Low > 1) ∨ (Low < 0)

▷ **High** (input_control) ..... integer ⇝ *integer*
Obere Schwelle für Hysterese-Schwellenwertoperation (negativ, falls keine Schwellenwertbildung erwünscht).
Defaultwert : 40
Wertevorschläge : High ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
Typischer Wertebereich : 1 ≤ High ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : ((High > 1) ∨ (High < 0)) ∧ (High ≥ Low)

Beispiel

```
read_image(Image,'fabrik')
edges_image(Image,Amp,Dir,'lanser2',0.5,'none',-1,-1)
hysteresis_threshold(Amp,Margin,20,30,30).
```

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert edges_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

edges_image ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

info_edges

Mögliche Nachfolgerfunktionen

threshold, hysteresis_threshold, close_edges_length

Alternativen

sobel_dir, frei_dir, kirsch_dir, prewitt_dir, robinson_dir

Siehe auch

info_edges, nonmax_suppression_amp, hysteresis_threshold, bandpass_image

Literatur

S.Lanser, W.Eckstein: “Eine Modifikation des Deriche-Verfahrens zur Kantendetektion”; 13. DAGM-Symposium, München; Informatik Fachberichte 290; Seite 151 - 158; Springer-Verlag; 1991.

S.Lanser: “Detektion von Stufenkanten mittels rekursiver Filter nach Deriche”; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: “Finding Edges and Lines in Images”; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

J.Canny: “A Computational Approach to Edge Detection”; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-8, vol. 6; S. 679-698; 1986.

R.Deriche: “Using Canny’s Criteria to Derive a Recursively Implemented Optimal Edge Detector”; International Journal of Computer Vision; vol. 1, no. 2; S. 167-187; 1987.

R.Deriche: “Optimal Edge Detection Using Recursive Filtering”; Proc. of the First International Conference on Computer Vision, London; S. 501-505; 1987.

R.Deriche: “Fast Algorithms for Low-Level Vision”; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; S. 78-87; 1990.

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.

Modul

Image filters

edges_sub_pix (Image : Edges : Filter, Alpha, Low, High :)

Sub-pixel-genaue Kantendetektion mit Deriche-, Lanser-, Shen- oder Canny-Filtern.

edges_sub_pix dient der Detektion von Stufenkanten mittels rekursiver Filter (nach Deriche, Lanser und Shen) bzw. mittels des konventionell über Filtermasken realisierten „Derivative of Gaussian“ Filter, den auch Canny verwendet. Konkret stehen somit die Kantenoperatoren

’deriche1’, ’lanser1’, ’deriche2’, ’lanser2’, ’shen’, ’mshen’, ’canny’ und ’sobel’

zur Verfügung (Parameter Filter).

Die extrahierten Kanten werden als sub-pixel-genaue XLD-Konturen in Edges zurückgeliefert. Für jeden Kantenpunkt werden die folgenden Attribute definiert (siehe get_contour_attrib_xld): **’edge_direction’**: Die Kantenrichtung **’angle’**: Richtung der Normalenvektoren der Kontur (so orientiert, daß die Normalenvektoren auf die rechte Seite der Kontur in Durchlaufrichtung der Kontur zeigen; die Winkel sind bzgl. der Zeilenachse des Bildes angegeben.) **’response’**: Die Amplitude der Kante

’edge_direction’	Die Kantenrichtung
’angle’	Richtung der Normalenvektoren der Kontur (so orientiert, daß die Normalenvektoren auf die rechte Seite der Kontur in Durchlaufrichtung der Kontur zeigen; die Winkel sind bzgl. der Zeilenachse des Bildes angegeben.)
’response’	Die Amplitude der Kante

Die „Filterbreite“ (i.e. das Einzugsgebiet der Filter) ist frei wählbar und mittels info_edges für konkrete Werte des Parameters Alpha abzuschätzen. Sie nimmt bei den Deriche-, Lanser- und Shen-Filtern mit Alpha ab bzw. beim Canny-Filter zu (Alpha ist in diesem Fall die Standardabweichung der zugrundeliegenden Gaußfunktion). „Breite“ Filter weisen eine höhere Rauschinvarianz, aber auch ein verringertes Auflösungsvermögen für Bilddetails auf. Nicht-rekursive Filter, wie hier die Canny-Filter, werden häufig mittels Filtermasken realisiert. In diesem Fall erhöht sich die Laufzeit natürlich mit wachsender Filterbreite. Die Laufzeit der rekursiven Filter ist hingegen konstant. Das Einzugsgebiet der Deriche-, Lanser bzw. Shen-Filter ist also ohne Mehraufwand beliebig vergrößerbar. Der daraus resultierennde Laufzeitvorteil gegenüber dem Canny-Operator nimmt naturgemäß mit wachsender „Filterbreite“ zu. Zur Randbehandlung wird bei den rekursiven Filtern angenommen, das Bildsignal falle außerhalb des Bildes auf Null ab. Beim Canny-Operator wird hingegen das Bild durch „Vervielfachung“ der Randpunkte fortgesetzt. Vergleichbare Einzugsgebiete der Filter erhält man bei folgender Wahl von Alpha:

$$\begin{aligned}
\texttt{Alpha}('lanser1') &= \texttt{Alpha}('deriche1') \\
\texttt{Alpha}('deriche2') &= \texttt{Alpha}('deriche1')/2 \\
\texttt{Alpha}('lanser2') &= \texttt{Alpha}('deriche2') \\
\texttt{Alpha}('shen') &= \texttt{Alpha}('deriche1')/2 \\
\texttt{Alpha}('mshen') &= \texttt{Alpha}('shen') \\
\texttt{Alpha}('canny') &= 1.77/\texttt{Alpha}('deriche1')
\end{aligned}$$

Die hier eingesetzten rekursiven Filter verzerren in ihrer ursprünglichen Form (’deriche1’, ’deriche2’, ’shen’) die Amplituden diagonaler Kanten. Diese Fehler sind in den zugehörigen modifizierten Operatoren ’lanser1’, ’lanser2’ und ’mshen’ (bei unveränderter Laufzeit) beseitigt.

Bei relativ kleinem Einzugsgebiet der Filter (11×11) Bildpunkte, etwa für Alpha (’lanser2’ = 0.5) liefern alle (modifizierten) Operatoren praktisch identische Ergebnisse. Erst bei „breiteren“ Filtern zeigen sich Unterschiede: Insbesondere fallen dann die Shen-basierten Operatoren qualitativ ab. Allerdings sind sie die schnellsten der implementierten Operatoren dicht gefolgt von den Deriche-Operatoren.

`edges_sub_pix` verknüpft die einzelnen Kantenpunkte mit einem Hysterese-Schwellenwert-artigem Verfahren zu Kanten, wie es auch in `lines_gauss` verwendet wird. Dabei werden Punkte, deren Amplitude größer als `High` ist, sofort als sichere Kantenpunkte akzeptiert. Punkte, deren Amplitude kleiner als `Low` ist, werden sofort verworfen. Alle Punkte, die eine zweite Ableitung zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind (siehe `lines_gauss` und `hysteresis_threshold`).

Da Kantenextraktoren oftmals bestimmte Kreuzungspunkte nicht extrahieren können, kann durch Anhängen von **’_junctions’** an die oben beschriebenen Werte von `Filter` versucht werden, diese mit anderen Mitteln zu extrahieren. Dieser Modus ist analog zu dem bei `lines_gauss` verfügbaren Modus zur Vervollständigung von Kreuzungspunkten.

Parameter

- ▷ **`Image`** (input_object) . . . . . image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **`Edges`** (output_object) . . . . . xld_cont-array ⇝ *Hobject*
  Extrahierte Kanten.
- ▷ **`Filter`** (input_control) . . . . . string ⇝ *string*
  Gewünschter Kanten-Operator.
  Defaultwert : ’lanser2’
  Werteliste : `Filter` ∈ {’deriche1’, ’lanser1’, ’deriche2’, ’lanser2’, ’shen’, ’mshen’, ’canny’, ’sobel’, ’deriche1_junctions’, ’lanser1_junctions’, ’deriche2_junctions’, ’lanser2_junctions’, ’shen_junctions’, ’mshen_junctions’, ’canny_junctions’, ’sobel_junctions’}
- ▷ **`Alpha`** (input_control) . . . . . real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei ’canny’ umgekehrt).
  Defaultwert : 0.5
  Wertevorschläge : `Alpha` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.9, 1.1}
  Typischer Wertebereich : 0.2 ≤ `Alpha` ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Alpha` > `0.0`
- ▷ **`Low`** (input_control) . . . . . integer ⇝ *integer*
  Untere Schwelle für Hysterese-Schwellenwertoperation.
  Defaultwert : 20
  Wertevorschläge : `Low` ∈ {5, 10, 15, 20, 25, 30, 40}
  Typischer Wertebereich : 1 ≤ `Low` ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5
  Restriktion : `Low` > `0`

▷ **High** (input_control) ..... integer ⇝ *integer*
Obere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert : 40
Wertevorschläge : High ∈ {10, 15, 20, 25, 30, 40, 50, 60, 70}
Typischer Wertebereich : 1 ≤ High ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : (High > 0) ∧ (High ≥ Low)

Beispiel

```
read_image(Image,'fabrik')
edges_sub_pix(Image,Edges,'lanser2',0.5,20,40).
```

Komplexität

Sei A die Anzahl von Pixeln in der Region von Image. Dann ist die Laufzeitkomplexität $O(A * Sigma)$ für den Canny-Filter und $O(A)$ für die rekursiven Lanser-, Deriche- und Shen-Filter.

Sei $S = Width * Height$ die Anzahl der Pixel in Image. Dann benötigt edges_sub_pix mindestens $60 * S$ Bytes an temporärem Speicher bei der Ausführung.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert edges_sub_pix den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

edges_sub_pix ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Alternativen

sobel_dir, frei_dir, kirsch_dir, prewitt_dir, robinson_dir, edges_image

Siehe auch

info_edges, hysteresis_threshold, bandpass_image, lines_gauss, lines_facet

Literatur

S.Lanser, W.Eckstein: “Eine Modifikation des Deriche-Verfahrens zur Kantendetektion”; 13. DAGM-Symposium, München; Informatik Fachberichte 290; Seite 151 - 158; Springer-Verlag; 1991.

S.Lanser: “Detektion von Stufenkanten mittels rekursiver Filter nach Deriche”; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: “Finding Edges and Lines in Images”; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

J.Canny: “A Computational Approach to Edge Detection”; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-8, vol. 6; S. 679-698; 1986.

R.Deriche: “Using Canny’s Criteria to Derive a Recursively Implemented Optimal Edge Detector”; International Journal of Computer Vision; vol. 1, no. 2; S. 167-187; 1987.

R.Deriche: “Optimal Edge Detection Using Recursive Filtering”; Proc. of the First International Conference on Computer Vision, London; S. 501-505; 1987.

R.Deriche: “Fast Algorithms for Low-Level Vision”; IEEE Transactions on Pattern Analysis and Machine Intelligence; PAMI-12, no. 1; S. 78-87; 1990.

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.

Modul

Sub-pixel operators

frei_amp (Image : ImageEdgeAmp : :)

Kantendetektion (Amplitude) mit dem Frei-Chen-Operator.

frei_amp berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.

Beispiel

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

Ergebnis

frei_amp liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

frei_amp ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, sigma_image, median_image, smooth_image

Alternativen

sobel_amp, kirsch_amp, prewitt_amp, robinson_amp, roberts

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Image filters

frei_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Frei-Chen-Operator.

frei_dir berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & \sqrt{2} & 1 \\ 0 & 0 & 0 \\ -1 & -\sqrt{2} & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen. In dem Parameter ImageEdgeDir wird die Kantenrichtungen zurückgeliefert. Diese werden in 2 Grad Schritten kodiert, d.h.

eine Kantenrichtung von x Grad (bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[Ex, Ey]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme E_x/E_y	$Kantenrichtung$r	
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$+/-$	$]0, 90[$
von rechts nach links	$+/0$	90
von rechts oben nach links unten	$+/+$	$]90, 180[$
von oben nach unten	$0/+$	180
von links oben nach rechts unten	$-/+$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$-/-$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.
- ▷ **ImageEdgeDir** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : direction
  Kantenrichtung.

Beispiel

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

Ergebnis

`frei_dir` liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`frei_dir` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `sigma_image`, `median_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `robinson_dir`, `prewitt_dir`, `kirsch_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Image filters

highpass_image (Image : Highpass : Width, Height :)

Verstärkung hochfrequenter Bildanteile.

highpass_image verstärkt hochfrequente Bildanteile durch eine lineare Filterung mit folgender Filtermatrix (am Beispiel 7×5):

$$\begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & -35 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Die Anwendung der Prozedur entspricht der Anwendung des Mittelwertfilters (mean_image), gefolgt von der Subtraktion des Originalgrauwerts vom Ergebnis. Zu der Differenz wird 128 addiert, d.h. der Nulldurchgang hat den Wert 128.

Der Filter hebt hochfrequente Anteile im Bild hervor (Kanten und Ecken). Die Trennfrequenz wird über die Größe (Height $\times$ Width) der Filtermatrix festgelegt: Je größer die Matrix ist, desto niedriger wird die Trennfrequenz.

Als Randbehandlungen werden die Grauwerte an den Objekträndern gespiegelt. Über- bzw. Unterlauf von Grauwerten wird beschnitten (255 bzw. 0).

Achtung

Werden für Height oder Width gerade statt ungerade Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebilder.
- ▷ **Highpass** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Hochpaß-gefilterte Ausgabebilder.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 9
  Wertevorschläge : Width $\in \{3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101\}$
  Typischer Wertebereich : $3 \leq$ Width ≤ 501
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : (Width ≥ 3) $\wedge$ odd(Width)
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 9
  Wertevorschläge : Height $\in \{3, 5, 7, 9, 11, 13, 17, 21, 29, 41, 51, 73, 101\}$
  Typischer Wertebereich : $3 \leq$ Height ≤ 501
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : (Height ≥ 3) $\wedge$ odd(Height)

Beispiel

```
highpass_image(Image,&Highpass,7,5);
threshold(Highpass,&Region,60.0,255.0);
skeleton(Region,&Skeleton);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert highpass_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

highpass_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

threshold, skeleton

Alternativen

`mean_image`, `sub_image`, `convol_image`, `bandpass_image`

Siehe auch

`dyn_threshold`

Modul

Image filters

`info_edges ( : : Filter, Mode, Alpha : Size, Coeffs )`

Abschätzung des Einzugsgebiets der Filter aus `edges_image`.

`info_edges` liefert eine Abschätzung des Einzugsgebiets der in der Routine `edges_image` verwendeten Filter. Dazu werden die zugrundeliegenden kontinuierlichen Impulsantworten der Filter abgetastet bis ein Filterkoeffizient kleiner als fünf Prozent des Maximalkoeffizienten ist. `Alpha` ist der Filterparameter (siehe `edges_image`). Es werden sieben Kantenoperatoren unterstützt (Parameter `Filter`):

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

Über den Parameter `Mode` ('edge'/'smooth') wird festgelegt, ob der entsprechende Kanten- oder Glättungsfilter gemeint ist. Der Canny-Operator (der sich auf die Gaußfunktion abstützt) wurde dabei konventionell mittels Filtermasken implementiert (die anderen verwenden rekursive Filter). Im Falle der Canny-Filter werden daher zusätzlich zur Filtergröße auch die Filterkoeffizienten der eindimensionalen Impulsantworten $f(n)$ mit $n \geq 0$ in `Coeffs` zurückgeliefert.

Parameter

- ▷ **`Filter`** (input_control) ..... string ⇝ *string*
  Name des Kanten-Operators.
  Defaultwert : 'lanser2'
  Werteliste : `Filter` ∈ {'deriche1', 'lanser1', 'deriche2', 'lanser2', 'shen', 'mshen', 'canny'}
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  1D-Kantenfilter ('edge') oder 1D-Glättungsfilter ('smooth').
  Defaultwert : 'edge'
  Werteliste : `Mode` ∈ {'edge', 'smooth'}
- ▷ **`Alpha`** (input_control) ..... real ⇝ *real*
  Filterparameter: kleine Werte bewirken starke Glättung, also auch weniger Bilddetails (bei canny umgekehrt).
  Defaultwert : 0.5
  Typischer Wertebereich : $0.2 \leq$ `Alpha` ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Alpha` $>$ `0.0`
- ▷ **`Size`** (output_control) ..... integer ⇝ *integer*
  Größe des Einzugsgebietes des Filters in Pixeln.
- ▷ **`Coeffs`** (output_control) ..... integer-array ⇝ *integer*
  Falls Cannyfilter: Koeffizienten der „positiven“ Hälfte der 1D-Impulsantwort.

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `info_edges` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`info_edges` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

edges_image, threshold, skeleton

Siehe auch

edges_image

Modul

Image filters

kirsch_amp (Image : ImageEdgeAmp : :)

Kantendetektion mit dem Kirsch-Operator.

kirsch_amp berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende acht Filtermasken zugrunde:

$$\begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix}$$

$$\begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
kirsch_amp(Image,Kirsch_amp)
threshold(Kirsch_amp,Edges,128,255).
```

Ergebnis

kirsch_amp liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

kirsch_amp ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, sigma_image, median_image, smooth_image

Alternativen

sobel_amp, frei_amp, prewitt_amp, robinson_amp, roberts

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Image filters

kirsch_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Kirsch-Operator.

kirsch_dir berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende acht Filtermasken zugrunde:

$$\begin{matrix} -3 & -3 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & 5 \end{matrix}$$

$$\begin{matrix} -3 & 5 & 5 \\ -3 & 0 & 5 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & 5 & -3 \\ 5 & 0 & -3 \\ -3 & -3 & -3 \end{matrix}$$

$$\begin{matrix} 5 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & -3 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ 5 & 0 & -3 \\ 5 & 5 & -3 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & -3 \\ 5 & 5 & 5 \end{matrix}$$

$$\begin{matrix} -3 & -3 & -3 \\ -3 & 0 & 5 \\ -3 & 5 & 5 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen. In dem Parameter ImageEdgeDir wird die Kantenrichtungen zurückgeliefert als $x/2$. Die Kantenrichtungen entsprechen der Richtung der Maske mit der maximalen Filterantwort.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.
- ▷ **ImageEdgeDir** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : direction
  Kantenrichtung.

Beispiel

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

Ergebnis

`kirsch_dir` liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`kirsch_dir` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `sigma_image`, `median_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `robinson_dir`, `prewitt_dir`, `frei_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Image filters

laplace (Image : ImageLaplace : FilterType, Size, NeighbourhoodType :)

Calculate the Laplace operator by using finite differences.

`laplace` filtert die Eingabebilder (`Image`) mittels des Laplace-Operators. Abhängig vom Parameter `NeighbourhoodType` werden folgende einfache lokale Approximationen an den Laplace-Operator zugrundegelegt:

'n_4'

$$\begin{matrix} & 1 & \\ 1 & -4 & 1 \\ & 1 & \end{matrix}$$

'n_8'

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

'n_8_isotrop'

$$\begin{matrix} 10 & 22 & 10 \\ 22 & -128 & 22 \\ 10 & 22 & 10 \end{matrix}$$

Der 'n_8' Filter entspricht der Filtermaske, die in `highpass_image(O:R:3,3:)` verwendet wird. Für eine 3×3 Laplace-Filterung (`Size` = 3) wird der entsprechende Filter direkt angewandt. Für größere Laplace-Filter (`Size` = 5,7,9 und 11) wird das Eingangsbild zuerst mit dem Gaußfilter gleicher Größe geglättet und anschließend einer der drei obigen Filter eingesetzt. Somit ist

```
laplace(O:R:int4,S,N:)
```

für `Size` > 3 äquivalent zu

```
gauss_image(O:G:S:) ▷
laplace(G:R:int4,3,N:).
```

`laplace` liefert entweder den Absolutbetrag des Laplace-gefilterten Eingabebildes (`FilterType` 'abs') in einem Byte-Bild oder unmittelbar das Filterergebnis (`FilterType` 'int4') in einem int4-Bild zurück.

Achtung

`laplace` ist nur für Byte-Bilder implementiert.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebilder.
- ▷ **`ImageLaplace`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int4
  Laplace-gefilterte Ausgabebilder.
- ▷ **`FilterType`** (input_control) ..... string ⇝ *string*
  Berechnung des Absolutbetrags des Filterergebnisses in Byte-Bild oder Original-Filterergebnis in int4-Bild.
  Defaultwert : 'int4'
  Werteliste : `FilterType` ∈ {'abs', 'int4'}
- ▷ **`Size`** (input_control) ..... integer ⇝ *integer*
  Filtergröße.
  Defaultwert : 3
  Werteliste : `Size` ∈ {3, 5, 7, 9, 11}
- ▷ **`NeighbourhoodType`** (input_control) ..... string ⇝ *string*
  Nachbarschaft für den Laplace-Operator
  Defaultwert : 'n_8_isotrop'
  Werteliste : `NeighbourhoodType` ∈ {'n_4', 'n_8', 'n_8_isotrop'}

Beispiel

```
read_image(&Image,"mreut");
laplace(Image,&Laplace,"int4",3,"n_8_isotrop");
zero_crossing(Laplace,&ZeroCrossings);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `laplace` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`laplace` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`zero_crossing`, `dual_threshold`

Alternativen

`diff_of_gauss`, `laplace_of_gauss`

Siehe auch

`highpass_image`, `edges_image`

Modul

Image filters

laplace_of_gauss (Image : ImageLaplace : Sigma :)

LoG-Operator (Laplace of Gaussian).

laplace_of_gauss berechnet den Laplaceoperator für beliebige Sigma. Die Formel für den Laplace-Operator lautet:

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

Die Ableitungen bei laplace_of_gauss werden durch Ableitungen der Gaußfunktion angenähert, wodurch sich folgende Formel für den Faltungsoperator ergibt:

$$\Delta G_\sigma(x,y) =$$
$$\frac{1}{2\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^2}-1\right)\left[\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)\right]$$

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Eingabebild.
- ▷ **ImageLaplace** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : int2
  Laplace des Eingabebildes.
- ▷ **Sigma** (input_control) ..... number ⇝ *real* / integer
  Glättungsparameter der Gaußfunktion.
  Defaultwert : 2.0
  Wertevorschläge : Sigma ∈ {0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 4.0, 5.0, 7.0}
  Typischer Wertebereich : 0.7 ≤ Sigma ≤ 5.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : (Sigma > 0.7) ∧ (Sigma ≤ 25.0)

Beispiel

```
read_image(&Image,"mreut");
laplace_of_gauss(Image,&Laplace,2.0);
zero_crossing(Laplace,&ZeroCrossings);
```

Parallelisierungsinformation

laplace_of_gauss ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

zero_crossing, dual_threshold

Alternativen

laplace, diff_of_gauss, derivate_gauss

Siehe auch

derivate_gauss

Modul

Image filters

prewitt_amp (Image : ImageEdgeAmp : :)

Kantendetektion (Amplitude) mit dem Prewitt-Operator.

prewitt_amp berechnet eine Näherung der ersten Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
prewitt_amp(Image,Prewitt)
threshold(Prewitt,Edges,128,255).
```

Ergebnis

prewitt_amp liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

prewitt_amp ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, sigma_image, median_image, smooth_image

Mögliche Nachfolgerfunktionen

threshold, gray_skeleton, nonmax_suppression_amp, close_edges, close_edges_length

Alternativen

sobel_amp, kirsch_amp, frei_amp, robinson_amp, roberts

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Image filters

prewitt_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Prewitt-Operator.

prewitt_dir berechnet eine Näherung der erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort der Maske A und B eingetragen. In dem Parameter `ImageEdgeDir` wird die Kantenrichtungen zurückgeliefert. Diese werden in 2 Grad Schritten kodiert, d.h. eine Kantenrichtung von x Grad (bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[Ex, Ey]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme	E_x/E_y	$Kantenrichtung$r
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$+/-$	$]0, 90[$
von rechts nach links	$+/0$	90
von rechts oben nach links unten	$+/+$	$]90, 180[$
von oben nach unten	$0/+$	180
von links oben nach rechts unten	$-/+$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$-/-$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **`ImageEdgeAmp`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.
- ▷ **`ImageEdgeDir`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : direction
  Kantenrichtung.

Beispiel

```
read_image(Image,'fabrik')
prewitt_dir(Image,PrewittA,PrewittD)
threshold(PrewittA,Edges,128,255).
```

Ergebnis

`prewitt_dir` liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`prewitt_dir` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `sigma_image`, `median_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `robinson_dir`, `frei_dir`, `kirsch_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Image filters

`roberts ( Image : ImageRoberts : FilterType : )`

Kantendetektion mit dem Roberts-Filter.

roberts berechnet die erste Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Legt man folgende Maske zugrunde,

$$\begin{array}{cc} A & B \\ C & D \end{array}$$

dann leiten sich die verschiedenen Filtertypen wie folgt ab:

'roberts_max'	$max(\|A-D\|, \|B-C\|)$
'gradient_max'	$max(\|A+B-(C+D)\|, \|A+C-(B+D)\|)$
'gradient_sum'	$\|A+B-(C+D)\| + \|A+C-(B+D)\|$

Tritt ein Überlauf ein, so werden die Werte beschnitten. Das Ergebnis der Berechnung wird in dem Pixel mit den Koordinaten von „*D*“ abgelegt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebilder.
- ▷ **ImageRoberts** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Gefilterte Ausgabebilder.
- ▷ **FilterType** (input_control) ..... string ⇝ *string*
  Filtertyp.
  Defaultwert : 'gradient_sum'
  Werteliste : FilterType ∈ {'roberts_max', 'gradient_max', 'gradient_sum'}

Beispiel

```
read_image(Image,'fabrik')
roberts(Image,Roberts,'roberts_max')
threshold(Roberts,Margin,128,255).
```

Ergebnis

roberts liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

roberts ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image

Mögliche Nachfolgerfunktionen

threshold, skeleton

Alternativen

edges_image, sobel_amp, frei_amp, kirsch_amp, prewitt_amp

Siehe auch

laplace, highpass_image, bandpass_image

Modul

Image filters

robinson_amp (Image : ImageEdgeAmp : :)

Kantendetektion (Amplitude) mit dem Robinson-Operator.

robinson_amp berechnet eine Näherung der ersten Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Mit robinson_amp werden auf ein Eingabebild vier der ursprünglich acht 3×3-Filter von Robinson

angewandt. Die anderen 4 Masken entstehen durch Multiplikation der Masken mit -1. Die Masken besitzen nur die Werte 0,1,-1,2,-2.

$$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{matrix}$$

$$\begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix}$$

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **ImageEdgeAmp** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.

Beispiel

```
read_image(Image,'fabrik')
robinson_amp(Image,Robinson_amp)
threshold(Robinson_amp,Edges,128,255).
```

Ergebnis

robinson_amp liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

robinson_amp ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, sigma_image, median_image, smooth_image

Alternativen

sobel_amp, frei_amp, prewitt_amp, robinson_amp, roberts

Siehe auch

bandpass_image, laplace_of_gauss

Modul

Image filters

robinson_dir (Image : ImageEdgeAmp, ImageEdgeDir : :)

Kantendetektion (Amplitude und Richtung) mit dem Robinson-Operator.

robinson_dir berechnet eine Näherung der ersten Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Mit robinson_dir werden auf ein Eingabebild vier der ursprünglich acht 3×3-Filter von Robinson angewandt. Die anderen 4 Masken entstehen durch Multiplikation der Masken mit -1. Die Masken besitzen nur die Werte 0,1,-1,2,-2.

$$\begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix}$$

$$\begin{matrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{matrix}$$

$$\begin{matrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{matrix}$$

$$\begin{matrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{matrix}$$

Im Ausgabebild ist die maximale Filterantwort aller Masken eingetragen. In dem Parameter `ImageEdgeDir` wird die Kantenrichtungen zurückgeliefert als $x/2$. Die Kantenrichtungen entsprechen der Richtung der Maske mit der maximalen Filterantwort.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **`ImageEdgeAmp`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.
- ▷ **`ImageEdgeDir`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : direction
  Kantenrichtung.

Beispiel

```
read_image(Image,'fabrik')
robinson_dir(Image,Robinson_dirA,Robinson_dirD)
threshold(Robinson_dirA,Res,128,255).
```

Ergebnis

`robinson_dir` liefert immer den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`robinson_dir` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `sigma_image`, `median_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`hysteresis_threshold`, `threshold`, `gray_skeleton`, `nonmax_suppression_dir`, `close_edges`, `close_edges_length`

Alternativen

`edges_image`, `sobel_dir`, `kirsch_dir`, `prewitt_dir`, `frei_dir`

Siehe auch

`bandpass_image`, `laplace_of_gauss`

Modul

Image filters

`sobel_amp` (Image : EdgeAmplitude : FilterType, Size :)

Kantendetektion (Amplitude) mit dem Sobel-Operator.

sobel_amp berechnet eine Art von erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Diese Masken werden je nach Filtertyp unterschiedlich verwendet. (a und b stehen im folgenden für die Ergebnisse der Faltung mit A und B in einem Bildpunkt):

'sum_sqrt'	$\sqrt{a^2 + b^2}$
'sum_abs'	$(\|a\| + \|b\|)/2$
'thin_sum_abs'	$(thin(\|a\|) + thin(\|b\|))/2$
'thin_max_abs'	$max(thin(\|a\|), thin(\|b\|))$
'x'	b
'y'	a

$thin(x)$ ist dabei für ein vertikales Maximum (Maske A) bzw. ein horizontales Maximum (Maske B) gleich x und 0 sonst. Damit wird bei **'thin_sum_abs'** und **'thin_max_abs'** eine Verdünnung des Gradientenbildes erreicht. Bei Byte-Bildern wird bei dem Modus **'x'** bzw. **'y'** als Ergebnis ein int1-Bild zurückgeliefert. Für eine 3×3 Sobel-Filterung (Size = 3) werden die Filter A und B direkt angewandt. Für größere Sobel-Filter (Size = 5,7, 9, 11 und 13) wird das Eingabebild dagegen zunächst mit dem Gaußfilter der Größe Size-2 geglättet. Somit ist

```
sobel_amp(I:E,Dir:FilterTyp,S:)
```

für Size > 3 äquivalent zu

```
gauss_image(I:G:S-2) ▷
sobel_amp(G:E:FilterType,3:).
```

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
Eingabebild.

▷ **EdgeAmplitude** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : int1 / int2
Ausgabebild mit Kantenamplitude.

▷ **FilterType** (input_control) ..... string ⇝ *string*
Filtertyp.
Defaultwert : 'sum_abs'
Werteliste : FilterType ∈ {'sum_abs', 'thin_sum_abs', 'thin_max_abs', 'sum_sqrt', 'x', 'y'}

▷ **Size** (input_control) ..... integer ⇝ *integer*
Filtergröße.
Defaultwert : 3
Werteliste : Size ∈ {3, 5, 7, 9, 11, 13}

Beispiel

```
read_image(Image,'fabrik')
sobel_amp(Image,Amp,'sum_abs',3)
threshold(Amp,Edg,128,255).
```

Ergebnis

sobel_amp liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`sobel_amp` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `mean_image`, `anisotrope_diff`, `sigma_image`

Mögliche Nachfolgerfunktionen

`threshold`, `nonmax_suppression_amp`, `gray_skeleton`

Alternativen

`frei_amp`, `roberts`, `kirsch_amp`, `prewitt_amp`, `robinson_amp`

Siehe auch

`laplace`, `highpass_image`, `bandpass_image`

Modul

Image filters

sobel_dir (Image : EdgeAmplitude, EdgeDirection : FilterType, Size :)

Kantendetektion (Amplitude und Richtung) mit dem Sobel-Operator.

`sobel_dir` berechnet eine Art von erster Ableitung der Grauwertdaten und wird als Kantenfilter eingesetzt. Dem Filter liegen folgende zwei Filtermasken zugrunde:

$$A = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

$$B = \begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Diese Masken werden je nach Filtertyp unterschiedlich verwendet. (a und b stehen im folgenden für die Ergebnisse der Faltung mit A und B in einem Bildpunkt):

’sum_sqrt’	$\sqrt{a^2 + b^2}$
’sum_abs’	$(\|a\| + \|b\|)/2$

Für eine 3×3 Sobel-Filterung (`Size` = 3) werden die Filter A und B direkt angewandt. Für größere Sobel-Filter (`Size` = 5,7,9 und 11) wird das Eingabebild dagegen zunächst mit dem Gaußfilter der Größe `Size`-2 geglättet. Somit ist

```
sobel_dir(I:Amp,Dir:FilterTyp,S:)
```

für `Size` > 3 äquivalent zu

```
gauss_image(I:G:S-2) ▷
sobel_dir(G:Amp,Dir:FilterType,3:).
```

Im Parameter `EdgeDirection` wird die Kantenrichtungen zurückgeliefert. Diese werden in 2-Grad-Schritten kodiert, d.h. eine Kantenrichtung von x Grad (bezogen auf die Horizontale) wird zu $x/2$ im entsprechenden Ergebnisbild. Außerdem wird auch die Richtung der Helligkeitsänderung berücksichtigt. Bezeichnet $[E_x, E_y]$ den Bildgradienten, ergeben sich folgende Kantenrichtungen r (zurückgeliefert als $r/2$) zwischen 0 und 359 Grad:

Helligkeitszunahme	E_x/E_y	Kantenrichtung r
von unten nach oben	$0/+$	0
von rechts unten nach links oben	$+/-$	$]0, 90[$
von rechts nach links	$+/0$	90
von rechts oben nach links unten	$+/+$	$]90, 180[$
von oben nach unten	$0/+$	180
von links oben nach rechts unten	$-/+$	$]180, 270[$
von links nach rechts	$+/0$	270
von links unten nach rechts oben	$-/-$	$]270, 360[$

In Bildpunkten mit Kantenamplitude 0 wird als Kantenrichtung der Wert 255 (undefinierte Richtung) zurückgeliefert.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild.
- ▷ **`EdgeAmplitude`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kantenamplitude.
- ▷ **`EdgeDirection`** (output_object) .....(multichannel-)image(-array) ⇝ *Hobject* : direction
  Kantenrichtung.
- ▷ **`FilterType`** (input_control) ..... string ⇝ *string*
  Filtertyp.
  Defaultwert : 'sum_abs'
  Werteliste : `FilterType` ∈ {'sum_abs', 'sum_sqrt'}
- ▷ **`Size`** (input_control) ..... integer ⇝ *integer*
  Filtergröße.
  Defaultwert : 3
  Werteliste : `Size` ∈ {3, 5, 7, 9, 11, 13}

Beispiel

```
read_image(Image,'fabrik')
sobel_dir(Image,Amp,Dir,'sum_abs',3)
threshold(Amp,Edg,128,255).
```

Ergebnis

`sobel_dir` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`sobel_dir` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `mean_image`, `anisotrope_diff`, `sigma_image`

Mögliche Nachfolgerfunktionen

`nonmax_suppression_dir`, `hysteresis_threshold`, `threshold`

Alternativen

`edges_image`, `frei_dir`, `kirsch_dir`, `prewitt_dir`, `robinson_dir`

Siehe auch

`roberts`, `laplace`, `highpass_image`, `bandpass_image`

Modul

Image filters

3.8 Linien

bandpass_image (Image : ImageBandpass : FilterType :)

Kantendetektion mittels Bandpaßfilterung.

bandpass_image dient als Kantenfilter. Die Prozedur führt eine lineare Filterung mit folgender Filtermatrix durch:

FilterType: 'lines'
In Gegensatz zu Kantenfiltern sobel_amp detektiert dieser Filter keine Grauwertkanten sondern Linien, also zwei dicht zusammenliegende Kanten.

$$\begin{array}{rrrrr} 0 & -2 & -2 & -2 & 0 \\ -2 & 0 & 3 & 0 & -2 \\ -2 & 3 & 12 & 3 & -2 \\ -2 & 0 & 3 & 0 & -2 \\ 0 & -2 & -2 & -2 & 0 \end{array}$$

Als Randbehandlungen werden die Grauwerte an den Objekträndern gespiegelt. Über- bzw. Unterlauf von Grauwerten wird beschnitten (255 bzw. 0). Die Ergebnisbilder werden in ImageBandpass abgelegt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebilder.
- ▷ **ImageBandpass** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Bandpaßgefilterte Bilder.
- ▷ **FilterType** (input_control) ..... string ⇝ *string*
  Filtertyp, derzeit nur 'lines'.
  Defaultwert : 'lines'
  Werteliste : FilterType ∈ {'lines'}

Beispiel

```
bandpass_image(Image,&LineImage,"lines");
threshold(LineImage,&Lines,60.0,255.0);
skeleton(Lines,&ThinLines);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert bandpass_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

bandpass_image ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

threshold, skeleton

Alternativen

convol_image, topographic_sketch, texture_laws

Siehe auch

highpass_image, gray_skeleton

Modul

Image filters

lines_facet (Image : Lines : MaskSize, Low, High, LightDark :)

Erkennen von Linien mit dem Facet-Modell.

Mit lines_facet können Linien (gekrümmt-lineare Strukturen) aus einem Bild Image extrahiert werden. Die extrahierten Linien werden in Lines als sub-pixel-genaue XLD-Konturen zurückgegeben. Der Parameter LightDark bestimmt, ob helle oder dunkle Linien extrahiert werden sollen.

Zur Extraktion werden in jedem Punkt des Bildes unter Verwendung des Facet-Modells die Parameter eines quadratischen Polynoms in x und y als Least-Squares-Anpassung an die Bilddaten berechnet. Der Parameter MaskSize bestimmt dabei die Größe des Bereichs, über den angepaßt wird. Große Werte für MaskSize sorgen für eine stärkere Glättung der Eingabedaten, können aber zu ungenauerer Lokalisation der Linien und zu stark schwingenden Linien führen. Mit Hilfe der Parameter des Polynoms wird in jedem Bildpunkt die Linienrichtung bestimmt. Bildpunkte, die ein lokales Maximum in der zweiten Richtungsableitung senkrecht zur Linienrichtung besitzen, werden als Linienpunkte markiert. Die gefundenen Linienpunkte werden hierauf zu Konturen zusammengefaßt. Dabei werden Punkte, die eine zweite Richtungsableitung, die größer als High ist, sofort als sichere Linienpunkte akzeptiert. Punkte, deren zweite Ableitung kleiner als Low ist, werden sofort verworfen. Alle Punkte, die eine zweite Ableitung zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind. Dies ist ähnlich zu einer Hysterese-Schwellenwert-Operation mit unbegrenzter Pfadlänge (siehe hysteresis_threshold). Allerdings wird diese Funktion nicht benutzt, da sonst keine sub-pixel-genaue Extraktion möglich wäre.

Für die Wahl der Schwellenwerte gilt sinngemäß das bei der Beschreibung von lines_gauss gesagte. Ein Wert von Sigma = 1.5 dort entspricht in etwa einer Maskengröße MaskSize von 5 hier.

Die extrahierten Linien werden in einer topologisch „sauberen“ Struktur in Lines zurückgeliefert. Das bedeutet, daß die Linien an Kreuzungspunkten sauber aufgetrennt werden.

lines_facet definiert folgende Attribute für jeden Linienpunkt:

’angle’ Der Winkel der Richtung senkrecht zur Linie
’response’ Die Größe der zweiten Ableitung

Diese Attribute können mit get_contour_attrib_xld ausgelesen werden.

Achtung

Je kleiner der Einzugsbereich MaskSize des Operators gewählt wird, desto mehr kurze, zerstückelte Linien werden gefunden. Dadurch kann die Verarbeitungszeit beträchtlich ansteigen.

Parameter

▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte / int1 / int2 / int4 / real
Eingabebild.

▷ **Lines** (output_object) ..... xld_cont-array ⇝ *Hobject*
Extrahierte Linien.

▷ **MaskSize** (input_control) ..... integer ⇝ *integer*
Größe der zu verwendenden Maske zur Bestimmung der Parameter des Facet-Modells.
Defaultwert : 5
Werteliste : MaskSize ∈ {3, 5, 7, 9, 11}

▷ **Low** (input_control) ..... number ⇝ *real* / integer
Untere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert : 3
Wertevorschläge : Low ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10}
Typischer Wertebereich : 0 ≤ Low ≤ 20
Empfohlene Schrittweite : 0.5
Restriktion : Low ≥ 0

▷ **High** (input_control) ..... number ⇝ *real* / integer
Obere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert : 8
Wertevorschläge : High ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25}
Typischer Wertebereich : 0 ≤ High ≤ 35
Empfohlene Schrittweite : 0.5
Restriktion : (High ≥ 0) ∧ (High ≥ Low)

▷ **LightDark** (input_control) . . . . . string ⇝ *string*
Helle oder dunkle Linien extrahieren.
Defaultwert : 'light'
Werteliste : LightDark ∈ {'dark', 'light'}

Beispiel

```
/* Detection of lines in an aerial image */
read_image(Image,'mreut4_3')
lines_facet(Image,Lines,5,3,8,'light')
disp_xld(Lines,WindowHandle).
```

Komplexität

Sei A die Anzahl von Pixeln in der Region von Image. Dann ist die Laufzeitkomplexität $O(A * MaskSize)$.

Sei $S = Width * Height$ die Anzahl der Pixel in Image. Dann benötigt lines_facet mindestens $55 * S$ Bytes an temporärem Speicher bei der Ausführung.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert lines_facet den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

lines_facet ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

gen_polygons_xld

Alternativen

lines_gauss

Siehe auch

bandpass_image, dyn_threshold, topographic_sketch

Literatur

A. Busch: „Fast Recognition of Lines in Digital Images Without User-Supplied Parameters". In H. Ebner, C. Heipke, K.Eder, eds., „Spatial Information from Digital Photogrammetry and Computer Vision", International Archives of Photogrammetry and Remote Sensing, Vol. 30, Part 3/1, pp. 91-97, 1994.

Modul

Sub-pixel operators

lines_gauss (Image : Lines : Sigma, Low, High, LightDark, ExtractWidth, CorrectPositions, CompleteJunctions :)

Erkennen von Linien und deren Breite.

Mit lines_gauss können Linien (gekrümmt-lineare Strukturen) aus einem Bild Image extrahiert werden. Die extrahierten Linien werden in Lines als sub-pixel-genaue XLD-Konturen zurückgegeben. Der Parameter LightDark bestimmt, ob helle oder dunkle Linien extrahiert werden sollen. Wenn ExtractWidth auf **'true'** gesetzt wird, so wird für jeden Linienpunkt die Linienbreite extrahiert. Wenn CorrectPositions auf **'true'** gesetzt wird, kompensiert lines_gauss die Effekte von asymmetrischen Linien (Linien mit unterschiedlichem Kontrast auf beiden Seiten der Linie), und korrigiert die Position und Breite der Linie. Dieser Parameter wird nur beachtet, falls ExtractWidth=**'true'**. Da der Linienextraktor aus differentialgeometrischen Gründen bestimmte Kreuzungspunkte nicht extrahieren kann, wird versucht, diese mit anderen Mitteln zu extrahieren, falls CompleteJunctions=**'true'** gesetzt wird.

Zur Extraktion werden in jedem Punkt des Bildes unter Verwendung der partiellen Ableitungen einer Gaußschen Glättungsmaske die Parameter eines quadratischen Polynoms in x und y berechnet. Der Parameter Sigma bestimmt dabei die Stärke der Glättung. Große Werte für Sigma sorgen für eine stärkere Glättung der Eingabedaten, können aber zu ungenauerer Lokalisation der Linien führen. Im Gegensatz zu lines_facet ist aber kein Schwingen der Linien für große Glättung zu beobachten. Ein weiterer Vorteil gegenüber lines_facet ist die genauere Lokalisation der Linien. Mit Hilfe der Parameter des Polynoms wird in jedem Bildpunkt die

Linienrichtung bestimmt. Bildpunkte, die ein lokales Maximum in der zweiten Richtungsableitung senkrecht zur Linienrichtung besitzen, werden als Linienpunkte markiert. Die gefundenen Linienpunkte werden hierauf zu Konturen zusammengefaßt. Dabei werden Punkte, die eine zweite Richtungsableitung, die größer als `High` ist, sofort als sichere Linienpunkte akzeptiert. Punkte, deren zweite Ableitung kleiner als `Low` ist, werden sofort verworfen. Alle Punkte, die eine zweite Ableitung zwischen diesen zwei Werten besitzen, werden akzeptiert, wenn sie durch einen Pfad mit sicheren Punkten verbunden sind. Dies ist ähnlich zu einer Hysterese-Schwellenwert-Operation mit unbegrenzter Pfadlänge (siehe `hysteresis_threshold`). Allerdings wird diese Funktion nicht benutzt, da sonst keine sub-pixel-genaue Extraktion möglich wäre.

Bei der Wahl der Schwellenwerte `High` und `Low` ist zu beachten, daß die zweite Richtungsableitung von der Amplitude und Breite der Linie, sowie von der Größe des Glättungsparameters `Sigma` abhängt. Dabei ist die Abhängigkeit von der Amplitude der Linie linear, d.h., je größer die Amplitude, desto größer die zweite Ableitung. Für die Breite der Linie besteht ein näherungsweise exponentieller Zusammenhang: Je breiter die Linie ist, desto kleiner ist die zweite Ableitung. Analoges gilt für die Abhängigkeit von `Sigma`: Je größer `Sigma` gewählt wird, desto kleiner wird die zweite Ableitung. Das bedeutet, daß für starke Glättung entsprechend kleinere Werte für `High` und `Low` gewählt werden müssen. Zwei Beispiele sollen dies verdeutlichen: Wenn im Bild 5 Pixel breite Linien mit einer Amplitude größer als 100 und einer Glättung `Sigma` = 1.5 extrahiert werden sollen, so sollte `High` größer als 14 gewählt werden. Wenn dagegen 10 Pixel breite Linien mit einer Amplitude größer als 100 und einer Glättung `Sigma` = 3 gefunden werden sollen, so sollte `High` größer als 3.5 gesetzt werden. Als Werte für `Low` empfehlen sich Werte zwischen 0.25 `High` und 0.5 `High`.

Die extrahierten Linien werden in einer topologisch „sauberen“ Struktur in `Lines` zurückgeliefert. Das bedeutet, daß die Linien an Kreuzungspunkten sauber aufgetrennt werden.

`lines_gauss` definiert folgende Attribute für jeden Linienpunkt, falls `ExtractWidth` auf **’false’** gesetzt wurde:

’angle’ Der Winkel der Richtung senkrecht zur Linie
’response’ Die Größe der zweiten Ableitung

Falls `ExtractWidth` auf **’true’** und `CorrectPositions` auf **’false’** gesetzt wurde, werden zusätzlich noch folgende Attribute definiert:

’width_left’ Die Linienbreite links von der Linie
’width_right’ Die Linienbreite rechts von der Linie

Zusätzlich werden noch folgende Linienattribute gesetzt, falls `CorrectPositions` auf **’true’** gesetzt wurde:

’asymmetry’ Die Asymmetrie des Linienpunkts
’contrast’ Der Kontrast des Linienpunkts

Dabei ist der Wert der Asymmetrie so gewählt, daß er positive Werte annimmt, wenn die Asymmetrie, d.h. der kleinere Gradient, auf der rechten Seite der Linie liegt, und daß er negative Werte annimmt, wenn die Asymmetrie auf der linken Seite der Linie liegt. Diese Attribute können mit `get_contour_attrib_xld` ausgelesen werden.

Achtung

Im allgemeinen, aber besonders, wenn die Linienbreite extrahiert werden soll, sollte `Sigma` $\geq w/\sqrt{3}$ gewählt werden, wobei w die Breite (der halbe Durchmesser) der zu extrahierenden Linien im Bild ist. Als unterste Grenze *muß* `Sigma` $\geq w/2.5$ gewählt werden. Falls z.B. Linien mit einer Breite von 4 Pixel (Durchmesser 8 Pixel) extrahiert werden sollen, sollte `Sigma` ≥ 2.3 gewählt werden.

Parameter

- ▷ **`Image`** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Eingabebild.
- ▷ **`Lines`** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Extrahierte Linien.
- ▷ **`Sigma`** (input_control) ..... number ⇝ *real* / integer
  Stärke der Gaußschen Glättung.
  Defaultwert : 1.5
  Wertevorschläge : `Sigma` $\in \{1, 1.2, 1.5, 1.8, 2, 2.5, 3, 4, 5\}$
  Typischer Wertebereich : $0.7 \leq$ `Sigma` ≤ 20
  Empfohlene Schrittweite : 0.1

▷ **Low** (input_control) ..... number ⇝ *real* / integer
Untere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert : 3
Wertevorschläge : Low ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10}
Typischer Wertebereich : 0 ≤ Low ≤ 20
Empfohlene Schrittweite : 0.5
Restriktion : Low ≥ 0

▷ **High** (input_control) ..... number ⇝ *real* / integer
Obere Schwelle für Hysterese-Schwellenwertoperation.
Defaultwert : 8
Wertevorschläge : High ∈ {0, 0.5, 1, 2, 3, 4, 5, 8, 10, 12, 15, 18, 20, 25}
Typischer Wertebereich : 0 ≤ High ≤ 35
Empfohlene Schrittweite : 0.5
Restriktion : (High ≥ 0) ∧ (High ≥ Low)

▷ **LightDark** (input_control) ..... string ⇝ *string*
Helle oder dunkle Linien extrahieren.
Defaultwert : 'light'
Werteliste : LightDark ∈ {'dark', 'light'}

▷ **ExtractWidth** (input_control) ..... string ⇝ *string*
Soll die Linienbreite extrahiert werden?
Defaultwert : 'true'
Werteliste : ExtractWidth ∈ {'true', 'false'}

▷ **CorrectPositions** (input_control) ..... string ⇝ *string*
Sollen Linienposition und -breite korrigiert werden?
Defaultwert : 'true'
Werteliste : CorrectPositions ∈ {'true', 'false'}

▷ **CompleteJunctions** (input_control) ..... string ⇝ *string*
Sollen die Kreuzungspunkte vervollständigt werden?
Defaultwert : 'true'
Werteliste : CompleteJunctions ∈ {'true', 'false'}

Beispiel

```
/* 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).
```

Komplexität

Sei A die Anzahl von Pixeln in der Region von Image. Dann ist die Laufzeitkomplexität $O(A * Sigma)$.

Sei $S = Width * Height$ die Anzahl der Pixel in Image. Dann benötigt lines_gauss mindestens $55 * S$ Bytes an temporärem Speicher bei der Ausführung.

Ergebnis

Sind die Parameterwerte korrekt und tritt kein Fehler während der Berechnung auf, liefert lines_gauss den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

lines_gauss ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

gen_polygons_xld

Alternativen

lines_facet

Siehe auch

bandpass_image, dyn_threshold, topographic_sketch

Literatur

C. Steger: “Extracting Curvilinear Structures: A Differential Geometric Approach”. In B. Buxton, R. Cipolla, eds.,

"Fourth European Conference on Computer Vision", Lecture Notes in Computer Science, Volume 1064, Springer Verlag, pp. 630-641, 1996.

C. Steger: "Extraction of Curved Lines from Images". In "13th International Conference on Pattern Recognition", Volume II, pp. 251-255, 1996.

C. Steger: "An Unbiased Detector of Curvilinear Structures". Technical Report FGBV-96-03, Forschungsgruppe Bildverstehen (FG BV), Informatik IX, Technische Universität München, July 1996.

Modul

Sub-pixel operators

3.9 Match

adapt_template (Image : : TemplateID :)

Anpassen eines Templates an die Größe eines Bildes.

adapt_template dient zur Anpassung eines Templates, das mit create_template erzeugt wurde, an die Größe eines Bildes. Vor der ersten Verwendung eines Templates mit Bildern einer anderen Größe kann adapt_template aufgerufen werden um Laufzeit beim ersten Aufruf des Matching Operators zu sparen. Wird adapt_template nicht explizit verwendet, erfolgt der Aufruf implizit beim ersten Matching mit einer anderen Bildgröße. Der Inhalt des Bildes ist irrelevant. Es wird nur die Breite von Image verwendet.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Bild, das die Größe des späteren Matching vorgibt.
- ▷ **TemplateID** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert adapt_template den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

adapt_template wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

create_template, create_template_rot, read_template

Mögliche Nachfolgerfunktionen

set_reference_template, best_match, fast_match, fast_match_mg, set_offset_template, best_match_mg, best_match_pre_mg, best_match_rot, best_match_rot_mg

Modul

Template matching

best_match (Image : : TemplateID, MaxError, SubPixel : Row, Column, Error)

Suche des besten Matching zwischen einem Template und einem Bild.

best_match führt ein Matching zwischen dem Template von TemplateID und Image durch. Dabei wird das Template so über die Punkte von Image geschoben, daß es immer vollständig innerhalb von Image liegt. best_match arbeitet ähnlich wie fast_match mit der Erweiterung, daß jedesmal wenn eine Position mit einem geringeren Matching-Fehler gefunden wurde, der Parameter MaxError intern entsprechend angepaßt (d.h. verkleinert) wird, um die Rechenzeit zu verkürzen. Abhängig vom Parameter SubPixel wird die Position in Subpixelgenauigkeit ausgegeben. Das Matching-Kriterium („displaced frame difference") ist wie folgt definiert:

$$error[row, col] = \frac{\sum_{u,v} |\texttt{Image}[row-u, col-v] - \texttt{TemplateID}[u,v]|}{area(TemplateID)}$$

Die Laufzeit des Verfahrens hängt von der Größe des Definitionsbereiches von `Image` ab. Es ist daher wichtig, den Definitionsbereich möglichst einzugrenzen, d.h. den Operator nur in einer möglichst eng umrissenen „region of interest“ anzuwenden. Der Parameter `MaxError` legt den maximalen Fehler fest, den die gesuchte Position haben darf. Um so kleiner dieser Wert ist, um so schneller läuft das Verfahren.

`Row` und `Column` liefern die Position des Best-Match, wobei `Error` die mittlere Abweichung der Grauwerte angiebt.

Falls keine Position mit einem Fehler unter `MaxError` gefunden wurde, wird die Position $(0, 0)$ und ein Matching-Fehler von 255 für `Error` geliefert. In diesem Fall müs der Wert für `MaxError` größer gewählt werden.

Der maximale Fehler der Position (ohne Rauschen) beträgt im sub-Pixel Modus 0.1 Pixel. Der mittlere Fehler betra”gt 0.03 Pixel.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **`TemplateID`** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.
- ▷ **`MaxError`** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert : 20
  Wertevorschläge : `MaxError` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 0, 70}
  Typischer Wertebereich : $0 \leq$ `MaxError` ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 3
- ▷ **`SubPixel`** (input_control) ..... string ⇝ *string*
  Subpixel Genauigkeit falls **’true’**.
  Defaultwert : ’false’
  Werteliste : `SubPixel` ∈ {’true’, ’false’}
- ▷ **`Row`** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenposition des Best-Match.
- ▷ **`Column`** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenposition des Best-Match.
- ▷ **`Error`** (output_control) ..... real(-array) ⇝ *real*
  Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `best_match` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`best_match` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`create_template`, `read_template`, `set_offset_template`, `set_reference_template`, `adapt_template`, `draw_region`, `draw_rectangle1`, `reduce_domain`

Alternativen

`fast_match`, `fast_match_mg`, `best_match_mg`, `best_match_pre_mg`, `best_match_rot`, `best_match_rot_mg`, `exhaustive_match`, `exhaustive_match_mg`

Modul

Template matching

`best_match_mg` `( Image : : TemplateID, MaxError, SubPixel, NumLevels, WhichLevels : Row, Column, Error )`

Searching the best grayvalue matches in a pyramid.

best_match_mg führt eine Suche nach dem besten Grauwert-Match in einer Pyramide durch. best_match_mg arbeitet analog zu best_match, es wird jedoch die Suche durch die Verwendung einer Auflösungspyramide beschleunigt. Als Eingabe dient ein Bild, eventuell mit eingeschränktem Definitionsbereich. Der Paramter MaxError gibt den maximalen Fehler vor, der bei dem Mustervergleich auftreten darf. Bei einem kleinen Wert beschleunigt sich der Operator, es kann jedoch vokommen, daß das Muster nicht gefunden wird. Der Wert von MaxError muß gegenüber best_match größer gewählt werden, da der Matching-Fehler auf höheren Pyramidenebenen oft größer ist.

Der Parameter SubPixel legt fest, ob das Ergebnis mit subpixel Genauigkeit bestimmt werden soll. SubPixel bestimmt wieviele Pyramidenstufen für die Suche verwendet werden. Hat SubPixel den Wert **1**, dann ist das Verahren identisch mit best_match, d.h. es wird nur auf den Originaldaten gerechnet. Für Werte größer als **1**, beginnt das Verfahren die Suche bei der geringsten Auflösung und sucht dort die Position mit dem geringsten Fehler. In der nächst höheren Auflösung wird dann die gefundene Position verfeinert. Dies wird bis zur höchsten Auflösung (Originaldaten) vorgesetzt (WhichLevels = **'all'**). Als alternatives Verfahren kann der Modus WhichLevels mit den Wert **'original'** verwendet werden. Hierbei wird auf allen Ebenen mit reduzierter Auflösung nicht nur der Punkt mit dem geringsten Fehler, sondern alle Punkte die unterhalb von MaxError liegen weiter untersucht. Dieses Verfahren ist langsamer, aber die Gefahr, daß der richtige Punkt nicht gefunden wird, ist geringer. Gegebenenfalls kann in diesem Modus eine geringere Auflösung, d.h. eine höhere Pyramidenstufe verwendet werden, was zu einer verbesserten Laufzeit führt. Neben den Modi **'all'** und **'original'** für WhichLevels kann die gewünschte Ebene bei der zwischen den Verfahren umgeschaltet werden soll auch explizit angegeben werden. Hier entspricht **0** dem Wert **'original'** und NumLevels - 1 entspricht dem Wert **'all'**. Ein Wert zwischen diesen beiden Extremen ist in den meißten Fällen ein guter Kompromis zwischen Stabilität und Laufzeit. Ein größerer Wert für WhichLevels verkürzt die Laufzeit, während ein kleinerer Wert das Verhalten stabiler macht. Der Wert für NumLevels muß gleich oder kleiner sein als die Anzahl der Ebenen die bei der Erzeugung des Templates verwendet wurde.

Die gefunden Position des Musters wird in Row, Column übergeben. Der hierbei aufgetretene Fehler steht in Error. Wenn kein Punkt unterhalb von MaxError gefunden wird hat Error den Wert **255** und Row und Column haben den Wert **0**. Der maximale Fehler der Position (ohne Rauschen) beträgt im sub-Pixel Modus 0.1 Pixel. Der mittlere Fehler betra"gt 0.03 Pixel.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **TemplateID** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.
- ▷ **MaxError** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert : 30
  Wertevorschläge : MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
  Typischer Wertebereich : 0 ≤ MaxError ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 3
- ▷ **SubPixel** (input_control) ..... string ⇝ *string*
  Subpixel Genauigkeit falls **'true'**.
  Defaultwert : 'false'
  Werteliste : SubPixel ∈ {'true', 'false'}
- ▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
  Anzahl der verwendeten Auflösungsebenen.
  Defaultwert : 4
  Werteliste : NumLevels ∈ {1, 2, 3, 4, 5, 6}
- ▷ **WhichLevels** (input_control) ..... integer ⇝ *integer* / string
  Auflösungsebene bis zu der die Methode „best match" verwendet wird.
  Defaultwert : 2
  Wertevorschläge : WhichLevels ∈ {'all', 'original', 0, 1, 2, 3, 4, 5, 6}
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenposition des Best-Match.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenposition des Best-Match.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_mg den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

best_match_mg ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain, set_reference_template, set_offset_template

Alternativen

fast_match, fast_match_mg, best_match, best_match_pre_mg, best_match_rot, best_match_rot_mg, exhaustive_match, exhaustive_match_mg

Modul

Template matching

best_match_pre_mg (ImagePyramid : : TemplateID, MaxError, SubPixel, NumLevels, WhichLevels : Row, Column, Error)

Searching the best grayvalue matches in a pre generated pyramid.

best_match_pre_mg führt eine Suche nach dem besten Grauwert-Match in einer Pyramide durch. best_match_pre_mg arbeitet analog zu best_match_mg, mit dem Unterschied, daß die Berechnung der Pyramid vor dem Aufruf mit gen_gauss_pyramid durchgeführt wurde. Dies verkürzt die Laufzeit, falls das Matching mehrfach durchgeführt werden soll, oder die Pyramide auch bei anderen Operationen genötigt wird. Die Pyramide muß mit dem Verkleinerungsfaktor **0.5** und dem Modus **’constant’** erzeugt werden.

Parameter

▷ **ImagePyramid** (input_object) . . . . . image-array ⇝ *Hobject* : byte
Bildpyramide, in dem das Pattern gefunden werden soll.

▷ **TemplateID** (input_control) . . . . . template ⇝ *integer*
Nummer des Templates.

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert : 30
Wertevorschläge : MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
Typischer Wertebereich : 0 ≤ MaxError ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 3

▷ **SubPixel** (input_control) . . . . . string ⇝ *string*
Subpixel Genauigkeit falls **’true’**.
Defaultwert : ’false’
Werteliste : SubPixel ∈ {’true’, ’false’}

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer*
Anzahl der verwendeten Auflösungsebenen.
Defaultwert : 3
Werteliste : NumLevels ∈ {1, 2, 3, 4, 5, 6}

▷ **WhichLevels** (input_control) . . . . . integer ⇝ *integer* / string
Auflösungsebene bis zu der die Methode „best match“ verwendet wird.
Defaultwert : ’original’
Wertevorschläge : WhichLevels ∈ {’all’, ’original’, 0, 1, 2, 3, 4, 5, 6}

▷ **Row** (output_control) . . . . . point.y ⇝ *real*
Zeilenposition des Best-Match.

▷ **Column** (output_control) . . . . . point.x ⇝ *real*
Spaltenposition des Best-Match.

▷ **Error** (output_control) . . . . . real ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `best_match_pre_mg` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`best_match_pre_mg` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gen_gauss_pyramid`, `create_template`, `read_template`, `adapt_template`, `draw_region`, `draw_rectangle1`, `reduce_domain`, `set_reference_template`

Alternativen

`fast_match`, `fast_match_mg`, `exhaustive_match`, `exhaustive_match_mg`

Modul

Template matching

best_match_rot (Image : : TemplateID, AngleStart, AngleExtend, MaxError, SubPixel : Row, Column, Angle, Error)

Suche des besten Matching zwischen einem Template und einem Bild mit Rotation.

`best_match_rot` führt ein Matching zwischen dem Template von `TemplateID` und `Image` durch. Es arbeitet analog zu `best_match` mit der Erweiterung, daß das Muster gedreht vorliegen kann. Die Parameter `AngleStart` und `AngleExtend` legen die maximale Rotation des Musters fest: `AngleStart` gibt größte Rotation gegen den Uhrzeiger an und `AngleExtend` die größte Rotation im Uhrzeigersinn relativ zu diesem Winkel. Beide Werte müssen kleiner oder gleich den entsprechenden Werte des Template bei dessen Erzeugung sein (siehe `create_template_rot`). Als Erweiterung gegenüber `best_match` liefert `best_match_rot` zu zusätzlichen Ausgabeparameter `Angle` der die gefundenen Rotationswinkel des Musters angibt. Die Genauigkeit hängt von dem Parameter AngleStep von `create_template_rot` ab. Im Fall von `SubPixel` = **’true’** wird die Position und der Winkel mit „subpixel“ Genauigkeit bestimmt.

Parameter

▷ **Image** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte
Eingabebild, in dem das Pattern gefunden werden soll.

▷ **TemplateID** (input_control) . . . . . template ⇝ *integer*
Nummer des Templates.

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinster auftretende Rotation des Musters.
Defaultwert : -0.39
Wertevorschläge : `AngleStart` ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtend** (input_control) . . . . . angle.rad ⇝ *real*
Maximale posivite Abweichung von `AngleStart`.
Defaultwert : 0.79
Wertevorschläge : `AngleExtend` ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
Restriktion : `AngleExtend` > `0`

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert : 30
Wertevorschläge : `MaxError` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
Typischer Wertebereich : 0 ≤ `MaxError` ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 3

▷ **SubPixel** (input_control) ..... string ⇝ *string*
Subpixel Genauigkeit falls **'true'**.
Defaultwert : 'false'
Werteliste : SubPixel ∈ {'true', 'false'}

▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
Zeilenposition des Best-Match.

▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
Spaltenposition des Best-Match.

▷ **Angle** (output_control) ..... angle.rad(-array) ⇝ *real*
Rotationswinkel des Musters.

▷ **Error** (output_control) ..... real(-array) ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_rot den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

best_match_rot ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

create_template_rot, read_template, set_offset_template, set_reference_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Alternativen

best_match_rot_mg

Siehe auch

best_match, best_match_mg

Modul

Template matching

best_match_rot_mg (Image : : TemplateID, AngleStart, AngleExtend, MaxError, SubPixel, NumLevels : Row, Column, Angle, Error)

Suche des besten Matching zwischen einem Template und einer Pyramide mit Rotation.

best_match_rot_mg führt ein Matching zwischen dem Template von TemplateID und Image durch. Es arbeitet analog zu best_match_mg mit der Erweiterung, daß das Muster wie bei best_match_rot gedreht vorliegen kann. Die Parameter AngleStart und AngleExtend legen die maximale Rotation des Musters fest: AngleStart gibt größte Rotation gegen den Uhrzeiger an und AngleExtend die größte Rotation im Uhrzeigersinn relativ zu diesem Winkel. Beide Werte müssen kleiner oder gleich den entsprechenden Werte des Template bei dessen Erzeugung sein (siehe create_template_rot). Als Erweiterung gegenüber best_match_mg liefert best_match_rot_mg zu zusätzlichen Ausgabeparameter Angle der den gefundenen Rotationswinkel des Musters angibt.

Der Wert von MaxError muß gegenüber best_match_rot größer gewählt werden, da der Matching-Fehler auf höheren Pyramidenebenen oft größer ist.

Im Fall von SubPixel = **'true'** wird die Position und der Winkel mit „subpixel" Genauigkeit bestimmt.

Der Wert von NumLevels muß kleiner oder gleich dem Wert sein, der bei der Erzeugung des Templates verwendet wurde.

Parameter

▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
Eingabebild, in dem das Pattern gefunden werden soll.

▷ **TemplateID** (input_control) ..... template ⇝ *integer*
Nummer des Templates.

▷ **AngleStart** (input_control) . . . . . angle.rad ⇝ *real*
Kleinster auftretende Rotation des Musters.
Defaultwert : -0.39
Wertevorschläge : AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}

▷ **AngleExtend** (input_control) . . . . . angle.rad ⇝ *real*
Maximale posivite Abweichung von AngleStart.
Defaultwert : 0.79
Wertevorschläge : AngleExtend ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
Restriktion : AngleExtend > 0

▷ **MaxError** (input_control) . . . . . real ⇝ *real*
Maximale mittlere Differenz der Grauwerte.
Defaultwert : 40
Wertevorschläge : MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
Typischer Wertebereich : 0 ≤ MaxError ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **SubPixel** (input_control) . . . . . string ⇝ *string*
Subpixel Genauigkeit falls **'true'**.
Defaultwert : 'false'
Werteliste : SubPixel ∈ {'true', 'false'}

▷ **NumLevels** (input_control) . . . . . integer ⇝ *integer*
Anzahl der verwendeten Auflösungsebenen.
Defaultwert : 3
Werteliste : NumLevels ∈ {1, 2, 3, 4, 5, 6}

▷ **Row** (output_control) . . . . . point.y(-array) ⇝ *real*
Zeilenposition des Best-Match.

▷ **Column** (output_control) . . . . . point.x(-array) ⇝ *real*
Spaltenposition des Best-Match.

▷ **Angle** (output_control) . . . . . angle.rad(-array) ⇝ *real*
Rotationswinkel des Musters.

▷ **Error** (output_control) . . . . . real(-array) ⇝ *real*
Mittlere Abweichung der Grauwerte des Best-Match.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert best_match_rot_mg den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

best_match_rot_mg ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

create_template_rot, set_reference_template, set_offset_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Alternativen

best_match_rot, best_match_mg

Siehe auch

fast_match

Modul

Template matching

clear_template (: : TemplateID :)

Freigabe des Speichers eines Templates.

clear_template gibt den Speicher eines Templates, das mit create_template oder create_template_rot angelegt wurde wieder frei. Das Template kann nach dem Aufruf nicht mehr verwendet werden. Der Wert von TemplateID ist ungültig. Die Nummer kann jedoch bei nachfolgenden Aufrufen von create_template oder create_template_rot wieder vergeben werden.

Parameter

▷ **TemplateID** (input_control) . . . . . template ⇝ *integer*
Nummer des Templates.

Ergebnis

Ist die Nummer des Templates gültig, dann liefert clear_template den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

clear_template ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_template, create_template_rot, read_template, write_template

Modul

Template matching

corner_response (Image : ImageCorner : Size, Weight :)

Suche von Ecken in Bildern.

corner_response ist ein Operator, um Grauwertecken in einem Bild zu extrahieren. Die Formel für die Berechnung lautet:

$$\begin{aligned} R(x,y) &= A(x,y) \cdot B(x,y) - C^2(x,y) - \texttt{Weight} \cdot (A(x,y) + B(x,y))^2 \\ A(x,y) &= W(u,v) * (\nabla_x I(x,y))^2 \\ B(x,y) &= W(u,v) * (\nabla_y I(x,y))^2 \\ C(c,y) &= W(u,v) * (\nabla_x I(x,y) \nabla_y I(x,y)) \end{aligned}$$

wobei I das Eingabe- und R das Ergebnisbild der Filterung darstellt. Zur Glättung (W) wird gauss_image, zur Berechnung der Ableitung (∇) wird sobel_amp verwendet.

Die Corner-Response-Funktion ist invariant gegenüber Rotation. Um eine günstige Abhängigkeit der Funktion $R(x,y)$ von den lokalen Gradienten zu erhalten, ist der Parameter Weight auf 0.04 zu setzen. Hiermit ergeben sich nur an Grauwertecken positive Werte für $R(x,y)$, während gerade Kanten negative Werte erhalten. Der Ausgabebildtyp ist identisch mit dem Eingabebildtyp. Bei Byte-Bildern als Eingabe werden also die negativen Werte auf 0 gesetzt. Falls dies nicht erwünscht ist, sollte das Bild mit convert_image_type in ein Real- oder Int2-Bild konvertiert werden.

Parameter

▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / real
Eingabebild.

▷ **ImageCorner** (output_object) . . . . . multichannel-image(-array) ⇝ *Hobject* : byte / int2 / real
Ergebnis der Filterung.
Parameteranzahl : ImageCorner = Image

▷ **Size** (input_control) . . . . . integer ⇝ *integer*
Gewünschte Filtergröße der Gaußmaske.
Defaultwert : 3
Werteliste : Size ∈ {3, 5, 7, 9, 11}

▷ **Weight** (input_control) . . . . . real ⇝ *real*
Gewichtung.
Defaultwert : 0.04
Typischer Wertebereich : 0.0 ≤ Weight ≤ 0.3
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.01

Beispiel

```
read_image(&Fabrik,"fabrik");
corner_response(Fabrik,&CornerResponse,3,0.04);
local_max(CornerResponse,&LocalMax);
disp_image(Fabrik,WindowHandle);
set_color(WindowHandle,"red");
disp_region(LocalMax,WindowHandle);
```

Parallelisierungsinformation

`corner_response` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`local_max`, `threshold`

Siehe auch

`gauss_image`, `sobel_amp`, `convert_image_type`

Literatur

C.G. Harris, M.J. Stephens, “A combined corner and edge detector”’; Proc. of the 4th Alvey Vision Conference; August 1988; pp. 147-152.

H. Breit, “Bestimmung der Kameraeigenbewegung und Gewinnung von Tiefendaten aus monokularen Bildfolgen”; Diplomarbeit am Lehrstuhl für Nachrichtentechnik der TU München; 30. September 1990.

Modul

Image filters

create_template (Template : : FirstError, NumLevel, Optimize, GrayValues : TemplateID)

Aufbereiten eines Musters für Template Matching.

`create_template` bereitet ein Muster (`Template`), das als Bild übergeben wird, für das Template Matching vor. Dem Template wird nach der Transformation eine Nummer zugewiesen (`TemplateID`), die bei der weiteren Verarbeitung verwendet wird. Die Form und Größe von `Template` sind beliebig. Es ist jedoch zu beachten, daß das Matching nur in dem Teil des Bildes ausgeführt wird, in dem `Template` vollständig hineinpaßt.

Das Template sollte so gewählt werden, daß keine Grauwerte des Hintergrundes enthalten sind. Hierbei kann ausgenutzt werden, daß die Form des Template nicht auf Rechtecke beschränkt ist. Zu Gewinnung des Templates können Segmentationoperationen wie `threshold` verwendet werden. Soll die Position des Templates später mit sub-Pixel-Genauigkeit bestimmt werden, dann muß `Template` zusätzlich um ein Pixel kleiner sein als das eigenliche Muster. Die kann z.B. mit dem Operator `erosion_circle` realsiert werden.

Der Parameter `NumLevel` gibt die Anzahl der Pyramidenebenen an (`NumLevel` = 1 bedeutet, daß nur die Originalgrauwerte verwendet werden), die beim Matching maximal verwendet werden können. Die Anzahl kann bei Matching aber kleiner sein als dieser Wert. Falls das Template durch das Verkleinern zu klein wird, reduziert sich die maximale Anzahl von Pyramidenstufen automatisch (ohne Fehlermeldung!).

Der Parameter `GrayValues` legt fest, ob die Originalgrauwerte (**’original’** bzw. **’normalized’**) oder die Kantenamplitude (**’gradient’** bzw. **’sobel’**) verwendet werden soll. Bei **’original’** wird die Summe der Differenzen als Merkmal verwendet. Dieses Verfahren ist abhängig von der Beleuchtung. Bei **’normalized’** wird die normierte Summe der Differenzen verwendet. Dieses Merkmal ist unabhängig gegenüber Grauwertänderungen aber etwas langsamer ist nicht ganz so stabil. Falls die Grauwerte sich nicht ändern ist **’original’** vorzuziehen. Die Kantenamplitude ist eine weitere Methode unabhängig von der Beleuchtung zu sein. Der Nachteil ist die höhere Laufzeit und eine stärkere Abhängigkeit gegenüber Formänderungen. Der Modus **’gradient’** ist etwas schneller aber auch empfindlicher gegen Rauschen. Der maximale Fehler beim Matching muß bei der Kantenamplitude typischerweise größer gewählt werden. Als Alternative zu den Gradientenverfahren kann der Operator `set_offset_template` verwendet werden, falls die Beleuchtungsänderung bekannt ist.

Der Parameter `Optimize` legt fest ob das Template für eine schnellere Verarbeitung zusätzlich optimiert wird. Dies verlangsamt die Erzeugung des Templates, verkürzt aber das Matching. Zusätzlich wird durch die Optimierung das Matching stabiler, d.h. der Verlust einer korrekten Matches wird unwahrscheinlicher.

Als Referenzwert für das Matching wird der Schwerpunkt verwendet. D.h. wenn man das Template auf das Originalbild anwendet wird der Schwerpunkt zurückgegeben. Dieser Voreinstellung kann mit `set_reference_template` angepaßt werden.

Im sub-Pixel Modus wird eine zusätzliche Positionskorrektur verwendet: Das Template wird auf die Originaldaten angewandt und die hierbei entstehende Abweichung der gefundenen Position vom Schwerpunkt wird als Korrekturvektor mit verwendet. Dies ist für Muster die in einem konstrastreichen Kontext liegen von Bedeutung. Bei den meisten Muster ist dieser Korrekturvektor Null.

Das Pattern muß nach Gebrauch mit `clear_template` gelöscht werden und den Speicher wieder freizugeben. Vor der Verwendung kann das Template, das bildformatunabhängig gespeichert ist, noch mit `adapt_template` auf die Größe eines konkreten Bildes angepaßt werden.

Parameter

- ▷ **`Template`** (input_object) ..... image ⇝ *Hobject* : byte
  Eingabebild, dessen Definitionsbereich für das Pattern Matching aufbereitet wird.
- ▷ **`FirstError`** (input_control) ..... integer ⇝ *integer*
  Noch nicht verwendet.
  Defaultwert : 255
  Werteliste : `FirstError` ∈ {255}
- ▷ **`NumLevel`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert : 4
  Werteliste : `NumLevel` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **`Optimize`** (input_control) ..... string ⇝ *string*
  Art der Optimierung.
  Defaultwert : 'sort'
  Werteliste : `Optimize` ∈ {'none', 'sort'}
- ▷ **`GrayValues`** (input_control) ..... string ⇝ *string*
  Art der Grauwerte.
  Defaultwert : 'original'
  Werteliste : `GrayValues` ∈ {'original', 'normalized', 'gradient', 'sobel'}
- ▷ **`TemplateID`** (output_control) ..... template ⇝ *integer*
  Nummer des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_template` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`create_template` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`draw_region`, `reduce_domain`, `threshold`

Mögliche Nachfolgerfunktionen

`adapt_template`, `set_reference_template`, `clear_template`, `write_template`, `set_offset_template`, `best_match`, `best_match_mg`, `fast_match`, `fast_match_mg`

Alternativen

`create_template_rot`, `read_template`

Modul

Template matching

`create_template_rot` `( Template : : NumLevel, AngleStart, AngleExtend, AngleStep, Optimize, GrayValues : TemplateID )`

Aufbereiten eines Musters für Template Matching mit Rotation.

`create_template_rot` bereitet ein Muster, das als Bild übergeben wird, für das Template Matching vor. Als Erweiterung gegenüber `create_template` kann das Matching später mit rotierten Mustern ausgeführt werden.

Die Parameter AngleStart und AngleExtend legen die maximale Rotation des Musters fest: AngleStart gibt die größte Rotation gegen den Uhrzeiger an und AngleExtend die größte Rotation im Uhrzeigersinn relativ zu diesem Winkel. AngleExtend muß folglich immer kleiner als 2π sein. Mit dem Parameter AngleStep wird die maximale Winkelauflösung (auf der untersten Pyramidenebene) festgelegt.

Es ist zu beachten, daß alle möglichen Rotationen bei der Erzeugung des Templates berechnet werden um Laufzeit bei Matching zu sparen. Dies führt zu einer entsprechend hohen Laufzeit von create_template_rot und einem hohen Speicherbedarf des erzeugten Templates. Der Speichebedarf hängt von AngleExtend und AngleStep ab. Die Anzahl der Pyramidenstufen kann dagegen vernachlässigt werden. Falls A die Fläche von Template ist, dann ist der Speicherbedarf M in Byte etwa:

$$M = \frac{A * 12 * \texttt{AngleExtend}}{\texttt{AngleStep}}$$

Dem Template wird nach der Transformation eine Nummer zugewiesen (TemplateID), die bei der weiteren Verarbeitung verwendet wird.

Die Beschreibung der weiteren Parameter ist bei create_template zu finden.

Achtung

Es ist zu beachten, daß für jede Rotation Muster angelegt werden. Dies erhöht bei einer feinen Auflösung den Speicherbedarf entsprechend.

Parameter

- ▷ **Template** (input_object) ..... image ⇝ *Hobject* : byte
  Eingabebild, dessen Definitionsbereich für das Pattern Matching aufbereitet wird.
- ▷ **NumLevel** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert : 4
  Werteliste : NumLevel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
  Kleinster auftretende Rotation des Musters.
  Defaultwert : -0.39
  Wertevorschläge : AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtend** (input_control) ..... angle.rad ⇝ *real*
  Maximale posivite Abweichung von AngleStart.
  Defaultwert : 0.79
  Wertevorschläge : AngleExtend ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
  Restriktion : AngleExtend > 0
- ▷ **AngleStep** (input_control) ..... angle.rad ⇝ *real*
  Schrittweite (Winkelgenauigkeit) des Matchings.
  Defaultwert : 0.0982
  Wertevorschläge : AngleStep ∈ {0.3927, 0.1963, 0.0982, 0.0491, 0.0245}
  Restriktion : AngleStep > 0
- ▷ **Optimize** (input_control) ..... string ⇝ *string*
  Art der Optimierung.
  Defaultwert : 'sort'
  Werteliste : Optimize ∈ {'none', 'sort'}
- ▷ **GrayValues** (input_control) ..... string ⇝ *string*
  Art der Grauwerte.
  Defaultwert : 'original'
  Werteliste : GrayValues ∈ {'original', 'normalized', 'gradient', 'sobel'}
- ▷ **TemplateID** (output_control) ..... template ⇝ *integer*
  Nummer des Templates.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_template_rot den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

create_template_rot ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`draw_region`, `reduce_domain`, `threshold`

Mögliche Nachfolgerfunktionen

`best_match_rot`, `best_match_rot_mg`, `adapt_template`, `set_reference_template`, `clear_template`, `set_offset_template`, `write_template`

Alternativen

`create_template`

Modul

Template matching

`exhaustive_match` (Image, RegionOfInterest, ImageTemplate : ImageMatch : Mode :)

Matching zwischen einem Template und einem Bild.

`exhaustive_match` führt ein Matching zwischen `ImageTemplate` und `Image` innerhalb des Bildbereichs `RegionOfInterest` durch. Dabei wird `ImageTemplate` über alle Punkte von `Image`, die innerhalb der `RegionOfInterest` liegen, geschoben. Abhängig vom Parameter `Mode` wird ein Matching-Kriterium berechnet. Die Ergebniswerte werden in `ImageMatch` abgelegt.

Folgende Matching-Kriterien (`Mode`) stehen zur Verfügung:

'norm_correlation'

$$\texttt{ImageMatch}[i][j] = 255 \cdot \frac{\sum_{u,v} (\texttt{Image}[i-u][j-v] \cdot \texttt{ImageTemplate}[l-u][c-v])}{\sqrt{\sum_{u,v} (\texttt{Image}[i-u][j-v]^2) \cdot \sum_{u,v} (\texttt{ImageTemplate}[l-u][c-v]^2)}}$$

wobei $X[i][j]$ den Grauwert in der i-ten Zeile und j-ten Spalte des Bildes X bezeichnet. (l, c) ist der Schwerpunkt der Region von `ImageTemplate`. u und v werden so gewählt, daß alle Punkte des Templates erreicht werden, i, j laufen über die `RegionOfInterest`. An den Bildrändern werden nur die Teile von `ImageTemplate` berücksichtigt, die innerhalb des Bildes liegen (d.h. u und v werden entsprechend eingeschränkt). Wertebereich: 0 - 255 (best fit).

'dfd' Berechnung der mittleren „displaced frame difference“:

$$\texttt{ImageMatch}[i][j] = \frac{\sum_{u,v} |\texttt{Image}[i-u][j-v] - \texttt{ImageTemplate}[l-u][c-v]|}{AREA(ImageTemplate)}$$

Bezeichnungen wie bei 'norm_correlation'. AREA (X) steht für die Fläche der Region X. Wertebereich: 0 (best fit) - 255.

Sowohl die normierte Korrelation, als auch die „displaced frame difference“ sind in ihrer Berechnung (abhängig von der Fläche des `ImageTemplate`) sehr aufwendig. Es ist daher wichtig, die Eingaberegion (`RegionOfInterest`) möglichst einzugrenzen, d.h. den Filter nur in einer möglichst eng umrissenen „region of interest“ anzuwenden.

Qualitativ liefern beide Modi vergleichbare Ergebnisse. Der Modus 'dfd' ist jedoch um etwa Faktor 3.5 schneller.

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **`RegionOfInterest`** (input_object) ..... region ⇝ *Hobject*
  Suchbereich im Eingabebild.
- ▷ **`ImageTemplate`** (input_object) ..... image ⇝ *Hobject* : byte
  Dieser Bildbereich wird innerhalb von `RegionOfInterest` mit `Image` „gematcht“.
- ▷ **`ImageMatch`** (output_object) ..... image ⇝ *Hobject*
  Ergebnisbild: Werte des Matching-Kriteriums.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Gewünschtes Matching-Kriterium.
  Defaultwert : 'dfd'
  Werteliste : `Mode` ∈ {'norm_correlation', 'dfd'}

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `exhaustive_match` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`exhaustive_match` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`draw_region`, `draw_rectangle1`

Mögliche Nachfolgerfunktionen

`local_max`, `threshold`

Alternativen

`exhaustive_match_mg`

Modul

Image filters

exhaustive_match_mg (Image, ImageTemplate : ImageMatch : Mode, Level, Threshold :)

Matching zwischen einem Template und einem Bild in einer Auflösungspyramide.

`exhaustive_match_mg` ist eine Ergänzung zum Operator `exhaustive_match`, die ein Matching zwischen dem Bild `Image` und dem Template `ImageTemplate` durchführt. Dabei wird `ImageTemplate` über alle Punkte der Region von `Image` geschoben, abhängig vom Parameter `Mode` ein Matching-Kriterium berechnet und die Ergebniswerte in `ImageMatch` abgelegt.

In der Regel interessieren von solcherart gefilterten Bildern aber nur die Bereiche mit guten Matchingergebnissen. Die Größe des Suchbereichs, also die Region des Eingabebildes `Image`, bestimmt maßgeblich das Laufzeitverhalten des Matching-Filters. Daher wird bei `exhaustive_match_mg` mit reduzierter Bildauflösung zunächst eine „region of interest“ bestimmt, in der mit guten Matching-Ergebnissen zu rechnen ist, und im Anschluß daran nur innerhalb dieser Teilregion in normaler Auflösung das eigentliche Matching (vgl. `exhaustive_match`) durchgeführt. Dazu werden die Gauß-Pyramiden von `Image` und `ImageTemplate` aufgebaut (insbesondere werden auch die entsprechenden Regionen mittransformiert). Dann wird auf jeder Ebene der Auflösungspyramiden - beginnend mit der Startebene `Level` - das Matching innerhalb der aktuellen „region of interest“ ausgeführt. Dabei ist die „region of interest“ im Startlevel gleich der Region des Eingabebildes `Image`. Nach der Filterung wird mittels einer Schwellenwert-Operation eine neue „region of interest“ bestimmt und auf die nächste Auflösungsstufe transformiert:

`threshold`(..0,Threshold..), falls `Mode` = 'dfd'
`threshold`(..Threshold,255..), falls `Mode` = 'crosscorrelation'

In höchster Auflösung (`Level` 0) wird dann schließlich das abschließende Matching in der ermittelten „region of interest“ berechnet. Das Ausgabebild `ImageMatch` enthält das entsprechende Filterergebnis und als Region die endgültige „region of interest“, die mit einer Schwellenwert-Operation auf dem Ergebnisbild bestimmt wird.

`exhaustive_match_mg` ist daher kein reiner Filter, sondern ist auch zur Klasse der Regionentransformationen zu zählen.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **`ImageTemplate`** (input_object) ..... image ⇝ *Hobject* : byte
  Der Definitionsbereich dieses Bildes wird mit `Image` „gematcht“.
- ▷ **`ImageMatch`** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Ergebnisbild und -region: Werte des Matching-Kriteriums innerhalb der ermittelten „region of interest“.
  Parameteranzahl : `ImageMatch` = `Image`
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Gewünschtes Matching-Kriterium.
  Defaultwert : ’dfd’
  Werteliste : `Mode` ∈ {’crosscorrelation’, ’dfd’}
- ▷ **`Level`** (input_control) ..... integer ⇝ *integer*
  Startlevel in der Auflösungspyramide (höchste Auflösung: Level 0).
  Defaultwert : 1
  Werteliste : `Level` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}
  Restriktion : (`Level` < ld(width(`Image`))) ∧ (`Level` < ld(height(`Image`)))
- ▷ **`Threshold`** (input_control) ..... integer ⇝ *integer*
  Schwelle zur Bestimmung der „region of interest“.
  Defaultwert : 30
  Wertevorschläge : `Threshold` ∈ {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}
  Typischer Wertebereich : 0 ≤ `Threshold` ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5

Beispiel

```
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_mg(Image,Template,&ImageMatch,’dfd’1,30);
invert_image(ImageMatch,&ImageInvert);
local_max(ImageInvert,&BestFit);
disp_region(BestFit,WindowHandle);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `exhaustive_match_mg` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`exhaustive_match_mg` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`draw_region`, `draw_rectangle1`

Mögliche Nachfolgerfunktionen

`threshold`, `local_max`

Alternativen

exhaustive_match

Siehe auch

gen_gauss_pyramid

Modul

Image filters

fast_match (Image : Matches : TemplateID, MaxError :)

Suche nach allen guten Matches eines Templates und eines Bilds.

fast_match führt ein Matching zwischen dem Template von TemplateID und Image durch. Dabei wird das Template so über die Punkte von Image geschoben, daß es immer vollständig innerhalb von Image liegt. Das Matching-Kriterium („displaced frame difference“) ist wie folgt definiert:

$$error[row, col] = \frac{\sum_{u,v} |\texttt{Image}[row-u, col-v] - \texttt{TemplateID}[u,v]|}{area(TemplateID)}$$

Die Laufzeit des Verfahrens hängt von der Größe des Definitionsbereiches von Image ab. Es ist daher wichtig, den Definitionsbereich möglichst einzugrenzen, d.h. den Operator nur in einer möglichst eng umrissenen „region of interest“ anzuwenden. Der Parameter MaxError legt den maximalen Fehler fest, den die gesuchten Positionen haben dürfen. Um so kleiner dieser Wert ist, um so schneller läuft das Verfahren.

Alle Punkte, bei denen der Fehler des Matching kleiner als MaxError ist, werden in der Ausgaberegion Matches übergeben.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **Matches** (output_object) ..... region(-array) ⇝ *Hobject*
  Alle Punkte, bei denen der Fehler unter der Schwelle liegt.
- ▷ **TemplateID** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.
- ▷ **MaxError** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert : 20
  Wertevorschläge : MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30}
  Typischer Wertebereich : 0 ≤ MaxError ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fast_match den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fast_match ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Mögliche Nachfolgerfunktionen

connection, best_match

Alternativen

best_match, best_match_mg, fast_match_mg, exhaustive_match, exhaustive_match_mg

Modul

Template matching

fast_match_mg (Image : Matches : TemplateID, MaxError, NumLevel :)

Suche nach allen guten Grauwert-Matches in einer Pyramide.

fast_match_mg führt analog zu fast_match ein Matching zwischen dem Template von TemplateID und Image durch. Im Gegensatz zu fast_match beginnt die Suche nach guten Matches aber in verkleinerten Bildern (Pyramide). Die Anzahl der Ebenen der Pyramide werden mit NumLevel festgelegt. Dabei bedeutet der Wert 1, daß keine Pyramide verwendet wird. fast_match_mg ist in diesem Fall identisch mit fast_match. Bei dem Wert 2 beginnt die Suche bei dem Bild mit halber Kantenlänge. Für alle Punkte, die in dem verkleinerten Bild einen geringen Fehler hatten (kleiner als MaxError) wird die Suche dann bei den entsprechenden Positionen im Originalbild (Image) verfeinert. Optional kann bei dem Parameter NumLevel ein zweiter Wert übergeben werden. Dieser legt die Nummer der Ebene fest bei der mit dem Matching aufgehört werden soll. Es wird dann die bis dahin gefunden Region auf die Originalebene vergrößert. Die Werte sind größer oder gleich Null. Dabei entspricht Null der untersten Ebene und führt somit zu keiner Veränderung des Verfahrens. Bei der Wert 1 bricht das Matching eine Ebene oberhalb ab. Hierdurch wird Rechenzeit eingespart. Dies ist insbesondere dann sinnvoll, wenn es nicht auf eine genaue Lokalisation ankommt.

Die Laufzeit hängt dabei sehr von MaxError ab: um so größer der Schwellenwert ist um so länger dauert das Matching, da mehr Punkt weiter untersucht werden müssen. Falls MaxError jedoch zu klein ist, wird das gesuchte Muster nicht gefunden. Der Wert ist also für jede Anwendung durch Tests zu optimieren.

NumLevel gibt die Anzahl von Ebenen der Pyramide, einschließlich dem Ausgangsbild an.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild, in dem das Pattern gefunden werden soll.
- ▷ **Matches** (output_object) ..... region(-array) ⇝ *Hobject*
  Alle Punkte, bei denen der Fehler unter der Schwelle liegt.
- ▷ **TemplateID** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.
- ▷ **MaxError** (input_control) ..... real ⇝ *real*
  Maximale mittlere Differenz der Grauwerte.
  Defaultwert : 30
  Wertevorschläge : MaxError ∈ {0, 1, 2, 3, 4, 5, 6, 7, 9, 11, 15, 17, 20, 30, 40, 50, 60, 70}
  Typischer Wertebereich : 0 ≤ MaxError ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 3
- ▷ **NumLevel** (input_control) ..... integer(-array) ⇝ *integer*
  Anzahl Ebenen in der Pyramide.
  Defaultwert : 3
  Werteliste : NumLevel ∈ {1, 2, 3, 4, 5, 6, 7, 8}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert fast_match_mg den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fast_match_mg ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

create_template, read_template, adapt_template, draw_region, draw_rectangle1, reduce_domain

Alternativen

best_match, best_match_mg, fast_match, exhaustive_match, exhaustive_match_mg

Modul

Template matching

fill_dvf (Image : ImageExpanded : Width, Height, Iterations :)

Ausdehnen eines Verschiebungsvektorfeldes auf die undefinierten Bereiche.

fill_dvf berechnet in einem Verschiebungsvektorfeld für jeden undefinierten Punkt einen Wert durch die Mittelung seiner Nachbarn. Die Größe der Nachbarschaft wird durch die Parameter Width und Height vorgegeben. Da die Lücken größer als die Masken sein können, ist es möglich, den Filter mehrfach anzuwenden (Iterations). Korrekt besetzte Werte im Eingabebild werden nicht modifiziert.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : dvf
  Eingabebild.
- ▷ **ImageExpanded** (output_object) ..... image(-array) ⇝ *Hobject* : dvf
  Ergebnis der Ergänzung der fehlenden Punkte.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 5
  Wertevorschläge : Width ∈ {3, 5, 7, 9, 11, 15}
  Typischer Wertebereich : 3 ≤ Width ≤ 201
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 5
  Wertevorschläge : Height ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich : 3 ≤ Height ≤ 201
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 3
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
  Typischer Wertebereich : 1 ≤ Iterations ≤ 100
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Parallelisierungsinformation

fill_dvf ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

optical_flow_match

Modul

Image filters

gen_gauss_pyramid (Image : ImagePyramid : Mode, Scale :)

Berechnung einer Gauß-Pyramide.

gen_gauss_pyramid berechnet eine Pyramide von verkleinerten Bildern. Das Maß, um das jedes Nachfolgerbild verkleinert wird, gibt der Parameter Scale an. Beispielsweise wird mit einem Wert von 0.5 für Scale die Kantenlänge von Image auf die Hälfte verkürzt. Dies entspricht gerade der „normalen“ Pyramide.

Der Parameter Mode gibt die Art der Mittelung an. Eine Beschreibung hierzu ist bei affine_trans_image zu finden. Für den Fall, daß Scale gleich 0.5 ist, stehen zusätzlich die Werte **’min’** und **’max’** zu Verfügung. Hierbei wird das Minimum bzw. das Maximum der vier Punkte verwendet.

Es ist zu beachten, daß jede Ebene als ein eigenes Bild, d.h als eigenständiges Bildobjekt mit einer Matrix und eigenem Definitionsbereich ausgegeben wird. Wird das Ergebnis als ein mehrkanaliges Bild benötigt, muß der

Operator `image_to_channels` verwendet werden. Einzelne oder mehrere Ebenen können mit `select_obj` bzw. `copy_obj` selektiert werden.

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte / real
  Eingabebild.
- ▷ **`ImagePyramid`** (output_object) ..... image-array ⇝ *Hobject* : byte / real
  Ausgabebilder.
  Parameteranzahl : `ImagePyramid` > `Image`
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Art der Filtermaske.
  Defaultwert : 'weighted'
  Werteliste : `Mode` ∈ {'none', 'constant', 'weighted', 'min', 'max'}
- ▷ **`Scale`** (input_control) ..... real ⇝ *real*
  Verkleinerungsfaktor.
  Defaultwert : 0.5
  Wertevorschläge : `Scale` ∈ {0.2, 0.3, 0.4, 0.5, 0.6}
  Typischer Wertebereich : 0.1 ≤ `Scale` ≤ 0.9
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : (`0.1` < `Scale`) ∧ (`Scale` < `0.9`)

Beispiel

```
gen_gauss_pyramid(Image,Pyramid,"weighted",0.5);
count_obj(Pyramid,&num);
for (i=1; i<=num; i++)
{
  select_obj(Pyramid,&Single,i);
  disp_image(Single,WindowHandle);
  clear(Single);
}
```

Parallelisierungsinformation

`gen_gauss_pyramid` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`image_to_channels`, `count_obj`, `select_obj`, `copy_obj`

Alternativen

`zoom_image_size`, `zoom_image_factor`

Siehe auch

`affine_trans_image`

Modul

Image filters

`monotony` (Image : ImageMonotony : :)

Berechnung des Monotonieoperators.

`monotony` berechnet den Monotonieoperator. Hierzu wird die Anzahl der Punkte in der 8-Nachbarschaft gezählt, die echt kleiner als der aktuelle Grauwert sind. Diese Anzahl wird im Ausgabebild eingetragen.

Bei einem echten Maximum erhält man den Wert 8, bei einem Minimum oder auf einem Plateau ergibt sich der Wert 0. Bei einem Grat oder einem Hang erhält man entsprechende Zwischenwerte.

Um lokale Minima zu untersuchen, muß das Eingabebild vorher nur mit dem Befehl `invert_image` invertiert werden.

Der Monotonieoperator wird häufig als Vorbereitung für Matching-Verfahren eingesetzt, da er invariant gegen Helligkeitsänderungen ist.

Parameter

▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
Eingabebild.

▷ **ImageMonotony** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
Ergebnis des Monotonieoperators.
Parameteranzahl : `ImageMonotony` = `Image`

Beispiel

```
/* searching the strict maximums */
gauss_image(Image,&Gauss,5);
monotony(Gauss,&Monotony);
threshold(Monotony,Maxima,8.0,8.0);
```

Parallelisierungsinformation

`monotony` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `median_image`, `mean_image`, `smooth_image`, `invert_image`

Mögliche Nachfolgerfunktionen

`threshold`, `exhaustive_match`, `disp_image`

Alternativen

`local_max`, `topographic_sketch`, `corner_response`

Modul

Image filters

optical_flow_match (Image1, Image2 : VectorField : Mode, Threshold, Step, SizeWindow, SizeSearch, Weights :)

Berechnung des Verschiebungsvektorfeldes mittels Korrelationsmethoden.

`optical_flow_match` bestimmt ein Verschiebungsvektorfeld mittels Korrelationsmethoden. Es können ein- und mehrkanalige Bilder verwendet werden. Ein Fenster der Größe `SizeWindow` wird in einem Suchbereich der Größe `SizeSearch` über das zweite Bild geführt und dort mit den Grauwerten im ersten Bild verglichen. Die Zuordnung mit dem geringsten Fehler gibt den Vektor vor. Bei zwei Vektoren mit gleicher Bewertung wird der kürzere bevorzugt. Nach Bearbeitung eines Bildpunktes wird der nächste Bildpunkt im Abstand `Step` gewählt. Es muß also kein dichtes Verschiebungsvektorfeld entstehen.

Der Parameter `Threshold` gibt die maximale Abweichung von vorher geschätzter Korrelation und bester Schätzung eines Verschiebunsgvektors an.

Die Art des Verfahrens wird über den Parameter `Mode` bestimmt. Folgende Werte für `Mode` sind möglich:

’dfd’ Berechnung der ”displaced frame difference”, d.h. die Summe der Betragsdifferenzen.

”dfd_norm’ Berechnung die normalisierete ”displaced frame difference”, d.h. die Summe der Betragsdifferenzen normiert auf den Mittelwert.

Bei mehrkanaligen Bildern gibt der Parameter `Weights` die Gewichtung der einzelnen Kanäle vor. Für jeden Kanal wird jeweils ein Gewichtungsfaktor übergeben.

Bei den Parametern `SizeWindow`, `SizeSearch` und `Step` können auch zwei Werte übergeben werden. Der erste Wert gibt dann die Breite (bzw. Zeile), der zweite die Höhe (bzw. Spalte) an.

Parameter

- ▷ **`Image1`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Erstes Eingabebild (evtl. mehr als ein Kanal).
- ▷ **`Image2`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Zweites Eingabebild (evtl. mehr als ein Kanal).
  Parameteranzahl : `Image2` = `Image1`
- ▷ **`VectorField`** (output_object) ..... image(-array) ⇝ *Hobject* : dvf
  Zu berechnende Verschiebungsvektorfelder.
  Parameteranzahl : `VectorField` = `Image1`
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Art der Korrelation.
  Defaultwert : 'dfd'
  Werteliste : `Mode` ∈ {'dfd', 'dfd_norm'}
- ▷ **`Threshold`** (input_control) ..... integer ⇝ *integer*
  Maximale Abweichung von vorher geschätzter Korrelation und bester Schätzung eines Verschiebungsvektors.
  Defaultwert : 10
  Wertevorschläge : `Threshold` ∈ {1, 2, 3, 4, 5, 7, 10, 12, 15, 17, 20}
  Typischer Wertebereich : 1 ≤ `Threshold` ≤ 300
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5
- ▷ **`Step`** (input_control) ..... integer(-array) ⇝ *integer*
  Schrittweite.
  Defaultwert : 10
  Wertevorschläge : `Step` ∈ {1, 2, 3, 5, 7, 10, 15, 20, 30, 50}
  Typischer Wertebereich : 1 ≤ `Step` ≤ 300
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5
- ▷ **`SizeWindow`** (input_control) ..... integer(-array) ⇝ *integer*
  Größe des Korrelationsfensters.
  Defaultwert : 11
  Wertevorschläge : `SizeWindow` ∈ {3, 5, 7, 9, 11, 15, 20, 30}
  Typischer Wertebereich : 1 ≤ `SizeWindow` ≤ 30
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **`SizeSearch`** (input_control) ..... integer(-array) ⇝ *integer*
  Größe des Suchbereiches.
  Defaultwert : 20
  Wertevorschläge : `SizeSearch` ∈ {3, 5, 7, 10, 20, 30, 40, 50, 60}
  Typischer Wertebereich : 1 ≤ `SizeSearch` ≤ 60
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **`Weights`** (input_control) ..... integer(-array) ⇝ *integer*
  Gewichtung der Kanäle.
  Defaultwert : 1
  Wertevorschläge : `Weights` ∈ {1, 3, 5, 7, 10}
  Typischer Wertebereich : 1 ≤ `Weights` ≤ 10
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Beispiel

```
read_image(B1,'image1')
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).
```

Parallelisierungsinformation

`optical_flow_match` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`mean_image`, `gauss_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`fill_dvf`

Siehe auch

`dvf_to_int1`, `dvf_to_hom_mat2d`

Modul

Image filters

`read_template ( : : FileName : TemplateID )`

Einlesen eines Templates von Datei.

`read_template` liest ein Matching Template von Datei das mit `write_template` geschrieben wurde.

Parameter

- ▷ **`FileName`** (input_control) ..... filename ⇝ *string*
  Name der Datei.
- ▷ **`TemplateID`** (output_control) ..... template ⇝ *integer*
  Nummer des Templates.

Ergebnis

Ist der Dateiname korrekt, dann liefert `read_template` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_template` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`adapt_template`, `set_reference_template`, `set_offset_template`, `best_match`, `fast_match`, `best_match_rot`

Modul

Template matching

`set_offset_template ( : : TemplateID, GrayOffset : )`

Grauwertoffset für Template.

`set_offset_template` addiert einen Offset zu den Grauwerten des Templates um eine Grauwertänderung im Bild auszugleichen. Der Parameter `GrayOffset` gibt die Änderung gegenüber den Grauwerten des ursprünglichen Pattern an, wie sie bei dem Aufruf von `create_template` gesetzt waren. Der Wert von `GrayOffset` bezieht sich auf das Bild in dem das Matching ausgeführt wird: Bei einem helleren Bild ist ein positiver Wert, bei einem dunkleren Bild ein negativer Wert zu übergeben. `set_offset_template` ist bei jeder Grauwertänderung aufzurufen. Die Grauwerte können z.B. in einem Referenzbereich des Bildes mit `intensity` oder `min_max_gray` bestimmt werden.

Parameter

- ▷ **`TemplateID`** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.

▷ **GrayOffset** (input_control) ..... number ⇝ *integer*
Offset der Grauwert.
Defaultwert : 0
Wertevorschläge : GrayOffset ∈ {-10, -5, -2, -1, 0, 1, 2, 5, 10}
Typischer Wertebereich : -255 ≤ GrayOffset ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_offset_template den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_offset_template wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_template, adapt_template, read_template

Mögliche Nachfolgerfunktionen

best_match, best_match_mg, best_match_rot, fast_match, fast_match_mg

Modul

Template matching

set_reference_template (: : TemplateID, Row, Column :)

Referenzposition für ein Matching-Template setzen.

set_reference_template definiert die Referenzposition eines Templates neu. Als Default nach create_template oder create_template_rot wird der Ursprung des Bild $(0, 0)$ als Referenz verwendet, d.h. bei einer Verschiebung von Null wird der Schwerpunkt des Templates als Ergebnis geliefert. Mit set_reference_template kann nun ein beliebiger Referenzpunkt definiert werden. Verwendet man z.B. den Schwerpunkt als Referenz, dann wird bei einer Verschiebung von Null der Vektor $(0, 0)$ geliefert.

Parameter

▷ **TemplateID** (input_control) ..... template ⇝ *integer*
Nummer des Templates.
▷ **Row** (input_control) ..... point.y ⇝ *real*
Referenzposition des Templates (Zeile).
▷ **Column** (input_control) ..... point.x ⇝ *real*
Referenzposition des Templates (Spalte).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert set_reference_template den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_reference_template wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_template, create_template_rot, read_template, adapt_template

Mögliche Nachfolgerfunktionen

best_match, best_match_mg, best_match_rot, fast_match, fast_match_mg

Modul

Template matching

write_template (: : TemplateID, FileName :)

Schreiben eines Templates auf Datei.

write_template schreibt ein Matching Template auf Datei das mit read_template wieder gelesen werden kann.

Parameter

- ▷ **TemplateID** (input_control) ..... template ⇝ *integer*
  Nummer des Templates.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der Datei.

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert write_template den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_template ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_template, create_template_rot

Modul

Template matching

3.10 Rauschen

add_noise_distribution (Image : ImageNoise : Distribution :)

Synthetisches Verrauschen eines Bildes.

add_noise_distribution überlagert das Eingabebild (Image) mit Rauschen, das der Verteilung von Distribution entspricht. Es erfolgt ein Clipping der resultierenden Grauwerte auf den Bereich [0,255].

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Bild, das verrauscht werden soll.
- ▷ **ImageNoise** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Verrauschtes Bild.
  Parameteranzahl : ImageNoise = Image
- ▷ **Distribution** (input_control) ..... distribution.values-array ⇝ *real*
  Rauschverteilung.
  Parameteranzahl : 513

Beispiel

```
read_image(Image,'meer_rot')
disp_image(Image,WindowHandle)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle).
```

Ergebnis

add_noise_distribution liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

add_noise_distribution ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

gauss_distribution, sp_distribution, noise_distribution_mean

Alternativen

add_noise_white

Siehe auch

sp_distribution, gauss_distribution, noise_distribution_mean, add_noise_white

Modul

Image filters

add_noise_white (Image : ImageNoise : Amp :)

Synthetisches Verrauschen eines Bildes.

add_noise_white überlagert das Bild (Image) mit im Intervall [-Amp,Amp] gleichverteiltem, mittelwertfreiem, weißem Rauschen, das mittels der C-Funktion „drand48“ mit zeitabhängigem Seed generiert wird. Anschließend erfolgt ein Clipping der resultierenden Grauwerte auf den Bereich [0,255].

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Bild, das verrauscht werden soll.
- ▷ **ImageNoise** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Verrauschtes Bild.
  Parameteranzahl : ImageNoise = Image
- ▷ **Amp** (input_control) ..... real ⇝ *real*
  Additives Rauschen (in [-Amp,Amp] gleichverteilt)
  Defaultwert : 60.0
  Wertevorschläge : Amp ∈ {1.0, 2.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 1.0 ≤ Amp ≤ 1000.0
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : Amp > 0

Beispiel

```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
add_noise_white(Image,ImageNoise,90)
disp_image(ImageNoise,WindowHandle).
```

Ergebnis

add_noise_white liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

add_noise_white ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

add_noise_distribution

Siehe auch

add_noise_distribution, noise_distribution_mean, gauss_distribution, sp_distribution

Modul

Image filters

gauss_distribution (: : Sigma : Distribution)

Erzeugung einer Gaußschen Rauschverteilung.

gauss_distribution erzeugt eine Rauschverteilung mit der Verteilung einer Gaußglocke. Dabei gibt der Parameter Sigma die Standardabweichung des Rauschens an. Das Ergebnis (Distribution) dient in der Regel als Eingabe für den Befehl add_noise_distribution.

Parameter

▷ **Sigma** (input_control) ..... real ⇝ *real*
Standardabweichung der Verteilung.
Defaultwert : 2.0
Wertevorschläge : Sigma ∈ {1.5, 2.0, 3.0, 5.0, 10.0}
Typischer Wertebereich : 0.0 ≤ Sigma ≤ 100.0
Minimale Schrittweite : 0.1
Empfohlene Schrittweite : 1.0

▷ **Distribution** (output_control) ..... distribution.values-array ⇝ *real*
Zu berechnende Gaußverteilung.
Parameteranzahl : 513

Beispiel

```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
gauss_distribution(30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle).
```

Parallelisierungsinformation

gauss_distribution ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

add_noise_distribution

Alternativen

sp_distribution, noise_distribution_mean

Siehe auch

sp_distribution, add_noise_white, noise_distribution_mean

Modul

Image filters

noise_distribution_mean (ConstRegion, Image : : FilterSize : Distribution)

Rauschverteilung eines Bildes bestimmen.

noise_distribution_mean berechnet die Rauschverteilung in einem Auschnitt des Bildes Image. Der Parameter ConstRegion gibt den Bereich in dem Bild an, der nicht strukturiert ist. Die Grauwerte sollten hier also näherungsweise konstant sein. Die Abweichungen in den Grauwerten werden hier nur durch Rauschen erzeugt. Die Rauschwerteilung wird bestimmt, indem das Bild mit dem Mittelwertfilter (mean_image) geglättet und dann die punktweise Differenzen der Grauwerte (Original minus Mittelwertbild) bestimmt werden. Die Häufigkeit der auftretenden Differenzen werden in dem Parameter Distribution übergeben.

Achtung

Es ist zu beachten, das die Region in ConstRegion nicht zu dicht an einen Garuwertgradienten angrenzt, da durch die Mittelung auch die Nachbarschaft mit einbezogen wird. Konkret heißt das, daß der Abstand vom Rand zu einer Kante so groß wie die Maskengröße der Mittelung FilterSize sein muß.

Parameter

- ▷ **ConstRegion** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu analysierende Regionen mit konstanten Grauwerten.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
  Zugehöriges Bild.
- ▷ **FilterSize** (input_control) ..... integer ⇝ *integer*
  Filtergröße des Mittelwertfilters.
  Defaultwert : 21
  Wertevorschläge : FilterSize ∈ {5, 11, 15, 21, 31, 51, 101}
  Typischer Wertebereich : 3 ≤ FilterSize ≤ 501 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
- ▷ **Distribution** (output_control) ..... distribution.values-array ⇝ *real*
  Rauschverteilung aller Eingaberegionen.

Parallelisierungsinformation

noise_distribution_mean ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

draw_region, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, threshold, erosion_circle, gauss_image, smooth_image, sub_image

Mögliche Nachfolgerfunktionen

add_noise_distribution, disp_distribution

Siehe auch

mean_image, gauss_distribution

Modul

Image filters

sp_distribution (: : PercentSalt, PercentPepper : Distribution)

Erzeugung einer Salz- und Pfeffer- Rauschverteilung.

sp_distribution erzeugt eine Rauschverteilung mit Rauschwerten von 0 und 255. Die Parameter PercentSalt und PercentPepper geben dabei an, wieviel Prozent weiße und schwarze Punkte auftreten sollen. Die Summe der Parameter muß kleiner als 100 sein. Das Ergebnis (Distribution) dient in der Regel als Eingabe für den Befehl add_noise_distribution.

Parameter

- ▷ **PercentSalt** (input_control) ..... number ⇝ *real* / integer
  Prozentanteil von Salz im Rauschen.
  Defaultwert : 5.0
  Wertevorschläge : PercentSalt ∈ {1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0}
  Typischer Wertebereich : 0.0 ≤ PercentSalt ≤ 100.0
  Minimale Schrittweite : 0.1
  Empfohlene Schrittweite : 1.0
  Restriktion : (0.0 ≤ PercentSalt) ∧ (PercentSalt ≤ 100.0)
- ▷ **PercentPepper** (input_control) ..... number ⇝ *real* / integer
  Prozentanteil von Pfeffer im Rauschen.
  Defaultwert : 5.0
  Wertevorschläge : PercentPepper ∈ {1.0, 2.0, 5.0, 7.0, 10.0, 15.0, 20.0, 30.0}
  Typischer Wertebereich : 0.0 ≤ PercentPepper ≤ 100.0
  Minimale Schrittweite : 0.1
  Empfohlene Schrittweite : 1.0
  Restriktion : (0.0 ≤ PercentPepper) ∧ (PercentPepper ≤ 100.0)
- ▷ **Distribution** (output_control) ..... distribution.values-array ⇝ *real*
  Zu berechnende Verteilung.
  Parameteranzahl : 513

Beispiel

```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
sp_distribution(30,30,Dist)
add_noise_distribution(Image,ImageNoise,Dist)
disp_image(ImageNoise,WindowHandle)
```

Parallelisierungsinformation

`sp_distribution` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`add_noise_distribution`

Alternativen

`gauss_distribution`, `noise_distribution_mean`

Siehe auch

`gauss_distribution`, `noise_distribution_mean`, `add_noise_white`

Modul

Image filters

3.11 Sonstige

`convol_image` (Image : ImageResult : FilterMask, Margin :)

Faltung des Bildes mit beliebiger linearer Maske.

`convol_image` führt eine lineare Filterung mit Eingabebild `Image` durch. Die Filtermatrix, die durch `FilterMask` definiert ist, kann dabei entweder aus einer Datei oder einem Tupel generiert werden. Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (`Margin`) gewählt werden:

0 . . . 255 Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.

-1 Fortsetzung der Randpunkte.

-2 Zyklische Fortsetzung der Bildränder.

-3 Spiegelung der Bildpunkte an den Bildrändern.

Alle Bildpunkte werden mit der Konvolutionsmatrix gefaltet. Über- bzw. Unterlauf von Grauwerten wird beschnitten (0 bzw. 255). Wird in `FilterMask` ein Dateiname übergeben, wird die Filtermatrix aus einer Textdatei mit folgender Struktur eingelesen:

⟨Matrixgröße⟩
⟨Inverser Gewichtungsfaktor⟩
⟨Matrix⟩

In der ersten Zeile steht die Größe der Filtermatrix, die durch zwei Zahlen (durch Leerzeichen getrennt) angegeben wird (z.B. 3 3 für 3×3). Dabei ist die erste Zahl die Höhe der Filtermaske und die zweite Zahl ihre Breite. In der nächsten Zeile steht eine ganze Zahl, durch die das Ergebnis der Matrizenberechnung für einen Bildpunkt dividiert wird. Ab der nächsten Zeile steht die Filtermatrix, die durch ganze Zahlen (getrennt durch Leerzeichen) angegeben wird. Pro Textzeile wird eine Matrixzeile angegeben. Die Filtermatrix muß mit der Extension „.fil“ versehen sein. Diese wird bei Aufruf weggelassen. Soll die Filtermaske aus einem Tupel generiert werden, muß das in `FilterMask` übergebene Tupel ebenfalls dem oben beschriebenen Format genügen, mit dem Unterschied, daß in diesem Fall kein Zeilenumbruch notwendig ist.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Bilder, deren Grauwerte gefaltet werden sollen.
- ▷ **ImageResult** (output_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte
  Gefaltete Bilder.
- ▷ **FilterMask** (input_control) ..... string(-array) ⇝ *string* / integer
  Filtermaske als Dateiname oder Tupel.
  Defaultwert : 'sobel'
  Wertevorschläge : `FilterMask` ∈ {'sobel', 'laplace4', 'lowpas_3_3'}
- ▷ **Margin** (input_control) ..... integer ⇝ *integer*
  Randbehandlung: 0...255 (konstant), -1 (Randpunkte fortgesetzt), -2 (zyklische Fortsetzung), -3 (Spiegelung).
  Defaultwert : -3
  Typischer Wertebereich : -3 ≤ `Margin` ≤ 255

Parallelisierungsinformation

`convol_image` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Modul

Image filters

expand_domain_gray (InputImage : ExpandedImage : ExpansionRange :)

Expansion des Definitionsbereiches eines Bildes inklusive der Grauwerte.

`expand_domain_gray` expandiert die Grauwerte am Rand des Definitionsbereiches nach außen. Das Maß dieser Expansion wird durch den Parameter `ExpansionRange` festgelegt, der den Expansionsradius in Pixel beschreibt. Alle Filter in HALCON — die stets auf den Definitionsbereich des Bildes angewendet werden — beziehen abhängig von ihrer Filterweite auch eine bestimmte Anzahl von Grauwerten außerhalb des Definitionsbereiches mit in ihre Berechnung ein. Dies kann im Randbereich des Definitionsbereiches zu unerwünschten Effekten führen. Dieser Nachteil macht sich vor allem dann bemerkbar, wenn sich Vordergrund (Definitionsbereich) und Hintergrund des Bildes in ihrer Helligkeit stark voneinander unterscheiden. In diesem Fall kann es z.B. bei Anwendung eines Glättungsfilters zu unerwünschten Aufhellungen oder Abdunkelungen im Randbereich des Definitionsbereiches kommen. Um dies zu vermeiden, wird durch `expand_domain_gray` der Definitionsbereich im Vorfeld künstlich erweitert, indem die Grauwerte seiner Randpixel nach außen fortgesetzt werden. Mit der Expansion der Grauwerte wird auch der Definitionsbereich entsprechend erweitert, um die neu besetzten Pixel zu kennzeichnen. Daher ist es in vielen Fällen sinnvoll, nach `expand_domain_gray` ein `reduce_domain` oder `change_domain` auszuführen und anschließend die Filteroperation auf den reduzierten Definitionsbereich anzuwenden. Als Wert für `ExpansionRange` sollte die Hälfte der Filterbreite bzw. -höhe gewählt werden.

Parameter

- ▷ **InputImage** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Eingabebild mit zu erweiterndem Definitionsbereich.
- ▷ **ExpandedImage** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int2
  Ausgabebild mit neu berechneten Grauwerten im erweiterten Definitionsbereich.
- ▷ **ExpansionRange** (input_control) ..... integer ⇝ *integer*
  Radius der Grauwertexpansion in Pixel.
  Defaultwert : 2
  Wertevorschläge : `ExpansionRange` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16}
  Restriktion : `ExpansionRange` ≥ 1

Beispiel

```
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);
```

```
dyn_threshold(Fabrik_Label,Label_Mean_normal,Characters_normal,10,'dark');
dev_display(Fabrik);
dev_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');
dev_display(Fabrik);
dev_display(Characters_expanded);
/* Now, even in the border region the characters are recognized */
```

Komplexität

Sei L der Umfang des Definitionsbereiches. Die Laufzeit-Komplexität ist dann näherungsweise $O(L) * $ `ExpansionRange`.

Ergebnis

`expand_domain_gray` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`expand_domain_gray` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`reduce_domain`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `mean_image`, `dyn_threshold`

Siehe auch

`reduce_domain`, `mean_image`

Modul

Image filters

gray_inside (Image : ImageDist : :)

Berechnung des niedrigstmöglichen Grauwerts auf einem beliebigen Weg zum Bildrand für jeden Bildpunkt.

`gray_inside` „sucht“ sich für jeden Bildpunkt den günstigsten Weg zum Bildrand, d.h. den Weg, auf dem möglichst niedrige Grauwerte zu überwinden sind. Die Differenz des Grauwertes des jeweiligen Punktes zum maximalen Grauwert auf diesem Weg wird für jeden Punkt berechnet und als Ergebnisbild geliefert. Helle Bildbereiche im Ergebnisbild bedeuten also, daß diese Bereiche (die im Originalbild typischerweise eher dunkel sind) von helleren Bereichen eingeschlossen sind. Dunkle bzw. schwarze Bereiche im Ergebnisbild bedeuten, daß zwischen ihnen und dem Bildrand nur geringfügig hellere bzw. dunklere Bildbereiche existieren (was nicht heißen muß, daß sie nur von dunkleren Bereichen umgeben sind; es reicht bereits ein kleiner dunkler „Ausgang“). Der Wert 0 (schwarz) im Ergebnisbild bedeutet, daß entweder gleich helle oder dunklere Grauwerte auf dem Weg zum Bildrand auftreten.

Für die Durchführung wird zunächst die Operation `watersheds` benutzt, die eine Segmentation des Bildes in Basins und Watersheds durchführt. Wenn man sich das zu bearbeitende Bild als Grauwertgebirge vorstellt, bilden die Basins die Täler und Becken (mit niedrigeren Grauwerten) , die „Gebirgskämme“ sind die Watersheds. Zunächst werden die Watersheds jeweils benachbarten Regionen zugeteilt, so daß nur noch Basins vorhanden sind. Der Rand des Definitionsbereiches des Ausgangsbildes wird nun nach dem kleinsten Grauwert durchsucht und die Region(Basin) in der er liegt mit den Ergebniswerten belegt. Wenn der kleinste Grauwert am echten Bildrand liegt, können die Ergebniswerte sofort aus der Differenz zwischen Punkt-Grauwert und diesem berechnet werden. Liegt der gefundene kleinste Grauwert jedoch innerhalb des Bildes, muß der bereits existierende Ergebniswert aus der Nachbarregion besorgt werden um mit seiner Hilfe die neuen Ergebniswerte zu berechnen. Die Suche nach der Nachbarregion erfolgt mit 8-er Nachbarschaft. Nach Abzug der gefundenen Region vom Suchbereich erfolgt ein neuer Durchgang u.s.w. So wird Basin für Basin das Bild von außen nach innen „geschält“.

Wie auch bei watersheds empfiehlt es sich, vor gray_inside eine Glättung (z.B. gauss_image) des Bildes vorzunehmen, um die Entstehung zu vieler Einzelregionen im watersheds-Algorithmus (und dadurch sehr lange Laufzeiten) zu vermeiden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu bearbeitendes Bild.
- ▷ **ImageDist** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : int2
  Ergebnisbild.

Beispiel

```
read_image(Bild,'coin')
gauss_image (Bild,G_Bild,11)
open_window (0,0,512,512,0,'visible','',WindowHandle)
gray_inside(G_Bild,Ausgabebild)
disp_image (Ausgabebild,WindowHandle).
```

Ergebnis

gray_inside liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

gray_inside ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, smooth_image, mean_image, median_image

Mögliche Nachfolgerfunktionen

select_shape, area_center, count_obj

Siehe auch

watersheds

Modul

Image filters

gray_skeleton (Image : GraySkeleton : :)

Verdünnung von Grauwertbildern.

gray_skeleton führt eine Grauwert-Verdünnung mittels lokalem Schnitt durch. Bildlich gesprochen wird dabei das Grauwertgebirge auf die Gratlinien reduziert, indem die Bergflanken mit dem Grauwert der jeweiligen Talsohle besetzt werden. Die resultierenden „Maximumslinien" sind nur mehr maximal zwei Pixel dick. Die Prozedur ist insbesondere zur Verdünnung von Kantenbildern gedacht und insofern eine Alternative zu nonmax_suppression_amp. gray_skeleton erhält im Gegensatz zu nonmax_suppression_amp Konturen, ist allerdings auch ungleich zeitaufwendiger. Im Gegensatz zu skeleton verändert die Routine die Grauwerte der Bilder, läßt aber deren Regionen unverändert.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Bild, das verdünnt werden soll.
- ▷ **GraySkeleton** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Verdünntes Bild.

Beispiel

```
/* Seeking leafs of a beech tree in an aerial picture: */
read_image(Image,'wald1')
gray_skeleton(Image,Skelett)
mean_image(Skelett,MeanSkelett,7,7)
dyn_threshold(Skelett,MeanSkelett,Leafs,3,'light').
```

Ergebnis

`gray_skeleton` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gray_skeleton` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Nachfolgerfunktionen

`mean_image`

Alternativen

`nonmax_suppression_amp`, `nonmax_suppression_dir`, `local_max`

Siehe auch

`skeleton`, `gray_dilation_rect`

Modul

Image filters

`lut_trans` (Image : ImageResult : Lut :)

Transformation eines Bildes mit einer Grautabelle.

`lut_trans` transformiert ein Bild `Image` mittels einer Tabelle `Lut`. Die Tabelle funktioniert dabei wie eine Transformationsfunktion. Für byte-Bilder muß `Lut` ein Tupel der Länge 256 sein. Für int2-Bilder muß `Lut` ein Tupel der Länge 256 ¡= Länge ¡= 65536 sein. Falls die Länge von `Lut` ¡= 32768 ist, wird die Transformation nur auf den positiven Grauwerten durchgeführt, d.h., das erste Element von `Lut` spezifiziert den neuen Grauwert für den Eingabegrauwert 0. Falls `Lut` länger ist als 32768, so muß die Länge von `Lut` genau 65536 betragen. In diesem Fall werden sowohl die positiven alsauch die negativen Werte transformiert, wobei das erste Element den neuen Grauwert für den Eingabegrauwert -32768 spezifiziert, während das letzte Element den neuen Grauwert für den Eingabegrauwert 32767 spezifiziert. In jedem Fall werden diejenigen Ausgabegrauwerte, deren Eingabegrauwerte außerhalb des gültigen Bereiches von `Lut` liegen, auf 0 gesetzt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Bild, dessen Grauwerte transformiert werden sollen.
- ▷ **`ImageResult`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Ergebnisbild.
- ▷ **`Lut`** (input_control) ..... integer-array ⇝ *integer*
  Tabelle, die die Transformation angibt.

Beispiel

```
/* 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)
```

Ergebnis

`lut_trans` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`lut_trans` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Modul

Image filters

principal_comp (MultichannelImage : PCAImage : InfoPerComp)

Hauptachsentransformation (Principal Components Analysis) von mehrkanaligen Bildern.

`principal_comp` führt eine Hauptachsentransformation (Principal Components Analysis) von mehrkanaligen Bildern durch. Diese Transformation ist z.B. bei Bildern nützlich, die mit dem Thematic Mapper des Landsat Satelliten aufgenommen wurden. Da die einzelnen Kanäle zum Teil starke Korrelationen aufweisen, ist es wünschenswert, die Eingabebilder in Bilder zu transformieren, die möglichst wenig Korrelation aufweisen. Dies ist zum einen nützlich, um Speicherplatz zu sparen, da Bildkomponenten mit geringem Informationsgehalt vernachlässigt werden können, und zum anderen im Hinblick auf eine spätere Klassifikation.

Die Funktion `principal_comp` nimmt ein Eingabebild `MultichannelImage` und transformiert dieses mittels der Hauptachsentransformation in ein Ausgabebild `PCAImage`, das die gleiche Anzahl von Komponenten besitzt. In dem Ausgabeparameter `InfoPerComp` wird der jeweilige relative Informationsgehalt der einzelnen Komponenten zurückgegeben.

Parameter

- ▷ **MultichannelImage** (input_object) ..... multichannel-image ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Mehrkanaliges Grauwertbild.
- ▷ **PCAImage** (output_object) ..... multichannel-image ⇝ *Hobject* : real
  Mehrkanaliges Ausgabebild.
- ▷ **InfoPerComp** (output_control) ..... real-array ⇝ *real*
  Informationsgehalt der ausgegebenen Kanäle.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `principal_comp` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`principal_comp` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Image filters

symmetry (Image : ImageSymmetry : MaskSize, Direction, Exponent :)

Grauwertsymmetrie entlang einer Zeile.

`symmetry` berechnet die Symmetrie entlag einer Zeile. Dabei werden für jeden Bildpunkt die Grauwerte auf beiden „Seiten“ enlang der vorgegebenen Suchrichtung verglichen: Es wird jeweils der Betrag der Differenz zwischen zwei gleich weit entfernten Punkten bestimmt. Jeder dieser Differenzen wird mit dem Exponenten gewichtet (nach einer Division durch 255) und dann wird die Summe all dieser Differenzen gebildet.

$$sym := 255 - \frac{255}{\texttt{MaskSize}} \sum_{i=1}^{\texttt{MaskSize}} \left(\frac{|g(i) - g(-i)|}{255} \right)^{\texttt{Exponent}}$$

Bildpunkte mit einer hohen Symmetrie erhalten einen hohen Grauwert.

Achtung

Gegenwärtig können nur horizontale Suchlinien verwendet werden

Parameter

- ▷ **Image** (input_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **ImageSymmetry** (output_object) . . . . . (multichannel-)image(-array) ⇝ *Hobject* : byte
  Symmetriebild.
- ▷ **MaskSize** (input_control) . . . . . number ⇝ *integer*
  Länge des Suchbereiches.
  Defaultwert : 40
  Wertevorschläge : `MaskSize` ∈ {3, 5, 7, 10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 100, 120, 140, 180}
  Typischer Wertebereich : 3 ≤ `MaskSize` ≤ 1000
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
- ▷ **Direction** (input_control) . . . . . number ⇝ *real*
  Winkel der Untersuchungsrichtung.
  Defaultwert : 0.0
  Wertevorschläge : `Direction` ∈ {0.0}
  Typischer Wertebereich : 0.0 ≤ `Direction` ≤ 0.0
- ▷ **Exponent** (input_control) . . . . . number ⇝ *real*
  Exponent für Gewichtung.
  Defaultwert : 0.5
  Wertevorschläge : `Exponent` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1.0}
  Typischer Wertebereich : 0.05 ≤ `Exponent` ≤ 1.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : (`0` < `Exponent`) ∧ (`Exponent` ≤ `1`)

Beispiel

```
read_image(Image,'monkey')
symmetry(Image,ImageSymmetry,70,0.0,0.5)
threshold(ImageSymmetry,SymmPoints,170,255)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `symmetry` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`symmetry` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`threshold`

Modul

Image filters

topographic_sketch (Image : Sketch : :)

Topographic primal sketch eines Grauwertbildes.

Die Funktion `topographic_sketch` berechnet aus dem Grauwertbild `Image` den „topographic primal sketch“. Dabei wird die Umgebung der einzelnen Bildpunkte mit einem bikubischen Polynom approximiert („facet model“). Aus diesen Daten werden die ersten und zweiten Richtungsableitungen an der Pixelposition berechnet, und daraus eine Einteilung des Pixels in eine von 11 Klassen hergeleitet. Die Klassen werden im Ausgabebild `Sketch` als Zahlen zwischen 1 und 11 zurückgegeben. Die einzelnen Klassen lauten wie folgt:

Peak	1
Pit	2
Ridge	3
Ravine	4
Saddle	5
Flat	6
Hillside Slope	7
Hillside Convex	8
Hillside Concave	9
Hillside Saddle	10
Hillside Inflection	11

Um die einzelnen Klassen als Regionen zu erhalten ist auf dem Labelbild Sketch noch eine Schwellenwertoperation mit entsprechendem Schwellenwert auszuführen.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Das Bild, für das der topographic primal sketch berechnet werden soll.
- ▷ **Sketch** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Das Labelbild mit der Klasseneinteilung.

Beispiel

```
/* To extract the Ridges from a Image */
read_image(Image,'sinus')
topographic_sketch(Image,Sketch)
threshold(Sketch,Ridges,3,3).
```

Komplexität

Sei n die Anzahl der Bildpunkte im Eingabebild. Dann werden $O(n)$ Operationen benötigt.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `topographic_sketch` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`topographic_sketch` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Nachfolgerfunktionen

`threshold`

Literatur

R. Haralick, L. Shapiro: “Computer and Robot Vision, Volume I”; Reading, Massachusetts, Addison-Wesley; 1992; Kapitel 8.13.

Modul

Image filters

3.12 Textur

deviation_image (Image : ImageDeviation : Width, Height :)

Standardabweichnung der Grauwerte in Rechteckfenstern.

`deviation_image` transformiert die Grauwerte der Eingabebilder aus `Image` mit Hilfe einer Filtermaske (`Height`, `Width`), in der die Standardabweichung der Grauwerte berechnet wird. Bei Byte-Bildern wird das Ergebnis mit 2 multipliziert. Das Ergebnis wird in `ImageDeviation` übergeben. Die Steuerparameter `Height`, `Width` werden, falls sie jeweils einen geraden Wert haben, zu dem nächstgrößeren ungeraden Wert gewandelt. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int4 / real / int2
  Bilder, für deren Grauwerte die Standardabweichung berechnet werden sollen.
- ▷ **ImageDeviation** (output_object) ..... image(-array) ⇝ *Hobject* : byte / int4 / real / int2
  Bilder, die die Standardabweichung enthalten.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 11
  Werteliste : `Width` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
  Restriktion : (3 ≤ `Width`) ∧ odd(`Width`)
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 11
  Werteliste : `Height` ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
  Restriktion : (3 ≤ `Height`) ∧ odd(`Height`)

Beispiel

```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
deviation_image(Image,Deviation,9,9)
disp_image(Deviation,WindowHandle).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `deviation_image` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`deviation_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`entropy_image`, `entropy_gray`

Siehe auch

`convol_image`, `texture_laws`, `intensity`

Modul

Image filters

`entropy_image` (Image : ImageEntropy : Width, Height :)

Entropie der Grauwerte in einem Rechteckfenster.

`entropy_image` transformiert die Grauwerte der Eingabebilder aus `Image` mit Hilfe einer Filtermaske (`Height`, `Width`), in der die Entropie der Grauwerte berechnet wird. Das Ergebnis wird in `ImageEntropy` abgelegt. Die Entropie wird dabei mit 32 multipliziert. Die Steuerparameter `Height` und `Width` werden, falls sie jeweils einen geraden Wert haben, zu dem nächstgrößeren ungeraden Wert gewandelt. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Bilder, in denen die Entropie berechnet werden soll.
- ▷ **ImageEntropy** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Ergebnisbild.

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert : 9
Werteliste : Width ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
Wertevorschläge : Width ∈ {3, 5, 7, 9, 11, 13, 15}
Restriktion : (3 ≤ Width) ∧ odd(Width)

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert : 9
Werteliste : Height ∈ {3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25}
Wertevorschläge : Height ∈ {3, 5, 7, 9, 11, 13, 15}
Restriktion : (3 ≤ Height) ∧ odd(Height)

Beispiel

```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
entropy_image(Image,Entropy1,9,9)
disp_image(Entropy1,WindowHandle).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert entropy_image den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

entropy_image ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

entropy_gray

Siehe auch

energy_gabor, entropy_gray

Modul

Image filters

texture_laws (Image : ImageTexture : FilterTypes, Shift, FilterSize :)

Texturfilter nach Laws.

texture_laws führt eine oder mehrere Texturtransformationen (nach Laws) durch. Dazu wird das Bild in der Eingabekomponente der Eingabeobjekte mit einem (oder mehreren) der Filtern gefaltet. Bei den Filtern handelt es sich um:

9 verschiedene 3x3-Matrizen, die sich aus folgenden drei Vektoren berechnen lassen:

$$\begin{array}{lcl} \mathrm{l} & = & [\ 1 \quad 2 \quad 1] \\ \mathrm{e} & = & [-1 \quad 0 \quad 1] \\ \mathrm{s} & = & [-1 \quad 2 \quad -1] \end{array}$$

25 verschiedene 5x5-Matrizen, die sich aus folgenden fünf Vektoren berechnen lassen:

$$\begin{array}{lcl} \mathrm{l} & = & [\ 1 \quad 4 \quad 6 \quad 4 \quad 1] \\ \mathrm{e} & = & [-1 \quad -2 \quad 0 \quad 2 \quad 1] \\ \mathrm{s} & = & [-1 \quad 0 \quad 2 \quad 0 \quad -1] \\ \mathrm{r} & = & [\ 1 \quad -4 \quad 6 \quad -4 \quad 1] \\ \mathrm{w} & = & [-1 \quad 2 \quad 0 \quad -2 \quad 1] \end{array}$$

36 verschiedene 7x7-Matrizen, die sich aus folgenden sechs Vektoren berechnen lassen:

$$\begin{array}{lcr} \mathrm{l} & = & [\ 1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1] \\ \mathrm{e} & = & [-1 \quad -4 \quad -5 \quad 0 \quad 5 \quad 4 \quad 1] \\ \mathrm{s} & = & [-1 \quad -2 \quad 1 \quad 4 \quad 1 \quad -2 \quad -1] \\ \mathrm{r} & = & [-1 \quad -2 \quad -1 \quad 4 \quad -1 \quad -2 \quad -1] \\ \mathrm{w} & = & [-1 \quad 0 \quad 3 \quad 0 \quad -3 \quad 0 \quad 1] \\ \mathrm{o} & = & [-1 \quad 6 \quad -15 \quad 20 \quad -15 \quad 6 \quad -1] \end{array}$$

Bei dem meisten Filtern ist eine Reduktion des Grauwerte durch einen `Shift` durchzuführen. Auf diese Weise wird im Ausgabebild der Unterschied zwischen zwei Texturen des Eingabebildes vergleichbarer, sofern es sich um den geeigneten Filter handelt.

Der Name der Filter setzt sich aus den Buchstaben der beiden Vektoren zusammen (z.B. 'es'). Dabei gibt des erste Buchstabe die Faltung in Spaltenrichtung, der zweite die Faltung in Zeilenrichtung, an.

Wird mehr als ein Filter angegeben, so wird aus jedem (einkanligen) Eingabebild ein mehrkanaliges Ausgabebild erzeugt, bei dem jeder Kanal einer der angegebenen Transformationen entspricht.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Bilder, die texturtransformiert werden sollen.
- ▷ **`ImageTexture`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Texturbilder.
- ▷ **`FilterTypes`** (input_control) ..... string(-array) ⇝ *string*
  Gewünschte Filter (Name oder Nummer).
  Defaultwert : 'el'
  Wertevorschläge : `FilterTypes` ∈ {'ll', 'le', 'ls', 'lr', 'lw', 'lo', 'el', 'ee', 'es', 'er', 'ew', 'eo', 'sl', 'se', 'ss', 'sr', 'sw', 'so', 'rl', 're', 'rs', 'rr', 'rw', 'ro', 'wl', 'we', 'ws', 'wr', 'ww', 'wo', 'ol', 'oe', 'os', 'or', 'ow', 'oo'}
- ▷ **`Shift`** (input_control) ..... integer ⇝ *integer*
  Shift der Grauwerte zur Dynamikverringerung.
  Defaultwert : 2
  Werteliste : `Shift` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- ▷ **`FilterSize`** (input_control) ..... integer ⇝ *integer*
  Größe des Filterkerns.
  Defaultwert : 5
  Werteliste : `FilterSize` ∈ {3, 5, 7}

Beispiel

```
/* 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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `texture_laws` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`texture_laws` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`mean_image`, `gauss_image`, `median_image`, `histo_2dim`, `learn_ndim_norm`, `learn_ndim_box`, `threshold`

Alternativen

`convol_image`

Siehe auch

`class_2dim_sup`, `class_ndim_norm`

Literatur

Laws, K.I. “Textured image segmentation”; Ph.D. dissertation, Dept. of Engineering, Univ. Southern California, 1980

Modul

Image filters

3.13 Verbesserung

`emphasize` `( Image : ImageEmphasize : MaskWidth, MaskHeight, Factor : )`

Kontrast des Bildes verbessern.

`emphasize` hebt hochfrequente Anteile im Bild hervor (Kanten und Ecken). Das Ergebnisbild erscheint daher schärfer.

Die Prozedur führt zunächst eine Filterung mit dem Tiefpaß (`mean_image`) durch. Aus den so gewonnenen Grauwerten ($mean$) und den Originalgrauwerten ($orig$) werden die Ergebnisgrauwerte (res) wie folgt berechnet:

$$res := round((orig - mean) * \texttt{Factor}) + orig$$

`Factor` dient somit als Maß für die Kontraststeigerung. Die Trennfrequenz wird über die Größe der Filtermatrix festgelegt. Je größer die Matrix ist, desto niedriger wird die Trennfrequenz.

Als Randbehandlungen werden die Grauwerte an den Bildrändern gespiegelt. Über- bzw. Unterlauf von Grauwerten wird beschnitten.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Zu verbesserndes Bild.
- ▷ **`ImageEmphasize`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Kontrastverstärktes Bild.
- ▷ **`MaskWidth`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Tiefpaßmaske.
  Defaultwert : 7
  Wertevorschläge : `MaskWidth` ∈ {3, 5, 7, 9, 11, 15, 21, 25, 31, 39}
  Typischer Wertebereich : 3 ≤ `MaskWidth` ≤ 201
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
Höhe der Tiefpaßmaske.
Defaultwert : 7
Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9, 11, 15, 21, 25, 31, 39}
Typischer Wertebereich : 3 ≤ MaskHeight ≤ 201
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2

▷ **Factor** (input_control) ..... real ⇝ *real*
Stärke der Kontrastanhebung.
Defaultwert : 1.0
Wertevorschläge : Factor ∈ {0.3, 0.5, 0.7, 1.0, 1.4, 1.8, 2.0}
Typischer Wertebereich : 0.0 ≤ Factor ≤ 20.0 (sqrt)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.2
Restriktion : (0 < Factor) ∧ (Factor < 20)

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert emphasize den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system (::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

emphasize ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

mean_image, sub_image, laplace, add_image

Siehe auch

mean_image, highpass_image

Modul

Image filters

equ_histo_image (Image : ImageEquHisto : :)

Histogrammlinearisierung von Bildern

equ_histo_image dient der Kontrastverbesserung. Ausgangspunkt dafür ist das Histogramm der Eingabebilder. Es wird folgende einfache Grauwerttransformation $f(g)$ durchgeführt:

$$f(g) = 255 \sum_{x=0\ldots g} h(x)$$

Dabei bezeichnet $h(x)$ die relative Häufigkeit des Auftretens des Grauwertes x. Diese Transformation linearisiert das kumulative Histogramm. Maxima im ursprünglichen Histogramm werden „gespreizt“ und damit der Kontrast in Bildregionen mit diesen häufig auftretenden Grauwerten erhöht. Vermeintlich homogene Bereiche erhalten so besser sichtbare Strukturen. Andererseits wird natürlich auch das Rauschen im Bild entsprechend verstärkt. Minima im ursprünglichen Histogramm werden dual dazu „gestaucht“. Das transformierte Histogramm enthält dann

zwar Lücken, die verbleibenden verwendeten Grauwerte treten aber etwa gleich häufig auf („histogram equalization“).

Achtung

`equ_histo_image` dient in erster Linie der optischen Aufbereitung von Bildern für einen menschlichen Betrachter. So kann beispielsweise die durchgeführte (lokale) Kontrastspreizung in der Folge zur Detektion von Scheinkanten führen.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu verbesserndes Bild.
- ▷ **`ImageEquHisto`** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Bild mit linearisierten Grauwerten.

Parallelisierungsinformation

`equ_histo_image` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`scale_image`, `scale_image_max`, `illuminate`

Siehe auch

`scale_image`

Literatur

R.C. Gonzales, P. Wintz: „Digital Image Processing“; Second edition; Addison Wesley; 1987.

Modul

Image filters

`illuminate ( Image : ImageIlluminate : MaskWidth, MaskHeight, Factor : )`

Ausleuchten von Bildern.

`illuminate` dient der Kontrastverbesserung. Sehr dunkle Bildteile werden besser „ausgeleuchtet“, sehr helle „abgedunkelt“. Sei $orig$ der Original Grauwert und $mean$ der mittels `mean_image` und Filtergröße `MaskHeight` x `MaskWidth` ermittelte zugehörige Grauwert des tiefpaßgefilterten Bildes. Bei byte-Bildern ist val gleich 127, bei int2-Bildern ist val gleich dem Mittelwert. Dann ist der Ergebnisgrauwert new:

$$new := round((val - mean) * \texttt{Factor} + orig)$$

Der Tiefpaß sollte dabei recht groß dimensioniert werden (30 x 30 bis 200 x 200). Vernünftige Parameterkombinationen sind etwa:

MaskHeight	MaskWidth	Factor
40	40	0.55
100	100	0.7
150	150	0.8

d.h. je größer die Tiefpaßmaske gewählt wird, desto größer sollte auch `Factor` eingestellt werden.

Hingewiesen sei noch auf folgenden „Scheinwerfereffekt“: Befindet sich in einem Bild beispielsweise ein dunkles Objekt vor einer hellen Wand, so wird das Objekt, in unmittelbarer Nähe der Objektkontur aber auch die ohnehin schon helle Wand durch die Anwendung von `illuminate` aufgehellt. Dies entspricht in etwa dem Effekt, der sich durch Anstrahlen des Objekts mit einem starken Scheinwerfer ergeben. Analoges gilt auch für helle Objekte vor einem dunklen Hintergrund. Der fiktive „Scheinwerfer“ verdunkelt nun allerdings Objekte.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  Zu verbesserndes Bild.
- ▷ **ImageIlluminate** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2
  „Ausgeleuchtetes“ Bild.
- ▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
  Breite der Tiefpaßmaske.
  Defaultwert : 101
  Wertevorschläge : `MaskWidth` ∈ {31, 41, 51, 71, 101, 121, 151, 201}
  Typischer Wertebereich : 3 ≤ `MaskWidth` ≤ 299
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 10
- ▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Tiefpaßmaske.
  Defaultwert : 101
  Wertevorschläge : `MaskHeight` ∈ {31, 41, 51, 71, 101, 121, 151, 201}
  Typischer Wertebereich : 3 ≤ `MaskHeight` ≤ 299
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 10
- ▷ **Factor** (input_control) ..... real ⇝ *real*
  Damit wird der „Korrekturgrauwert“, der zu den Original Grauwerten addiert wird, skaliert.
  Defaultwert : 0.7
  Wertevorschläge : `Factor` ∈ {0.3, 0.5, 0.7, 1.0, 1.5, 2.0, 3.0, 5.0}
  Typischer Wertebereich : 0.0 ≤ `Factor` ≤ 5.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.2
  Restriktion : (`0` < `Factor`) ∧ (`Factor` < `5`)

Beispiel

```
read_image(Image,'fabrik')
disp_image(Image,WindowHandle)
illuminate(Image,Better,40,40,0.55)
disp_image(Better,WindowHandle).
```

Ergebnis

`illuminate` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`illuminate` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`

Alternativen

`scale_image_max`, `equ_histo_image`, `mean_image`, `sub_image`

Siehe auch

`emphasize`, `gray_histo`

Modul

Image filters

scale_image_max (Image : ImageScaleMax : :)

Maximale Grauwertspreizung auf den Wertebereich **0** *bis* **255**.

scale_image_max berechnet Minimum und Maximum und skaliert das Bild hiermit auf den maximalen Wertebereich eines Byte-Bildes. Hierdurch wird die Dynamik (Wertebereich) voll ausgeschöpft. Die Anzahl der unterschiedlichen Graustufen ändert sich nicht. Es entsteht aber i.a. ein visuell besserer Eindruck. Bei Bildern vom Typ **real**, **int2** und **int4** werden die Grauwerte auf den Bereich **0** bis **255** skaliert und als **byte**-Bild ausgegeben.

Achtung

Die Ausgabe ist immer ein Ergebnisbild vom Typ **byte**.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte / int2 / int4 / real
  Zu skalierendes Bild.
- ▷ **ImageScaleMax** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Kontrastverstärktes Bild.

Parallelisierungsinformation

scale_image_max ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Nachfolgerfunktionen

disp_image

Alternativen

equ_histo_image, scale_image, illuminate, convert_image_type

Siehe auch

min_max_gray, gray_histo

Modul

Image filters

3.14 Wienerfilter

gen_psf_defocus (: Psf : PSFwidth, PSFheight, Blurring :)

Erzeugung der Impulsantwort einer gleichmäßigen Defokussierung.

gen_psf_defocus erzeugt die Impulsantwort (im Ortsraum) einer gleichmäßigen Defokussierung als Bildobjekt vom Bildtyp ’real’. Seine Breite bzw. Höhe läßt sich durch PSFwidth und PSFheight angeben. Die Bildbeeinträchtigung betrifft das gesamte Bild gleichermaßen. Ihr Ausmaß wird durch Blurring angegeben. Dieser Parameter beschreibt den sogenannten „Defokussierungsradius“. Durch die falsche Fokussierung wird jeder Bildpunkt (im Prinzip) auf eine kleine Kreisfläche abgebildet. Dadurch wird das Ergebnisbild unscharf. Der Radius dieses Kreises wird also durch Blurring (in Bildpunkten) festgelegt. Falls er kleiner als null eingegeben wird, so wird sein Betrag verwendet. Das erzeugte float-Bild enthält die gewünschte Impulsantwort im Ortsraum, wobei ihre Darstellung voraussetzt, daß sich der Nullpunkt in der Bildecke („links oben“) befindet. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

Aufgrund dieser Darstellungsform ist es möglich, die erzeugte Impulsantwort direkt als Eingabe der Routine wiener_filter zu verwenden, um ein defokussiertes Bild mittels Wiener-Filterung zu restaurieren.

Parameter

- ▷ **Psf** (output_object) . . . . . image ⇝ *Hobject* : real
  Impulsantwort einer gleichmäßigen Defokussierung.
- ▷ **PSFwidth** (input_control) . . . . . integer ⇝ *integer*
  Breite der Impulsantwort-Bildmatrix.
  Defaultwert : 256
  Wertevorschläge : PSFwidth ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ PSFwidth
- ▷ **PSFheight** (input_control) . . . . . integer ⇝ *integer*
  Höhe der Impulsantwort-Bildmatrix.
  Defaultwert : 256
  Wertevorschläge : PSFheight ∈ {128, 256, 512, 1024}
  Typischer Wertebereich : 1 ≤ PSFheight
- ▷ **Blurring** (input_control) . . . . . real ⇝ *real*
  Stärke der Beeinträchtigung (Defokussierungsradius).
  Defaultwert : 5.0
  Wertevorschläge : Blurring ∈ {1.0, 5.0, 10.0, 15.0, 18.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_psf_defocus den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

gen_psf_defocus ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

simulate_motion, gen_psf_motion

Mögliche Nachfolgerfunktionen

simulate_defocus, wiener_filter, wiener_filter_ni

Siehe auch

simulate_defocus, gen_psf_motion, simulate_motion, wiener_filter, wiener_filter_ni

Literatur

Reginald L. Lagendijk, Jan Biemond: Iterative Identification and Restoration of Images, Kluwer Academic Publishers Boston/Dordrecht/London, 1991

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Modul

Wiener filter

gen_psf_motion (: Psf : PSFwidth, PSFheight, Blurring, Angle, Type :)

Erzeugung der Impulsantwort einer geradlinigen Bewegungsverschlechterung.

gen_psf_motion erzeugt die Impulsantwort (im Ortsraum) einer geradlinigen Bewegungsverschlechterung als Bildobjekt vom Bildtyp ’real’. Seine Breite bzw. Höhe läßt sich durch PSFwidth und PSFheight angeben. Die beschriebene Bewegung erfolgt entlang einer Linie. Ihre Richtung ist durch den Winkel zur x-Achse (gegen den Uhrzeigersinn) festgelegt. Dieser wird mit Angle im Gradmaß eingegeben. Zur Beschreibung von unterschiedlichen Bewegungsformen, stehen fünf Prototypen für die Impulsantwort zur Verfügung. Welcher Prototyp Verwendung findet, regelt Type. Folgende Werte sind möglich:

1. Umgekehrte Rampe (beschreibt in etwa eine Beschleunigung)
2. Umgekehrtes Trapezoid (beschreibt in etwa starke Beschleunigung)
3. Rechtecksfunktion (beschreibt (genau) konstante Geschwindigkeit)
4. Normales Trapezoid (beschreibt in etwa starkes Abbremsen)
5. Normale Rampe (beschreibt in etwa Abbremsen)

(Für alle anderen Werte wird automatisch die Rechtecksfunktion verwendet.) Die Bewegungsverschlechterung betrifft auf alle Fälle das gesamte eingegebene Bild. Ihr Ausmaß ist durch `Blurring` festgelegt. Dieser Parameter beschreibt, wieviele Bildpunkte, die auf der Geraden der Bewegungsrichtung hintereinander liegen, durch die Bewegungsverschlechterung betroffen sind. Die Anzahl der betroffenen Punkte wird durch die Bewegungsgeschwindigkeit, -beschleunigung und -zeitdauer bestimmt. Falls `Blurring` kleiner als null angegeben ist, so wird die Impulsantwort einer entsprechend starken Bewegung in entgegengesetzter Richtung erzeugt. Falls `Angle` kleiner als null eingegeben wird, so wird der Winkel im Uhrzeigersinn interpretiert. Falls `Angle` größr als 360 bzw. kleiner als -360 angegeben ist, so wird er modulo(360) in das Intervall $[0..360]$ bzw. $[-360..0]$ zurückgerechnet. Das erzeugte float-Bild enthält die gewünschte Impulsantwort im Ortsraum, wobei ihre Darstellung voraussetzt, daß sich der Nullpunkt in der Bildecke („links oben“) befindet. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2)-1$, $yb = 0..(M/2)-1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N-1$, $yb = 0..(M/2)-1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2)-1$, $yb = M/2..M-1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N-1$, $yb = M/2..M-1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

Aufgrund dieser Darstellungsform ist es möglich, die erzeugte Impulsantwort direkt als Eingabe der Routine `wiener_filter` zu verwenden, um ein bewegungsverschlechtertes Bild mittels Wiener-Filterung zu restaurieren.

Parameter

▷ **`Psf`** (output_object) ..... image ⇝ *Hobject* : real
Impulsantwort einer Bewegungsverschlechterung.

▷ **`PSFwidth`** (input_control) ..... integer ⇝ *integer*
Breite der Impulsantwort-Bildmatrix.
Defaultwert : 256
Wertevorschläge : `PSFwidth` ∈ {128, 256, 512, 1024}
Typischer Wertebereich : $1 \leq$ `PSFwidth`

▷ **`PSFheight`** (input_control) ..... integer ⇝ *integer*
Höhe der Impulsantwort-Bildmatrix.
Defaultwert : 256
Wertevorschläge : `PSFheight` ∈ {128, 256, 512, 1024}
Typischer Wertebereich : $1 \leq$ `PSFheight`

▷ **`Blurring`** (input_control) ..... real ⇝ *real*
Stärke der Bewegungsverschlechterung.
Defaultwert : 20.0
Wertevorschläge : `Blurring` ∈ {5.0, 10.0, 20.0, 30.0, 40.0}

▷ **`Angle`** (input_control) ..... integer ⇝ *integer*
Winkel zwischen der Bewegungsrichtung und der x-Achse (gegen den Uhrzeigersinn).
Defaultwert : 0
Wertevorschläge : `Angle` ∈ {0, 45, 90, 180, 270}

▷ **`Type`** (input_control) ..... integer ⇝ *integer*
Impulsantwort-Prototyp bzw. Form der Bewegung
Defaultwert : 3
Werteliste : `Type` ∈ {1, 2, 3, 4, 5}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_psf_motion` den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`gen_psf_motion` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_psf_motion, simulate_defocus, gen_psf_defocus

Mögliche Nachfolgerfunktionen

simulate_motion, wiener_filter, wiener_filter_ni

Siehe auch

simulate_motion, simulate_defocus, gen_psf_defocus, wiener_filter, wiener_filter_ni

Literatur

Anil K. Jain:Fundamentals of Digital Image Processing, Prentice-Hall International Inc., Englewood Cliffs, New Jersey, 1989

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Kha-Chye Tan, Hock Lim, B. T. G. Tan:„Restoration of Real-World Motion-Blurred Images“;S. 291-299 in: CVGIP Graphical Models and Image Processing, Vol. 53, No. 3, May 1991

Modul

Wiener filter

simulate_defocus (Image : DefocusedImage : Blurring :)

Simulation einer gleichmäßigen Defokussierung.

simulate_defocus simuliert eine Bildbeeinträchtigung. Ihre Ursache liegt in einer falschen Fokussierung des aufnehmenden Linsensystems. Die daraus resultierende Bildverschlechterung betrifft das gesamte Eingabebild gleichermaßen. Ihr Ausmaß wird durch Blurring angegeben. Dieser Parameter beschreibt den sogenannten „Defokussierungsradius“. Durch die falsche Fokussierung wird jeder Bildpunkt (im Prinzip) auf eine kleine Kreisfläche abgebildet. Dadurch wird das Ergebnisbild unscharf. Der Radius dieses Kreises wird also durch Blurring (in Bildpunkten) festgelegt. Falls er kleiner als null eingegeben wird, so wird sein Betrag verwendet. Die Simulation der Bildbeeinträchtigung erfolgt durch eine Faltung vom Eingabebild mit einer Impulsantwort, die eine gleichmäßige Defokussierung beschreibt. simulate_defocus erzeugt die benötigte Impulsantwort und multipliziert ihre (diskrete) Fouriertransformierte mit der des Eingabebildes. Das in den Ortsraum zurücktransformierte Produkt stellt das defokussierte Bild dar.

Achtung

Da zum Zwecke der Bildverschlechterungs-Simulation die diskrete Fouriertransformierte des Eingabebildes berechnet wird, müssen die Bildbreite und -länge Potenzen von 2 entsprechen (z.B. 64, 128, 256, 512).

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : any
  Bild, das beeinträchtigt werden soll.
- ▷ **DefocusedImage** (output_object) ..... image ⇝ *Hobject* : real
  Defokussiertes Bild.
- ▷ **Blurring** (input_control) ..... real ⇝ *real*
  Stärke der Beeinträchtigung (Defokussierungsradius).
  Defaultwert : 5.0
  Wertevorschläge : Blurring ∈ {1.0, 5.0, 10.0, 15.0, 18.0}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert simulate_defocus den Wert 2 (H_MSG_TRUE). Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Parallelisierungsinformation

simulate_defocus ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_psf_defocus, simulate_motion, gen_psf_motion

Mögliche Nachfolgerfunktionen

wiener_filter, wiener_filter_ni

Siehe auch

gen_psf_defocus, simulate_motion, gen_psf_motion

Literatur

Reginald L. Lagendijk, Jan Biemond: Iterative Identification and Restoration of Images, Kluwer Academic Publishers Boston/Dordrecht/London, 1991

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Modul

Wiener filter

simulate_motion (Image : MovedImage : Blurring, Angle, Type :)

Simulation einer geradlinigen Bewegungsverschlechterung.

simulate_motion simuliert eine Bildbeeinträchtigung. Ihre Ursache liegt in einer relativen Bewegung zwischen dem aufzunehmenden Objekt und dem Aufnahmesystem während der Aufnahme. Die Bewegung erfolgte dabei entlang einer Linie. Ihre Richtung ist durch den Winkel zur x-Achse (gegen den Uhrzeigersinn) festgelegt. Dieser wird mit Angle im Gradmaß eingegeben. Die Simulation erfolgt durch eine Faltung vom Eingabebild mit einer Impulsantwort, die die jeweilige Bewegungsart beschreibt. simulate_motion erzeugt die benötigte Impulsantwort und multipliziert ihre (diskrete) Fouriertransformierte mit der des Eingabebildes. Das in den Ortsraum zurücktransformierte Produkt stellt das bewegungsverschlechterte Bild dar. Zur Beschreibung von unterschiedlichen Bewegungsformen, stehen fünf Prototypen für die Impulsantwort zur Verfügung. Welcher Prototyp Verwendung findet, regelt Type. Folgende Werte sind möglich:

1. Umgekehrte Rampe (beschreibt in etwa eine Beschleunigung)
2. Umgekehrtes Trapezoid (beschreibt in etwa starke Beschleunigung)
3. Rechtecksfunktion (beschreibt (genau) konstante Geschwindigkeit)
4. Normales Trapezoid (beschreibt in etwa starkes Abbremsen)
5. Normale Rampe (beschreibt in etwa Abbremsen)

(Für alle anderen Werte wird automatisch die Rechtecksfunktion verwendet.) Die Bewegungsverschlechterung betrifft auf alle Fälle das gesamte eingegebene Bild. Ihr Ausmaß ist durch Blurring festgelegt. Dieser Parameter beschreibt, wieviele Bildpunkte, die auf einer Geraden in Bewegungsrichtung hintereinander liegen, durch die Bewegungsverschlechterung betroffen sind. Die Anzahl der betroffenen Punkte wird durch die Bewegungsgeschwindigkeit, -beschleunigung und -zeitdauer bestimmt. Falls Blurring kleiner als null angegeben ist, so wird eine entsprechend starke Bewegung in entgegengesetzter Richtung simuliert. Falls Angle kleiner als null eingegeben wird, so wird der Winkel im Uhrzeigersinn interpretiert. Falls Angle größer als 360 bzw. kleiner als -360 angegeben ist, so wird er modulo(360) in das Intervall $[0..360]$ bzw. $[-360..0]$ zurückgerechnet.

Achtung

Da zum Zwecke der Bildverschlechterungs-Simulation die diskrete Fouriertransformierte des Eingabebildes berechnet wird, müssen die Bildbreite und -länge Potenzen von 2 entsprechen (z.B. 64, 128, 256, 512).

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : any
  Bild, das beeinträchtigt werden soll.
- ▷ **MovedImage** (output_object) ..... image ⇝ *Hobject* : real
  Bewegungsverschlechtertes Bild.
- ▷ **Blurring** (input_control) ..... real ⇝ *real*
  Stärke der Bewegungsverschlechterung.
  Defaultwert : 20.0
  Wertevorschläge : Blurring ∈ {5.0, 10.0, 20.0, 30.0, 40.0}
- ▷ **Angle** (input_control) ..... integer ⇝ *integer*
  Winkel zwischen der Bewegungsrichtung und der x-Achse (gegen den Uhrzeigersinn).
  Defaultwert : 0
  Wertevorschläge : Angle ∈ {0, 45, 90, 180, 270}

▷ **Type** (input_control) . . . . . integer ⇝ *integer*
Implsantwort-Prototyp bzw. Form der Bewegung.
Defaultwert : 3
Werteliste : Type ∈ {1, 2, 3, 4, 5}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert simulate_motion den Wert 2 (H_MSG_TRUE). Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Parallelisierungsinformation

simulate_motion ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_psf_motion, gen_psf_motion

Mögliche Nachfolgerfunktionen

simulate_defocus, wiener_filter, wiener_filter_ni

Siehe auch

gen_psf_motion, simulate_defocus, gen_psf_defocus

Literatur

Anil K. Jain:Fundamentals of Digital Image Processing, Prentice-Hall International Inc., Englewood Cliffs, New Jersey, 1989

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995.

Kha-Chye Tan, Hock Lim, B. T. G. Tan:„Restoration of Real-World Motion-Blurred Images“;S. 291-299 in: CVGIP Graphical Models and Image Processing, Vol. 53, No. 3, May 1991

Modul

Wiener filter

wiener_filter (Image, Psf, FilteredImage : RestoredImage : :)

Bildrestauration mittels Wiener-Filterung.

wiener_filter liefert eine Abschätzung des unbeeinträchtigten Originalbildes. Die Abschätzung ist im Sinne eines minimalen, quadratischen Fehlers zwischen dem Originalbild und seiner Abschätzung optimal. Diese Abschätzung stellt eine restaurierte Version eines durch verschiedene Einflüsse (z.B. relative Bewegung zwischen aufzunehmendem Objekt und Aufnahmesystem, falsch fokussiertes Linsensystem, atmosphärische Störungen usw.) beeinträchtigten Bildes dar. Ausgangspunkt ist das beeinträchtigte Bild, das als Ausgabe eines gestörten, linearen Systems interpretiert wird. Das Verhalten eines linearen Systems, ist durch seine Impulsantwort bestimmt. Folgerichtig wird davon ausgegangen, daß das beeinträchtigte Bild das Ergebnis einer Faltung von Originalbild und Impulsantwort des linearen Systems darstellt. Diese Impulsantwort beschreibt die Bildaufnahme und die dabei aufgetretenen Störungen. Um auch Störungsursachen stochastischer Natur (Rauschen) einbeziehen zu können, wird zu dem Faltungsergebnis ein Rauschterm addiert. Insgesamt ergibt sich also das beeinträchtigte Bild aus

$$[Faltung(ImpulsantwortundOriginalbild)] + Rauschterm$$

Der Rauschterm beinhaltet zwei Komponenten, von denen jeweils eine das bildabhängige und bildunabhängige Rauschen beschreibt. Die Routine wiener_filter benötigt eine Bildversion, die (in der Vorverarbeitung durch den Anwender) mit einem Rauschfilter bearbeitet wurde. Diese rauschreduzierte Bildversion wird von wiener_filter dafür verwendet, die spektralen Leistungsdichten vom Rauschen und dem Originalbild abzuschätzen. Weiterhin ist durch den Anwender eine Impulsantwort zu erzeugen, die die spezielle Beeinträchtigung im Bild beschreibt. Die Impulsantwort wird als Bildmatrix eingegeben (Bildtyp: ’real’). Sie stellt die Impulsantwort im Ortsraum dar und muß dahingehend in die Bildmatrix eingetragen werden, daß der Nullpunkt in der Bildecke „links oben“ liegt. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$

- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N-1$, $yb = 0..(M/2)-1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2..-1$ und $y = -1..-M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2)-1$, $yb = M/2..M-1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N-1$, $yb = M/2..M-1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2..-1$ und $y = M/2..1$

Da die Durchführung der Wiener-Filterung im Fourierraum erfolgt, also die diskrete Fouriertransformierte des zu filternden Bildes berechnet werden muß, können nur (rechteckige) Bilder verarbeitet werden, deren Seitenlängen einer Potenz von 2 entsprechen. `wiener_filter` verwendet folgende Strategie zur Erzeugung des restaurierten Bildes:

- Abschätzen der spektralen Leistungsdichte des Originalbildes aus der des rauschgefilterten Eingabebildes.
- Subtrahieren der Bildwerte des rauschgefilterten Eingabebildes von denen des belassenen Eingabebildes ergibt eine Abschätzung der Rauschstärke an jedem Bildpunkt. Damit läßt sich die spektrale Leistungsdichte des Rauschens für jeden Bildpunkt abschätzen.
- Mit Hilfe des Quotienten der Leistungsdichten vom Rauschen und Originalbild und der Fouriertransformierten der Impulsantwort wird das Wiener-Filter gebildet.
- Durchführung der Wiener-Filterung durch Faltung von dem Eingabebild und der Frequenzantwort des Filters (= Multiplikation von deren diskreten Fouriertransformierten).
- Rücktransformiertes Ergebnis stellt das restaurierte Bild dar.

Das Restaurationsergebnis ist vom Bildtyp ’real’.

Achtung

Die Impulsantwort `Psf` muß vom Bildtyp ’real’ sein und dieselbe Bildbreite und -länge wie `Image` und `FilteredImage` besitzen.

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : any
  Beeinträchtigtes Bild, das gefiltert werden soll.
- ▷ **`Psf`** (input_object) ..... image ⇝ *Hobject* : real
  Impulsantwort (Point Spread Function) der Bildbeeinträchtigung (im Ortsraum mit dem Nullpunkt in der Bildecke angegeben).
- ▷ **`FilteredImage`** (input_object) ..... image ⇝ *Hobject* : any
  Beeinträchtigtes, durch ein Rauschfilter vorverarbeitetes Bild, das gefiltert werden soll.
- ▷ **`RestoredImage`** (output_object) ..... image ⇝ *Hobject* : real
  Restauriertes Bild.

Beispiel

```
/* Restoration of a noisy image (size=256x256), that was blurred by motion*/
Hobject object;
Hobject restored;
Hobject psf;
Hobject noisefiltered;
/* 1. Generate a Point-Spread-Function for a motion-blur with       */
/*    parameter a=10 and direction along the x-axis                 */
gen_psf(&psf,256,256,10,0,3);
/* 2. Noisefiltering of the image                                   */
median_image(object,&noisefiltered,"circle",2,0);
/* 3. Wiener-filtering                                              */
wiener_filter(object,psf,noisefiltered,&restored);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `wiener_filter` den Wert 2 (H_MSG_TRUE). Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Parallelisierungsinformation

`wiener_filter` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_psf_motion`, `simulate_motion`, `simulate_defocus`, `gen_psf_defocus`

Alternativen

`wiener_filter_ni`

Siehe auch

`simulate_motion`, `gen_psf_motion`, `simulate_defocus`, `gen_psf_defocus`

Literatur

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995

Azriel Rosenfeld, Avinash C. Kak: Digital Picture Processing, Computer Science and Aplied Mathematics, Academic Press New York/San Francisco/London 1982

Modul

Wiener filter

```
wiener_filter_ni ( Image, Psf,
NoiseRegion :  RestoredImage :  MaskWidth, MaskHeight : )
```

Bildrestauration mittels Wiener-Filterung.

`wiener_filter_ni` (ni = noise-estimation integrated) liefert eine Abschätzung des unbeeinträchtigten Originalbildes. Die Abschätzung ist im Sinne eines minimalen, quadratischen Fehlers zwischen dem Originalbild und seiner Abschätzung optimal. Diese Abschätzung stellt eine restaurierte Version eines durch verschiedene Einflüsse (z.B. relative Bewegung zwischen aufzunehmendem Objekt und Aufnahmesystem, falsch fokussiertes Linsensystem, atmosphärische Störungen usw.) beeinträchtigten Bildes dar. Ausgangspunkt ist das beeinträchtigte Bild, das als Ausgabe eines gestörten, linearen Systems interpretiert wird. Das Verhalten eines linearen Systems, ist durch seine Impulsantwort bestimmt. Folgerichtig wird davon ausgegangen, daß das beeinträchtigte Bild das Ergebnis einer Faltung von Originalbild und Impulsantwort des linearen Systems darstellt. Diese Impulsantwort beschreibt die Bildaufnahme und die dabei aufgetretenen Störungen. Um auch Störungsursachen stochastischer Natur (Rauschen) einbeziehen zu können, wird zu dem Faltungsergebnis ein Rauschterm addiert. Insgesamt ergibt sich also das beeinträchtigte Bild aus

$$[Faltung(ImpulsantwortundOriginalbild)] + Rauschterm$$

Der Rauschterm beinhaltet zwei Komponenten, von denen jeweils eine das bildabhängige und bildunabhängige Rauschen beschreibt. Die Routine `wiener_filter_ni` schätzt die Stärke des Rauschens innerhalb einer Bildregion ab. Dies geschieht, indem mittels einer (ungewichteten) Median-Filterung (mit einer rechteckigen Filtermaske) eine rauschreduzierte Version der Bildregion erzeugt wird. Die Stärke des Rauschens ergibt sich punktweise durch Subtraktion des Median-Wertes vom dortigen Grauwert. Daraus wird die durchschnittliche Rauschamplitude innerhalb der Bildregion berechnet. Als zweites wird der durchschnittliche Grauwert des gesamten zu filternden Bildes gebildet. Hiermit läßt sich das durchschnittliche Verhältnis vom Rauschen und Bildwert abschätzen. Das Quadrat dieses Quotienten ergibt wiederum eine Abschätzung des Verhältnisses der spektralen Leistungsdichten vom Rauschen und Bild. Bei der Wiener-Filterung mit `wiener_filter_ni` wird also (im Gegensatz zu der Version von `wiener_filter`) angenommen, daß dieses Verhältnis innerhalb des gesamten Bildraums näherungsweise konstant bleibt. Die Bildregion, die die Grundlage der Rauschabschätzung bildet, muß durch den Anwender in einer Vorverarbeitung bestimmt werden. Dabei ist zu beachten, daß eine möglichst homogene Region auszuwählen ist, da starke Texturen oder Kanten große Fehler bei der Rauschabschätzung verursachen können. Um dem Anwender einen (begrenzten) Einfluß auf die Abschätzung zu ermöglichen, sind die Breite bzw. Höhe der rechteckigen Filtermaske als Eingabe-Steuerparameter anzugeben. Weiterhin ist durch den Anwender eine Impulsantwort zu erzeugen, die die spezielle Beeinträchtigung im Bild beschreibt. Die Impulsantwort wird als Bildmatrix eingegeben (Bildtyp: ’real’). Sie stellt die Impulsantwort im Ortsraum dar und muß dahingehend in die Bildmatrix eingetragen werden, daß der Nullpunkt in der Bildecke „links oben“ liegt. Das bedeutet folgende Einteilung der insgesamt NxM-großen Bildmatrix in vier rechteckige Regionen:

- Erstes Rechteck („links oben“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem vierten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 0..N/2$ und $y = 0.. - M/2$
- Zweites Rechteck („rechts oben“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = 0..(M/2) - 1$)
  -entspricht dem dritten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = -1.. - M/2$
- Drittes Rechteck („links unten“): (Bildkoordinaten $xb = 0..(N/2) - 1$, $yb = M/2..M - 1$)
  -entspricht dem ersten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = 1..N/2$ und $y = M/2..0$
- Viertes Rechteck („rechts unten“): (Bildkoordinaten $xb = N/2..N - 1$, $yb = M/2..M - 1$)
  -entspricht dem zweiten Quadranten des kartesischen Koordinatensystems, enthält also die Werte der Impulsantwort für $x = -N/2.. - 1$ und $y = M/2..1$

Da die Durchführung der Wiener-Filterung im Fourierraum erfolgt, also die diskrete Fouriertransformierte des zu filternden Bildes berechnet werden muß, können nur (rechteckige) Bilder verarbeitet werden, deren Seitenlängen einer Potenz von 2 entsprechen. `wiener_filter_ni` verwendet folgende Strategie zur Erzeugung des restaurierten Bildes:

- Abschätzen des durchschnittlichen Verhältnisses der spektralen Leistungsdichten vom Rauschen und Originalbild (siehe oben).
- Mit Hilfe dieses Verhältnisses und der Fouriertransformierten der Impulsantwort wird das Wiener-Filter gebildet.
- Durchführung der Wiener-Filterung durch Faltung von dem Eingabebild und der Frequenzantwort des Filters (= Multiplikation von deren diskreten Fouriertransformierten).
- Rücktransformiertes Produkt stellt das restaurierte Bild dar.

Das Restaurationsergebnis ist vom Bildtyp ’real’.

Achtung

Die Impulsantwort `Psf` muß vom Bildtyp ’real’ sein und dieselbe Bildbreite und -länge wie `Image` besitzen. Die Bildregion zur Rauschabschätzung darf keine Bereiche außerhalb des Bildes umfassen. Die Filtermaskengröße `MaskWidth` und `MaskHeight` muß größer oder gleich 0 und kleiner oder gleich der Bildgröße sein. Werden für `MaskWidth` und `MaskHeight` gerade statt ungerader Werte übergeben, verwendet die Routine an ihrer Stelle die nächstgrößeren ungeraden Werte (damit ist der Schwerpunkt der Filtermaske immer eindeutig bestimmt).

Parameter

▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : any
Beeinträchtigtes Bild, das gefiltert werden soll.

▷ **`Psf`** (input_object) ..... image ⇝ *Hobject* : real
Impulsantwort (Point Spread Function) der Bildbeeinträchtigung (im Ortsraum mit dem Nullpunkt in der Bildecke angegeben).

▷ **`NoiseRegion`** (input_object) ..... region ⇝ *Hobject*
Bildregion, innerhalb der die durchschnittliche Stärke des Rauschens abgeschätzt werden soll.

▷ **`RestoredImage`** (output_object) ..... image ⇝ *Hobject* : real
Restauriertes Bild

▷ **`MaskWidth`** (input_control) ..... integer ⇝ *integer*
Breite der rechteckigen Filtermaske, die bei der Median-Filterung zur internen Rauschabschätzung verwendet wird.
Defaultwert : 3
Wertevorschläge : `MaskWidth` ∈ {3, 5, 7, 9}
Typischer Wertebereich : 0 ≤ `MaskWidth` ≤ width(Image)

▷ **`MaskHeight`** (input_control) ..... integer ⇝ *integer*
Höhe der rechteckigen Filtermaske, die bei der Median-Filterung zur internen Rauschabschätzung verwendet wird.
Defaultwert : 3
Wertevorschläge : `MaskHeight` ∈ {3, 5, 7, 9}
Typischer Wertebereich : 0 ≤ `MaskHeight` ≤ height(Image)

Beispiel

```
/* Restoration of a noisy image (size=256x256), that was blurred by motion*/
Hobject object;
Hobject restored;
Hobject psf;
Hobject noise_region;
/* 1. Generate a Point-Spread-Function for a motion-blur with       */
/*    parameter a=10 and direction of the x-axis                    */
gen_psf(&psf,256,256,10,0,3);
/* 2. Segmentation of a region for the noise-estimation             */
open_window(0,0,256,256,0,"visible",&WindowHandle);
disp_image(object,WindowHandle);
draw_region(&noise_region,draw_region);
/* 3. Wiener-filtering                                              */
wiener_filter_ni(object,psf,noise_region,&restored,3,3);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `wiener_filter_ni` den Wert 2 (H_MSG_TRUE). Bei einer leeren Eingabe wird mit einer entsprechenden Fehlermeldung abgebrochen.

Parallelisierungsinformation

`wiener_filter_ni` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_psf_motion`, `simulate_motion`, `simulate_defocus`, `gen_psf_defocus`

Alternativen

`wiener_filter`

Siehe auch

`simulate_motion`, `gen_psf_motion`, `simulate_defocus`, `gen_psf_defocus`

Literatur

M. Lückenhaus:„Grundlagen des Wiener-Filters und seine Anwendung in der Bildanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik; Lehrstuhl Prof. Radig; 1995

Azriel Rosenfeld, Avinash C. Kak: Digital Picture Processing, Computer Science and Aplied Mathematics, Academic Press New York/San Francisco/London 1982

Modul

Wiener filter

Kapitel 4

Graphik

4.1 Ausgabe

disp_arc (: : WindowHandle, CenterRow, CenterCol, Angle, BeginRow, BeginCol :)

Ausgabe von Bögen in ein Fenster.

disp_arc trägt einen oder mehrere Bögen, die durch den Schwerpunkt (CenterRow,CenterCol), den Winkel zwischen Anfang und Ende des Bogens (Angle in Bogenmaß) und den ersten Punkt (BeginRow,BeginCol) beschrieben werden, in das Ausgabefenster ein. Die Ausgabe des Bogens geschieht im Uhrzeigersinn. Die Parameter für die Ausgabe können - wie bei der Darstellung einer Region (disp_region) - mit den Prozeduren set_color, set_gray, set_draw, etc. bestimmt werden. Es können mehrere Bögen bei einem Aufruf gezeichnet werden, indem jeweils Tupel übergeben werden. Für die Behandlung der Farben bei mehr als einem Bogen, siehe set_color.

Achtung

Der Schwerpunkt muß innerhalb des Fensters liegen. Der Bogen muß mindestens einen Radius von 2 Pixel haben.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **CenterRow** (input_control) . . . . . arc.center.y ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : CenterRow ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ CenterRow ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **CenterCol** (input_control) . . . . . arc.center.x ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : CenterCol ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ CenterCol ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Angle** (input_control) . . . . . arc.angle.rad ⇝ *real* / integer
  Winkel zwischen Anfang und Ende des Bogens in Bogenmaß.
  Defaultwert : 3.1415926
  Wertevorschläge : Angle ∈ {0.0, 0.785398, 1.570796, 3.1415926, 6.283185}
  Typischer Wertebereich : 0.0 ≤ Angle ≤ 6.283185 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Angle > 0.0

▷ **BeginRow** (input_control) ..... arc.begin.y(-array) ⇝ *integer* / real
Zeilenindex des Anfangs des Bogens.
Defaultwert : 32
Wertevorschläge : BeginRow ∈ {0, 64, 128, 256}
Typischer Wertebereich : 0 ≤ BeginRow ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **BeginCol** (input_control) ..... arc.begin.x(-array) ⇝ *integer* / real
Spaltenindex des Anfangs des Bogens.
Defaultwert : 32
Wertevorschläge : BeginCol ∈ {0, 64, 128, 256}
Typischer Wertebereich : 0 ≤ BeginCol ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Beispiel

```
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).
```

Ergebnis

disp_arc liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

disp_arc ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi

Alternativen

disp_circle, disp_ellipse, disp_region, gen_circle, gen_ellipse

Siehe auch

open_window, open_textwindow, set_color, set_draw, set_rgb, set_hsi

Modul

System

disp_arrow (: : WindowHandle, Row1, Column1, Row2, Column2, Size :)

Ausgabe von Pfeilen in ein Fenster.

disp_arrow trägt einen oder mehrere Pfeile mit Anfangskoordinaten (Row1,Column1) und Endkoordinaten (Row2,Column2) in das Ausgabefenster ein. Die Pfeilspitzen befinden sich jeweils an den Endkoordinaten. Der Parameter Size bestimmt die Größe der Pfeilspitze. Besteht der Pfeil nur aus einem Punkt (Anfangspunkt = Endpunkt), wird nichts gezeichnet. Die Prozeduren, die zur Steuerung der Regionenausgabe verwendet werden, legen auch hier die Ausgabeparameter fest (z.B. set_draw, set_color, set_line_width).

Es können mehrere Pfeile bei einem Aufruf gezeichnet werden, indem Tupel von Koordinaten übergeben werden. Für die Behandlung der Farben bei mehr als einem Pfeil, siehe set_color.

Achtung

Die Anfangs und Endpunkte der Pfeile müssen innerhalb des Fensters liegen.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Row1** (input_control) . . . . . line.begin.y(-array) ⇝ *real* / integer
  Zeilenindex des Anfangspunktes.
  Defaultwert : 10.0
  Wertevorschläge : `Row1` ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich : 0.0 ≤ `Row1` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0
- ▷ **Column1** (input_control) . . . . . line.begin.x(-array) ⇝ *real* / integer
  Spaltenindex des Anfangspunktes.
  Defaultwert : 10.0
  Wertevorschläge : `Column1` ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich : 0.0 ≤ `Column1` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0
- ▷ **Row2** (input_control) . . . . . line.end.y(-array) ⇝ *real* / integer
  Zeilenindex des Endpunktes.
  Defaultwert : 118.0
  Wertevorschläge : `Row2` ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich : 0.0 ≤ `Row2` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0
- ▷ **Column2** (input_control) . . . . . line.end.x(-array) ⇝ *real* / integer
  Spaltenindex des Endpunktes.
  Defaultwert : 118.0
  Wertevorschläge : `Column2` ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich : 0.0 ≤ `Column2` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0
- ▷ **Size** (input_control) . . . . . number ⇝ *real* / integer
  Größe der Pfeilspitze.
  Defaultwert : 1.0
  Wertevorschläge : `Size` ∈ {1.0, 2.0, 3.0, 5.0}
  Typischer Wertebereich : 0.0 ≤ `Size` ≤ 20.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0
  Restriktion : `Size` > `0.0`

Beispiel

```
set_color(WindowHandle,['red','green'])
disp_arrow(WindowHandle,[10,10],[10,10],[118,110],[118,118],1.0).
```

Ergebnis

`disp_arrow` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`disp_arrow` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `set_rgb`, `set_hsi`

Alternativen

`disp_line`, `gen_region_polygon`, `disp_region`

Siehe auch

`open_window`, `open_textwindow`, `set_color`, `set_draw`, `set_line_width`

Modul

System

disp_channel (MultichannelImage : : WindowHandle, Channel :)

Ausgabe von mehrkanaligen Bildern.

disp_channel gibt ein Bild in das Ausgabefenster aus. Es können auch mehrere Bilder gleichzeitig ausgegeben werden. Hierbei werden diese nacheinander gezeichnet. Falls die Definitionsbereiche der Bilder sich überlappen, ist nur das zuletzt gezeichnete Bild sichtbar. Der Parameter Channel gibt die Nummer des Kanals an, der ausgegeben werden soll. Bei RGB-Bildern sind die drei Farbkanäle als Tupel anzugeben. Weitere Informationen sind bei disp_image zu finden.

Parameter

- ▷ **MultichannelImage** (input_object) ..... image(-array) ⇝ *Hobject*
  Auszugebende Mehrkanalbilder.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Channel** (input_control) ..... integer(-array) ⇝ *integer*
  Nummer des Kanals oder die Nummern der RGB-Kanäle.
  Defaultwert : 1
  Werteliste : Channel ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Beispiel

```
   /* Tranformation from rgb to gray  */
read_image(Image,'patras')
disp_color(Image,WindowHandle)
rgb1_to_gray(Image,GrayImage)
disp_image(GrayImage,WindowHandle).
```

Ergebnis

Falls die übergebenen Bilder definierte Werte enthalten und ein gültiger Ausgabemodus eingestellt ist, liefert disp_channel den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

disp_channel ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_rgb, set_lut, set_hsi

Alternativen

disp_image, disp_color

Siehe auch

open_window, open_textwindow, reset_obj_db, set_lut, draw_lut, dump_window

Modul

System

disp_circle (: : WindowHandle, Row, Column, Radius :)

Ausgabe von Kreisen in ein Fenster.

disp_circle trägt einen oder mehrere Kreise, die durch den Schwerpunkt (Row, Column) und den Radius beschrieben werden, in das Ausgabefenster ein. Liegen die angegebenen Koordinaten ganz oder teilweise außerhalb der Fensterkoordinaten, dann wird der Kreis entsprechend beschnitten.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (disp_region) mit den Prozeduren wie set_color, set_gray, set_draw, etc. bestimmt werden.

Es können mehrere Kreise bei einem Aufruf gezeichnet werden, indem Tupel von Koordinaten übergeben werden. Für die Behandlung der Farben bei mehr als einem Kreis, siehe set_color.

Achtung

Der Schwerpunkt des Kreises muß innerhalb des Fensters liegen.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Row** (input_control) . . . . . circle.center.y(-array) ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : `Row` ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ `Row` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) . . . . . circle.center.x(-array) ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : `Column` ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ `Column` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Radius** (input_control) . . . . . circle.radius(-array) ⇝ *real* / integer
  Radius des Kreises.
  Defaultwert : 64
  Wertevorschläge : `Radius` ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ `Radius` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `Radius` > `0.0`

Beispiel

```
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).
```

Ergebnis

`disp_circle` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`disp_circle` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `set_rgb`, `set_hsi`

Alternativen

`disp_ellipse`, `disp_region`, `gen_circle`, `gen_ellipse`

Siehe auch

`open_window`, `open_textwindow`, `set_color`, `set_draw`, `set_rgb`, `set_hsi`

Modul

System

`disp_color ( ColorImage : : WindowHandle : )`

Ausgabe eines RGB-Bildes.

disp_color trägt die drei Kanäle eines Farbbildes in das Ausgabefenster ein. Der erste Kanal ist der Rotkanal, der zweite der Grünkanal und der dritte der Blaukanal. disp_color kann mit disp_channel simuliert werden.

Achtung

Wegen der Beschränkung der Anzahl von Bildschirmfarben ist die Darstellung von Farbbildern i.a. nicht originalgetreu.

Parameter

- ▷ **ColorImage** (input_object) ..... image ⇝ *Hobject*
  Auszugebendes Farbbild.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.

Beispiel

```
/* disp_color(ColorImage) is identical to: */
Herror my_disp_color(Hobject ColorImage, Htuple *WindowHandle) {
  Htuple  Tupel;
  create_tuple(&Tupel,3);
  set_i(Tupel,1,0);
  set_i(Tupel,2,1);
  set_i(Tupel,3,2);
  T_disp_channel(ColorImage,*WindowHandle,Tupel);
  destroy_tuple(Tupel);
}
```

Ergebnis

Falls das übergebene Bild definierte Werte enthält und ein gültiger Ausgabemodus eingestellt ist, dann liefert disp_color den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

disp_color ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_rgb, set_lut, set_hsi

Alternativen

disp_channel, disp_obj

Siehe auch

disp_image, open_window, open_textwindow, reset_obj_db, set_lut, draw_lut, dump_window

Modul

System

disp_distribution (: : WindowHandle, Distribution, Row, Column, Scale :)

Ausgabe einer Rauschverteilung.

disp_distribution gibt eine Rauschverteilung im Fenster aus. Die Parameter entsprechen denen von set_paint(WindowHandle,'histogram') oder gen_region_histo. Rauschverteilung können mit Operationen wie gauss_distribution oder noise_distribution_mean erzeugt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Distribution** (input_control) ..... real-array ⇝ *real*
  Grauwertverteilung (513 Werte).

▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
Zeilenindex des Mittelpunktes.
Defaultwert : 256
Wertevorschläge : Row ∈ {0, 64, 128, 256}
Typischer Wertebereich : 0 ≤ Row ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
Spaltenindex des Mittelpunktes.
Defaultwert : 256
Wertevorschläge : Column ∈ {0, 64, 128, 256}
Typischer Wertebereich : 0 ≤ Column ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Scale** (input_control) . . . . . integer ⇝ *integer*
Größe der Darstellung.
Defaultwert : 1
Wertevorschläge : Scale ∈ {1, 2, 3, 4, 5, 6}

Beispiel

```
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)
noise_distribution_mean(Region,Image,21,Distribution)
disp_distribution (WindowHandle,Distribution,100,100,3).
```

Parallelisierungsinformation

disp_distribution ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi, noise_distribution_mean, gauss_distribution

Siehe auch

gen_region_histo, set_paint, gauss_distribution, noise_distribution_mean

Modul

System

disp_ellipse (: : WindowHandle, CenterRow, CenterCol, Phi, Radius1, Radius2 :)

Ausgabe von Ellipsen.

disp_ellipse trägt eine oder mehrere Ellipsen mit dem Schwerpunkt (CenterRow,CenterCol), der Orientierung Phi (in Bogenmaß) und den Halbradien Radius1 und Radius2 in das Ausgabefenster ein.

Es kann mehr als eine Ellipse ausgegeben werden, indem Tupel von Punkten, Winkeln und Halbradien übergeben werden. Die Parameter für die Ausgabe können wie bei der Darstellung von Regionen (disp_region) mit den Prozeduren set_color, set_draw, set_line_width, etc. bestimmt werden. Für die Behandlung der Farben bei mehr als einer Ellipse, siehe set_color.

Achtung

Der Schwerpunkt muß innerhalb der Fensterkoordinaten liegen.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **CenterRow** (input_control) . . . . . ellipse.center.y(-array) ⇝ *integer*
  Zeilenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : CenterRow ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ CenterRow ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **CenterCol** (input_control) . . . . . ellipse.center.x(-array) ⇝ *integer*
  Spaltenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : CenterCol ∈ {0, 64, 128, 256}
  Typischer Wertebereich : 0 ≤ CenterCol ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Phi** (input_control) . . . . . ellipse.angle.rad(-array) ⇝ *real* / integer
  Orientierung der Ellipse in Bogenmaß.
  Defaultwert : 0.0
  Wertevorschläge : Phi ∈ {0.0, 0.785398, 1.570796, 3.1415926, 6.283185}
  Typischer Wertebereich : 0.0 ≤ Phi ≤ 6.283185 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
- ▷ **Radius1** (input_control) . . . . . ellipse.radius1(-array) ⇝ *real* / integer
  Lange Halbachse der Ellipse.
  Defaultwert : 24.0
  Wertevorschläge : Radius1 ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich : 0.0 ≤ Radius1 ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **Radius2** (input_control) . . . . . ellipse.radius2(-array) ⇝ *real* / integer
  Kurze Halbachse der Ellipse.
  Defaultwert : 14.0
  Wertevorschläge : Radius2 ∈ {0.0, 64.0, 128.0, 256.0}
  Typischer Wertebereich : 0.0 ≤ Radius2 ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disp_ellipse den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

disp_ellipse ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi, elliptic_axis, area_center

Alternativen

disp_circle, disp_region, gen_ellipse, gen_circle

Siehe auch

open_window, open_textwindow, set_color, set_rgb, set_hsi, set_draw, set_line_width

Modul

System

disp_image (Image : : WindowHandle :)

Ausgabe von Grauwertdaten.

disp_image gibt die Grauwerte von einem Bild auf dem Ausgabefenster aus. Es werden dabei diejenigen Grauwerte verwendet, die innerhalb des Definitionsbereiches (set_comprise(::WindowHandle,'object':)) oder des gesamten Bildes (set_comprise (::WindowHandle,'image':)) liegen. Voreingestellt ist, daß nur die definierten Grauwerte verwendet werden.

Bei der Ausgabe von Graubildern wird die Anzahl der Graustufen i.a. reduziert. Dies ist durch die Tastsache bedingt, daß noch Farben für die Darstellung von Graphiken (z.B. set_color) und den Windowmanager reserviert werden müssen. Zudem stehen abhängig von den auf der Ausgabemaschine verfügbaren Bitplanes häufig weniger als 256 Farben (acht Bitplanes) zur Verfügung. Die Anzahl der tatsächlich für die Ausgabe von Graustufen reservierten „Farben“ kann mittels get_system abgefragt und vor dem Öffnen des ersten Fensters mittels set_system auch verändert werden. Für 8 Bitplanes sind beispielsweise 200 reale Graustufen voreingestellt.

Die Reduktion der Anzahl der Graustufen ist unproblematisch, solange nur Grauwertinformation ausgegeben wird, da der Mensch 256 verschiedenen Graustufen ohnehin nicht unterscheiden kann. Sind jedoch bestimmte Grauwerte für die Speicherung von Regioneninformation verwendet worden (was nicht der Vorgehensweise von HALCON entspricht), so kann dies bei der Darstellung zu Verwechslungen führen, da somit unterschiedliche Zahlenwerte mit dem gleichen Grauwert auf dem Bildschirm abgebildet werden. Sollten derartige Bilder vorliegen, empfiehlt es sich, die Prozedur label_to_region zu verwenden, um das Bild mit den Markierungen in HALCON-Objekte umzuwandeln.

Werden Bilder vom Type 'int2', 'int4', 'real' oder 'complex' ausgegeben, so wird zunächst der kleinste und der größte Grauwert bestimmt. Danach werden die Bildpunkte auf die zur Verfügung stehende Anzahl von Graustufen (abhängig von der Ausgabemaschine, ca. 200) skaliert. Hierbei kann es vorkommen, daß einige Pixelwerte extrem andere Werte als alle übrigen haben. Dies führt dazu, daß entweder ein ganz weißes oder schwarzes Bild auf dem Fenster erscheint. Um festzustellen, ob es sich um ein binäres Bild handelt, kann man z.B. min_max_gray verwenden. Gegebenenfalls kann das Bild vor der Ausgabe mit scale_image und convert_image_type in der gewünschten Weise angepaßt werden.

Achtung

Wurde bei set_paint ein falscher Ausgabemodus gesetzt, dann erfolgt die Fehlermeldung erst bei disp_image.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Auszugebendes Grauwertbild.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.

Beispiel

```
/* Output of a gray image: */
read_image(Image1,'affe')
disp_image(Image1,WindowHandle).
```

Ergebnis

Falls das übergebene Bild definierte Werte enthält und ein gültiger Ausgabemodus eingestellt ist, dann liefert disp_image den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

disp_image ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `scale_image`, `convert_image_type`, `min_max_gray`

Alternativen

`disp_obj`, `disp_color`

Siehe auch

`open_window`, `open_textwindow`, `reset_obj_db`, `set_comprise`, `set_paint`, `set_lut`, `draw_lut`, `paint_gray`, `scale_image`, `convert_image_type`, `dump_window`

Modul

System

`disp_line` (: : WindowHandle, Row1, Column1, Row2, Column2 :)

Zeichnen von Linien in ein Fenster.

`disp_line` trägt eine oder mehrere Linien mit Anfangskoordinaten (`Row1`,`Column1`) und Endkoordinaten (`Row2`,`Column2`) in das Ausgabefenster ein. Bei der Darstellung von mehr als einer Linie müssen die Koordinaten dabei jeweils in Form eines Tupels übergeben werden. Für die Behandlung der Farben bei mehr als einer Linie, siehe `set_color`.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (`disp_region`) mit den Prozeduren `set_color`, `set_gray`, `set_draw`, `set_line_width`, etc. bestimmt werden.

Achtung

Die Anfangs- und Endpunkte der Linien müssen innerhalb des Fensters liegen.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`Row1`** (input_control) ..... line.begin.y(-array) ⇝ *real*
  Zeilenindex des Anfangspunktes.
  Defaultwert : 32
  Wertevorschläge : `Row1` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Row1` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Column1`** (input_control) ..... line.begin.x(-array) ⇝ *real*
  Spaltenindex des Anfangspunktes.
  Defaultwert : 32
  Wertevorschläge : `Column1` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Column1` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Row2`** (input_control) ..... line.end.y(-array) ⇝ *real*
  Zeilenindex des Endpunktes.
  Defaultwert : 64
  Wertevorschläge : `Row2` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Row2` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Column2`** (input_control) ..... line.end.x(-array) ⇝ *real*
  Spaltenindex des Endpunktes.
  Defaultwert : 64
  Wertevorschläge : `Column2` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Column2` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

Beispiel

```
/* 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).
```

Ergebnis

`disp_line` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`disp_line` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Alternativen

`disp_arrow`, `disp_rectangle1`, `disp_rectangle2`, `disp_region`, `gen_region_polygon`, `gen_region_points`

Siehe auch

`open_window`, `open_textwindow`, `set_color`, `set_rgb`, `set_hsi`, `set_insert`, `set_line_width`

Modul

System

`disp_obj ( Object : : WindowHandle : )`

Ausgabe von Bildobjekten (Bilder, Regionen, XLD).

`disp_obj` gibt die übergebenen Objekte entsprechend ihrer Art (Bilder, Regionen, XLD) aus. `disp_obj` entspricht `disp_image` bei einem einkanaligen Bild, `disp_color` bei einem dreikanaligen Bild, `disp_region` bei Regionen und `disp_xld` bei XLD.

Parameter

- ▷ **`Object`** (input_object) ..... object(-array) ⇝ *Hobject*
  Auszugebendes Bildobject.
- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.

Beispiel

```
/* Output of a gray image: */
read_image(Image1,'affe')
disp_obj(Image1,WindowHandle)
threshold(Image,Region,0,128)
disp_obj(Region,WindowHandle)
```

Ergebnis

Falls das übergebene Objekt definierte Werte enthält und ein gültiger Ausgabemodus eingestellt ist, dann liefert `disp_obj` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`disp_obj` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `scale_image`, `convert_image_type`, `min_max_gray`

Alternativen

`disp_color`, `disp_image`, `disp_xld`, `disp_region`

Siehe auch

`open_window`, `open_textwindow`, `reset_obj_db`, `set_comprise`, `set_paint`, `set_lut`, `draw_lut`, `paint_gray`, `scale_image`, `convert_image_type`, `dump_window`

Modul

System

`disp_polygon` (: : WindowHandle, Row, Column :)

Zeichnen eines Polygonzuges.

`disp_polygon` trägt ein Polygon mit den Zeilenkoordinaten `Row` und Spaltenkoordinaten `Column` in das Ausgabefenster ein. Die Parameter `Row` und `Column` werden jeweils in Form von Tupeln übergeben. Zwischen den angegebenen Punkten werden Geraden gezeichnet. Der Anfangs- und Endpunkt des Polygons wird nicht verbunden.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (`disp_region`) mit Prozeduren wie `set_color`, `set_rgb`, etc. bestimmt werden.

Achtung

Die angegebenen Koordinaten müssen innerhalb des Fensters liegen.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`Row`** (input_control) ..... polygon.y-array ⇝ *integer* / real
  Zeilenindex
  Defaultwert : '[16,80,80]'
  Wertevorschläge : `Row` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Row` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Column`** (input_control) ..... polygon.x-array ⇝ *integer* / real
  Spaltenindex
  Defaultwert : '[48,16,80]'
  Wertevorschläge : `Column` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Column` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

Beispiel

```
/* display a rectangle */

disp_rectangle1_margin1(long WindowHandle,
  long Row1, long Column1,
  long Row2, long Column2)
{
  Htuple Row, Col;
  create_tuple(&Row,4) ;
  create_tuple(&Col,4) ;

  set_i(Row,Row1,0) ;
  set_i(Col,Column1,0) ;
```

```
  set_i(Row,Row1,1) ;
  set_i(Col,Column2,1) ;

  set_i(Row,Row2,2) ;
  set_i(Col,Column2,2) ;

  set_i(Row,Row2,3) ;
  set_i(Col,Column1,3) ;

  set_i(Row,Row1,4) ;
  set_i(Col,Column1,4) ;

  T_disp_polygon(WindowHandle,Row,Col) ;

}
```

Ergebnis

`disp_polygon` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`disp_polygon` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Alternativen

`disp_line`, `gen_region_polygon`, `disp_region`

Siehe auch

`open_window`, `open_textwindow`, `set_color`, `set_rgb`, `set_hsi`, `set_insert`, `set_line_width`

Modul

System

`disp_rectangle1` (: : WindowHandle, Row1, Column1, Row2, Column2 :)

Ausgabe von achsenparallelen Rechtecken.

`disp_rectangle1` trägt ein oder mehrere Rechtecke, die jeweils durch die linke obere Ecke (`Row1`,`Column1`) und die rechte untere Ecke (`Row2`,`Column2`) beschrieben werden, in das Ausgabefenster ein. Liegen die angegebenen Koordinaten ganz oder teilweise außerhalb der Fensterkoordinaten, dann wird das Rechteck entsprechend beschnitten. Sollen mehrere Rechtecke dargestellt werden, sind die Koordinaten als Tupel zu übergeben.

Die Parameter für die Ausgabe können wie bei der Darstellung einer Region (`disp_region`) mit den Prozeduren `set_color`, `set_gray`, `set_draw`, `set_line_width`, etc. bestimmt werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`Row1`** (input_control) ..... rectangle.origin.y(-array) ⇝ *real* / integer
  Zeilenindex der linken oberen Ecke.
  Defaultwert : 16
  Wertevorschläge : `Row1` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Row1` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *real* / integer
Spaltenindex der linken oberen Ecke.
Defaultwert : 16
Wertevorschläge : Column1 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich : 0 ≤ Column1 ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *real* / integer
Zeilenindex der rechten unteren Ecke.
Defaultwert : 48
Wertevorschläge : Row2 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich : 0 ≤ Row2 ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Row2 ≥ Row1

▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *real* / integer
Spaltenindex der rechten unteren Ecke.
Defaultwert : 80
Wertevorschläge : Column2 ∈ {0, 64, 128, 256, 511}
Typischer Wertebereich : 0 ≤ Column2 ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Column2 ≥ Column1

Beispiel

```
set_color(WindowHandle,'green')
draw_region(MyRegion,WindowHandle)
smallest_rectangle1(MyRegion,R1,C1,R2,C2)
disp_rectangle1(WindowHandle,R1,C1,R2,C2).
```

Ergebnis

disp_rectangle1 liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

disp_rectangle1 ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_rgb, set_lut, set_hsi, set_draw, set_color, set_colored, set_line_width

Alternativen

disp_rectangle2, gen_rectangle1, disp_region, disp_line, set_shape

Siehe auch

open_window, open_textwindow, set_color, set_draw, set_line_width

Modul

System

disp_rectangle2 (: : WindowHandle, CenterRow, CenterCol, Phi, Length1, Length2 :)

Ausgabe von beliebig orientierten Rechtecken.

disp_rectangle2 trägt ein oder mehrere Rechtecke mit dem Schwerpunkt (CenterRow,CenterCol), der Orientierung Phi (in Bogenmaß) und den halben Kantenlängen Length1 und Length2 in das Ausgabefenster ein. Es kann mehr als ein Rechteckt gezeichnet werden, indem jeweils Tupel von Koordinaten übergeben werden. Für die Behandlung der Farben bei mehr als einem Rechteck, siehe set_color.

Die Parameter für die Ausgabe können wie bei der Darstellung von Regionen (disp_region) mit den Prozeduren set_color, set_draw, set_line_width, etc. bestimmt werden.

Achtung

Der Schwerpunkt muß innerhalb der Fensterkoordinaten liegen.

Parameter

- ▷ **`WindowHandle`** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`CenterRow`** (input_control) . . . . . rectangle2.center.y(-array) ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert : 48
  Wertevorschläge : `CenterRow` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `CenterRow` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`CenterCol`** (input_control) . . . . . rectangle2.center.x(-array) ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert : 64
  Wertevorschläge : `CenterCol` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `CenterCol` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Phi`** (input_control) . . . . . rectangle2.angle.rad(-array) ⇝ *real* / integer
  Orientierung des Rechtecks in Bogenmaß.
  Defaultwert : 0.0
  Wertevorschläge : `Phi` ∈ {0.0, 0.785398, 1.570796, 3.1415926, 6.283185}
  Typischer Wertebereich : 0.0 ≤ `Phi` ≤ 6.283185 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
- ▷ **`Length1`** (input_control) . . . . . rectangle2.hwidth(-array) ⇝ *real* / integer
  Hälfte der größeren Seitenlänge.
  Defaultwert : 48
  Wertevorschläge : `Length1` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Length1` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Length2`** (input_control) . . . . . rectangle2.hheight(-array) ⇝ *real* / integer
  Hälfte der kürzeren Seitenlänge.
  Defaultwert : 32
  Wertevorschläge : `Length2` ∈ {0, 64, 128, 256, 511}
  Typischer Wertebereich : 0 ≤ `Length2` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : `Length2` < `Length1`

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `disp_rectangle2` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`disp_rectangle2` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_rgb`, `set_lut`, `set_hsi`, `set_draw`, `set_color`, `set_colored`, `set_line_width`

Alternativen

disp_region, gen_rectangle2, disp_rectangle1, set_shape

Siehe auch

open_window, open_textwindow, disp_region, set_color, set_draw, set_line_width

Modul

System

disp_region (DispRegions : : WindowHandle :)

Ausgabe von Regionen in ein Fenster.

disp_region trägt die Regionen aus DispRegions in das Ausgabefenster ein. Die Parameter für die Ausgabe können mit den Prozeduren set_color, set_gray, set_draw, set_line_width, etc. bestimmt werden.

Die Farbe(n) für die Darstellung der Regionen wird mit set_color, set_rgb, set_gray oder set_colored festgelegt. Wird mehr als eine Region ausgegeben und ist mehr als eine Farbe gesetzt (siehe set_color, set_colored), dann werden die Regionen zyklisch mit den angegebenen Farben ausgegeben.

Die Form der Region kann für die Ausgabe mit set_paint verändert werden (z.B. umschließender Kreis, konvex Hülle, etc.). Mit set_draw wird festgelegt, ob die Region ausgefüllt oder nur der Rand gezeichnet werden soll. Falls nur der Rand gezeichnet wird, kann mit set_line_width dessen Dicke in Pixeln und mit set_line_style das Darstellungsmuster angegeben werden.

Parameter

- **DispRegions** (input_object) ..... region(-array) ⇝ *Hobject*
  Auszugebende Regionen.
- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.

Beispiel

```
/* 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).
```

Ergebnis

disp_region liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

disp_region ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_rgb, set_lut, set_hsi, set_shape, set_line_style, set_insert, set_fix, set_draw, set_color, set_colored, set_line_width

Alternativen

disp_obj, disp_arrow, disp_line, disp_circle, disp_rectangle1, disp_rectangle2, disp_ellipse

Siehe auch

open_window, open_textwindow, set_color, set_colored, set_draw, set_shape, set_paint, set_gray, set_rgb, set_hsi, set_pixel, set_line_width, set_line_style, set_insert, set_fix, paint_region, dump_window

Modul

System

disp_xld (XLDObject : : WindowHandle :)

Ausgabe der Daten eines XLD-Objekts.

Mit disp_xld können XLD-Objekte eines beliebigen Typs ausgegeben werden.

Parameter

- ▷ **XLDObject** (input_object) ..... xld-array ⇝ *Hobject*
  Auszugebendes XLD-Objekt.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster Id.

Parallelisierungsinformation

disp_xld ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Siehe auch

disp_image, disp_region, disp_channel, disp_color, disp_line, disp_arc

Modul

Sub-pixel operators

4.2 Farbtabelle

disp_lut (: : WindowHandle, Row, Column, Scale :)

Graphische Darstellung der Farbtabelle.

disp_lut gibt die Farbtabelle („look-up-table“, kurz: „lut“) auf dem Fenster graphisch aus. Eine Farbtabelle ist die Umsetzung der Grauwerte eines Bildes in Farben, bzw. Grauwerten auf dem Bildschirm. Diese Umsetzung kann bei den meisten Ausgabegeräten mit set_lut verändert werden. disp_lut erzeugt eine graphische Darstellung dieser drei Umsetzungstabellen (für Rot, Grün und Blau). Die Ausgabe erfolgt im Fenster mit der logischen Fensternummer WindowHandle und gibt für dieses die Farbtabelle aus. Die Parameter Row und Column geben die Zeile bzw. Spalte des Mittelpunktes der Grafik (für die Positionierung) an. Der Skalierungsfaktor Scale legt die Größe fest. Dabei bedeutet 1, daß 256 Werte dargestellt werden; bei 2 werden 128 Werte dargestellt, bei 3 werden 64 Werte dargestellt, usw. Bei Farbtabellen für Schwarz/Weiß - Darstellungen erfolgt die Ausgabe der Grafik in der aktuellen Farbe (siehe: set_color, set_rgb, etc.). Bei Falschfarbtabellen werden die Farben rot, grün und blau für die Darstellung der jeweiligen Farbkomponenten verwendet.

Achtung

disp_lut ist nur für Rechner geeignet, die mit Farbtabellen arbeiten.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Schwerpunktes der Graphik.
  Defaultwert : 128
  Typischer Wertebereich : $0 \leq$ Row ≤ 511
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Schwerpunktes der Graphik.
  Defaultwert : 128
  Typischer Wertebereich : $0 \leq$ Column ≤ 511

▷ **Scale** (input_control) ..... integer ⇝ *integer*
Verkleinerungsfaktor.
Defaultwert : 1
Werteliste : Scale ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Typischer Wertebereich : 0 ≤ Scale ≤ 20

Beispiel

```
set_lut(WindowHandle,'color')
disp_lut(WindowHandle,256,256,1)
get_mbutton(WindowHandle,_,_,_)
set_lut(WindowHandle,'sqrt')
disp_lut(WindowHandle,128,128,2).
```

Ergebnis

Ist Ausgabefenster gültig und verfügt der Rechner über Farbtabellen, dann liefert disp_lut den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

disp_lut ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

set_lut

Siehe auch

open_window, open_textwindow, draw_lut, set_lut, set_fix, set_pixel, write_lut, get_lut, set_color

Modul

System

draw_lut (: : WindowHandle :)

Interaktive Manipulation der aktuellen Farbtabelle.

draw_lut dient zur interaktiven Manipulation der aktuellen Farbtabelle des Gerätes, auf dem das gerade aktive Ausgabefenster geöffnet wurde.

In einem 2D-Diagramm mit den Graustufen als x-Achse können mit gedrückter linker Maustaste die zugehörigen Rot-, Grün- und Blauintensitäten (immer „von links nach rechts“) verändert werden. Die Auswahl des zu bearbeitenden Farbkanals erfolgt zuvor ebenfalls mit der linken Maustaste. Alternativ dazu lassen sich den Grauwerten der x-Achse auch reine Grauwertintensitäten (grauer „Farbkanal“) zuordnen. Die rechte Maustaste beendet die Prozedur. Die so modifizierte Farbtabelle läßt sich mittels write_lut speichern und später mit set_lut wieder laden. Ein get_lut im Anschluß an draw_lut liefert unmittelbar die RGB-Tupel der Farbtabelle. Diese können ebenfalls als Eingabe für set_lut verwendet werden.

Achtung

draw_lut kann nur bei Rechnern verwendet werden, die Farbtabellen für die Ausgabe verwenden, die dynamisch geändert werden können.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fensteridentifikator.

Beispiel

```
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').
```

Ergebnis

draw_lut liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_lut ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_lut_style, set_lut, write_lut, disp_lut

Alternativen

set_fix, set_rgb

Siehe auch

write_lut, set_lut, get_lut, disp_lut

Modul

System

get_fixed_lut (: : WindowHandle : Mode)

Abfrage der Fixierung der Farbtabelle bei Echtfarbenbildern

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Modus für Fixierung.
  Defaultwert : ’true’
  Werteliste : Mode ∈ {’true’, ’false’}

Parallelisierungsinformation

get_fixed_lut ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_fixed_lut

Modul

System

get_lut (: : WindowHandle : LookUpTable)

Abfragen der aktuellen Farbtabelle.

get_lut gibt den Namen oder die Werte der Farbtabelle des gültigen Ausgabefensters aus, die von der Prozedur disp_image (indirekt auch von disp_region etc.) verwendet wird. Gesetzt wird die Farbtabelle mit der Prozedur set_lut. Handelt es sich bei der aktuellen Tabelle um eine Systemtabelle (die nicht durch set_fix modifiziert wurde), so wird der Name der Tabelle angegeben. Handelt es sich um eine modifizierte Tabelle, um eine von Datei eingelesene oder um eine Tabelle für Pseudo-Echtfarbenausgabe, so wird die Tabelle in Form von RGB-Werten ausgegeben.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **LookUpTable** (output_control) ..... string-array ⇝ *string* / integer
  Name der Farbtabelle oder Tupel der RGB-Werte.

Ergebnis

get_lut liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_lut` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`draw_lut`, `set_lut`

Alternativen

`set_fix`, `get_pixel`

Siehe auch

`set_lut`, `draw_lut`

Modul

System

get_lut_style (: : WindowHandle : Hue, Saturation, Intensity)

Modifikationsparameter der Farbtabelle abfragen.

`get_lut_style` liefert die Werte, die mit `set_lut_style` gesetzt wurden. Voreingestellt sind:

Hue: 0.0
Saturation 1.0
Intensity 1.0

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **Hue** (output_control) ..... real ⇝ *real*
  Modifikation des Farbwertes.
- ▷ **Saturation** (output_control) ..... real ⇝ *real*
  Modifikation der Sättigung.
- ▷ **Intensity** (output_control) ..... real ⇝ *real*
  Modifikation der Intensität.

Ergebnis

`get_lut_style` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_lut_style` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`set_lut_style`, `set_lut`

Siehe auch

`set_lut_style`

Modul

System

query_lut (: : WindowHandle : LookUpTable)

Abfragen aller verfügbaren Farbtabellen.

`query_lut` gibt die Namen aller Farbtabellen des aktuellen Gerätes an. Diese Farbtabellen können mit `set_lut` gesetzt werden. Immer vorhanden ist die Farbtabelle ’default’.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fensteridentifikator.
- ▷ **LookUpTable** (output_control) . . . . . string-array ⇝ *string*
  Namen der Farbtabellen.

Ergebnis

query_lut liefert den Wert 2 (H_MSG_TRUE), falls ein gültiges Fenster geöffnet ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

query_lut ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_lut_style, set_lut, write_lut, disp_lut

Siehe auch

set_lut

Modul

System

set_fixed_lut (: : WindowHandle, Mode :)

Festhalten der Farbtabelle bei Echtfarbenbilder.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fensteridentifikator.
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Modus für Fixierung.
  Defaultwert : ’true’
  Werteliste : Mode ∈ {’true’, ’false’}

Parallelisierungsinformation

set_fixed_lut ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

get_fixed_lut

Modul

System

set_lut (: : WindowHandle, LookUpTable :)

Setzen einer Farbtabelle.

set_lut setzt die Farbtabelle des Gerätes (Bildschirm), auf dem das Ausgabefenster dargestellt wird. Eine Farbtabelle ist die Umsetzung der „Grauwerte“ der Bildmatrix in einen Grauton oder eine Farbe auf dem Bildschirm. Für jeden Wert (0..255) wird festgelegt, in welcher Kombination aus rot grün und blau er dargestellt werden soll. Hierbei werden jedem Grauwert drei Intensitäten für die Grundfarben zugeordnet; deshalb spricht man von einer Farbtabelle. Die Umsetzung erfolgt in Echtzeit und bei jedem Bildschirmaufbau (typisch 60 bis 70 mal pro Sekunde) neu. Dies ermöglicht es, die Farbtabelle zu wechseln, um ein anderes Erscheinungsbild zu erhalten. Es ist zu beachten, daß die Farbtabelle nicht bei jedem Rechner gewechselt werden kann (z.B.: Schwarz/Weiß bzw. Echtfarbe).

Die Farbtabellen in HALCON sind (bei einem Rechner mit 256 Farben) in drei Bereiche aufgeteilt:

S: Systembereich, bzw. Benutzerbereich

G: Grafikfarben

B: Bilddaten

Die Farben in S stammen von den Anwendungen, die vor dem Start von HALCON aktiv waren, und weiterhin erhalten bleiben sollen. Die Grafikfarben G werden für Prozeduren wie `disp_region`, `disp_circle`, etc. verwendet und sind in allen Farbtabellen gleich besetzt, d.h. die Ausgaben, die in einer Grafikfarbe erfolgen bleiben bei allen Farbtabellen erhalten. Grafikfarben werden mit `set_color` und `set_rgb` gesetzt. Die Grautöne bzw. Farben der Bilddaten B werden von der Prozedur `disp_image` verwendet. Diese können sich in Abhängigkeit von der aktuellen Farbtabelle ändern. Eine Ausnahme vom oben Beschriebenen machen die Prozeduren `set_gray` (setzen einer Farbe aus dem Bereich B für Prozeduren wie `disp_region`) und die Prozedur `set_fix`, die eine Modifikation der Grafikfarben ermöglicht.

Bei den heute üblichen Bildschirmen kann nur eine Farbtabelle pro Bildschirm geladen werden. `set_lut` kann jedoch für jedes Fenster getrennt aufgerufen werden. Für dieses Problem gibt es folgende Lösung: Es wird immer diejenige Tabelle aktiviert, die dem gerade aktiven Fenster zugeordnet ist (dabei hat hier aktiv nichts mit `set_lut`, sondern vielmehr mit dem „Window-Manager“ zu tun).

Farb- und Grautabellen koennen können auch zusammen mit Echtfarben Graphikkarten verwendet werden. In diesem Fall wird die Tabelle durch Software simuliert, das heißt, daß das ein Bild bei der Ausgabe über die Tabelle bearbeitet wird.

WindowsNT spezifisch: Wird die Grafikkarte nicht im Echtfarbenmodus verwendet, so muß nach dem Einstellen der Farbtabelle das Bild neue ausgegeben werden.

Die Namen aller Farbtabellen können mit `query_lut` abgefragt werden. Die Farbtabellen unterscheiden sich in dem für die Grauwerte vorgesehenen Bereich. In diesem Bereich zeigen sie folgendes Verhalten:

Grauwerttabellen (1-7 Bildebenen)

’default’: Es werden die beiden Grundfarben (i.allg. schwarz und weiß) verwendet.

Farbtabelle (Echtfarbe, Statische Graustufen)

’default’: Von der Hardware vorgegebene Tabelle.

Grauwerttabellen (256 Farben)

’default’: Wie ’linear’.

’linear’: Lineares Ansteigen der Grauwerte von 0(schwarz) nach 255 (weiß).

’inverse’: Inverse Funktion zu ’linear’.

’sqr’: Ansteigen der Grauwerte als quadratische Funktion.

’inv_sqr’: Inverse Funktion zu ’sqr’.

’cube’: Ansteigen der Grauwerte als 3. Potenz-Funktion.

’inv_cube’: Inverse Funktion zu ’cube’.

’sqrt’: Ansteigen der Grauwerte als Quadratwurzelfunktion.

’inv_sqrt’: Inverse Funktion zu ’sqrt’.

’cubic_root’: Ansteigen der Grauwerte als 3. Wurzel-Funktion.

’inv_cubic_root’: Inverse Funktion zu ’cubic_root’.

Farbtabellen (256 Farben)

’color1’: Linearer Übergang von rot über grün nach blau.

’color2’: Fließender Übergang von gelb über rot, blau nach grün.

’color3’: Fließender Übergang von gelb über rot, blau, grün, rot nach blau.

’color4’: Fließender Übergang von gelb über rot nach blau.

’three’: Darstellung der drei Farben rot, grün, blau.

’six’: Darstellung der sechs Grundfarben gelb, rot, magenta, blau, cyan, grün.

’twelve’: Darstellung von 12 Farben.

’twenty_four’: Darstellung von 24 Farben.

'rainbow': Darstellung der Spektralfarben von rot über grün nach blau.

'temperature': Wärmetabelle von schwarz über rot und gelb nach weiß.

'change1': Farbänderung nach jedem Pixel der Tabelle, und zwar abwechselnd die sechs Grundfarben.

'change2': 5-fache Farbänderung von grün über rot nach blau.

'change3': 3-fache Farbänderung von grün über rot nach blau.

Weiterhin kann eine Farbtabelle aus einer Datei eingelesen werden. Diese Datei muß in jeder Zeile drei Zahlen von 0 bis 255 enthalten, wobei die erste Zahl dem Rotanteil, die zweite dem Grünanteil und die dritte dem Blauanteil für die Darstellung einer Farbe entspricht. Die Anzahl der Zeilen ist beliebig. Dabei wird die erste Zeile als erster Grauwert, die letzte als letzter Grauwert verwendet. Ist die Anzahl der Zeilen geringer als die Zahl der Grauwerte in der Farbtabelle, so werden die vorhandenen Werte gleichmäßig über das ganze Intervall verteilt und die fehlenden Werte durch Interpolation berechnet. Ist die Anzahl zu hoch, werden (gleichmäßig verteilt) Zeilen bei der Besetzung der Farbtabelle ignoriert. Diese Datei muß unter dem Namen „LookUpTable.lut" abgespeichert sein. Der Dateiname wird ohne Extension angegeben. Die Datei wird im aktuellen Directory und dann in einem ausgezeichneten Directory (siehe `set_system(::'lut_dir',<Pfad>:)`) gesucht. Der Aufruf von `set_lut` kann auch mit einem Tupel von RGB-Werten erfolgen. Diese werden dann direkt gesetzt. Es ist zu beachten, daß die Anzahl der Parameterwerte, der Anzahl der aktuell verwendeten Pixel in der Farbtabelle entsprechen müßen.

Achtung

`set_lut` kann nur bei Bildschirmen mit 256 Graustufen/Farben verwendet werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **`LookUpTable`** (input_control) ..... string(-array) ⇝ *string* / integer
  Name der Farbtabelle, Werte der Farbtabelle (RGB) oder der Name der Datei.
  Defaultwert : 'default'
  Wertevorschläge : `LookUpTable` ∈ {'default', 'linear', 'inverse', 'sqr', 'inv_sqr', 'cube', 'inv_cube', 'sqrt', 'inv_sqrt', 'cubic_root', 'inv_cubic_root', 'color1', 'color2', 'color3', 'color4', 'three', 'six', 'twelfe', 'twenty_four', 'rainbow', 'temperature', 'cyclic_gray', 'cyclic_temperature', 'hsi', 'change1', 'change2', 'change3'}

Beispiel

```
read_image(Image,'affe')
query_lut(WindowHandle,LUTs)
for(1,|LUTs|,i)
  set_lut(WindowHandle,LUTs[i])
  fwrite_string(['current table ',LUTs[i]])
  fnew_line()
  get_mbutton(WindowHandle,_,_,_)
loop().
```

Ergebnis

`set_lut` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist, die Hardware über eine Farbtabelle verfügt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_lut` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`query_lut`, `draw_lut`, `get_lut`

Mögliche Nachfolgerfunktionen

`write_lut`

Alternativen

`draw_lut`, `set_fix`, `set_pixel`

Siehe auch

`get_lut`, `query_lut`, `draw_lut`, `set_fix`, `set_color`, `set_rgb`, `set_hsi`, `write_lut`

Modul

System

set_lut_style (: : WindowHandle, Hue, Saturation, Intensity :)

Verändern der Farbtabelle.

set_lut_style verändert die Farbtabelle des Gerätes, auf dem das Ausgabefenster dargestellt wird. Dabei haben die drei Parameter folgende Bedeutung:

Hue: Rotation des Farbraumes
Hue = 1.0 entspricht einer einmaligen Rotation des Farbraumes.
Keine Veränderung: Hue = 0.0
Komplementärfarben: Hue = 0.5

Saturation: Veränderung der Sättigung
Keine Veränderung: Saturation = 1.0
Graubild: Saturation = 0.0

Intensity Veränderung der Helligkeit
Keine Veränderung: Intensity = 1.0
schwarzes Bild: Intensity = 0.0

Verändert werden nur der Teil der Farbtabelle, der für die Darstellung der Bildinformation verwendet wird. Die Modifikationsparameter bleiben erhalten bis set_lut_style erneut aufgerufen wird. Ein Aufruf von set_lut hat keinen Einfluß auf diese Parameter.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- **Hue** (input_control) ..... real ⇝ *real*
  Farbwertmodifikation.
  Defaultwert : 0.0
  Typischer Wertebereich : $0.0 \leq$ Hue ≤ 1.0
  Restriktion : $(0.0 \leq \text{Hue}) \wedge (\text{Hue} \leq 1.0)$
- **Saturation** (input_control) ..... real ⇝ *real*
  Modifikation der Sättigung.
  Defaultwert : 1.5
  Typischer Wertebereich : $0.0 \leq$ Saturation
  Restriktion : $0.0 \leq$ Saturation
- **Intensity** (input_control) ..... real ⇝ *real*
  Modifikation der Intensität.
  Defaultwert : 1.5
  Typischer Wertebereich : $0.0 \leq$ Intensity
  Restriktion : $0.0 \leq$ Intensity

Beispiel

```
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).
```

Ergebnis

set_lut_style liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_lut_style` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_lut_style`

Mögliche Nachfolgerfunktionen

`set_lut`

Alternativen

`set_lut`, `scale_image`

Siehe auch

`get_lut_style`

Modul

System

`write_lut ( : : WindowHandle, FileName : )`

Aktuelle Farbtabelle in eine Datei schreiben.

`write_lut` legt die aktuelle Farbtabelle (das ist der für Grauwerte relevante Ausschnitt der Tabelle) des gültigen Ausgabefenster in der Datei FileName ab. Sie kann dann später mittels `set_lut` wieder geladen werden.

Achtung

`write_lut` kann nur auf Bildschirmen mit 256 Farben verwendet werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **`FileName`** (input_control) ..... filename ⇝ *string*
  Name der Datei für die aktuelle Farbtabelle.
  Defaultwert : ’/tmp/lut’

Beispiel

```
read_image(Image,'affe')
disp_image(Image,WindowHandle)
draw_lut(WindowHandle)
write_lut(WindowHandle,'test_lut').
```

Ergebnis

Ist der Dateiname in Ordnung und hat das Fenster die richtigen Eigenschaften (256 Farben), dann liefert `write_lut` den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, wird 5 (H_MSG_FAIL) zurückgeliefert. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`write_lut` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`draw_lut`, `set_lut`

Siehe auch

`set_lut`, `draw_lut`, `set_pixel`, `get_pixel`

Modul

System

4.3 Fenster

clear_rectangle (: : WindowHandle, Row1, Column1, Row2, Column2 :)

Löschen eines Rechtecks auf dem Ausgabefenster.

clear_rectangle löscht alle Einträge in dem Rechteck, das durch die linke obere Ecke (Row1,Column1) und die rechte untere Ecke (Row2,Column2) aufgespannt wird. Löschen bedeutet, daß das angegebene Rechteck auf die Hintergrundfarbe gesetzt wird (siehe open_window, open_textwindow).

Falls mehr als ein Rechteck gelöscht werden soll, können auch mehrere Rechtecke übergeben werden, d.h. die Parameter Row1, Column1, Row2 und Column2 sind jeweils Tupel.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- **Row1** (input_control) ..... rectangle.origin.y(-array) ⇝ *integer*
  Zeilenindex der linken oberen Ecke.
  Defaultwert : 10
  Typischer Wertebereich : 0 ≤ Row1 ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- **Column1** (input_control) ..... rectangle.origin.x(-array) ⇝ *integer*
  Spaltenindex der linken oberen Ecke.
  Defaultwert : 10
  Typischer Wertebereich : 0 ≤ Column1 ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- **Row2** (input_control) ..... rectangle.corner.y(-array) ⇝ *integer*
  Zeilenindex der rechten unteren Ecke.
  Defaultwert : 118
  Typischer Wertebereich : 0 ≤ Row2 ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : Row2 > Row1
- **Column2** (input_control) ..... rectangle.corner.x(-array) ⇝ *integer*
  Spaltenindex der rechten unteren Ecke.
  Defaultwert : 118
  Typischer Wertebereich : 0 ≤ Column2 ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : Column2 ≥ Column1

Beispiel

```
/* "Interaktives" Loeschen eines Rechtecks im Ausgabefenster: */
draw_rectangle1(WindowHandle,L1,C1,L2,C2)
clear_rectangle(WindowHandle,L1,C1,L2,C2).
```

Ergebnis

Ist ein Ausgabefenster vorhanden und sind die angegebenen Parameter korrekt, dann liefert clear_rectangle den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

clear_rectangle ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_draw, set_color, set_colored, set_line_width, set_rgb, set_hsi, draw_rectangle1

Alternativen

clear_window, disp_rectangle1

Siehe auch

open_window, open_textwindow

Modul

System

clear_window (: : WindowHandle :)

Löschen eines Ausgabefensters.

clear_window löscht alle Einträge im Ausgabefenster. Das Fenster (Hintergrund und Rand) wird in seinen Anfangszustand versetzt. Parameter, die dem Fenster zugeordnet sind (z.B. mit set_color, set_paint, etc.), bleiben unverändert.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fenster-Identifikator.

Beispiel

```
clear_window(WindowHandle).
```

Ergebnis

Ist das Ausgabefenster gültig, dann liefert clear_window den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

clear_window ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Alternativen

clear_rectangle, disp_rectangle1

Siehe auch

open_window, open_textwindow

Modul

System

close_window (: : WindowHandle :)

Schließen eines Ausgabefensters.

close_window schließt ein Fenster, das mit open_window oder mit open_textwindow geöffnet wurde. Das Ausgabegerät bzw. der Fensterbereich stehen danach wieder für neue Aufrufe von open_window und open_textwindow zur Verfügung.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fenster-Identifikator.

Ergebnis

Ist das Ausgabefenster gültig, dann liefert close_window den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_window ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`

Siehe auch

`open_window`, `open_textwindow`

Modul

System

`copy_rectangle` (: : `WindowHandleSource, WindowHandleDestination, Row1, Column1, Row2, Column2, DestRow, DestColumn` :)

Kopieren aller Pixel eines Rechtecks zwischen Ausgabefenstern.

`copy_rectangle` kopiert alle Pixel, die innerhalb des durch die Parameter `Row1`, `Column1`, `Row2` und `Column2` beschriebenen Rechtecks liegen, aus dem angegebenen Window mit der logischen Fensternummer `WindowHandleSource` in das angegebene Window mit der logischen Fensternummer `WindowHandleDestination`. Die Zielposition wird durch die linke obere Ecke des Rechtecks (`DestRow`, `DestColumn`) festgelegt.

Falls mehr als ein Rechteck bewegt werden soll, können diese gleichzeitig (in Form von Tupeln) übergeben werden.

`copy_rectangle` ist dafür gedacht, aufbereitete Graphiken aus einem „unsichtbaren“ Fenster in ein sichtbares Fenster zu kopieren. Hierzu wird ein Fenster mit der Option 'buffer' geöffnet. In dieses Fenster wird die Graphik ausgegeben und danach in ein sichtbares Fenster kopiert. Der Vorteil dieser Vorgehensweise besteht darin, daß `copy_rectangle` viel schneller als Ausgabeprozeduren wie z.B. `disp_channel` ist. Dies ist insbesondere bei Demo-Programmen von Vorteil. Es lassen sich hiermit auch kurze „clips“ realisieren: Hierzu wird für jedes Bild der Folge ein Fenster vom Typ 'buffer' angelegt und die Daten darauf ausgegeben. Ausgegeben wird die Bildfolge, indem alle Puffer nacheinander in ein sichtbares Fenster kopiert werden.

Achtung

Beide Fenster müssen auf demselben Rechner liegen.

Parameter

- ▷ **`WindowHandleSource`** (input_control) . . . . . window ⇝ *integer*
  Nummer des Quellfensters.
- ▷ **`WindowHandleDestination`** (input_control) . . . . . window ⇝ *integer*
  Nummer des Zielfensters.
- ▷ **`Row1`** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *integer*
  Zeilenindex der linken oberen Ecke im Quellfenster.
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `Row1` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Column1`** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *integer*
  Spaltenindex der linken oberen Ecke im Quellfenster.
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `Column1` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Row2`** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *integer*
  Zeilenindex der rechten unteren Ecke im Quellfenster.
  Defaultwert : 128
  Typischer Wertebereich : 0 ≤ `Row2` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `Row2` ≥ `Row1`

▷ **Column2** (input_control) ..... rectangle.corner.x(-array) ⇝ *integer*
Spaltenindex der rechten unteren Ecke im Quellfenster.
Defaultwert : 128
Typischer Wertebereich : 0 ≤ Column2 ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : Column2 ≥ Column1

▷ **DestRow** (input_control) ..... point.y(-array) ⇝ *integer*
Zeilenindex des linken oberen Ecks im Zielfenster.
Defaultwert : 0
Typischer Wertebereich : 0 ≤ DestRow ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **DestColumn** (input_control) ..... point.x(-array) ⇝ *integer*
Spaltenindex des linken oberen Ecks im Zielfenster.
Defaultwert : 0
Typischer Wertebereich : 0 ≤ DestColumn ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Beispiel

```
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).
```

Ergebnis

Ist das Ausgabefenster gültig und sind die angegebenen Parameter korrekt, dann liefert copy_rectangle den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

copy_rectangle ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Mögliche Nachfolgerfunktionen

close_window

Alternativen

move_rectangle, slide_image

Siehe auch

open_window, open_textwindow

Modul

System

dump_window (: : WindowHandle, Device, FileName :)

Ausgabe des Fensterinhaltes auf Datei.

dump_window gibt den Inhalt des Fensters auf eine Datei aus. Diese Datei kann dann mit geeigneten Druckern oder anderen Programmen weiterverarbeitet werden. Der Bildschirminhalt wird dabei jeweils für ein spezielles

Gerät (Device) aufbereitet, d.h. so formatiert, daß die Datei direkt ausgedruckt oder von einem Graphikprogramm weiterverarbeitet werden kann.

Für die Umsetzung der Grauwerte wird die aktuelle Farbtabelle des Fensters verwendet, d.h. die Werte von set_lut_style bleiben unberücksichtigt.

Die Grauwerte auf dem Bildschirm haben i.a. eine schlechtere Auflösung als die Originalgrauwerte der ausgegebenen Bilder. Dies führt dazu, daß für die Darstellung von Grauwerten auch nur eine verringerte Auflösung (voreingestellt sind z.B. 200 Graustufen bei 8 Bitplanes) zur Verfügung steht (siehe auch disp_channel). Die Anzahl der tatsächlich verfügbaren Graustufen kann mittels get_system abgefragt und vor dem Öffnen des ersten Fensters mittels set_system geändert werden.

Mögliche Werte für Device

'postscript': PostScript - Datei. Dateiendung: <FileName>.ps

'postscript',Breite,Höhe: PostScript - Datei mit Angabe der Ausgabegröße. $Width$ und $Height$ beziehen sich dabei auf die Größe des Ausdrucks. In diesem Fall wird ein Tupel mit den drei Werten als Device übergeben. Dateiendung: <FileName>.ps

'tiff': 1 Byte inkl. aktueller Farbtabelle oder 3 Byte pro Sample (je nach Graphikkarte), unkomprimiert; Dateiendung: <FileName>.tiff

'bmp': Windows-BMP-Format, RGB-Bild, 3 Byte pro Pixel; Dateiendung: <FileName>.bmp

'jpeg': JPEG-Format (verlustbehaftete Komprimierung); Zusammen mit dem Format kann ein Qualitätsmaß angeben werden, das die Komprimierungsrate und die Qualität des gespeicherten Bildes bestimmt. Große Werte (Maximum 100) erzeugen eine relativ große Datei mit hoher Qualität, bei kleine Werten nimmt die Qualität und die Dateigröße deutlich ab; Bsp.: 'jpeg 30'.

Achtung

Unter X-Windows muß das HALCON-Fenster vollständig auf dem Root-Fenster sichtbar sein, da sonst der Fensterinhalt aufgrund von Einschränkungen in X-Windows nicht ausgelesen werden kann. Sollen Ausgaben von größeren Darstellungen in eine Datei geschrieben werden, bietet sich der Windowtyp **'pixmap'** an.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Device** (input_control) ..... string(-array) ⇝ *string* / integer
  Name des Zielgerätes bzw. des Graphikformats.
  Defaultwert : 'postscript'
  Werteliste : Device ∈ {'postscript', 'tiff', 'bmp', 'jpeg', "jpeg 100", "jpeg 80", "jpeg 60", "jpeg 40", "jpeg 20"}
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Dateiname (ohne Extension).
  Defaultwert : 'halcon_dump'

Beispiel

```
/* 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').
```

Ergebnis

dump_window liefert den Wert 2 (H_MSG_TRUE), falls das passende Fenster gültig ist und die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

dump_window ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

open_window, set_draw, set_color, set_colored, set_line_width, open_textwindow, disp_region

Mögliche Nachfolgerfunktionen

`system_call`

Siehe auch

`open_window`, `open_textwindow`, `set_system`

Modul

System

`get_window_extents` (: : WindowHandle : Row, Column, Width, Height)

Information über Größe und Position eines Fensters.

`get_window_extents` gibt die Position der linken oberen Ecke, sowie die Breite und die Höhe des Ausgabefensters aus.

Achtung

Die Größe und die Position eines Fensters können durch den Windowmanager verändert werden, ohne daß dies vom Programm aus vorgenommen wurde. Deshalb können sich die Werte, die `get_window_extents` liefert, durch diesen Seiteneffekt ändern.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`Row`** (output_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilenindex der linken oberen Ecke des Fensters.
- ▷ **`Column`** (output_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltenindex der linken oberen Ecke des Fensters.
- ▷ **`Width`** (output_control) ..... rectangle.extent.x ⇝ *integer*
  Breite des Fensters.
- ▷ **`Height`** (output_control) ..... rectangle.extent.y ⇝ *integer*
  Höhe des Fensters.

Beispiel

```
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).
```

Ergebnis

`get_window_extents` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_window_extents` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `open_textwindow`

Siehe auch

`set_window_extents`, `open_window`, `open_textwindow`

Modul

System

get_window_pointer3 (: : WindowHandle : ImageRed, ImageGreen, ImageBlue, Width, Height)

Zugriff auf die Pixeldaten eines Fensters.

get_window_pointer3 ermöglicht (bei einigen Fenstersystemen) den direkten Zugriff auf die Bitmap. Rückgabewerte sind die drei Zeiger auf die Farbauszüge eines 24-Bit Fensters (ImageRed, ImageGreen,ImageBlue) sowie die Fenstergröße (Width,Height). In der Programmiersprache C ist der Typ der Bildpunkte unsigned char.

Achtung

get_window_pointer3 ist nur für den Fenster-Typ 'pixmap' realisiert.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- **ImageRed** (output_control) ..... integer ⇝ *integer*
  Zeiger auf den Rotkanal der Pixeldaten.
- **ImageGreen** (output_control) ..... integer ⇝ *integer*
  Zeiger auf den Grünkanal der Pixeldaten.
- **ImageBlue** (output_control) ..... integer ⇝ *integer*
  Zeiger auf den Blaukanal der Pixeldaten.
- **Width** (output_control) ..... extent.x ⇝ *integer*
  Länge einer Bildzeile.
- **Height** (output_control) ..... extent.y ⇝ *integer*
  Anzahl der Bildzeilen.

Ergebnis

get_window_pointer3 liefert den Wert 2 (H_MSG_TRUE), falls das Fenster vom Typ 'pixmap' existiert und gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_window_pointer3 ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Siehe auch

open_window, set_window_type

Modul

System

get_window_type (: : WindowHandle : WindowType)

Abfrage des Fenstertyps.

get_window_type bestimmt, von welchem Typ bzw. welcher Graphiksoftware das Ausgabegerät des Windows ist. Die verfügbaren Typen von Ausgabegeräten können mit der Prozedur query_window_type abgefragt werden. get_window_type kann dort sinnvoll eingesetzt werden, wo man maschinenunabhängige Programme entwickeln möchte. Mögliche Werte sind:

'X-Window' X-Window Version 11.

'pixmap' Fenster werden nicht angezeigt, sondern nur im Speicher verwaltet. Auf diese Weise können Halcon-Programme auch auf Rechnern ohne Graphikbildschirm portiert werden.

'PostScript' Die Ausgabe von Objekten erfolgt in eine PostScript-Datei.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **WindowType** (output_control) . . . . . string ⇝ *string*
  Art des Fensters

Beispiel

```
open_window(100,100,200,200,'root','visible','',WindowHandle)
get_window_type(WindowHandle,WindowType)
fwrite_string(['Window type: ',WindowType])
fnew_line().
```

Ergebnis

get_window_type liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_window_type ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Siehe auch

query_window_type, set_window_type, get_window_pointer3, open_window, open_textwindow

Modul

System

move_rectangle (: : WindowHandle, Row1, Column1, Row2, Column2, DestRow, DestColumn :)

Kopieren innerhalb eines Ausgabefensters.

move_rectangle kopiert alle Einträge in dem Rechteck (Row1,Column1), (Row2,Column2) des Ausgabefensters an eine neue Position innerhalb desselben Fensters. Diese Position wird durch die linke obere Ecke (DestRow,DestColumn) festgelegt. Die durch das Bewegen des Rechtecks „bloßgelegten“ Bereiche des Fensters werden auf die Farbe des Hintergrundes gesetzt.

Falls mehrere Rechtecke auf einmal bewegt werden sollen, können die Parameter jeweils in Form von Tupeln übergeben werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Row1** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *integer*
  Zeilenindex der linken oberen Ecke des Quell-Rechtecks.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ Row1 ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *integer*
  Spaltenindex der linken oberen Ecke des Quell-Rechtecks.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ Column1 ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *integer*
Zeilenindex der rechten unteren Ecke des Quell-Rechtecks.
Defaultwert : 64
Typischer Wertebereich : 0 ≤ Row2 ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *integer*
Spaltenindex der rechten unteren Ecke des Quell-Rechtecks.
Defaultwert : 64
Typischer Wertebereich : 0 ≤ Column2 ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **DestRow** (input_control) . . . . . point.y(-array) ⇝ *integer*
Zeilenindex der linken oberen Ecke der Zielposition.
Defaultwert : 64
Typischer Wertebereich : 0 ≤ DestRow ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **DestColumn** (input_control) . . . . . point.x(-array) ⇝ *integer*
Spaltenindex der linken oberen Ecke der Zielposition.
Defaultwert : 64
Typischer Wertebereich : 0 ≤ DestColumn ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Beispiel

```
/* "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).
```

Ergebnis

Ist das Ausgabefenster gültig und sind die angegebenen Parameter korrekt, dann liefert move_rectangle den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

move_rectangle ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Alternativen

copy_rectangle

Siehe auch

open_window, open_textwindow

Modul

System

new_extern_window (: : WINHWnd, WINHDC, Row, Column, Width, Height : WindowHandle)

Erzeugen eines virtuellen Grafikfensters unter Windows NT.

new_extern_window öffnet ein neues virtuelles Fenster. Virtuell bedeutet hier, daß eigenlich kein neues Fenster erstellt wird, sondern, daß das Fenster, dessen Windows-NT-Handle im Parameter WINHWnd übergeben wird, zur Ausgabe von Grauwertdaten, Regionen, Graphik sowie für Textausgabe verwendet werden. Einstellungen für die Ausgabe von Daten, wie die Farbe können entweder über die Befehle von Halcon oder durch den enstprechenden Windows-NT-Befehl erfolgen.

Beispiel: Setzen der Zeichenfarbe:

```
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);
```

Interaktive Funktionen, wie zum Beispiel `draw_region`, `draw_circle` oder `get_mbutton`, können in diesem Fenster nicht verwendet werden. Die folgenden Operatoren können verwendet werden:

- Grauwertausgabe: `set_paint`, `set_comprise`, (`set_lut` und `set_lut_style` nach der Ausgabe)
- Regionen: `set_color`, `set_rgb`, `set_hsi`, `set_gray`, `set_pixel`, `set_shape`, `set_line_width`, `set_insert`, `set_line_style`, `set_draw`
- Bildausschnitt: `set_part`
- Text: `set_font`

Die aktuell gesetzten Werte können mit Prozeduren wie `get_shape` abgefragt werden. Da einige Parameter durch die Hardware (Auflösung/Farben) bestimmt werden, kann man mit Prozeduren wie `query_color` die aktuell verfügbaren Resourcen abfragen.

Im Parameter `WINHWnd` wird der Fensterhandle des Windows-NT-Fensters übergeben, in das die Ausgaben von Halcon erfolgen sollen. Im Parameter `WINHDC` wird der Gerätekontext des Fensters mit dem Handle `WINHWnd` übergeben. Dieser Gerätekontext wird in den Ausgaberoutinen von Halcon verwendet.

Der Ursprung des Koordinatensystems des Fensters liegt in der linken oberen Ecke (Koordinaten: (0,0)). Nach unten nimmt der Zeilenindex zu (maximal: Fensterhöhe-1), nach rechts steigt der Spaltenindex (maximal: Fensterbreite-1) an.

Für die Parameter `Width` und `Height` kann der Wert **-1** als Größe verwendet werden. Dies bedeutet, daß der entsprechende Wert automatisch bestimmt werden soll. Dies ist insbesondere dann von Bedeutung, wenn das Pixelverhältnis nicht gleich 1.0 ist (siehe `set_system`). Wird einer der beiden Parameter als **-1** angegeben, so wird er durch die Größe bestimmt, die sich aus dem Pixelverhältnis ergibt. Werden beide Paramater mit **-1** festgelegt, so werden sie auf das aktuelle Bildformat gesetzt.

Die Position und Größe des Fensters kann sich im Laufe des Programms ändern. Dies kann zum einen mit `set_window_extents` erfolgen, aber auch durch externe Eingriffe (Windowmanager) hervorgerufen werden. Für den zweiten Fall ist die Prozedur `set_window_extents` vorgesehen.

Die Farbe für Graphik und Schrift, die für Ausgabeprozeduren wie `disp_region` oder `disp_circle` verwendet wird, wird mit `set_rgb`, `set_hsi`, `set_gray` oder `set_pixel` festgelegt. Mit `set_insert` wird eingestellt, wie die Graphik mit dem Inhalt des Bildwiederholspeichers verknüpft wird. Hierbei kann z.B. mit `set_insert(::'not':)` erreicht werden, daß nach zweimaligem Zeichnen eines Kreises an der gleichen Position dieser wieder beseitigt wird.

Der Inhalt von Fenstern wird nicht gesichert, falls sich Fenster überlappen. Dies muß durch den Programmcode, der das Windows-NT-Fenster verwaltet, sichergestellt werden.

Für die Ausgabe von Graphik (`disp_image`, `disp_region`, etc.) kann mit der Prozedur `set_part` das Fenster so eingestellt werden, daß es einen logischen Ausschnitt des Bildformates darstellt. Dies hat insbesondere zur Folge, daß von Bildern und Regionen lediglich dieser Ausschnitt (entsprechend vergrößert) ausgegeben wird. Bevor Sie Ihr Fenster schließen, müssen Sie das Halcon-Fenster schließen.

Die Schritte zur Verwendung von new_extern_window:

Erzeugung:
- Erzeugen eines Windows-Fensters.
- Aufruf von new_extern_window mit dem `WINHWnd` des vorher erzeugten Fensters.

Verwendung:
- Vor dem Zeichnen in das Fenster muß stets die Methode set_window_dc aufgerufen werden, um so sicherzustellen, daß die Halcon Zeichenroutinen den richtigen DC verwenden. Nach dem Zeichnen sollte erneut set_window_dc aufgerufen werden, diesmal jedoch mit der Adresse eines Longs, mit Inhalt 0. Hierdurch kann Halcon seine angelegten Zeichenobjekte ordnungsgemäß freigeben.

Löschen: • Zum Löschen muß close_window aufgerufen werden.

Achtung

Es ist zu beachten, daß die Parameter `Row`, `Column`, `Width` und `Height` durch das Ausgabegerät, d.h. die Größe des Windows-NT-Desktops, beschränkt sind.

Parameter

- ▷ **`WINHWnd`** (input_control) ..... integer ⇝ *integer*
  Windows-Fensterhandle eines vorher geöffneten Fensters.
  Restriktion : `WINHWnd` $\neq$ `0`
- ▷ **`WINHDC`** (input_control) ..... integer ⇝ *integer*
  Gerätekontext des WINHWnd
  Restriktion : `WINHDC` $\neq$ `0`
- ▷ **`Row`** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilenkoordinate der linken oberen Ecke.
  Defaultwert : 0
  Restriktion : `Row` $\geq$ `0`
- ▷ **`Column`** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltenkoordinate der linken oberen Ecke.
  Defaultwert : 0
  Restriktion : `Column` $\geq$ `0`
- ▷ **`Width`** (input_control) ..... rectangle.extent.x ⇝ *integer*
  Breite des Fensters.
  Defaultwert : 512
  Restriktion : (`Width` $>$ `0`) $\vee$ (`Width` $=$ `-1`)
- ▷ **`Height`** (input_control) ..... rectangle.extent.y ⇝ *integer*
  Höhe des Fensters.
  Defaultwert : 512
  Restriktion : (`Height` $>$ `0`) $\vee$ (`Height` $=$ `-1`)
- ▷ **`WindowHandle`** (output_control) ..... window ⇝ *integer*
  Fenster-Identifikator.

Beispiel

```
HTuple m_tHalconWindow ;
Hobject m_objImage ;

WM_CREATE:
/* here you should create your extern halcon window*/
HTuple tWnd, tDC ;
::set_check("~father") ;
tWnd = (INT)((INT*)&m_hWnd) ;
tDC = (INT)(INT*)GetWindowDC() ;
::new_extern_window(tWnd, tDC, 0, 0, sizeTotal.cx, sizeTotal.cy, &m_tHalconWindow) ;
::set_check("father") ;

WM_PAINT:
/* here you can draw halcon objects */
long l = 0 ;
if (m_thWindow != -1) {
/* don´t forget to set the dc !! */
HTuple tDC((INT)(INT*)&pDC->m_hDC) ;
HTuple tDCNull((INT)(INT*)&l) ;
::set_window_dc(m_tHalconWindow,tDC) ;
::disp_obj(pDoc->m_objImage, m_tHalconWindow) ;
/* release the graphic objects */
::set_window_dc(m_tHalconWindow, tDCNull) ;
}
```

```
WM_CLOSE:
/* close the halcon window */
if (m_tHalconWindow != -1) {
::close_window(m_tHalconWindow) ;
}
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert new_extern_window den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

new_extern_window ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

reset_obj_db

Mögliche Nachfolgerfunktionen

set_color, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_tshape, set_window_extents, get_window_extents, query_color, set_check, set_system

Alternativen

open_window, open_textwindow

Siehe auch

open_window, disp_region, disp_image, disp_color, set_lut, query_color, set_color, set_rgb, set_hsi, set_pixel, set_gray, set_part, set_part_style, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_window_extents, get_window_extents, set_window_attr, set_check, set_system

Modul

System

open_textwindow (: : Row, Column, Width, Height, BorderWidth, BorderColor, BackgroundColor, FatherWindow, Mode, Machine : WindowHandle)

Öffnen eines Textfensters.

open_textwindow öffnet ein neues Textfenster, das zur Aus- und Eingabe von Text, sowie für die Ausgabe von Bildern verwendet werden kann. Unter Verwendung der logischen Fensternummer WindowHandle können alle Ausgaben (write_string, read_string, disp_region, etc.) auf dieses Fenster geleitet werden.

Textfenster besitzen neben dem Mauscursor auch einen Textcursor, der die aktuelle Schreibposition bezeichnet (genauer: die linke untere Ecke des auszugebenden Strings ohne Berücksichtigung der Unterlängen). Seine Position wird durch einen Unterstrich (oder eine andere Form) angezeigt (die Anzeige dieser Position kann aber auch ausgeschaltet sein (= Defaulteinstellung); vgl. set_tshape). Mit den Prozeduren set_tposition und get_tposition kann die Position gesetzt bzw. abgefragt werden.

Beim Öffnen eines Textfensters wird die Cursorposition auf (H,0) gesetzt, wobei H die Höhe des Defaultfonts abzüglich der Unterlängen angibt. Der Cursor wird aber nicht angezeigt. Die Ausgabe beginnt also beim Schreiben in der linken oberen Ecke des Fensters.

Die Farben für den Hintergrund und den Bildrand können mit query_color abgefragt werden. Dies kann auch mit einem Fenster vom Typ ’invisible’ durchgeführt werden. Das Clipping von Texten bei der Ausgabe (write_string) an den Fensterrändern kann mit set_check(::’~text’:) eingeschaltet werden, d.h. es wird dann keine Fehlermeldung mehr erzeugt, falls der Text über den Rand des Fensters hinausragt.

Der Ursprung des Koordinatensystems des Fensters liegt in der linken oberen Ecke (Koordinaten: (0,0)). Nach unten nimmt der Zeilenindex zu (maximal: Height-1), nach rechts steigt der Spaltenindex (maximal: Width-1) an.

Der Parameter Machine gibt den Namen des Rechners an, auf dem das Fenster geöffnet werden soll. Im Falle von X-Window wird bei TCP-IP nur der Name, bei DEC-Net wird noch ein Doppelpunkt hinter den Namen gesetzt. Der „Server“ bzw. der „Screen“ wird nicht mit angegeben. Übergibt man den leeren String, so wird

die Environmentvariable DISPLAY zur Festlegung des Zielrechners verwendet. Hierbei wird der Name in der üblichen Syntax

```
<Host>:0.0
```

angegeben.

Die Position und Größe des Fensters kann sich im Laufe des Programms ändern. Dies kann zum einen mit `set_window_extents` erfolgen, aber auch durch externe Eingriffe (Windowmanager) hervorgerufen werden. Für den zweiten Fall ist die Prozedur `set_window_extents` vorgesehen.

Dem Fenster wird beim Öffnen ein sog. Defaultfont zugeordnet. Dieser wird für Prozeduren wie `write_string` verwendet und kann mit `set_font` nach dem Aufruf von `open_textwindow` überschrieben werden. Es ist jedoch auch möglich, den Defaultfont mit dem Aufruf `set_system(::'default_font',<Fontname>:)` vor dem Öffnen eines Fensters (und auch aller weiteren) festzulegen (siehe hierzu auch `query_font`).

Die Farbe der Schrift (`write_string`, `read_string`) wird mit `set_color`, `set_rgb`, `set_hsi`, `set_gray` oder `set_pixel` festgelegt. Mit `set_insert` wird bestimmt, wie der Text (bzw. die Graphik) mit dem Inhalt des Bildwiederholspeichers verknüpft wird. Hierbei kann z.B. mit `set_insert`(::'not':) erreicht werden, daß nach zweimaligem Schreiben des Textes an der gleichen Position die Schrift wieder beseitigt wird.

Normalerweise wird jede Ausgabe (z.B. mit `write_string`, `disp_region`, `disp_circle`, etc.) in ein Fenster durch einen sog. „Flush“ abgeschlossen. Dies bewirkt, daß die Daten nach Beendigung der Ausgabeprozedur auf dem Bildschirm vollständig sichtbar werden. Dies ist aber nicht in allen Fällen erforderlich, insbesondere dann, wenn ständig Ausgaben erfolgen, oder eine Maus-Prozedur aktiv ist. Hier ist es günstiger (d.h. schneller), wenn die Daten gepuffert werden, bis genügend Daten vorhanden sind. Dieses Verhalten kann durch den Aufruf von `set_system(::'flush_graphic','false':)` eingestellt werden.

Der Inhalt von Fenstern wird (falls die Treibersoftware dies unterstützt) gesichert; d.h. er bleibt erhalten, auch wenn das Fenster von anderen Fenstern überlappt wird. Dies ist jedoch nicht in allen Fällen notwendig: Wenn man ein Textfenster z.B. als Vaterfenster für andere Fenster verwendet, so kann für dieses der Sicherungsmechanismus abgestellt und damit der hierfür nötige Speicher eingespart werden. Dies geschieht mit dem Aufruf von `set_system(::'backing_store','false':)` vor dem Öffnen des Fensters.

Unterschied: Graphikfenster - Textfenster

- Im Gegensatz zu Graphikfenstern (`open_window`) können für Textfenster mehr Parameter (Farbe, Rand) beim Öffnen festgelegt werden.
- Nur Textfenster können zum Einlesen von Benutzerdaten verwendet werden (`read_string`).
- Die Ausgabe von Bildern, Regionen und Graphiken wird bei Textfenstern an den Rändern „geclippt“, während bei Graphikfenstern „gezoomt“ wird.
- Das Koordinatensystem (z.B. bei `get_mbutton` oder `get_mposition`) ist unabhängig von der Bildgröße immer in Bildschirmkoordinaten. Die maximalen Koordinaten entsprechen also der Größe des Fensters minus 1. Im Gegensatz dazu wird bei Graphikfenstern (`open_window`) immer ein Koordinatensystem verwendet, das dem Bildformat entspricht.

Der Parameter `Mode` legt den Modus des Fensters fest. Dieser kann folgende Werte annehmen:

'visible': Dies ist der normale Modus für Textfenster: Das Fenster wird entsprechend den Parametern erzeugt und alle Ein- und Ausgaben sind möglich.

'invisible': Unsichtbare Fenster werden nicht auf dem Bildschirm dargestellt. Die Parameter `Row`, `Column`, `BorderWidth`, `BorderColor`, `BackgroundColor` und `FatherWindow` sind ohne Bedeutung. Ausgaben auf diese Fenster haben keine Wirkung. Eingaben (`read_string`, Maus, etc.) sind nicht möglich. Diese Fenster werden verwendet, wenn man Darstellungsparameter für ein Ausgabegerät abfragen möchte, ohne ein (sichtbares) Fenster zu öffnen. Übliche Anfragen sind z.b. `query_color` und `get_string_extents`.

'transparent': Diese Fenster sind durchsichtig: Das Fenster selbst ist nicht sichtbar (Rand und Hintergrund), jedoch sind alle sonstigen Operationen möglich und alle Ausgaben werden dargestellt. Die Parameter `BorderColor` und `BackgroundColor` sind hier ohne Bedeutung. Eine übliche Verwendung für diesen Modus ist die Erzeugung von Maus-sensitiven Bereichen.

'buffer': Dies sind ebenfalls nicht sichtbare Fenster. Die Ausgabe von Bildern, Regionen und Graphik ist auf dem Bildschirm nicht sichtbar, wird jedoch im Speicher gepuffert. Die Parameter `Row`, `Column`, `BorderWidth`, `BorderColor`, `BackgroundColor` und `FatherWindow` sind ohne Bedeutung. Puffer-Fenster verwendet man beispielsweise, wenn man Ausgaben (im Hintergrund) vorbereitet und schließlich mit `copy_rectangle` in ein sichtbares Fenster kopiert. Eine andere Anwendung wäre die schnelle Verarbeitung von Bildauschnitten bei interaktiven Manipulationen. Texteingaben und Mausinteraktion ist mit dem Modus 'buffer' nicht möglich.

Achtung

Es ist zu beachten, daß die Parameter `Row`, `Column`, `Width` und `Height` durch das Ausgabegerät beschränkt sind. Wird ein Vaterfenster (`FatherWindow` <> 'root') angegeben, dann sind die Koordinaten relativ zum diesem Fenster.

Parameter

▷ **`Row`** (input_control) ..... rectangle.origin.y ⇝ *integer*
Zeilenindex der linken oberen Ecke.
Defaultwert : 0
Typischer Wertebereich : 0 ≤ `Row` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : `Row` ≥ `0`

▷ **`Column`** (input_control) ..... rectangle.origin.x ⇝ *integer*
Spaltenindex der linken oberen Ecke.
Defaultwert : 0
Typischer Wertebereich : 0 ≤ `Column` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : `Column` ≥ `0`

▷ **`Width`** (input_control) ..... rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert : 256
Typischer Wertebereich : 0 ≤ `Width` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : `Width` > `0`

▷ **`Height`** (input_control) ..... rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert : 256
Typischer Wertebereich : 0 ≤ `Height` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : `Height` > `0`

▷ **`BorderWidth`** (input_control) ..... integer ⇝ *integer*
Breite des Fensterrandes.
Defaultwert : 2
Typischer Wertebereich : 0 ≤ `BorderWidth` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : `BorderWidth` ≥ `0`

▷ **`BorderColor`** (input_control) ..... string ⇝ *string*
Farbe des Fensterrandes.
Defaultwert : 'white'

▷ **`BackgroundColor`** (input_control) ..... string ⇝ *string*
Hintergrundfarbe.
Defaultwert : 'black'

▷ **`FatherWindow`** (input_control) ..... integer ⇝ *integer* / string
Logische Nummer des Vaterfensters. Für den Bildschirm als Vater kann 'root' oder 0 eingegeben werden.
Defaultwert : 0
Restriktion : `FatherWindow` ≥ `0`

▷ **Mode** (input_control) ..... string ⇝ *string*
Fenstermodus.
Defaultwert : 'visible'
Werteliste : Mode ∈ {'visible', 'invisible', 'transparent', 'buffer'}

▷ **Machine** (input_control) ..... string ⇝ *string*
Name des Rechners, auf dem das Fenster geöffnet werden soll oder leerer String.
Defaultwert : ''

▷ **WindowHandle** (output_control) ..... window ⇝ *integer*
Fenster-Identifikator.

Beispiel

```
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)
repeat()
get_mposition(WindowHandle,Row,Column,Button)
get_grayval(Image,Row,Column,1,Gray)
write_string(WindowHandle,['  Position (',Row,',',Column,')  '])
write_string(WindowHandle,['Gray value (',Gray,')  '])
new_line(WindowHandle)
until(Button = 4)
close_window(WindowHandle)
clear_obj(Image).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert open_textwindow den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

open_textwindow ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

reset_obj_db

Mögliche Nachfolgerfunktionen

set_color, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_tshape, set_window_extents, get_window_extents, query_color, set_check, set_system

Alternativen

open_window

Siehe auch

write_string, read_string, new_line, get_string_extents, get_tposition, set_color, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_tshape, set_window_extents, get_window_extents, query_color, set_check, set_system

Modul

System

open_window (: : Row, Column, Width, Height, FatherWindow, Mode, Machine : WindowHandle)

Öffnen eines Graphikfensters.

open_window öffnet ein neues Fenster, das zur Ausgabe von Grauwertdaten, Regionen, Graphik sowie für Textausgabe verwendet werden kann. Alle Ausgaben (disp_region, disp_image, etc.) werden auf dieses Fenster geleitet, wenn dieselbe logische Fensternummer WindowHandle verwendet wird.

Der Hintergrund des erzeugten Fensters wird schwarz vorbesetzt und es hat einen weißen Rand von 2 Pixel Breite (siehe set_window_attr(::'border_width',<Breite>:).

Einem Fenster werden bestimmte Parameter für die Aufbereitung der auszugebenenden Daten zugeordnet, die bei der eigentlichen Ausgabe (z.B. mit disp_image oder disp_region) berücksichtigt werden. Diese Parameter werden aber nicht von der Ausgabeprozedur, sondern von „Konfigurationsprozeduren“ festgelegt. Will man z.B. die Farbe Rot für die Ausgabe von Regionen setzen, so muß vor dem Befehl disp_region der Aufruf set_color(::WindowHandle,'red':) erfolgen. Diese Parameter werden immer für das Fenster mit der logischen Fensternummer WindowHandle gesetzt und bleiben dem Fenster solange zugeordnet, bis sie überschrieben werden. Es stehen u.a. folgende Konfigurationsprozeduren zur Verfügung:

- Grauwertausgabe: set_paint, set_comprise, (set_lut und set_lut_style nach der Ausgabe)
- Regionen: set_color, set_rgb, set_hsi, set_gray, set_pixel, set_shape, set_line_width, set_insert, set_line_style, set_draw
- Bildausschnitt: set_part
- Text: set_font

Die aktuell gesetzten Werte können mit Prozeduren wie get_shape abgefragt werden. Da einige Parameter durch die Hardware (Auflösung/Farben) bestimmt werden, kann man mit Prozeduren wie query_color die aktuell verfügbaren Resourcen abfragen.

Der Ursprung des Koordinatensystems des Fensters liegt in der linken oberen Ecke (Koordinaten: (0,0)). Nach unten nimmt der Zeilenindex zu (maximal: Bildhöhe - 1), nach rechts steigt der Spaltenindex (maximal: Bildbreite - 1) an. Es ist zu beachten, daß der Wertebereich des Koordinatensystems unabhängig von der Fenstergröße ist. Er wird nur durch das Bildformat (siehe reset_obj_db) festgelegt.

Der Parameter Machine gibt den Namen des Rechners an, auf dem das Fenster geöffnet werden soll. Im Falle von X-Window wird bei TCP-IP nur der Name, bei DEC-Net wird noch ein Doppelpunkt hinter den Namen gesetzt. Der „Server“ bzw. der „Screen“ wird nicht mit angegeben. Übergibt man den leeren String, so wird die Environmentvariable DISPLAY zur Festlegung des Zielrechners verwendet. Hierbei wird der Name in der üblichen Syntax

```
<Host>:0.0
```

angegeben.

Für die Parameter Width und Height kann der Wert „-1“ als Größe verwendet werden. Dies bedeutet, daß der entsprechende Wert automatisch bestimmt werden soll. Dies ist insbesondere dann von Bedeutung, wenn das Pixelverhältnis nicht gleich 1.0 ist (siehe set_system). Wird einer der beiden Parameter als „-1“ angegeben, so wird er durch die Größe bestimmt, die sich aus dem Pixelverhältnis ergibt. Werden beide Paramater mit „-1“ festgelegt, so werden sie auf das maximale Bildformat gesetzt (weitergehende Informationen zum maximalen Bildformat finden sich in der Beschreibung von get_system unter „width“ und „height“).

Die Position und Größe des Fensters kann sich im Laufe des Programms ändern. Dies kann zum einen mit set_window_extents erfolgen, aber auch durch externe Eingriffe (Windowmanager) hervorgerufen werden. Für den zweiten Fall ist die Prozedur set_window_extents vorgesehen.

Dem Fenster wird beim Öffnen ein sog. Defaultfont zugeordnet. Dieser wird für Prozeduren wie write_string verwendet und kann mit set_font nach dem Aufruf von open_window überschrieben werden. Es ist jedoch auch möglich, den Defaultfont mit dem Aufruf set_system(::'default_font',<Fontname>:) vor dem Öffnen eines Fensters (und auch aller weiteren) festzulegen (siehe hierzu auch query_font).

Die Farbe für Graphik und Schrift, die für Ausgabeprozeduren wie disp_region oder disp_circle verwendet wird, wird mit set_rgb, set_hsi, set_gray oder set_pixel festgelegt. Mit set_insert wird eingestellt, wie die Graphik mit dem Inhalt des Bildwiederholspeichers verknüpft wird. Hierbei kann z.B. mit set_insert(::'not':) erreicht werden, daß nach zweimaligem Zeichnen eines Kreises an der gleichen Position dieser wieder beseitigt wird.

Normalerweise wird jede Ausgabe (z.B. disp_region, disp_image, disp_circle, etc.) in ein Fenster durch einen sog. „Flush“ abgeschlossen. Dies bewirkt, daß die Daten nach Beendigung der Ausgabeprozedur auf dem Bildschirm vollständig sichtbar werden. Dies ist aber nicht in allen Fällen erforderlich, insbesondere dann,

wenn ständig Ausgaben erfolgen, oder ein Maus-Prozedur aktiv ist. Hier ist es günstiger (d.h. schneller) wenn die Daten gepuffert werden, bis genügend Daten vorhanden sind. Dieses Verhalten kann durch den Aufruf von `set_system(::'flush_graphic','false':)` eingestellt werden.

Der Inhalt von Fenstern wird (falls die Treibersoftware dies unterstützt) gesichert; d.h. er bleibt erhalten auch wenn es von anderen Fenstern überlappt wurde. Dies ist jedoch nicht in allen Fällen notwendig. Wenn der Inhalt eines Fensters ständig neu aufgebaut wird (`copy_rectangle`), so kann für dieses der Sicherungsmechanismus abgestellt und damit der hierfür nötige Speicher eingespart werden. Dies geschieht mit dem Aufruf von `set_system (::'backing_store','false':)` vor dem Öffnen des Fensters. Durch dieses Vorgehen wird nicht nur Speicher, sondern auch Rechenzeit gespart, was beispielsweise für die Ausgabe von Videoclips von Bedeutung ist (siehe `copy_rectangle`).

Für die Ausgabe von Graphik (`disp_image`, `disp_region`, etc.) kann mit der Prozedur `set_part` das Fenster so eingestellt werden, daß es einen logischen Ausschnitt des Bildformates darstellt. Dies hat insbesondere zur Folge, daß von Bildern und Regionen lediglich dieser Ausschnitt (entsprechend vergrößert) ausgegeben wird.

Unterschied: Graphikfenster - Textfenster

- Bei Graphikfenstern ist das Layout nicht so variabel handhabbar wie bei Textfenstern.
- Nur Textfenster können zum Einlesen von Benutzerdaten verwendet werden (`read_string`).
- Bei der Ausgabe von Bildern, Regionen und Graphiken wird bei Graphikfenstern „gezoomt“: Unabhängig von Größe und Seitenverhältnis des Fensters werden die Bilder so transformiert, daß sie füllend in dem Fenster ausgegeben werden. Bei Textfenstern dagegen wird die Größe des Fensters bei der Ausgabe nicht beachtet (nur falls Clipping nötig ist).
- Bei Graphikfenstern ist das Koordinatensystem des Fensters immer gleich dem Koordinatensystem des Bildformates, während bei Textfenstern das Koordinatensystem unabhängig von der Bildgröße immer den Bildschirmkoordinaten entspricht.

Der Parameter `Mode` legt den Modus des Fensters fest. Dieser kann folgende Werte annehmen:

’visible’: Dies ist der normale Modus fuer Graphikfenster: Das Fenster wird entsprechend den Parametern erzeugt und alle Ein- und Ausgaben sind möglich.

’invisible’: Unsichtbare Fenster werden nicht auf dem Bildschirm dargestellt. Die Parameter `Row`, `Column` und `FatherWindow` sind ohne Bedeutung. Ausgaben auf diese Fenster haben keine Wirkung. Eingaben (`read_string`, Maus, etc.) sind nicht möglich. Diese Fenster werden verwendet, wenn man Darstellungsparameter für ein Ausgabegerät abfragen möchte, ohne ein (sichtbares) Fenster zu öffnen. Übliche Anfragen sind z.B. `query_color` oder `get_string_extents`.

’transparent’: Diese Fenster sind durchsichtig: Das Fenster selbst ist nicht sichtbar (Rand und Hintergrund), jedoch sind alle sonstigen Operationen möglich und alle Ausgaben werden dargestellt. Eine übliche Verwendung fuer diesen Modus ist die Erzeugung von Maus-sensitiven Bereichen.

’buffer’: Dies sind ebenfalls nicht sichtbare Fenster. Die Ausgabe von Bildern, Regionen und Graphiken ist auf dem Bildschirm nicht sichtbar, wird jedoch im Speicher gepuffert. Die Parameter `Row`, `Column` und `FatherWindow` sind ohne Bedeutung. Puffer-Fenster verwendet man beispielsweise, wenn man Ausgaben (im Hintergrund) vorbereitet und schließlich mit `copy_rectangle` in ein sichtbares Fenster kopiert. Eine andere Anwendung wäre die schnelle Verarbeitung von Bildausschnitten bei interaktiven Manipulationen. Texteingaben und Mausinteraktion ist mit dem Modus ’buffer’ nicht möglich.

Achtung

Es ist zu beachten, daß die Parameter `Row`, `Column`, `Width` und `Height` durch das Ausgabegerät beschränkt sind. Wird ein Vaterfenster (`FatherWindow` <> ’root’) angegeben, dann sind die Koordinaten relativ zu diesem Fenster.

Parameter

▷ **Row** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
Zeilenindex der linken oberen Ecke.
Defaultwert : 0
Typischer Wertebereich : 0 ≤ Row (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : Row ≥ 0

▷ **Column** (input_control) ..... rectangle.origin.x ⇝ *integer*
Spaltenindex der linken oberen Ecke.
Defaultwert : 0
Typischer Wertebereich : 0 ≤ Column (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : Column ≥ 0

▷ **Width** (input_control) ..... rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert : 256
Typischer Wertebereich : 0 ≤ Width (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : (Width > 0) ∨ (Width = -1)

▷ **Height** (input_control) ..... rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert : 256
Typischer Wertebereich : 0 ≤ Height (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : (Height > 0) ∨ (Height = -1)

▷ **FatherWindow** (input_control) ..... integer ⇝ *integer* / string
Logische Nummer des Vaterfensters. Für den Bildschirm als Vater kann 'root' oder 0 eingegeben werden.
Defaultwert : 0
Restriktion : FatherWindow ≥ 0

▷ **Mode** (input_control) ..... string ⇝ *string*
Fenstermodus.
Defaultwert : 'visible'
Werteliste : Mode ∈ {'visible', 'invisible', 'transparent', 'buffer'}

▷ **Machine** (input_control) ..... string ⇝ *string*
Name des Rechners, auf dem das Fenster geöffnet werden soll oder leerer String.
Defaultwert : ''

▷ **WindowHandle** (output_control) ..... window ⇝ *integer*
Fenster-Identifikator.

Beispiel

```
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')
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert open_window den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

open_window ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

reset_obj_db

Mögliche Nachfolgerfunktionen

set_color, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_tshape, set_window_extents, get_window_extents, query_color, set_check, set_system

Alternativen

open_textwindow

Siehe auch

disp_region, disp_image, disp_color, set_lut, query_color, set_color, set_rgb, set_hsi, set_pixel, set_gray, set_part, set_part_style, query_window_type, get_window_type, set_window_type, get_mposition, set_tposition, set_window_extents, get_window_extents, set_window_attr, set_check, set_system

Modul

System

query_window_type (: : : WindowTypes)

Abfrage der verfügbaren Fenstertypen.

query_window_type gibt ein Tupel zurück, das alle Geräte bzw. Softwaresysteme enthält, mit denen Bildobjekte dargestellt werden können. query_window_type kann dort sinnvoll eingesetzt werden, wo man maschinenunabhängige Programme entwickeln möchte. Mögliche Werte sind:

'X-Window' X-Window Version 11.

'pixmap' Fenster werden nicht angezeigt, sondern nur im Speicher verwaltet. Auf diese Weise können Halcon-Programme auch auf Rechnern ohne Graphikbildschirm portiert werden.

'PostScript' Die Ausgabe von Objekten erfolgt in eine PostScript-Datei.

Parameter

▷ **WindowTypes** (output_control) ..... string-array ⇝ *string*
Namen der verfügbaren Fenstertypen.

Ergebnis

query_window_type liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

query_window_type ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

reset_obj_db

Modul

System

set_window_attr (: : AttributeName, AttributeValue :)

Setzen von Fenstereigenschaften.

set_window_attr dient zum Setzen von bestimmten Eigenschaften von Graphikfenstern. Damit können folgende Defaultparameter eines Fenster verändert werden:

'border_width' Breite des Fensterrandes in Pixel.

'border_color' Farbe des Fensterrandes.

'background_color' Hintergrundfarbe des Fensters.

'window_title' Name des Fensters in der Titlebar.

Achtung

`set_window_attr` muß *vor* dem Aufruf von `open_window` aufgerufen werden.

Parameter

- ▷ **`AttributeName`** (input_control) ..... string ⇝ *string*
  Name des Attributs, das verändert werden soll.
  Werteliste : `AttributeName` ∈ {'border_width', 'border_color', 'background_color', 'window_title'}
- ▷ **`AttributeValue`** (input_control) ..... string ⇝ *string* / integer
  Attributwert, der gesetzt werden soll.
  Werteliste : `AttributeValue` ∈ {0, 1, 2, 'white', 'black', 'MyName', 'default'}

Ergebnis

`set_window_attr` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_window_attr` ist *wiedereintrittsfähig* („reentrant"), *lokal* auszuführen („local") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_window`, `set_draw`, `set_color`, `set_colored`, `set_line_width`, `open_textwindow`

Siehe auch

`open_window`

Modul

System

`set_window_dc ( : : WindowHandle, WINHDC : )`

Setzen des Grafikkontextes eines virtuellen Grafikfensters (Windows NT).

`set_window_dc` setzt den Gerätekontext eines zuvor mit `new_extern_window` angelegten Fensters. Alle Ausgaben (`disp_region`, `disp_image`, etc.) werden auf dem zu diesen Gerätekontext gehörenden Fensters getätigt.

Im Parameter `WINHDC` wird der Gerätekontext des Windows-Fensters, das zur Ausgabe der Daten verwendet werden soll, übergeben. Dieser Gerätekontext wird in den Ausgaberoutinen von Halcon verwendet.

Achtung

Es ist zu beachten, daß das Fenster `WindowHandle` vorher mit `new_extern_window` angelegt worden sein muss.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`WINHDC`** (input_control) ..... integer ⇝ *integer*
  Gerätekontext des WINHWnd
  Restriktion : `WINHDC` ≠ `0`

Beispiel

```
hWnd = createWINDOW(...)
new_extern_window(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_lut(WindowHandle,'temperature')
set_color(WindowHandle,'blue')
```

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_window_dc` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_window_dc` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`new_extern_window`

Mögliche Nachfolgerfunktionen

`disp_image`, `disp_region`

Siehe auch

`new_extern_window`, `disp_region`, `disp_image`, `disp_color`, `set_lut`, `query_color`, `set_color`, `set_rgb`, `set_hsi`, `set_pixel`, `set_gray`, `set_part`, `set_part_style`, `query_window_type`, `get_window_type`, `set_window_type`, `get_mposition`, `set_tposition`, `set_window_extents`, `get_window_extents`, `set_window_attr`, `set_check`, `set_system`

Modul

System

`set_window_extents ( : : WindowHandle, Row, Column, Width, Height : )`

Verändern der Position und Größe eines Fensters.

`set_window_extents` positioniert die linke obere Ecke des Ausgabefensters auf (`Row`,`Column`) und verändert gleichzeitig die Größe des Fensters auf `Width` und `Height`.

Achtung

Wird die Größe des Fensters verändert, so erfolgt nicht automatisch eine Anpassung der dargestellten Daten an das neue Format. Dies muß vom Programm aus vorgenommen werden, indem die Daten erneut ausgegeben werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`Row`** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilenindex der linken oberen Ecke der Zielposition.
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `Row` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Column`** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltenindex der linken oberen Ecke der Zielposition.
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `Column` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

▷ **Width** (input_control) . . . . . rectangle.extent.x ⇝ *integer*
Breite des Fensters.
Defaultwert : 512
Typischer Wertebereich : 0 ≤ Width (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Height** (input_control) . . . . . rectangle.extent.y ⇝ *integer*
Höhe des Fensters.
Defaultwert : 512
Typischer Wertebereich : 0 ≤ Height (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Ergebnis

set_window_extents liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt

Parallelisierungsinformation

set_window_extents ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Siehe auch

get_window_extents, open_window, open_textwindow

Modul

System

set_window_type (: : WindowType :)

Festlegen des Fenstertyps.

set_window_type bestimmt, auf welchen Typ von Ausgabegerät die Ausgaben erfolgen sollen. Diese Einstellung wird von der Prozedur open_window beim Öffnen der Fenster verwendet. Verschiedene Fenster können auch auf verschieden Typen von Ausgabegeräten geöffnet werden. Dazu stellt man vor dem Öffnen jeweils den gewünschten Typ ein. Die verfügbaren Typen von Ausgabegeräten können über die Prozedur query_window_type abgefragt werden. Mögliche Werte sind:

’X-Window’ X-Window Version 11.

’pixmap’ Fenster werden nicht angezeigt, sondern nur im Speicher verwaltet. Auf diese Weise können Halcon-Programme auch auf Rechnern ohne Graphikbildschirm portiert werden.

’PostScript’ Die Ausgabe von Objekten erfolgt in eine PostScript-Datei.

’WIN32-Window’ Microsoft Windows.

set_window_type kann dort sinnvoll eingesetzt werden, wo man maschinenunabhängige Programme entwickeln möchte.

Parameter

▷ **WindowType** (input_control) . . . . . string ⇝ *string*
Namen des Fenstertyps, der gesetzt werden soll.
Defaultwert : ’X-Window’
Werteliste : WindowType ∈ {’X-Window’, ’WIN32-Window’, ’pixmap’, ’PostScript’}

Ergebnis

set_window_type liefert den Wert 2 (H_MSG_TRUE), falls der Ausgabegerätetyp verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_window_type ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`

Siehe auch

`open_window`, `open_textwindow`, `query_window_type`, `get_window_type`

Modul

System

`slide_image` `( : : WindowHandleSource1, WindowHandleSource2, WindowHandle : )`

Interaktive Ausgabe aus zwei Fenster-Puffern.

`slide_image` teilt das Fenster in Abhängigkeit von der Mausposition horizontal in zwei logische Bereiche. In den oberen wird der Inhalt des ersten angegebenen Fensters kopiert, in den unteren der Inhalt des zweiten Fensters. Drückt man die linke Maustaste, kann die Grenze zwischen den beiden Bereichen verschoben werden (die Maus darf dazu auch außerhalb des Fensters bewegt werden. Dabei legt die Position der Maus relativ zum Fenster die Trennlinie fest).

Durch einen Klick mit der rechten Maustaste in das Fenster wird die Prozedur `slide_image` beendet.

Die Prozedur `slide_image` läßt sich sinnvoll verwenden, um die Auswirkung etwa einer Filteroperation auf ein Bild zu visualisieren. Die Ausgabe erfolgt auf das aktuell gesetzte Fenster (`WindowHandle`).

Achtung

Die drei Fenster müssen die gleiche Größe haben und sich auf dem gleichen Rechner befinden.

Parameter

- ▷ **`WindowHandleSource1`** (input_control) ..... window ⇝ *integer*
  Logische Fensternummer des „oberen Fensters“.
- ▷ **`WindowHandleSource2`** (input_control) ..... window ⇝ *integer*
  Logische Fensternummer des „unteren Fensters“.
- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.

Beispiel

```
read_image(Image,'fabrik')
sobel_amp(Image,Amp,'sum_abs',3)
open_window(0,0,-1,-1,'root','buffer','',WindowHandle)
disp_image(Amp,WindowHandle)
sobel_dir(Image,Dir,'sum_abs',3)
open_window(0,0,-1,-1,'root','buffer','',WindowHandle)
disp_image(Dir,WindowHandle)
open_window(0,0,-1,-1,'root','visible','',WindowHandle)
slide_image(Puffer1,Puffer2,WindowHandle).
```

Ergebnis

`slide_image` liefert den Wert 2 (H_MSG_TRUE), falls beide Fenster existieren und ein Fenster davon gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`slide_image` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`

Alternativen

`copy_rectangle`, `get_mposition`

Siehe auch

`open_window`, `open_textwindow`, `move_rectangle`

Modul

System

4.4 Gnuplot

gnuplot_close (: : GnuplotFileID :)

Schließen von offenen gnuplot-Dateien oder Beendigung von laufenden gnuplot-Unterprozessen.

gnuplot_close schließt alle offenen Dateien, die mit gnuplot_open_file geöffnet wurden, oder beendet den gnuplot-Unterprozeß, der mit gnuplot_open_pipe gestartet wurde. In letzterem Fall werden auch alle temporären Dateien, die zur 3D-Darstellung von Bildern und von Steuerparametern benötigt werden, wieder gelöscht. Daher muß nach Beendigung einer Plot-Sequenz unbedingt gnuplot_close ausgeführt werden. GnuplotFileID ist der Identifikator des zugehörigen gnuplot-Ausgabestroms.

Parameter

▷ **GnuplotFileID** (input_control) . . . . . gnuplot_id ⇝ *integer*
Identifikator fuer den gnuplot-Ausgabestrom.

Ergebnis

gnuplot_close liefert 2 (H_MSG_TRUE), falls GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gnuplot_close wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

gnuplot_open_pipe, gnuplot_open_file, gnuplot_plot_image

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_plot_image

Modul

System

gnuplot_open_file (: : FileName : GnuplotFileID)

Öffnen einer gnuplot-Datei fuer Bilder und Steuerparameter.

gnuplot_open_file erlaubt die Ausgabe von Bildern und Steuerparametern in einem Format, das später von gnuplot verarbeitet werden kann. Der Parameter FileName gibt den Basisnamen für die mit gnuplot_plot_image zu erstellenden Plots an. gnuplot_open_file erzeugt daraus eine Steuerdatei mit dem Namen <FileName>.gp, in der die einzelnen Plot-Befehle abgelegt werden. Für jedes ausgegebene Bild (oder Steuereparamater) wird dann von gnuplot_plot_image eine Datei angelegt, die die eigentlichen Daten enthält, und den Namen <FileName>.dat.<Zahl> erhält, wobei Zahl die Nummer des Plots in der aktuellen Folge ist. Die erzeugte Steuerdatei kann später vom Benutzer an die eigenen Bedürfnisse angepaßt werden. Nach Ausgabe des letzten Plots muß gnuplot_close ausgeführt werden, um die offenen Dateien zu schließen. Der zugehörige Identifikator für den gnuplot-Ausgabestrom wird in GnuplotFileID zurückgeliefert.

Parameter

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Basisname für die Steuer- und Ausgabedateien.

▷ **GnuplotFileID** (output_control) . . . . . gnuplot_id ⇝ *integer*
Identifikator fuer den gnuplot-Ausgabestrom.

Ergebnis

gnuplot_open_file liefert den Wert 2 (H_MSG_TRUE), falls die Steuerdatei geöffnet werden konnte. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gnuplot_open_file wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

gnuplot_plot_image, gnuplot_close

Alternativen

gnuplot_open_pipe

Siehe auch

gnuplot_open_pipe, gnuplot_close, gnuplot_plot_image

Modul

System

gnuplot_open_pipe (: : : GnuplotFileID)

Öffnen einer Pipe zu einem gnuplot-Unterprozeß zur Darstellung von Bildern und Steuerparametern.

gnuplot_open_pipe öffnet eine Pipe zu einem gnuplot-Unterprozeß mit dem nachfolgend durch gnuplot_plot_image Bilder als 3D-Plots und gnuplot_plot_ctrl Steuerparameter dargestellt werden können. Der Unterprozeß muß nach Ausgabe des letzten Plots mit gnuplot_close beendet werden. Der zugehörige Identifikator für den gnuplot-Ausgabestrom wird in GnuplotFileID zurückgeliefert.

Achtung

gnuplot_open_pipe ist nur unter Unix implementiert, da sich gnuplot unter Windows (wgnuplot) nicht durch einen externen Prozeß steuern läßt.

Parameter

- ▷ **GnuplotFileID** (output_control) ..... gnuplot_id ⇝ *integer*
  Identifikator fuer den gnuplot-Ausgabestrom.

Ergebnis

gnuplot_open_pipe liefert den Wert 2 (H_MSG_TRUE), falls der gnuplot-Unterprozeß kreiert werden konnte. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gnuplot_open_pipe wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

gnuplot_plot_image, gnuplot_plot_ctrl, gnuplot_close

Alternativen

gnuplot_open_file

Modul

System

gnuplot_plot_ctrl (: : GnuplotFileID, Values :)

Ausgabe von Tupeln durch gnuplot.

gnuplot_plot_ctrl gibt ein Tupel unter Verwendung von gnuplot als Plot aus. Falls vorher ein gnuplot-Unterprozeß mittels gnuplot_open_pipe gestartet wurde, erfolgt die Darstellung in einem gnuplot-Fenster. Ansonsten erfolgt die Ausgabe in eine Datei, die später mit gnuplot verarbeitet werden kann. In beiden Fällen wird der gnuplot-Ausgabestrom durch GnuplotFileID bestimmt.

Parameter

- ▷ **GnuplotFileID** (input_control) ..... gnuplot_id ⇝ *integer*
  Identifikator fuer den gnuplot-Ausgabestrom.
- ▷ **Values** (input_control) ..... number-array ⇝ *real* / integer
  Auszugebende Werte (y-Werte).

Ergebnis

gnuplot_plot_ctrl liefert den Wert 2 (H_MSG_TRUE), falls nur ganze Zahlen oder Gleitpunktzahlen übergeben wurden, GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist und die Datei für den aktuellen plot erzeugt werden konnte. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gnuplot_plot_ctrl wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

gnuplot_open_pipe, gnuplot_open_file

Mögliche Nachfolgerfunktionen

gnuplot_close

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_close

Modul

System

gnuplot_plot_funct_1d (: : GnuplotFileID, Function :)

Ausgabe einer Funktion durch gnuplot.

gnuplot_plot_ctrl gibt eine Funktion unter Verwendung von gnuplot als Plot aus. Falls vorher ein gnuplot-Unterprozeß mittels gnuplot_open_pipe gestartet wurde, erfolgt die Darstellung in einem gnuplot-Fenster. Ansonsten erfolgt die Ausgabe in eine Datei, die später mit gnuplot verarbeitet werden kann. In beiden Fällen wird der gnuplot-Ausgabestrom durch GnuplotFileID bestimmt.

Parameter

- **GnuplotFileID** (input_control) ..... gnuplot_id ⇝ *integer*
  Identifikator fuer den gnuplot-Ausgabestrom.
- **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Auszugebende Funktion.

Ergebnis

gnuplot_plot_funct_1d liefert den Wert 2 (H_MSG_TRUE), falls GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist und die Datei für den aktuellen plot erzeugt werden konnte. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gnuplot_plot_funct_1d wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

gnuplot_open_pipe, gnuplot_open_file

Mögliche Nachfolgerfunktionen

gnuplot_close

Alternativen

gnuplot_plot_ctrl

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_close

Modul

System

gnuplot_plot_image (Image : : GnuplotFileID, SamplesX, SamplesY, ViewRotX, ViewRotZ, Hidden3D :)

Ausgabe von Bildern durch gnuplot.

gnuplot_plot_image gibt Bilder unter Verwendung von gnuplot als 3D-Plots aus. Falls vorher ein gnuplot-Unterprozeß mittels gnuplot_open_pipe gestartet wurde, erfolgt die Darstellung in einem gnuplot-Fenster. Ansonsten erfolgt die Ausgabe in eine Datei, die später mit gnuplot verarbeitet werden kann. In beiden Fällen wird der gnuplot-Ausgabestrom durch GnuplotFileID bestimmt. Die Parameter SamplesX und SamplesY geben an, wieviele Datenpunkte in x- bzw. y-Richtung an gnuplot ausgegeben werden sollen. Sie sind vollkommen analog zu den Variablen samples und isosamples in gnuplot. Mit den Parametern ViewRotX und ViewRotZ läßt sich die Position des Beobachters festlegen. ViewRotX gibt die Rotation des Koordinatensystems um die x-Achse an. Dies entspricht im wesentlichen der Neigung des Plots. ViewRotZ gibt die Rotation des Plots um die z-Achse an. Diese beiden Parameter entsprechen den ersten beiden Parametern des ’set view’-Befehls von gnuplot. Hidden3D gibt an, ob die verdeckten Oberflächenteile entfernt werden sollen oder nicht. Dies entspricht dem ’set

hidden3d'-Befehl von gnuplot. Falls ein einziges Bild übergeben wird, erfolgt die Ausgabe in einen separaten Plot. Falls mehrere Bilder übergeben werden, werden alle in einem einzigen Plot dargestellt.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Auszugebendes Bild.
- ▷ **GnuplotFileID** (input_control) ..... gnuplot_id ⇝ *integer*
  Identifikator fuer den gnuplot-Ausgabestrom.
- ▷ **SamplesX** (input_control) ..... integer ⇝ *integer*
  Anzahl der Datenpunkte in x-Richtung.
  Defaultwert : 64
  Typischer Wertebereich : $2 \leq$ SamplesX ≤ 10000
  Restriktion : SamplesX ≥ 2
- ▷ **SamplesY** (input_control) ..... integer ⇝ *integer*
  Anzahl der Datenpunkte in y-Richtung.
  Defaultwert : 64
  Typischer Wertebereich : $2 \leq$ SamplesY ≤ 10000
  Restriktion : SamplesY ≥ 2
- ▷ **ViewRotX** (input_control) ..... number ⇝ *real* / integer
  Rotation des Plots um die x-Achse.
  Defaultwert : 60
  Typischer Wertebereich : $0 \leq$ ViewRotX ≤ 180
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10
  Restriktion : $(0 \leq$ ViewRotX$) \wedge ($ViewRotX $\leq 180)$
- ▷ **ViewRotZ** (input_control) ..... number ⇝ *real* / integer
  Rotation des Plots um die z-Achse.
  Defaultwert : 30
  Typischer Wertebereich : $0 \leq$ ViewRotZ ≤ 360
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10
  Restriktion : $(0 \leq$ ViewRotZ$) \wedge ($ViewRotZ $\leq 360)$
- ▷ **Hidden3D** (input_control) ..... string ⇝ *string*
  Ausgabe als hidden-surface-plot.
  Defaultwert : 'hidden3d'
  Werteliste : Hidden3D $\in$ {'hidden3d', 'nohidden3d'}

Ergebnis

gnuplot_plot_image liefert den Wert 2 (H_MSG_TRUE), falls GnuplotFileID ein gültiger gnuplot-Ausgabestrom ist und die Datei für den aktuellen plot erzeugt werden konnte. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gnuplot_plot_image wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

gnuplot_open_pipe, gnuplot_open_file

Mögliche Nachfolgerfunktionen

gnuplot_close

Siehe auch

gnuplot_open_pipe, gnuplot_open_file, gnuplot_close

Modul

System

4.5 Maus

get_mbutton (: : WindowHandle : Row, Column, Button)

Abfragen der Mausposition bei Tastendruck.

get_mbutton liefert die Koordinaten der Maus im Ausgabefenster und die dabei betätigten Tasten Button:

1: Linke Taste,

2: Mittlere Taste,

4: Rechte Taste.

Es wird so lange gewartet, bis eine der Tasten in dem angegebenen HALCON-Fenster gedrückt wird. Falls mehr als eine Taste betätigt wird, ergibt sich der Wert als die Summe der einzelnen Tasten. Der Ursprung des Koodinatensystems (0,0) liegt im linken oberen Eck des Fensters. Der Zeilenindex nimmt nach unten, der Spaltenindex nach rechts zu. Bei Graphikfenstern geht das Koordinatensystem bis: (Bildhöhe-1,Bildbreite-1) (siehe: open_window, reset_obj_db), bei Textfenstern bis: (Fensterhöhe-1,Fensterbreite-1) (siehe open_textwindow).

Achtung

get_mbutton kommt erst zurück, wenn mit der Maus in das Fenster geklickt wird.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- ▷ **Row** (output_control) ..... point.y ⇝ *integer*
  Zeilenindex der Mausposition innerhalb der Fensters.
- ▷ **Column** (output_control) ..... point.x ⇝ *integer*
  Spaltenindex der Mausposition innerhalb der Fensters.
- ▷ **Button** (output_control) ..... integer ⇝ *integer*
  Betätigte Taste(n).

Ergebnis

get_mbutton liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

get_mbutton ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Alternativen

get_mposition

Siehe auch

open_window, open_textwindow

Modul

System

get_mposition (: : WindowHandle : Row, Column, Button)

Abfragen der Mausposition.

get_mposition liefert die Koordinaten der Maus im Ausgabefenster und den Code für die betätigten Tasten. Die Daten werden unabhängig vom Zustand der Tasten (betätigt oder nicht betätigt) abgeliefert. Werden mehrere Tasten gleichzeitig betätigt, dann werden die zugehörigen Zahlenwerte addiert. Die Werte für Button:

0: Keine Taste,

1: Linke Taste,

2: Mittlere Taste,

4: Rechte Taste.

Der Ursprung des Koodinatensystems (0,0) liegt im linken oberen Eck des Fensters. Der Zeilenindex nimmt nach unten, der Spaltenindex nach rechts zu. Bei Graphikfenstern geht das Koordinatensystem bis: (Bildhöhe-1, Bildbreite-1) (siehe: open_window, reset_obj_db), bei textfenstern bis: (Fensterhöhe-1,Fensterbreite-1) (siehe open_textwindow).

Achtung

`get_mposition` scheitert (FAIL), falls sich die Maus nicht innerhalb des Fensters befindet. In diesem Fall werden in den drei Parametern keine Werte übergeben.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- ▷ **`Row`** (output_control) ..... point.y ⇝ *integer*
  Zeilenindex der Mausposition innerhalb der Fensters.
- ▷ **`Column`** (output_control) ..... point.x ⇝ *integer*
  Spaltenindex der Mausposition innerhalb der Fensters.
- ▷ **`Button`** (output_control) ..... integer ⇝ *integer*
  Betätigte Taste(n) oder 0.

Ergebnis

`get_mposition` liefert den Wert 2 (H_MSG_TRUE). Befindet sich der Cursor ausserhalb des Fensters, wird 5 (H_MSG_FAIL) ausgegeben.

Parallelisierungsinformation

`get_mposition` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`

Alternativen

`get_mbutton`

Siehe auch

`open_window`, `open_textwindow`

Modul

System

`get_mshape ( : : WindowHandle : Cursor )`

Abfragen der Mauscursor Form.

`get_mshape` gibt für das Ausgabefenster den Namen des momentan verwendeten Mauscursors aus. Gesetzt wird ein neuer Mauscursor mit der Prozedur `set_mshape`. Unter einem Mauscursor (im Gegensatz zum Textcursor; beide werden einfach als `Cursor` bezeichnet, wenn Mißverständnisse ausgeschlossen sind) wird hier das Muster verstanden, das synchron zur Maus auf dem Bildschirm dargestellt wird.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **`Cursor`** (output_control) ..... string ⇝ *string*
  Name der Cursorform.

Ergebnis

`get_mshape` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`get_mshape` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`, `query_mshape`

Mögliche Nachfolgerfunktionen

`set_mshape`

Siehe auch

`set_mshape`, `query_mshape`

Modul

System

query_mshape (: : WindowHandle : ShapeNames)

Abfragen aller Formen für den Maus-Cursor.

query_mshape gibt für das Ausgabefenster die Namen aller Mauscursormuster aus. Diese können von der Prozedur set_mshape verwendet werden. Unter einem Mauscursor (im Gegensatz zu Textcursor; beide werden einfach als Cursor bezeichnet, wenn Mißverständnisse ausgeschlossen sind) wird hier das Muster verstanden, das bei der Bewegung der Maus auf dem Bildschirm erscheint. Ist keine Maus vorhanden, dann wird das leere Tupel als Ergebnis abgeliefert.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **ShapeNames** (output_control) ..... string-array ⇝ *string*
  Namen der Cursor.

Ergebnis

query_mshape liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

query_mshape ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, get_mshape

Mögliche Nachfolgerfunktionen

set_mshape

Siehe auch

set_mshape, get_mshape

Modul

System

set_mshape (: : WindowHandle, Cursor :)

Setzen der Mauscursor Form.

set_mshape setzt das Muster, das für den Mauscursor des Ausgabefensters verwendet werden soll. Unter Mauscursor (im Gegensatz zu Textcursor; beide werden einfach als Cursor bezeichnet, wenn Mißverständnisse ausgeschlossen sind) wird hier das Muster verstanden, das auf dem Bildschirm bei einer Bewegung der Maus dargestellt wird. Ein Tupel aller verfügbaren Muster kann mit der Prozedur query_mshape abgefragt werden. Der Cursor ist dem Fenster im weiteren zugeordnet (d.h. das Muster wird bei Eintritt in das Fenster verwendet) und zwar unabhängig davon, welches Fenster die aktuellen Ausgaben verarbeitet.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster-Identifikator.
- ▷ **Cursor** (input_control) ..... string ⇝ *string*
  Name des Mauscursors.
  Defaultwert : ’arrow’

Ergebnis

set_mshape liefert den Wert 2 (H_MSG_TRUE), falls für das Fenster der angegebene Cursor definiert ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_mshape ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, query_mshape, get_mshape

Siehe auch

get_mshape, query_mshape

Modul

System

4.6 Parameter

get_comprise (: : WindowHandle : Mode)

Abfragen der Behandlung der Bildmatrix bei der Ausgabe.

get_comprise gibt den von den Prozeduren disp_image und disp_color verwendeten Modus des Ausgabefensters WindowHandle für die Behandlung von Grauwerten an. Es wird hierdurch festgelegt, ob nur die Grauwerte der Objekte oder das ganze Bild verwendet wird. Diese Abfrage wird verwendet, wenn der aktuelle Modus kurzzeitig überschrieben werden soll. Hierzu wird zunächst der aktuelle Wert abgefragt, dann überschrieben und schließlich auf den alten Wert zurückgesetzt. Das Zurücksetzen erfolgt mit set_comprise.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Aktueller Ausgabemodus für Bilder.

Ergebnis

get_comprise liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_comprise wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_comprise, disp_image, disp_image

Siehe auch

set_comprise, disp_image, disp_color

Modul

System

get_draw (: : WindowHandle : Mode)

Abfragen des aktuellen Modus für das Ausfüllen von Regionen.

get_draw gibt den Ausfüllmodus Mode des Ausgabefensters aus, der von Prozeduren wie disp_region, disp_circle, disp_arrow, disp_rectangle1, disp_rectangle2 etc. verwendet wird. Gesetzt wird der Darstellungsparameter mit der Prozedur set_draw.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Aktueller Regionen-Füllmodeus.

Ergebnis

get_draw liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_draw wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_draw, disp_region

Siehe auch

set_draw, disp_region, set_paint

Modul

System

get_fix (: : WindowHandle : Mode)

Abfragen des aktuellen Fixiermodus.

get_fix liefert den durch set_fix gesetzten Fixierungsmodus für die aktuelle Farbtabelle des gültigen Fensters .

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- ▷ **Mode** (output_control) ..... string ⇝ *string*
  Aktueller Fixiermodus.

Ergebnis

Wenn das Fenster gültig ist liefert get_fix 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung ausgeführt.

Parallelisierungsinformation

get_fix wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_fix, set_pixel, set_rgb

Siehe auch

set_fix

Modul

System

get_hsi (: : WindowHandle : Hue, Saturation, Intensity)

Abfragen der aktuellen Farbe in HSI-Kodierung.

get_hsi gibt die Ausgabefarben, bzw. Grauwerte für das Ausgabefenster an. Diese werden durch Hue (Farbwert), Saturation (Sättigung) und Intensity (Intensität) beschrieben. get_hsi entspricht der Prozedur get_pixel, mit dem Unterschied, daß hier nicht die Indizes der Farbtabelle, sondern die Einträge in der Farbtabelle ausgegeben werden. Die Werte die man mit get_hsi erhält können mit set_hsi wieder gesetzt werden.

Achtung

Die Werte von get_hsi können durch Rundungsfehler ungenau sein. Sie entsprechen nicht genau den Werten die z.B. mit set_hsi gesetzt wurden (intern wird die Farbe in RGB gehalten).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Hue** (output_control) ..... integer-array ⇝ *integer*
  Farbwert der aktuellen Zeichenfarbe.
- ▷ **Saturation** (output_control) ..... integer-array ⇝ *integer*
  Sättigung der aktuellen Zeichenfarbe.
- ▷ **Intensity** (output_control) ..... integer-array ⇝ *integer*
  Intensität der aktuellen Zeichenfarbe.

Ergebnis

get_hsi liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_hsi` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`set_hsi`, `set_rgb`, `disp_image`

Siehe auch

`set_hsi`, `set_color`, `set_rgb`, `trans_to_rgb`, `trans_from_rgb`

Modul

System

get_icon (: Icon : WindowHandle :)

Abfragen des Icons, das bei der Regionenausgabe verwendet wird

`get_icon` fragt das Icon, das mit `set_icon` gesetzt wurde.

Parameter

- ▷ **Icon** (output_object) ..... region ⇝ *Hobject*
  Icon für Schwerpunkt.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.

Beispiel

```
/* draw a region and an icon.  */
/* set it and get it again. */
draw_region(&Region,WindowHandle) ;
draw_region(&Icon,WindowHandle) ;
set_icon(Icon) ;
set_shape(WindowHandle,"icon") ;
disp_region(Region,WindowHandle) ;
get_icon(&OldIcon) ;
disp_region(OldIcon,WindowHandle) ;
```

Ergebnis

`get_icon` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`get_icon` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`set_icon`

Mögliche Nachfolgerfunktionen

`disp_region`

Modul

System

get_insert (: : WindowHandle : Mode)

Abfragen der aktuellen Darstellungsfunktion.

`get_insert` gibt den Zeichenmodus des Ausgabefensters aus, der von Prozeduren wie `disp_region`, `disp_line`, `disp_rectangle1`, etc. verwendet wird. Gesetzt wird der Parameter mit der Prozedur `set_insert`. Mögliche Werte für `Mode` können mit Hilfe der Prozedur `query_insert` abgefragt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (output_control) . . . . . string ⇝ *string*
  Zeichenmodus.

Ergebnis

get_insert liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_insert wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

query_insert

Mögliche Nachfolgerfunktionen

set_insert, disp_image

Siehe auch

set_insert, query_insert, disp_region, disp_line

Modul

System

get_line_approx (: : WindowHandle : Approximation)

Abfragen des aktuellen Approximationsfehlers für Konturen.

get_line_approx gibt einen Parameter für die Darstellungsart für Konturen der Regionenausgabe des Ausgabefensters an, die von der Prozedur disp_region verwendet wird. Konkret steuert dieser Parameter die Polygonapproximation für Konturen (0 ⇔ keine Approximation). Gesetzt wird der Parameter mit der Prozedur set_line_approx. Der Parameter ist nur von Bedeutung, wenn die Ränder von Objekten dargestellt werden. Insbesondere wenn mit set_line_style ein Muster gesetzt wurde.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Approximation** (output_control) . . . . . string ⇝ *integer*
  Aktueller Approximationsfehler für Konturdarstellung.

Ergebnis

get_line_approx liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_line_approx wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_line_approx, set_line_style, disp_region

Siehe auch

get_region_polygon, set_line_approx, set_line_style, disp_region

Modul

System

get_line_style (: : WindowHandle : Style)

Abfragen des aktuellen Graphikmodus für Konturen.

get_line_style gibt die Darstellungsart für Konturen der Regionenausgabe des Ausgabefensters an, die von Prozeduren wie disp_region, disp_line, disp_polygon, etc. verwendet wird. Gesetzt wird der Parameter mit der Prozedur set_line_style. Der Parameter ist nur von Bedeutung, wenn die Ränder von Objekten dargestellt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Style** (output_control) . . . . . integer-array ⇝ *integer*
  Muster für Konturdarstellung.

Ergebnis

`get_line_style` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_line_style` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Siehe auch

`set_line_style`, `disp_region`

Modul

System

get_line_width (: : WindowHandle : Width)

Abfragen der aktuellen Strichstärke für Konturen.

`get_line_width` gibt die Strichstärke der Regionenausgabe des Ausgabefensters an, die von Prozeduren wie `disp_region`, `disp_line`, `disp_polygon`, etc. verwendet wird. Gesetzt wird der Parameter mit der Prozedur `set_line_width`. Der Parameter ist nur von Bedeutung, wenn die Ränder von Objekten dargestellt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Width** (output_control) . . . . . integer ⇝ *integer*
  Aktuelle Strichstärke für Konturen.

Ergebnis

`get_line_width` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_line_width` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`set_line_width`, `set_line_style`, `disp_region`

Siehe auch

`set_line_width`, `disp_region`

Modul

System

get_paint (: : WindowHandle : Mode)

Abfragen des aktuellen Darstellungsmodus für Grauwerte.

`get_paint` gibt den Modus des Ausgabefensters für die Darstellung von Grauwerten, der von der Prozedur `disp_image` verwendet wird, an. Das Abfragen wird benutzt, wenn der aktuelle Modus kurzzeitig überschrieben werden soll. Hierzu wird zunächst der aktuelle Wert abgefragt, dann überschrieben und schließlich auf den alten Wert zurückgesetzt. Das Zurücksetzen erfolgt mit `set_paint`. Alle verfügbaren Modi können mit der Prozedur `query_paint` abgefragt werden. Der Parameter `Mode` enthält den Namen des Ausgabemodus. Falls ein Modus über Parameter genauer gesteuert werden kann, werden diese Werte (in einem Tupel) nach dem Namen übergeben. Die Reihenfolge der Werte ist die gleiche wie beim Setzen (`set_paint`).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (output_control) ..... string-array ⇝ *string* / integer
  Name und Parameter des aktuellen Ausgabemodus.

Ergebnis

`get_paint` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_paint` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`query_paint`

Mögliche Nachfolgerfunktionen

`set_paint`, `disp_region`, `disp_image`

Siehe auch

`set_paint`, `query_paint`, `disp_image`

Modul

System

get_part (: : WindowHandle : Row1, Column1, Row2, Column2)

Abfragen des Bildausschnitts.

`get_part` liefert den linken oberen und den rechten unteren Eckpunkt des Bildausschnitts, der im Ausgabefenster dargestellt wird. Dieser Bildausschnitt kann mittels `set_part` verändert werden (Als Voreinstellung wird immer das volle Bild ausgegeben).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Row1** (output_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilennummer des linken oberen Eckpunkts des Bildausschnitts.
- ▷ **Column1** (output_control) ..... rectangle.origin.x ⇝ *integer*
  Spaltennummer des linken oberen Eckpunkts des Bildausschnitts.
- ▷ **Row2** (output_control) ..... rectangle.corner.y ⇝ *integer*
  Zeilennummer des rechten unteren Eckpunkts des Bildausschnitts.
- ▷ **Column2** (output_control) ..... rectangle.corner.x ⇝ *integer*
  Spaltennummer des rechten unteren Eckpunkts des Bildausschnitts.

Ergebnis

`get_part` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_part` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`set_part`, `disp_region`, `disp_image`

Siehe auch

`set_part`, `disp_image`, `disp_region`, `disp_color`

Modul

System

get_part_style (: : WindowHandle : Style)

Abfrage der aktuellen Interpolationsmethode für die Graubildausgabe.

get_part_style fragt die Interpolationsmethode ab, die für die Ausgabe eines Bildausschnitts im Ausgabefenster verwendet wird. Interpoliert werden muß immer dann, wenn das Ausgabefenster größer als das Bildformat bzw. als der mittels set_part eingestellte Bildausschnitt für die Ausgabe ist. Derzeit werden drei Modi (Style) unterstützt:

0 keine Interpolation (eventuell niedrigere Qualität, sehr schnell).

1 ungewichtete Interpolation zwischen den Grauwerten (mittlere Qualität und Laufzeit).

2 gewichtete Interpolation zwischen den Grauwerten (höchste Qualität, langsam).

Die aktuelle Einstellung kann mittels set_part_style verändert werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Style** (output_control) ..... integer ⇝ *integer*
  Interpolationsmethode für die Bildschirmdarstellung: 0 (schnell,niedrige Qualität) bis 2 (langsam, höchste Qualität).
  Werteliste : Style ∈ {0, 1, 2}

Ergebnis

get_part_style liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_part_style wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_part_style, disp_region, disp_image

Siehe auch

set_part_style, set_part, disp_image, disp_color

Modul

System

get_pixel (: : WindowHandle : Pixel)

Abfragen des aktuellen Farbtabellenindex.

get_pixel gibt die Kodierung des Ausgabegrauwerts bzw. der Ausgabefarbe für das Ausgabefenster aus. Wird mit den Prozeduren set_color oder set_gray der Modus für die Ausgabe (disp_region, disp_ellipse, etc.) auf Farbe(n) bzw. Grauwert(e) gesetzt, dann wird der angegebene Werte in einen internen Code umgewandelt. Dieser Code wird dann (physikalisch) bei der Darstellung auf dem Bildschirm verwendet. Die Umsetzung ist von den Abbildungseigenschaften und dem Zustand des Ausgabegerätes abhängig und kann bei verschiedenen Programmläufen unterschiedlich ausfallen. Der Begriff „Pixel“ ist nicht mit dem des „Pixels“ in der Bildverarbeitung zu verwechseln (hierfür gibt es die Prozedur get_grayval). Ein Pixel ist hier i.a. der Index in der Farbtabelle.

get_pixel erlaubt es, die Ausgabeart zu sichern, ohne daß man wissen muß, ob Farbe(n) oder Grauwert(e) gesetzt wurden. Zurückgesetzt wird der Parameter mit der Prozedur set_pixel.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Pixel** (output_control) ..... string-array ⇝ *integer*
  Index in der aktuellen Farbtabelle.

Ergebnis

get_pixel liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_pixel wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_pixel, disp_region, disp_image

Siehe auch

set_pixel, set_fix

Modul

System

get_rgb (: : WindowHandle : Red, Green, Blue)

Abfragen der aktuellen Farbe in RGB-Kodierung.

get_rgb gibt die Ausgabefarben bzw. Grauwerte für das Ausgabefenster an. Diese sind durch die drei Farbanteile: rot, grün und blau beschrieben. get_rgb entspricht der Prozedur get_pixel, mit dem Unterschied, daß hier nicht die Indizes der Farbtabelle, sondern die Einträge in der Farbtabelle ausgegeben werden. Die Werte die man mit get_rgb erhält, können mit set_rgb wieder gesetzt werden.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- **Red** (output_control) ..... integer-array ⇝ *integer*
  Rotanteil der aktuellen Farbe.
- **Green** (output_control) ..... integer-array ⇝ *integer*
  Grünanteil der aktuellen Farbe.
- **Blue** (output_control) ..... integer-array ⇝ *integer*
  Blauanteil der aktuellen Farbe.

Ergebnis

get_rgb liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_rgb wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_rgb, disp_region, disp_image

Siehe auch

set_rgb

Modul

System

get_shape (: : WindowHandle : DisplayShape)

Abfragen der aktuellen Ausgabeform für Regionen.

get_shape gibt die Gestalt an, in der Regionen gezeichnet werden. Der erhaltene Wert kann mit set_shape gesetzt werden. Alle verfügbaren Darstellungsarten können mit query_shape abgefragt werden.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- **DisplayShape** (output_control) ..... string ⇝ *string*
  Ausgabeform für Regionen.

Ergebnis

get_shape liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_shape wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

query_shape

Mögliche Nachfolgerfunktionen

set_shape, disp_region

Siehe auch

set_shape, query_shape, disp_region

Modul

System

query_all_colors (: : WindowHandle : Colors)

Abfrage aller Farbnamen.

query_all_colors gibt die Namen aller Farben aus, die HALCON bekannt sind. Das heißt natürlich nicht, daß all diese Farben tatsächlich verwendet werden können. Vielmehr ist nur ein Teil dieser Farben verfügbar (siehe query_color). Allerdings kann mittels set_system vor dem Öffnen des ersten Fensters festgelegt werden, wie viele und welche der Farben verwendet werden sollen. Die HALCON Graphikfarben dienen zur Darstellung von Regionen (disp_region, disp_polygon, disp_circle, etc.). Sie können mit der Prozedur set_color gesetzt werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Colors** (output_control) ..... string-array ⇝ *string*
  Namen der Farben.

Beispiel

```
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]).
```

Ergebnis

query_all_colors liefert immer den Wert 2 (H_MSG_TRUE)

Parallelisierungsinformation

query_all_colors ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_system, set_color, disp_region

Siehe auch

query_color, set_system, set_color, disp_region, open_window, open_textwindow

Modul

System

query_color (: : WindowHandle : Colors)

Abfrage aller im Fenster darstellbaren Farbnamen.

query_color gibt für das Ausgabefenster die Namen aller Farben aus, die bei der Darstellung von Regionen (disp_region, disp_polygon, disp_circle, etc.) verwendet werden können. Diese können mit der

Prozedur set_color gesetzt werden. Bei einem S/W-Bildschirm liefert query_color die Werte 'black' und 'white'. Diese beiden „Farben" stehen bei jedem Bildschirm zur Verfügung. Bei Bildschirmen mit Graustufen werden auch einige Grautöne angegeben (z.B.: 'dim gray'). Bei Bildschirmen mit Farbtabellen wird eine Liste von darstellbaren Farben zurückgeliefert. Per Default beginnt das Tupel der Farben dabei mit S/W, gefolgt von den drei Grundfarben ('red','green','blue') und einigen Grautönen. Es besteht aber auch die Möglichkeit, vor dem Öffnen des ersten Fensters mittels set_system(::'graphic_colors',...:) die von HALCON zur Verfügung gestellten Graphikfarben selbst zusammenzustellen. Eine Liste aller dem System bekannten Farben erhält man dazu mit dem Aufruf query_all_colors(::WindowHandle:Colors). Genau diese Liste wird übrigens bei Bildschirmen mit Echtfarbenausgabe auch von query_color ausgegeben. Dies sind natürlich nicht alle Farben, die dargestellt werden können. Diese müßen dann mit set_rgb oder set_hsi direkt gewählt werden. Werden Farbnamen verwendet, die HALCON zwar bekannt sind, aber auf dem Rechner nicht darstellbar sind, dann wählt HALCON eine ähnliche Farbe aus (und erzeugt keine Fehlermeldung) falls set_check(::'~color':) aufgerufen wird.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Colors** (output_control) ..... string-array ⇝ *string*
  Namen der Farben.

Beispiel

```
open_window(0,0,-1,-1,'root','invisible',"",WindowHandle)
query_color(WindowHandle,Colors)
close_window(WindowHandle)
fwrite_string(['Displayable colors: ',Farben]).
```

Ergebnis

query_color liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt

Parallelisierungsinformation

query_color ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Nachfolgerfunktionen

set_color, disp_region

Siehe auch

query_all_colors, set_color, disp_region, open_window, open_textwindow

Modul

System

query_colored (: : : PossibleNumberOfColors)

Abfrage der Farbenzahl für bunte Ausgaben.

query_colored gibt alle zuläßigen Parameterwerte für set_colored aus. Bei set_colored wird festgelegt in wievielen Farben Regionen, bzw. Graphik ausgegeben wird.

Parameter

- ▷ **PossibleNumberOfColors** (output_control) ..... integer-array ⇝ *integer*
  Tupel der möglichen Farbenzahlen.

Beispiel

```
regiongrowing(Image,Seg,5,5,6,100)
query_colored(Colors)
set_colored(WindowHandle,Colors[1])
disp_region(Seg,WindowHandle).
```

Ergebnis

`query_colored` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`query_colored` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`set_colored`, `set_color`, `disp_region`

Alternativen

`query_color`

Siehe auch

`set_colored`, `set_color`

Modul

System

query_gray (: : WindowHandle : Grayval)

Abfrage der darstellbaren Graustufen.

`query_gray` gibt für das Ausgabefenster alle exakt reproduzierbaren Grauwerte aus, die bei der Darstellung von Grauwerten (`disp_image`) verwendet werden. Diese können bei der Prozedur `set_gray` gesetzt werden. Die Zahl der darstellbaren Graustufen läßt sich übrigens vor dem Öffnen des ersten Fensters auf der Ausgabemaschine mittels `set_system(::'num_gray_*',...:)` verändern.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Grayval** (output_control) ..... integer-array ⇝ *integer*
  Tupel der darstellbaren Graustufen.

Ergebnis

`query_gray` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt

Parallelisierungsinformation

`query_gray` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`set_gray`, `disp_region`

Siehe auch

`set_gray`, `disp_image`

Modul

System

query_insert (: : WindowHandle : Mode)

Abfragen der Graphikmodi.

`query_insert` gibt an, in welcher Art die Pixelwerte in das Ausgabefenster eingetragen werden können. Eine Möglichkeit ist z.B., daß die neuen Pixel die vorhandenen Werte überschreiben. Im allgemeinen wird ein funktionaler Zusammenhang aus vorhandenen Werten und neuen Werten abgeleitet.

Mögliche Darstellungsfunktionen:

’copy’: vorhandene Pixel überschreiben

’xor’: trage ein: vorhandene Pixel „xor“ neue Pixel

’not’: komplementiere bisher dargestellte Pixel

Immer vorhanden ist die Funktion ’copy’.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (output_control) . . . . . string-array ⇝ *string*
  Name der Darstellungsprozedur.

Ergebnis

query_insert liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

query_insert ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

set_insert, disp_region

Siehe auch

set_insert, get_insert

Modul

System

query_line_width (: : : Min, Max)

Abfragen der möglichen Strichstärken.

query_line_width gibt an, wie dick (in Pixel) der Rand einer Region mindestens (Min) und höchstens (Max) dargestellt werden kann. Das Setzen der Randstärke erfolgt mit set_line_width und wird von Operationen wie disp_region, disp_line, disp_circle, disp_rectangle1, disp_rectangle2 etc. verwendet, falls der Zeichenmodus auf „Rand“ (set_draw(::WindowHandle,’margin’:)) eingestellt ist.

Parameter

- ▷ **Min** (output_control) . . . . . integer ⇝ *integer*
  Darstellbare Mindeststärke.
- ▷ **Max** (output_control) . . . . . integer ⇝ *integer*
  Darstellbare Maximalstärke.

Ergebnis

query_line_width liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

query_line_width ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

get_line_width, set_line_width, set_line_style, disp_line

Siehe auch

disp_circle, disp_line, disp_rectangle1, disp_rectangle2, disp_region, set_line_width, get_line_width, set_line_style

Modul

System

query_paint (: : WindowHandle : Mode)

Abfragen der Darstellungsmöglichkeiten für Grauwerte.

query_paint gibt für das Ausgabefenster die Namen aller Grauwertdarstellungsarten (z.B. ’gray’, ’3D-plot’, ’contourline’, etc.) aus. Diese können von der Prozedur set_paint verwendet, d.h. aktiviert werden.

query_paint gibt nur die Namen der Darstellungsarten, nicht aber die zusätzlichen Parameter (die bei einigen Arten festgelegt werden können) an.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (output_control) ..... string-array ⇝ *string*
  Namen der Darstellungsmodi.

Ergebnis

query_paint liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt

Parallelisierungsinformation

query_paint ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

get_paint, set_paint, disp_image

Siehe auch

set_paint, get_paint, disp_image

Modul

System

query_shape (: : : DisplayShape)

Abfrage der Darstellungsmodi für Regionen.

query_shape gibt für das Ausgabefenster die Namen aller Regionendarstellungsarten (z.B. ’original’, ’circle’, ’rectangle1’, ’rectangle2’, ’ellipse’, etc.) aus. Diese können von der Prozedur set_shape verwendet (d.h. aktiviert) werden.

Parameter

- ▷ **DisplayShape** (output_control) ..... string-array ⇝ *string*
  Namen der Darstellungsmodi für Regionen.

Ergebnis

query_shape liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt

Parallelisierungsinformation

query_shape ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

get_shape, set_shape, disp_region

Siehe auch

set_shape, get_shape, disp_region

Modul

System

set_color (: : WindowHandle, Color :)

Setzen der Ausgabefarbe(n).

set_color setzt für das Ausgabefenster die Farbe(n), in der die Regionen dargestellt werden sollen. Ein Tupel aller verfügbaren Farben kann mit der Prozedur query_color abgefragt werden. Die „Farben“ ’black’ und ’white’ sind für alle Bildschirme verfügbar. Werden Farben verwendet, die auf dem Bildschirm nicht darstellbar sind, so kann mit dem Aufruf von set_check(::’~color’:) erreicht werden, daß HALCON eine Farbe für die Ausgabe verwendet, die der angegebenen möglichst ähnlich ist. Außerdem besteht die Möglichkeit, vor

dem Öffnen des ersten Fensters mittels set_system(::'graphic_colors',...:) die von HALCON zur Verfügung gestellten Graphikfarben selbst festzulegen. Wird eine einzige Farbe angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Regionen und geometrischen Objekten modulo dieser Farben. Im Beispiel (s.u.) wird der erste Kreis in rot, der zweite in grün und der dritte wieder in rot ausgegeben. Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, daß es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen: Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen.

Die gesetzte(n) Farbe(n) werden so lange verwendet, bis erneut set_color, set_pixel, set_rgb, set_hsi oder set_gray aufgerufen wird.

Verwendet wird dieser Darstellungsparameter bei Prozeduren wie disp_region, disp_line, disp_rectangle1, disp_arrow etc. Die Farbe wird aber auch bei der graphischen Darstellung von Grauwerten verwendet (z.B.: '3D-plot','histogram', 'contourline', etc.; siehe hierzu set_paint).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Color** (input_control) ..... string(-array) ⇝ *string*
  Namen der Ausgabefarben.
  Defaultwert : 'white'

Beispiel

```
set_color(WindowHandle,['red','green'])
disp_circle(WindowHandle,[100,200,300],[200,300,100],[100,100,100]).
```

Ergebnis

set_color liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und für dieses die angegebene Farbe(n) darstellbar sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_color ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

query_color

Mögliche Nachfolgerfunktionen

disp_region

Alternativen

set_rgb, set_hsi

Siehe auch

get_rgb, disp_region, set_fix, set_paint

Modul

System

set_colored (: : WindowHandle, NumberOfColors :)

Setzen einer bunten Ausgabe.

set_colored ist eine Kurzform für bestimmte set_color Aufrufe und dient z.B. dazu, eine Menge von Regionen in unterschiedlichen Farben darzustellen. NumberOfColors bestimmt die Anzahl von Farben, die dazu alternierend verwendet werden sollen. Zulässige Werte von NumberOfColors können mittels query_colored abgefragt werden. Außerdem besteht die Möglichkeit, vor dem Öffnen des ersten Fensters mittels set_system (::'graphic_colors',...:) die von HALCON zur Verfügung gestellten Graphikfarben selbst festzulegen.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **NumberOfColors** (input_control) ..... integer ⇝ *integer*
  Anzahl der Ausgabefarben.
  Defaultwert : 12
  Werteliste : NumberOfColors ∈ {3, 6, 12}

Ergebnis

set_colored liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_colored ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

query_colored, set_color

Mögliche Nachfolgerfunktionen

disp_region

Siehe auch

query_colored, set_color, disp_region

Modul

System

set_comprise (: : WindowHandle, Mode :)

Setzen der Behandlung der Bildmatrix bei der Ausgabe.

set_comprise gibt an, ob bei der Ausgabe von Grauwerten (disp_image) nur solche Grauwerte die zu dem auszugebenden Objekt gehören (’object’) oder die gesamte Bildkomponente (’image’) ausgegeben werden soll. Voreinstellung ist ’object’.

Achtung

Falls der Modus ’image’ gewählt wurde, werden bei der Ausgabe undefinierte Grauwertbereiche dargestellt werden. Diese sind je nach Kontext schwarz oder mit beliebigen Speicherinhalten besetzt. Siehe hierzu das Beispiel.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für die Ausgabe von Grauwerten.
  Defaultwert : ’object’
  Werteliste : Mode ∈ {’image’, ’object’}

Beispiel

```
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().
```

Ergebnis

`set_comprise` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_comprise` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_comprise`

Mögliche Nachfolgerfunktionen

`disp_image`

Siehe auch

`get_comprise`, `disp_image`, `disp_color`

Modul

System

set_draw (: : WindowHandle, Mode :)

Ausfüllmodus für Regionen setzen.

`set_draw` gibt an, ob Regionen ausgefüllt (’fill’) oder nur der Rand (’margin’) dargestellt werden soll. Der angegebene Wert wird dem Fenster mit der logischen Fensternummer `WindowHandle` zugeordnet. Die Ausgabe erfolgt mit Prozeduren wie `disp_region`, `disp_circle`, `disp_rectangle1`, `disp_rectangle2`, `disp_arrow` etc. Der Ausfüllmodus ist auch bei der graphischen Darstellung von Grauwerten relevant (z.B. `set_paint(::WindowHandle,'histogram':)`). Falls der Modus auf ’margin’ gesetzt ist, kann der Rand noch mit `set_line_width`, `set_line_approx` und `set_line_style` beeinflußt werden.

Achtung

Falls der Modus ’margin’ gewählt wurde und die Strichstärke größer als eins ist, werden manche Hohlflächen nicht ausgegeben.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Ausfüllmodus für Regionen.
  Defaultwert : ’fill’
  Werteliste : `Mode` ∈ {’fill’, ’margin’}

Ergebnis

`set_draw` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_draw` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_draw`

Mögliche Nachfolgerfunktionen

`disp_region`

Siehe auch

`get_draw`, `disp_region`, `set_paint`, `disp_image`, `set_line_width`, `set_line_style`

Modul

System

set_fix (: : WindowHandle, Mode :)

Setzen der Farbtabellenfixierung.

Verhalten bei Mode = **'true'**: set_fix fixiert das Pixel, das zuletzt durch eine der Funktionen set_pixel, set_gray, set_color, set_hsi oder set_rgb ermittelt wurde. (Bemerkung: Ein Pixel ist hier der Index in die aktuelle Farbtabelle). Fixierten Pixeln wird durch Setzen einer Farbe oder eines Grauwertes (set_color, set_rgb, set_hsi oder set_gray) die entsprechende neue Farbe zugewiesen (d.h überschrieben). Hierdurch ist es möglich an einer Stelle in der Farbtabelle jede Farbe (set_color), jeden Grauwert (set_gray) und jede Farbkombination (set_rgb, set_hsi) zu erzeugen.

Wenn Mode den Wert 'false' hat, wird die Fixierung zurückgenommen. Durch fortgesetzte Anwendung von set_pixel, set_fix(::WindowHandle,'true':), set_rgb und set_fix (::WindowHandle,'false':) kann eine Farbtabelle modifiziert oder erzeugt werden.

Achtung

Es ist zu beachten, daß bei der Verwendung von set_fix auch die Farben von „nicht HALCON Fenster“ verändert werden können.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fensteridentifikator.
- **Mode** (input_control) ..... string ⇝ *string*
  Modus für Fixierung.
  Defaultwert : 'true'
  Werteliste : Mode ∈ {'true', 'false'}

Ergebnis

Wenn das Fenster gültig ist, die Hardware über eine Farbtabelle verfügt und der Parameter richtig besetzt ist, liefert set_fix 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung ausgeführt.

Parallelisierungsinformation

set_fix ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

get_fix

Mögliche Nachfolgerfunktionen

set_pixel, set_rgb

Siehe auch

get_fix, set_pixel, set_rgb, set_color, set_hsi, set_gray

Modul

System

set_gray (: : WindowHandle, GrayValues :)

Setzen eines Grauwertes für die Regionenausgabe.

set_gray gibt an, daß Regionen in bestimmten Grauwerten (GrayValues) dargestellt werden sollen. Unter einem Grauwert wird hier der Bereich der Farbtabelle verstanden, der für die Darstellung von Grauwertdaten (disp_image mit set_paint(::WindowHandle,'gray':)) verwendet wird. Hierbei handelt es sich um die Einträge der Farbtabelle, die von set_lut modifiziert werden. Unter einem „Grauwert“ ist die Farbe zu verstehen, in der ein Bildpunkt mit dem gleichen Wert dargestellt wird (was nicht notwendigerweise ein Grauton ist). Weiterhin ist zu beachten, daß sich die Farbe der Grafik, deren Aussehen mit set_gray festgelegt wurde, beim Laden einer neuen Farbtabelle i.A. ändern wird.

Wird ein Grauton als Grafikfarbe benötigt (d.h. keine Farbänderung durch set_lut), so kann dieser z.B. mit set_color(::WindowHandle,'gray':) gesetzt werden.

Wird ein einziger Grauwert angegeben, dann erfolgen alle Ausgaben in diesem Grauwert. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung der Tupel von Regionen und geometrischen Objekten modulo

dieser Grauwerte. Der erste Kreis im Beispiel (s.u.) wird in dem Grauwert 100, der zweite in dem Grauwert 200 und der dritte wieder in dem Grauwert 100 ausgegeben. Bei jedem Aufruf einer Ausgabeprozedur wird mit dem ersten Grauwert begonnen. Es ist dabei zu beachten, daß es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen: Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in dem ersten Grauwert. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen.

Der Aufruf der Prozeduren (`set_gray`, `set_color`, `set_rgb`, `set_hsi`) überschreibt die Werte, die vorher gesetzt wurden. Je nach Geräteeigenschaft können nicht alle Grauwerte dargestellt werden. Der Wertebereich des Parameters `GrayValues` (0..255) wird entsprechend auf die darstellbaren Graustufen abgebildet. In jedem Fall wird der Grauwert 0 als schwarz und der Grauwert 255 als weiß dargestellt. Mit der Prozedur `query_gray` können die darstellbaren Grauwerte abgefragt werden. Außerdem besteht die Möglichkeit, vor dem Öffnen des ersten Fensters auf der Ausgabemaschine mittels `set_system(::'num_gray_*',...:)` die Zahl der tatsächlich dargestellten Graustufen zu verändern. Mit dem Aufruf: `set_check(::'~color':)` wird HALCON aufgefordert keine Fehlermeldung zu erzeugen, falls eine Grauwert verwendet wurde, der auf dem Bildschirm nicht darstellbar ist, sondern eine Näherung für diesen Wert zu suchen.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **`GrayValues`** (input_control) ..... integer(-array) ⇝ *integer*
  Grauwerte für Regionendarstellung.
  Defaultwert : 255
  Wertevorschläge : `GrayValues` ∈ {0, 1, 2, 10, 16, 32, 64, 100, 120, 128, 250, 251, 252, 253, 254, 255}
  Typischer Wertebereich : $0 \leq$ `GrayValues` ≤ 255

Beispiel

```
set_gray(WindowHandle,[100,200])
disp_circle(WindowHandle,[100,200,300],[200,300,100],[100,100,100]).
```

Ergebnis

`set_gray` returns 2 (H_MSG_TRUE) if `GrayValues` is displayable and the window is valid. Otherwise an exception handling is raised.

Parallelisierungsinformation

`set_gray` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`disp_region`

Siehe auch

`get_pixel`, `set_color`

Modul

System

`set_hsi` (: : WindowHandle, Hue, Saturation, Intensity :)

Festlegen der Ausgabefarbe in der HSI-Kodierung.

`set_hsi` setzt für das Ausgabefenster die Farbe(n), bzw. die Grauwerte, in der Regionen dargestellt werden sollen. Die gewünschten Werte werden durch `Hue` (Farbwert), `Saturation` (Sättigung) und `Intensity` (Intensität) festgelegt. Die Umrechnung von HSI nach RGB erfolgt mit der Formel:

$$
\begin{aligned}
H &= (2\pi\texttt{Hue})/255 \\
I &= (\sqrt{6}\texttt{Intensity})/255 \\
M1 &= (\sin(H)\texttt{Saturation})/(255\sqrt{6}) \\
M2 &= (\cos(H)\texttt{Saturation})/(255\sqrt{2})
\end{aligned}
$$

$$
\begin{aligned}
R &= (2M1 \quad\quad + I)/(4\sqrt{6}) \\
G &= (-M1 + M2 + I)/(4\sqrt{6} \\
B &= (-M1 - M2 + I)/(4\sqrt{6})
\end{aligned}
$$

$$\begin{aligned} Red &= R * 255 \\ Green &= G * 255 \\ Blue &= B * 255 \end{aligned}$$

Wird nur eine Kombination angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Region und geometrischen Objekten modulo dieser Farben. Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, daß es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen: Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte die aus mehreren Zusammenhangskomponenten bestehen.

Die gesetzte Farbe(n) werden so lange verwendet, bis erneut `set_color`, `set_pixel`, `set_rgb` oder `set_gray` aufgerufen wird. Verwendet wird dieser Darstellungsparameter bei Prozeduren wie `disp_region`, `disp_line`, `disp_rectangle1`, `disp_rectangle2`, `disp_arrow`, etc. Die Farbe wird aber auch bei der graphischen Darstellung von Grauwerten verwendet (z.B.: '3D-plot','histogram', 'contourline', etc.; siehe hierzu `set_paint`).

Achtung

Nicht für jeden Farbwert kann die gewünschte Intensität erzeugt werden. In diesem Fall wird die Intensität automatisch entsprechend erniedrigt.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **`Hue`** (input_control) ..... integer-array ⇝ *integer*
  Farbwert der aktuellen Zeichenfarbe.
  Defaultwert : 30
  Typischer Wertebereich : $0 \leq$ `Hue` ≤ 255
  Restriktion : $($`0` $\leq$ `Hue`$) \wedge ($`Hue` $\leq$ `255`$)$
- ▷ **`Saturation`** (input_control) ..... integer-array ⇝ *integer*
  Sättigung der aktuellen Zeichenfarbe.
  Defaultwert : 255
  Typischer Wertebereich : $0 \leq$ `Saturation` ≤ 255
  Restriktion : $($`0` $\leq$ `Saturation`$) \wedge ($`Saturation` $\leq$ `255`$)$
- ▷ **`Intensity`** (input_control) ..... integer-array ⇝ *integer*
  Intensität der aktuellen Zeichenfarbe.
  Defaultwert : 84
  Typischer Wertebereich : $0 \leq$ `Intensity` ≤ 255
  Restriktion : $($`0` $\leq$ `Intensity`$) \wedge ($`Intensity` $\leq$ `255`$)$

Ergebnis

`set_hsi` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und für dieses die angegebene Farbe(n) darstellbar sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_hsi` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`get_hsi`

Mögliche Nachfolgerfunktionen

`disp_region`

Siehe auch

`get_hsi`, `get_pixel`, `trans_from_rgb`, `trans_to_rgb`, `disp_region`

Modul

System

`set_icon ( Icon : : WindowHandle : )`

Setzen eines Icons für die Regionenausgabe.

set_icon setzt ein Icon, das bei der Ausgabe von Regionen (disp_region) im Schwerpunkt der Region dargestellt werden soll. Die Verwendung dieses Icons wird durch set_shape aktiviert.

Parameter

- ▷ **Icon** (input_object) ..... region ⇝ *Hobject*
  Icon für Schwerpunkt.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.

Beispiel

```
/* draw a region and an icon */
draw_region(&Region,WindowHandle) ;
draw_region(&Icon,WindowHandle) ;
set_icon(Icon) ;
set_shape(WindowHandle,"icon") ;
disp_region(Region,WindowHandle) ;
```

Ergebnis

set_icon returns 2 (H_MSG_TRUE) if exactly one region is passed. Otherwise an exception handling is raised.

Parallelisierungsinformation

set_icon wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region

Mögliche Nachfolgerfunktionen

set_shape, disp_region

Modul

System

set_insert (: : WindowHandle, Mode :)

Festlegen der Darstellungsfunktion in den Bildwiederholspeicher.

set_insert gibt an, in welcher Art die Pixelwerte in das Ausgabefenster (Bildwiederholspeicher) eingetragen werden. Möglich ist etwa, daß die neuen Pixel die vorhandenen Werte überschreiben. Im allgemeinen wird ein funktionaler Zusammenhang aus vorhandenen Werten und neuen Werten abgeleitet. Der angegebene Wert wird dem Fenster zugeordnet, das zum Zeitpunkt des Aufrufs gültig ist. Die Ausgabe der Objekte erfolgt mit Prozeduren wie disp_region, disp_polygon, disp_circle.

Mögliche Darstellungsfunktionen:

’copy’: vorhandenen Wert mit Pixelwert überschreiben

’xor’: trage ein: vorhandener Wert „xor“ Pixelwert

’not’: komplementiere bisher dargestellten Wert

Abhängig von der jeweiligen Implementierung sind nicht alle Funktionen verfügbar (siehe query_insert). Immer vorhanden ist jedoch die Funktion ’copy’.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Name der Darstellungsfunktion.
  Defaultwert : ’copy’
  Werteliste : Mode ∈ {’copy’, ’xor’, ’not’}

Ergebnis

`set_insert` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_insert` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`query_insert`, `get_insert`

Mögliche Nachfolgerfunktionen

`disp_region`

Siehe auch

`get_insert`, `query_insert`

Modul

System

`set_line_approx ( : : WindowHandle, Approximation : )`

Festlegen der Glättung für die Konturdarstellung.

`set_line_approx` manipuliert die Ausgabe von Regionenrändern (Konturen). Werte größer Null für `Approximation` bewirken eine Polygonapproximation ≈ Glättung (mit maximaler Abweichung des Polygons von der Kontur von `Approximation` Punkten). Die Approximation erfolgt mittels des Verfahrens, das auch in `get_region_polygon` verwendet wird. `set_line_approx` ist dann wichtig, wenn `set_line_style` zur Ausgabe von Konturmustern verwendet wird.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **`Approximation`** (input_control) ..... integer ⇝ *integer*
  Maximale Abweichung von der Originalkontur.
  Defaultwert : 0
  Typischer Wertebereich : 0 ≤ `Approximation`
  Restriktion : `Approximation` ≥ `0`

Beispiel

```
/* 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).
```

Ergebnis

`set_line_approx` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_line_approx` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_line_approx`

Mögliche Nachfolgerfunktionen

`disp_region`

Alternativen

`get_region_polygon`, `disp_polygon`

Siehe auch

get_line_approx, set_line_style, set_draw, disp_region

Modul

System

set_line_style (: : WindowHandle, Style :)

Festlegen des Musters für die Konturdarstellung.

set_line_style gibt an, wie der Rand einer Region dargestellt werden soll. Diese Information wird von Prozeduren wie disp_region, disp_line, disp_polygon etc. benötigt. Der aktuell eingestellte Wert kann mit der Prozedur get_line_style abgefragt werden. Konkret enthält Style (derzeit maximal fünf) Zahlenpaare. Der erste Wert ist jeweils die Länge des sichtbaren Konturstücks, der zweite die Länge des darauf folgenden Zwischenraums. Die Zahlenpaare werden für die Ausgabe einer Kontur zyklisch herangezogen.

Achtung

set_line_style führt automatisch eine leichte Polygonapproximation durch (entsprechend set_line_approx(::WindowHandle,3:)). Dieser Wert kann mit set_line_approx nur vergrößert werden.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- **Style** (input_control) ..... integer-array ⇝ *integer*
  Muster der Kontur.
  Defaultwert : '[]'

Beispiel

```
* 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,[])
```

Ergebnis

set_line_style liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_line_style ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

get_line_style

Mögliche Nachfolgerfunktionen

disp_region

Siehe auch

get_line_style, set_line_approx, disp_region

Modul

System

set_line_width (: : WindowHandle, Width :)

Festlegen der Strichstärke für die Konturdarstellung.

set_line_width gibt an, wie dick (in Pixel) der Rand einer Region dargestellt werden soll. Die Darstellung erfolgt mit Prozeduren wie disp_region, disp_line, disp_polygon, etc. Mit der Prozedur get_line_width kann der Ausgabeparameter für das Ausgabefenster abgefragt werden. Nicht bei allen Geräten besteht die Möglichkeit die Randstärke zu verändern. Dies kann mit der Prozedur query_line_width abgefragt werden.

Achtung

Die Strichstärke ist nur von Bedeutung, falls der Modus set_draw(::WindowHandle,'margin':) eingestellt wurde. Falls die Strichstärke größer als 1 ist, können Hohlflächen von Regionen nicht immer korrekt ausgegeben werden.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Width** (input_control) ..... integer ⇝ *integer*
  Strichstärke für Konturdarstellung.
  Defaultwert : 1
  Restriktion : (Width ≥ 1) ∧ (Width ≤ 2000)

Ergebnis

set_line_width liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_line_width wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

query_line_width, get_line_width

Mögliche Nachfolgerfunktionen

disp_region

Siehe auch

get_line_width, query_line_width, set_draw, disp_region

Modul

System

set_paint (: : WindowHandle, Mode :)

Modus für die Darstellung von Grauwerten festlegen.

set_paint setzt den Modus des Ausgabefensters für die Darstellung von Grauwerten (ein- oder mehrkanalig), der von der Prozedur disp_image verwendet wird.

Im Folgenden werden die verschiedenen Arten, in denen Grauwertdaten ausgegeben werden können, dargestellt. Vorab sei bemerkt, daß der Modus 'default' in 99% der Anwendungen die richtige Wahl ist (und deshalb auch als Voreinstellung verwendet wird).

Die Darstellung von Grauwerten wird von den Eigenschaften der Hardware bestimmt. So erfolgt auf einem Bildschirm mit einer Bildebene bis sieben Bildebenen nur eine Binärkodierung der Grauwertdaten. Stehen acht oder mehr Bildebenen zur Verfügung, so wird eine gute Auflösung erzielt. Für binäre Darstellung stehen Techniken mit Dither-Matrix (schnell aber schlechte Auflösung), minimalem Fehler (gut aber langsam) und Schwellenwertbildung zur Verfügung. Bei der Schwellenwertoperation kann der Schwellenwert als 2. Parameterwert übergeben werden (insgesamt wird also ein Tupel aus der Zeichenreihe 'threshold' und dem Schwellenwert, z.B. 100 verwendet: ['threshold',100]).

Bei acht Bildebenen werden ca. 200 Graustufen für die Darstellung verwendet. Es ist hier aber auch möglich, binäre Darstellungen zu verwenden.

Eine weitere Art der Grauwertdarstellung ist das Histogramm ('histogram'). Die zusätzlichen Parameterwerte Row (2. Parameterwert) und Column (3. Parameterwert) geben die Zeile bzw. Spalte des Mittelpunktes des Histogramms (für die Positionierung) an. Der Skalierungsfaktor Scale (4. Parameterwert) gibt die Größe des Histogramms an. Dabei bedeutet 1, daß 256 Werte unterschieden werden; bei 2 werden 128 Werte unterschieden, bei 3 werden 64 Werte unterschieden, usw. Die Übergabe der 4 Werte erfolgt mit dem einen Parameter Mode, indem

die Werte als ein Tupel verwendet werden; z.B.: ['histogram', 256,256,1]. Wird nur der erste Wert übergeben ('histogram'), dann wird für die Positionierung die Voreinstellung, bzw. der letzte gesetzte Wert verwendet. Für die Berechnung des Histogramms siehe auch `gray_histo`. Für die Darstellung von Histogrammen gelten die gleichen Parameter wie für Prozeduren der Art `disp_region` etc. Dies sind z.B.: `set_color`, `set_draw`, etc.

Weiterhin ist die Darstellung der relativen Häufigkeit der Anzahl der Zusammenhangskomponenten (der Schwellenwertoperation) möglich ('component_histogram'). Für die Berechnung des Komponentenhistogramms siehe `shape_histo_all`. Die Art der Positionierung und der Darstellung entspricht dem Modus 'histogram'.

Bei der Verwendung von 'mean' werden die Regionen der Objekte mit dem mittleren Grauwert gezeichnet.

Die Parameterwerte 'row' und 'column' ermöglichen es, Zeilen bzw. Spalten eines Grauwertbildes darzustellen. Die Position (Zeilen- Spaltenindex) wird mit dem 2. Parameterwert angegeben. Mit dem 3. Parameterwert wird der Skalierungsfaktor angegeben. Dieser wird in Prozentwerten festgelegt. Dabei entspricht 100 der Auflösung von einem Pixel pro Graustufe. 50 entspricht z.B. 0.5 Pixel pro Graustufe.

Graubilder können auch als Gebirge aufgefaßt werden. Hierzu steht ein Höhenplot ('contourline') und eine pseudo-3D-Darstellung ('3D-plot' und '3D-plot_hidden') zur Verfügung.

Zweikanalige Bilder können als Vektorfelder ausgegeben werden ('vectorfield').

Dreikanalige Bilder werden als RGB-Daten interpretiert. Es stehen hierfür drei verschiedene Darstellungsmodi zur Verfügung, wobei zwei optional mit dem Floyd-Steinberg Verfahren optimiert werden können.

Alle verfügbaren Modi können mit der Prozedur `query_paint` abgefragt werden.

Bei den Modi, die mehr als einen Wert benötigen sind folgende Techniken möglich:

- Es wird nur den Name des Modus angegeben. Hier werden die Voreinstellungen oder die zuletzt gesetzten Werte verwendet. Beispiel: `set_paint(::WindowHandle,'contourline':)`
- Es werden alle Werte übergeben. Hierdurch ist es möglich alle Eigenschaften der Ausgabe festzulegen. Beispiel: `set_paint(::WindowHandle,['contourline',10,1]:)`
- Nur die ersten n Werte werden angeben. Hierbei werden die nicht angegebenen Werte nicht verändert. Beispiel: `set_paint(::WindowHandle,['contourline',10]:)`
- Einige der Werte werden durch einen Stern ersetzt ('*'). Dies bedeutet, daß dieser Wert nicht verändert werden soll. Beispiel: `set_paint(::WindowHandle,['contourline','*',1]:)`

Ist der Modus 'default' gewählt, dann muß für die Ausgabe von 2- und 3-kanaligen Bildern kein Aufruf von `set_paint` erfolgen. Es wird dann automatisch ein geeignetes Verfahren aufgerufen.

Die Ausgabe der Grauwerte wird auch von anderen Prozeduren (und deren Parametern) beeinflußt. Dies sind insbesondere `set_part`, `set_part_style`, `set_lut` und `set_lut_style`. Bei einigen Ausgabemodi von `set_paint` werden die Grauwerte graphisch dargestellt (z.B. 'histogram','contourline','3D-plot', etc.). Dies hat zur Folge, daß auch Parameter, die mit `set_color`, `set_rgb`, `set_hsi`, `set_pixel`, `set_shape`, `set_line_width` und `set_insert` bestimmt wurden, Einfluß auf die Darstellung haben. Dies kann z.B. bei `set_shape(::'convex':)` und `set_paint(::WindowHandle,'histogram':)` zu unerwarteten Ergebnissen führen. Es wird nämlich die konvexe Hülle des Histogramms ausgegeben. Sollten also unerwartete Ergebnisse erscheinen, so sind die Werte der Darstellungsparameter für Regionen zu überprüfen.

Modi:

- Einkanalige Bilder

  'default' optimale Grauwertdarstellung bei gegebener Hardware ('gray' oder 'floyd_steinberg') (Voreinstellung)

  'gray' Graubild

  'mean' Mittlerer Grauwert

  'dither4_1' Binärbild, Dithermatrix 4x4

  'dither4_2' Binärbild, Dithermatrix 4x4

  'dither4_3' Binärbild, Dithermatrix 4x4

  'dither8_1' Binärbild, Dithermatrix 8x8

  'floyd_steinberg' Binärbild, beste Grauwertsimulation

  ['threshold',Threshold]

  'threshold' Binärbild, Schwellenwert: Voreinstellung (128)

['threshold',200] Binärbild, Schwellenwert: beliebig (hier 200)

['histogram',Line,Column,Scale]

'histogram' Darstellung der Grauwerte als Histogramm
Voreinstellung der Position: max. Größe, in der Mitte des Fensters.

['histogram',256,256,2] Darstellung der Grauwerte als Histogramm
Positionierung: Mittelpunkt bei Bildformat 512 (256,256)
Größe: 2 (d.i. Hälfte der Größe)

['component_histogram',Line,Column,Scale]

'component_histogram' Darstellung aller Zusammenhangskomponenten als Histogramm
Positionierung: Voreinstellung

['component_histogram',256,256,1] Darstellung aller Zusammenhangskomponenten als Histogramm
Positionierung: Mittelpunkt bei Bildformat 512 (256,256)
Größe: 1 (maximale Größe)

['row',Line,Scale]

'row' Darstellung des Grauwertprofils entlang einer Bildzeile.
Zeile: Bildmitte (Voreinstellung)
Skalierung: 50

['row',100,20] Darstellung des Grauwertprofils der Zeile 100, wobei die Skalierung der Grauwert 0.2 (20%) beträgt.

['column',Column,Scale]

'column' Darstellung des Grauwertprofils entlang einer Bildspalte.
Zeile: Bildmitte (Voreinstellung)
Skalierung: 50

['column',100,20] Darstellung des Grauwertprofils der Spalte 100, wobei die Skalierung der Grauwert 0.2 (20%) beträgt.

['contourline',Step,Colored]

'contourline' Die Grauwerte werden in Form eines Höhenplots dargestellt: Die Höhenunterschiede können mit dem Parameter Step festgelegt werden. Voreingestellt ist eine Schrittweite von 30 Graustufe, das ergibt also maximal 8 Höhenstufen (bei 256 Graustufen).
Es kann noch festgelegt werden, ob die Linien in der voreingestellten Farbe (z.B. `set_color`) oder in ihrem Grauwert gezeichnet werden sollen.
Dies geschieht durch den Paramater Colored (0 = einfarbig / 1 = Graustufen). Als Voreinstellung wird die Farbe für Regionenausgabe verwendet.

['contourline',15,1] Höhenplot mit der Schrittweite von 15 und der Darstellung durch Grauwerte.

['3D-plot', Step, Colored, EyeHeight, EyeDistance, ScaleGray, LinePos, ColumnPos]

'3D-plot' Die Grauwerte werden als 3-dimensionales Gebirge interpretiert. Gezeichnet werden Linien entlang der X- und Y-Achse des Bildes mit dem Abstand Step (2. Parameterwert). Der 3. Wert (Colored) legt fest, ob in der Voreingestellten Farbe, oder in dem jeweiligen Grauwert gezeichnet werden soll. Die Projektion des 3-D Bildes kann durch die Parameter EyeHeight und EyeDistance beeinflußt werden. Der Wertebereich dieser Parameter beträgt 1..255. Mit ScaleGray werden Grauwerte bei dunklen Bildern überhöht (100 = keine Veränderung). Da bei extremen Werte von EyeHeight und EyeDistance das Bild etwas verschoben wird, kann mit LinePos und ColumnPos die Graphik entsprechend verschoben werden. Der Wertebereich dieser Parameter ist -127..127.

['3D-plot', 5, 1, 110, 160, 150, 70, -10] Linienabstand: 5 Punkte
Farbig: ja (1)
Augenhöhe: 110
Augenentfernung: 160
Grauwertskalierung: 1.5 (150)
Verschiebung Zeile: 70 Punkte nach unten
Verschiebung Spalte: 10 Punkte nach rechts

['3D-plot_hidden', Step, Colored, EyeHeight, EyeDistance, ScaleGray, LinePos, ColumnPos]

'3D-plot_hidden' wie '3D-plot' aber mit Verdeckung.

- Zweikanalige Bilder

'default' Ausgabe als Verschiebungsvektorfeld.

['vectorfield', Step, MinLengh, ScaleLength]

'vectorfield' Ausgabe von Verschiebungsvektorfeldern. Prozeduren wie `optical_flow_match` erzeugen zwei Bilder, wobei das eine die x-Verschiebung das andere die y-Verschiebung der Bildpunkte enthält. Bei diesem Modus wird für jeden Vektor ein Pfeil gezeichnet. Die Schrittweite (Parameterwert: Step) ist entsprechend zu der Schrittweite bei `optical_flow_match` zu wählen. Zu kurze Vektoren können mit dem Dritten Parameterwert (MinLengh) unterdrückt werden. Der vierte Parameterwert (ScaleLength) skaliert die Länge der Vektoren. Vektoren, die das Ausgabefenster verlassen, werden nicht gezeichnet.

['vectorfield',16,2,3] Ausgabe jedes 16. Vektors der eine Länge von mindestens 2 Punkten hat. Jeder Vektor wird für die Darstellung um den Faktor 3 verlängert.

- Dreikanalige Bilder

'default' Ausgabe als RGB-Bild mit 'median_cut'.

'television' Farbadditionsalgorithmus für RGB-Bilder (d.h. 3 RGB-Komponenten bei `disp_image` nötig)
Schnelles Verfahren, beim dem alle Farbbilder die gleiche Farbtabelle verwenden.
Nachteil: Verringerung der Auflösung und nur bei guten (hellen) Bildschirmen einsetzbar.

'grid_scan' Grid-Scan-Algorithmus für RGB-Bilder (d.h. 3 RGB- Komponenten bei `disp_image` nötig)
Mittelschnelles Verfahren, das eine, für das Farbbild optimierte, Tabelle erzeugt.
Nachteile: Keine fließenden Farbübergänge und verschiedene Farbtabellen für jedes Bild.

'grid_scan_floyd_steinberg' grid_scan mit Floyd-Steinberg-Algorithmus für fließende Farbübergänge.

'median_cut' Median-Cut-Algorithmus für RGB-Bilder (d.h. 3 RGB- Komponenten bei `disp_image` nötig) Verfahren ähnlich wie 'grid_scan'.
Nachteile: Keine fließenden Farbübergänge und verschiedene Farbtabellen für jedes Bild.

'median_cut_floyd_steinberg' 'median_cut' mit Floyd-Steinberg-Algorithmus für fließende Farbübergänge.

Achtung

- Es ist zu beachten, daß bei der Darstellung von Echtfarbenbildern ('television', 'grid_scan', etc.) die Farbtabellen verändert werden.
- Falls ein falscher Ausgabemodus gesetzt wird, erfolgt die Fehlermeldung teilweise erst bei dem Aufruf von `disp_image`.
- Die Ausgabe der Grauwerte wird teilweise auch von Darstellungsparametern für Regionen beeinflußt. Dies kann zu unerwarteten Ergebnissen führen.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
Fenster_id.

▷ **`Mode`** (input_control) ..... string-array ⇝ *string* / integer
Name des Grauwertmodus, eventuell mit Parameter.
Defaultwert : 'default'
Werteliste : `Mode` ∈ {"default", "histogram", "row", "column", "contourline", "3D-plot", "3D-plot_hidden", "3D-plot_point"}

Beispiel

```
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')
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).
```

Ergebnis

`set_paint` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_paint` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`query_paint`, `get_paint`

Mögliche Nachfolgerfunktionen

`disp_image`

Siehe auch

`get_paint`, `query_paint`, `disp_image`, `set_shape`, `set_rgb`, `set_color`, `set_gray`

Modul

System

set_part (: : WindowHandle, Row1, Column1, Row2, Column2 :)

Veränderung des darzustellenden Bildausschnitts.

`set_part` verändert den Bildausschnitt, der im Ausgabefenster dargestellt wird. (`Row1`,`Column1`) bezeichnet dabei den linken oberen und (`Row2`,`Column2`) den rechten unteren Eckpunkt des Ausschnitts. Er wird von allen Ausgaberoutinen sowohl für Grauwerte `disp_image`, `disp_color` als auch für Regionen (`disp_region`)) beachtet. Ist nur ein Teil eines Bildes darzustellen, wird dieser Ausschnitt auf die volle Fenstergröße vergrößert. Die dafür eingesetzte Interpolations- Methode kann mittels `set_part_style` eingestellt werden. Die Daten des Bildausschnitts liefert die Prozedur `get_part`.

Neben dem direkten Setzen des Bildausschnitts werden die folgenden Sonderfälle unterstützt:

`Row1` = `Column1` = `Row2` = `Column2` = -1: Der Bildausschnitt entspricht genau der Fenstergröße, d.h. das Bild wird nicht gezoomt, gegebenfalls aber abgeschnitten.

`Row1`, `Column1` > -1 und `Row2` = `Column2` = -1: Der Bildausschnitt entspricht der Größe des letzten dargestellten Bildes, d.h. das letzte Bild wird vollständig in dem Fenster dargestellt und dafür gegebenfalls gezoomt.

Achtung

Wenn ein Bildausschnitt gewählt wurde, dann sind einige Prozeduren wie `draw_region`, `draw_ellipse`, etc. nicht mehr zu verwenden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Row1** (input_control) . . . . . rectangle.origin.y ⇝ *integer*
  Zeilennummer des linken oberen Eckpunkts des Bildausschnitts.
  Defaultwert : 0
- ▷ **Column1** (input_control) . . . . . rectangle.origin.x ⇝ *integer*
  Spaltennummer des linken oberen Eckpunkts des Bildausschnitts.
  Defaultwert : 0
- ▷ **Row2** (input_control) . . . . . rectangle.corner.y ⇝ *integer*
  Zeilennummer des rechten unteren Eckpunkts des Bildausschnitts.
  Defaultwert : -1
  Restriktion : (Row2 ≥ Row1) ∨ (Row2 = -1)
- ▷ **Column2** (input_control) . . . . . rectangle.corner.x ⇝ *integer*
  Spaltennummer des rechten unteren Eckpunkts des Bildausschnitts
  Defaultwert : -1
  Restriktion : (Column2 ≥ Column1) ∨ (Column2 = -1)

Beispiel

```
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).
```

Ergebnis

set_part liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_part wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

get_part

Mögliche Nachfolgerfunktionen

set_part_style, disp_image, disp_region

Alternativen

affine_trans_image

Siehe auch

get_part, set_part_style, disp_region, disp_image, disp_color

Modul

System

set_part_style (: : WindowHandle, Style :)

Festlegen des Interpolationsmethode für die Grauwertdarstellung.

set_part_style legt die Interpolationsmethode fest, die für die Ausgabe eines Bildausschnittes im Ausgabefenster verwendet werden soll. Interpoliert werden muß immer dann, wenn das Ausgabefenster größer oder kleiner als das Bildformat bzw. als der mittels set_part eingestellte Bildausschnitt für die Ausgabe ist. Der Parameter wird also auch verwendet, wenn mit set_part kein Ausschnitt gesetzt wurde, aber das Bild für die Ausgabe verkleinert oder vergrößert werden muß (wegen der Größe des Fenster). Derzeit werden drei Modi (Style) unterstützt:

0 keine Interpolation (eventuell niedrigere Qualität, sehr schnell).

1 ungewichtete Interpolation zwischen den Grauwerten (mittlere Qualität und Laufzeit).

2 gewichtete Interpolation zwischen den Grauwerten (höchste Qualität, langsam).

Die aktuelle Einstellung kann mittels `get_part_style` abgefragt werden.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
Fenster_id.

▷ **`Style`** (input_control) ..... integer ⇝ *integer*
Interpolationsmethode für die Bildschirmdarstellung: 0 (schnell,niedrige Qualität) bis 2 (langsam, höchste Qualität).
Defaultwert : 0
Werteliste : `Style` ∈ {0, 1, 2}

Ergebnis

`set_part_style` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_part_style` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_part_style`

Mögliche Nachfolgerfunktionen

`set_part`, `disp_image`, `disp_region`

Alternativen

`affine_trans_image`

Siehe auch

`get_part_style`, `set_part`, `disp_image`, `disp_color`

Modul

System

`set_pixel ( : : WindowHandle, Pixel : )`

Setzen eines Index in der Farbtabelle.

`set_pixel` setzt die Pixelwerte, die mit `get_pixel` abgefragt wurden. Farben (`set_color`, `set_rgb`, etc.) und Grauwert (`set_gray`) werden gemeinsam in eine ganze Zahl kodiert. Diese Zahl wird Pixel genannt. Ein Pixel ist ein Index in der Farbtabelle und hat einen Wertebereich von 0,1 bei S/W-Bildschirmen und 0..255 bei Farbbildschirmen mit 8 Bildebenen. Dieses Pixel ist also nicht mit dem Pixel („picture element“) der Bildverarbeitung (Bildpunkt) zu verwechseln. Deshalb wird bei HALCON immer zwischen Pixel und Bildpunkt (oder auch Grauwert) unterschieden.

Dieser Code (Pixel) wird mit `get_pixel` abgefragt und kann mit `set_pixel` wieder zurückgesetzt werden. Der Wertebereich des Parameters beginnt bei Null und läuft bis zu einer oberen Grenze, wobei diese der Anzahl darstellbarer Graustufen bzw. Farbwerte der Ausgabegeräte entspricht.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
Fenster_id.

▷ **`Pixel`** (input_control) ..... integer-array ⇝ *integer*
Index in der aktuellen Farbtabelle.
Defaultwert : 128
Typischer Wertebereich : 0 ≤ `Pixel` ≤ 255

Ergebnis

`set_pixel` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_pixel` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_pixel`

Mögliche Nachfolgerfunktionen

`disp_image`, `disp_region`

Alternativen

`set_rgb`, `set_color`, `set_hsi`

Siehe auch

`get_pixel`, `set_lut`, `disp_region`, `disp_image`, `disp_color`

Modul

System

`set_rgb ( : : WindowHandle, Red, Green, Blue : )`

Setzen der Ausgabefarbe durch RGB-Werte.

`set_rgb` setzt für das Ausgabefenster die Farbe(n), bzw. die Grauwerte, in der die Regionen dargestellt werden sollen. Die gewünschten Werte werden durch die drei Farbanteile: rot, grün und blau festgelegt. Wird nur eine Kombination angegeben, dann erfolgen alle Ausgaben in dieser Farbe. Wird ein Tupel von Werten übergeben, dann erfolgt die Darstellung von Tupeln von Regionen und geometrischen Objekten modulo dieser Farben.

Bei jedem Aufruf einer Ausgabeprozedur wird mit der ersten Farbe begonnen. Es ist dabei zu beachten, daß es auf die Anzahl der Objekte ankommt, die bei einem Aufruf ausgegeben werden sollen. Wird immer nur ein Objekt pro Aufruf ausgegeben, dann erfolgt dies immer in der ersten Farbe. Dies gilt auch für Objekte, die aus mehreren Zusammenhangskomponenten bestehen. Die gesetzte(n) Farbe(n) wird so lange verwendet, bis erneut `set_color`, `set_pixel`, `set_rgb` oder `set_gray` aufgerufen wird. Verwendet wird dieser Darstellungsparameter bei Prozeduren wie `disp_region`, `disp_line`, `disp_rectangle1`, `disp_rectangle2`, `disp_arrow`, etc.

Achtung

Falls die angegebene Farbe nicht verfügbar ist, führt dies die einem Exception. Nach dem Aufruf von `set_check (::'~color':)` sucht HALCON die nächste Farbe zu der angegebenen und unterdrückt den Fehler.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **`Red`** (input_control) ..... integer-array ⇝ *integer*
  Rotanteil der Farbe.
  Defaultwert : 255
  Typischer Wertebereich : $0 \leq$ `Red` ≤ 255
  Restriktion : (`0` $\leq$ `Red`) $\wedge$ (`Red` $\leq$ `255`)
- ▷ **`Green`** (input_control) ..... integer-array ⇝ *integer*
  Grünanteil der Farbe.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ `Green` ≤ 255
  Restriktion : (`0` $\leq$ `Green`) $\wedge$ (`Green` $\leq$ `255`)
- ▷ **`Blue`** (input_control) ..... integer-array ⇝ *integer*
  Blauanteil der Farbe.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ `Blue` ≤ 255
  Restriktion : (`0` $\leq$ `Blue`) $\wedge$ (`Blue` $\leq$ `255`)

Ergebnis

`set_rgb` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und für dieses die angegebene(n) Farbe(n) darstellbar sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_rgb` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`disp_image`, `disp_region`

Alternativen

`set_hsi`, `set_color`, `set_gray`

Siehe auch

`set_fix`, `disp_region`

Modul

System

`set_shape` (: : WindowHandle, Shape :)

Festlegen der Ausgabeform für Regionen.

`set_shape` gibt die Gestalt an, in der Regionen gezeichnet werden sollen. Der angegebene Wert wird dem Fenster mit der logischen Fensternummer `WindowHandle` zugeordnet. Die Ausgabe von Regionen erfolgt mit `disp_region`. Alle verfügbaren Darstellungsarten können mit `query_shape` abgefragt werden. Modi:

’original’: Die Form wird unverändert ausgegeben. Dabei können trotzdem Modifikationen durch Paramater wie `set_line_width` und `set_line_approx` verursacht werden (die gilt genauso für die anderen Modi).

’outer_circle’: Jede Region wird durch den kleinsten, umschließenden Kreis dargestellt. Siehe: `smallest_circle`.

’inner_circle’: Jede Region wird durch den größten Umkreis dargestellt. Siehe: `inner_circle`.

’ellipse’: Jede Region wird durch eine Ellipse mit gleichen Momenten und Orientierung angenähert. Siehe: `elliptic_axis`.

’rectangle1’: Jede Region wird durch das kleinste umschließende Rechtecke, parallel zu den Koordinatenachsen dargestellt. Siehe: `smallest_rectangle1`.

’rectangle2’: Jede Region wird durch das kleinste umschließende Rechteck dargestellt. Siehe: `smallest_rectangle2`.

’convex’: Es wird die konvexe Hülle jeder Region ausgegeben. Siehe: `shape_trans`.

’icon’ Es wird ein Icon, das mit `set_icon` gesetzt wurde, im Schwerpunkt der Region ausgegeben.

Achtung

Wird eine andere Prozedur als `disp_region` verwendet, dann dürfte die Ausgabe zu verblüffenden Ergebnissen führen: bei `disp_image` mit `set_paint(::WindowHandle,’histogram’:)` und `set_shape (::WindowHandle,’convex’:)` wird z.B. die konvexe Hülle des Histogramms ausgegeben.

Parameter

▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
Fenster_id.

▷ **`Shape`** (input_control) ..... string ⇝ *string*
Art der Regionenausgabe.
Defaultwert : ’original’
Werteliste : `Shape` ∈ {’original’, ’convex’, ’outer_circle’, ’inner_circle’, ’rectangle1’, ’rectangle2’, ’ellipse’, ’icon’}

Beispiel

```
read_image(Image,’fabrik’)
regiongrowing(Image,Seg,5,5,6,100)
set_colored(WindowHandle,12)
set_shape(WindowHandle,’rectangle2’)
disp_region(Seg,WindowHandle).
```

Ergebnis

`set_shape` liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt und das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_shape` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`set_icon`, `query_shape`, `get_shape`

Mögliche Nachfolgerfunktionen

`disp_region`

Siehe auch

`get_shape`, `query_shape`, `disp_region`

Modul

System

4.7 Text

`get_font` `( : : WindowHandle : Font )`

Abfragen des aktuellen Fonts.

`get_font` gibt den Namen des Fonts des Ausgabefensters aus, der von den Operatoren `write_string`, `read_string` etc. verwendet wird. Gesetzt wird der Parameter mit dem Operator `set_font`. Sowohl Textfenster als auch Bildfenster verfügen über Fonts. Beiden Fensterarten ist ein Default-Font zugeordnet, der vor dem Öffnen des Fensters durch `set_system('default_font',Fontname)` geändert werden kann. Alle verfügbaren Fonts können mit Hilfe des Operators `query_font` abgefragt werden.

Parameter

- **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- **`Font`** (output_control) ..... string ⇝ *string*
  Name des aktuellen Fonts.

Beispiel

```
get_font(WindowHandle,CurrentFont)
set_font(WindowHandle,MyFont)
write_string(WindowHandle,['The name of my Font is:',Myfont])
new_line(WindowHandle)
set_font(WindowHandle,CurrentFont)
```

Ergebnis

`get_font` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`get_font` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`, `query_font`

Mögliche Nachfolgerfunktionen

`set_font`

Siehe auch

`set_font`, `query_font`, `open_window`, `open_textwindow`, `set_system`

Modul

System

get_string_extents (: : WindowHandle, Values : Ascent, Descent, Width, Height)

Größe eines Strings abfragen.

get_string_extents bestimmt die Breite und Höhe von Strings in dem Font, der momentan für das Fenster eingestellt ist. Ausserdem wird die Ausdehnung oberhalb und unterhalb der Schreiblinie angegeben (Ascent bzw. Descent).

Die Größenangaben erfolgen in dem Koordinatensystem des Fensters (d.h. bei Textfenstern in Pixelkoordinaten). Mit Hilfe von get_string_extents ist es möglich, Textausgabe und Texteingabe unabhängig vom Font zu programmieren. Die Umsetzung von ganzen Zahlen und Gleitpunktzahlen zu Zeichenreihen erfolgt genauso wie bei write_string.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster Identifikator.
- ▷ **Values** (input_control) . . . . . string(-array) ⇝ *string* / real / integer
  Zu untersuchende Werte.
  Defaultwert : 'test_string'
- ▷ **Ascent** (output_control) . . . . . integer ⇝ *integer*
  Maximale Höhe über der Schreiblinie.
- ▷ **Descent** (output_control) . . . . . integer ⇝ *integer*
  Maximale Tiefe unter der Schreiblinie.
- ▷ **Width** (output_control) . . . . . integer ⇝ *integer*
  Länge des Textes.
- ▷ **Height** (output_control) . . . . . integer ⇝ *integer*
  Höhe des Textes.

Ergebnis

get_string_extents liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_string_extents ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, set_font

Mögliche Nachfolgerfunktionen

set_tposition, write_string, read_string, read_char

Siehe auch

set_tposition, set_font

Modul

System

get_tposition (: : WindowHandle : Row, Column)

Position des Textcursors abfragen.

get_tposition liefert die momentane Position des Textcursors im Ausgabefenster. Die Positionsangabe erfolgt im Koordinatensystem des Fensters, d.h. bei Textfenstern in Pixelkoordinaten. Die nächste Textausgabe auf dieses Fenster erfolgt ab der Cursorposition, wobei diese die linke Ecke der Schreiblinie des auszugebenden Strings (d.h. ohne Berücksichtigung von Unterlängen) bezeichnet. Die Cursorposition verändert sich bei der Ausgabe bzw. Eingabe von Text (write_string, read_string) und durch explizite Neupositionierung (set_tposition, new_line).

Achtung

Wenn der auszugebende Text nicht vollständig in das Fenster paßt, wird ein Exception ausgelöst. Dies kann durch set_check('~text') unterdrückt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster Identifikator.
- ▷ **Row** (output_control) . . . . . point.y ⇝ *integer*
  Zeilenindex der Textposition.
- ▷ **Column** (output_control) . . . . . point.x ⇝ *integer*
  Spaltenindex der Textposition.

Ergebnis

get_tposition liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_tposition ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, set_font

Mögliche Nachfolgerfunktionen

set_tposition, write_string, read_string, read_char

Siehe auch

new_line, read_string, set_tposition, write_string, set_check

Modul

System

get_tshape (: : WindowHandle : TextCursor)

Form des Textcursors abfragen.

get_tshape gibt für das Ausgabefenster die Art des momentan verwendeten Textcursors aus. Gesetzt wird ein neuer Textcursor mit dem Operator set_tshape.

Unter einem Textcursor (im Gegensatz zum Mauscursor) wird hier die Markierung der aktuellen Schreibposition verstanden (die allerdings auch unsichtbar sein kann). Welche Darstellungsarten für den Textcursor existieren, kann mit Hilfe des Operators query_tshape erfragt werden.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster Identifikator.
- ▷ **TextCursor** (output_control) . . . . . string ⇝ *string*
  Name des aktuellen Textcursors.

Ergebnis

get_tshape liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_tshape ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, set_font

Mögliche Nachfolgerfunktionen

set_tshape, set_tposition, write_string, read_string, read_char

Siehe auch

set_tshape, query_tshape, write_string, read_string

Modul

System

new_line (: : WindowHandle :)

Ausgabe eines Zeilenvorschubs im Ausgabefenster.

new_line setzt die Position des Textcursors im Ausgabefenster in Abhängigkeit vom momentan eingestellten Font auf den Anfang der nächsten Zeile. Die nächste Textausgabe auf dieses Fenster erfolgt ab der Cursorposition, wobei die Cursorposition die linke Ecke der Schreiblinie des auszugebenden Textes (d.h. ohne Berücksichtigung von Unterlängen) bezeichnet.

Falls im Fenster keine weitere Zeile mehr Platz hat, weil der untere Fensterrand erreicht ist, wird der Inhalt des Fensters um eine Zeilenhöhe nach oben geschoben („gescrolt“). Damit die Neupositionierung in der nächsten Zeile korrekt erfolgt, muß beim Aufruf von new_line schon der für die nächste Ausgabe gewünschte Font eingestellt sein. Die Cursorposition verändert sich bei der Ausgabe bzw. Eingabe von Text (write_string, read_string) und durch explizite Neupositionierung (set_tposition).

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.

Ergebnis

new_line liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

new_line wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, set_font, write_string

Alternativen

get_tposition, get_string_extents, set_tposition, move_rectangle

Siehe auch

write_string, set_font

Modul

System

query_font (: : WindowHandle : Font)

Abfragen der verfügbaren Fonts.

query_font gibt für das Ausgabefenster die Namen aller Fonts aus, die für die Textausgabe verwendet werden können. Diese können von dem Operator set_font gesetzt werden. Verwendet werden die Fonts von write_string, read_char, read_string und new_line.

Achtung

Die Fonts unterscheiden sich oft stark auf verschiedenen Rechnern. Man muß i.a. davon ausgehen, daß query_font auf verschiedenen Rechnern auch verschiedene Fonts liefert.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- ▷ **Font** (output_control) ..... string-array ⇝ *string*
  Tupel mit verfügbaren Fontnamen.

Beispiel

```
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
```

Ergebnis

query_font liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

query_font ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Mögliche Nachfolgerfunktionen

set_font, write_string, read_string, read_char

Siehe auch

set_font, write_string, read_string, read_char, new_line

Modul

System

query_tshape (: : WindowHandle : TextCursor)

Abfrage der verfügbaren Textcursor.

query_tshape gibt für das Ausgabefenster die Namen aller Darstellungsarten des Textcursors aus. Diese können von dem Operator set_tshape verwendet werden.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- **TextCursor** (output_control) ..... string-array ⇝ *string*
  Namen der verfügbaren Textcursor.

Ergebnis

query_tshape liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

query_tshape ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow

Mögliche Nachfolgerfunktionen

set_tshape, write_string, read_string

Siehe auch

set_tshape, get_shape, set_tposition, write_string, read_string

Modul

System

read_char (: : WindowHandle : Char, Code)

Einlesen eines Zeichens aus einem Textfenster.

read_char liest im Eingabefenster (= Ausgabefenster) ein Zeichen von der Tastatur ein. Falls es sich um ein druckbares Zeichen handelt, wird dieses in Char zurückgegeben. Falls eine Steuertaste gedrückt wurde, ist dies am Wert von Code erkennbar. Einige wichtige Tasten sind dadurch identifizierbar. Mögliche Werte für Code sind:

'character': druckbares Zeichen

'left': Cursor links

'right': Cursor rechts

'up': Cursor oben

'down': Cursor unten

'insert': Einfügetaste

'none': keine der angeführten Tasten

Achtung

Bei dem Fenster muß es sich um ein Textfenster handeln.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- ▷ **`Char`** (output_control) ..... string ⇝ *string*
  Eingelesenes Zeichen (falls kein Steuerzeichen).
- ▷ **`Code`** (output_control) ..... string ⇝ *string*
  Code für eingelesenes Zeichen.

Ergebnis

`read_char` liefert den Wert 2 (H_MSG_TRUE), falls das Textfenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_char` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`open_textwindow`, `set_font`

Alternativen

`read_string`, `fread_char`, `fread_string`

Siehe auch

`write_string`, `set_font`

Modul

System

`read_string ( : : WindowHandle, InString, Length : OutString )`

Einlesen eines Strings in einem Textfenster.

`read_string` liest im Eingabefenster (= Ausgabefenster) einen String mit vorgegebener Maximallänge (`Length`) von der Tastatur ein. Der String wird ab der momentanen Textcursorposition des Fensters unter Verwendung des eingestellten Fonts eingelesen. Die Maximallänge (`Length`) muß so gewählt sein, daß der rechte Rand des Fensters nicht überschritten wird. Der Operator kann ein Default-String (`InString`) übergeben werden, der vom Benutzer editiert oder direkt übernommen werden kann. Der Textcursor befindet sich nach Beendigung der Eingabe am Ende des editierten Strings. Editiermöglichkeiten:

RETURN Eingabe abschließen

BACKSPACE Zeichen links vom Cursor löschen und Textcursor um eine Position nach links.

Achtung

The window has to be a text window.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster Identifikator.
- ▷ **InString** (input_control) . . . . . string ⇝ *string*
  String, der vor dem Einlesen bereits sichtbar ist.
  Defaultwert : ”
- ▷ **Length** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Zeichen.
  Defaultwert : 32
  Restriktion : `Length` > `0`
- ▷ **OutString** (output_control) . . . . . string ⇝ *string*
  Eingelesener String.

Ergebnis

`read_string` liefert den Wert 2 (H_MSG_TRUE), falls das Textfenster gültig ist und ein String mit maximaler Länge bis zum rechten Fensterrand Platz hat. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_string` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_textwindow`, `set_font`

Alternativen

`read_char`, `fread_string`, `fread_char`

Siehe auch

`set_tposition`, `new_line`, `open_textwindow`, `set_font`, `set_color`

Modul

System

set_font (: : WindowHandle, Font :)

Setzen eines Fonts.

`set_font` setzt den Font für Ausgabefenster, der von den Operatoren `write_string`, `read_string` etc. verwendet wird. Sowohl Textfenster als auch Bildfenster verfügen über Fonts. Beim Öffnen eines Fensters wird diesem ein Default-Font zugeordnet, der anschließend mit `set_font` verändert werden kann. Auf UNIX Systemen können alle verfügbaren Fonts mit Hilfe des Operators `query_font` abgefragt werden. Der Default-Font kann mit `set_system('default_font',Fontname)` geändert werden. Fonts können nicht bei Operationen auf Dateien verwendet werden.

Die Syntax für die Angabe eines Fonts (in `Font`) unterscheidet sich für UNIX und Windows NT Systeme: In Windows NT wird ein string mit folgendem Aufbau verwendet:

```
-FontName-Height-Width-Italic-Underlined-Strikeout-[Bold-]
```

wobei “Italic”, “Underlined”, “Strikeout” und “Bold” die Werte 1 und 0 annehmen können, um das jeweilige Fontmerkmal zu aktivieren bzw. zu deaktivieren. Für weitere Einzelheiten sei auf die Windows NT Dokumentation verwiesen.

Achtung

Die verfügbaren Fonts unterscheiden sich stark bei verschiedenen Rechnern. Es empfiehlt sich daher mit Wildcards, Fonttabellen oder mit `query_font` zu arbeiten.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster Identifikator.
- ▷ **Font** (input_control) . . . . . string ⇝ *string*
  Name des Fonts, der gesetzt wird.

Beispiel

```
// UNIX
set_font(WindowHandle,'-*-courier-*-*-*-*-18-*-*-*-*-*-*-*')
// Windows NT
// Windows NT
set_font(WindowHandle,'-Arial-18-*-*-*-*-')
write_string(WindowHandle,'Text with Font size of 18 pixels')
new_line(WindowHandle)
set_font(WindowHandle,'-Arial-18-*-*-*-*-1-')
write_string(WindowHandle,'Text with Font size of 18 pixels and bold')
new_line(WindowHandle)
```

Ergebnis

`set_font` liefert den Wert 2 (H_MSG_TRUE), falls der Name korrekt ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_font` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`

Mögliche Nachfolgerfunktionen

`query_font`

Siehe auch

`get_font`, `query_font`, `open_textwindow`, `open_window`

Modul

System

set_tposition (: : WindowHandle, Row, Column :)

Setzen der Schreibposition im einem Fenster.

`set_tposition` setzt die Position des Textcursors im Ausgabefenster. Als Referenz wird das linke obere Eck eines Großbuchstabens verwendet.

Die Positionsangabe erfolgt in dem Koordinatensystem des Fensters und muß bzgl. der Größe des Fensters und des momentan eingestellten Fonts zulässig sein. Die Position des Textcursors wird z.B. durch einen Unterstrich markiert. Die nächste Textausgabe auf dieses Fenster erfolgt ab der Cursorposition, wobei die Position des Textcursors die linke Ecke der Schreiblinie des auszugebenden Strings (d.h. ohne Berücksichtigung von Unterlängen, die Zeichenhöhe über der Schreiblinie sei als Fontoberlänge bezeichnet) angibt.

Die Cursorposition verändert sich bei der Ausgabe bzw. Eingabe von Text (`write_string`, `read_string`) und durch explizite Neupositionierung (`set_tposition`, `new_line`). Um die Anzeige der Cursorposition im Fenster zu löschen, kann die Form des Textcursors mit `set_tshape` auf 'invisible' gesetzt werden.

Achtung

Falls ein String ab der angegebenen Position nicht mehr in das Fenster paßt wird ein Exception ausgelöst. Dieses Fehlermeldung kann mit `set_check('~text')` unterdrückt werden.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex der Schreibposition.
  Defaultwert : 24
- **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex der Schreibposition.
  Defaultwert : 12

Ergebnis

`set_tposition` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und die Parameterwerte zuläßig sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_tposition` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`

Mögliche Nachfolgerfunktionen

`set_tshape`, `write_string`, `read_string`

Alternativen

`new_line`

Siehe auch

`read_string`, `set_tshape`, `write_string`

Modul

System

`set_tshape` (: : WindowHandle, TextCursor :)

Form des Textcursors setzen.

`set_tshape` setzt die Darstellungsart, die für den Textcursor des Ausgabefensters verwendet werden soll.

Unter Textcursor (im Gegensatz zu Mauscursor; beide werden einfach als Cursor bezeichnet, wenn Mißverständnisse ausgeschlossen sind) wird hier die Markierung der aktuellen Schreibposition verstanden.

Ein Tupel aller verfügbaren Arten kann mit dem Operator `query_tshape` abgefragt werden.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster Identifikator.
- ▷ **`TextCursor`** (input_control) ..... string ⇝ *string*
  Name der Cursorform.
  Defaultwert : ’invisible’

Ergebnis

`set_tshape` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und für dieses die angegebene Form definiert ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_tshape` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `open_textwindow`, `query_tshape`, `get_tshape`

Mögliche Nachfolgerfunktionen

`write_string`, `read_string`

Siehe auch

`get_tshape`, `query_tshape`, `write_string`, `read_string`

Modul

System

`write_string` (: : WindowHandle, String :)

Ausgabe von Text in ein Fenster.

write_string gibt im Ausgabefenster einen String aus, an der momentanen Textcursorposition. Der auszugebende Text darf nicht länger als der bis zum rechten Fensterrand verbleibende Platz sein (gegebenenfalls kann mit Hilfe des Operators get_string_extents die Länge des Strings abgefragt werden).

Bei der Ausgabe wird der gerade für das Fenster eingestellte Font verwendet. Der Textcursor steht im Anschluß an die Ausgabe hinter dem geschriebenen Text.

write_string kann alle drei Datentypen ausgeben. Die Konversion in eine Zeichenreihe erfolgt nach folgenden Regeln:

- Strings werden unverändert übernommen.
- Ganze Zahlen werden ohne Leerzeichen (vor, bzw. hinter der Zahl) konvertiert.
- Gleitpunktzahlen werden, soweit möglich mit einem Gleitpunkt und ohne Exponent ausgegeben.
- Die sich ergebenden Zeichenreihen werden ohne Leerzeichen hintereinander ausgegeben.

Für die Pufferung von Texten siehe set_system mit dem Flag 'flush_graphic'.

Achtung

Falls ein Klipping am Fensterrand erwünscht ist, kann die Fehlermeldung mit set_check('~text') ausgeschaltet werden.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fenster Identifikator.

▷ **String** (input_control) ..... string(-array) ⇝ *string* / integer / real
Tupel von auszugebenden Werten (beliebige Typen).
Defaultwert : 'hello'

Ergebnis

write_string liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist und der auszugebende Text in der aktuellen Zeile noch Platz hat (siehe set_check). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_string ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

open_window, open_textwindow, set_font, get_string_extents

Alternativen

fwrite_string

Siehe auch

set_tposition, get_string_extents, open_textwindow, set_font, set_system, set_check

Modul

System

4.8 Zeichnen

drag_region1 (SourceRegion : DestinationRegion : WindowHandle :)

Interaktives Verschieben einer Region.

drag_region1 dient dazu, die Region mit der Maus auf dem Bildschirm zu verschieben. Nach dem Aufruf von drag_region1 wird die Region sichtbar, sobald die linke Maustaste betätigt wird. Es wird dabei nur der Rand der Region dargestellt. Als Darstellungsmodus wird für die Dauer der Prozedur der Modus 'not' (siehe set_draw) verwendet. Der Mauscursor befindet sich während des Verschiebens im Schwerpunkt der Region. Wird die Maus mit gedrückter linker Maustaste verschoben, folgt die dargestellte Region - verzögert - dieser Bewegung. Wird die rechte Maustaste gedrückt, so wird drag_region1 beendet. Die Darstellung der Region auf dem Bildschirm verschwindet. Ausgegeben wird eine Region, die der letzten Position auf dem Bildschirm entspricht. Es können auch mehrere Regionen auf einmal übergeben werden. Die Grauwerte können mit der Prozedur affine_trans_image verschoben werden.

Achtung

Die Grauwerte der Regionen werden nicht verschoben. Da die Eingaberegion verschoben wird, ist nicht sichergestellt, daß die Grauwerte der Ausgaberegionen sinnvoll besetzt sind. Dieser Fall kann eintreten, wenn die Grauwerte der Eingaberegionen nicht das ganze Bild umfassen.

Parameter

- ▷ **SourceRegion** (input_object) ..... region-array ⇝ *Hobject*
  Zu verschiebende Regionen.
- ▷ **DestinationRegion** (output_object) ..... region-array ⇝ *Hobject*
  Verschobene Regionen.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.

Beispiel

```
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().
```

Ergebnis

drag_region1 liefert den Wert 2 (H_MSG_TRUE), falls eine Region eingegeben wird, das Fenster gültig ist und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt. Das Verhalten bei einer leeren Eingabe kann mit set_system (::'no_object_result',<Result>:) bestimmt werden.

Parallelisierungsinformation

drag_region1 ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

get_mposition, move_region

Siehe auch

set_insert, set_draw, affine_trans_image

Modul

System

drag_region2 (SourceRegion : DestinationRegion : WindowHandle, Row, Column :)

Interaktives Verschieben einer Region mit Angabe des Fixpunktes.

drag_region2 dient dazu, die Region mit der Maus auf dem Bildschirm zu verschieben. Es entspricht der Prozedur drag_region1, mit dem Unterschied, daß hier die Position des Maus-Cursors angegeben werden kann.

Achtung

Die Grauwerte der Regionen werden nicht verschoben. Da die Eingaberegionen verschoben wird, ist nicht sichergestellt, daß die Grauwerte der Ausgaberegionen sinnvoll besetzt sind. Dieser Fall kann eintreten, wenn die Grauwerte der Eingaberegionen nicht das ganze Bild umfassen.

Parameter

- ▷ **SourceRegion** (input_object) ..... region-array ⇝ *Hobject*
  Zu verschiebende Regionen.
- ▷ **DestinationRegion** (output_object) ..... region-array ⇝ *Hobject*
  Verschobene Regionen.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Bezugspunktes.
  Defaultwert : 100
  Wertevorschläge : Row ∈ {0, 64, 128, 256, 512}
  Typischer Wertebereich : 0 ≤ Row ≤ 1024
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Bezugspunktes.
  Defaultwert : 100
  Wertevorschläge : Column ∈ {0, 64, 128, 256, 512}
  Typischer Wertebereich : 0 ≤ Column ≤ 1024

Ergebnis

drag_region2 liefert den Wert 2 (H_MSG_TRUE), falls eine Region eingegeben wird, das Fenster gültig ist und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt. Das Verhalten bei einer leeren Eingabe kann mit set_system (::'no_object_result',<Result>:) bestimmt werden.

Parallelisierungsinformation

drag_region2 ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert, affine_trans_image

Alternativen

get_mposition, move_region, drag_region1, drag_region3

Siehe auch

set_insert, set_draw, affine_trans_image

Modul

System

drag_region3 (SourceRegion, MaskRegion : DestinationRegion : WindowHandle, Row, Column :)

Interaktives Verschieben einer Region mit der Beschränkung der Positionen.

drag_region3 dient dazu, die Region mit der Maus auf dem Bildschirm zu verschieben. Es entspricht der Prozedur drag_region2, mit der Erweiterung, daß hier auch noch alle Punkte angegeben werden, auf denen sich die Maus bewegen darf. Wird die Maus außerhalb dieses Bereiches (MaskRegion) geführt, dann wird die Region auf dem Punkt mit dem geringsten Abstand zur Maus innerhalb von MaskRegion dargestellt.

Achtung

Die Grauwerte der Regionen werden nicht verschoben. Da die Eingaberegion verschoben wird, ist nicht sichergestellt, daß die Grauwerte der Ausgaberegionen sinnvoll besetzt sind. Dieser Fall kann eintreten, wenn die Grauwerte der Eingaberegionen nicht das ganze Bild umfassen.

Parameter

- ▷ **SourceRegion** (input_object) ..... region-array ⇝ *Hobject*
  Zu verschiebende Regionen.

▷ **MaskRegion** (input_object) ..... region-array ⇝ *Hobject*
Punkte auf denen sich die Region bewegen darf.

▷ **DestinationRegion** (output_object) ..... region-array ⇝ *Hobject*
Verschobene Regionen.

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fenster_id.

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert : 100
Wertevorschläge : `Row` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich : 0 ≤ `Row` ≤ 1024

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert : 100
Wertevorschläge : `Column` ∈ {0, 64, 128, 256, 512}
Typischer Wertebereich : 0 ≤ `Column` ≤ 1024

Ergebnis

`drag_region3` liefert den Wert 2 (H_MSG_TRUE), falls eine Region eingegeben wird, das Fenster gültig ist und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt. Das Verhalten bei einer leeren Eingabe kann mit `set_system (::'no_object_result',<Result>:)` bestimmt werden.

Parallelisierungsinformation

`drag_region3` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`, `get_mposition`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`, `affine_trans_image`

Alternativen

`get_mposition`, `move_region`, `drag_region1`, `drag_region2`

Siehe auch

`set_insert`, `set_draw`, `affine_trans_image`

Modul

System

draw_circle (: : WindowHandle : Row, Column, Radius)

Interaktives Erstellen eines Kreises.

`draw_circle` liefert die Parameter für einen Kreis, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Kreises wird mit der linken Maustaste an die Stelle geklickt, die als Mittelpunkt für den Kreis verwendet werden soll. Während die Taste gedrückt bleibt, kann die Länge des `Radius` durch Verschieben der Maus verändert werden. Nach einem erneuten Mausklick in die Mitte des erzeugten Kreises, kann dieser beliebig verschoben werden. Mit einem Klick in die Nähe des Kreisbogens kann der Kreis „angefaßt“ werden, um den `Radius` zu verändern. Ein Klick mit der rechten Maustaste beendet die Prozedur. Der Kreis ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fenster_id.

▷ **Row** (output_control) ..... circle.center.y ⇝ *real*
Zeilenindex des Schwerpunktes.

▷ **Column** (output_control) ..... circle.center.x ⇝ *real*
Spaltenindex des Schwerpunktes.
▷ **Radius** (output_control) ..... circle.radius ⇝ *real*
Radius des Kreises.

Beispiel

```
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).
```

Ergebnis

draw_circle liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_circle ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_circle_mod, draw_ellipse, draw_region

Siehe auch

gen_circle, draw_rectangle1, draw_rectangle2, draw_polygon, set_insert

Modul

System

draw_circle_mod (: : WindowHandle, RowIn, ColumnIn, RadiusIn : Row, Column, Radius)

Interaktives Erstellen eines Kreises.

draw_circle_mod liefert die Parameter für einen Kreis, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Kreises werden die Koordinaten RowIn und ColumnIn des Mittelpunktes des Kreises mit Radius RadiusIn erwartet. Nach einem erneuten Mausklick in die Mitte des erzeugten Kreises, kann dieser beliebig verschoben werden. Mit einem Klick in die Nähe des Kreisbogens kann der Kreis „angefaßt“ werden, um den Radius zu verändern. Ein Klick mit der rechten Maustaste beendet die Prozedur. Der Kreis ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
Fenster_id.
▷ **RowIn** (input_control) ..... circle.center.y ⇝ *real*
Zeilenindex des Mittelpunktes.
▷ **ColumnIn** (input_control) ..... circle.center.x ⇝ *real*
Spaltenindex des Mittelpunktes.
▷ **RadiusIn** (input_control) ..... circle.radius1 ⇝ *real*
Radius des Kreises.
▷ **Row** (output_control) ..... circle.center.y ⇝ *real*
Zeilenindex des Mittelpunktes.
▷ **Column** (output_control) ..... circle.center.x ⇝ *real*
Spaltenindex des Mittelpunktes.
▷ **Radius** (output_control) ..... circle.radius ⇝ *real*
Radius des Kreises.

Beispiel

```
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).
```

Ergebnis

draw_circle_mod liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_circle_mod ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_circle, draw_ellipse, draw_region

Siehe auch

gen_circle, draw_rectangle1, draw_rectangle2, draw_polygon, set_insert

Modul

System

draw_ellipse (: : WindowHandle : Row, Column, Phi, Radius1, Radius2)

Interaktives Erstellen einer Ellipse.

draw_ellipse liefert die Parameter für eine beliebig orientierte Ellipse, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Die zu erzeugende Ellipse wird durch ihren Mittelpunkt, ihre zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung der Ellipse wird mit der linken Maustaste der Mittelpunkt der Ellipse festgelegt. Während die Taste gedrückt bleibt, kann die Länge (Radius1) und die Orientierung (Phi) der ersten Halbachse festgelegt werden. Dabei wird für die zweite Halbachse vorläufig eine Default-Länge verwendet, die anschließend aber noch verändert werden kann. Nach einem erneuten Mausklick in die Mitte der erzeugten Ellipse, kann diese beliebig verschoben werden. Mit einem Klick in die Nähe eines Scheitels kann dieser „angefaßt“ werden, um die Länge der zugehörigen Halbachse zu verändern. Die Orientierung kann nur verändert werden, wenn ein Scheitel der ersten Halbachse angefaßt wird.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Die Ellipse ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- **Row** (output_control) ..... ellipse.center.y ⇝ *real*
  Zeilenindex des Mittlepunktes.
- **Column** (output_control) ..... ellipse.center.x ⇝ *real*
  Spaltenindex des Mittlepunktes.
- **Phi** (output_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der ersten Halbachse in Bogenmaß.
- **Radius1** (output_control) ..... ellipse.radius1 ⇝ *real*
  Erste Halbachse.
- **Radius2** (output_control) ..... ellipse.radius2 ⇝ *real*
  Zweite Halbachse.

Beispiel

```
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).
```

Ergebnis

`draw_ellipse` liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe `set_insert`) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`draw_ellipse` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`open_window`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `disp_region`, `set_colored`, `set_line_width`, `set_draw`, `set_insert`

Alternativen

`draw_ellipse_mod`, `draw_circle`, `draw_region`

Siehe auch

`gen_ellipse`, `draw_rectangle1`, `draw_rectangle2`, `draw_polygon`, `set_insert`

Modul

System

`draw_ellipse_mod ( : : WindowHandle, RowIn, ColumnIn, PhiIn, Radius1In, Radius2In : Row, Column, Phi, Radius1, Radius2 )`

Interaktives Erstellen einer Ellipse.

`draw_ellipse_mod` liefert die Parameter für eine beliebig orientierte Ellipse, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Die zu erzeugende Ellipse wird durch ihren Mittelpunkt, ihre zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung der Ellipse werden die Parametern `RowIn`, `ColumnIn`,`PhiIn`,`Radius1In`,`Radius2In` erwartet. Während die Taste gedrückt bleibt, kann die Länge (`Radius1`) und die Orientierung (`Phi`) der ersten Halbachse festgelegt werden. Dabei wird für die zweite Halbachse vorläufig eine Default-Länge verwendet, die anschließend aber noch verändert werden kann. Nach einem erneuten Mausklick in die Mitte der erzeugten Ellipse, kann diese beliebig verschoben werden. Mit einem Klick in die Nähe eines Scheitels kann dieser „angefaßt“ werden, um die Länge der zugehörigen Halbachse zu verändern. Die Orientierung kann nur verändert werden, wenn ein Scheitel der ersten Halbachse angefaßt wird.

Ein Klick mit der rechten Maustaste beendet die Prozedur. Die Ellipse ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **`RowIn`** (input_control) ..... ellipse.center.y ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **`ColumnIn`** (input_control) ..... ellipse.center.x ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **`PhiIn`** (input_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.

▷ **Radius1In** (input_control) . . . . . ellipse.radius1 ⇝ *real*
Größere Halbachse.

▷ **Radius2In** (input_control) . . . . . ellipse.radius1 ⇝ *real*
Kleinere Halbachse.

▷ **Row** (output_control) . . . . . ellipse.center.y ⇝ *real*
Zeilenindex des Mittlepunktes.

▷ **Column** (output_control) . . . . . ellipse.center.x ⇝ *real*
Spaltenindex des Mittlepunktes.

▷ **Phi** (output_control) . . . . . ellipse.angle.rad ⇝ *real*
Orientierung der ersten Halbachse in Bogenmaß.

▷ **Radius1** (output_control) . . . . . ellipse.radius1 ⇝ *real*
Erste Halbachse.

▷ **Radius2** (output_control) . . . . . ellipse.radius2 ⇝ *real*
Zweite Halbachse.

Beispiel

```
read_image(Image,'affe')
draw_ellipse_mod(WindowHandle,RowIn,ColumnIn,PhiIn,Radius1In,Radius2In,Row,Column,Phi,Ra
gen_ellipse(Ellipse,Row,Column,Phi,Radius1,Radius2)
reduce_domain(Image,Ellipse,GrayEllipse)
sobel_amp(GrayEllipse,Sobel,'sum_abs',3)
disp_image(Sobel,WindowHandle).
```

Ergebnis

draw_ellipse_mod liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_ellipse_mod ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_ellipse, draw_circle, draw_region

Siehe auch

gen_ellipse, draw_rectangle1, draw_rectangle2, draw_polygon, set_insert

Modul

System

draw_line (: : WindowHandle : Row1, Column1, Row2, Column2)

Zeichen einer Gerade.

draw_line liefert die Parameter für eine Gerade, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung der Gerade wird mit der linken Maustaste für einen Startpunkt der Gerade geklickt. Während die Taste gedrückt bleibt, kann die Gerade in jede beliebige Richtung „aufgezogen“ werden. Nach einem erneuten Mausklick in die Mitte der erzeugten Gerade, kann diese beliebig verschoben werden. Klickt man auf einen Endpunkt der erzeugten Gerade, kann diesen Punkt verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Die Gerade ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Row1** (output_control) ..... line.begin.y ⇝ *real*
  Zeilenindex des ersten Punktes der Gerade.
- ▷ **Column1** (output_control) ..... line.begin.x ⇝ *real*
  Spaltenindex des ersten Punktes der Gerade.
- ▷ **Row2** (output_control) ..... line.end.y ⇝ *real*
  Zeilenindex des zweiten Punktes der Gerade.
- ▷ **Column2** (output_control) ..... line.end.x ⇝ *real*
  Spaltenindex des zweiten Punktes der Gerade.

Beispiel

```
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().
```

Ergebnis

draw_line liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_line ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_line, set_colored, set_line_width, set_draw, set_insert

Siehe auch

draw_line_mod, gen_rectangle1, draw_circle, draw_ellipse, set_insert

Modul

System

draw_line_mod (: : WindowHandle, Row1In, Column1In, Row2In, Column2In : Row1, Column1, Row2, Column2)

Zeichen einer Gerade.

draw_line_mod liefert die Parameter für eine Gerade, die interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung der Gerade werden die Koordinaten Row1In, Column1In des Startpunktes und Row2In,Column2In des Endpunktes erwartet. Klickt man auf einen Endpunkt der erzeugten Gerade, kann diesen Punkt verschoben werden. Nach einem erneuten Mausklick in die Mitte der erzeugten Gerade, kann diese beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Die Gerade ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Row1In** (input_control) ..... line.begin.y ⇝ *real*
  Zeilenindex des ersten Punktes der Gerade.
- ▷ **Column1In** (input_control) ..... line.begin.x ⇝ *real*
  Spaltenindex des ersten Punktes der Gerade.
- ▷ **Row2In** (input_control) ..... line.end.y ⇝ *real*
  Zeilenindex des zweiten Punktes der Gerade.
- ▷ **Column2In** (input_control) ..... line.end.x ⇝ *real*
  Spaltenindex des zweiten Punktes der Gerade.
- ▷ **Row1** (output_control) ..... line.begin.y ⇝ *real*
  Zeilenindex des ersten Punktes der Gerade.
- ▷ **Column1** (output_control) ..... line.begin.x ⇝ *real*
  Spaltenindex des ersten Punktes der Gerade.
- ▷ **Row2** (output_control) ..... line.end.y ⇝ *real*
  Zeilenindex des zweiten Punktes der Gerade.
- ▷ **Column2** (output_control) ..... line.end.x ⇝ *real*
  Spaltenindex des zweiten Punktes der Gerade.

Beispiel

```
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().
```

Ergebnis

draw_line_mod returns 2 (H_MSG_TRUE), if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

Parallelisierungsinformation

draw_line_mod ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_line, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_line, draw_ellipse, draw_region

Siehe auch

gen_circle, draw_rectangle1, draw_rectangle2

Modul

System

draw_point (: : WindowHandle : Row, Column)

Zeichen eines Punktes.

draw_point liefert die Parameter für einen Punkt, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Punktes wird mit der linken Maustaste geklickt. Während die Taste gedrückt bleibt, kann der Punkt in jede beliebige Richtung „verschoben“ werden. Nach einem erneuten Mausklick kann dieses beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Der Punkt ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenindex des Punktes.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenindex des Punktes.

Beispiel

```
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().
```

Ergebnis

draw_point returns 2 (H_MSG_TRUE), if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

Parallelisierungsinformation

draw_point ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_line, set_colored, set_line_width, set_draw, set_insert

Siehe auch

draw_point_mod, draw_circle, draw_ellipse, set_insert

Modul

System

draw_point_mod (: : WindowHandle, RowIn, ColumnIn : Row, Column)

Zeichen eines Punktes.

draw_point_mod liefert die Parameter für einen Punkt, der interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Punktes werden die Koordinaten RowIn und ColumnIn erwartet. Nach einem Mausklick kann dieses beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Der Punkt ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **RowIn** (input_control) ..... point.y ⇝ *real*
  Zeilenindex des Punktes.
- ▷ **ColumnIn** (input_control) ..... point.x ⇝ *real*
  Spaltenindex des Punktes.
- ▷ **Row** (output_control) ..... point.y ⇝ *real*
  Zeilenindex des Punktes.
- ▷ **Column** (output_control) ..... point.x ⇝ *real*
  Spaltenindex des Punktes.

Beispiel

```
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_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().
```

Ergebnis

draw_point_mod returns 2 (H_MSG_TRUE), if the window is valid and the needed drawing mode is available. If necessary, an exception handling is raised.

Parallelisierungsinformation

draw_point_mod ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_line, set_colored, set_line_width, set_draw, set_insert

Siehe auch

draw_point, draw_circle, draw_ellipse, set_insert

Modul

System

draw_polygon (: PolygonRegion : WindowHandle :)

Interaktives Erstellen eines Polygonzuges.

draw_polygon liefert ein Bild, dessen Region genau die interaktiv mit der Maus eingegebene Bildpunkte umfaßt (die Grauwerte bleiben undefiniert).

Gezeichnet wird im Ausgabefenster mit gedrückter linker Maustaste. Wird die linke Maustaste losgelassen und an einer anderen Stelle wieder gedrückt, dann wird zwischen diesen beiden Punkten ein Geradenstück gezogen. Ein Druck auf die rechte Maustaste beendet die Eingabe. Für das Zeichnen wird die Farbe verwendet die mit set_color, set_rgb, etc. eingestellt wurde.

Um das erzeugte PolygonRegion für die weitere Verarbeitung mit Grauwerten zu unterlegen, kann die Prozedur reduce_domain verwendet werden.

Achtung

Die gezeichnete Kontur wird nicht automatisch geschlossen, insbesondere auch nicht „aufgefüllt“.

Die Grauwerte des Ausgabeobjektes sind undefiniert.

Parameter

- ▷ **PolygonRegion** (output_object) . . . . . region ⇝ *Hobject*
  Region, die alle gezeichneten Punkte umfaßt.
- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.

Beispiel

```
draw_polygon(Polygon,WindowHandle)
convex(Polygon,Filled)
disp_region(Filled,WindowHandle).
```

Ergebnis

draw_polygon liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_polygon ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw

Alternativen

draw_region, draw_circle, draw_rectangle1, draw_rectangle2, boundary

Siehe auch

reduce_domain, fill_up, set_color

Modul

System

draw_rectangle1 (: : WindowHandle : Row1, Column1, Row2, Column2)

Zeichen eines Rechtecks parallel zu den Koordinatenachsen.

draw_rectangle1 liefert die Parameter für ein Rechteck parallel zu den Koordinatenachsen, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Rechtecks wird mit der linken Maustaste für eine Ecke des Rechtecks geklickt. Während die Taste gedrückt bleibt, kann das Rechteck in jede beliebige Richtung „aufgezogen“ werden. Nach einem erneuten Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Mit einem Klick in die Nähe einer Seite kann diese „angefaßt“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung zu dieser Seite zu verändern. Klickt man auf eine Ecke des erzeugten Rechtecks, kann diese Ecke verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Row1** (output_control) . . . . . rectangle.origin.y ⇝ *real*
  Zeilenindex des linken oberen Ecks.
- ▷ **Column1** (output_control) . . . . . rectangle.origin.x ⇝ *real*
  Spaltenindex des linken oberen Ecks.
- ▷ **Row2** (output_control) . . . . . rectangle.corner.y ⇝ *real*
  Zeilenindex des rechten unteren Ecks.
- ▷ **Column2** (output_control) . . . . . rectangle.corner.x ⇝ *real*
  Spaltenindex des rechten unteren Ecks.

Beispiel

```
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)
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fwrite_string([',(',Row2,',',Column2,')'])
fnew_line().
```

Ergebnis

draw_rectangle1 liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_rectangle1 ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle1_mod, draw_rectangle2, draw_region

Siehe auch

gen_rectangle1, draw_circle, draw_ellipse, set_insert

Modul

System

draw_rectangle1_mod (: : WindowHandle, Row1In, Column1In, Row2In, Column2In : Row1, Column1, Row2, Column2)

Zeichen eines Rechtecks parallel zu den Koordinatenachsen.

draw_rectangle1_mod liefert die Parameter für ein Rechteck parallel zu den Koordinatenachsen, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Für die Erzeugung des Rechtecks werden die Parametern Row1In, Column1In,Row2In und Column2In erwartet. Nach einem Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Mit einem Klick in die Nähe einer Seite kann diese „angefaßt“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung zu dieser Seite zu verändern. Klickt man auf eine Ecke des erzeugten Rechtecks, kann diese Ecke verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- **Row1In** (input_control) ..... rectangle.origin.y ⇝ *real*
  Zeilenindex des linken oberen Ecks.
- **Column1In** (input_control) ..... rectangle.origin.x ⇝ *real*
  Spaltenindex des linken oberen Ecks.
- **Row2In** (input_control) ..... rectangle.corner.y ⇝ *real*
  Zeilenindex des rechten unteren Ecks.

- ▷ **Column2In** (input_control) ..... rectangle.corner.x ⇝ *real*
  Spaltenindex des rechten unteren Ecks.
- ▷ **Row1** (output_control) ..... rectangle.origin.y ⇝ *real*
  Zeilenindex des linken oberen Ecks.
- ▷ **Column1** (output_control) ..... rectangle.origin.x ⇝ *real*
  Spaltenindex des linken oberen Ecks.
- ▷ **Row2** (output_control) ..... rectangle.corner.y ⇝ *real*
  Zeilenindex des rechten unteren Ecks.
- ▷ **Column2** (output_control) ..... rectangle.corner.x ⇝ *real*
  Spaltenindex des rechten unteren Ecks.

Beispiel

```
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,Co
set_part(WindowHandle,Row1,Column1,Row2,Column2)
disp_image(Image,WindowHandle)
fwrite_string(['Clipping = (',Row1,',',Column1,')'])
fwrite_string(['-(',Row2,',',Column2,')'])
fnew_line().
```

Ergebnis

draw_rectangle1_mod liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_rectangle1_mod ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle1, draw_rectangle2, draw_region

Siehe auch

gen_rectangle1, draw_circle, draw_ellipse, set_insert

Modul

System

draw_rectangle2 (: : WindowHandle : Row, Column, Phi, Length1, Length2)

Interaktives Erstellen eines beliebig orientierten Rechtecks.

draw_rectangle2 liefert die Parameter für ein beliebig orientiertes Rechteck, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Das zu erzeugende Rechteck wird durch seinen Mittelpunkt, seine zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung des Rechtecks wird mit der linken Maustaste der Mittelpunkt des Rechtecks festgelegt. Während die Taste gedrückt bleibt, kann die Länge (Length1) und die Orientierung (Phi) der ersten Halbachse festgelegt werden. Dabei wird für die zweite Halbachse vorläufig eine Default-Länge verwendet, die anschließend aber noch verändert werden kann. Nach einem erneuten Mausklick in die Mitte des erzeugten Rechtecks,

kann dieses beliebig verschoben werden. Mit einem Klick in die Nähe einer Seite kann diese „angefaßt“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung zu dieser Seite zu verändern. Die Orientierung kann nur verändert werden, wenn eine Seite senkrecht zur ersten Halbachse angefaßt wird. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **Row** (output_control) . . . . . rectangle2.center.y ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **Column** (output_control) . . . . . rectangle2.center.x ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **Phi** (output_control) . . . . . rectangle2.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.
- ▷ **Length1** (output_control) . . . . . rectangle2.hwidth ⇝ *real*
  Größere Halbachse.
- ▷ **Length2** (output_control) . . . . . rectangle2.hheight ⇝ *real*
  Kleinere Halbachse.

Ergebnis

draw_rectangle2 liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_rectangle2 ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle2_mod, draw_rectangle1, draw_region

Siehe auch

gen_rectangle2, draw_circle, draw_ellipse, set_insert

Modul

System

draw_rectangle2_mod (: : WindowHandle, RowIn, ColumnIn, PhiIn, Length1In, Length2In : Row, Column, Phi, Length1, Length2)

Interaktives Erstellen eines beliebig orientierten Rechtecks.

draw_rectangle2_mod liefert die Parameter für ein beliebig orientiertes Rechteck, das interaktiv durch den Benutzer im Fenster erzeugt worden ist.

Das zu erzeugende Rechteck wird durch seinen Mittelpunkt, seine zwei Halbachsen und den Winkel der ersten Halbachse zur horizontalen Koordinatenachse beschrieben.

Für die Erzeugung des Rechtecks werden die Parametern RowIn, ColumnIn,PhiIn,Length1In,Length2In erwartet. Mit einem Klick in die Nähe einer Seite kann diese „angefaßt“ werden, um die Ausdehnung des Rechtecks in senkrechter Richtung (Length2) zu dieser Seite zu verändern. Die Orientierung (Phi) kann nur verändert werden, wenn eine Seite senkrecht zur ersten Halbachse angefaßt wird. Nach einem erneuten Mausklick in die Mitte des erzeugten Rechtecks, kann dieses beliebig verschoben werden. Ein Klick mit der rechten Maustaste beendet die Prozedur.

Das Rechteck ist nach Beendigung der Prozedur nicht mehr auf dem Fenster sichtbar.

Parameter

- ▷ **WindowHandle** (input_control) . . . . . window ⇝ *integer*
  Fenster_id.
- ▷ **RowIn** (input_control) . . . . . rectangle2.center.y ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **ColumnIn** (input_control) . . . . . rectangle2.center.x ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **PhiIn** (input_control) . . . . . rectangle2.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.
- ▷ **Length1In** (input_control) . . . . . rectangle2.hwidth ⇝ *real*
  Größere Halbachse.
- ▷ **Length2In** (input_control) . . . . . rectangle2.hheight ⇝ *real*
  Kleinere Halbachse.
- ▷ **Row** (output_control) . . . . . rectangle2.center.y ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **Column** (output_control) . . . . . rectangle2.center.x ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **Phi** (output_control) . . . . . rectangle2.angle.rad ⇝ *real*
  Orientierung der größeren Halbachse in Bogenmaß.
- ▷ **Length1** (output_control) . . . . . rectangle2.hwidth ⇝ *real*
  Größere Halbachse.
- ▷ **Length2** (output_control) . . . . . rectangle2.hheight ⇝ *real*
  Kleinere Halbachse.

Ergebnis

draw_rectangle2_mod liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig und der benötigte Zeichenmodus (siehe set_insert) verfügbar ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_rectangle2_mod ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw, set_insert

Alternativen

draw_rectangle2, draw_rectangle1, draw_rectangle2, draw_region

Siehe auch

gen_rectangle2, draw_circle, draw_ellipse, set_insert

Modul

System

draw_region (: Region : WindowHandle :)

Interaktives Erstellen einer geschlossenen Region.

draw_region liefert ein Bild, dessen Region die interaktiv mit der Maus eingegebene Bildregion umfaßt (die Grauwerte bleiben undefiniert). Gezeichnet wird im Ausgabefenster mit gedrückter linker Maustaste. Die linke Maustaste kann auch punktweise betätigt werden; hierdurch wird eine Gerade zwischen den angeklickten Punkten gezogen. Ein Druck auf die rechte Maustaste beendet die Eingabe und schließt die Kontur. Anschließend wird das Bild „aufgefüllt“, enthält also den gesamten mit der Maus umfahrenen Bildbereich.

Für das Zeichnen wird die Farbe verwendet die mit set_color, set_rgb, etc. eingestellt wurde.

Achtung

Die Grauwerte des Ausgabeobjektes sind undefiniert.

Parameter

- ▷ **Region** (output_object) ..... region ⇝ *Hobject*
  Interaktiv erstellte Region.
- ▷ **WindowHandle** (input_control) ..... window ⇝ *integer*
  Fenster_id.

Beispiel

```
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).
```

Ergebnis

draw_region liefert den Wert 2 (H_MSG_TRUE), falls das Fenster gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

draw_region ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

open_window

Mögliche Nachfolgerfunktionen

reduce_domain, disp_region, set_colored, set_line_width, set_draw

Alternativen

draw_circle, draw_ellipse, draw_rectangle1, draw_rectangle2

Siehe auch

draw_polygon, reduce_domain, fill_up, set_color

Modul

System

Kapitel 5

Klassifikation

clear_sampset (: : SampKey :)

Speicher eines Datensatzes freigeben.

clear_sampset gibt den Speicherbereich, der von einem mit read_sampset eingelesenen Trainingsdatensatz belegt wird, wieder frei. Dieser kann nur mittels read_sampset wiederverwendet werden.

Parameter

▷ **SampKey** (input_control) . . . . . feature_set ⇝ *integer*
Nummer des Datensatzes.

Ergebnis

clear_sampset liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls der Schlüssel SampKey nicht existiert.

Parallelisierungsinformation

clear_sampset ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, enquire_class_box, learn_class_box, write_class_box

Siehe auch

test_sampset_box, learn_sampset_box, read_sampset

Modul

Tools

close_all_class_box (: : :)

Löschen aller Klassifikatoren.

close_all_class_box löscht alle Klassifikatoren und stellt den Speicherplatz wieder zur Verfügung. Die gesamte gelernte Information geht verloren.

Achtung

Da alle Klassifikatoren geschlossen werden, sind alle vorhandenen Handle ungültig.

Ergebnis

Gelingt es, die Klassifikatoren zu schließen, liefert close_all_class_box den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_all_class_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Alternativen

close_class_box

Modul

Tools

close_class_box (: : ClassifHandle :)

Löschen des Klassifikators.

close_class_box löscht den Klassifikator und stellt den Speicherplatz wieder zur Verfügung. Die gesamte gelernte Information geht verloren. Es kann gegebenenfalls write_class_box verwendet werden um die trainierten Daten abzuspeichern.

Parameter

▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
Nummer des zu verwendenen Klassifikators.

Ergebnis

close_class_box liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

close_class_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, enquire_class_box, learn_class_box, write_class_box

Siehe auch

create_class_box, enquire_class_box, learn_class_box

Modul

Tools

create_class_box (: : : ClassifHandle)

Erzeugen eines neuen Klassifikators.

create_class_box erzeugt einen neuen, lernfähigen Klassifikator. Alle im Kapitel Klassifikation beschriebenen Prozeduren beziehen sich auf einen derart initialisierten Klassifikator (vom Typ 2). Eine genauere Beschreibung ist beim Operator learn_class_box zu finden.

Parameter

▷ **ClassifHandle** (output_control) . . . . . class_box ⇝ *integer*
Nummer des zu verwendenen Klassifikators.

Ergebnis

create_class_box liefert den Wert 2 (H_MSG_TRUE), falls der Parameter korrekt ist. Falls ein Klassifikator mit diesem Namen bereits existiert, oder nicht genügend Speicher vorhanden ist, wird eine Exception-Behandlung ausgelöst.

Parallelisierungsinformation

create_class_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

learn_class_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

learn_class_box, enquire_class_box, close_class_box

Modul

Tools

descript_class_box (: : ClassifHandle, Dimensions :)

Beschreibung des Klassifikators.

Ein Klassifikator verwendet eine Menge von Hyperquadern für jede Klasse mit denen versucht wird, die Merkmalsvektoren innerhalb der Klasse einzufangen. descript_class_box gibt für jede Klasse die Ausdehnung jedes zugehörigen Quaders entlang der Dimensionen 1 bis Dimensions (auf standard_output) aus.

Parameter

- **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- **Dimensions** (input_control) . . . . . integer ⇝ *integer*
  Höchste auszugebende Dimension.
  Defaultwert : 3

Ergebnis

descript_class_box liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

descript_class_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, learn_class_box, set_class_box_param

Mögliche Nachfolgerfunktionen

enquire_class_box, learn_class_box, write_class_box, close_class_box

Siehe auch

create_class_box, enquire_class_box, learn_class_box, read_class_box, write_class_box

Modul

Tools

enquire_class_box (: : ClassifHandle, FeatureList : Class)

Klassifikation eines Merkmalstupels.

FeatureList ist ein Tupel von beliebigen Gleitpunkt- oder ganzen Zahlen (Merkmalen), das mit Hilfe eines früher entsprechend trainierten (learn_class_box) Klassifikators einer Klasse zugeordnet werden soll. Es ist möglich, Merkmale als unbekannt anzugeben, indem anstelle einer Zahl das Zeichen ’∗’ angegeben wird. Falls n Werte angegeben sind, werden automatisch alle weiteren, also Merkmal n+1 bis max, als undefiniert angenommen. Näheres zur Funktionsweise siehe learn_class_box.
Die Prozeduren learn_class_box und enquire_class_box können abwechselnd aufgerufen werden, so daß bereits in der Lernphase klassifiziert werden kann. Auf diese Weise läßt sich feststellen, wann ein zufriedenstellendes Verhalten erreicht wurde.

Parameter

- **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- **FeatureList** (input_control) . . . . . real-array ⇝ *real* / integer / string
  Merkmalsvektor, der klassifiziert werden soll.
  Defaultwert : 1.0
- **Class** (output_control) . . . . . integer ⇝ *integer*
  Nummer der Klasse, der der Merkmalsvektor zugeordnet wurde.

Ergebnis

enquire_class_box liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

enquire_class_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`create_class_box`, `learn_class_box`, `set_class_box_param`

Mögliche Nachfolgerfunktionen

`learn_class_box`, `write_class_box`, `close_class_box`

Alternativen

`enquire_reject_class_box`

Siehe auch

`test_sampset_box`, `learn_class_box`, `learn_sampset_box`

Modul

Tools

`enquire_reject_class_box ( : : ClassifHandle, FeatureList : Class )`

Klassifikation eines Merkmalstupels mit Rückweisungsklasse.

`FeatureList` ist ein Tupel von beliebigen Gleitpunkt- oder ganzen Zahlen (Merkmalen), das mit Hilfe eines früher entsprechend trainierten (`learn_class_box`) Klassifikators einer Klasse zugeordnet werden soll. Es ist möglich, Merkmale als unbekannt anzugeben, indem anstelle einer Zahl das Zeichen '$*$' angegeben wird. Falls n Werte angegeben sind, werden automatisch alle weiteren, also Merkmal n+1 bis max, als undefiniert angenommen. Kann der Merkmalsvektor keiner Klasse zugeordnet werden, d.h. der Vektor liegt nicht innerhalb einer der hyper-Boxen, dann wird im Gegensatz zu `enquire_class_box` nicht die nächste Klasse, sonder die Rückweisungklasse **-1** als Ergebnis übergeben.
Näheres zur Funktionsweise siehe `learn_class_box`.
Die Prozeduren `learn_class_box` und `enquire_class_box` können abwechselnd aufgerufen werden, so daß bereits in der Lernphase klassifiziert werden kann. Auf diese Weise läßt sich feststellen, wann ein zufriedenstellendes Verhalten erreicht wurde.

Parameter

- **`ClassifHandle`** (input_control) ..... class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- **`FeatureList`** (input_control) ..... real-array ⇝ *real* / integer / string
  Merkmalsvektor, der klassifiziert werden soll.
  Defaultwert : 1.0
- **`Class`** (output_control) ..... integer ⇝ *integer*
  Nummer der Klasse, der der Merkmalsvektor zugeordnet wurde oder -1 für die Rückweisungsklasse.

Ergebnis

`enquire_reject_class_box` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`enquire_reject_class_box` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`create_class_box`, `learn_class_box`, `set_class_box_param`

Mögliche Nachfolgerfunktionen

`learn_class_box`, `write_class_box`, `close_class_box`

Alternativen

`enquire_class_box`

Siehe auch

`test_sampset_box`, `learn_class_box`, `learn_sampset_box`

Modul

Tools

get_class_box_param (: : ClassifHandle, Flag : Value)

Informationen über den aktuellen Parameter.

get_class_box_param holt die Parameter des Klassifikators. Die Bedeutung der Parameter wird bei set_class_box_param näher erklärt.

Vorbesetzung:
'min_samples_for_split' = 80,
'split_error' = 0.1,
'prop_constant' = 0.25

Parameter

- ▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- ▷ **Flag** (input_control) . . . . . string ⇝ *string*
  Name des Systemparameters.
  Defaultwert : 'split_error'
  Werteliste : Flag ∈ {'split_error', 'prop_constant', 'used_memory', 'min_samples_for_split'}
- ▷ **Value** (output_control) . . . . . number ⇝ *real* / integer
  Wert des Systemparameters.

Ergebnis

get_class_box_param liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls Flag mit falschen Werten belegt wird.

Parallelisierungsinformation

get_class_box_param ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, enquire_class_box, learn_class_box, write_class_box

Mögliche Nachfolgerfunktionen

set_class_box_param, learn_class_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

create_class_box, set_class_box_param

Modul

Tools

learn_class_box (: : ClassifHandle, Features, Class :)

Trainieren des Klassifikators.

Features ist ein Tupel von beliebigen Gleitpunktzahlen oder ganzen Zahlen (Merkmalen), das der Klasse Class, die durch eine ganze Zahl angegeben wird, zugeordnet werden soll. Die Prozedur enquire_class_box kann später verwendet werden, um zu beliebigen Vektoren (=Tupeln) die plausibelste Klasse zu finden. Das Verfahren versucht, die Menge der zu einer Klasse gehörigen Vektoren durch Hyperquader im Merkmalsraum zu beschreiben, wobei nach Bedarf auch mehrere Quader pro Klasse erzeugt werden. Daher ist es auch möglich, disjunkte Konzepte zu lernen, das heißt solche, die in mehrere „Punktwolken“ im Merkmalsraum zerfallen. Die Datenstruktur bleibt dem Benutzer verborgen und ist nur mit Hilfe der in diesem Abschnitt beschriebenen Prozeduren zugänglich.
Es ist möglich, Merkmale als unbekannt anzugeben, indem anstelle einer Zahl das Zeichen '∗' angegeben wird. Falls n Werte angegeben werden, werden automatisch alle weiteren, also Merkmal n+1 bis max, als undefiniert angenommen.
Die Prozeduren learn_class_box und enquire_class_box können abwechselnd aufgerufen werden, so daß bereits in der Lernphase klassifiziert werden kann. Auf diese Weise läßt sich feststellen, wann ein zufriedenstellendes Verhalten erreicht wurde.
Der Klassifikator wird durch weiteres Training nur größer, das heißt es ist nicht ratsam, nach Erreichen eines zufriedenstellenden Verhaltens weiterzutrainieren.

Parameter

- ▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- ▷ **Features** (input_control) . . . . . number-array ⇝ *real* / integer / string
  Zu lernender Merkmalsvektor.
  Defaultwert : '[1.0,1.5,2.0]'
- ▷ **Class** (input_control) . . . . . integer ⇝ *integer*
  Klasse, dem der Vektor zugeordnet werden soll.
  Defaultwert : 1

Ergebnis

learn_class_box liefert im Normalfall den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls Speicherplatzprobleme auftreten. Die Anzahl der Klassen ist beschränkt, bei Überschreitung dieser Grenze tritt ebenfalls eine Exception-Behandlung ein.

Parallelisierungsinformation

learn_class_box ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, enquire_class_box

Mögliche Nachfolgerfunktionen

test_sampset_box, learn_class_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

test_sampset_box, close_class_box, create_class_box, enquire_class_box, learn_sampset_box

Modul

Tools

learn_sampset_box (: : ClassifHandle, SampKey, Outfile, NSamples, StopError, ErrorN :)

Training des Klassifikators mit einem Datensatz.

learn_sampset_box trainiert den Klassifikator mit den Daten zum Schlüssel SampKey (siehe read_sampset). Der Trainingsvorgang wird spätestens nach NSamples Beispielen abgebrochen. Falls NSamples größer als die Anzahl der Beispiele in SampKey ist, wird zyklisch von vorne begonnen. Falls der Fehler den Wert StopError unterschreitet, wird der Trainingsvorgang vorzeitig beendet. StopError berechnet sich dabei N / ErrorN, wobei N die Anzahl der Beispiele ist, die während der letzten ErrorN Trainingsbeispiele falsch klassifiziert wurden. Typischerweise ist ErrorN die Anzahl der Beispiele in SampKey und NSamples ein Vielfaches hiervon. Soll ein Datensatz mit 100 Beispielen höchstens fünf mal durchlaufen werden und bei einem Fehler von weniger als 5% abgebrochen werden, dann lauten die entsprechenden Werte NSamples = 500, ErrorN = 100, StopError = 0.05. Ein Protokoll des Lernvorgangs wird in die Datei Outfile geschrieben.

Parameter

- ▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- ▷ **SampKey** (input_control) . . . . . feature_set ⇝ *integer*
  Nummer des Trainingsdatensatzes.
- ▷ **Outfile** (input_control) . . . . . filename ⇝ *string*
  Name der Protokolldatei.
  Defaultwert : 'training_prot'
- ▷ **NSamples** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der zu lernenden Merkmalsvektoren.
  Defaultwert : 500
- ▷ **StopError** (input_control) . . . . . real ⇝ *real*
  Klassifikationsfehler für Abbruch.
  Defaultwert : 0.05

▷ **ErrorN** (input_control) . . . . . integer ⇝ *integer*
Fehler bei der Zuordnung
Defaultwert : 100

Ergebnis

learn_sampset_box liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls der Schlüssel SampKey nicht existiert oder Probleme beim Öffnen der Datei auftreten.

Parallelisierungsinformation

learn_sampset_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box

Mögliche Nachfolgerfunktionen

test_sampset_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

test_sampset_box, enquire_class_box, learn_class_box, read_sampset

Modul

Tools

read_class_box (: : ClassifHandle, FileName :)

Einlesen eines Klassifikators von Datei.

read_class_box übernimmt den in der Datei FileName (siehe write_class_box) gespeicherten Klassifikator. Die Werte des Klassifikators werden dabei überschrieben.

Achtung

Alle Werte des Klassifikators werden überschrieben.

Parameter

▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
Nummer des zu verwendenen Klassifikators.

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Dateiname des Klassifikators.
Defaultwert : ’klassifikator1’

Ergebnis

read_class_box liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls die Datei FileName nicht geöffnet werden konnte, oder falls die Datei das falsche Format hatte.

Parallelisierungsinformation

read_class_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box

Mögliche Nachfolgerfunktionen

test_sampset_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

create_class_box, write_class_box

Modul

Tools

read_sampset (: : FileName : SampKey)

Einlesen eines Trainingsdatensatzes von Datei.

Die Trainingsbeispiele sind über den Schlüssel SampKey mit den Prozeduren clear_sampset und learn_sampset_box zugreifbar. Die Datei kann mit einem Editor erstellt werden. Jede Zeile enthält einen Merkmalsvektor mit zugehöriger Klasse. Ein Beispiel für das Format:

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

Diese Zeile spezifiziert einen Merkmalsvektor, der zur Klasse 3 gehört und bei dem das dritte Merkmal unbekannt ist. Merkmale von fünf aufwärts werden ebenfalls als unbekannt angenommen. Kommentare der Form /* ... */ können an beliebigen Stellen eingefügt werden.

Parameter

- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Dateiname des Trainingsdatensatzes.
  Defaultwert : 'sampset1'
- ▷ **SampKey** (output_control) ..... feature_set ⇝ *integer*
  Identifikation des Trainingsdatensatzes.

Ergebnis

read_sampset liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls die Datei nicht geöffnet werden kann, falls sie Syntaxfehler enthält oder falls nicht genug Speicher zur Verfügung steht.

Parallelisierungsinformation

read_sampset ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box

Mögliche Nachfolgerfunktionen

test_sampset_box, enquire_class_box, write_class_box, close_class_box, clear_sampset

Siehe auch

test_sampset_box, clear_sampset, learn_sampset_box

Modul

Tools

set_class_box_param (: : ClassifHandle, Flag, Value :)

Setzen von Systemparametern zur Klassifikation.

set_class_box_param ändert Parameter, die den Lernvorgang beim Aufruf von learn_class_box beeinflussen. Die Parameter werden nur bei dem Klassifikator ClassifHandle geändert, bei allen anderen Klassifikatoren bleiben sie unverändert. 'min_samples_for_split' ist die Anzahl der Trainingsbeispiele, die mindestens in einen Quader des Klassifikators gefallen sein müssen, bevor sich dieser Quader teilen darf. 'split_error' gibt den kritischen Fehler an, bei dessen Überschreitung sich ein Quader teilt, falls er schon mehr als 'min_samples_for_split' Trainingsbeispiele eingefangen hat. 'prop_constant' beeinflußt die Ausdehnung der Quader. Sie ist proportional zum durchschnittlichen Abstand der Trainingsbeispiele in diesem Quader zum Mittelpunkt des Quaders. Genauer gesagt gilt:

$$\text{Ausdehnung} \times \text{Prop} = \text{durchschnittlicher Abstand vom Erwartungswert.}$$

Diese Beziehung gilt entlang jeder Dimension. Innerhalb eines Quaders werden also die Dimensionen des Merkmalsraums als unabhängig angenommen.

Die Parameter sind mit weitgehend problemunabhängigen Defaultwerten vorbesetzt, die nicht grundlos verändert werden sollten. Die Parameter sind nur beim Lernvorgang von Bedeutung. Sie haben keinen Einfluß auf das Verhalten von enquire_class_box.

Vorbesetzung: 'min_samples_for_split' = 80,
'split_error' = 0.1,
'prop_constant' = 0.25

Parameter

- ▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- ▷ **Flag** (input_control) . . . . . string ⇝ *string*
  Name des gewünschten Parameters.
  Defaultwert : 'split_error'
  Wertevorschläge : `Flag` ∈ {'min_samples_for_split', 'split_error', 'prop_constant'}
- ▷ **Value** (input_control) . . . . . number ⇝ *real* / integer
  Wert des Parameters.
  Defaultwert : 0.1

Ergebnis

`set_class_box_param` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`set_class_box_param` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`create_class_box`, `enquire_class_box`

Mögliche Nachfolgerfunktionen

`learn_class_box`, `test_sampset_box`, `write_class_box`, `close_class_box`, `clear_sampset`

Siehe auch

`enquire_class_box`, `get_class_box_param`, `learn_class_box`

Modul

Tools

test_sampset_box (: : ClassifHandle, SampKey : Error)

Klassifikation einer Menge von Vektoren.

Im Unterschied zu `learn_sampset_box` findet kein Lernen statt. Typischerweise verwendet man `test_sampset_box`, um einen Klassifikator auf unabhängige Testdaten anzuwenden. `Error` gibt dann Auskunft über die Anwendbarkeit des Gelernten auf neue Beispiele.

Parameter

- ▷ **ClassifHandle** (input_control) . . . . . class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- ▷ **SampKey** (input_control) . . . . . feature_set ⇝ *integer*
  Schlüssel der Testdaten.
- ▷ **Error** (output_control) . . . . . real ⇝ *real*
  Fehler bei der Zuordnung.

Ergebnis

`test_sampset_box` liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls der Schlüssel `SampKey` nicht existiert oder Probleme beim Öffnen der Datei auftreten.

Parallelisierungsinformation

`test_sampset_box` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`create_class_box`, `learn_class_box`, `set_class_box_param`

Mögliche Nachfolgerfunktionen

`enquire_class_box`, `learn_class_box`, `write_class_box`, `close_class_box`, `clear_sampset`

Siehe auch

`enquire_class_box`, `learn_class_box`, `learn_sampset_box`, `read_sampset`

Modul

Tools

write_class_box (: : ClassifHandle, FileName :)

Ausgabe des Klassifikators auf Datei.

write_class_box sichert den Klassifikator in eine Datei. Die Daten können bei Bedarf mit read_class_box wieder eingelesen werden.

Achtung

Falls eine Datei mit diesem Namen bereits existiert, wird sie ohne Warnung überschrieben. Die Datei kann nicht editiert werden.

Parameter

- **ClassifHandle** (input_control) ..... class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.
- **FileName** (input_control) ..... filename ⇝ *string*
  Name der Datei, auf die die Daten geschrieben werden.
  Defaultwert : 'klassifikator1'

Ergebnis

write_class_box liefert den Wert 2 (H_MSG_TRUE). Eine Exception-Behandlung wird ausgelöst, falls die Datei FileName nicht geöffnet werden konnte.

Parallelisierungsinformation

write_class_box ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, enquire_class_box, learn_class_box, test_sampset_box, write_class_box

Mögliche Nachfolgerfunktionen

close_class_box, clear_sampset

Siehe auch

create_class_box, read_class_box

Modul

Tools

Kapitel 6

Linien

6.1 Merkmale

line_orientation (: : RowBegin, ColBegin, RowEnd, ColEnd : Phi)

Berechnung der Orientierung von Linien.

line_orientation liefert die Orientierung ($-\pi/2 < $ Phi $\leq \pi/2$) der übergebenen Linien. Falls mehr als eine Linie behandelt werden soll, können die Zeilen- und Spaltenindizes als Tupel übergeben werden. In diesem Fall ist Phi natürlich ebenfalls ein Tupel und enthält die entsprechenden Orientierungen.

Die Prozedur wird typischerweise auf Modellinien angewandt, um anschließend mittels select_lines dazu parallele Bildlinien — die z.B. mit detect_edge_segments gefunden wurden — auszuwählen.

Parameter

- ▷ **RowBegin** (input_control) . . . . . line.begin.y(-array) ⇝ *real* / integer
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **ColBegin** (input_control) . . . . . line.begin.x(-array) ⇝ *real* / integer
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **RowEnd** (input_control) . . . . . line.end.y(-array) ⇝ *real* / integer
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **ColEnd** (input_control) . . . . . line.end.x(-array) ⇝ *real* / integer
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **Phi** (output_control) . . . . . angle.rad(-array) ⇝ *real*
  Orientierung der Eingabelinien.

Ergebnis

line_orientation liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

line_orientation ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

sobel_amp, edges_image, threshold, hysteresis_threshold, split_skeleton_region, split_skeleton_lines

Mögliche Nachfolgerfunktionen

set_line_width, disp_line

Alternativen

line_position, select_lines, partition_lines

Siehe auch

line_position, select_lines, partition_lines, detect_edge_segments

Modul

Region processing

line_position (: : RowBegin, ColBegin, RowEnd, ColEnd : RowCenter, ColCenter, Length, Phi)

Berechnung des Schwerpunktes, der Länge und der Orientierung von Linien.

line_position liefert den Schwerpunkt (RowCenter, ColCenter), die (euklidische) Länge (Length) und die Orientierung ($-\pi/2 <$ Phi $\leq \pi/2$) der übergebenen Linien. Falls mehr als eine Linie behandelt werden soll, können die Zeilen- und Spaltenindizes als Tupel übergeben werden. In diesem Fall sind die Ausgabeparameter natürlich ebenfalls Tupel.

Die Routine wird beispielsweise auf Modellinien angewandt, um Suchbereiche für die Kantendetektion (detect_edge_segments) zu bestimmen.

Parameter

- ▷ **RowBegin** (input_control) ..... line.begin.y(-array) ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **ColBegin** (input_control) ..... line.begin.x(-array) ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **RowEnd** (input_control) ..... line.end.y(-array) ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **ColEnd** (input_control) ..... line.end.x(-array) ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **RowCenter** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenkoordinaten der Schwerpunkte der Eingabelinien.
- ▷ **ColCenter** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenindizes der Schwerpunkte der Eingabelinien.
- ▷ **Length** (output_control) ..... real(-array) ⇝ *real*
  Euklidische Länge der Eingabelinien.
- ▷ **Phi** (output_control) ..... angle.rad(-array) ⇝ *real*
  Orientierung der Eingabelinien.

Ergebnis

line_position liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

line_position ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

sobel_amp, edges_image, threshold, hysteresis_threshold, split_skeleton_region, split_skeleton_lines

Mögliche Nachfolgerfunktionen

set_line_width, disp_line

Alternativen

line_orientation, select_lines, partition_lines

Siehe auch

line_orientation, select_lines, partition_lines, detect_edge_segments

Modul

Region processing

partition_lines (: : RowBeginIn, ColBeginIn, RowEndIn, ColEndIn, Feature, Operation, Min, Max : RowBeginOut, ColBeginOut, RowEndOut, ColEndOut, FailRowBOut, FailColBOut, FailRowEOut, FailColEOut)

Partitionierung von Linien gemäß verschiedener Kriterien.

partition_lines teilt Linien anhand verschiedener Kriterien in zwei Mengen auf. Für jede Eingabelinie werden die angegebenen Merkmale (Feature) berechnet. Wenn jedes (Operation = ’and’) oder mindestens eines (Operation = ’or’) der so berechneten Merkmale in den vorgegebenen Grenzen (Min,Max)l liegt, wird

die Linie in die erste Menge (Parameter `RowBeginOut` bis `ColEndOut`) übernommen, andernfalls in die zweite Menge (Parameter `FailRowBOut` bis `FailColEOut`).

Bedingung: $Min_i \leq Feature_i(Linie) \leq Max_i$

Mögliche Werte für `Feature`:

'length' (euklidische) Länge der Linie

'row' Zeilenindex des Schwerpunkts

'column' Spaltenindex des Schwerpunkts

'phi' Orientierung der Linie ($-\frac{\pi}{2} < \varphi \leq \frac{\pi}{2}$)

Achtung

Wird nur ein Merkmal verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie übergeben werden.

Parameter

- ▷ **`RowBeginIn`** (input_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **`ColBeginIn`** (input_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **`RowEndIn`** (input_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **`ColEndIn`** (input_control) ..... line.end.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **`Feature`** (input_control) ..... string(-array) ⇝ *string*
  Gewünschte Merkmale.
  Werteliste : `Feature` ∈ {'length', 'row', 'column', 'phi'}
- ▷ **`Operation`** (input_control) ..... string ⇝ *string*
  Gewünschte Verknüpfung.
  Werteliste : `Operation` ∈ {'and', 'or'}
- ▷ **`Min`** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Untere Grenzen der Merkmale oder 'min'.
  Defaultwert : 'min'
- ▷ **`Max`** (input_control) ..... string(-array) ⇝ *string* / integer / real
  Obere Grenzen der Merkmale oder 'max'.
  Defaultwert : 'max'
- ▷ **`RowBeginOut`** (output_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **`ColBeginOut`** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **`RowEndOut`** (output_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **`ColEndOut`** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Linien, die die Bedingung(en) erfüllen.
- ▷ **`FailRowBOut`** (output_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) nicht erfüllen.
- ▷ **`FailColBOut`** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Linien, die die Bedingung(en) nicht erfüllen.
- ▷ **`FailRowEOut`** (output_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Linien, die die Bedingung(en) nicht erfüllen.
- ▷ **`FailColEOut`** (output_control) ..... line.end.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Linien, die die Bedingung(en) nicht erfüllen.

Ergebnis

`partition_lines` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`partition_lines` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Mögliche Nachfolgerfunktionen

`set_line_width`, `disp_line`

Alternativen

`line_orientation`, `line_position`, `select_lines`, `select_lines_longest`

Siehe auch

`select_lines`, `select_lines_longest`, `detect_edge_segments`, `select_shape`

Modul

Region processing

`select_lines` (: : RowBeginIn, ColBeginIn, RowEndIn, ColEndIn, Feature, Operation, Min, Max : RowBeginOut, ColBeginOut, RowEndOut, ColEndOut)

Auswahl von Linien gemäß verschiedener Kriterien.

`select_lines` wählt Linien anhand verschiedener Kriterien aus. Für jede Eingabelinie werden die angegebenen Merkmale (`Feature`) berechnet. Wenn jedes (`Operation` = ’and’) oder mindestens eines (`Operation` = ’or’) der so berechneten Merkmale in den vorgegebenen Grenzen (`Min`,`Max`) liegt, wird die Linie in die Ausgabe übernommen.

Bedingung: $Min_i \leq Feature_i(Linie) \leq Max_i$

Mögliche Werte für `Feature`:

’length’ (euklidische) Länge der Linie

’row’ Zeilenindex des Schwerpunkts

’column’ Spaltenindex des Schwerpunkts

’phi’ Orientierung der Linie ($-\frac{\pi}{2} < \varphi \leq \frac{\pi}{2}$)

Achtung

Wird nur ein Merkmal verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie übergeben werden.

Parameter

- ▷ **`RowBeginIn`** (input_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **`ColBeginIn`** (input_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **`RowEndIn`** (input_control) ..... line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **`ColEndIn`** (input_control) ..... line.end.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **`Feature`** (input_control) ..... string(-array) ⇝ *string*
  Gewünschte Merkmale.
  Defaultwert : ’length’
  Werteliste : `Feature` ∈ {’length’, ’row’, ’column’, ’phi’}
- ▷ **`Operation`** (input_control) ..... string ⇝ *string*
  Gewünschte Verknüpfung.
  Defaultwert : ’and’
  Werteliste : `Operation` ∈ {’and’, ’or’}

- ▷ **Min** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
  Untere Grenzen der Merkmale oder 'min'.
  Defaultwert : 'min'
- ▷ **Max** (input_control) . . . . . string(-array) ⇝ *string* / integer / real
  Obere Grenzen der Merkmale oder 'max'.
  Defaultwert : 'max'
- ▷ **RowBeginOut** (output_control) . . . . . line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.
- ▷ **ColBeginOut** (output_control) . . . . . line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.
- ▷ **RowEndOut** (output_control) . . . . . line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Ausgabelinien.
- ▷ **ColEndOut** (output_control) . . . . . line.end.x-array ⇝ *integer*
  Spaltenindizes der Endpunkte der Ausgabelinien.

Ergebnis

`select_lines` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_lines` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Mögliche Nachfolgerfunktionen

`set_line_width`, `disp_line`

Alternativen

`line_orientation`, `line_position`, `partition_lines`

Siehe auch

`partition_lines`, `select_lines_longest`, `detect_edge_segments`, `select_shape`

Modul

Region processing

select_lines_longest (: : RowBeginIn, ColBeginIn, RowEndIn, ColEndIn, Num : RowBeginOut, ColBeginOut, RowEndOut, ColEndOut)

Auswahl der längsten Eingabelinien.

`select_lines_longest` wählt aus den durch die Tupel `RowBeginIn`, `ColBeginIn`, `RowEndIn` und `ColEndIn` beschriebenen Eingabelinien die `Num` längsten Eingabelinien aus.

Parameter

- ▷ **RowBeginIn** (input_control) . . . . . line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **ColBeginIn** (input_control) . . . . . line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
- ▷ **RowEndIn** (input_control) . . . . . line.end.y-array ⇝ *integer*
  Zeilenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **ColEndIn** (input_control) . . . . . line.end.x-array ⇝ *integer*
  Spaltenkoordinaten der Endpunkte der Eingabelinien.
- ▷ **Num** (input_control) . . . . . integer ⇝ *integer*
  (Maximal) gewünschte Anzahl von Ausgabelinien.
  Defaultwert : 10
- ▷ **RowBeginOut** (output_control) . . . . . line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.
- ▷ **ColBeginOut** (output_control) . . . . . line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **RowEndOut** (output_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **ColEndOut** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenindizes der Endpunkte der Ausgabelinien.

Ergebnis

`select_lines_longest` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_lines_longest` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `edges_image`, `threshold`, `hysteresis_threshold`, `split_skeleton_region`, `split_skeleton_lines`

Mögliche Nachfolgerfunktionen

`set_line_width`, `disp_line`

Alternativen

`line_orientation`, `line_position`, `select_lines`, `partition_lines`

Siehe auch

`select_lines`, `partition_lines`, `detect_edge_segments`, `select_shape`

Modul

Region processing

6.2 Zugriff

approx_chain (: : Row, Column, MinWidthCoord, MaxWidthCoord, ThreshStart, ThreshEnd, ThreshStep, MinWidthSmooth, MaxWidthSmooth, MinWidthCurve, MaxWidthCurve, Weight1, Weight2, Weight3 : ArcCenterRow, ArcCenterCol, ArcAngle, ArcBeginRow, ArcBeginCol, LineBeginRow, LineBeginCol, LineEndRow, LineEndCol, Order)

Approximation einer Kontur durch Bögen und Linien.

Die Koordinaten einer Kurve werden durch eine Folge von Linien und Kreisbögen approximiert. Das Verfahren durchläuft dabei für bestimmte Parameter Wertebereiche. In der Parameterliste der Funktion werden die Grenzen dieser Bereiche explizit angegeben (MinWidthCoord ... MaxWidthCoord, ThreshStart ... ThreshEnd, MinWidthSmooth ... MaxWidthSmooth, MinWidthCurve ... MaxWidthCurve). Zusätzlich muß für den Parameterbereich des Schwellenwertes für spitze Ecken auch die Schrittweite angegeben werden (ThreshStep). Durch Einengung der durchlaufenen Bereiche kann die Laufzeit der Berechnung verkürzt, das Ergebnis aber eventuell verschlechtert werden.

Durch die Parameter Weight1, Weight2 und Weight3 kann angegeben werden, ob die gewünschte Gewichtung mehr auf Genauigkeit der Approximation oder Erhalt möglichst vieler großer oder möglichst weniger kleiner Segmente liegt. Für (Weight1,Weight2,Weight3) erzeugt also (1,0,0) eine sehr genaue Approximation (0,1,1) eine Approximation mit möglichst wenigen und großen Segmenten.

Das Ergebnis des Verfahren wird getrennt nach Bögen und Linien zurückgegeben. Ist man an der Reihenfolge der Segmente interessiert, so können die einzelnen Ergebniselemente sukzessive aus den Ergebnistupeln ausgelesen werden, wobei dem Rückgabeparameter Order die Reihenfolge entnommen werden kann (0: nächstes Element ist nächstes Liniensegment, 1: nächstes Element ist nächstes Bogensegment).

Achtung

Konturen, die eventuell nur aus einem Segment bestehen können, sollten auch mit einem Schwellenwertmaximum (ThreshEnd) > 1.0 untersucht werden, da ansonsten auf jeden Fall mindestens ein „Eckpunkt“ bestimmt wird.

Parameter

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenwert der Kontur.
Defaultwert : 32

▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
Spaltenwert der Kontur.
Defaultwert : 32

▷ **MinWidthCoord** (input_control) . . . . . real ⇝ *real*
Minimale Breite des Gaußoperators zur Koordinatenglättung (> 0.4).
Defaultwert : 0.5
Wertevorschläge : MinWidthCoord $\in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\}$
Typischer Wertebereich : $0.4 \leq$ MinWidthCoord ≤ 3.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **MaxWidthCoord** (input_control) . . . . . real ⇝ *real*
Maximale Breite des Gaußoperators zur Koordinatenglättung.
Defaultwert : 2.4
Wertevorschläge : MaxWidthCoord $\in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\}$
Typischer Wertebereich : $0.4 \leq$ MaxWidthCoord ≤ 3.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **ThreshStart** (input_control) . . . . . real ⇝ *real*
Minimaler Schwellenwert der Krümmung für die Akzeptierung einer Ecke (relativ zur größten auftretenden Krümmung).
Defaultwert : 0.3
Wertevorschläge : ThreshStart $\in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$
Typischer Wertebereich : $0.1 \leq$ ThreshStart ≤ 0.9 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **ThreshEnd** (input_control) . . . . . real ⇝ *real*
Maximaler Schwellenwert der Krümmung für die Akzeptierung einer Ecke (relativ zur größten auftretenden Krümmung).
Defaultwert : 0.9
Wertevorschläge : ThreshEnd $\in \{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$
Typischer Wertebereich : $0.1 \leq$ ThreshEnd ≤ 0.9 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **ThreshStep** (input_control) . . . . . real ⇝ *real*
Schrittweite für den Schwellenwertanstieg.
Defaultwert : 0.2
Wertevorschläge : ThreshStep $\in \{0.3, 0.4, 0.5\}$
Typischer Wertebereich : $0.1 \leq$ ThreshStep ≤ 0.9 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **MinWidthSmooth** (input_control) . . . . . real ⇝ *real*
Minimale Breite des Gaußoperators zur Glättung der Krümmungsfunktion (> 0.4).
Defaultwert : 0.5
Wertevorschläge : MinWidthSmooth $\in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\}$
Typischer Wertebereich : $0.4 \leq$ MinWidthSmooth ≤ 3.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **MaxWidthSmooth** (input_control) . . . . . real ⇝ *real*
Maximale Breite des Gaußoperators zur Glättung der Krümmungsfunktion.
Defaultwert : 2.4
Wertevorschläge : MaxWidthSmooth $\in \{0.5, 0.7, 1.0, 1.2, 1.5, 1.7\}$
Typischer Wertebereich : $0.4 \leq$ MaxWidthSmooth ≤ 3.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **MinWidthCurve** (input_control) . . . . . integer ⇝ *integer*
Minimale Breite des Kurvenbereichs zur Krümmungsbestimmung (> 0).
Defaultwert : 2
Wertevorschläge : MinWidthCurve $\in \{2, 5, 7\}$
Typischer Wertebereich : $1 \leq$ MinWidthCurve ≤ 12 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2

▷ **MaxWidthCurve** (input_control) . . . . . integer ⇝ *integer*
Maximale Breite des Kurvenbereichs zur Krümmungsbestimmung.
Defaultwert : 12
Wertevorschläge : MaxWidthCurve $\in \{2, 5, 7\}$
Typischer Wertebereich : $1 \leq$ MaxWidthCurve ≤ 20 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2

▷ **Weight1** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für die Approximationsgenauigkeit.
Defaultwert : 1.0
Wertevorschläge : Weight1 $\in \{0.0, 0.5, 1.0\}$
Typischer Wertebereich : $0.0 \leq$ Weight1 ≤ 1.0 (lin)
Minimale Schrittweite : 0.1
Empfohlene Schrittweite : 0.5

▷ **Weight2** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für große Segmente.
Defaultwert : 1.0
Wertevorschläge : Weight2 $\in \{0.0, 0.5, 1.0\}$
Typischer Wertebereich : $0.0 \leq$ Weight2 ≤ 1.0 (lin)
Minimale Schrittweite : 0.1
Empfohlene Schrittweite : 0.5

▷ **Weight3** (input_control) . . . . . real ⇝ *real*
Gewichtungsfaktor für kleine Segmente.
Defaultwert : 1.0
Wertevorschläge : Weight3 $\in \{0.0, 0.5, 1.0\}$
Typischer Wertebereich : $0.0 \leq$ Weight3 ≤ 1.0 (lin)
Minimale Schrittweite : 0.1
Empfohlene Schrittweite : 0.5

▷ **ArcCenterRow** (output_control) . . . . . arc.center.y-array ⇝ *integer*
Zeile des Mittelpunkts eines Bogens.

▷ **ArcCenterCol** (output_control) . . . . . arc.center.x-array ⇝ *integer*
Spalte des Mittelpunkts eines Bogens.

▷ **ArcAngle** (output_control) . . . . . arc.angle.rad-array ⇝ *real*
Winkel eines Bogens.

▷ **ArcBeginRow** (output_control) . . . . . arc.begin.y-array ⇝ *integer*
Zeile des Startpunkts eines Bogens.

▷ **ArcBeginCol** (output_control) . . . . . arc.begin.x-array ⇝ *integer*
Spalte des Startpunkts eines Bogens.

▷ **LineBeginRow** (output_control) . . . . . line.begin.y-array ⇝ *integer*
Zeile des Startpunkts einer Linie.

▷ **LineBeginCol** (output_control) . . . . . line.begin.x-array ⇝ *integer*
Spalte des Startpunkts einer Linie.

▷ **LineEndRow** (output_control) . . . . . line.end.y-array ⇝ *integer*
Zeile des Endpunkts einer Linie.

▷ **LineEndCol** (output_control) . . . . . line.end.x-array ⇝ *integer*
Spalte des Endpunkts einer Linie.

▷ **Order** (output_control) . . . . . integer-array ⇝ *integer*
Reihenfolge von Linien- (Wert 0) und Bogensegmenten (Wert1).

Beispiel

```
/* read edge image  */
read_image(&Image,"fig1_kan");
/* construct edge region  */
hysteresis_threshold(Image,&RK1,64,255,40,1);
connection(RK1,&Rand);
/* fetch chain code  */
T_get_region_contour(Rand,&Rows,&Columns);
firstline = get_i(Tline,0);
firstcol = get_i(Tcol,0);
/* approximation with lines and circular arcs */
set_d(t1,0.4,0);
set_d(t2,2.4,0);

set_d(t3,0.3,0);
set_d(t4,0.9,0);

set_d(t5,0.2,0);

set_d(t6,0.4,0);
set_d(t7,2.4,0);

set_i(t8,2,0);
set_i(t9,12,0);

set_d(t10,1.0,0);
set_d(t11,1.0,0);
set_d(t12,1.0,0);

T_approx_chain(Rows,Columns,t1,t2,t3,t4,t5,t6,t7,t8,t9,t10,t11,t12,
               &Bzl,&Bzc,&Br,&Bwl,&Bwc,&Ll0,&Lc0,&Ll1,&Lc1,&order);
nob = length_tuple(Bzl);
nol = length_tuple(Ll0);
/* draw lines and arcs */
set_i(WindowHandleTuple,WindowHandle,0) ;
set_line_width(WindowHandle,4);
if (nob>0) T_disp_arc(Bzl,Bzc,Br,Bwl,Bwc);
set_line_width(WindowHandle,1);
if (nol>0) T_disp_line(WindowHandleTuple,Ll0,Lc0,Ll1,Lc1);
```

Ergebnis

approx_chain liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

approx_chain ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

sobel_amp, edges_image, get_region_contour, threshold, hysteresis_threshold

Mögliche Nachfolgerfunktionen

set_line_width, disp_arc, disp_line

Alternativen

get_region_polygon, approx_chain_simple

Siehe auch

get_region_chain, smallest_circle, disp_circle, disp_line

Modul

Region processing

approx_chain_simple (: : Row, Column : ArcCenterRow, ArcCenterCol, ArcAngle, ArcBeginRow, ArcBeginCol, LineBeginRow, LineBeginCol, LineEndRow, LineEndCol, Order)

Approximation einer Kontur durch Bögen und Linien.

Die Koordinaten einer Kurve werden durch eine Folge von Linien und Kreisbögen approximiert.

Das Ergebnis des Verfahren wird getrennt nach Bögen und Linien zurückgegeben. Ist man an der Reihenfolge der Segmente interessiert, so können die einzelnen Ergebniselemente sukzessive aus den Ergebnistupeln ausgelesen werden, wobei dem Rückgabeparameter Order die Reihenfolge entnommen werden kann (0: nächstes Element ist nächstes Liniensegment, 1: nächstes Element ist nächstes Bogensegment).

approx_chain_simple verhält sich ähnlich wie approx_chain nur bei approx_chain_simple sind die fehlende Parameter intern wie folgt belegt: 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 ⇝ *integer*
  Zeilenwert der Kontur.
  Defaultwert : 32
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenwert der Kontur.
  Defaultwert : 32
- ▷ **ArcCenterRow** (output_control) ..... arc.center.y-array ⇝ *integer*
  Zeile des Mittelpunkts eines Bogens.
- ▷ **ArcCenterCol** (output_control) ..... arc.center.x-array ⇝ *integer*
  Spalte des Mittelpunkts eines Bogens.
- ▷ **ArcAngle** (output_control) ..... arc.angle.rad-array ⇝ *real*
  Winkel eines Bogens.
- ▷ **ArcBeginRow** (output_control) ..... arc.begin.y-array ⇝ *integer*
  Zeile des Startpunkts eines Bogens.
- ▷ **ArcBeginCol** (output_control) ..... arc.begin.x-array ⇝ *integer*
  Spalte des Startpunkts eines Bogens.
- ▷ **LineBeginRow** (output_control) ..... line.begin.y-array ⇝ *integer*
  Zeile des Startpunkts einer Linie.
- ▷ **LineBeginCol** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spalte des Startpunkts einer Linie.
- ▷ **LineEndRow** (output_control) ..... line.end.y-array ⇝ *integer*
  Zeile des Endpunkts einer Linie.
- ▷ **LineEndCol** (output_control) ..... line.end.x-array ⇝ *integer*
  Spalte des Endpunkts einer Linie.
- ▷ **Order** (output_control) ..... integer-array ⇝ *integer*
  Reihenfolge von Linien- (Wert 0) und Bogensegmenten (Wert1).

Beispiel

```
/* read edge image */
read_image(&Image,"fig1_kan");
/* construct edge region */
hysteresis_threshold(Image,&RK1,64,255,40,1);
connection(RK1,&Rand);
/* fetch chain code */
T_get_region_contour(Rand,&Rows,&Columns);
firstline = get_i(Tline,0);
firstcol = get_i(Tcol,0);
/* approximation with lines and circular arcs */
T_approx_chain_simple(Rows,Columns,
                &Bzl,&Bzc,&Br,&Bwl,&Bwc,&Ll0,&Lc0,&Ll1,&Lc1,&order);
```

```
nob = length_tuple(Bzl);
nol = length_tuple(Ll0);
/* draw lines and arcs */
set_i(WindowHandleTuple,WindowHandle,0) ;
set_line_width(WindowHandle,4);
if (nob>0) T_disp_arc(Bzl,Bzc,Br,Bwl,Bwc);
set_line_width(WindowHandle,1);
if (nol>0) T_disp_line(WindowHandleTuple,Ll0,Lc0,Ll1,Lc1);
```

Ergebnis

`approx_chain_simple` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`approx_chain_simple` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `edges_image`, `get_region_contour`, `threshold`, `hysteresis_threshold`

Mögliche Nachfolgerfunktionen

`set_line_width`, `disp_arc`, `disp_line`

Alternativen

`get_region_polygon`, `approx_chain`

Siehe auch

`get_region_chain`, `smallest_circle`, `disp_circle`, `disp_line`

Modul

Region processing

Kapitel 7

Morphologie

7.1 Grauwerte

dual_rank (Image : ImageRank : MaskType, Radius, ModePercent, Margin :)

Opening, Median und Closing mit Kreis- oder Rechteckmaske.

dual_rank führt eine nichtlineare Transformation der Grauwerte aller Eingabebilder (Image) durch. Als strukturierende Elemente können Kreise oder Quadrate verwendet werden. dual_rank ist die Hintereinanderschaltung von zwei Aufrufen von rank_image. Bei dem ersten Aufruf wird der Ranggrauwert mit dem angegeben Rang (ModePercent) berechnet. Das Ergebnis dieser Berechnung ist die Eingabe zu einem erneuten Aufruf von rank_image, wobei diesmal der Rangwert $100-$ModePercent verwendet wird.

Es kann bei der Filterung zwischen verschiedenen Randbehandlungen (Margin) gewählt werden:

0...255	Bildpunkte außerhalb der Bildränder werden als konstant (mit dem angegebenen Grauwert) angenommen.
-1	Fortsetzung der Randpunkte.
-2	zyklische Fortsetzung der Bildränder.
-3	Spiegelung der Bildpunkte an den Bildrändern.

Die Berechnung einer Rangfilterung erfolgt nach folgendem Schema: Die angebene Maske wird so über das zu filterndes Bild geschoben, daß der Schwerpunkt der Maske alle Bildpunkte einmal berührt. Für jeden solchen Bildpunkt werden alle Nachbarpunkte, die von der Maske überdeckt werden, bzgl. ihrer Grauwerte aufsteigend sortiert. Jede solche sortierte Sequenz von Grauwerten enthält somit genau so viele Grauwerte, wie die Maske Bildpunkte hat. Aus diesen Sequenzen wird das n-größte Element (= ModePercent, Rangwert von $0 \ldots 100$ in Prozent) ausgewählt und als Ergebnisgrauwert bei dem jeweiligen Ausgabebild eingetragen.

Hat ModePercent den Wert 0, so erhält man ein Grauwert-Opening (gray_opening). Setzt man ModePercent auf 50, so erhält man den bekannten Medianfilter (median_image), zweimal hintereinander angewandt. Bei dem Wert 100 berechnet dual_rank das Grauwert-Closing (gray_closing). Mit den übrigen Parametern kann fließend zwischen zwischen diesen Operationen „überblendet“ werden.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Zu filterndes Bild.
- ▷ **ImageRank** (output_object) ..... multichannel-image(-array) ⇝ *Hobject*
  Gefiltertes Bild.
- ▷ **MaskType** (input_control) ..... string ⇝ *string*
  Form der Maske.
  Defaultwert : 'circle'
  Werteliste : MaskType ∈ {'circle', 'rectangle'}

▷ **Radius** (input_control) ..... integer ⇝ *integer*
Radius der FIltermaske.
Defaultwert : 1
Wertevorschläge : Radius ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 15, 19, 25, 31, 39, 47, 59}
Typischer Wertebereich : 1 ≤ Radius ≤ 101
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2

▷ **ModePercent** (input_control) ..... integer ⇝ *integer*
Modus des Filters: Bei 0 erhält man ein Grauwert-Opening, bei 50 einen Median und bei 100 ein Grauwert-Closing.
Defaultwert : 10
Wertevorschläge : ModePercent ∈ {0, 2, 5, 10, 15, 20, 40, 50, 60, 80, 85, 90, 95, 98, 100}
Typischer Wertebereich : 0 ≤ ModePercent ≤ 100
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2

▷ **Margin** (input_control) ..... integer ⇝ *integer*
Randbehandlung: 0...255 (konstant), -1 (Randpunkte fortgesetzt), -2 (zyklische Fortsetzung), -3 (Spiegelung).
Defaultwert : -3
Typischer Wertebereich : -3 ≤ Margin ≤ 255

Beispiel

```
read_image(Image,'fabrik')
dual_rank(Image,ImageOpening,'circle',10,10,-3)
disp_image(ImageOpening,WindowHandle).
```

Komplexität

Pro Bildpunkt: O($\sqrt{F} * 10$) mit F = Fläche des strukturierenden Elementes.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert dual_rank den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

dual_rank ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

read_image

Mögliche Nachfolgerfunktionen

threshold, dyn_threshold, sub_image, regiongrowing

Alternativen

rank_image, gray_closing, gray_opening, median_image

Siehe auch

gen_circle, gen_rectangle1, gray_erosion_rect, gray_dilation_rect, sigma_image

Literatur

W. Eckstein, O. Munkelt “Extracting Objects from Digital Terrain Model” Remote Sensing and Reconstruction for Threedimensional Objects and Scenes, SPIE Symposium on Optical Science, Engeneering, and Instrumentation, July 1995, San Diego

Modul

Image filters

gen_disc_se (: SE : Width, Height, Smax :)

Ellipsoidförmige strukturierendes Element für die Grauwertmorphologie.

gen_disc_se erzeugt ein ellipsoidförmiges strukturierendes Element (SE) für die Grauwertmorphologie von Bildern. Die Parameter Width und Height geben die Größe der beiden Achsen der Ellipse an. Der Wert Smax gibt den maximalen Grauwert an, den das strukturierende Element annimmt. Zur Erzeugung von strukturierenden Elementen siehe read_gray_se.

Parameter

- **SE** (output_object) ..... image ⇝ *Hobject* : byte
  Erzeugtes strukturierendes Element.
- **Width** (input_control) ..... integer ⇝ *integer*
  Breite des erzeugten strukturierenden Elements.
  Defaultwert : 5
  Wertevorschläge : Width ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}
  Typischer Wertebereich : 0 ≤ Width ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- **Height** (input_control) ..... integer ⇝ *integer*
  Höhe des erzeugten strukturierenden Elements.
  Defaultwert : 5
  Wertevorschläge : Height ∈ {0, 1, 2, 3, 4, 5, 10, 15, 20}
  Typischer Wertebereich : 0 ≤ Height ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- **Smax** (input_control) ..... integer ⇝ *integer*
  Maximaler Grauwert des erzeugten strukturierenden Elements.
  Defaultwert : 0
  Wertevorschläge : Smax ∈ {0, 1, 2, 5, 10, 20, 30, 40}
  Typischer Wertebereich : 0 ≤ Smax ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_disc_se den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_disc_se ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

gray_erosion, gray_dilation, gray_opening, gray_closing, gray_tophat, gray_bothat

Alternativen

read_gray_se

Siehe auch

read_image, paint_region, paint_gray, crop_part

Modul

Image filters

gray_bothat (Image, SE : ImageBotHat : :)

Bottom-Hat-Transformation auf Bildern.

gray_bothat führt eine Bottom-Hat-Transformation auf dem Bild Image mit dem strukturierenden Element SE durch. Die Bottom-Hat-Transformation eines Bildes i mit einem strukturierenden Element s ist definiert als

$$bothat(i, s) = (i \bullet s) - i,$$

d.h. die Differenz des Closings des Bildes mit s und des Bildes (siehe gray_closing). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **SE** (input_object) ..... image ⇝ *Hobject* : byte
  Strukturierendes Element.
- ▷ **ImageBotHat** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Bottom Hat.

Ergebnis

gray_bothat liefert genau dann den Wert 2 (H_MSG_TRUE), wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_bothat ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

read_gray_se, gen_disc_se

Mögliche Nachfolgerfunktionen

threshold

Alternativen

gray_closing

Siehe auch

gray_tophat, top_hat, gray_erosion_rect, sub_image

Modul

Image filters

gray_closing (Image, SE : ImageClosing : :)

Grauwert-Closing auf Bildern.

gray_closing führt ein Grauwert-Closing auf dem Bild Image mit dem strukturierenden Element SE durch. Das Grauwert-Closing eines Bildes i mit einem strukturierenden Element s ist definiert als

$$i \bullet s = (i \oplus s) \ominus s \ ,$$

d.h. eine Dilatation des Bildes mit s gefolgt von einer Erosion mit s (siehe gray_dilation und gray_erosion). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **SE** (input_object) ..... image ⇝ *Hobject* : byte
  Strukturierendes Element.
- ▷ **ImageClosing** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Ergebnisbild.

Ergebnis

gray_closing liefert genau dann den Wert 2 (H_MSG_TRUE), wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_closing ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

read_gray_se

Alternativen

dual_rank

Siehe auch

closing, gray_dilation, gray_erosion

Modul

Image filters

gray_dilation (Image, SE : ImageDilation : :)

Grauwert-Dilatation auf Bildern.

gray_dilation führt eine Grauwertdilatation auf dem Bild Image mit dem strukturierenden Element SE durch. Die Grauwertdilatation eines Bildes i mit einem strukturierenden Element s an der Pixel-Position x ist definiert durch:

$$(i \oplus s)(x) = \max\{f(x-z) + s(z) | z \in S\}$$

Hierbei ist S der Definitionsbereich des strukturierenden Elements s, d.h. die Pixel, an denen $s(z) > 0$ ist (siehe read_gray_se).

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **SE** (input_object) ..... image ⇝ *Hobject* : byte
  Strukturierendes Element.
- ▷ **ImageDilation** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Ergebnisbild.

Ergebnis

gray_dilation liefert genau dann den Wert 2 (H_MSG_TRUE), wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_dilation ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

read_gray_se

Mögliche Nachfolgerfunktionen

sub_image, gray_erosion

Alternativen

gray_dilation_rect

Siehe auch

gray_opening, gray_closing, dilation1, gray_skeleton

Modul

Image filters

gray_dilation_rect (Image : ImageMax : MaskHeight, MaskWidth :)

Maximum der Grauwerte in einem Rechteck.

gray_dilation_rect transformiert die Grauwerte der Eingabebilder aus Image mit Hilfe einer Filtermaske (MaskHeight, MaskWidth), in der das Maximum der Grauwerte berechnet wird. Das Ergebnis wird in ImageMax zurückgeliefert. Die Steuerparameter MaskHeight,MaskWidth werden, wenn sie einen geraden Wert haben, auf den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte / direction / int4 / real
  Bilder, für deren Grauwerte die Maxima berechnet werden sollen.
- ▷ **ImageMax** (output_object) ..... image(-array) ⇝ *Hobject* : byte / direction / int4 / real
  Bilder, die die Maxima enthalten.

▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
Höhe der Filtermaske.
Defaultwert : 11
Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich : 3 ≤ MaskHeight ≤ 511
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskHeight)

▷ **MaskWidth** (input_control) ..... extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert : 11
Wertevorschläge : MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich : 3 ≤ MaskWidth ≤ 511
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskWidth)

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gray_dilation_rect den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_dilation_rect ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Siehe auch

gray_skeleton

Modul

Image filters

gray_erosion (Image, SE : ImageErosion : :)

Grauwert-Erosion auf Bildern.

gray_erosion führt eine Grauwerterosion auf dem Bild Image mit dem strukturierenden Element SE durch. Die Grauwerterosion eines Bildes i mit einem strukturierenden Element s an der Pixel-Position x ist definiert durch:

$$(i \ominus s)(x) = \min\{f(x+z) - s(z) | z \in S\}$$

Hierbei ist S der Definitionsbereich des strukturierenden Elements s, d.h. die Pixel, an denen $s(z) > 0$ ist (siehe read_gray_se).

Parameter

▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
Eingabebild.

▷ **SE** (input_object) ..... image ⇝ *Hobject* : byte
Strukturierendes Element.

▷ **ImageErosion** (output_object) ..... image(-array) ⇝ *Hobject* : byte
Ergebnisbild.

Ergebnis

gray_erosion liefert genau dann den Wert 2 (H_MSG_TRUE), wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_erosion ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

read_gray_se

Mögliche Nachfolgerfunktionen

`gray_dilation`, `sub_image`

Alternativen

`gray_erosion_rect`

Siehe auch

`gray_opening`, `gray_closing`, `erosion1`, `gray_skeleton`

Modul

Image filters

`gray_erosion_rect ( Image : ImageMin : MaskHeight, MaskWidth : )`

Minimum der Grauwerte in einem Rechteck.

`gray_erosion_rect` transformiert die Grauwerte der Eingabebilder aus `Image` mit Hilfe einer Filtermaske (`MaskHeight`, `MaskWidth`), in der das Minimum der Grauwerte berechnet wird. Das Ergebnis wird im Parameter `ImageMin` übergeben. Die Steuerparameter `MaskHeight`, `MaskWidth` werden, wenn sie einen geraden Wert haben, in den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / direction / int4 / real
  Bilder, für deren Grauwerte die Minima berechnet werden sollen.
- ▷ **`ImageMin`** (output_object) ..... image(-array) ⇝ *Hobject* : byte / direction / int4 / real
  Bilder, die die Minima enthalten.
- ▷ **`MaskHeight`** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 11
  Wertevorschläge : `MaskHeight` ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich : 3 ≤ `MaskHeight` ≤ 511 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(`MaskHeight`)
- ▷ **`MaskWidth`** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 11
  Wertevorschläge : `MaskWidth` ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich : 3 ≤ `MaskWidth` ≤ 511 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(`MaskWidth`)

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gray_erosion_rect` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gray_erosion_rect` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Siehe auch

`gray_dilation_rect`

Modul

Image filters

gray_opening (Image, SE : ImageOpening : :)

Grauwert-Opening auf Bildern.

gray_opening führt ein Grauwert-Opening auf dem Bild Image mit dem strukturierenden Element SE durch. Das Grauwert-Opening eines Bildes i mit einem strukturierenden Element s ist definiert als

$$i \circ s = (i \ominus s) \oplus s \ ,$$

d.h. eine Erosion des Bildes mit s gefolgt von einer Dilatation mit s (siehe gray_erosion und gray_dilation). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **SE** (input_object) ..... image ⇝ *Hobject* : byte
  Strukturierendes Element.
- ▷ **ImageOpening** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Ergebnisbild.

Ergebnis

gray_opening liefert genau dann den Wert 2 (H_MSG_TRUE), wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_opening ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

read_gray_se

Alternativen

dual_rank

Siehe auch

opening, gray_dilation, gray_erosion

Modul

Image filters

gray_range_rect (Image : ImageResult : MaskHeight, MaskWidth :)

Maximale Amplitude der Grauwerte in einem Rechteck.

gray_range_rect transformiert die Grauwerte der Eingabebiler aus Image mit Hilfe einer Filtermaske (MaskHeight, MaskWidth), in der die Spanne $(\max - \min)$ der Grauwerte berechnet wird. Das Ergebnis wird in die Ausgabebilder ImageResult eingetragen. Die Steuerparameter MaskHeight, MaskWidth werden, wenn sie einen geraden Wert haben, in den nächstgrößeren ungeraden Wert transformiert. An den Bildrändern wird eine Spiegelung der Randpunkte durchgeführt.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte / direction / int4 / real
  Bilder, für deren Grauwerte die Amplitude berechnet werden sollen.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *Hobject* : byte / direction / int4 / real
  Bilder, die das Ergebnis enthalten.
- ▷ **MaskHeight** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 11
  Wertevorschläge : MaskHeight ∈ {3, 5, 7, 9, 11, 13, 15}
  Typischer Wertebereich : 3 ≤ MaskHeight ≤ 511 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : odd(MaskHeight)

▷ **MaskWidth** (input_control) . . . . . extent.x ⇝ *integer*
Breite der Filtermaske.
Defaultwert : 11
Wertevorschläge : MaskWidth ∈ {3, 5, 7, 9, 11, 13, 15}
Typischer Wertebereich : 3 ≤ MaskWidth ≤ 511 (lin)
Minimale Schrittweite : 2
Empfohlene Schrittweite : 2
Restriktion : odd(MaskWidth)

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gray_range_rect den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_range_rect ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*, *Domänen-Ebene*).

Alternativen

gray_dilation_rect, gray_erosion_rect, sub_image

Modul

Image filters

gray_tophat (Image, SE : ImageTopHat : :)

Top-Hat-Transformation auf Bildern.

gray_tophat führt eine Top-Hat-Transformation auf dem Bild Image mit dem strukturierenden Element SE durch. Die Top-Hat-Transformation eines Bildes i mit einem strukturierenden Element s ist definiert als

$$tophat(i, s) = i - (i \circ s),$$

d.h. die Differenz des Bildes und seines Openings mit s (siehe gray_opening). Zur Erzeugung von strukturierenden Elementen siehe read_gray_se und gen_disc_se.

Parameter

▷ **Image** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte
Eingabebild.
▷ **SE** (input_object) . . . . . image ⇝ *Hobject* : byte
Strukturierendes Element.
▷ **ImageTopHat** (output_object) . . . . . image(-array) ⇝ *Hobject* : byte
Ergebnisbild.

Ergebnis

gray_tophat liefert genau dann den Wert 2 (H_MSG_TRUE), wenn das strukturierende Element keine leere Region ist. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gray_tophat ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

read_gray_se, gen_disc_se

Mögliche Nachfolgerfunktionen

threshold

Alternativen

gray_opening

Siehe auch

gray_bothat, top_hat, gray_erosion_rect, sub_image

Modul

Image filters

read_gray_se (: SE : FileName :)

Laden von strukturierenden Elementen für die Grauwertmorphologie.

read_gray_se lädt ein strukturierendes Element für die Grauwertmorphologie von Datei ein. Die Dateinamen für strukturierende Elemente müssen mit '.gse' (für grayscale structuring element) enden. Dieses Suffix wird von read_gray_se automatisch an den übergebenen Dateinamen angehängt, muß also nicht mit übergeben werden. Die Daten müssen im ASCII-Format vorliegen und wie folgt aufgebaut sein: Die ersten zwei Zahlen der Datei geben die Breite und Höhe des strukturierenden Elements an. Diese stellen ein Rechteck dar, das das strukturierende Element vollständig umschließt. Beide Werte müssen größer als 0 sein. Danach folgen Breite*Höhe ganze Zahlen, die durch Leerzeichen getrennt sein müssen, mit folgender Interpretation: Werte, die kleiner als 0 sind, werden als nicht zur Region des strukturierenden Elements gehörig angesehen, d.h. sie werden bei den morphologischen Operationen nicht berücksichtigt. Dies erlaubt die Erzeugung von irregulär geformten und nicht zusammenhängenden strukturierenden Elementen. Alle anderen Werte werden als entsprechende Grauwerte für die Morphologie interpretiert. Die Darstellung von strukturierenden Elementen erfolgt intern als Byte-Bild. Dazu werden negative Elemente des strukturierenden Elements auf den Grauwert 0 abgebildet und alle anderen Grauwerte um 1 erhöht. Daher kann man auch gewöhnliche Bilder als strukturierende Elemente verwenden. Dabei sollte allerdings darauf geachtet werden, daß die Bildabmessungen nicht zu groß sind, da die Laufzeit proportional zur Fläche des Bildes multipliziert mit der Fläche des strukturierenden Elements ist.

Parameter

- **SE** (output_object) ..... image ⇝ *Hobject* : byte
  Erzeugtes strukturierendes Element.
- **FileName** (input_control) ..... filename ⇝ *string*
  Name der einzulesenden Datei.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_gray_se den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, wird 5 (H_MSG_FAIL) zurückgeliefert. Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_gray_se ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

gray_erosion, gray_dilation, gray_opening, gray_closing, gray_tophat, gray_bothat

Alternativen

gen_disc_se

Siehe auch

read_image, paint_region, paint_gray, crop_part

Modul

Image filters

7.2 Region

bottom_hat (Region, StructElement : RegionBottomHat : :)

Lücken zwischen Regionen.

bottom_hat berechnet die Bottom-Hat-Operation, indem zunächst eine Closing-Operation mit StructElement (siehe closing) durchgeführt wird. Von diesem Zwischenergebnis wird die Originalregion abgezogen. Im Gegensatz zu closing, das aus Einzelregionen bei entsprechender Voraussetzung mehrere Regionen zu einer größeren zusammenschließt, bestimmt bottom_hat die Region, welche für diesen Zusammenschluß notwendig ist.

Die Position von StructElement ist ohne Bedeutung, da die Closing-Operation translationsinvariant bzgl. StructElement ist.

Das struktuierende Element (StructElement) kann mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionBottomHat** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis des Bottom-Hat-Operators.

Beispiel

```
read_image (Image,'/bilder/name.ext')
threshold (Image,Regions,128,255)
gen_circle (Circle,0,0,16)
bottom_hat (Regions,Circle,RegionBottomHat).
```

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `bottom_hat` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`bottom_hat` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`closing`, `difference`

Siehe auch

`top_hat`, `morph_hat`, `gray_bothat`, `opening`

Modul

Morphology

boundary (Region : RegionBorder : BoundaryType :)

Reduktion einer Region auf deren Rand.

`boundary` berechnet die Kontur einer Region mit Hilfe morphologischer Operationen. Durch den Parameter `BoundaryType` wird zwischen drei Arten der Randerzeugung unterschieden:

’inner’, ’inner_filled’ und ’outer’.

`boundary` berechnet für jede Region die zugehörige Kontur. Die Ergebnisregionen bestehen nur aus einem minimalen Rand der Eingaberegionen. Bei dem Parameterwert ’inner’ liegt die Kontur innerhalb der Eingaberegionen, bei ’outer’ außerhalb. Im Modus ’inner_filled’ werden zusätzlich Hohlflächen unterdrückt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die auf den Rand reduziert werden.
- ▷ **RegionBorder** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnisregionen.

▷ **BoundaryType** (input_control) . . . . . string ⇝ *string*
Art der Regionenrandes.
Defaultwert : 'inner'
Werteliste : BoundaryType ∈ {'inner', 'outer', 'inner_filled'}

Beispiel

```
#include "HalconCpp.h"

main()
{
  HWindow  w;
  HRegion  circ1 = HRegion::GenCircle (20, 10, 10.5);

  circ1.Display (w);
  w.Click ();

  HRegion marg1 = circ1.Boundary ("inner");
  w.SetColor ("red");
  marg1.Display (w);
  w.Click ();

  return(0);
}
```

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(3\sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion boundary den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

boundary ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

dilation_circle, erosion_circle, difference

Siehe auch

fill_up

Modul

Morphology

closing (Region, StructElement : RegionClosing : :)

Schließen von Lücken.

Die Hintereinanderschaltung von Dilatation und Minkowski-Subtraktion definiert die morphologische Grundfunktion `closing`. `closing` dient zum Glätten von Regionenrändern. Löcher in Regionen die kleiner als `StructElement` sind, werden geschlossen. Bei allen `closing`-Varianten verschmelzen die einzelnen Regionen nicht. `closing` verwendet intern als Bezugspunkt des strukturierenden Elementes `StructElement` den Schwerpunkt. Die Position von `StructElement` ist ohne Bedeutung, da die Closing-Operation translationsinvariant bzgl. `StructElement` ist.

`StructElement` kann mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points` etc. erzeugt werden.

Achtung

`closing` wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muß vorher ein `union1` oder `union2` ausgeführt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen auf die die Closing-Operation angewandt wird.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element (lageinvariant).
- ▷ **`RegionClosing`** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen mit aufgefüllten Lücken.

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

main()
{
  cout << "Reproduction of 'closing ()' using " << endl;
  cout << "'dilation()' and 'minkowski_sub1()'" << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegion      circ = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs = (img >= 128).Connection();

  HRegionArray dilreg = regs.Dilation1 (circ, 1);
  HRegionArray minsub = dilreg.MinkowskiSub1 (circ, 1);

                        img.Display (w);      w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();
  w.SetColor ("green"); dilreg.Display (w);   w.Click ();
  w.SetColor ("blue");  minsub.Display (w);   w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot \sqrt{F2}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `closing` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`

- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`closing` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`closing_circle`, `closing_golay`

Siehe auch

`dilation1`, `erosion1`, `opening`, `minkowski_sub1`

Modul

Morphology

`closing_circle` (Region : RegionClosing : Radius :)

Closing mit einer Kreismaske.

Das Funktionsverhalten von `closing_circle` ist analog zu `closing` d.h. es werden die Ränder einer Eingaberegion geglättet und Löcher innerhalb einer Region deren Größe kleiner ist als das kreisförmige, strukturierende Element mit dem `Radius`, geschlossen. Die `closing_circle`-Operation ist als die Hintereinanderschaltung von Dilatation und Minkowski-Subtraktion mit einer Kreismaske definiert.

Achtung

`closing_circle` wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muß vorher ein `union1` oder `union2` ausgeführt werden.

Parameter

▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen auf die der Closing-Operator angewandt wird.

▷ **`RegionClosing`** (output_object) ..... region(-array) ⇝ *Hobject*
Regionen mit aufgefüllten Lücken.

▷ **`Radius`** (input_control) ..... real ⇝ *real* / integer
Radius der Kreismaske.
Defaultwert : 3.5
Wertevorschläge : `Radius` ∈ {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}
Typischer Wertebereich : $0.5 \leq$ `Radius` ≤ 511.5 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 1.0

Beispiel

```
my_closing_circle(Hobject In, double Radius, Hobject *Out)
{
  Hobject  tmp, StructElement;
  gen_circle(StructElement,100.0,100.0,Radius);
  dilation1(In,StructElement,&tmp,1);
  minkowski_sub1(tmp,StructElement,Out,1);
  clear_obj(tmp); clear_obj(StructElement);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(4 \cdot \sqrt{F1} \cdot \texttt{Radius}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `closing_circle` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`closing_circle` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`rank_region`, `fill_up`, `closing`, `closing_circle`, `closing_golay`

Siehe auch

`dilation1`, `minkowski_sub1`, `erosion1`, `opening`

Modul

Morphology

`closing_golay` (Region : RegionClosing : GolayElement, Rotation :)

Closing-Operation mit Golay-Alphabet als strukturierendes Element.

`closing_golay` ist als Hintereinanderschaltung von Minkowski-Addition und Minkowski-Subtraktion definiert. Dabei wird zuerst die Minkowski-Addition der Eingaberegion (`Region`) mit dem durch `GolayElement` und `Rotation` ausgewählten strukturierenden Element des Golay-Alphabets durchgeführt. Mit dem Ergebnis der Minkowski-Addition und dem um 180° gedrehten strukturierenden Element, wird dann eine Minkowski-Subtraktion durchgeführt.

Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten

’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’.

Die Rotationsnummer (`Rotation`) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Durch `closing_golay` werden Löcher, die kleiner sind als das strukturierende Element, geschlossen und die Regionenränder geglättet. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Werte von `Rotation` zulässig sind. Bei einigen Werten für `Rotation` entsteht die identische Abbildung.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen,die verarbeitet werden sollen.
- ▷ **`RegionClosing`** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen, auf die die Closing-Operation angewandt wurde.
- ▷ **`GolayElement`** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : ’h’
  Werteliste : `GolayElement` ∈ {’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’}

▷ **Rotation** (input_control) ..... integer ⇝ *integer*
Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zuläßig.
Defaultwert : 0
Werteliste : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion closing_golay den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

closing_golay ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

closing

Siehe auch

erosion_golay, dilation_golay, opening_golay, hit_or_miss_golay, thinning_golay, thickening_golay, golay_elements

Modul

Morphology

closing_rectangle1 (Region : RegionClosing : Width, Height :)

Closing mit einer Rechtecksmaske.

closing_rectangle1 setzt sich aus der Abfolge der Funktionsaufrufe dilation_rectangle1 und erosion_rectangle1 zusammen. Die Größe des rechteckigen, strukturierenden Elementes wird durch die Parameter Width und Height bestimmt. Wie bei allen closing-Varianten werden die Ränder einer Eingaberegion geglättet und Löcher innerhalb einer Region deren Größe kleiner ist als das rechteckige strukturierende Element geschlossen.

Achtung

closing_rectangle1 wird für jede Region einzeln angewandt. Eine Verschmelzung von Regionen muß explizit mit Routinen wie union1 durchgeführt werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen die verarbeitet werden sollen.

▷ **RegionClosing** (output_object) ..... region(-array) ⇝ *Hobject*
Ergebnis des Closing-Operators.

▷ **Width** (input_control) ..... extent.x ⇝ *integer* / real
Breite des Rechtecks.
Defaultwert : 10
Wertevorschläge : Width ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
Typischer Wertebereich : 1 ≤ Width ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer* / real
Höhe des Rechtecks.
Defaultwert : 10
Wertevorschläge : Height ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
Typischer Wertebereich : 1 ≤ Height ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot log\_2(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion closing_rectangle1 den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

closing_rectangle1 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

closing

Siehe auch

dilation_rectangle1, erosion_rectangle1, opening_rectangle1, gen_rectangle1

Modul

Morphology

dilation1 (Region, StructElement : RegionDilation : Iterations :)

Ausdehnen von Regionen.

dilation1 berechnet die Dilatation der Eingaberegionen mit einem strukturierenden Element. Die Anwendung von dilation1 glättet die Ränder der Regionen. Gleichzeitig vergrößert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, daß vorher nicht zusammenhängende Regionen verschmolzen werden. Dennoch bleiben solche Regionen logisch verschiedene Region. Die Dilatation ist eine mengentheoretische Regionenoperation. Sie verwendet die Operation Vereinigung.

Seien M (StructElement) und R (Region) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{dilation1}(R, M) := \bigcup_{m \in M} t_{-\vec{v}_m}(R)$$

Es wird für jeden Punkt m in M eine Translation mit der Region R durchgeführt. Die Vereinigung über all diese Verschiebungen ist die Dilatation der Region R mit M. dilation1 entspricht der Prozedur minkowski_add1 mit dem Unterschied, daß bei dilation1 das strukturierende Element punktgespiegelt

wird. Die Position von `StructElement` ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Elemente ausgeführt werden sollen. Als Eingaberegion für die Iteration n wird die Ergebnisregion der Iteration $(n-1)$ verwendet. Aus der obigen Definition ergibt sich, daß bei einem leeren strukturierenden Element eine leere Region erzeugt wird.

Strukturierende Elemente (`StructElement`) können mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points` etc. erzeugt werden.

Achtung

Eine Dilatation führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung der Prozedur zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muß zuerst ein `union1` durchgeführt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen,die verarbeitet werden sollen.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element.
- ▷ **`RegionDilation`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Dilation-Operation.
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : $1 \leq$ `Iterations` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation1` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dilation1` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `add_channels`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `minkowski_add2`, `dilation2`, `dilation_golay`, `dilation_seq`

Siehe auch

`erosion1`, `erosion2`, `opening`, `closing`

Modul

Morphology

dilation2 (Region, StructElement : RegionDilation : Row, Column, Iterations :)

Ausdehnen von Regionen (mit Bezugspunkt).

dilation2 berechnet die Dilatation der Eingaberegionen mit einem strukturierenden Element (StructElement) und dem Bezugspunkt, der durch Row und Column charakterisiert wird. dilation2 entspricht der Prozedur dilation1 mit dem Unterschied, daß der Bezugspunkt des strukturierenden Elementes frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Elemente ausgeführt werden sollen. Als Eingaberegion für die Iteration n wird die Ergebnisregion der Iteration $(n-1)$ verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird eine leere Region erzeugt.

Strukturierende Elemente (StructElement) können mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Achtung

Eine Dilatation führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung der Prozedur zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muß zuerst ein union1 durchgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für Dilation-Operation.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Dilation-Operation.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Defaultwert : 0
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Defaultwert : 0
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations $\in \{1, 2, 3, 4, 5, 7, 11, 17, 25, 32, 64, 128\}$
  Typischer Wertebereich : $1 \leq$ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \text{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion dilation2 den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

dilation2 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

reduce_domain, add_channels, select_shape, area_center, connection

Alternativen

minkowski_add1, minkowski_add2, dilation1, dilation_golay, dilation_seq

Siehe auch

erosion1, erosion2, opening, closing

Modul

Morphology

dilation_circle (Region : RegionDilation : Radius :)

Dilatation mit einer Kreismaske.

dilation_circle führt eine Minkowski-Addition mit einer Kreismaske aus. Da die Kreismaske symmetrisch ist, ist dies identisch mit einer Dilatation. Die Größe des Kreises, der als strukturierendes Element verwendet wird, ist durch Radius angegeben.

Die Wirkung dieser Funktion ist das Vergrößern der Region, die Glättung der Ränder, sowie das Verschließen von Löchern innerhalb der Region, die kleiner sind als die Kreismaske. Sinnvollerweise wird der Radius auf Werte wie 3.5, 5.5 etc. gesetzt, um so Translationen der Region zu vermeiden. Bei ganzzahligen Radien hat der Kreis nämlich keinen ganzzahligen Schwerpunkt, der aber gerundet werden muß.

Achtung

dilation_circle wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muß vorher ein union1 oder union2 ausgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen auf die der Dilation-Operator angewandt wird.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen mit aufgefüllten Lücken.
- ▷ **Radius** (input_control) ..... real ⇝ *real* / integer
  Radius der Kreismaske.
  Defaultwert : 3.5
  Wertevorschläge : Radius ∈ {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}
  Typischer Wertebereich : 0.5 ≤ Radius ≤ 511.5 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

main()
{
  cout << "Reproduction of 'dilation_circle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Blue  = after dilation " << endl;
  cout << "Red   = before dilation " << endl;

  HByteImage img("monkey");
  HWindow    w;
```

```
  HRegion      circ   = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs   = (img >= 128).Connection();
  HRegionArray minadd = regs.MinkowskiAdd1 (circ, 1);

                        img.Display (w);      w.Click ();
  w.SetColor ("blue");  minadd.Display (w);   w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \texttt{Radius} \cdot \sqrt{F1}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_circle` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dilation_circle` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `minkowski_add2`, `expand_region`, `dilation1`, `dilation2`, `dilation_rectangle1`

Siehe auch

`gen_circle`, `erosion_circle`, `closing_circle`, `opening_circle`

Modul

Morphology

dilation_golay (Region : RegionDilation : GolayElement, Iterations, Rotation :)

Ausdehnen von Regionen (Golay).

`dilation_golay` berechnet die Dilatation der Eingaberegionen mit einem ausgewählten strukturierenden Element aus dem Golay-Alphabet, das durch `GolayElement` bestimmt ist. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’.

Die Rotationsnummer `Rotation` gibt dabei an, welche Rotation, bzw. ob die Vordergrund- (gerade) oder die Hintergrundvariante (ungerade Werte) des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt. Das strukturierende Element wird dabei über die zu verarbeitende Region (`Region`) geschoben. Für alle Positionen des strukturierenden Elementes, an denen sie sich mit dem Objekt schneidet, wird der Bezugspunkt (relativ zum strukturierenden Element) in die Ausgaberegion aufgenommen. Es wird also die Vereinigungsmenge über alle Verschiebungen des strukturierenden Elementes über die Region gebildet.

Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die Iteration n wird die Ergebnisregion der Iteration $(n-1)$ verwendet.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Werte von `Rotation` zulässig sind. Bei einigen Werten für `Rotation` entsteht die identische Abbildung.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`RegionDilation`** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Dilation-Operation.
- ▷ **`GolayElement`** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : `GolayElement` ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ `Iterations` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Rotation`** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zuläßig.
  Defaultwert : 0
  Werteliste : `Rotation` ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_golay` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dilation_golay` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`dilation1`, `dilation2`, `dilation_seq`

Siehe auch

`erosion_golay`, `opening_golay`, `closing_golay`, `hit_or_miss_golay`, `thinning_golay`, `thickening_golay`, `golay_elements`

Modul

Morphology

dilation_rectangle1 (Region : RegionDilation : Width, Height :)

Dilatation mit einem Rechteck.

dilation_rectangle1 führt eine Dilatation auf der/den angegebenen Region(en) Region durch. Das strukturierende Element für diese Operation bildet ein Rechteck mit der Größe Width × Height. Die Funktion führt zu einer Expansion der Region und zum Schließen von Löchern, die kleiner als das angegebene Rechteck sind, innerhalb der Regionen.

dilation_rectangle1 ist eine sehr schnelle Operation, da die Höhe des Rechtecks nur logarithmisch, die Breite gar nicht in die Komplexitat eingeht. Dies führt auch bei sehr großen Rechtecken (Kante > 100) zu einem sehr gutem Laufzeitverhalten.

Achtung

dilation_rectangle1 wird für jede Region einzeln angewandt. Sollen Lücken zwischen den Regionen geschlossen werden, dann muß vorher ein union1 oder union2 ausgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Dilation-Operation.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Rechtecks.
  Defaultwert : 10
  Wertevorschläge : Width ∈ {1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100, 150, 200}
  Typischer Wertebereich : 1 ≤ Width ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Rechtecks.
  Defaultwert : 10
  Wertevorschläge : Height ∈ {1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100, 150, 200}
  Typischer Wertebereich : 1 ≤ Height ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

main()
{
  cout << "Reproduction of 'dilation_rectangle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Blue  = after dilation " << endl;
  cout << "Red   = after segmentation " << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegionArray regs   = (img >= 220).Connection();
  HRegionArray dilreg = regs.DilationRectangle1 (2, 4);

                        img.Display (w);      w.Click ();
  w.SetColor ("blue");  dilreg.Display (w);   w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();

  return(0);
}
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot ld(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_rectangle1` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dilation_rectangle1` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `minkowski_add2`, `expand_region`, `dilation1`, `dilation2`, `dilation_circle`

Siehe auch

`gen_rectangle1`, `gen_region_polygon_filled`

Modul

Morphology

dilation_seq (Region : RegionDilation : GolayElement, Iterations :)

Ausdehnen von Regionen (sequentiell).

`dilation_seq` berechnet die sequentielle Dilatation der Eingaberegionen mit dem durch `GolayElement` ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt `dilation_seq` die Prozedur `dilation_golay` mit allen Rotationen des strukturierenden Elements so oft durch, wie der Parameter `Iterations` angibt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

’l’, ’d’, ’c’, ’f’, ’h’, ’k’.

Für die Skelettbildung werden meistens die Elemente ’l’ und ’m’ verwendet. Es werden nur die „Vordergrund Elemente“ des Golayalphabets (gerade Rotationsnummern) verwendet. Die Elemente ’i’ und ’e’ erzeugen hier die identische Abbildung, d.h. sie verändern die Eingaberegion nicht. Die Elemente ’l’, ’m’ und ’f2’ sind im Vordergrund identisch. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionDilation** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Dilation-Operation.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : ’h’
  Werteliste : `GolayElement` ∈ {’l’, ’d’, ’c’, ’f’, ’h’, ’k’}

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert : 1
Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich : 1 ≤ `Iterations` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\texttt{Iterations} \cdot 20 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `dilation_seq` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dilation_seq` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`dilation1`, `dilation2`, `dilation_golay`

Siehe auch

`erosion_seq`, `hit_or_miss_seq`, `thinning_seq`

Modul

Morphology

erosion1 (Region, StructElement : RegionErosion : Iterations :)

Verdünnung von Regionen.

`erosion1` berechnet die Erosion der Eingaberegionen mit einem strukturierenden Element. Die Anwendung von `erosion1` glättet die Ränder der Regionen. Gleichzeitig verkleinert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, daß vorher zusammenhängende Regionen getrennt werden. Dennoch bleiben solche Regionen logisch eine Region. Die Erosion ist eine mengentheoretische Regionenoperation. Sie verwendet die Operation Durchschnitt.

Seien M (`StructElement`) und R (`Region`) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{erosion1}(R, M) := \bigcap_{m \in M} t_{-\vec{v}_m}(R).$$

Es wird für jeden Punkt aus M eine Translation mit der Region R durchgeführt. Der Durchschnitt über all diesen Verschiebungen ist die Erosion der Region R mit M. `erosion1` entspricht der Prozedur `minkowski_sub1` mit dem Unterschied, daß bei `erosion1` das strukturierende Element punktgespiegelt wird. Die Position von `StructElement` ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet. Bei Verwendung eines leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (`StructElement`) können mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points`, etc. erzeugt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für die Erosion.
- ▷ **`RegionErosion`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Erosion.
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ `Iterations` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `erosion1` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`erosion1` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`connection`, `reduce_domain`, `select_shape`, `area_center`

Alternativen

`minkowski_sub1`, `minkowski_sub2`, `erosion2`, `erosion_golay`, `erosion_seq`

Siehe auch

`transpose_region`

Modul

Morphology

`erosion2` `( Region, StructElement : RegionErosion : Row, Column, Iterations : )`

Verdünnung von Regionen (mit Bezugspunkt).

erosion2 berechnet die Erosion der Eingaberegionen mit einem strukturierenden Element StructElement und einem Bezugspunkt, der duch Row und Column bestimmt ist. Die Prozedur führt eine Regionentransformation durch. erosion2 entspricht der Prozedur erosion1 mit dem Unterschied, daß bei erosion2 der Bezugspunkt frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für die Erosion
- ▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Erosion.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ Row ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunkts.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ Column ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}$
  Typischer Wertebereich : $1 \leq$ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion erosion2 den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

erosion2 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_sub2`, `minkowski_sub1`, `erosion1`, `erosion_golay`, `erosion_seq`

Siehe auch

`transpose_region`, `gen_circle`, `gen_rectangle2`, `gen_region_polygon`

Modul

Morphology

`erosion_circle` (Region : RegionErosion : Radius :)

Erosion mit einer Kreismaske.

`erosion_circle` führt eine Minkowski-Subtraktion mit einer Kreismaske aus. Die Größe des Kreises, der als strukturieredes Element verwendet wird, ist durch `Radius` angegeben.

Die erkennbare Wirkung dieser Funktion ist die Glättung der Regionenränder. Zusätzlich werden Regionen, die flächenmäßig kleiner sind als die Kreismaske eliminiert. Sinnvollerweise wird der `Radius` auf Werte wie 3.5, 5.5 etc. gesetzt, um so Translationen der Region zu vermeiden. Bei ganzzahligen Radien hat der Kreis nämlich keinen ganzzahligen Schwerpunkt, der aber gerundet werden muß.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen auf die der Erosion-Operator angewandt wird.
- ▷ **`RegionErosion`** (output_object) ..... region(-array) ⇝ *Hobject*
  Verkleinerte Regionen.
- ▷ **`Radius`** (input_control) ..... real ⇝ *real* / integer
  Radius der Kreismaske.
  Defaultwert : 3.5
  Wertevorschläge : `Radius` ∈ {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}
  Typischer Wertebereich : 0.5 ≤ `Radius` ≤ 511.5 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

main()
{
  cout << "Simulation of 'erosion_circle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Red   = after segmentation " << endl;
  cout << "Blue  = after erosion " << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegion      circ   = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs   = (img >= 128).Connection();
  HRegionArray minsub = regs.MinkowskiSub1 (circ, 1);

                         img.Display (w);      w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();
  w.SetColor ("blue");  minsub.Display (w);   w.Click ();

  return(0);
```

}

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \texttt{Radius} \cdot \sqrt{F1}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `erosion_circle` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`erosion_circle` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`connection`, `reduce_domain`, `select_shape`, `area_center`

Alternativen

`minkowski_sub1`

Siehe auch

`gen_circle`, `dilation_circle`, `closing_circle`, `opening_circle`

Modul

Morphology

erosion_golay (Region : RegionErosion : GolayElement, Iterations, Rotation :)

Verdünnung von Regionen (Golay).

Diese Prozedur berechnet die Erosion der Eingaberegionen mit dem strukturierenden Element aus dem Golay-Alphabet, das durch `GolayElement` ausgewählt wird. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’.

Die Rotationsnummer (`Rotation`) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt. `erosion_golay` ist eine Regionenoperation. Das strukturierende Element wird dabei über die zu verarbeitende Region `Region` geschoben. Für alle Positionen des strukturierenden Elementes, an denen sie sich mit dem Objekt völlig deckt, wird der Bezugspunkt relativ zum strukturierenden Element gesetzt. Es wird also die Schnittmenge über alle Verschiebungen des strukturierenden Elementes über die Region gebildet.

Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Werte von `Rotation` möglich sind. Bei einigen Werten für `Rotation` entsteht die identische Abbildung.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Erosion.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert : 0
  Werteliste : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion erosion_golay den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

erosion_golay ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

erosion_seq, erosion1, erosion2

Siehe auch

dilation_golay, opening_golay, closing_golay, hit_or_miss_golay, thinning_golay, thickening_golay, golay_elements

Modul

Morphology

erosion_rectangle1 (Region : RegionErosion : Width, Height :)

Erosion mit einem Rechteck.

erosion_rectangle1 führt eine Erosion auf der/den angegebenen Region(en) Region durch. Das strukturierende Element für diese Operation bildet ein Rechteck mit den Größenparametern Width und Height. Die Funktion führt zu einer Verkleinerung der Region.

erosion_rectangle1 ist eine sehr schnelle Operation, da die Höhe des Rechtecks nur logarithmisch, die Breite gar nicht in die Komplexitat eingeht. Dies führt auch bei sehr großen Rechtecken (Kante > 100) zu einem sehr gutem Laufzeitverhalten.

Regionen, die schmale Verbindungsstege zwischen größeren Flächen besitzen werden zwar optisch getrennt, jedoch bleibt die Region logisch als eine Region erhalten.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Erosions-Operation.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite des Rechtecks.
  Defaultwert : 10
  Wertevorschläge : Width $\in \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100\}$
  Typischer Wertebereich : $1 \leq$ Width ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe des Rechtecks.
  Defaultwert : 10
  Wertevorschläge : Height $\in \{1, 2, 3, 4, 6, 10, 15, 20, 30, 50, 70, 100\}$
  Typischer Wertebereich : $1 \leq$ Height ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot ld(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion erosion_rectangle1 den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

erosion_rectangle1 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

erosion1, minkowski_sub1

Siehe auch

gen_rectangle1

Modul

Morphology

erosion_seq (Region : RegionErosion : GolayElement, Iterations :)

Verdünnung von Regionen (sequentiell).

erosion_seq berechnet die sequentielle Erosion der Eingaberegionen mit dem durch GolayElement ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt erosion_seq die Prozedur erosion_golay mit allen Rotationen des strukturierenden Elements so oft durch, wie der Parameter Iterations angibt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten

'l', 'd', 'c', 'f', 'h', 'k'.

Es werden nur die „Vordergrund Elemente" des Golayalphabets (gerade Rotationsnummern) verwendet. Die Elemente 'i' und 'e' erzeugen hier die identische Abbildung, d.h. sie verändern die Eingaberegion nicht. Die Elemente 'l', 'm' und 'f2' sind im Vordergrund identisch. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt. Für die Skelettbildung werden meistens die Elemente 'l' und 'm' verwendet.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionErosion** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Erosion.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : GolayElement ∈ {'l', 'd', 'c', 'f', 'h', 'k'}
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\texttt{Iterations} \cdot 20 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion erosion_seq den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

erosion_seq ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

connection, reduce_domain, select_shape, area_center

Alternativen

erosion_golay, erosion1, erosion2

Siehe auch

dilation_seq, hit_or_miss_seq, thinning_seq

Modul

Morphology

fitting (Region, StructElements : RegionFitted : :)

Closing nach Opening mit mehreren strukturierenden Elementen.

fitting führt die beiden Grundoperationen opening und closing nacheinander aus. Die acht strukturierenden Elemente, die typischerweise zum Einsatz kommen, können mit der Funktion gen_struct_elements erzeugt werden. Jedoch ist auch der Einsatz einzelner, eigens erzeugter Elemente sinnvoll. Sei(en) R die Eingaberegion(en) und M_i bezeichne die strukturierenden Elemente. Sei weiterhin P das Zwischenergebnis nach dem opening und Q das Endergebnis, dann läßt sich die Funktion formal folgendermaßen schreiben:

$$P = \bigcup_{i=1}^{n} (R \circ M_i)$$
$$Q = \bigcap_{i=1}^{n} (P \bullet M_i)$$

Regionen, die größer sind als die strukturierenden Elemente bleiben erhalten, während kleine Löcher beim Closing geschlossen werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die bearbeitet werden.
- ▷ **StructElements** (input_object) ..... region(-array) ⇝ *Hobject*
  Strukturierende Elemente.
- ▷ **RegionFitted** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die strukturierenden Elemente enthalten.

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion fitting den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

fitting ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_struct_elements, gen_region_points

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

opening, closing, connection, select_shape

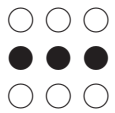
Modul

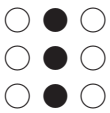
Morphology

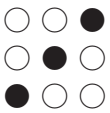
gen_struct_elements (: StructElements : Type, Row, Column :)

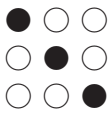
Erzeugen von strukturierenden Elementen.

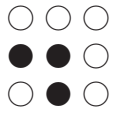
gen_struct_elements dient zur Erzeugung von acht strukturierenden Elementen, die üblicherweise für die Funktion fitting Verwendung finden. Die Standardbelegung **'noise'** für den Parameter Type erzeugt Elemente, die insbesonders für die Eliminierung von Rauschen geeignet sind.

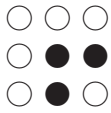

M_1

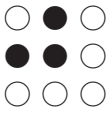

M_2

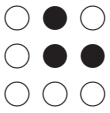

M_3


M_4


M_5


M_6


M_7


M_8

Parameter

▷ **`StructElements`** (output_object) ..... region(-array) ⇝ *Hobject*
Erzeugte strukturierende Elemente.

▷ **`Type`** (input_control) ..... string ⇝ *string*
Art der strukturierende Elemente.
Defaultwert : 'noise'
Werteliste : `Type` ∈ {'noise'}

▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Bezugspunktes.
Defaultwert : 1
Wertevorschläge : `Row` ∈ {0, 1, 10, 50, 100, 200, 300, 400}
Typischer Wertebereich : $-\infty \leq$ `Row` $\leq \infty$ (lin)

▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Bezugspunktes.
Defaultwert : 1
Wertevorschläge : `Column` ∈ {0, 1, 10, 50, 100, 200, 300, 400}
Typischer Wertebereich : $-\infty \leq$ `Column` $\leq \infty$ (lin)

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `gen_struct_elements` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`gen_struct_elements` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`fitting`, `hit_or_miss`, `opening`, `closing`, `erosion2`, `dilation2`

Siehe auch

`golay_elements`

Modul

Morphology

`golay_elements` `( : StructElement1, StructElement2 : GolayElement, Rotation, Row, Column : )`

Strukturierende Elemente des Golay-Alphabets.

`golay_elements` erzeugt die strukturierenden Elemente aus dem Golay-Alphabet. Der Parameter `GolayElement` legt den Namen, `Rotation` die Rotation fest. Die strukturierenden Elemente sind für `hit_or_miss` ausgelegt: In `StructElement1` wird das strukturierende Element für den Vordergrund in `StructElement2` das für den Hintergrund übergeben. `Row` und `Column` legen den Bezugspunkt fest.

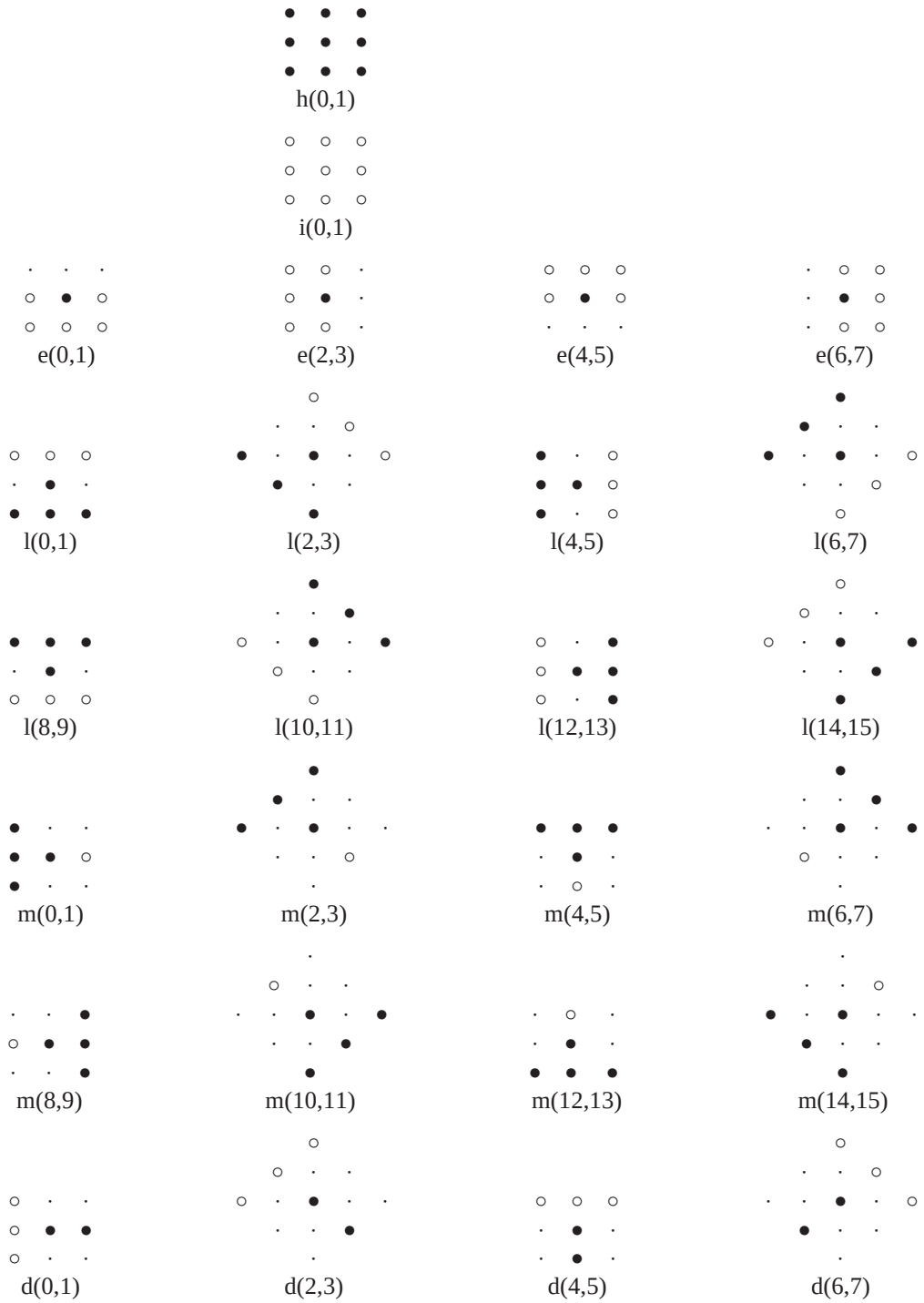
Die Rotationen sind von 0 bis 15 durchnumeriert. Das heißt aber nicht, daß es 16 verschiedene Rotationen gibt: Die geraden Werte bezeichnen die Rotationen der Vordergrundelemente, die ungeraden die der Hintergrundelemente.

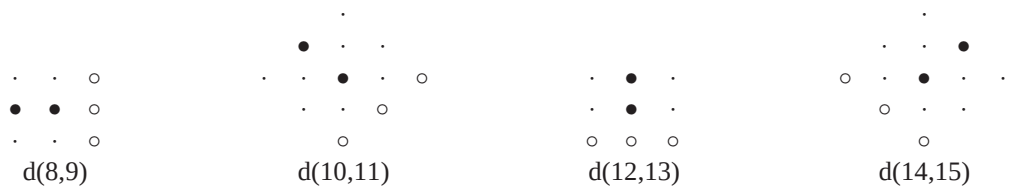
Bei `golay_elements` werden nur die geraden Werte angegeben und legen den Wert für `StructElement1` fest. Die nächstgrößere (ungerade) Zahl wird intern berechnet und bestimmt den Wert für `StructElement2`. Für die Elemente 'h' und 'i' gibt es keine Rotationen, folglich sind nur 0 und 1 als „Rotationen" möglich (bei `golay_elements` folglich nur 0). Das Element 'e' hat nur 4 Rotationen, folglich liegen die gültigen Werte zwischen 0 und 7 (bei `golay_elements` ergeben sich die Werte 0, 2, 4 und 6).

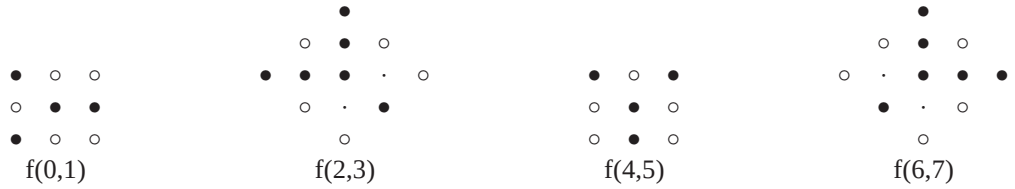
Die Grafiken zeigen die Elemente des Golay-Alphabets mit allen realisierten Rotationen. Die verwendeten Zeichen haben folgende Bedeutung:

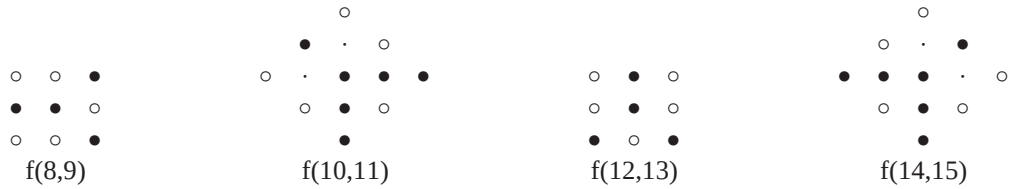
- ● Vordergrundpixel
- ○ Hintergrundpixel
- · Pixel mit beliebigem Wert

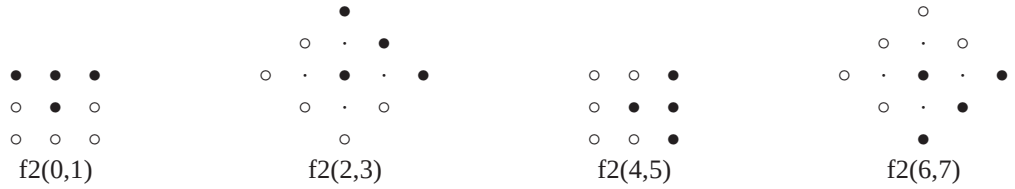
Die Namen der Elemente und ihre Rotationsnummern stehen jeweils unter der Zeichnung. Die Elemente mit gerader Nummer enthalten die Vordergrundpixel, die Elemente mit ungerader Rotationsnummer die Hintergrundpixel.



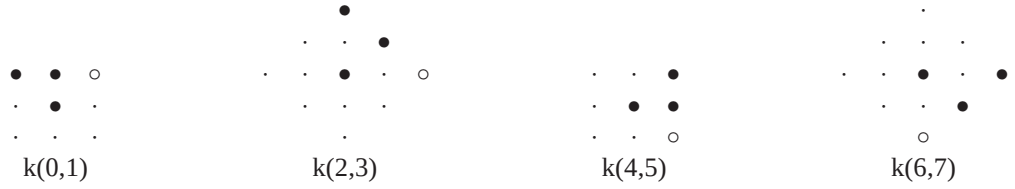

d(8,9) d(10,11) d(12,13) d(14,15)

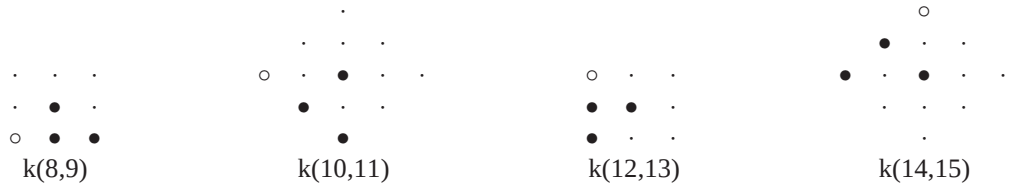

f(0,1) f(2,3) f(4,5) f(6,7)


f(8,9) f(10,11) f(12,13) f(14,15)

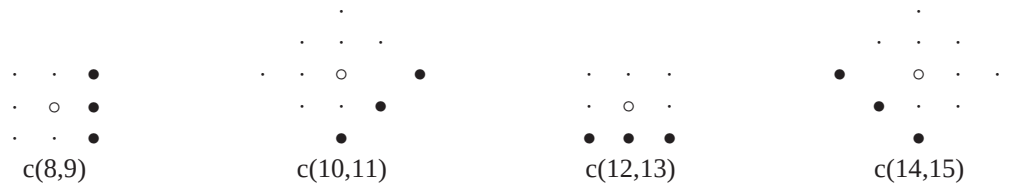

f2(0,1) f2(2,3) f2(4,5) f2(6,7)


f2(8,9) f2(10,11) f2(12,13) f2(14,15)


k(0,1) k(2,3) k(4,5) k(6,7)


k(8,9) k(10,11) k(12,13) k(14,15)


c(0,1) c(2,3) c(4,5) c(6,7)


c(8,9) c(10,11) c(12,13) c(14,15)

Parameter

- ▷ **StructElement1** (output_object) . . . . . region ⇝ *Hobject*
  Strukturierendes Element für den Vordergrund.
- ▷ **StructElement2** (output_object) . . . . . region ⇝ *Hobject*
  Strukturierendes Element für den Hintergrund.
- ▷ **GolayElement** (input_control) . . . . . string ⇝ *string*
  Name des strukturierenden Elementes.
  Defaultwert : 'l'
  Werteliste : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Rotation** (input_control) . . . . . integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert : 0
  Werteliste : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14}
- ▷ **Row** (input_control) . . . . . point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Defaultwert : 16
  Wertevorschläge : Row ∈ {0, 16, 32, 128, 256}
  Typischer Wertebereich : 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) . . . . . point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Defaultwert : 16
  Wertevorschläge : Column ∈ {0, 16, 32, 128, 256}
  Typischer Wertebereich : 0 ≤ Column ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Ergebnis

Bei korrekter Parametrisierung liefert die Prozedur golay_elements den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

golay_elements ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

hit_or_miss

Alternativen

gen_region_points, gen_struct_elements, gen_region_polygon_filled

Siehe auch

dilation_golay, erosion_golay, opening_golay, closing_golay, hit_or_miss_golay, thickening_golay

Literatur

J. Serra: „Image Analysis and Mathematical Morphology". Volume I. Academic Press, 1982

Modul

Morphology

hit_or_miss (Region, StructElement1, StructElement2 : RegionHitMiss : Row, Column :)

Hit-or-Miss-Operation für Regionen.

hit_or_miss berechnet die Hit-or-Miss-Transformation. Die Prozedur führt dabei zuerst eine Erosion der Region Region mit dem strukturierenden Element StructElement1 durch. Im zweiten Schritt wird eine Erosion der komplementären Region mit dem strukturierenden Element StructElement2 durchgeführt. Aus den beiden Zwischenergebnissen wird die Schnittmenge gebildet.

Die Hit-or-Miss-Transformation wählt genau die Punkte aus, deren Umgebung die Bedingungen der strukturierenden Elemente StructElement1 und StructElement2 erfüllen. StructElement1 bezeichnet dabei die

Punkte des Vordergrundes, `StructElement2` die Punkte des Hintergrundes. Für ein sinnvolles Ergebnis müssen `StructElement1` und `StructElement2` wie Schlüssel und Schloß zusammenpassen. `StructElement1` und `StructElement2` sind in jedem Fall disjunkt. `Row` und `Column` bezeichnen Zeilen- und Spaltennummer des Bezugspunktes.

Strukturierende Elemente (`StructElement1`, `StructElement2`) können mit Prozeduren wie `golay_elements`, `gen_struct_elements`, `gen_region_points`, etc. erzeugt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`StructElement1`** (input_object) ..... region ⇝ *Hobject*
  Erosionsmaske für die Eingaberegionen.
- ▷ **`StructElement2`** (input_object) ..... region ⇝ *Hobject*
  Erosionsmaske für die Komplemente der Eingaberegionen.
- ▷ **`RegionHitMiss`** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Hit-or-Miss-Operation.
- ▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Defaultwert : 16
  Wertevorschläge : `Row` ∈ {0, 16, 32, 128, 256}
  Typischer Wertebereich : 0 ≤ `Row` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Defaultwert : 16
  Wertevorschläge : `Column` ∈ {0, 16, 32, 128, 256}
  Typischer Wertebereich : 0 ≤ `Column` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, $F1$ die Fläche des strukturierenden Elementes 1 und $F2$ die Fläche der des strukturierenden Elementes 2, dann ist die Laufzeitkomplexität für ein Objekt:

$$O\left(\sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right) \quad .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `hit_or_miss` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`hit_or_miss` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`golay_elements`, `gen_struct_elements`, `threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`difference`, `reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`hit_or_miss_golay`, `hit_or_miss_seq`, `erosion2`, `dilation2`

Siehe auch

`thinning`, `thickening`, `gen_region_points`, `gen_region_polygon_filled`

Modul

Morphology

hit_or_miss_golay (Region : RegionHitMiss : GolayElement, Rotation :)

Hit-or-Miss-Operation für Regionen (Golay).

hit_or_miss_golay berechnet die Hit-or-Miss-Transformation der Eingaberegionen (abgestützt auf das Golay-Alphabet). Dazu wird zunächst eine Erosion der Regionen mit dem durch GolayElement ausgewählten strukturierenden Element des Golay-Alphabets durchgeführt. Im zweiten Schritt folgt eine Erosion der Komplemente der Regionen mit der Hintergrund-Maske des strukturierenden Elements. Aus den beiden Zwischenergebnissen wird die Schnittmenge gebildet. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e','i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Hit-or-Miss-Transformation wählt genau die Punkte aus, deren Umgebung die Bedingungen des gewählten Golay-Elements erfüllt.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Rotationen möglich sind.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionHitMiss** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Hit-or-Miss-Operation.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert : 0
  Werteliste : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion hit_or_miss_golay den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hit_or_miss_golay ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

hit_or_miss_seq, hit_or_miss

Siehe auch

erosion_golay, dilation_golay, opening_golay, closing_golay, thinning_golay, thickening_golay, golay_elements

Modul

Morphology

hit_or_miss_seq (Region : RegionHitMiss : GolayElement :)

Hit-or-Miss-Operation für Regionen (sequentiell).

hit_or_miss_seq berechnet die Hit-or-Miss-Transformation einer Region mit allen rotierten Versionen des durch GolayElement gewählten strukturierenden Elements aus dem Golay-Alphabet. Als Ergebnis wird die Vereinigungsmenge der Zwischenergebnisse aus den einzelnen Rotationen geliefert. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionHitMiss** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Hit-or-Miss-Operation.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}

Komplexität

Sei F die Fläche einer Eingaberegion und R die Anzahl der Rotationen, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(R \cdot 6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion hit_or_miss_seq den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hit_or_miss_seq ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

hit_or_miss_golay, hit_or_miss

Siehe auch

thinning_seq, thickening_seq

Modul

Morphology

minkowski_add1 (Region, StructElement : RegionMinkAdd : Iterations :)

Ausdehnen von Regionen.

minkowski_add1 berechnet die Minkowski-Addition der Eingaberegionen mit einem strukturierenden Element. Die Anwendung von minkowski_add1 glättet die Ränder der Regionen. Gleichzeitig vergrößert sich die Fläche

dieser Regionen. Darüberhinaus kann es vorkommen, daß vorher nicht zusammenhängende Regionen verschmolzen werden. Dennoch bleiben solche Regionen logisch verschiedene Region. Die Minkowski-Addition ist eine mengentheoretische Regionenoperation. Sie basiert auf Translation und Vereinigung.

Seien M `StructElement` und R (`Region`) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{minkowski\_add1}(R, M) := \bigcup_{m \in M} t_{\vec{v}_m}(R)$$

Es wird für jeden Punkt m in M eine Translation mit der Region R durchgeführt. Die Vereinigung über all diese Verschiebungen ist die Dilatation der Region R mit M. Der Unterschied zwischen `minkowski_add1` und `dilation1` ist die Punktspiegelung des strukturierenden Elements (Negation des Verschiebungsvektors) bei `dilation1`. Die Position von `StructElement` ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet. Aus der obigen Definition ergibt sich, daß bei einem leeren strukturierenden Element eine leere Region erzeugt wird.

Strukturierende Elemente (`StructElement`) können mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points` etc. erzeugt werden.

Achtung

Eine Minkowski-Addition führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung der Prozedur zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muß zuerst ein `union1` durchgeführt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element.
- ▷ **`RegionMinkAdd`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Minkowski-Addition.
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ `Iterations` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `minkowski_add1` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`minkowski_add1` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add2`, `dilation1`

Siehe auch

`transpose_region`, `minkowski_sub1`

Modul

Morphology

`minkowski_add2` (`Region, StructElement : RegionMinkAdd : Row, Column, Iterations : )`

Ausdehnen von Regionen (mit Bezugspunkt).

`minkowski_add2` berechnet die Minkowski-Addition der Eingaberegionen mit einem strukturierenden Element (`StructElement`) und dem Bezugspunkt, der durch `Row` und `Column` charakterisiert wird. `minkowski_add2` entspricht der Prozedur `minkowski_add1` mit dem Unterschied, daß der Bezugspunkt des strukturierenden Elementes frei gewählt werden kann. Der Parameter `Iterations` bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird ein leere Region erzeugt.

Strukturierende Elemente können mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points` etc. erzeugt werden.

Achtung

Eine Minkowski-Addition führt grundsätzlich zu einer Vergrößerung der Regionen. Eng benachbarte Regionen, die nach Ausführung der Prozedur zusammenstoßen oder sich überlappen werden weiterhin als zwei getrennte Regionen behandelt. Um eine Vereinigung zweier Regionen zu erreichen muß zuerst ein `union1` durchgeführt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element.
- ▷ **`RegionMinkAdd`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Minkowski-Addition.
- ▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunktes.
  Typischer Wertebereich : $1 \leq$ `Row` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunktes.
  Typischer Wertebereich : $1 \leq$ `Column` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Anzahl der Iterationen.
Defaultwert : 1
Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich : 1 ≤ `Iterations` (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `minkowski_add2` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`minkowski_add2` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `dilation1`

Siehe auch

`transpose_region`

Modul

Morphology

minkowski_sub1 (Region, StructElement : RegionMinkSub : Iterations :)

Verdünnung von Regionen.

`minkowski_sub1` berechnet die Minkowski-Subtraktion der Eingaberegionen mit einem Strukturierenden Element. Die Anwendung von `minkowski_sub1` glättet die Ränder der Regionen. Gleichzeitig verkleinert sich die Fläche dieser Regionen. Darüberhinaus kann es vorkommen, daß vorher zusammenhängende Regionen getrennt werden. Dennoch bleiben solche Regionen logisch eine Region. Die Minkowski-Subtraktion ist eine mengentheoretische Regionenoperation. Sie verwendet die Operation Durchschnitt.

Seien M (`StructElement`) und R (`Region`) Regionen, wobei M das „strukturierende Element“ und R die zu verarbeitende Region darstellt. Sei weiterhin m ein Punkt aus M, dann wird der Verschiebungsvektor $\vec{v}_m = (dx, dy)$ definiert als die Differenz des Schwerpunktvektors von M mit dem Vektor $\vec{m}$. Die Translation einer Region R um einen Vektor $\vec{v}$ sei mit $t_{\vec{v}_m}(R)$ bezeichnet. Dann ist

$$\texttt{minkowski\_sub1}(R, M) := \bigcap_{m \in M} t_{\vec{v}_m}(R)$$

Es wird für jeden Punkt aus M eine Translation mit der Region R durchgeführt. Der Durchschnitt über all diesen Verschiebungen ist die Minkowski-Differenz der Region R mit M. Der Unterschied zwischen

minkowski_sub1 und erosion1 ist die Punktspiegelung des strukturierenden Elemente (Negation des Verschiebungsvektors) bei erosion1. Die Position von StructElement ist ohne Bedeutung, da die Verschiebungsvektoren bzgl. des Schwerpunktes bestimmt werden.

Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet. Bei Verwendung eines leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element.
- ▷ **RegionMinkSub** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Minkowski-Subtraktion.
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion minkowski_sub1 den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

minkowski_sub1 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

minkowski_sub2, erosion1

Siehe auch

transpose_region

Modul

Morphology

minkowski_sub2 (Region, StructElement : RegionMinkSub : Row, Column, Iterations :)

Verdünnung von Regionen (mit Bezugspunkt).

minkowski_sub2 berechnet die Minkowski-Subtraktion der Eingaberegionen mit einem Strukturierenden Element StructElement und dem Bezugspunkt in Row und Column. minkowski_sub2 entspricht der Prozedur minkowski_sub1 mit dem Unterschied, daß bei minkowski_sub2 der Bezugspunkt frei gewählt werden kann. Der Parameter Iterations bezeichnet die Anzahl der Iterationen, die mit dem strukturierenden Element ausgeführt werden sollen. Als Eingaberegion für die n-te Iteration wird die Ergebnisregion der $(n-1)$-ten Iteration verwendet.

Bei Verwendung des leeren strukturierenden Elementes wird die maximale Region erzeugt.

Strukturierende Elemente (StructElement) können mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element.
- ▷ **RegionMinkSub** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Erosion.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert : 0
  Wertevorschläge : Row ∈ {0, 10, 16, 32, 64, 100, 128}
  Typischer Wertebereich : 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunkts.
  Defaultwert : 0
  Wertevorschläge : Column ∈ {0, 10, 16, 32, 64, 100, 128}
  Typischer Wertebereich : 0 ≤ Column ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 1 ≤ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \texttt{Iterations}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion minkowski_sub2 den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`minkowski_sub2` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_sub1`, `erosion1`, `erosion2`, `erosion_golay`, `erosion_seq`

Siehe auch

`gen_circle`, `gen_rectangle2`, `gen_region_polygon`

Modul

Morphology

morph_hat (Region, StructElement : RegionMorphHat : :)

Vereinigung von `bottom_hat` und `top_hat`.

`morph_hat` vereinigt die Flächen, die bei der `opening`-Operation wegfallen mit den Flächen, die bei der `closing`-Operation hinzukommen. Dies entspricht also der Vereinigung von `top_hat` und `bottom_hat`. Die Position von `StructElement` ist ohne Bedeutung.

Das strukturierende Element (`StructElement`) kann mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points`, etc. erzeugt werden.

Achtung

Die Regionen werden einzeln bearbeitet.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionMorphHat** (output_object) ..... region(-array) ⇝ *Hobject*
  Vereinigung von Top-Hat und Bottom-Hat.

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

main()
{
  cout << "Reproduction of 'dilation_circle ()'" << endl;
  cout << "First = original image " << endl;
  cout << "Red   = after segmentation " << endl;
  cout << "Blue  = after erosion " << endl;

  HByteImage img("monkey");
  HWindow    w;

  HRegion      circ   = HRegion::GenCircle (10, 10, 1.5);
  HRegionArray regs   = (img >= 128).Connection();
  HRegionArray tophat = regs.TopHat (circ);
```

```
  HRegionArray bothat = regs.BottomHat (circ);
  HRegionArray unionX = tophat.Union2 (bothat);

                        img.Display (w);      w.Click ();
  w.SetColor ("red");   regs.Display (w);     w.Click ();
  w.SetColor ("blue");  tophat.Display (w);   w.Click ();
  w.SetColor ("green"); bothat.Display (w);   w.Click ();
  w.SetColor ("white"); unionX.Display (w);   w.Click ();

  return(0);
}
```

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion morph_hat den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

morph_hat ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

top_hat, bottom_hat, union2

Siehe auch

opening, closing

Modul

Morphology

morph_skeleton (Region : RegionSkeleton : :)

Mophologisches Skelett einer Region.

morph_skeleton berechnet das Skelett der Eingaberegionen (Region) mit Hilfe morphologischer Transformationen. Die Berechnung liefert ein nicht zusammenhängendes Skelett (Lücken in den Diagonalen) mit einer Breite von ein oder zwei Pixeln. Zur Berechnung wird das Golay-Element 'h' verwendet. Hieraus ergibt sich die Verwendung der 8-ter Nachbarschaft. Dies entspricht der Maximumsnorm.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionSkeleton** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis der Skelettierung.

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion morph_skeleton den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`morph_skeleton` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`skeleton`, `reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`skeleton`, `thinning`

Siehe auch

`thinning_seq`, `morph_skiz`

Modul

Morphology

`morph_skiz` (Region : RegionSkiz : Iterations1, Iterations2 :)

Verdünnen von Regionen.

`morph_skiz` führt zuerst ein sequentielles Thinning (`thinning_seq`) mit dem Element ’l’ des Golay-Alphabets durch. Die Anzahl der Durchläufe bestimmt der Parameter `Iterations1`. Mit dem Ergebnis wird dann ein sequentielles Thinning mit dem Element ’e’ des Golay-Alphabets durchgeführt. Die Anzahl der Durchläufe bestimmt hier der Parameter `Iterations2`. Durch die Skiz-Operation wird eine Art Skelett gebildet und die Äste dann verkürzt. Wird die Skiz-Operation auf die komplementäre Region angewandt, so werden die Regionen durch das entstandene Skelett voneinander getrennt.

Werden bei `Iterations1` oder `Iterations2` sehr große Werte oder ’maximal’ übergeben, dann bricht die Verarbeitung ab, wenn keine Veränderung mehr auftritt.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`RegionSkiz`** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Skiz-Operators.
- ▷ **`Iterations1`** (input_control) ..... integer ⇝ *integer* / string
  Iterationen für das sequentielle Thinning mit dem Element ’l’des Golay-Alphabets.
  Defaultwert : 100
  Wertevorschläge : `Iterations1` ∈ {’maximal’, 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200, 300, 400}
  Typischer Wertebereich : 0 ≤ `Iterations1` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Iterations2`** (input_control) ..... integer ⇝ *integer* / string
  Iterationen für das sequentielle Thinning mit dem Element ’e’ des Golay-Alphabets.
  Defaultwert : 1
  Wertevorschläge : `Iterations2` ∈ {’maximal’, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
  Typischer Wertebereich : 0 ≤ `Iterations2` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O((\texttt{Iterations1} + \texttt{Iterations2}) \cdot 3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `morph_skiz` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`morph_skiz` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`pruning`, `reduce_domain`, `select_shape`, `area_center`, `connection`, `background_seg`, `complement`

Alternativen

`skeleton`, `thinning_seq`, `morph_skeleton`, `interjacent`

Siehe auch

`thinning`, `hit_or_miss_seq`, `difference`

Modul

Morphology

`opening ( Region, StructElement : RegionOpening : : )`

Auftrennen von Lücken.

Die `opening`-Operation ist als die Hintereinanderschaltung von Erosion und Minkowsi-Addition definiert. Durch Anwendung von `opening` bleiben größere Strukturen weitgehend erhalten, kleine Regionen wie Linien und Punkte sowie feine Strukturen werden gelöscht. Im Gegensatz dazu bewirkt eine `closing`-Operation, daß kleine Unterbrechungen, Lücken und Risse erhalten bleiben bzw. ausgefüllt werden (siehe hierzu `closing`).

`opening` dient zur Elimination von kleinen Regionen (kleiner als `StructElement`) und zum Glätten der Ränder. `opening` verwendet intern als Bezugspunkt den Schwerpunkt des strukturierenden Elementes. Die Position von `StructElement` ist aber ohne Bedeutung, da Opening translationsinvariant gegenüber dem strukturierende Element ist.

`StructElement` kann mit Prozeduren wie `gen_circle`, `gen_rectangle1`, `gen_rectangle2`, `gen_ellipse`, `draw_region`, `gen_region_polygon`, `gen_region_points`, etc. erzeugt werden.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`StructElement`** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element (lageinvariant).
- ▷ **`RegionOpening`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis des Opening-Operators.

Beispiel

```
/* 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).
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot \sqrt{F2}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `opening` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`opening` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`minkowski_add1`, `erosion1`, `opening_circle`

Siehe auch

`gen_circle`, `gen_rectangle2`, `gen_region_polygon`

Modul

Morphology

`opening_circle ( Region : RegionOpening : Radius : )`

Auftrennen von Lücken mit einer Kreismaske.

`opening_circle` ist als die Hintereinanderschaltung von Erosion und Minkowsi-Addition mit einer Kreismaske definiert (siehe Beispiel). `opening_circle` dient zur Elimination von kleinen Regionen (kleiner als die Kreismaske) und zum Glätten der Ränder.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`RegionOpening`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis des Opening-Operators.
- ▷ **`Radius`** (input_control) ..... real ⇝ *real* / integer
  Radius der Kreismaske.
  Defaultwert : 3.5
  Wertevorschläge : `Radius` ∈ {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}
  Typischer Wertebereich : 0.5 ≤ `Radius` ≤ 511.5 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0

Beispiel

```
/* 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).
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(4 \cdot \sqrt{F1} \cdot \texttt{Radius}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `opening_circle` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`opening_circle` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening`, `dilation1`, `minkowski_add1`, `gen_circle`

Siehe auch

`transpose_region`

Modul

Morphology

opening_golay (Region : RegionOpening : GolayElement, Rotation :)

Auftrennen von Lücken mit einem Golay-Element.

`opening_golay` ist als Hintereinanderschaltung von Minkowski-Subtraktion und Minkowski-Addition. Dabei wird zuerst die Minkowski-Subtraktion der Eingaberegionen (`Region`) mit dem durch `GolayElement` und `Rotation` ausgewählten strukturierenden Element aus dem Golay-Alphabet durchgeführt. Mit dem Ergebnis der Minkowski-Subtraktion und dem um 180° gedrehten strukturierenden Element, wird dann eine Minkowski-Addition durchgeführt.

Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (`Rotation`) gibt dabei an, welche `Rotation` des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Durch `opening_golay` werden Regionen, die kleiner sind als das strukturierende Element, entfernt und die Objektränder geglättet.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Werte von `Rotation` zulässig sind. Bei einigen Werten für `Rotation` entsteht die identische Abbildung.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`RegionOpening`** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Opening-Operators.
- ▷ **`GolayElement`** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : `GolayElement` ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **`Rotation`** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zuläßig.
  Defaultwert : 0
  Werteliste : `Rotation` ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `opening_golay` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`opening_golay` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening_seg`, `opening`

Siehe auch

`erosion_golay`, `dilation_golay`, `closing_golay`, `hit_or_miss_golay`, `thinning_golay`, `thickening_golay`, `golay_elements`

Modul

Morphology

`opening_rectangle1` (Region : RegionOpening : Width, Height :)

Opening mit einem Rechteck.

`opening_rectangle1` setzt sich aus der Abfolge der Funktionsaufrufe `erosion_rectangle1` und `dilation_rectangle1` zusammen. Die Größe des rechteckigen, strukturierenden Elementes wird durch die Parameter `Width` und `Height` bestimmt. Wie bei allen `opening`-Varianten bleiben größere Strukturen erhalten, während kleine Regionen wie Linien und Punkte sowie feine Strukturen gelöscht werden.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen die verarbeitet werden sollen.

▷ **RegionOpening** (output_object) ..... region(-array) ⇝ *Hobject*
Ergebnis des Opening-Operators.

▷ **Width** (input_control) ..... extent.x ⇝ *integer* / real
Breite des Rechtecks.
Defaultwert : 10
Wertevorschläge : `Width` ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
Typischer Wertebereich : 1 ≤ `Width` ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Height** (input_control) ..... extent.y ⇝ *integer* / real
Höhe des Rechtecks.
Defaultwert : 10
Wertevorschläge : `Height` ∈ {1, 2, 3, 4, 5, 7, 9, 12, 15, 19, 25, 33, 45, 60, 110, 150, 200}
Typischer Wertebereich : 1 ≤ `Height` ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und H die Höhe des Rechtecks, dann ist die Laufzeitkomplexität für eine Region:

$$O(2 \cdot \sqrt{F1} \cdot ld(H)) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `opening_rectangle1` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`opening_rectangle1` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `watersheds`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening`, `gen_rectangle1`, `dilation_rectangle1`, `erosion_rectangle1`

Siehe auch

`opening_seg`, `opening_golay`

Modul

Morphology

opening_seg (Region, StructElement : RegionOpening : :)

Trennen von sich überlappenden Regionen.

Die `opening_seg`-Operation ist als die Hintereinanderschaltung von `erosion1`, `connection` und `dilation1` definiert (siehe Beispiel). Es wird dabei nur eine Iteration bei `erosion1` und `dilation1` ausgeführt.

opening_seg dient zum Trennen von sich überlappenden Regionen, deren Überlappungsbereich kleiner als StructElement ist. Es ist zu beachten, daß sich die Ergebnisregionen überlappen können ohne dabei zu verschmelzen (siehe expand_region). opening_seg verwendet als Bezugspunkt den Schwerpunkt des strukturierenden Elementes.

Das strukturierende Element StructElement kann mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionOpening** (output_object) ..... region-array ⇝ *Hobject*
  Das Ergebnis des Opening-Operators.

Beispiel

```
/* 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]).
```

Komplexität

Sei $F1$ die Fläche einer Eingaberegion und $F2$ die Fläche des strukturierenden Elementes, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\sqrt{F1} \cdot \sqrt{F2} \cdot \sqrt{\sqrt{F1}}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion opening_seg den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

opening_seg ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

expand_region, reduce_domain, select_shape, area_center, connection

Alternativen

erosion1, connection, dilation1

Modul

Morphology

pruning (Region : RegionPrune : Length :)

Beschneiden von Ästen einer Region.

pruning entfernt aus einem Skelett (Region) Äste einer vorgegebenen Länge, die durch Length vorgegeben ist. Die übrigen Äste beleiben unverändert.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
Regionen, die verarbeitet werden sollen.

▷ **RegionPrune** (output_object) ..... region(-array) ⇝ *Hobject*
Das Ergebnis des Pruning-Operators.

▷ **Length** (input_control) ..... integer ⇝ *integer*
Länge der Äste die entfernt werden.
Defaultwert : 2
Wertevorschläge : Length ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50}
Typischer Wertebereich : 1 ≤ Length ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(\texttt{Length} \cdot 3 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion pruning den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

pruning ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

morph_skiz, skeleton, thinning_seq

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Siehe auch

morph_skeleton, junctions_skeleton

Modul

Morphology

thickening (Region, StructElement1, StructElement2 : RegionThick : Row, Column, Iterations :)

Vergrößerung der Region um das Ergebnis von Hit-or-Miss.

thickening berechnet das Thickening der Eingaberegionen mit morphologischen Operationen. Die Prozedur führt dabei zuerst eine Hit-or-Miss-Transformation (vgl. hit_or_miss) durch und fügt die dabei gefundenen Punkte zur Eingaberegion hinzu. Der Parameter Iterations bestimmt die Anzahl der Iterationen, die durchgeführt werden.

Für die Wahl der strukturierenden Elemente StructElement1 und StructElement2 sowie für Row und Column gelten dieselben Empfehlungen wie bei hit_or_miss.

Die strukturierenden Elemente (StructElement1 und StructElement2) können z.B. mit der Prozedur golay_elements erzeugt werden.

Achtung

Wenn der Bezugspunkt von StructElement1 zum Vordergrund gehört, ändert sich die Region nicht.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement1** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für den Vordergrund
- ▷ **StructElement2** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für den Hintergrund
- ▷ **RegionThick** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Thickening-Operators.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert : 16
  Wertevorschläge : Row ∈ {0, 2, 4, 8, 16, 32, 128}
  Typischer Wertebereich : 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunkts.
  Defaultwert : 16
  Wertevorschläge : Column ∈ {0, 2, 4, 8, 16, 32, 128}
  Typischer Wertebereich : 0 ≤ Column ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationsschritte.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200, 400}
  Typischer Wertebereich : 1 ≤ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, $F1$ die Fläche des strukturierenden Elementes 1 und $F2$ die Fläche der des strukturierenden Elementes 2, dann ist die Laufzeitkomplexität für ein Objekt:

$$O\left(\texttt{Iterations} \cdot \sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right) \quad .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thickening den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

thickening ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

golay_elements, threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

thickening_golay, thickening_seq

Siehe auch

hit_or_miss

Modul

Morphology

thickening_golay (Region : RegionThick : GolayElement, Rotation :)

Vergrößerung einer Region um das Ergebnis von Hit-or-Miss (Golay).

thickening_golay berechnet das Thickening der Eingaberegionen mit morphologischen Operationen (abgestützt auf das Golay-Alphabet). Die Prozedur führt dazu eine Hit-or-Miss-Transformation (vgl. hit_or_miss_golay) durch und fügt die dabei gefundenen Punkte zur Eingaberegion hinzu. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Rotationsnummer (Rotation) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind unter golay_elements beschrieben.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Rotationen möglich sind.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionThick** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Thickening-Operators.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zulässig.
  Defaultwert : 0
  Werteliste : Rotation ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thickening_golay den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

thickening_golay ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

thickening, thickening_seq

Siehe auch

erosion_golay, hit_or_miss_golay

Modul

Morphology

thickening_seq (Region : RegionThick : GolayElement, Iterations :)

Vergrößerung einer Region um das Ergebnis von Hit-or-Miss (sequentiell).

thickening_seq berechnet das sequentielle Thickening der Eingaberegionen mit dem durch GolayElement ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt thickening_seq die Prozedur thickening_golay mit allen Rotationen des strukturierneden Elements so oft durch, wie der Parameter Iterations angibt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'.

Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von golay_elements aufgeführt. Bei allen Elementen außer 'c' werden Vorder- und Hintergrundanteil vertauscht, damit sie Einfluß auf den äußeren Rand der Region nehmen. Mit dem Element 'c' kann eine konvexe Hülle erzeugt werden, wenn genügend Iterationen durchgeführt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionThick** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Thickening-Operators.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : 'h'
  Werteliste : GolayElement ∈ {'l', 'm', 'd', 'c', 'e', 'i', 'f', 'f2', 'h', 'k'}
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationsschritte.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50, 70, 100, 200}
  Typischer Wertebereich : 1 ≤ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(Iterations \cdot 6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thickening_seq den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

thickening_seq ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

reduce_domain, select_shape, area_center, connection

Alternativen

thickening_golay, thickening

Siehe auch

erosion_golay, thinning_seq

Modul

Morphology

thinning (Region, StructElement1, StructElement2 : RegionThin : Row, Column, Iterations :)

Verdünnung einer Region um das Ergebnis von Hit-or-Miss.

thinning berechnet die Verdünnung der Eingaberegionen mit morphologischen Operationen. Dazu führt die Prozedur eine Hit-or-Miss-Transformation (vgl. hit_or_miss) durch und entfernt die dabei gefundenen Punkte aus den Eingaberegionen. Der Parameter Iterations bestimmt die Anzahl der Iterationen, die durchgeführt werden.

Für die Wahl der strukturierende Elemente StructElement1 und StructElement2 sowie für Row und Column gelten dieselben Empfehlungen wie bei hit_or_miss.

Strukturierende Elemente (StructElement1, StructElement2) können mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement1** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für den Vordergrund
- ▷ **StructElement2** (input_object) ..... region ⇝ *Hobject*
  Strukturierendes Element für den Hintergrund
- ▷ **RegionThin** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Thinning-Operators.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeile des Bezugspunkts.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ Row ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spalte des Bezugspunkts.
  Defaultwert : 0
  Typischer Wertebereich : $0 \leq$ Column ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl der Iterationen.
  Defaultwert : 1
  Wertevorschläge : Iterations $\in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 17, 20, 30, 40, 50\}$
  Typischer Wertebereich : $1 \leq$ Iterations (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, $F1$ die Fläche des strukturierenden Elementes 1 und $F2$ die Fläche der des strukturierenden Elementes 2, dann ist die Laufzeitkomplexität für ein Objekt:

$$O\left(\text{Iterations} \cdot \sqrt{F} \cdot \left(\sqrt{F1} + \sqrt{F2}\right)\right) \quad .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thinning den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`thinning` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`thinning_golay`, `thinning_seq`

Siehe auch

`hit_or_miss`

Modul

Morphology

thinning_golay (Region : RegionThin : GolayElement, Rotation :)

Verdünnung einer Region um das Ergebnis von Hit-or-Miss (Golay).

`thinning_golay` berechnet die Verdünnung der Eingaberegionen mit morphologischen Operationen (abgestützt auf das Golay-Alphabet). Dazu führt die Prozedur eine Hit-or-Miss-Transformation (vgl. `hit_or_miss_golay`) aus und entfernt die dabei gefundenen Punkte aus den Eingaberegionen. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’.

Die Rotationsnummer (`Rotation`) gibt dabei an, welche Rotation des gewählten Elements verwendet werden soll. Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Achtung

Es ist zu beachten, daß nicht bei jedem Golayelement alle Rotationen möglich sind.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **RegionThin** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Thinning-Operators.
- ▷ **GolayElement** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : ’h’
  Werteliste : `GolayElement` ∈ {’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’}
- ▷ **Rotation** (input_control) ..... integer ⇝ *integer*
  Rotation des Golay-Elements. Je nach Element sind nicht alle Rotationen zuläßig.
  Defaultwert : 0
  Werteliste : `Rotation` ∈ {0, 2, 4, 6, 8, 10, 12, 14, 1, 3, 5, 7, 9, 11, 13, 15}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `thinning_golay` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`thinning_golay` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`thinning_seq`, `thinning`

Siehe auch

`erosion_golay`, `hit_or_miss_golay`

Modul

Morphology

`thinning_seq` (Region : RegionThin : GolayElement, Iterations :)

Sequentielles Verdünnen einer Region um das Ergebnis von Hit-or-Miss.

`thinning_seq` berechnet das sequentielle Thinning der Eingaberegionen mit dem durch `GolayElement` ausgewählten strukturierenden Element aus dem Golay-Alphabet. Dazu führt `thinning_seq` die Prozedur `thinning_golay` für die Eingaberegionen mit allen Rotationen des strukturierenden Elements so oft durch, wie der Parameter `Iterations` angibt. Wird `Iterations` groß genug gewählt, erzeugt die Prozedur mit den Elementen ’l’ bzw. ’m’ das Skelett der Regionen. Bei dem Element ’c’ werden Vorder- und Hintergrundanteil vertauscht, damit es den inneren Rand beeinflußt. Wird bei `Iterations` ein sehr großer Werte oder ’maximal’ übergeben, dann bricht die Verarbeitung ab, wenn keine Veränderung mehr auftritt. Für die strukturierenden Elemente gibt es folgende Wahlmöglichkeiten:

’l’ Skelett, ähnlich wie bei `skeleton`. Dieses strukturierende Element wird auch bei `morph_skiz` verwendet.

’m’ Ein Skelett mit vielen „Haaren“ und mehrfachen (parallelen) Ästen.

’d’ Skelett ohne mehrfach Äste aber mit vielen Lücken. Ähnlich zu `morph_skeleton`.

’c’ Gleichmäßige Erosion der Region.

’e’ Ein Pixel dicke Linien werden verkürzt. Dieses strukturierende Element wird auch bei `morph_skiz` verwendet.

’i’ Isolierte Punkte werden entfernt. (Nur `Iterations` = 1 sinnvoll).

’f’ Verzweigungspunkte (y-junctions) werden eliminiert. (Nur `Iterations` = 1 sinnvoll).

’f2’ Es werden ein Pixel lange Äste und Ecken entfernt. (Nur `Iterations` = 1 sinnvoll).

’h’ Eine Art von innerem Rand, der aber dicker als bei `boundary` ist, wird erzeugt, (Nur `Iterations` = 1 sinnvoll).

’k’ Verzweigungspunkte werden elimiert und dabei auch neue erzeugt.

Die Golay-Elemente mit allen möglichen Rotationen sind in der Funktionsbeschreibung von `golay_elements` aufgeführt.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **`RegionThin`** (output_object) ..... region(-array) ⇝ *Hobject*
  Das Ergebnis des Thinning-Operators.
- ▷ **`GolayElement`** (input_control) ..... string ⇝ *string*
  Strukturierendes Element aus dem Golay-Alphabet.
  Defaultwert : ’l’
  Werteliste : `GolayElement` ∈ {’l’, ’m’, ’d’, ’c’, ’e’, ’i’, ’f’, ’f2’, ’h’, ’k’}

▷ **Iterations** (input_control) ..... integer ⇝ *integer* / string
Anzahl der Iterationsschritte. Bei 'f', 'f2', 'h' und 'i' ist nur der Wert 1 sinnvoll.
Defaultwert : 20
Wertevorschläge : Iterations ∈ {'maximal', 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30, 40, 50, 70, 100, 150, 200}
Typischer Wertebereich : 1 ≤ Iterations (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für ein Objekt:

$$O(Iterations \cdot 6 \cdot \sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion thinning_seq den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

thinning_seq ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, union1, watersheds, class_ndim_norm, gen_circle, gen_ellipse, gen_rectangle1, gen_rectangle2, draw_region, gen_region_points, gen_struct_elements, gen_region_polygon_filled

Mögliche Nachfolgerfunktionen

pruning, reduce_domain, select_shape, area_center, connection, complement

Alternativen

skeleton, morph_skiz, expand_region

Siehe auch

hit_or_miss_seq, erosion_golay, difference, thinning_golay, thinning, thickening_seq

Modul

Morphology

top_hat (Region, StructElement : RegionTopHat : :)

Ausbuchtungen von Regionen.

top_hat berechnet das opening von Region mit StructElement. Von der Eingaberegion wird dieses Zwischenergebnis abgezogen. Im Gegensatz zu opening, das Einzelregionen bei entsprechender Voraussetzung in mehrere Regionen auftrennt, bestimmt top_hat die Region, welche bei dieser Trennung entfernt wird.

Die Position von StructElement ist ohne Bedeutung, da opening translationsinvariant bzgl. StructElement ist.

Das strukturierende Element (StructElement) kann mit Prozeduren wie gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse, draw_region, gen_region_polygon, gen_region_points, etc. erzeugt werden.

Parameter

- ▷ **Region** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Regionen, die verarbeitet werden sollen.
- ▷ **StructElement** (input_object) . . . . . region ⇝ *Hobject*
  Strukturierendes Element (lageinvariant).
- ▷ **RegionTopHat** (output_object) . . . . . region(-array) ⇝ *Hobject*
  Ergebnis des Top-Hat-Operators.

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion `top_hat` den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: `set_system('no_object_result',<RegionResult>)`
- leere Region: `set_system('empty_region_result',<RegionResult>)`

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`top_hat` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `union1`, `watersheds`, `class_ndim_norm`, `gen_circle`, `gen_ellipse`, `gen_rectangle1`, `gen_rectangle2`, `draw_region`, `gen_region_points`, `gen_struct_elements`, `gen_region_polygon_filled`

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Alternativen

`opening`, `difference`

Siehe auch

`bottom_hat`, `morph_hat`, `gray_tophat`, `opening`

Modul

Morphology

Kapitel 8

Objekt

8.1 Information

count_obj (Objects : : : Number)

Anzahl von Objekten in einem Tupel.

count_obj bestimmt für den Objektparameter Objects die Anzahl der Objekte, die dieser enthält. Dabei ist zu beachten, daß Objekt nicht gleichzusetzen ist mit Zusammenhangskomponente (siehe connection). Somit ist beispielsweise die Anzahl der Objekte einer Region, die aus drei nicht zusammenhängenden Teilen besteht, gleich 1.

Achtung

In Prolog und Lisp ist die Länge der Liste nicht unbedingt gleich der Anzahl der Objekte. Dies ist der Fall, wenn Objektschlüssel enthalten sind, die im Kompaktmodus erzeugt wurden (Schlüssel aus Kompakt- und Normalmodus können gemischt verwendet werden). Siehe hierzu set_system (::'compact_object',<true/false>:).

Parameter

- ▷ **Objects** (input_object) ..... object-array ⇝ *Hobject*
  Zu untersuchende Objekte.
- ▷ **Number** (output_control) ..... integer ⇝ *integer*
  Anzahl der Objekte im Tupel Objects.

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects}|)$.

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte sind in der HALCON-Datenbank vorhanden, dann liefert count_obj den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen.

Parallelisierungsinformation

count_obj ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

copy_obj, obj_to_integer, connection, set_system

Modul

Basic operators

get_channel_info (Object : : Request, Channel : Information)

Informationen über die Komponenten eines Bildobjektes.

get_channel_info liefert Informationen über die Komponenten eines Bildobjektes. Folgende Abfragen (Request) sind derzeit möglich:

'creator' Ausgabe der Namen der Prozeduren, die die Komponenten (nicht das Objekt) ursprünglich angelegt haben.

'type' Ausgabe des Typs der Bildkomponente ('byte', 'int1', 'int2', 'int4', 'real', 'direction', 'cyclic', 'complex', 'dvf', 'lut'). Die Komponente 0 ist vom Typ 'region' oder 'xld'.

Im Tupel `Channel` werden die Nummern der Komponenten angegeben, über die Informationen gewünscht werden. `Information` enthält nach Ausführung von `get_channel_info` ein Tupel von Strings (je ein String pro Eintrag in `Channel`) mit den gewünschten Informationen.

Parameter

▷ **`Object`** (input_object) ..... object ⇝ *Hobject*
Zu untersuchendes Bildobjekt.

▷ **`Request`** (input_control) ..... string ⇝ *string*
Gewünschte Information über Objektkomponenten.
Defaultwert : 'creator'
Werteliste : `Request` ∈ {'creator', 'type'}

▷ **`Channel`** (input_control) ..... channel(-array) ⇝ *integer*
Zu untersuchende Komponenten (0 für Region/XLD).
Defaultwert : 0
Wertevorschläge : `Channel` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8}

▷ **`Information`** (output_control) ..... string(-array) ⇝ *string*
Angeforderte Information.

Ergebnis

Sind die Parameter korrekt, liefert `get_channel_info` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_channel_info` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_image`

Siehe auch

`count_relation`

Modul

Basic operators

`get_obj_class` (`Object` : : : `Class`)

Name der Klasse eines Bildobjektes.

`get_obj_class` liefert zu jedem Objekt den Namen der zugehörigen Klasse. Folgende Klassen sind möglich:

'image' Objekt mit Region (Definitionsbereich) und mindestens einem Kanal.

'region' Objekt mit einer Region ohne Grauwerte.

'xld_cont' XLD-Objekt als Kontur

'xld_poly' XLD-Objekt als Polygon

'xld_parallel' XLD-Objekt mit parallelen Polygonen

Parameter

▷ **`Object`** (input_object) ..... object(-array) ⇝ *Hobject*
Zu untersuchende Bildobjekte.

▷ **`Class`** (output_control) ..... string(-array) ⇝ *string*
Name der Klasse.

Ergebnis

Sind die Parameter korrekt, liefert `get_obj_class` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_obj_class` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`disp_image`, `disp_region`, `disp_xld`

Siehe auch

`get_channel_info`, `count_relation`

Modul

Basic operators

test_equal_obj (Objects1, Objects2 : : :)

Vergleich von Bildobjekten auf Gleichheit.

`test_equal_obj` vergleicht die Regionen und Grauwertkomponenten aller Objekte der beiden Eingabeparameter. Verglichen wird das n-te Objekt in `Objects1` mit dem n-ten Objekt in `Objects2` (für alle n). Falls alle entsprechenden Regionen gleich sind und auch die Anzahl der Regionen identisch ist, liefert `test_equal_obj` den Wert TRUE, ansonsten FALSE.

Achtung

Bildmatrizen werden nicht auf ihren Inhalt verglichen. Zwei Bilder sind daher „gleich“, wenn sie an der gleichen Stelle im Speicher stehen. Falls die Eingabeparameter leer sind und das Verhalten mit `set_system (::'no_object_result','true':)` gesetzt wurde, liefert `test_equal_obj` TRUE, da alle Eingaben (= leere Menge) gleich sind.

Parameter

- ▷ **Objects1** (input_object) ..... object-array ⇝ *Hobject*
  Testobjekte.
- ▷ **Objects2** (input_object) ..... object-array ⇝ *Hobject*
  Vergleichsobjekte.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität bei Ergebnis TRUE $O(1)$ oder $O(\sqrt{F})$ und bei Ergebnis FALSE $O(\sqrt{F})$.

Ergebnis

`test_equal_obj` liefert den Wert 2 (H_MSG_TRUE), wenn die beiden Objekttupel identisch sind. Unterscheiden sich die Tupel an mindestens einer Stelle, liefert `test_equal_obj` 3 (H_MSG_FALSE). Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Unterscheidet sich die Anzahl der Objekte, wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`test_equal_obj` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

`test_equal_region`

Rückgabewert

bool

Modul

Basic operators

test_equal_region (Regions1, Regions2 : : :)

Test, ob die Regionen von zwei Objekten identisch sind.

`test_equal_region` vergleicht die Regionen der beiden Eingabeparameter. Verglichen wird das n-te Element in `Regions1` mit dem n-ten Objekt in `Regions2` (für alle n). Falls alle Regionen gleich sind und die Anzahl der Regionen identisch ist, liefert `test_equal_region` den Wert TRUE, ansonsten FALSE.

Parameter

▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
Testregionen.

▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
Vergleichsregionen.
Parameteranzahl : `Regions1` = `Regions2`

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität bei Ergebnis TRUE $O(1)$ oder $O(\sqrt{F})$, bei Ergebnis FALSE $O(\sqrt{F})$.

Ergebnis

`test_equal_region` liefert den Wert 2 (H_MSG_TRUE), wenn die beiden Objekttupel identisch sind. Unterscheiden sich die Tupel an mindestens einer Stelle, liefert `test_equal_region` 3 (H_MSG_FALSE). Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Unterscheidet sich die Anzahl der Objekte, wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`test_equal_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`intersection`, `complement`, `area_center`

Siehe auch

`test_equal_obj`

Rückgabewert

bool

Modul

Basic operators

`test_obj_def ( Object : : : )`

Test, ob ein Objekt bereits gelöscht wurde.

`test_obj_def` überprüft, ob das Objekt noch in der HALCON-Datenbank existiert (d.h. ob das Surrogat noch gültig ist). Diese Überprüfung ist vor allem vor dem Löschen von Objekten sinnvoll, wenn nicht ausgeschlossen werden kann, daß durch einen früheren Löschbefehl (`clear_obj`) das Objekt schon gelöscht wurde.

Achtung

`test_obj_def` kann TRUE liefern, auch wenn das Objekt bereits gelöscht wurde. Die Surrogate von gelöschten Objekten werden nämlich wieder für neue Objekte verwendet. Siehe hierzu das Beispiel.

Parameter

▷ **Object** (input_object) ..... object ⇝ *Hobject*
Zu untersuchendes Objekt.

Beispiel

```
Herror err;
circle(&Circle,100.0,100.0,100.0);
err = test_obj_def(Circle);
printf("Result for test_obj_def (H_MSG_TRUE): %d\n",err);
clear_obj(Circle);
err = test_obj_def(Circle);
printf("Result for test_obj_def (H_MSG_FALSE): %d\n",err);
gen_rectangle1(&Rectangle,200.0,200.0,300.0,300.0);
err = test_obj_def(Circle);
printf("Result for test_obj_def (H_MSG_TRUE!!!): %d\n",err);
```

Komplexität

Die Laufzeitkomplexität beträgt $O(1)$.

Ergebnis

`test_obj_def` liefert den Wert 2 (H_MSG_TRUE), falls ein Objekt mit diesem Surrogat in der HALCON-Datenbank vorhanden ist, ansonsten wird der Wert 3 (H_MSG_FALSE) geliefert. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen.

Parallelisierungsinformation

`test_obj_def` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`clear_obj`, `gen_circle`, `gen_rectangle1`

Siehe auch

`set_check`, `clear_obj`, `reset_obj_db`

Rückgabewert

bool

Modul

Basic operators

8.2 Manipulation

`clear_obj` (Objects : : :)

Löschen von Bildobjekten aus der HALCON-Datenbank.

`clear_obj` löscht ikonische Objekte in der Datenbank, die nicht mehr benötigt werden. Es ist zu beachten, daß `clear_obj` in allen Wirtssprachen (außer Smalltalk und C++) die einzige Möglichkeit ist, Objekte in der Datenbank zu löschen und damit den Speicherplatz freizugeben.

Bildmatrizen und Regionen werden von mehreren Bildobjekten gemeinsam verwendet (Speicherplatz!). Dies hat Konsequenzen für das Löschen: Eine Region oder ein Bild werden erst gelöscht, wenn alle Bildobjekte, die sie verwenden gelöscht wurden.

Mit `reset_obj_db` kann das System zurückgesetzt und alle verbliebenen Bildobjekte gelöscht werden.

Achtung

Zur Verwendung von lokalen Variablen. Da beim Verlassen einer Prozedur (oder einer Regel in Prolog) nur die lokalen Variablen beseitigt werden, die Datenbank von HALCON dabei aber nicht modifiziert wird, ist es nötig, vor dem Verlassen der Prozedur lokale Objekte zu löschen. Besondere Vorsicht ist geboten, wenn in Prolog Backtracking auftreten kann (vor dem Erreichen von `clear_obj`).

Parameter

▷ **`Objects`** (input_object) ..... object(-array) ⇝ *Hobject*
Zu löschende Objekte.

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden, dann liefert `clear_obj` den Wert 2 (H_MSG_TRUE). Sind nicht alle Surrogate zuläßig (z.B. schon gelöscht), so führt dies zu einer Exception-Behandlung, die zur Folge hat, daß auch korrekte Surrogate nicht gelöscht werden. Mit dem Aufruf `set_check (::'~clear':)` wird erreicht, daß bei fehlerhaften Surrogaten keine Fehlerbehandlung erfolgt und alle möglichen Löschungen ausgeführt werden. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`clear_obj` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`test_obj_def`

Alternativen

`reset_obj_db`

Siehe auch

`test_obj_def`, `set_check`

Modul

Basic operators

concat_obj (Objects1, Objects2 : ObjectsConcat : :)

Konkatenieren von zwei Objekttupeln zu einem Tupel von Objekten.

concat_obj konkateniert die Objekttupel Objects1 und Objects2 zu einem neuen Tupel ObjectsConcat. Dieses enthält also alle Bildobjekte, die in den beiden Eingabelisten enthalten sind:

ObjectsConcat = [Objects1,Objects2]

In ObjectsConcat werden erst alle Objekte aus Objects1 und dann die Objekte von Objects2 abgelegt. Die Reihenfolge der Objekte wird dabei nicht verändert. Zu beachten ist, daß dabei, wie üblich, alle Objekte, aber nicht die Bilder und Regionen kopiert werden. concat_obj ist speziell für die HALCON/C-Version. Bei Sprachen wie Smalltalk Prolog, Lisp und C++ wird sie nicht benötigt.

concat_obj ist nicht mit union1 oder union2 zu verwechseln, bei denen die Regionen vereinigt werden, also die Anzahl der Objekte verändert wird.

Mit concat_obj ist es möglich, Objekte verschiedener Bildobjekttypen (z.B. Bilder und XLD-Konturen) zu einem Objekt zu konkatenieren. Dies ist ausschließlich dann sinnvoll, wenn es unumgänglich ist, in einer Objektvariable z.B. alle Ergebnisse einer Bildverarbeitungssequenz aufzusammeln. Dabei ist zu beachten, daß die einzigen Operatoren, die diese Art von gemischten Objekttupeln verarbeiten können, concat_obj, copy_obj, select_obj und disp_obj sind. In HDevelop dürfen aus technischen Gründen keine Objekttupel gemischten Typs erzeugt werden.

Parameter

- ▷ **Objects1** (input_object) ..... object(-array) ⇝ *Hobject*
  Objekttupel 1.
- ▷ **Objects2** (input_object) ..... object(-array) ⇝ *Hobject*
  Objekttupel 2.
- ▷ **ObjectsConcat** (output_object) ..... object-array ⇝ *Hobject*
  Enthält (Kopien aller) Objekte aus Objects1 und Objects2.

Beispiel

```
/* generate a tuple of a circle and a rectangle */

gen_circle(&Circle,200.0,400.0,23.0);
gen_rectangle1(&Rectangle,23.0,44.0,203.0,201.0);
concat_obj(Circle,Rectangle,&CirclAndRectangle);
clear_obj(Circle); clear_obj(Rectangle);
disp_region(CircleAndRectangle,WindowHandle);
```

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects1}| + |\texttt{Objects2}|)$;

Speicherplatzkomplexität der Ergebnisdaten: $O(|\texttt{Objects1}| + |\texttt{Objects2}|)$

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden, dann liefert concat_obj den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen.

Parallelisierungsinformation

concat_obj ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

count_obj, copy_obj, select_obj, disp_obj

Modul

Basic operators

copy_obj (Objects : ObjectsSelected : Index, NumObj :)

Kopieren von Bildobjekten in der HALCON-Datenbank.

copy_obj kopiert NumObj Bildobjekte ab Position Index (beginnend mit 1) aus dem Tupel von Eingabeobjekten in Objects. Das resultierende Tupel von Ausgabeobjekten wird in ObjectsSelected abgelegt. Wird für NumObj der Wert -1 verwendet, dann werden alle Objekte ab Index kopiert. Beim Kopieren werden die Region und die Bilder nicht dupliziert. Es werden neue Objekte angelegt, die Verweise auf die Regionen und Bilder enthalten. Die Anzahl von Objekten in einem Tupel kann mit count_obj abgefragt werden.

Parameter

- ▷ **Objects** (input_object) ..... object(-array) ⇝ *Hobject*
  Objekte, die kopiert werden sollen.
- ▷ **ObjectsSelected** (output_object) ..... object(-array) ⇝ *Hobject*
  Kopierte Objekte.
- ▷ **Index** (input_control) ..... integer ⇝ *integer*
  Ab dieser Position in dem Tupel Object werden Bildobjekte kopiert.
  Defaultwert : 1
  Wertevorschläge : Index ∈ {1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000}
  Typischer Wertebereich : 1 ≤ Index
  Restriktion : Index ≤ number(Objects)
- ▷ **NumObj** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu kopierenden Bildobjekte oder -1.
  Defaultwert : 1
  Wertevorschläge : NumObj ∈ {-1, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500, 1000, 2000, 5000}
  Typischer Wertebereich : -1 ≤ NumObj
  Restriktion : ((NumObj + Index) ≤ number(Objects)) ∧ (NumObj ≠ 0)

Beispiel

```
/* 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().
```

Komplexität

Laufzeitkomplexität: $O(|\texttt{Objects}| + \texttt{NumObj})$;

Speicherkomplexität der Ausgabedaten: $O(\texttt{NumObj})$

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden und die Parameter korrekt, dann liefert copy_obj den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

copy_obj ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

count_obj

Alternativen

select_obj

Siehe auch

count_obj, concat_obj, obj_to_integer, copy_image

Modul

Basic operators

gen_empty_obj (: EmptyObject : :)

Erzeugen eines leere Objekttupels.

gen_empty_obj erzeugt eine leeres Tupel. Dies bedeutet, daß der Ausgabeparameter keine Objekte enthält. count_obj liefert also 0. Es kann aber clear_obj für die Ausgabe aufgerufen werden. Es ist zu beachten, daß keine Objekte nicht mit einer leeren Region zu verwechseln sind. Bei einer leeren Region, d.h. einer Region mit 0 Bildpunkten liefert count_obj den Wert 1.

Parameter

▷ **EmptyObject** (output_object) ..... object ⇝ *Hobject*
Keine Objekte.

Parallelisierungsinformation

gen_empty_obj ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Image / region / XLD management

integer_to_obj (: Objects : SurrogateTuple :)

Konversion von „ganzen Zahlen“ zu Surrogaten (Bildobjekten).

integer_to_obj arbeitet invers zu obj_to_integer. Alle Surrogate auf Steuerparameterposition werden als Objekte abgelegt. Es ist zu beachten, daß im Gegensatz zu obj_to_integer die Objekte dupliziert werden. integer_to_obj ist speziell für HALCON/C, da hier Bildobjekte und Steuerparameter unterschiedlich behandelt werden.

Achtung

Die Objekte werden in der Datenbank dupliziert.

Parameter

▷ **Objects** (output_object) ..... object-array ⇝ *Hobject*
Surrogate auf Bildobjektposition.

▷ **SurrogateTuple** (input_control) ..... integer-array ⇝ *integer*
Tuple von Surrogaten.

Ergebnis

integer_to_obj liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind, es sich also um Objektschlüßel handelt. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

integer_to_obj ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

obj_to_integer

Modul

Basic operators

obj_to_integer (Objects : : Index, Number : SurrogateTuple)

Konversion von Surrogaten (Bildobjekten) zu „ganzen Zahlen“.

obj_to_integer speichert ab der angegebenen Stelle (Index) die gewünschte Anzahl von Surrogaten der Objekte (Objektschlüßel) aus Objects als integer-Werte im Ausgabeparameter SurrogateTuple. Damit wird ein Direktzugriff auf ein beliebiges Element von Objects möglich. Im Zusammenhang mit count_obj (liefert die Anzahl der in Objects enthaltenen Objekte) kann man so eine sukzessive Abarbeitung der einzelnen

Objekte in `Objects` durchführen. Die Objekte werden von `obj_to_integer` nicht dupliziert und dürfen deshalb nicht gelöscht (`clear_obj`) werden.

Achtung

Die Daten der Objekte werden nicht dupliziert.

Parameter

- ▷ **`Objects`** (input_object) ..... object-array ⇝ *Hobject*
  Objekte, deren Surrogate ausgegeben werden.
- ▷ **`Index`** (input_control) ..... integer ⇝ *integer*
  Ab diesem Element werden die Surrogate der Objekte ausgegeben.
  Defaultwert : 1
  Typischer Wertebereich : $1 \leq$ `Index`
- ▷ **`Number`** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu übergebenden Surrogate.
  Defaultwert : 1
  Restriktion : (`Number` + `Index`) $\leq$ number(`Objects`)
- ▷ **`SurrogateTuple`** (output_control) ..... integer-array ⇝ *integer*
  Tupel der Surrogate

Beispiel

```
/* Access the i-th element: */
obj_to_integer(Objects,i,1,Surrogat).
```

Komplexität

Laufzeitkomplexität: $O(|$`Objects`$|$ + `Number`)

Ergebnis

`obj_to_integer` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system (::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`obj_to_integer` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`test_obj_def`

Alternativen

`copy_obj`, `select_obj`, `copy_image`, `gen_image_proto`

Siehe auch

`integer_to_obj`, `count_obj`

Modul

Basic operators

`select_obj` (Objects : ObjectSelected : Index :)

Selektieren eines Bildobjektes.

`select_obj` kopiert ein Bildobjekt an der Position `Index` (beginnend mit 1) aus dem Tupel von Eingabeobjekten in `Objects`. Beim Kopieren werden die Region und die Bilder nicht dupliziert. Es werden neue Objekte angelegt, die Verweise auf die Regionen und Bilder enthalten. Die Anzahl von Objekten in einem Tupel kann mit `count_obj` abgefragt werden.

Parameter

- ▷ **`Objects`** (input_object) ..... object(-array) ⇝ *Hobject*
  Objekte, aus denen eines ausgewählt werden sollen.
- ▷ **`ObjectSelected`** (output_object) ..... object ⇝ *Hobject*
  Selektiertes Objekt.

▷ **Index** (input_control) . . . . . integer ⇝ *integer*
Das Objekt mit diesem Index kopieren.
Defaultwert : 1
Wertevorschläge : `Index` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 50, 100, 200, 500, 1000, 2000, 5000}
Typischer Wertebereich : 1 ≤ `Index`

Beispiel

```
/* Access to all Regions */

count_obj(Regions,&Num);
for (i=1; i<=Num; i++)
{
  select_obj(Regions,&Single,i);
  T_get_region_polygon(Single,5.0,&Row,&Column);
  T_disp_polygon(WindowHandleTuple,Row,Column);
  destroy_tuple(Row);
  destroy_tuple(Column);
  clear_obj(Single);
}
```

Komplexität

Laufzeitkomplexität: $O(|$`Objects`$|$

Ergebnis

Sind die Surrogate korrekt, d.h. alle Objekte in der HALCON-Datenbank vorhanden und die Parameter korrekt, dann liefert `select_obj` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system(::'no_object_result',<Result>:)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_obj` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`count_obj`

Alternativen

`copy_obj`

Siehe auch

`count_obj`, `concat_obj`, `obj_to_integer`

Modul

Basic operators

Kapitel 9

Regionen

9.1 Affine-Abbildungen

affine_trans_region (Region : RegionAffineTrans : HomMat2D, Interpolate :)

Transformation einer Region mittels einer affinen Abbildung.

affine_trans_region dient zur Vergrößerung, Verkleinerung, Drehung oder Verschiebung einer Region. Dazu verwendet die Prozedur eine Transformationsmatrix, die in HomMat2D übergeben wird. Diese kann mit den Routinen hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate und hom_mat2d_translate aufgebaut werden. Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Der Parameter Interpolate gibt dabei an, ob die Transformation mit einer internen Interpolation ausgeführt werden soll. Dies kann insbesondere bei Vergrößerungen zu glatteren Regionenrändern führen. Allerdings steigt dadurch die Laufzeit erheblich.

Achtung

affine_trans_region ist i.a. nicht reversibel (Clipping und Rasterung bei Rotation und Streckung).

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu transformierende Region(en).
- ▷ **RegionAffineTrans** (output_object) ..... region(-array) ⇝ *Hobject*
  Region(en), die transformiert wurde(n).
  Parameteranzahl : RegionAffineTrans = Region
- ▷ **HomMat2D** (input_control) ..... affine2d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 6
- ▷ **Interpolate** (input_control) ..... string ⇝ *string*
  Soll die Transformation mit Interpolation ausgeführt werden?
  Defaultwert : 'false'
  Werteliste : Interpolate ∈ {'true', 'false'}

Ergebnis

`affine_trans_region` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`affine_trans_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`hom_mat2d_identity`, `hom_mat2d_scale`, `hom_mat2d_translate`, `hom_mat2d_invert`, `hom_mat2d_rotate`

Mögliche Nachfolgerfunktionen

`select_shape`

Alternativen

`move_region`, `mirror_region`, `zoom_region`

Siehe auch

`affine_trans_image`

Modul

Region processing

`mirror_region` (Region : RegionMirror : RowColumn, WidthHeight :)

Spiegelung von Regionen an der x- oder y-Achse.

`mirror_region` spiegelt die Eingaberegion(en) an der x- oder y-Achse (Parameter `RowColumn`). Der Parameter `WidthHeight` spezifiziert dabei die zugehörige Bildbreite bzw. Bildhöhe. Dieser Wert entspricht also dem Index der Spiegelungsachse * 2.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu transformierende Region(en).
- ▷ **`RegionMirror`** (output_object) ..... region(-array) ⇝ *Hobject*
  Gespiegelte Region(en).
  Parameteranzahl : `RegionMirror` = `Region`
- ▷ **`RowColumn`** (input_control) ..... string ⇝ *string*
  Spiegelungsachse.
  Defaultwert : ’row’
  Werteliste : `RowColumn` ∈ {’column’, ’row’}
- ▷ **`WidthHeight`** (input_control) ..... integer ⇝ *integer*
  Höhe, bzw. Breite des zugehörigen Bildes.
  Defaultwert : 512
  Wertevorschläge : `WidthHeight` ∈ {128, 256, 512, 525, 768, 1024}
  Typischer Wertebereich : 1 ≤ `WidthHeight` ≤ 1024 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `WidthHeight` > `0`

Beispiel

```
read_image(&Image,"affe");
threshold(Image,&Seg,128.0,255.0);
mirror_region(Seg,&Mirror,"row",512);
disp_region(Mirror,WindowHandle);
```

Parallelisierungsinformation

`mirror_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`affine_trans_region`

Siehe auch

`zoom_region`

Modul

Region processing

`move_region ( Region : RegionMoved : Row, Column : )`

Verschieben einer Region.

`move_region` verschiebt alle Eingaberegionen um den Vektor (`Row`, `Column`). Dabei erfolgt gegebenenfalls ein Clipping der Ergebnisregionen an dem aktuellen Bildformat.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Region(en), die verschoben werden soll(en).
- ▷ **`RegionMoved`** (output_object) ..... region(-array) ⇝ *Hobject*
  Modifizierte Region(en).
  Parameteranzahl : `RegionMoved` = `Region`
- ▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
  Um diesen Wert werden alle Regionen vertikal verschoben.
  Defaultwert : 30
  Wertevorschläge : `Row` ∈ {-128, -64, -32, -16, -10, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128}
  Typischer Wertebereich : -512 ≤ `Row` ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
  Um diesen Wert werden alle Regionen horizontal verschoben.
  Defaultwert : 30
  Wertevorschläge : `Column` ∈ {-128, -64, -32, -16, -10, -8, -4, -2, -1, 0, 1, 2, 4, 5, 8, 10, 16, 32, 64, 128}
  Typischer Wertebereich : -512 ≤ `Column` ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität: $O(\sqrt{F})$.

Ergebnis

`move_region` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`move_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Siehe auch

affine_trans_image, mirror_region, zoom_region

Modul

Region processing

transpose_region (Region : Transposed : Row, Column :)

Punktspiegelung einer Region.

transpose_region führt eine Punktspiegelung der Eingaberegionen durch. Der Symmetriepunkt S wird durch Column und Row bestimmt. Der Bildpunkt P' eines Punktes P ist durch folgende Forderung festgelegt:

Ist $P = S$, dann ist $P' = S$. Der Punkt S ist also ein Bezugspunkt. Ist $P \neq S$, dann ist S der Mittelpunkt der Strecke PP'. Die Abbildungsgleichungen der dieser Punktspiegelung mit dem Zentrum $S =$ Column, Row ermittelt man aus den Beziehungen:

$$\begin{aligned} \texttt{Column} &= \frac{x + x'}{2} \\ \texttt{Row} &= \frac{y + y'}{2}. \end{aligned}$$

Werden Row und Column auf den Ursprung gesetzt, so erhält man die in der Morphologie übliche Punktspiegelung. Typischerweise wird transpose_region zur Punktspiegelung eines strukturierenden Elementes verwendet.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die gespiegelt werden.
- ▷ **Transposed** (output_object) ..... region(-array) ⇝ *Hobject*
  Transponierte Regionen
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Bezugspunktes.
  Defaultwert : 0
  Wertevorschläge : Row ∈ {0, 64, 128, 256, 512}
  Typischer Wertebereich : 0 ≤ Row ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Bezugspunktes.
  Defaultwert : 0
  Wertevorschläge : Column ∈ {0, 64, 128, 256, 512}
  Typischer Wertebereich : 0 ≤ Column ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität für eine Region:

$$O(\sqrt{F}) \ .$$

Ergebnis

Bei korrekter Parametrisierung liefert die Funktion transpose_region den Wert 2 (H_MSG_TRUE). Das Funktionsverhalten für die beiden Fälle leere und keine Eingaberegion läßt sich wie folgt kontrollieren:

- keine Region: set_system('no_object_result',<RegionResult>)
- leere Region: set_system('empty_region_result',<RegionResult>)

Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`transpose_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`reduce_domain`, `select_shape`, `area_center`, `connection`

Siehe auch

`dilation1`, `opening`, `closing`

Modul

Morphology

`zoom_region` (Region : RegionZoom : ScaleWidth, ScaleHeight :)

Zooming von Regionen.

`zoom_region` vergrößert oder verkleinert die Regionen `Region` in x- und y-Richtung um die angegebenen Skalierungsfaktoren `ScaleWidth` und `ScaleHeight`.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu transformierende Region(en).
- ▷ **`RegionZoom`** (output_object) ..... region(-array) ⇝ *Hobject*
  Gezoomte Region(en).
  Parameteranzahl : `RegionZoom` = `Region`
- ▷ **`ScaleWidth`** (input_control) ..... extent.x ⇝ *real*
  Skalierungsfaktor in x-Richtung.
  Defaultwert : 2.0
  Wertevorschläge : `ScaleWidth` ∈ {0.25, 0.5, 1.0, 2.0, 3.0}
  Typischer Wertebereich : 0.0 ≤ `ScaleWidth` ≤ 100.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.5
- ▷ **`ScaleHeight`** (input_control) ..... extent.y ⇝ *real*
  Skalierungsfaktor in y-Richtung.
  Defaultwert : 2.0
  Wertevorschläge : `ScaleHeight` ∈ {0.25, 0.5, 1.0, 2.0, 3.0}
  Typischer Wertebereich : 0.0 ≤ `ScaleHeight` ≤ 100.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.5

Parallelisierungsinformation

`zoom_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Siehe auch

`zoom_image_size`, `zoom_image_factor`

Modul

Region processing

9.2 Generierung

gen_checker_region (: RegionChecker : WidthRegion, HeightRegion, WidthPattern, HeightPattern :)

Erzeugen einer schachbrettartigen Region.

gen_checker_region liefert eine schachbrettartige Region. Dabei gehören jeweils die schwarzen Felder des Schachbretts zur Region. Die horizontale und vertikale Ausdehnung der Region wird durch WidthRegion bzw. HeightRegion begrenzt, die Größe der Felder des Schachbrettes durch WidthPattern $\times$ HeightPattern.

Achtung

Wird das Muster sehr klein gewählt (WidthPattern < 4) und die Begrenzung groß, so benötigt die erzeugte Region viel Speicher.

Parameter

- ▷ **RegionChecker** (output_object) ..... region $\rightsquigarrow$ *Hobject*
  Erzeugte Schachbrett-Region.
- ▷ **WidthRegion** (input_control) ..... extent.x $\rightsquigarrow$ *integer*
  Größter auftretender x-Wert der Region.
  Defaultwert : 511
  Wertevorschläge : WidthRegion $\in \{10, 20, 31, 63, 127, 255, 300, 400, 511\}$
  Typischer Wertebereich : $1 \leq$ WidthRegion ≤ 1024 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : WidthRegion ≥ 1
- ▷ **HeightRegion** (input_control) ..... extent.y $\rightsquigarrow$ *integer*
  Größter auftretender y-Wert der Region.
  Defaultwert : 511
  Wertevorschläge : HeightRegion $\in \{10, 20, 31, 63, 127, 255, 300, 400, 511\}$
  Typischer Wertebereich : $1 \leq$ HeightRegion ≤ 1024 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : HeightRegion ≥ 1
- ▷ **WidthPattern** (input_control) ..... extent.y $\rightsquigarrow$ *integer*
  Breite eines Feldes des Schachbrettes.
  Defaultwert : 64
  Wertevorschläge : WidthPattern $\in \{1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500\}$
  Typischer Wertebereich : $1 \leq$ WidthPattern ≤ 1024 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : (WidthPattern > 0) $\wedge$ (WidthPattern $<$ WidthRegion)
- ▷ **HeightPattern** (input_control) ..... extent.y $\rightsquigarrow$ *integer*
  Höhe eines Feldes des Schachbrettes.
  Defaultwert : 64
  Wertevorschläge : HeightPattern $\in \{1, 2, 4, 8, 16, 20, 32, 64, 100, 128, 200, 300, 500\}$
  Typischer Wertebereich : $1 \leq$ HeightPattern ≤ 1024 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : (HeightPattern > 0) $\wedge$ (HeightPattern $<$ HeightRegion)

Beispiel

```
gen_checker_region(Checker,512,512,32,64:)
set_draw(WindowHandle,'fill')
set_part(WindowHandle,0,0,511,511)
disp_region(Checker,WindowHandle)
```

Komplexität

Der benötigte Speicher (in Byte) für die Region ist:

$O((\texttt{WidthRegion} * \texttt{HeightRegion})/\texttt{WidthPattern})$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `gen_checker_region` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch `set_system ('clip_region',<'true'/'false'>)` festgelegt.

Parallelisierungsinformation

`gen_checker_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`paint_region`

Alternativen

`gen_grid_region`, `gen_region_polygon_filled`, `gen_region_points`, `gen_region_runs`, `gen_rectangle1`, `concat_obj`, `gen_random_region`, `gen_random_regions`

Siehe auch

`hamming_change_region`, `reduce_domain`

Modul

Region processing

`gen_circle ( : Circle : Row, Column, Radius : )`

Erzeugen eines Kreises.

`gen_circle` generiert einen oder mehrere Kreise, die durch Mittelpunkt und `Radius` beschrieben werden. Sollen mehrere Kreise erzeugt werden, sind die Koordinaten in Form von Tupeln zu übergeben.

Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch `get_system` und `reset_obj_db`.

Wird für den Radius ein ganzzahliger Wert (1,2,3..) angegeben, so ergibt sich ein geradzahliger Durchmesser und damit ein unsymmetrischer Kreis. Bei ungeradzahligem Durchmesser (Radius = 1.5,2.5,3.5...) erhält man einen symmetrischen Kreis.

Falls der Kreis über den Bildrand reicht, wird je nach dem Wert des Systemflags ’clip_region’ (`set_system`) der Kreis auf das aktuelle Bildformat beschnitten.

Parameter

- ▷ **`Circle`** (output_object) . . . . . region(-array) ⇝ *Hobject*
  Erzeugter Kreis.
- ▷ **`Row`** (input_control) . . . . . circle.center.y(-array) ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert : 200.0
  Wertevorschläge : `Row` ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
  Typischer Wertebereich : 1.0 ≤ `Row` ≤ 1024.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **`Column`** (input_control) . . . . . circle.center.x(-array) ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert : 200.0
  Wertevorschläge : `Column` ∈ {0.0, 10.0, 50.0, 100.0, 200.0, 300.0}
  Typischer Wertebereich : 1.0 ≤ `Column` ≤ 1024.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **`Radius`** (input_control) . . . . . circle.radius(-array) ⇝ *real* / integer
  Radius des Kreises.
  Defaultwert : 100.5
  Wertevorschläge : `Radius` ∈ {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}
Typischer Wertebereich : 1.0 ≤ Radius ≤ 1024.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : Radius > 0.0

Beispiel

```
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).
```

Komplexität

Laufzeitkomplexität: $O(\texttt{Radius} * 2)$

Speicherplatzkomplexität (Byte): $O(\texttt{Radius} * 8)$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_circle den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'true'/'false'>) festgelegt. Entsteht durch Clipping (der Kreis liegt völlig außerhalb des Bildformats) eine leere Region, dann legt set_system ('store_empty_region',<true/false>) fest, ob die leere Region ausgegeben wird.

Parallelisierungsinformation

gen_circle ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain

Alternativen

gen_ellipse, gen_region_polygon_filled, gen_region_points, gen_region_runs, draw_circle

Siehe auch

disp_circle, set_shape, smallest_circle, reduce_domain

Modul

Region processing

gen_ellipse (: Ellipse : Row, Column, Phi, Radius1, Radius2 :)

Erzeugen einer Ellipse.

gen_ellipse generiert eine oder mehrere Ellipsen mit dem Schwerpunkt (Row, Column), der Orientierung Phi und den Halbradien Radius1 und Radius2. Der Winkel wird in Bogenmaß bzgl. der x-Achse in mathematisch positivem Drehsinn angegeben. Es kann mehr als eine Region erzeugt werden, indem Tupel von Parameterwerten übergeben werden.

Der Schwerpunkt muß innerhalb der Bildkoordinaten liegen. Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch get_system und reset_obj_db. Falls die Ellipse über den Bildrand reicht, wird je nach dem Wert des Systemflags 'clip_region' (set_system) die Ellipse auf das aktuelle Bildformat beschnitten.

Parameter

▷ **Ellipse** (output_object) . . . . . region(-array) ⇝ *Hobject*
Erzeugte Ellipse(n).

▷ **Row** (input_control) . . . . . ellipse.center.y(-array) ⇝ *real* / integer
Zeilenindex des Schwerpunktes.
Defaultwert : 200.0
Wertevorschläge : Row ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich : 1.0 ≤ Row ≤ 1024.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Column** (input_control) . . . . . ellipse.center.x(-array) ⇝ *real* / integer
Spaltenindex des Schwerpunktes.
Defaultwert : 200.0
Wertevorschläge : Column ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich : 1.0 ≤ Column ≤ 1024.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Phi** (input_control) . . . . . ellipse.angle.rad(-array) ⇝ *real* / integer
Orientierung des längeren Radius (Radius1).
Defaultwert : 0.0
Wertevorschläge : Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Typischer Wertebereich : -1.178097 ≤ Phi ≤ 1.178097 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **Radius1** (input_control) . . . . . ellipse.radius1(-array) ⇝ *real* / integer
Längerer Radius.
Defaultwert : 100.0
Wertevorschläge : Radius1 ∈ {2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich : 1.0 ≤ Radius1 ≤ 1024.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : Radius1 > 0

▷ **Radius2** (input_control) . . . . . ellipse.radius2(-array) ⇝ *real* / integer
Kürzerer Radius.
Defaultwert : 60.0
Wertevorschläge : Radius2 ∈ {1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Typischer Wertebereich : 1.0 ≤ Radius2 ≤ 1024.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : (Radius2 > 0) ∧ (Radius2 ≤ Radius1)

Beispiel

```
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).
```

Komplexität

Laufzeitkomplexität: $O(\texttt{Radius1} * 2)$

Speicherplatzkomplexität (Byte): $O(\texttt{Radius1} * 8)$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_ellipse den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_ellipse ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`paint_region`, `reduce_domain`

Alternativen

`gen_circle`, `gen_region_polygon_filled`, `draw_ellipse`

Siehe auch

`disp_ellipse`, `set_shape`, `smallest_circle`, `reduce_domain`

Modul

Region processing

`gen_empty_region` (: EmptyRegion : :)

Erzeugen eine leere Region.

`gen_empty_region` erzeugt eine leere Region. Dies bedeutet, daß der Ausgabeparameter ein Objekt enthält. `count_obj` liefert also 1. Die Fläche der Region ist 0. Die meißten Formmerkmale sind undefiniert (0). Es ist zu beachten, daß eine leere Region nicht mit dem leeren Tupel zu verwechseln ist.

Parameter

▷ **`EmptyRegion`** (output_object) . . . . . region ⇝ *Hobject*
Leere Region (keine Punkte).

Parallelisierungsinformation

`gen_empty_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Region processing

`gen_grid_region` (: RegionGrid : RowSteps, ColumnSteps, Type, Width, Height :)

Erzeugen einer Region aus Linien oder Punkten.

`gen_grid_region` erzeugt ein Raster, das aus Linien (`Type` = ’lines’) oder Punkten (`Type` = ’points’) aufgebaut ist. Bei ’lines’ werden durchgezogene Linien bei ’points’ nur die Schnittpunkte der Linien ausgegeben. Ausgehend von dem Punkt (0,0) bis zum Punkt (`Height`-1,`Width`-1) wird das Raster in der Schrittweite `RowSteps` in Zeilenrichtung und `ColumnSteps` in Spaltenrichtung aufgebaut. Im Modus ’lines’ kann `RowSteps` bzw. `ColumnSteps` auf Null gesetzt werden. In diesem Fall werden nur Spalten bzw. Zeilen erzeugt.

Achtung

Wird das Muster sehr klein gewählt (`RowSteps` $<$ 4 or `ColumnSteps` $<$ 4), so benötigt die erzeugte Region viel Speicher.

Im Modus ’points’ dürfen `RowSteps` und `ColumnSteps` nicht auf Null gesetzt werden.

Parameter

▷ **`RegionGrid`** (output_object) . . . . . region ⇝ *Hobject*
Erzeugte Linien/Punkte-Region.

▷ **`RowSteps`** (input_control) . . . . . extent.y ⇝ *integer* / real
Schrittweite in Zeilenrichtung oder Null.
Defaultwert : 10
Wertevorschläge : `RowSteps` $\in \{0, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 100\}$
Typischer Wertebereich : $0 \leq$ `RowSteps` ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : (`RowSteps` $>$ `1`) $\vee$ (`RowSteps` $=$ `0`)

▷ **ColumnSteps** (input_control) . . . . . extent.x ⇝ *integer* / real
Schrittweite in Spaltenrichtung oder Null.
Defaultwert : 10
Wertevorschläge : ColumnSteps ∈ {0, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 100}
Typischer Wertebereich : 0 ≤ ColumnSteps ≤ 512 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : (ColumnSteps > 1) ∨ (ColumnSteps = 0)

▷ **Type** (input_control) . . . . . string ⇝ *string*
Art des erzeugten Musters.
Defaultwert : 'lines'
Werteliste : Type ∈ {'lines', 'points'}

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Maximale Breite des Musters.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Width ≥ 1

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Maximale Höhe des Musters.
Defaultwert : 512
Wertevorschläge : Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Height ≥ 1

Beispiel

```
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).
```

Komplexität

Der benötigte Speicher (in Byte) für die Region ist:
$O((ImageWidth/ColumnSteps) * (ImageHeight/RowSteps))$

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_grid_region den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_grid_region ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

reduce_domain, paint_region

Alternativen

gen_region_line, gen_region_polygon, gen_region_points, gen_region_runs

Siehe auch

gen_checker_region, reduce_domain

Modul

Region processing

gen_random_region (: RegionRandom : Width, Height :)

Erzeugen einer Zufallsregion.

gen_random_region liefert eine Zufallsregion. Dabei wird im Bildbereich $[0 \ldots \texttt{Width}-1][0 \ldots \texttt{Height}-1]$ jeder Punkt mit Wahrscheinlichkeit 0.5 in die Region aufgenommen. Man kann sich die erzeugte Region als die Schwellenwertbildung in einem Bild mit Rauschen vorstellen.

Diese Prozedur ist insbesondere wichtig für die Erzeugung von unkorrelierten Binärmustern. Das Zufallsmuster wird mit der C-Funktion "nrand48()" erzeugt.

Achtung

Wenn Width und Height groß gewählt werden (> 100), dann kann die erzeugte Region durch die intern verwendete Lauflängenkodierung viel Speicherplatz benötigen. Die Grauwerte der Ausgaberegion sind undefiniert.

Parameter

- ▷ **RegionRandom** (output_object) ..... region ⇝ *Hobject*
  Erzeugte Zufallsregion mit Ausdehnung Width x Height.
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Maximale horizontale Ausdehnung der Zufallsregion.
  Defaultwert : 128
  Wertevorschläge : Width ∈ {16, 32, 50, 64, 100, 128, 256, 300, 400, 512}
  Typischer Wertebereich : $1 \leq$ Width ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : Width > 0
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Maximale vertikale Ausdehnung der Zufallsregion.
  Defaultwert : 128
  Wertevorschläge : Height ∈ {16, 32, 50, 64, 100, 128, 256, 300, 400, 512}
  Typischer Wertebereich : $1 \leq$ Height ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : Height > 0

Komplexität

Der schlimmste Fall für die Speicherplatzkomplexität für die erzeugte Region (in Byte) ist: $O(Width*Height*2)$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_random_region den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_random_region ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain

Siehe auch

gen_checker_region, hamming_change_region, add_noise_distribution, add_noise_white, reduce_domain

Modul

Region processing

gen_random_regions (: Regions : Type, WidthMin, WidthMax, HeightMin, HeightMax, PhiMin, PhiMax, NumRegions, Width, Height :)

Erzeugen zufälliger Regionen wie Kreise, Rechtecke und Ellipsen.

gen_random_region dient zum Erzeugen von Kreisen, Rechtecken und Ellipsen, deren Parameter zufällig bestimmt werden. Es wird jeweils nur eine Unter- bzw. Obergrenze angegeben. Die Position ist immer zufällig

und kann nicht über Parameter bestimmt werden. Der Parameter `NumRegions` gibt an wie viele Regionen erzeugt werden sollen.

Parameter

- ▷ **`Regions`** (output_object) . . . . . region-array ⇝ *Hobject*
  Erzeugte Regionen.
- ▷ **`Type`** (input_control) . . . . . string ⇝ *string*
  Art der zu erzeugenden Regionen.
  Defaultwert : 'circle'
  Werteliste : `Type` ∈ {'circle', 'ring', 'ellipse', 'rectangle1', 'rectangle2'}
- ▷ **`WidthMin`** (input_control) . . . . . number ⇝ *real* / integer
  Minimale Breite der Region.
  Defaultwert : 10.0
  Wertevorschläge : `WidthMin` ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
  Typischer Wertebereich : 1.0 ≤ `WidthMin` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : `WidthMin` > `0`
- ▷ **`WidthMax`** (input_control) . . . . . number ⇝ *real* / integer
  Maximale Breite der Region.
  Defaultwert : 20.0
  Wertevorschläge : `WidthMax` ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
  Typischer Wertebereich : 1.0 ≤ `WidthMax` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : `WidthMax` > `0`
- ▷ **`HeightMin`** (input_control) . . . . . number ⇝ *real* / integer
  Minimale Höhe der Region.
  Defaultwert : 10.0
  Wertevorschläge : `HeightMin` ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
  Typischer Wertebereich : 1.0 ≤ `HeightMin` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : `HeightMin` > `0`
- ▷ **`HeightMax`** (input_control) . . . . . number ⇝ *real* / integer
  Maximale Höhe der Region.
  Defaultwert : 30.0
  Wertevorschläge : `HeightMax` ∈ {1.0, 3.0, 5.0, 10.0, 20.0, 40.0, 80.0}
  Typischer Wertebereich : 1.0 ≤ `HeightMax` ≤ 511.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : `HeightMax` > `0`
- ▷ **`PhiMin`** (input_control) . . . . . number ⇝ *real* / integer
  Minimaler Rotationswinkel der Region.
  Defaultwert : -0.7854
  Wertevorschläge : `PhiMin` ∈ {0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5}
  Typischer Wertebereich : 0.0 ≤ `PhiMin` ≤ 6.28 (lin)
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.10
  Restriktion : `PhiMin` > `0`
- ▷ **`PhiMax`** (input_control) . . . . . number ⇝ *real* / integer
  Maximaler Rotationswinkel der Region.
  Defaultwert : 0.7854
  Wertevorschläge : `PhiMax` ∈ {0.0, 0.1, 0.3, 0.6, 0.9, 1.2, 1.5}
  Typischer Wertebereich : 0.0 ≤ `PhiMax` ≤ 6.28 (lin)
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.10
  Restriktion : `PhiMax` > `0`

▷ **NumRegions** (input_control) . . . . . integer ⇝ *integer*
Anzahl der Regionen.
Defaultwert : 100
Wertevorschläge : NumRegions ∈ {1, 5, 20, 100, 200, 500, 1000, 2000}
Typischer Wertebereich : 1 ≤ NumRegions ≤ 2000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : NumRegions > 0

▷ **Width** (input_control) . . . . . integer ⇝ *integer*
Maximale horizontale Ausdehnung.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Width > 0

▷ **Height** (input_control) . . . . . integer ⇝ *integer*
Maximale vertikale Ausdehnung.
Defaultwert : 512
Wertevorschläge : Height ∈ {128, 256, 512, 1024}
Typischer Wertebereich : 1 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : Height > 0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_random_regions den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_random_regions ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

paint_region

Modul

Region processing

gen_rectangle1 (: Rectangle : Row1, Column1, Row2, Column2 :)

Erzeugen eines Rechtecks parallel zu den Koordinatenachsen.

gen_rectangle1 generiert ein oder mehrere Rechtecke parallel zu den Koordinatenachsen, welche durch das linke obere Eck (Row1, Column1) und das rechte untere Eck (Row2, Column2) beschrieben werden. Es kann mehr als eine Region erzeugt werden, indem ein Tupel von Eckpunkten übergeben wird. Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch get_system und reset_obj_db.

Parameter

▷ **Rectangle** (output_object) . . . . . region(-array) ⇝ *Hobject*
Erzeugtes Rechteck.

▷ **Row1** (input_control) . . . . . rectangle.origin.y(-array) ⇝ *real* / integer
Zeile des linken oberen Eckpunkts.
Defaultwert : 30.0
Wertevorschläge : Row1 ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 200.0}
Typischer Wertebereich : −∞ ≤ Row1 ≤ ∞ (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Column1** (input_control) . . . . . rectangle.origin.x(-array) ⇝ *real* / integer
Spalte des linken oberen Eckpunkts.
Defaultwert : 20.0
Wertevorschläge : Column1 ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 200.0}
Typischer Wertebereich : $-\infty \leq$ Column1 $\leq \infty$ (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Row2** (input_control) . . . . . rectangle.corner.y(-array) ⇝ *real* / integer
Zeile des rechten unteren Eckpunkts.
Defaultwert : 100.0
Wertevorschläge : Row2 ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0}
Typischer Wertebereich : $-\infty \leq$ Row2 $\leq \infty$ (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : Row2 $\geq$ Row1

▷ **Column2** (input_control) . . . . . rectangle.corner.x(-array) ⇝ *real* / integer
Spalte des rechten unteren Eckpunkts.
Defaultwert : 200.0
Wertevorschläge : Column2 ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0, 511.0}
Typischer Wertebereich : $-\infty \leq$ Column2 $\leq \infty$ (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : Column2 $\geq$ Column1

Beispiel

```
/* 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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_rectangle1 den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_rectangle1 ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain

Alternativen

gen_rectangle2, gen_region_polygon, fill_up, gen_region_runs, gen_region_points, gen_region_line

Siehe auch

draw_rectangle1, reduce_domain, smallest_rectangle1

Modul

Region processing

gen_rectangle2 (: Rectangle : Row, Column, Phi, Length1, Length2 :)

Erzeugen eines beliebig orientierten Rechtecks.

gen_rectangle2 generiert ein oder mehrere Rechtecke mit dem Schwerpunkt (Row, Column) , der Orientierung Phi und den halben Kantenlängen Length1 und Length2. Die Orientierung wird in Bogenmaß angegeben und gibt den Winkel zwischen der horizontalen Achse und Length1 an (mathematisch positiv). Das Koordinatensystem läuft von (0,0) (linkes oberes Eck) bis (Width-1,Height-1). Siehe hierzu auch get_system und reset_obj_db. Es kann mehr als eine Region erzeugt werden, indem ein Tupel von Eckpunkten übergeben wird.

Achtung

Die Grauwerte der Ausgabeobjekte sind undefiniert.

Parameter

- ▷ **Rectangle** (output_object) ..... region(-array) ⇝ *Hobject*
  Erzeugtes Rechteck.
- ▷ **Row** (input_control) ..... rectangle2.center.y(-array) ⇝ *real* / integer
  Zeilenindex des Schwerpunktes.
  Defaultwert : 50.0
  Wertevorschläge : Row ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typischer Wertebereich : $-\infty \leq$ Row $\leq \infty$ (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **Column** (input_control) ..... rectangle2.center.x(-array) ⇝ *real* / integer
  Spaltenindex des Schwerpunktes.
  Defaultwert : 100.0
  Wertevorschläge : Column ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
  Typischer Wertebereich : $-\infty \leq$ Column $\leq \infty$ (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **Phi** (input_control) ..... rectangle2.angle.rad(-array) ⇝ *real* / integer
  Winkel der Längsachse zur Horizontalen (Bogenmaß).
  Defaultwert : 0.0
  Wertevorschläge : Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
  Typischer Wertebereich : -1.178097 $\leq$ Phi $\leq$ 1.178097 (lin)
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.1
  Restriktion : $((-\text{pi}/2) < \text{Phi}) \wedge (\text{Phi} \leq (\text{pi}/2))$
- ▷ **Length1** (input_control) ..... rectangle2.hwidth(-array) ⇝ *real* / integer
  Halbe Breite.
  Defaultwert : 200.0
  Wertevorschläge : Length1 ∈ {3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0}
  Typischer Wertebereich : $-\infty \leq$ Length1 $\leq \infty$ (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **Length2** (input_control) ..... rectangle2.hheight(-array) ⇝ *real* / integer
  Halbe Höhe.
  Defaultwert : 100.0
  Wertevorschläge : Length2 ∈ {1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0}
  Typischer Wertebereich : $-\infty \leq$ Length2 $\leq \infty$ (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : Length2 $\leq$ Length1

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_rectangle2 den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_rectangle2 ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain

Alternativen

gen_rectangle1, gen_region_polygon_filled, gen_region_polygon, gen_region_points, fill_up

Siehe auch

draw_rectangle2, reduce_domain, smallest_rectangle2, gen_ellipse

Modul

Region processing

gen_region_histo (: Region : Histogram, Row, Column, Scale :)

Umwandlung eines Histogramms in eine Region.

gen_region_histo wandelt ein Histogramm, das mit gray_histo erzeugt wurde, in eine Region um. Die Wirkung der Steuerparameter entspricht denen bei der Ausgabe mit disp_image und set_paint.

Parameter

- ▷ **Region** (output_object) ..... region ⇝ *Hobject*
  Darstellung des Histogramms als Region.
- ▷ **Histogram** (input_control) ..... histogram-array ⇝ *integer*
  Eingabehistogramm.
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenkoordinate des Mittelpunktes.
  Defaultwert : 255
  Wertevorschläge : Row ∈ {100, 200, 255, 300, 400}
  Typischer Wertebereich : 0 ≤ Row ≤ 511
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenkoordinate des Mittelpunktes.
  Defaultwert : 255
  Wertevorschläge : Column ∈ {100, 200, 255, 300, 400}
  Typischer Wertebereich : 0 ≤ Column ≤ 511
- ▷ **Scale** (input_control) ..... integer ⇝ *integer*
  Verkleinerungsfakor.
  Defaultwert : 1
  Werteliste : Scale ∈ {1, 2, 3, 4, 5, 6, 7}
  Typischer Wertebereich : 1 ≤ Scale ≤ 10 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Ergebnis

gen_region_histo liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_region_histo ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gray_histo

Siehe auch

disp_channel, set_paint

Modul

Region processing

gen_region_hline (: Regions : Orientation, Distance :)

Abspeichern von in Hessescher Normalform beschriebenen Eingabelinien als Regionen.

gen_region_hline legt die in Hessescher Normalform beschriebenen Linien als Regionen ab. Eine Linie wird bestimmt durch den Abstand der Linie vom Ursprung (Distance, entpricht der Länge des Normalenvektors) und der Richtung des Normalenvektors (Orientation, entspricht der Orientierung der Linie $\pm\pi/2$). Die Richtungen wurden so definiert, daß bei Orientation = 0 der Normalenvektor in Richtung der X-Achse liegt, was einer senkrechten Linie entspricht. Bei Orientation = $\pi/2$ zeigt der Normalenvektor in Richtung der Y-Achse, d.h. es wird eine waagerechte Linie beschrieben.

Achtung

Die Linien werden am derzeitigen maximalem Bildformat geclippt.

Parameter

- ▷ **Regions** (output_object) ..... region(-array) ⇝ *Hobject*
  Erzeugte Regionen (für jede Linie eine), beschnitten auf maximales Bildformat.
  Parameteranzahl : Regions = Distance
- ▷ **Orientation** (input_control) ..... hesseline.angle.rad(-array) ⇝ *real* / integer
  Orientierung des Normalenvektors in Bogenmaß.
  Defaultwert : 0.0
  Wertevorschläge : Orientation ∈ {-0.78, 0.0, 0.78, 1.57}
  Typischer Wertebereich : $-\infty \leq$ Orientation $\leq \infty$ (lin)
  Empfohlene Schrittweite : 0.02
  Parameteranzahl : Orientation = Distance
- ▷ **Distance** (input_control) ..... hesseline.distance(-array) ⇝ *real* / integer
  Abstand der Linie vom Koordinatenursprung (0,0).
  Defaultwert : 200
  Wertevorschläge : Distance ∈ {10, 50, 100, 200, 300, 400}
  Typischer Wertebereich : $-\infty \leq$ Distance $\leq \infty$ (lin)
  Empfohlene Schrittweite : 1

Ergebnis

gen_region_hline liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

gen_region_hline ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

gen_region_line

Siehe auch

hough_lines

Modul

Region processing

gen_region_line (: RegionLines : BeginRow, BeginCol, EndRow, EndCol :)

Abspeichern von Eingabelinien als Regionen.

gen_region_line legt die übergebenen Linien (mit Anfangspunkt [BeginRow,BeginCol] und Endpunkt [EndRow, EndCol]) als Region ab.

Parameter

- ▷ **RegionLines** (output_object) ..... region(-array) ⇝ *Hobject*
  Erzeugte Regionen.
- ▷ **BeginRow** (input_control) ..... line.begin.y(-array) ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Eingabelinien.
  Defaultwert : 100
  Wertevorschläge : BeginRow ∈ {10, 50, 100, 200, 300, 400}
  Typischer Wertebereich : $-\infty \leq$ BeginRow $\leq \infty$ (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

▷ **BeginCol** (input_control) ..... line.begin.x(-array) ⇝ *integer*
Spaltenkoordinaten der Anfangspunkte der Eingabelinien.
Defaultwert : 50
Wertevorschläge : BeginCol ∈ {10, 50, 100, 200, 300, 400}
Typischer Wertebereich : $-\infty \leq$ BeginCol $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **EndRow** (input_control) ..... line.end.y(-array) ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Eingabelinien.
Defaultwert : 150
Wertevorschläge : EndRow ∈ {50, 100, 200, 300, 400, 500}
Typischer Wertebereich : $-\infty \leq$ EndRow $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **EndCol** (input_control) ..... line.end.x(-array) ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Eingabelinien.
Defaultwert : 250
Wertevorschläge : EndCol ∈ {50, 100, 200, 300, 400, 500}
Typischer Wertebereich : $-\infty \leq$ EndCol $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Ergebnis

gen_region_line liefert immer den Wert 2 (H_MSG_TRUE). Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'true'/'false'>) festgelegt.

Parallelisierungsinformation

gen_region_line ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

split_skeleton_lines

Alternativen

gen_region_hline

Modul

Region processing

gen_region_points (: Region : Rows, Columns :)

Abspeichern einzelner Punkte als Bildregion.

gen_region_points erzeugt eine Region, die durch eine Anzahl von Punkten beschrieben wird. Die Punkte müßen in keiner festen Reihenfolge abgespeichert werden, doch ergibt sich das beste Laufzeitverhalten, wenn die Punkte aufsteigend sortiert sind. Die Ordnung lautet:

$$(l_1, c_1) \leq (l_2, c_2) := (l_1 < l_2) \quad \vee \quad (l_1 = l_2) \wedge (c_1 \leq c_2)$$

Die angegebenen Koordinaten stehen für zwei aufeinanderfolgende Punkte in dem Tupel.

Parameter

▷ **Region** (output_object) ..... region ⇝ *Hobject*
Erzeugte Region.

▷ **Rows** (input_control) ..... coordinates.y(-array) ⇝ *integer*
Zeilen der Punkte in der Region.
Defaultwert : 100
Wertevorschläge : Rows ∈ {0, 10, 30, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Rows $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Columns** (input_control) ..... coordinates.x(-array) ⇝ *integer*
Spalten der Punkte in der Region.
Defaultwert : 100
Wertevorschläge : Columns ∈ {0, 10, 30, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Columns $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Parameteranzahl : Columns = Rows

Komplexität

Sei F die Anzahl der Punkte. Falls die Punkte aufsteigend sortiert sind, dann ist die Laufzeitkomplexität: $O(F)$, sonst $O(\log(F) * F)$.

Ergebnis

gen_region_points liefert den Wert 2 (H_MSG_TRUE), falls die Punkte innerhalb des Bildformats liegen. Ansonsten wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'true'/'false'>) festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt set_system('store_empty_region',<true/false>) fest, ob die Region ausgegeben wird.

Parallelisierungsinformation

gen_region_points ist *wiedereintrittsfähig* („reentrant"), *lokal* auszuführen („local") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_region_points

Mögliche Nachfolgerfunktionen

paint_region, reduce_domain

Alternativen

gen_region_polygon, gen_region_runs, gen_region_line

Siehe auch

reduce_domain

Modul

Region processing

gen_region_polygon (: Region : Rows, Columns :)

Abspeichern eines Polygons als ein Bildobjekt.

gen_region_polygon erzeugt eine Region aus einem Polygonzug, der durch eine Folge von Zeilen- und Spaltenkoordinaten beschrieben wird. Die erzeugte Region besteht aus den Punkten der dadurch definierten Strecken, wobei zwischen den Stützpunkten linear interpoliert wird.

Achtung

Die Region wird nicht automatisch geschlossen und nicht „aufgefüllt". Die Grauwerte der Ausgaberegion sind undefiniert.

Parameter

▷ **Region** (output_object) ..... region ⇝ *Hobject*
Erzeugte Region.

▷ **Rows** (input_control) ..... polygon.y-array ⇝ *integer*
Zeilenindizes der Stützpunkte der Regionenkontur.
Defaultwert : 100
Wertevorschläge : Rows ∈ {0, 10, 30, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Rows $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Columns** (input_control) ..... polygon.x-array ⇝ *integer*
Spaltenindizes der Stützpunkte der Regionenkontur.
Defaultwert : 100
Wertevorschläge : Columns ∈ {0, 10, 30, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Columns $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Parameteranzahl : Columns = Rows

Beispiel

```
/* 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).
```

Ergebnis

Falls die Stützpunkte korrekt sind, liefert gen_region_polygon den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'true'/'false'>) festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt set_system('store_empty_region',<true/false>) fest, ob die Region ausgegeben wird.

Parallelisierungsinformation

gen_region_polygon ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_region_polygon, draw_polygon

Alternativen

gen_region_polygon_filled, gen_region_points, gen_region_runs

Siehe auch

fill_up, reduce_domain, get_region_polygon, draw_polygon

Modul

Region processing

gen_region_polygon_filled (: Region : Rows, Columns :)

Abspeichern eines Polygons als eine „aufgefüllte“ Region.

gen_region_polygon_filled erzeugt eine Region aus einem Polygon, das die Eckpunkte der Region (Zeilen- und Spaltenkoordinaten) entweder im oder gegen den Uhrzeigersinn enthält. Im Gegensatz zu gen_region_polygon wird hier eine „aufgefüllte“ Region zurückgeliefert.

Parameter

▷ **Region** (output_object) ..... region ⇝ *Hobject*
Erzeugte Region.

▷ **Rows** (input_control) ..... polygon.y-array ⇝ *integer*
Zeilenindizes der Stützpunkte der Regionenkontur.
Defaultwert : 100
Wertevorschläge : Rows ∈ {0, 10, 30, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Rows $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Columns** (input_control) ..... polygon.x-array ⇝ *integer*
Spaltenindizes der Stützpunkte der Regionenkontur.
Defaultwert : 100
Wertevorschläge : Columns ∈ {0, 10, 30, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Columns $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Parameteranzahl : Columns = Rows

Beispiel

```
/* Polygon approximation */
T_get_region_polygon(Region,7,&Row,&Column);
T_gen_region_polygon_filled(&Pol,Row,Column);
/* fill up with original gray value */
reduce_domain(Image,Pol,&New);
```

Ergebnis

Falls die Stützpunkte korrekt sind, liefert gen_region_polygon_filled den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system('clip_region',<'true'/'false'>) festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt set_system('store_empty_region',<true/false>) fest, ob die Region ausgegeben wird.

Parallelisierungsinformation

gen_region_polygon_filled ist *wiedereintrittsfähig* („reentrant"), *lokal* auszuführen („local") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_region_polygon, draw_polygon

Alternativen

gen_region_polygon, gen_region_points, draw_polygon

Siehe auch

gen_region_polygon, reduce_domain, get_region_polygon, gen_region_runs

Modul

Region processing

gen_region_runs (: Region : Row, ColumnBegin, ColumnEnd :)

Erzeugen einer Bildregion aus einer Lauflängenkodierung.

gen_region_runs erzeugt eine Region, die über die eingegebene Sehnenstruktur beschrieben ist. Die Sehnendarstellung entsteht, indem man eine Region zeilenweise mit aufsteigender Zeilenzahl (= von „oben" nach „unten") untersucht. Jede Zeile wird von links nach rechts durchlaufen (aufsteigende Spaltenzahl). Dabei werden alle Anfangs- und Endpunkte von Regionenabschnitten (=Sehnen) gespeichert. Eine Region läßt sich somit durch eine Folge von Sehnen beschreiben, wobei eine Sehne durch Zeilennummer, sowie Anfangs- und Endpunkte (Spaltennummer) definiert ist.

Die Abspeicherung erfolgt am schnellsten, wenn die Sehnen sortiert sind. Die Ordnung lautet:

$$(l_1, b_1, e_1) \leq (l_2, b_2, e_2) := (l_1 < l_2) \quad \vee \quad (l_1 = l_2) \wedge (b_1 \leq b_2)$$

.

Parameter

▷ **Region** (output_object) ..... region ⇝ *Hobject*
Erzeugte Region.

▷ **Row** (input_control) ..... chord.y(-array) ⇝ *integer*
Zeilen der Sehnen.
Defaultwert : 100
Wertevorschläge : Row ∈ {0, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ Row $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **ColumnBegin** (input_control) ..... chord.x1(-array) ⇝ *integer*
Spalten der Anfangspunkte der Sehnen.
Defaultwert : 50
Wertevorschläge : ColumnBegin ∈ {0, 50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ ColumnBegin $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Parameteranzahl : ColumnBegin = Row

▷ **ColumnEnd** (input_control) ..... chord.x2(-array) ⇝ *integer*
Spalten der Endpunkte der Sehnen.
Defaultwert : 200
Wertevorschläge : ColumnEnd ∈ {50, 100, 200, 300, 500}
Typischer Wertebereich : $-\infty \leq$ ColumnEnd $\leq \infty$ (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Parameteranzahl : ColumnEnd = Row
Restriktion : ColumnEnd ≥ ColumnBegin

Komplexität

Sei F die Anzahl der Punkte. Falls die Punkte aufsteigend sortiert sind, dann ist die Laufzeitkomplexität: $O(F)$, sonst $O(\log(F) * F)$.

Ergebnis

gen_region_runs liefert den Wert 2 (H_MSG_TRUE), falls die Daten korrekt sind, ansonsten wird eine Exception-Behandlung durchgeführt. Das Clipping am aktuellen Bildformat wird durch set_system ('clip_region',<'true'/'false'>) festgelegt. Wird eine leere Region erzeugt (durch das Clipping oder eine leere Eingabe), dann legt set_system('store_empty_region',<true/false>) fest, ob die Region ausgegeben wird.

Parallelisierungsinformation

gen_region_runs ist *wiedereintrittsfähig* („reentrant"), *lokal* auszuführen („local") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_region_runs

Alternativen

gen_region_points, gen_region_polygon, gen_region_line, gen_region_polygon_filled

Siehe auch

reduce_domain

Modul

Region processing

label_to_region (LabelImage : Regions : :)

Regionen mit gleichen Grauwerten suchen.

label_to_region segmentiert Bilder in Regionen gleichen Grauwerts. Dabei wird für jeden Grauwert im Bild genau eine Ausgaberegion erzeugt. Dies entspricht einem mehrfachen Aufruf der Prozedur threshold, gefolgt von der Konkatenation der entstandenen Regionen (concat_obj). Verwandt ist die Routine auch mit regiongrowing. label_to_region führt allerdings kein connection durch, d.h. die Ausgaberegionen müssen nicht zusammenhängen. Eine typische Anwendung von label_to_region ist die Segmentation „gelabelter" Bilder, daher auch der Name.

Die Anzahl der Ausgaberegionen wird durch den Systemparameter 'max_outp_obj_par' begrenzt, der mittels

get_system(::'max_outp_obj_par':<Anzahl>)

abgefragt werden kann.

Achtung

label_to_region ist nicht für Real-Bilder implementiert. Die Eingabebilder dürfen keine negativen Grauwerte enthalten.

Parameter

- ▷ **LabelImage** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2 / int4
  „Gelabeltes“ Bild.
- ▷ **Regions** (output_object) ..... region-array ⇝ *Hobject*
  Regionen mit konstantem Grauwert.

Komplexität

Sei $x1$ die minimale X-Koordinate, $x2$ die maximale X-Koordinate, $y1$ die minimale Y-Koordinate und $y2$ die maximale Y-Koordinate eines auftretenden Grauwertes und N die Anzahl der verschiedenen Grauwerte, dann ist die Laufzeitkomplexität $O(N * (x2 - x1 + 1) * (y2 - y1 + 1))$

Ergebnis

Sind die Grauwerte in einem zulässigen Breich, dann liefert label_to_region den Wert 2 (H_MSG_TRUE). Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

label_to_region ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

min_max_gray, sobel_amp, gauss_image, reduce_domain, diff_of_gauss

Mögliche Nachfolgerfunktionen

connection, dilation1, erosion1, opening, closing, rank_region, shape_trans, skeleton

Siehe auch

threshold, concat_obj, regiongrowing, region_to_label

Modul

Region processing

9.3 Mengen

complement (Region : RegionComplement : :)

Komplement einer Region.

complement berechnet das Komplement der eingegebenen Region(en).

Wenn das Systemflag **'clip_region'** auf **'false'** steht (siehe set_system), wird das Komplement nur virtuell ausgeführt, d.h. es wird bei Region der Eintrag für das Komplement auf TRUE gesetzt. Im weiteren werden dann die Gesetze von de Morgan verwendet, um die Folgeoperationen korrekt auszuführen.

Wenn das Systemflag **'clip_region'** auf **'true'** gesetzt ist, wird die Differenz aus dem größten aktuellen Bild (siehe reset_obj_db) und der Region berechnet.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Eingaberegion(en).
- ▷ **RegionComplement** (output_object) ..... region(-array) ⇝ *Hobject*
  Berechnete Komplement(e).
  Parameteranzahl : RegionComplement = Region

Ergebnis

`complement` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`complement` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`select_shape`

Siehe auch

`difference`, `union1`, `union2`, `intersection`, `reset_obj_db`, `set_system`

Modul

Region processing

`difference` (Region, Sub : RegionDifference : :)

Differenz von Mengen von Regionen.

`difference` berechnet die flächenmäßige Differenz von Regionen:

(Regionen aus `Region`) − (Regionen aus `Sub`)

Eine Ergebnisregion berechnet sich aus der Eingaberegion (`Region`) minus aller Punkte der Regionen aus `Sub`.

Achtung

Leere Eingaberegionen sind bei beiden Parametern zuläßig. Es ist zu beachten, daß leere Regionen enstehen können. Hierbei ist der Wert von **’store_empty_region’** zu beachten.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu bearbeitende Region(en).
- ▷ **`Sub`** (input_object) ..... region(-array) ⇝ *Hobject*
  Diese Region(en) (ihre Vereinigung) werden von Region abgezogen.
- ▷ **`RegionDifference`** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Differenz.

Beispiel

```
/* provides the region X without the points in Y */
difference(X,Y,RegionDifference)
```

Komplexität

Sei N die Anzahl der Region und F_1 deren mittlere Fläche und F_2 die Gesamtfläche aller Regionen aus `Sub`, dann ist die Laufzeitkomplexität: $O(F\_1 * \log(F\_1) + N * (\sqrt{F\_1} + \sqrt{F\_2}))$.

Ergebnis

`difference` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`difference` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Siehe auch

`intersection`, `union1`, `union2`, `complement`

Modul

Region processing

intersection (Region1, Region2 : RegionIntersection : :)

Durchschnitt von Mengen von Regionen.

`intersection` bildet den Durchschnitt aller Regionen aus `Region1` mit den Regionen aus `Region2`. Jede Region aus `Region1` wird mit allen Regionen aus `Region2` geschnitten. Die Reihenfolge der Regionen aus `Region1` ist identisch mit der Reihenfolge der bearbeiteten Regionen in `RegionIntersection` (d.h. erste Eingaberegion = erste geschnittene Region, usw.).

Achtung

Leere Eingaberegionen sind zulässig. Da leere Regionen auftreten können ist das Flag **'store_empty_region'** zu beachten.

Parameter

- **Region1** (input_object) ..... region(-array) ⇝ *Hobject*
  Jede Region aus Region1 wird mit der Vereinigung der Regionen aus Region2 geschnitten.
- **Region2** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die für die Berechnung vereinigt werden.
- **RegionIntersection** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnis der Durchschnittsbildung.
  Parameteranzahl : `RegionIntersection` $\leq$ `Region1`

Komplexität

Sei N die Anzahl der Regionen in `Region1` und F_1 deren mittlere Fläche und F_2 die Gesamtfläche aller Regionen aus `Region2`, dann ist die Laufzeitkomplexität: $O(F_1 \log(F_1) + N * (\sqrt{F_1} + \sqrt{F_2}))$.

Ergebnis

`intersection` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`intersection` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Siehe auch

`union1`, `union2`, `complement`

Modul

Region processing

union1 (Region : RegionUnion : :)

Vereinigung von Regionen.

`union1` bestimmt die Vereinigung der Eingaberegionen und erzeugt daraus eine Ausgaberegion, die alle Punkte der Eingaberegionen umfaßt.

Parameter

- ▷ **Region** (input_object) . . . . . region-array ⇝ *Hobject*
  Regionen, die vereinigt werden sollen.
- ▷ **RegionUnion** (output_object) . . . . . region ⇝ *Hobject*
  Ergebnisregion, die die Vereinigung aller Eingaberegionen ist.
  Parameteranzahl : `RegionUnion` ≤ `Region`

Beispiel

```
/* Union of segmentation results: */
threshold(Image,Region1,128,255)
dyn_threshold(Image,Mean,Region2,5,'light')
concat_obj(Region1,Region2,Regions)
union1(Regions,RegionUnion).
```

Komplexität

Sei F die Summe aller Flächen der Eingaberegionen, dann ist die Laufzeitkomplexität: $O(\log(\sqrt{F}) * \sqrt{F})$.

Ergebnis

`union1` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`union1` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`union2`

Siehe auch

`intersection`, `complement`

Modul

Region processing

union2 (Region1, Region2 : RegionUnion : :)

Regionenvereinigung mit zwei Eingabeparametern.

`union2` vereinigt jede Region aus `Region1` mit der Vereinigung aller Regionen aus `Region2`. `union2` ist also nicht kommutativ!

Parameter

- ▷ **Region1** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Jede Region aus Region1 wird mit der Vereinigung aller Regionen aus Region2 vereinigt.
- ▷ **Region2** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Die Vereinigung der Regionen aus Region2 wird mit jeder Region aus Region1 vereinigt.
- ▷ **RegionUnion** (output_object) . . . . . region(-array) ⇝ *Hobject*
  Ergebnisregionen, die die Vereinigung mit den Eingaberegionen aus Region2 sind.
  Parameteranzahl : `RegionUnion` = `Region1`

Komplexität

Sei F die Summe aller Flächen der Eingaberegionen, dann ist die Laufzeitkomplexität: $O(\log(\sqrt{F}) * \sqrt{F})$.

Ergebnis

`union2` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer

Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`union2` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`union1`

Siehe auch

`intersection`, `complement`

Modul

Region processing

9.4 Merkmale

`area_center` (Regions : : : Area, Row, Column)

Fläche und Schwerpunkt von Regionen.

`area_center` berechnet die Fläche und den Schwerpunkt der Eingaberegionen. Die Fläche ist definiert als die Anzahl von Bildpunkten einer Region. Der Schwerpunkt berechnet sich als der Mittelwert der Zeilen- bzw. Spaltenkoordinaten aller Punkte.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **`Area`** (output_control) ..... integer(-array) ⇝ *integer*
  Fläche der Region.
- ▷ **`Row`** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **`Column`** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

main()
{
  Tuple   area, row, column;

  HImage   img ("affe");
  HWindow  w;

  img.Display (w);
  w.Click ();

  HRegionArray   reg = (img >= 164).Connection ();

  reg.Display (w);
```

```
  w.Click ();

  area = reg.AreaCenter (&row, &column);

  for (int i = 0; i < reg.Num (); i++)
  {
    cout << "Row    [" << i << "]" << "= " << row[i].D ();
    cout << "\t\tColumn [" << i << "]" << "= " << column[i].D () << endl;
  }

  cout << "Total number of regions: " << reg.Num () << endl;
  return(0);
}
```

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

`area_center` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`area_center` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Siehe auch

`select_shape`

Modul

Region processing

`circularity` (Regions : : : Circularity)

Formfaktor für die Kreisförmigkeit einer Region.

`circularity` berechnet die Ähnlichkeit der Eingaberegion mit einem Kreis.

Berechnung: Sei F die Fläche der Region und max der maximale Abstand vom Schwerpunkt zu allen Konturpunkten, dann ist der Formfaktor C definiert als:

$$C = \frac{F}{(max^2 * \pi)}$$

Der Formfaktor C ist bei einem Kreis gleich 1. Wenn die Region langgestreckt ist oder Hohlflächen hat, ist C kleiner als 1. `circularity` spricht besonders auf große Ausbuchtungen, Hohlflächen und nicht zusammenhängende Regionen an.

Bei einer leeren Region liefert `circularity` (soweit kein anderes Verhalten festgelegt wurde (siehe `set_system`)) den Wert 0. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **`Circularity`** (output_control) ..... real(-array) ⇝ *real*
  Rundheit der Eingaberegion(en).
  Zusicherung : $(0 \leq \texttt{Circularity}) \wedge (\texttt{Circularity} \leq 1.0)$

Beispiel

```
/* Comparison between shape factors of rectangle, circle and ellipse: */
gen_rectangle1(R1,10,10,20,20)
gen_rectangle2(R2,100,100,0.0,100,20)
gen_ellipse(E100,100,0.0,100,20)
gen_circle(,C,100,100,20)
circularity([R1,R2,E,C],M)
fwrite_string(FileId,['quadrate:  ',M[1]])
fnew_line(FileId)
fwrite_string(FileId,['rectangle: ',M[2]])
fnew_line(FileId)
fwrite_string(FileId,['ellipse:   ',M[3]])
fnew_line(FileId)
fwrite_string(FileId,['circle:    ',M[4]])
fnew_line(FileId)
```

Ergebnis

`circularity` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`circularity` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`roundness`, `compactness`, `convexity`, `eccentricity`

Siehe auch

`area_center`, `select_shape`

Modul

Region processing

`compactness` (Regions : : : Compactness)

Formfaktor für die Kompaktheit einer Region.

`compactness` berechnet die Kompaktheit der Eingaberegionen.

Berechnung: Sei L die Länge der Kontur (siehe `contlength`) und F die Fläche der Region, dann ist der Formfaktor C definiert als:

$$C = \frac{L^2}{4F\pi}$$

Der Formfaktor C ist bei einem Kreis gleich 1. Ist die Region langgestreckt oder hat sie Hohlflächen, dann ist C größer als 1. `compactness` spricht auf den Verlauf der Kontur (Rauhigkeit) und auf Hohlflächen an. Bei einer leeren Region liefert `compactness` den Wert 0, soweit kein anderes Verhalten festgelegt wurde (siehe `set_system`). Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
Zu untersuchende Region(en).

▷ **Compactness** (output_control) ..... real(-array) ⇝ *real*
Kompaktheit der Eingaberegion(en).
Zusicherung : (Compactness ≥ 1.0) ∨ (Compactness = 0)

Ergebnis

compactness liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

compactness ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

compactness, convexity, eccentricity

Siehe auch

contlength, area_center, select_shape

Modul

Region processing

connect_and_holes (Regions : : : NumConnected, NumHoles)

Anzahl der Zusammenhangskomponenten und Hohlflächen.

connect_and_holes berechnet die Anzahl der Zusammenhangskomponenten und die Anzahl der Hohlflächen einer jeden Region aus Regions.

Wird mehr als eine Region übergeben, dann werden die Zahlenwerte der Ausgabesteuerparameter NumConnected und NumHoles jeweils in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
Zu untersuchende Region(en).

▷ **NumConnected** (output_control) ..... integer(-array) ⇝ *integer*
Anzahl der Zusammenhangskomponenten einer Region.

▷ **NumHoles** (output_control) ..... integer(-array) ⇝ *integer*
Anzahl der Hohlflächen einer Region.

Ergebnis

connect_and_holes liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt.

Parallelisierungsinformation

connect_and_holes ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

euler_number

Siehe auch

connection, fill_up, fill_up_shape, union1

Modul

Region processing

contlength (Regions : : : ContLength)

Konturlänge einer Region.

contlength berechnet für jede Region aus Regions die Gesamtlänge der Kontur (Summe über alle Zusammenhangskomponenten der Region). Dabei wird der Abstand von zwei benachbarten Konturpunkten parallel zu den Koordinatenachsen mit 1, der Abstand in der Diagonalen mit $\sqrt{2}$ bewertet. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte der Konturlänge in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position einer Region im Eingabetupel entspricht. Bei einer leeren Region liefert contlength den Wert 0.

Achtung

Die Kontur von Hohlflächen wird nicht mitgerechnet.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **ContLength** (output_control) ..... real(-array) ⇝ *real*
  Konturlänge der Eingaberegion(en).
  Zusicherung : ContLength $\geq$ 0

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: " << argv[0] << " <# of regions> " << endl;
    return (-1);
  }

  HWindow        w;
  HRegionArray   reg;

  int NumOfElements = atoi (argv[1]);

  cout << "Draw " << NumOfElements << " regions " << endl;

  for (int i=0; i < NumOfElements; i++)
  {
    reg[i] = w.DrawRegion ();
  }

  Tuple circ = reg.Circularity ();
  Tuple cont = reg.Contlength ();

  for (i = 0; i < NumOfElements; i++)
  {
    cout << "Circularity of " << i+1 << ". region = " << circ[i].D();
    cout << "\t\t Contour Length of" << i+1 <<
            ". region = " << cont[i].D() << endl;
  }

  w.Click ();
  return(0);
}
```

Ergebnis

contlength liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

contlength ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Mögliche Nachfolgerfunktionen

get_region_contour

Alternativen

compactness

Siehe auch

area_center, get_region_contour

Modul

Region processing

convexity (Regions : : : Convexity)

Formfaktor für die Konvexität einer Region.

convexity berechnet die Konvexität jeder Eingaberegion aus Regions.

Berechnung: Sei F_c die Fläche der konvexen Hülle und F_o die Originalfläche der Region, dann ist der Formfaktor C definiert als:

$$C = \frac{F_o}{F_c}$$

Der Formfaktor C ist gleich 1, wenn die Region konvex ist (z.B. Rechteck, Kreis etc.). Sind Einbuchtungen oder Hohlflächen vorhanden, dann ist C kleiner als 1.

Bei einer leeren Region liefert convexity (soweit kein anderes Verhalten festgelegt wurde (siehe set_system)) den Wert 0. Wird mehr als eine Region übergeben, dann werden die Zahlenwerte des Formfaktors in einem Tupel abgespeichert, wobei die Position eines Wertes in dem Tupel der Position der Region im Eingabetupel entspricht.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **Convexity** (output_control) ..... real(-array) ⇝ *real*
  Konvexität der Eingaberegion(en).
  Zusicherung : Convexity $\leq$ 1

Ergebnis

convexity liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

convexity ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Siehe auch

select_shape, area_center, shape_trans

Modul

Region processing

diameter_region (Regions : : : Row1, Column1, Row2, Column2, Diameter)

Maximaler Abstand zweier Randpunkte einer Region.

diameter_region berechnet den maximalen Abstand zweier Randpunkte einer Region. Es werden die Koordiaten der beiden Extrempunkte und der Abstand zwischen diesen Punkten zurückgegeben.

Achtung

Bei leerer Region kann das Ergebnis von Row1, Column1, Row2 und Column2 (alle sind 0) zu Konfusionen führen.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Row1** (output_control) ..... line.begin.y(-array) ⇝ *integer*
  Zeilenindex des ersten Extrempunktes.
- ▷ **Column1** (output_control) ..... line.begin.x(-array) ⇝ *integer*
  Spaltenindex des ersten Extrempunktes.
- ▷ **Row2** (output_control) ..... line.end.y(-array) ⇝ *integer*
  Zeilenindex des zweiten Extrempunktes.
- ▷ **Column2** (output_control) ..... line.end.x(-array) ⇝ *integer*
  Spaltenindex des zweiten Extrempunktes.
- ▷ **Diameter** (output_control) ..... number(-array) ⇝ *real*
  Abstand der beiden Extrempunkte.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

diameter_region liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

diameter_region ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, runlength_features

Mögliche Nachfolgerfunktionen

disp_line

Alternativen

smallest_rectangle2

Modul

Region processing

eccentricity (Regions : : : Anisometry, Bulkiness, StructureFactor)

Aus den Ellipsenparametern abgeleitete Formmerkmale.

eccentricity berechnet drei Formmerkmale, die aus den geometrischen Momenten hergeleitet sind.

Definition: Seien die Ellipsenradien Ra, Rb und die Fläche F der Region gegeben (siehe elliptic_axis), dann gilt:

$$\texttt{Anisometry} = \frac{Ra}{Rb}$$

$$\texttt{Bulkiness} = \frac{\pi \cdot Ra \cdot Rb}{F}$$

$$\texttt{StructureFactor} = \texttt{Anisometry} \cdot \texttt{Bulkiness} - 1$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **`Anisometry`** (output_control) ..... real(-array) ⇝ *real*
  Formmerkmal (bei einem Kreis = 1.0).
  Zusicherung : `Anisometry` $\geq$ `1.0`
- ▷ **`Bulkiness`** (output_control) ..... real(-array) ⇝ *real*
  Berechnetes Formmerkmal.
- ▷ **`StructureFactor`** (output_control) ..... real(-array) ⇝ *real*
  Berechnetes Formmerkmal.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

`eccentricity` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`eccentricity` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Siehe auch

`elliptic_axis`, `moments_region_2nd`, `select_shape`, `area_center`

Modul

Region processing

`elliptic_axis` (Regions : : : Ra, Rb, Phi)

Parameter der äquivalenten Ellipse.

`elliptic_axis` berechnet die Radien und die Orientierung der Ellipse, die die „gleiche Orientierung“ und das “gleiche Seitenverhältnis“ wie die Eingaberegion haben. Es können auch mehrere Eingaberegionen als Tupel in `Regions` übergeben werden. Es wird die Länge des Hauptradius `Ra` und des Nebenradius `Rb` sowie die Orientierung der Hauptachse bezüglich der Horizontalen (`Phi`) bestimmt. Der Winkel wird dabei im Bogenmaß angegeben.

Berechnung:

Seien die Momente M_{20}, M_{02} und M_{11} normiert auf die Fläche gegeben (siehe `moments_region_2nd`), dann berechnen sich die Radien `Ra` und `Rb` zu:

$$\texttt{Ra} = \frac{\sqrt{8(M_{20} + M_{02} + \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

$$\texttt{Rb} = \frac{\sqrt{8(M_{20} + M_{02} - \sqrt{(M_{20} - M_{02})^2 + 4M_{11}^2})}}{2}$$

Die Orientierung `Phi` ist definiert durch:

$$\texttt{Phi} = -0.5 atan2(2M_{11}, M_{02} - M_{20})$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system('no_object_result',<Result>)`).

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **`Ra`** (output_control) ..... real(-array) ⇝ *real*
  Hauptradius (normiert auf die Fläche).
  Zusicherung : `Ra` $\geq$ `0.0`
- ▷ **`Rb`** (output_control) ..... real(-array) ⇝ *real*
  Nebenradius (normiert auf die Fläche).
  Zusicherung : (`Rb` $\geq$ `0.0`) $\wedge$ (`Rb` $\leq$ `Ra`)
- ▷ **`Phi`** (output_control) ..... real(-array) ⇝ *real*
  Winkel zwischen Hauptradius und x-Achse (Bogenmaß).
  Zusicherung : ((−`pi`/`2`) $<$ `Phi`) $\wedge$ (`Phi` $\leq$ (`pi`/`2`))

Beispiel

```
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)
```

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

`elliptic_axis` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`elliptic_axis` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Mögliche Nachfolgerfunktionen

`gen_ellipse`

Alternativen

`smallest_rectangle2`, `orientation_region`

Siehe auch

`moments_region_2nd`, `select_shape`, `set_shape`

Literatur

R. Haralick, L. Shapiro “Computer and Robot Vision” Addison-Wesley, 1992, pp. 73-75

Modul

Region processing

`euler_number ( Regions : : : EulerNumber )`

Berechnung der Eulerzahl.

Die Prozedur euler_number berechnet die Eulerzahl, d.h. die Differenz von der Anzahl der Zusammenhangskomponenten und der Anzahl der Hohlflächen.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **EulerNumber** (output_control) ..... integer(-array) ⇝ *integer*
  Berechnete Eulerzahl.

Ergebnis

euler_number liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

euler_number ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

connect_and_holes

Modul

Region processing

find_neighbors (Regions1, Regions2 : : MaxDistance : RegionIndex1, RegionIndex2)

Suchen von direkten Nachbarn.

find_neighbors bestimmt benachbarte Regionen. Regions1 und Regions2 enthalten dabei die zu testenden Regionen. Regions1 kann dabei auf drei verschiedene Arten besetzt werden:

- Regions1 ist leer:
  In diesem Fall werden alle Regionen in Regions2 permutativ auf Nachbarschaft getestet.
- Regions1 besteht aus einer Region:
  Die Regionen von Regions1 werden mit allen Regionen in Regions2 verglichen.
- Regions1 besteht aus gleich vielen Regionen wie Regions2:
  Hier werden jeweils die Regionen an n-ter Position in Regions1 und Regions2 auf die Nachbarschaftsbeziehung untersucht.

find_neighbors verwendet die Maximums-Norm bei der Berechnung des Abstandes. Der maximale Abstand wird mit MaxDistance angegeben. Zueinander benachbarte Regionen stehen an n-ter Position in RegionIndex1 und RegionIndex2, d.h. Region mit Index RegionIndex1[n] aus Regions1 ist benachbart zur Region mit Index RegionIndex2[n] aus Regions2.

Achtung

Verdeckte Regionen werden nicht gefunden!

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
  Ausgangsregionen.
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
  Vergleichsregionen.

▷ **MaxDistance** (input_control) ..... integer ⇝ *integer*
Maximaler Abstand der Regionen.
Defaultwert : 1
Wertevorschläge : MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 10, 15, 20, 50}
Typischer Wertebereich : 1 ≤ MaxDistance ≤ 255
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **RegionIndex1** (output_control) ..... integer-array ⇝ *integer*
Indizes der gefundenen Regionen aus Regions1.

▷ **RegionIndex2** (output_control) ..... integer-array ⇝ *integer*
Indizes der gefundenen Regionen aus Regions2.

Ergebnis

find_neighbors liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

find_neighbors ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Siehe auch

spatial_relation, select_region_spatial, expand_region, interjacent, boundary

Modul

Region processing

get_region_index (Regions : : Row, Column : Index)

Index aller Regionen, die einen übergebenen Punkt enthalten.

get_region_index liefert den Index aller Regionen in Regions (Wertebereich 1 bis n), die den Testpunkt (Row,Column) enthalten, d.h.:

$$|\texttt{Regions}[n] \cap \{(\texttt{Row}, \texttt{Column})\}| \neq \emptyset$$

Die zurückgegebenen Indizes können z.B. in select_obj verwendet werden, um die Regionen, die den Testpunkt enthalten, zu selektieren.

Achtung

Es kann vorkommen, daß mehr als eine Region den Punkt enthält, falls sich die Regionen überlappen. In diesem Fall werden all diese Regionen ausgegeben. Falls keine Region den angegebenen Punkt enthält, wird das leere Tupel (=keine Region) ausgegeben.

Parameter

▷ **Regions** (input_object) ..... region-array ⇝ *Hobject*
Zu untersuchende Regionen.

▷ **Row** (input_control) ..... point.y ⇝ *integer*
Zeilenindex des Testpunktes.
Defaultwert : 100
Typischer Wertebereich : $-\infty \leq$ Row $\leq \infty$ (lin)

▷ **Column** (input_control) ..... point.x ⇝ *integer*
Spaltenindex des Testpunktes.
Defaultwert : 100
Typischer Wertebereich : $-\infty \leq$ Column $\leq \infty$ (lin)

▷ **Index** (output_control) ..... integer(-array) ⇝ *integer*
Index der Regionen die den Testpunkt enthalten.

Komplexität

Sei F die Fläche einer Region und N die Anzahl der Regionen, dann beträgt die Laufzeitkomplexität im Mittel $O(ln(\sqrt{F}) * N)$.

Ergebnis

`get_region_index` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_region_index` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`select_region_point`

Siehe auch

`get_mbutton`, `get_mposition`, `test_region_point`

Modul

Region processing

`get_region_thickness` (Region : : : Thickness, Histogramm)

Zugriff auf die Dicke einer Region entlang der Hauptachse.

`get_region_thickness` berechnet die Dicke der Regionen entlang der Hauptachse (siehe `elliptic_axis`) für jeden Punkt der Strecke. Die Dicke an einer Stelle der Hauptachse ist dabei als der Abstand der am weitesten auseinanderliegenden Schnittpunkte der Kontur mit dem Lot auf die Hauptachse im betreffenden Punkt definiert. Zusätzlich liefert `get_region_thickness` noch das `Histogramm` der Dicken der Region. Die Länge des Histogramms entspricht dabei der größten aufgetretenen Dicke in der betrachteten Region.

Achtung

Es darf nur eine Region übergeben werden. Besitzt die Region mehrere Zusammenhangskomponenten, wird nur die erste ausgewertet. Alle anderen werden ignoriert

Parameter

- ▷ **`Region`** (input_object) ..... region ⇝ *Hobject*
  Zu analysierende Region.
- ▷ **`Thickness`** (output_control) ..... integer-array ⇝ *integer*
  Dicke der Region entlang ihrer Hauptachse.
- ▷ **`Histogramm`** (output_control) ..... integer-array ⇝ *integer*
  Histogramm der Dicke der Region entlang ihrer Hauptachse.

Ergebnis

`get_region_thickness` liefert den Wert 2 (H_MSG_TRUE), falls genau eine Region übergeben wird. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen.

Parallelisierungsinformation

`get_region_thickness` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `threshold`, `connection`, `select_shape`, `select_obj`

Siehe auch

`copy_obj`, `elliptic_axis`

Modul

Region processing

hamming_distance (Regions1, Regions2 : : : Distance, Similarity)

Hamming-Abstand zweier Regionen.

hamming_distance liefert den Hamming-Abstand zweier Regionen, d.h. die Zahl der Punkte der Regionen, die sich unterscheiden (Distance), also die Zahl der Punkte, die in der einen Region enthalten sind, in der anderen jedoch nicht:

$$\texttt{Distance} = |\texttt{Regions1} \cap \overline{\texttt{Regions2}}| + |\texttt{Regions2} \cap \overline{\texttt{Regions1}}|$$

Der Parameter Similarity beschreibt die Ähnlichkeit der beiden Regionen, basierend auf dem Hamming-Abstand Distance:

$$\texttt{Similarity} = 1 - \frac{\texttt{Distance}}{|\texttt{Regions1}| + |\texttt{Regions2}|}$$

Sind beide Regionen leer, dann wird Similarity auf 0 gesetzt. Es werden immer die Regionen mit dem gleichen Index aus den beiden Eingabeparametern miteinander verglichen.

Achtung

In beiden Eingabeparametern muß die gleiche Anzahl von Regionen übergeben werden.

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
  Vergleichsregionen.
- ▷ **Distance** (output_control) ..... integer(-array) ⇝ *integer*
  Hamming-Abstand zweier Regionen.
  Zusicherung : Distance $\geq$ 0
- ▷ **Similarity** (output_control) ..... real(-array) ⇝ *real*
  Ähnlichkeit zweier Regionen.
  Zusicherung : (0 $\leq$ Similarity) $\wedge$ (Similarity $\leq$ 1)

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

hamming_distance liefert den Wert 2 (H_MSG_TRUE), falls die Anzahl der Objekte in beiden Parametern gleich ist und nicht 0 beträgt. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hamming_distance ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

intersection, complement, area_center

Siehe auch

hamming_change_region

Modul

Region processing

hamming_distance_norm (Regions1, Regions2 : : Norm : Distance, Similarity)

Hamming-Abstand zweier Regionen nach Normierung.

hamming_distance_norm liefert den Hamming-Abstand zweier Regionen, d.h. die Zahl der Punkte der Regionen, die sich unterscheiden (Distance). Vor der Berechnung des Abstandes wird eine Normierung der Region in Regions1 auf die Region in Regions2 durchgeführt. Das Ergebnis ist die Zahl der Punkte, die in der einen Region enthalten sind, in der anderen jedoch nicht:

$$\texttt{Distance} = |Norm(\texttt{Regions1}) \cap \overline{\texttt{Regions2}}| + |\texttt{Regions2} \cap \overline{Norm(\texttt{Regions1})}|$$

Der Parameter Similarity beschreibt die Ähnlichkeit der beiden Regionen, basierend auf dem Hamming-Abstand Distance:

$$\texttt{Similarity} = 1 - \frac{\texttt{Distance}}{|Norm(\texttt{Regions1})| + |\texttt{Regions2}|}$$

Folgende Arten der Normierung stehen zur Verfügung:

'center': Die Region wird so verschoben, daß beide den gleichen Schwerpunkt haben.

Sind beide Regionen leer, dann wird Similarity auf 0 gesetzt. Es werden immer die Regionen mit dem gleichen Index aus den beiden Eingabeparametern miteinander verglichen.

Achtung

In beiden Eingabeparametern muß die gleiche Anzahl von Regionen übergeben werden.

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
  Vergleichsregionen
- ▷ **Norm** (input_control) ..... string(-array) ⇝ *string*
  Art der Normierung.
  Defaultwert : 'center'
  Werteliste : Norm ∈ {'center'}
- ▷ **Distance** (output_control) ..... integer(-array) ⇝ *integer*
  Hamming-Abstand zweier Regionen.
  Zusicherung : Distance ≥ 0
- ▷ **Similarity** (output_control) ..... real(-array) ⇝ *real*
  Ähnlichkeit zweier Regionen.
  Zusicherung : (0 ≤ Similarity) ∧ (Similarity ≤ 1)

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

hamming_distance_norm liefert den Wert 2 (H_MSG_TRUE), falls die Anzahl der Objekte in beiden Parametern gleich ist und nicht 0 beträgt. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels set_system('no_object_result',¡Result¿) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',¡Result¿) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hamming_distance_norm ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

intersection, complement, area_center

Siehe auch

hamming_change_region

Modul

Region processing

inner_circle (Regions : : : Row, Column, Radius)

Größter Inkreis einer Region.

inner_circle bestimmt den größten Inkreis einer Region, also den Kreis mit dem größten Flächeninhalt unter allen Kreisen, der in die Region paßt. Für diesen Kreis werden der Mittelpunkt (Row,Column) und der Radius (Radius) berechnet. Die Ausgabe der Prozedur ist so gewählt, daß sie als Eingabe für die HALCON-Prozeduren disp_circle und gen_circle verwendet werden kann.

Werden mehrere Regionen in Regions übergeben, so werden entsprechende Tupel als Ausgabeparameter zurückgegeben. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Achtung

Wenn mehrere Inkreise bei einer Region vorhanden sind, wird nur eine Lösung ausgegeben.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Row** (output_control) ..... circle.center.y(-array) ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **Column** (output_control) ..... circle.center.x(-array) ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **Radius** (output_control) ..... circle.radius(-array) ⇝ *real*
  Radius des Inkreises.
  Zusicherung : Radius $\geq$ 0

Beispiel

```
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)
```

Komplexität

Sei F die Fläche der Region und R der Radius des Inkreises, dann beträgt die Laufzeitkomplexität $O(\sqrt{F} * R)$.

Ergebnis

inner_circle liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

inner_circle ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, runlength_features

Mögliche Nachfolgerfunktionen

gen_circle, disp_circle

Alternativen

erosion_circle

Siehe auch

set_shape, select_shape, smallest_circle

Modul

Region processing

moments_region_2nd (Regions : : : M11, M20, M02, Ia, Ib)

Geometrische Momente von Regionen.

moments_region_2nd berechnet die Momente (M20, M02) und das Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen (M11). Weiterhin werden die Hauptträgheitsachsen (Ia, Ib) berechnet.

Berechnung: Es seien Z_0 und S_0 die Koordinaten des Schwerpunktes einer Region R mit Fläche F. Dann sind die Momente M_{ij} definiert durch:

$$M_{ij} = \sum_{(Z,S)\in R} (Z_0 - Z)^i (S_0 - S)^j$$

wobei Z und S alle Punkte der Region R durchlaufen.
Weiterhin sei

$$h = \frac{M20 + M02}{2}$$

dann sind Ia und Ib definiert durch:

$$\texttt{Ia} = h + \sqrt{h^2 - M20 * M02 + M11^2}$$

$$\texttt{Ib} = h - \sqrt{h^2 - M20 * M02 + M11^2}$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **M11** (output_control) ..... real(-array) ⇝ *real*
  Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.
- ▷ **M20** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (zeilenabhängig).
- ▷ **M02** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (spaltenabhängig).
- ▷ **Ia** (output_control) ..... real(-array) ⇝ *real*
  Die eine Hauptträgheitsachse.
- ▷ **Ib** (output_control) ..... real(-array) ⇝ *real*
  Die andere Hauptträgheitsachse.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

moments_region_2nd liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

moments_region_2nd ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

moments_region_2nd_invar

Siehe auch

elliptic_axis

Modul

Region processing

moments_region_2nd_invar (Regions : : : M11, M20, M02)

Geometrische Momente von Regionen.

moments_region_2nd_invar berechnet die normierten Momente (M20, M02) und das Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen (M11).

Berechnung: Es seien Z_0 und S_0 die Koordinaten des Schwerpunktes einer Region R mit Fläche F. Dann sind die Momente M_{ij} definiert durch:

$$M_{ij} = \frac{1}{F^2} \sum_{(Z,S) \in R} (Z_0 - Z)^i (S_0 - S)^j$$

wobei Z und S alle Punkte der Region R durchlaufen.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **M11** (output_control) ..... real(-array) ⇝ *real*
  Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.
- ▷ **M20** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (zeilenabhängig).
- ▷ **M02** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung (spaltenabhängig).

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

moments_region_2nd_invar liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

moments_region_2nd_invar ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

moments_region_2nd

Siehe auch

elliptic_axis

Modul

Region processing

moments_region_2nd_rel_invar (Regions : : : PHI1, PHI2)

Geometrische Momente von Regionen.

moments_region_2nd_rel_invar berechnet die relativen normierten Momente (PHI1, PHI2).

Berechnung: Die Momente $PHI1$ und $PHI2$ sind definiert durch:

$$PHI_1 = V_{20} + V_{02}$$

$$PHI_2 = (V_{20} + V_{02})^2 + V_{11}^2$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **PHI1** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **PHI2** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.

Ergebnis

moments_region_2nd_rel_invar liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

moments_region_2nd_rel_invar ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

moments_region_2nd

Siehe auch

elliptic_axis

Modul

Region processing

moments_region_3rd (Regions : : : M21, M12, M03, M30)

Geometrische Momente von Regionen.

moments_region_3rd berechnet die unveränderliche zentrale Momente (M21, M12, M03, M30) mit Ordnung $(p + q)$.

Berechnung: Es seien x und y die Koordinaten des Schwerpunktes einer Region R mit Fläche Z. Dann sind die Momente M_{pq} definiert durch:

$$M_{pq} = \sum_{i=1} MZ(x_i, y_i)(x_i - x)^p (y_i - y)^q$$

wobei $x = \frac{m_{10}}{m_{00}}$ und $y = \frac{m_{01}}{m_{00}}$ sind.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **M21** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).
- ▷ **M12** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M03** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M30** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{Z})$.

Ergebnis

moments_region_3rd liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

moments_region_3rd ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

moments_region_2nd

Siehe auch

elliptic_axis

Modul

Region processing

moments_region_3rd_invar (Regions : : : M21, M12, M03, M30)

Geometrische Momente von Regionen.

moments_region_3rd_invar berechnet die invariante Momente (M21, M12, M03, M30).

Berechnung: Die invariante Momente M_{pq} sind definiert durch:

$$M_{pq} = \frac{\mu_{pq}}{\mu^3}$$

wobei $p + q >= 2$ und $\mu = \mu_{00} = m_{00}$ sind.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **M21** (output_control) . . . . . real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).
- ▷ **M12** (output_control) . . . . . real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M03** (output_control) . . . . . real(-array) ⇝ *real*
  Moment 3. Ordnung (spaltenabhängig).
- ▷ **M30** (output_control) . . . . . real(-array) ⇝ *real*
  Moment 3. Ordnung (zeilenabhängig).

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{Z})$.

Ergebnis

`moments_region_3rd_invar` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

`moments_region_3rd_invar` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`moments_region_2nd`

Siehe auch

`elliptic_axis`

Modul

Region processing

moments_region_central (Regions : : : I1, I2, I3, I4)

Geometrische Momente von Regionen.

`moments_region_central` berechnet die zentrale Momente (`I1`, `I2`, `I3`, `I4`).

Berechnung: Die invariante Momente I_i sind definiert durch: $I_1 = \mu_{20}\mu_{02} - \mu_{11}^2$

$$I_2 = (\mu_{30}\mu_{03} - \mu_{21}\mu_{12})^2 - 4(\mu_{30}\mu_{12} - \mu_{21}^2)(\mu_{21}\mu_{03} - \mu_{12}^2)$$

$$I_3 = \mu_{20}(\mu_{21}\mu 03 - \mu_{12}^2) - \mu_{11}(\mu_{30}\mu_{03} - \mu_{21}\mu_{12}) + \mu_{02}(\mu_{30}\mu_{12} - \mu_{21}^2)$$

$$I_4 = \mu_{30}^2\mu_{02}^3 - 6\mu_{30}\mu_{21}\mu_{11}\mu_{02}^2 + 6\mu_{30}\mu_{12}\mu_{02}(2\mu_{11}^2 - \mu_{20}\mu_{02})$$
$$+ \mu_{30}\mu_{03}(6\mu_{20}\mu_{11}\mu_{02} - 8\mu_{11}^3) + 9\mu_{21}^2\mu_{20}\mu_{02}^2 - 18\mu_{21}\mu_{12}\mu_{20}\mu_{11}\mu_{02}$$
$$+ 6\mu_{21}\mu_{03}\mu_{20}(2\mu_{11}^2 - \mu_{20}\mu_{02}) + 9\mu_{12}^2\mu_{20}^2\mu_{02} - 6\mu_{12}\mu_{03}\mu_{11}\mu_{20}^2 + \mu_{03}^2\mu_{20}^3$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **I1** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **I2** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **I3** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **I4** (output_control) ..... real(-array) ⇝ *real*
  Moment 3. Ordnung.

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{Z})$.

Ergebnis

`moments_region_central` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

`moments_region_central` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`moments_region_2nd`

Siehe auch

`elliptic_axis`

Modul

Region processing

moments_region_central_invar (Regions : : : PSI1, PSI2, PSI3, PSI4)

Geometrische Momente von Regionen.

`moments_region_central_invar` berechnet die Momente (`PSI1`, `PSI2`, `PSI3`, `PSI4`), die bei Bewegung und lineare Transformationen unverändert bleiben.

Berechnung: Die Momente ψ_i sind definiert durch: $\psi_1 = \frac{I_1}{\mu^4}$

$\psi_2 = \frac{I_2}{\mu^1 0}$

$\psi_3 = \frac{I_3}{\mu^7}$

$\psi_4 = \frac{I_4}{\mu^1 1}$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **PSI1** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **PSI2** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **PSI3** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.
- ▷ **PSI4** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung.

Komplexität

Sei Z die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{Z})$.

Ergebnis

`moments_region_central_invar` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt..

Parallelisierungsinformation

`moments_region_central_invar` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`moments_region_2nd`

Siehe auch

`elliptic_axis`

Modul

Region processing

orientation_region (Regions : : Phi)

Orientierung einer Region.

`orientation_region` berechnet die Orientierung (`Phi`) der Region. Das Verfahren basiert auf dem Winkel der mit `elliptic_axis` bestimmt wird. Es wird jedoch zusätzlich der Punkt auf dem Rand mit maximalem Abstand bestimmt. Ist dessen Spaltenkoordinate kleiner als die Spaltenkoordinate des Schwerpunkts so wird π zu dem Winkel hinzugezählt.

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system('no_object_result',<Result>)`).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **Phi** (output_control) ..... real(-array) ⇝ *real*
  Orientierung der Region (Bogenmaß).
  Zusicherung : $((-\texttt{pi}/2) < \texttt{Phi}) \wedge (\texttt{Phi} \leq ((3 \cdot \texttt{pi})/2))$

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

`orientation_region` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das

Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`orientation_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Mögliche Nachfolgerfunktionen

`disp_arrow`

Alternativen

`elliptic_axis`, `smallest_rectangle2`

Siehe auch

`moments_region_2nd`, `line_orientation`

Modul

Region processing

roundness (Regions : : : Distance, Sigma, Roundness, Sides)

Formfaktoren aus der Kontur.

`roundness` untersucht den Abstand der Kontur vom Schwerpunkt der Fläche. Im einzelnen wird der mittlere Abstand (`Distance`), die Abweichung vom mittleren Abstand (`Sigma`) und zwei daraus abgeleitete Formmerkmale bestimmt. `Roundness` ist das Verhältnis von Mittelwert zu Standardabweichung und `Sides` gibt die Anzahl der Polygonstücke an, falls es sich um ein regelmäßiges Polygon handelt.

Die Kontur zur Berechnung der Merkmale wird in Abhängigkeit von der globalen Nachbarschaft bestimmt (siehe `set_system`).

Berechnung:

Sei p der Flächenschwerpunkt, p_i die Punkte und F die Fläche der Kontur.

$$\texttt{Distance} = \frac{1}{F} \sum ||p - p\_i||$$

$$\texttt{Sigma}^2 = \frac{1}{F} \sum (||p - p\_i|| - \texttt{Distance})^2$$

$$\texttt{Roundness} = 1 - \frac{\texttt{Sigma}}{\texttt{Distance}}$$

$$\texttt{Sides} = 1.4111 \left(\frac{\texttt{Distance}}{\texttt{Sigma}} \right)^{0.4724}$$

Wird mehr als eine Region übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index einer Region in der Eingabe entspricht.

Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **Distance** (output_control) ..... real(-array) ⇝ *real*
  Mittlerer Abstand vom Schwerpunkt.
  Zusicherung : `Distance` $\geq$ `0.0`
- ▷ **Sigma** (output_control) ..... real(-array) ⇝ *real*
  Standardabweichung von `Distance`.
  Zusicherung : `Sigma` $\geq$ `0.0`

▷ **Roundness** (output_control) ..... real(-array) ⇝ *real*
Formfaktor für Rundheit.
Zusicherung : `Roundness` $\leq$ `1.0`

▷ **Sides** (output_control) ..... real(-array) ⇝ *real*
Anzahl der Polygonseiten.
Zusicherung : `Sides` $\geq$ `0`

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(\sqrt{F})$.

Ergebnis

`roundness` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`roundness` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`compactness`

Siehe auch

`contlength`

Literatur

R. Haralick, L. Shapiro “Computer and Robot Vision” Addison-Wesley, 1992, pp. 61

Modul

Region processing

`runlength_distribution ( Region : : : Foreground, Background )`

Verteilung der Sehnenlängen einer Region.

`runlength_distribution` berechnet die Verteilung der Sehnenlängen einer Region von Vorder- und Hintergrund. Es wird gezählt wie häufig eine bestimmte Länge auftritt. Sehnen der Länge unendlich werden nicht gezählt. Der Hintergrund sind also alle Hohlflächen der Region. Es werden so viele Werte übergeben, wie die maximale Länge von Vorder- bzw. Hintergrund vorgibt. Die Länge der beiden Tupel ist i.A. unterschiedlich. Der erste Eintrag der Tupel ist immer 0 (keine Sehnen der Länge 0). Gibt es keine Zwischenräume wird bei `Background` das leere Tupel übergeben. Analog wird bei einer leeren Region bei `Foreground` das leere Tupel übergeben.

Parameter

▷ **Region** (input_object) ..... region ⇝ *Hobject*
Zu untersuchende Region.

▷ **Foreground** (output_control) ..... integer-array ⇝ *integer*
Längenverteilung der Region (Vordergrund).

▷ **Background** (output_control) ..... integer-array ⇝ *integer*
Längenverteilung des Hintergrundes.

Komplexität

Sei n die Anzahl der Sehnen der Region, dann beträgt die Laufzeitkomplexität $O(n)$.

Ergebnis

`runlength_distribution` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Wird mehr als eine Region übergeben, wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`runlength_distribution` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `select_obj`

Alternativen

`runlength_features`

Siehe auch

`runlength_features`

Modul

Region processing

`runlength_features` (`Regions` : : : `NumRuns`, `KFactor`, `LFactor`, `MeanLength`, `Bytes`)

Charakteristische Werte zur Lauflängenkodierung von Regionen.

`runlength_features` berechnet zu jeder Eingaberegion aus `Regions` die Anzahl der Sehnen, die bei der Speicherung dieser Region mit Hilfe der Lauflängenkodierung benötigt werden. Weiterhin wird der sogenannte „K-Faktor“ bestimmt, der angibt, um wieviel die Anzahl der Sehnen vom Ideal des Quadrates abweichen, bei dem dieser Wert gleich 1.0 ist.

Der K-Faktor (`KFactor`) berechnet sich nach der Formel:

$$\texttt{KFactor} = \frac{\texttt{NumRuns}}{\sqrt{Area}}$$

,

wobei $Area$ die Fläche der Region angibt. Es ist zu beachten, daß der K-Faktor kleiner als 1.0 werden kann (bei langgestreckten horizontalen Regionen).

Der L-Faktor (`LFactor`) gibt die mittlere Anzahl von Sehnen für jeden in der Region vorkommenden Zeilenindex aus.

`MeanLength` gibt die mittlere Länge der Sehnen an. Der Parameter `Bytes` gibt an, wieviele Bytes zur Kodierung der Region mit Lauflängen benötigt werden.

Achtung

Alle mit `runlength_features` berechneten Merkmale sind nicht rotationsinvariant, da die Lauflängenkodierung richtungsabhängig ist. `runlength_features` dient nicht zur Berechnung von Formmerkmalen, sondern zur Kontrolle und Analyse der Leistungsfähigkeit der Lauflängenkodierung.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **`NumRuns`** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl der Sehnen.
  Zusicherung : `0` ≤ `NumRuns`
- ▷ **`KFactor`** (output_control) ..... real(-array) ⇝ *real*
  Speicherfaktor gegenüber einem Quadrat.
  Zusicherung : `0` ≤ `KFactor`
- ▷ **`LFactor`** (output_control) ..... real(-array) ⇝ *real*
  Mittlere Anzahl von Sehnen pro Zeile.
  Zusicherung : `0` ≤ `LFactor`
- ▷ **`MeanLength`** (output_control) ..... real(-array) ⇝ *real*
  Mittlere Länge der Sehnen.
  Zusicherung : `0` ≤ `MeanLength`
- ▷ **`Bytes`** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl an Bytes, die zur Kodierung der Region benötigt werden.
  Zusicherung : `0` ≤ `Bytes`

Komplexität

Die Laufzeitkomplexität beträgt $O(1)$.

Ergebnis

`runlength_features` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`runlength_features` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Siehe auch

`runlength_features`, `runlength_distribution`

Modul

Region processing

`select_region_point ( Regions : DestRegions : Row, Column : )`

Auswahl aller Regionen, die einen übergebenen Punkt enthalten.

`select_region_point` selektiert alle Regionen aus `Regions`, die den Testpunkt (`Row`,`Column`) enthalten, d.h.:

$$|\texttt{Regions}[n] \cap \{(\texttt{Row}, \texttt{Column})\}| = 1$$

Achtung

Es kann vorkommen, daß mehr als eine Region den Punkt enthält, falls sich die Regionen überlappen. In diesem Fall werden all diese Regionen ausgegeben. Falls keine Region den angegebenen Punkt enthält, wird das leere Tupel (=keine Region) ausgegeben.

Parameter

- ▷ **`Regions`** (input_object) ..... region-array ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **`DestRegions`** (output_object) ..... region-array ⇝ *Hobject*
  Alle Regionen, die den Testpunkt enthalten.
- ▷ **`Row`** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Testpunktes.
  Defaultwert : 100
  Typischer Wertebereich : $-\infty \leq \texttt{Row} \leq \infty$ (lin)
- ▷ **`Column`** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Testpunktes.
  Defaultwert : 100
  Typischer Wertebereich : $-\infty \leq \texttt{Column} \leq \infty$ (lin)

Beispiel

```
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)')
  fnew_line(FileId)
  get_mbutton(WindowHandle,Row,Column,Button)
  select_region_point(Seg,Single,Row,Column)
  disp_region(Single,WindowHandle)
endwhile
```

Komplexität

Sei F die Fläche einer Region und N die Anzahl der Regionen, dann beträgt die Laufzeitkomplexität im Mittel $O(ln(\sqrt{F}) * N)$.

Ergebnis

`select_region_point` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_region_point` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`test_region_point`

Siehe auch

`get_mbutton`, `get_mposition`

Modul

Region processing

```
select_region_spatial ( Regions1,
Regions2 : :  Direction :  RegionIndex1, RegionIndex2 )
```

Lagebeziehung von Regionen.

`select_region_spatial` wählt die Regionen aus `Regions2` aus, die der Nachbarschaftsbeziehung `Direction` genügen. Die zu untersuchenden Regionen sind in `Regions1` bzw. `Regions2` zu übergeben. `Regions1` kann dabei auf drei verschiedene Arten besetzt werden:

- `Regions1` ist leer:
  In diesem Fall werden alle Regionen in `Regions2` permutativ auf Nachbarschaft getestet.
- `Regions1` besteht aus einer Region:
  Die Regionen von `Regions1` werden mit allen Regionen in `Regions2` verglichen.
- `Regions1` besteht aus gleich vielen Regionen wie `Regions2`:
  Hier werden jeweils die Regionen an n-ter Position in `Regions1` und `Regions2` auf die Nachbarschaftsbeziehung untersucht.

Mögliche Werte für `Direction` sind:

’left’: `Regions2` ist links von `Regions1`

’right’: `Regions2` ist rechts von `Regions1`

’above’: `Regions2` ist oberhalb von `Regions1`

’below’: `Regions2` ist unterhalb von `Regions1`

`select_region_spatial` berechnet die Schwerpunkte der zu vergleichenden Regionen und entscheidet anhand des Winkels zwischen der Schwerpunktgeraden und der x-Achse, ob die Richtungsbeziehung erfüllt ist. Die Relation ist jeweils in dem Bereich von -45 Grad bis +45 Grad um die Koordinatenachsen erfüllt. Die Richtungsrelation ist also so zu verstehen, daß der Schwerpunkt der zweiten Region links (bzw. rechts, oben, unten) vom Schwerpunkt der ersten Region liegen muß. Die Indizes der Regionen, die die Richtungsbeziehung erfüllen, stehen an n-ter Position in `RegionIndex1` und `RegionIndex2`, d.h. die Region mit Index `RegionIndex2`[n] steht mit Region mit Index `RegionIndex1`[n] in der angegebenen Relation. Der Zugriff auf Regionen über den Index kann mit `copy_obj` erfolgen.

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
  Ausgangsregionen
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
  Vergleichsregionen
- ▷ **Direction** (input_control) ..... string ⇝ *string*
  Gewünschte Nachbarschaftsbeziehung.
  Defaultwert : 'left'
  Werteliste : `Direction` ∈ {'left', 'right', 'above', 'below'}
- ▷ **RegionIndex1** (output_control) ..... integer-array ⇝ *integer*
  Indizes in den Eingabetupeln (`Regions1` bzw. `Regions2`).
- ▷ **RegionIndex2** (output_control) ..... integer-array ⇝ *integer*
  Indizes in den Eingabetupeln (`Regions1` bzw. `Regions2`).

Ergebnis

`select_region_spatial` liefert den Wert 2 (H_MSG_TRUE), falls `Regions2` nicht leer ist. Das Verhalten bei leerem Parameter `Regions2` (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_region_spatial` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`area_center`, `intersection`

Siehe auch

`spatial_relation`, `find_neighbors`, `copy_obj`, `obj_to_integer`

Modul

Region processing

select_shape (Regions : SelectedRegions : Features, Operation, Min, Max :)

Auswahl von Regionen mit Hilfe von Formmerkmalen.

`select_shape` wählt Regionen anhand ihrer Form aus. Für jede Eingaberegion aus `Regions` werden die angegebenen Merkmale (`Features`) berechnet. Wenn jedes (`Operation` = 'and') oder mindestens eines (`Operation` = 'or') der so berechneten Merkmale in den vorgegebenen Grenzen (`Min`,`Max`) liegt, wird die Region in die Ausgabe übernommen (dupliziert).
Bedingung: $Min_i \leq Feature_i(Object) \leq Max_i$

Mögliche Werte für `Features`:

'area': Fläche des Objektes

'row': Zeilenindex der Schwerpunkts

'column': Spaltenindex der Schwerpunkts

'width': Breite der Region

'height': Höhe der Region

'row1': Zeilenindex der linken oberen Ecke

'column1': Spaltenindex der linken oberen Ecke

'row2': Zeilenindex der rechten unteren Ecke

'column2': Spaltenindex der rechten unteren Ecke

'circularity': Kreisförmigkeit (vgl. `circularity`)

'compactness': Kompaktheit (vgl. `compactness`)

'contlength': Gesamtlänge der Kontur (vgl. `contlength`)

'convexity': Konvexität (vgl. `convexity`)

'ra': Hauptradius der äquivalenten Ellipse (vgl. `elliptic_axis`)

'rb': Nebenradius der äquivalenten Ellipse (vgl. `elliptic_axis`)

'phi': Orientierung der äquivalenten Ellipse (vgl. `elliptic_axis`)

'anisometry' Anisometrie (vgl. `eccentricity`)

'bulkiness' Bulkiness (vgl. Operator `eccentricity`)

'struct_factor' Struktur Faktor (vgl. Operator `eccentricity`)

'outer_radius': Radius des kleinsten umschließenden Kreises (vgl. `smallest_circle`)

'inner_radius': Radius des größten Inkreises (vgl. `inner_circle`)

'dist_mean': Mittlerer Abstand zwischen dem Rand der Region und ihrem Schwerpunkt (vgl. Operator `roundness`)

'dist_deviation': Standardabweichung des Abstands vom Regionenrand zum Schwerpunkt (vgl. Operator `roundness`)

'roundness': Rundheit (vgl. Prozedur `roundness`)

'num_sides': Anzahl Polygonseiten (vgl. Prozedur `roundness`)

'connect_num': Anzahl der Zusammenhangskomponenten (vgl. Operator `connect_and_holes`)

'holes_num': Anzahl der Löcher (vgl. Operator `connect_and_holes`)

'max_diameter': Maximale Ausdehnung der Region (vgl. Operator `diameter_region`)

'orientation': Orientierung der Region (vgl. Operator `orientation_region`)

'euler_number': Eulerzahl (vgl. Operator `euler_number`)

'rect2_phi': Orientierung des kleinsten umschließenden Rechtecks (vgl. Operator `smallest_rectangle2`)

'rect2_len1': Halbe Länge des kleinsten umschließenden Rechtecks (vgl. Operator `smallest_rectangle2`)

'rect2_len2': Halbe Breite des kleinsten umschließenden Rechtecks (vgl. Operator `smallest_rectangle2`)

'moments_m11': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd`)

'moments_m20': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd`)

'moments_m02': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd`)

'moments_ia': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd`)

'moments_ib': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd`)

'moments_m11_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd_invar`)

'moments_m20_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd_invar`)

'moments_m02_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd_invar`)

'moments_phi1': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd_rel_invar`)

'moments_phi2': Geometrische Regionenmomente (vgl. Operator `moments_region_2nd_rel_invar`)

'moments_m21': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)

'moments_m12': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)

'moments_m03': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)

'moments_m30': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd`)

'moments_m21_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)

'moments_m12_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)

'moments_m03_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)

'moments_m30_invar': Geometrische Regionenmomente (vgl. Operator `moments_region_3rd_invar`)

'moments_i1': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)

'moments_i2': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)

'moments_i3': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)

'moments_i4': Geometrische Regionenmomente (vgl. Operator `moments_region_central`)

'moments_psi1': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)

'moments_psi2': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)

'moments_psi3': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)

'moments_psi4': Geometrische Regionenmomente (vgl. Operator `moments_region_central_invar`)

Wird nur ein Merkmal (`Features`) verwendet, dann ist der Wert von `Operation` bedeutungslos. Mehrere Merkmale werden in der Reihenfolge abgearbeitet, in der sie eingegeben werden.

Parameter

- ▷ **`Regions`** (input_object) ..... region-array ⇝ *Hobject*
  Regionen, die untersucht werden sollen.
- ▷ **`SelectedRegions`** (output_object) ..... region-array ⇝ *Hobject*
  Regionen, die die Bedingung erfüllen.
- ▷ **`Features`** (input_control) ..... string(-array) ⇝ *string*
  Zu testende Formmerkmale.
  Defaultwert : 'area'
  Werteliste : `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_ia', 'moments_ib', 'moments_m11_invar', 'moments_m20_invar', 'moments_m02_invar', 'moments_phi1', 'moments_phi2', 'moments_m21', '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*
  Verknüpfungsart der einzelnen Merkmale.
  Defaultwert : 'and'
  Werteliste : `Operation` ∈ {'and', 'or'}
- ▷ **`Min`** (input_control) ..... real(-array) ⇝ *real* / integer / string
  Untere Grenzen der Merkmale oder 'min'.
  Defaultwert : 150.0
  Typischer Wertebereich : 0.0 ≤ `Min` ≤ 99999.0
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 1.0
- ▷ **`Max`** (input_control) ..... real(-array) ⇝ *real* / integer / string
  Obere Grenzen der Merkmale oder 'max'.
  Defaultwert : 99999.0
  Typischer Wertebereich : 0.0 ≤ `Max` ≤ 99999.0
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 1.0
  Restriktion : `Max` ≥ `Min`

Beispiel

```
/* 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)
```

Ergebnis

`select_shape` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingabeobjekte vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system ('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`select_shape` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Mögliche Nachfolgerfunktionen

`select_shape`, `select_gray`, `shape_trans`, `reduce_domain`, `count_obj`

Siehe auch

`area_center`, `circularity`, `compactness`, `contlength`, `convexity`, `elliptic_axis`, `eccentricity`, `inner_circle`, `smallest_circle`, `smallest_rectangle1`, `smallest_rectangle2`, `roundness`, `connect_and_holes`, `diameter_region`, `orientation_region`, `moments_region_2nd`, `moments_region_2nd_invar`, `moments_region_2nd_rel_invar`, `moments_region_3rd`, `moments_region_3rd_invar`, `moments_region_central`, `moments_region_central_invar`, `select_obj`

Modul

Region processing

`select_shape_proto` (Regions, Pattern : SelectedRegions : Feature, Min, Max :)

Auswahl von Regionen, die in einer bestimmten Beziehung zueinander stehen.

`select_shape_proto` setzt zwei Regionen zueinander in Beziehung. Es wird jeweils die i-te Region aus `Regions` mit der Vereinigung der Regionen aus `Pattern` verglichen. Die Grenzen (`Min` und `Max`) werden je nach Merkmal absolut oder in Prozent (0..100) angegeben. Mögliche Werte für `Feature` sind:

’distance_dilate’ Der minimale Abstand in der Maximumsnorm vom Rand von `Pattern` zum Rand jeder Region aus `Regions` wird bestimmt (vgl. `distance_rr_min_dil`).

’distance_contour’ Der minimale euklidsche Abstand vom Rand von `Pattern` zum Rand jeder Region aus `Regions` wird bestimmt. (vgl. `distance_rr_min`).

’distance_center’ Der euklidsche Abstand vom Schwerpunkt von `Pattern` zum Schwerpunkt jeder Region aus `Regions` wird bestimmt.

’covers’ Es wird geprüft wie gut die Region `Pattern` in die Regionen aus `Regions` paßt. Wenn es keine Verschiebung gibt, sodaß `Pattern` eine Teilmenge von `Regions` ist, so ist die Überdeckung 0. Wenn `Pattern` nach einer entsprechenden Verschiebung mit der Region übereinstimmt, ist die Überdeckung 100. Ansonsten wird die Fläche des Opening von `Regions` mit `Pattern` mit der Fläche von `Regions` ins Verhältnis gesetzt (in Prozent).

’fits’ Es wird getestet, ob `Pattern` so verschoben werden kann, daß es in `Regions` paßt. Wenn dies möglich ist, wird die zugehörige Region aus `Regions` kopiert. Die Parameter `Min` und `Max` haben hier keine Bedeutung.

’overlaps_abs’ Die Fläche der Schnittmenge von `Pattern` und jeder Region in `Regions` wird berechnet.

’overlaps_rel’ Die Fläche der Schnittmenge von `Pattern` und jeder Region in `Regions` wird berechnet. Die relative Überlappung ergibt sich aus dem Verhältnis der Fläche der Schnittmenge und der Fläche der jeweiligen Region aus `Regions` (in Prozent).

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **`Pattern`** (input_object) ..... region(-array) ⇝ *Hobject*
  Region, die mit `Regions` verglichen wird.
- ▷ **`SelectedRegions`** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die die Bedingung erfüllen.
- ▷ **`Feature`** (input_control) ..... string(-array) ⇝ *string*
  Zu testende Formmerkmale.
  Defaultwert : ’covers’
  Werteliste : `Feature` ∈ {’distance_center’, ’distance_dilate’, ’distance_contour’, ’covers’, ’fits’, ’overlaps_abs’, ’overlaps_rel’}

▷ **Min** (input_control) . . . . . number ⇝ *real* / integer
Untere Grenze des Merkmals.
Defaultwert : 50.0
Wertevorschläge : Min ∈ {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}
Typischer Wertebereich : 0.0 ≤ Min
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 5.0

▷ **Max** (input_control) . . . . . number ⇝ *real* / integer
Obere Grenze des Merkmals.
Defaultwert : 100.0
Wertevorschläge : Max ∈ {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}
Typischer Wertebereich : 0.0 ≤ Max
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 5.0

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: " << argv[0] << " <radius of circle>" << endl;
    exit (1);
  }

  double   rad = atof (argv[1]);
  HImage   img ("affe");
  HWindow  w;

  img.Display (w);

  HRegion       circ = HRegion::GenCircle (100, 100, rad);
  HRegionArray  reg  = img.Regiongrowing (3, 3, 5, 0);
  HRegionArray  seg  = reg.SelectShapeProto (circ, "fits", 0, 0);

  w.SetColor  ("red");
  seg.Display (w);
  w.Click ();
  return(0);
}
```

Ergebnis

select_shape_proto liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

select_shape_proto ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

connection, draw_region, gen_circle, gen_rectangle1, gen_rectangle2, gen_ellipse

Mögliche Nachfolgerfunktionen

select_gray, shape_trans, reduce_domain, count_obj

Alternativen

select_shape

Siehe auch

opening, erosion1, distance_rr_min_dil, distance_rr_min

Modul

Region processing

select_shape_std (Regions : SelectedRegions : Shape, Percent :)

Auswahl von Regionen einer vorgegebenen Form.

select_shape_std vergleicht die Form der übergebenen Regionen mit den vorgegebenen Formen. Wenn die Region eine ähnliche Gestalt hat, wird sie in die Ausgabe übernommen. Mögliche Werte für Shape sind:

'max_area' Es wird die größte Region ausgewählt.

'rectangle1' Es wird mit smallest_rectangle1 das umschließende Rechteck parallel zu den Koordinatenachsen bestimmt. Wenn die Flächendifferenz in Prozent größer als Percent ist, wird die Region übernommen.

'rectangle2' Es wird mit smallest_rectangle2 das kleinste umschließende Rechteck mit beliebiger Orientierung bestimmt. Wenn die Flächendifferenz in Prozent größer als Percent ist, wird die Region übernommen.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Eingaberegionen, die selektiert werden soll.
- ▷ **SelectedRegions** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen mit gewünschter Form.
- ▷ **Shape** (input_control) ..... string ⇝ *string*
  Zu testende Formmerkmale.
  Defaultwert : 'max_area'
  Werteliste : Shape ∈ {'max_area', 'rectangle1', 'rectangle2'}
- ▷ **Percent** (input_control) ..... real ⇝ *real*
  Ähnlichkeitsmaß.
  Defaultwert : 70.0
  Wertevorschläge : Percent ∈ {10.0, 30.0, 50.0, 60.0, 70.0, 80.0, 90.0, 95.0, 100.0}
  Typischer Wertebereich : 0.0 ≤ Percent ≤ 100.0 (lin)
  Minimale Schrittweite : 0.1
  Empfohlene Schrittweite : 10.0

Parallelisierungsinformation

select_shape_std ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, smallest_rectangle1, smallest_rectangle2

Alternativen

intersection, complement, area_center

Siehe auch

smallest_rectangle1, smallest_rectangle2

Modul

Region processing

smallest_circle (Regions : : : Row, Column, Radius)

Kleinster umschliessender Kreis einer Region.

smallest_circle bestimmt den kleinsten umschließenden Kreis einer Region, also den Kreis mit dem kleinsten Flächeninhalt unter allen Kreisen, die die Region enthalten. Für diesen Kreis werden der Mittelpunkt (Row,Column) und der Radius (Radius) berechnet. Die Prozedur findet Anwendung, wenn z.B. die Lage und Größe von kreisförmigen Objekten (z.B. Münzen) bestimmt werden soll, die aber aufgrund schlechter Segmentation im Inneren nicht homogen sind oder unterbrochene Ränder besitzen. Die Ausgabe der Prozedur ist so gewählt, daß sie als Eingabe für die HALCON-Prozeduren disp_circle und gen_circle verwendet werden kann.

Werden mehrere Regionen in Regions übergeben, so werden entsprechende Tupel als Ausgabeparameter zurückgegeben. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Row** (output_control) ..... circle.center.y(-array) ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **Column** (output_control) ..... circle.center.x(-array) ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **Radius** (output_control) ..... circle.radius(-array) ⇝ *real*
  Radius des umschliessenden Kreises.
  Zusicherung : Radius $\geq$ 0

Beispiel

```
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)
smallest_circle(H,Row,Column,Radius)
gen_circle(Circles,Row,Column,Radius)
set_draw(WindowHandle,'margin')
disp_region(Circles,WindowHandle)
```

Komplexität

Sei F die Fläche der Region und N die Anzahl der Stützpunkte der konvexen Hülle, dann beträgt die Laufzeitkomplexität $O(\sqrt{F} + N^3)$.

Ergebnis

smallest_circle liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

smallest_circle ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, runlength_features

Mögliche Nachfolgerfunktionen

gen_circle, disp_circle

Alternativen

elliptic_axis, smallest_rectangle1, smallest_rectangle2

Siehe auch

set_shape, select_shape, inner_circle

Modul

Region processing

smallest_rectangle1 (Regions : : : Row1, Column1, Row2, Column2)

Umschließendes Rechteck parallel zu den Koordinatenachsen.

smallest_rectangle1 berechnet das umschließende Rechteck aller Eingaberegionen (parallel zu den Koordinatenachsen). Das umschließende Rechteck wird durch die Koordinaten der Eckpunkte (Row1,Column1,Row2,Column2) beschrieben.

Wird mehr als eine Region in Regions übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes im Tupel dem Index einer Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0, soweit kein anderes Verhalten eingestellt wurde (siehe set_system).

Achtung

Bei leerer Region kann das Ergebnis von Row1,Column1, Row2 und Column2 (alle sind 0) zu Konfusionen führen.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Row1** (output_control) ..... rectangle.origin.y(-array) ⇝ *integer*
  Zeilenindex des linken oberen Eckpunkts.
- ▷ **Column1** (output_control) ..... rectangle.origin.x(-array) ⇝ *integer*
  Spaltenindex des linken oberen Eckpunkts.
- ▷ **Row2** (output_control) ..... rectangle.corner.y(-array) ⇝ *integer*
  Zeilenindex des rechten unteren Eckpunkts.
- ▷ **Column2** (output_control) ..... rectangle.corner.x(-array) ⇝ *integer*
  Spaltenindex des rechten unteren Eckpunkts.

Komplexität

Sei F die Fläche einer Region, dann beträgt die Laufzeitkomplexität im Mittel $O(sqrt(F))$.

Ergebnis

smallest_rectangle1 liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

smallest_rectangle1 ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection, runlength_features

Mögliche Nachfolgerfunktionen

disp_rectangle1, gen_rectangle1

Alternativen

smallest_rectangle2, area_center

Siehe auch

select_shape

Modul

Region processing

smallest_rectangle2 (Regions : : : Row, Column, Phi, Length1, Length2)

Kleinstes umschließendes Rechteck mit beliebiger Orientierung.

smallest_rectangle2 bestimmt das kleinste umschließende Rechteck einer Region, also das Rechteck mit dem kleinsten Flächeninhalt unter allen Rechtecken, die die Region enthalten. Für dieses Rechteck werden der Mittelpunkt, der Neigungswinkel und die beiden Halbmesser berechnet.

Die Prozedur findet Anwendung, wenn z.B. die Lage einer Szenerie von mehreren Regionen (z.B. gedruckter Text auf einem rechteckigen Papier bzw. mit rechteckigem Druckbild (Blocksatz)) gefunden werden soll. Die Parameter von `smallest_rectangle2` sind so gewählt, daß sie direkt als Eingabe für die HALCON-Prozeduren `disp_rectangle2` und `gen_rectangle2` verwendet werden können.

Wird mehr als eine Region in `Regions` übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes im Tupel dem Index einer Region in der Eingabe entspricht. Bei leerer Region haben alle Parameter den Wert 0.0, soweit kein anderes Verhalten eingestellt wurde (siehe `set_system`).

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **`Row`** (output_control) ..... rectangle2.center.y(-array) ⇝ *real*
  Zeilenindex des Mittelpunktes.
- ▷ **`Column`** (output_control) ..... rectangle2.center.x(-array) ⇝ *real*
  Spaltenindex des Mittelpunktes.
- ▷ **`Phi`** (output_control) ..... rectangle2.angle.rad(-array) ⇝ *real*
  Orientierung des umschließenden Rechtecks (Bogenmaß)
  Zusicherung : $((-\texttt{pi}/2) < \texttt{Phi}) \wedge (\texttt{Phi} \leq (\texttt{pi}/2))$
- ▷ **`Length1`** (output_control) ..... rectangle2.hwidth(-array) ⇝ *real*
  Erster Halbmesser (halbe Länge) des umschließenden Rechtecks.
  Zusicherung : $\texttt{Length1} \geq \texttt{0.0}$
- ▷ **`Length2`** (output_control) ..... rectangle2.hheight(-array) ⇝ *real*
  Zweiter Halbmesser (halbe Breite) des umschließenden Rechtecks.
  Zusicherung : $(\texttt{Length2} \geq \texttt{0.0}) \wedge (\texttt{Length2} \leq \texttt{Length1})$

Beispiel

```
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)
```

Komplexität

Sei F die Fläche der Region und N die Anzahl der Stützpunkte der konvexen Hülle, dann beträgt die Laufzeitkomplexität $O(\sqrt{F} + N^2)$.

Ergebnis

`smallest_rectangle2` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit `set_system('empty_region_result',<Result>)` bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`smallest_rectangle2` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`, `runlength_features`

Mögliche Nachfolgerfunktionen

`disp_rectangle2`, `gen_rectangle2`

Alternativen

`elliptic_axis`, `smallest_rectangle1`

Siehe auch

`smallest_circle`, `set_shape`

Modul

Region processing

spatial_relation (Regions1, Regions2 : : Percent : RegionIndex1, RegionIndex2, Relation1, Relation2)

Lagebeziehung von Regionen bzgl. der Koordinatenachsen.

spatial_relation wählt Regionen aus, die um Percent Prozent „links“, „rechts“, „oberhalb“ oder „unterhalb“ von anderen Regionen liegen. Regions1 und Regions2 enthalten die zu vergleichenden Regionen. Regions1 kann dabei auf drei verschiedene Arten besetzt werden:

- Regions1 ist leer:
  In diesem Fall werden alle Regionen in Regions2 permutativ auf Nachbarschaft getestet.
- Regions1 besteht aus einer Region:
  Die Regionen von Regions1 werden mit allen Regionen in Regions2 verglichen.
- Regions1 besteht aus gleich vielen Regionen wie Regions2:
  Hier werden jeweils die Regionen an n-ter Position in Regions1 und Regions2 auf die Nachbarschaftsbeziehung untersucht.

Der Prozentsatz Percent wird so interpretiert, daß die Fläche der zweiten Region um mindestens Percent Prozent echt links/rechts bzw. oberhalb/unterhalb der Regionengrenzen der ersten Region liegen muß. Die Indizes der Regionen, die mindestens eine dieser Bedingungen erfüllen, stehen dann an n-ter Position in den Ausgabeparametern RegionIndex1 und RegionIndex2. Zusätzlich enthalten die Ausgabeparameter Relation1 und Relation2 an n-ter Position die Art der Relation, in der das Regionenpaar (RegionIndex1[n], RegionIndex2[n]) steht, d.h. Region mit Index RegionIndex2[n] steht mit Region mit Index RegionIndex1[n] in der Relation1[n] und Relation2[n].

Mögliche Werte für Relation1 und Relation2 sind:

Relation1: ’left’, ’right’ oder ”

Relation2: ’above’, ’below’ oder ”

In RegionIndex1 und RegionIndex2 werden die Indizes der Regionen in den Tupeln der Eingaberegionen (Regions1 bzw. Regions2) als Bildidentifikatoren eingetragen. Der Zugriff auf ausgewählte Regionen über den Index kann mit copy_obj erfolgen.

Parameter

- **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
  Ausgangsregionen.
- **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
  Vergleichsregionen.
- **Percent** (input_control) ..... integer ⇝ *integer*
  Prozentzahl der Fläche der Vergleichsregion, die links/rechts bzw. oberhalb/unterhalb der Regionengrenzen der Ausgangsregion liegen muß.
  Defaultwert : 50
  Wertevorschläge : Percent ∈ {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
  Typischer Wertebereich : 0 ≤ Percent ≤ 100 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : (0 ≤ Percent) ∧ (Percent ≤ 100)
- **RegionIndex1** (output_control) ..... integer-array ⇝ *integer*
  Indizes der Regionen im Tupel der Eingaberegionen, die die Lagerelation erfüllen.
- **RegionIndex2** (output_control) ..... integer-array ⇝ *integer*
  Indizes der Regionen im Tupel der Eingaberegionen, die die Lagerelation erfüllen.
- **Relation1** (output_control) ..... string-array ⇝ *string*
  Horizontale Lagerelation, in der RegionIndex2[n] mit RegionIndex1[n] steht.
- **Relation2** (output_control) ..... string-array ⇝ *string*
  Vertikale Lagerelation, in der RegionIndex2[n] mit RegionIndex1[n] steht.

Ergebnis

spatial_relation liefert den Wert 2 (H_MSG_TRUE), falls Regions2 nicht leer und Percent korrekt besetzt ist. Das Verhalten bei leerem Parameter Regions2 (keine Eingaberegionen vorhanden) läßt sich mittels

set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

spatial_relation ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

area_center, intersection

Siehe auch

select_region_spatial, find_neighbors, copy_obj, obj_to_integer

Modul

Region processing

test_region_point (Regions : : Row, Column :)

Test, ob eine Region einen gegebenen Punkt enthält.

test_region_point prüft, ob mindestens eine Eingaberegion aus Regions den Testpunkt (Row,Column) enthält.

Achtung

Bei leerer Eingabe (= keine Region) wird im Fall set_system('no_object_result','true') FALSE als Ergebnis ausgegeben (keine Region enthält den Punkt).

Der Testpunkt ist in einer leeren Region nicht enthalten (kein Punkt der Region stimmt mit dem Punkt überein). Sind alle Regionen leer, wird ebenfalls FALSE ausgegeben. Ein leere Region verhält sich also genauso, als wenn sie nicht vorhanden wäre.

Parameter

- ▷ **Regions** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Region(en).
- ▷ **Row** (input_control) ..... point.y ⇝ *integer*
  Zeilenindex des Testpunktes.
  Defaultwert : 100
  Typischer Wertebereich : $0 \leq$ Row ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Column** (input_control) ..... point.x ⇝ *integer*
  Spaltenindex des Testpunktes.
  Defaultwert : 100
  Typischer Wertebereich : $0 \leq$ Column ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Komplexität

Sei F die Fläche einer Region und N die Anzahl der Regionen, dann beträgt die mittlere Laufzeitkomplexität $O(\ln(\sqrt{F}) * N)$.

Ergebnis

test_region_point liefert den Wert 2 (H_MSG_TRUE), falls eine Region den Testpunkt enthält. Ist dies nicht der Fall, gibt test_region_point 3 (H_MSG_FALSE) aus. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

test_region_point ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

union1, intersection, area_center

Siehe auch

select_region_point

Rückgabewert

bool

Modul

Region processing

9.5 Transformation

background_seg (Foreground : BackgroundRegions : :)

Zusammenhängende Regionen des Hintergrundes.

background_seg liefert zusammenhängende Hintergrundregionen. Die Prozedur wird typischerweise nach der Kantensuche (inkl. Verdünnung) angewandt, um die von den Kanten begrenzten Bild-Segmente zu ermitteln. Die Berechnung der Zusammenhangskomponenten erfolgt mit der 4-er Nachbarschaft.

Parameter

- ▷ **Foreground** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionkanten.
- ▷ **BackgroundRegions** (output_object) ..... region-array ⇝ *Hobject*
  Die zusammenhängenden Hintergrundkomponenten in Foreground.

Beispiel

```
/* 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).
```

Komplexität

Sei F die Fläche des Hintergrundes, H die Bildhöhe und N die Anzahl der Ergebnisregionen, dann ist die Laufzeitkomplexität: $O(H + \sqrt{F} * \sqrt{N})$.

Ergebnis

background_seg liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

background_seg ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, connection, regiongrowing, pouring, class_ndim_norm

Mögliche Nachfolgerfunktionen

select_shape

Alternativen

`complement`, `connection`

Siehe auch

`threshold`, `hysteresis_threshold`, `skeleton`, `expand_region`, `set_system`, `sobel_amp`, `edges_image`, `roberts`, `bandpass_image`

Modul

Region processing

`clip_region` (`Region` : `RegionClipped` : `Row1, Column1, Row2, Column2` :)

Reduktion der Region auf ein Rechteck.

`clip_region` bildet den Durchschnitt aller Regionen in der Eingabe mit dem Rechteck das durch die vier Steuerparameter festgelegt wird. Dabei ist `clip_region` effizienter als `intersection` mit einem Rechteck das als Region (`gen_rectangle1`) dargestellt ist.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu transformierende Regionen.
- ▷ **`RegionClipped`** (output_object) ..... region(-array) ⇝ *Hobject*
  Beschnittene Regionen.
- ▷ **`Row1`** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeilen-Koordinate linkes oberes Eck des Rechtecks.
  Defaultwert : 0
  Wertevorschläge : `Row1` ∈ {0, 128, 200, 256}
  Typischer Wertebereich : $-\infty \leq$ `Row1` $\leq \infty$ (lin)
- ▷ **`Column1`** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spalten-Koordinate linkes oberes Eck.
  Defaultwert : 0
  Wertevorschläge : `Column1` ∈ {0, 128, 200, 256}
  Typischer Wertebereich : $-\infty \leq$ `Column1` $\leq \infty$ (lin)
- ▷ **`Row2`** (input_control) ..... rectangle.corner.y ⇝ *integer*
  Zeilen-Koordinate rechtes unteres Eck des Rechtecks.
  Defaultwert : 256
  Wertevorschläge : `Row2` ∈ {128, 200, 256, 512}
  Typischer Wertebereich : 0 ≤ `Row2` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **`Column2`** (input_control) ..... rectangle.corner.x ⇝ *integer*
  Spalten-Koordinate rechtes unteres Eck.
  Defaultwert : 256
  Wertevorschläge : `Column2` ∈ {128, 200, 256, 512}
  Typischer Wertebereich : 0 ≤ `Column2` ≤ 511 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

Ergebnis

`clip_region` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system ('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`clip_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`intersection`, `gen_rectangle1`, `clip_region_rel`

Modul

Region processing

`clip_region_rel` (`Region` : `RegionClipped` : `Top, Bottom, Left, Right` :)

Clipping der Region relativ zu seiner Größe.

`clip_region_rel` bildet den Durchschnitt der Region mit einem Rechteck das innerhalb der Region liegt. Die Größe des Rechtecks wird durch das umschließende Rechteck bestimmt, welches um die Werte der vier Steuerparameter verkleinert wird. Alle vier Parameter enthalten positive Zahlen oder Null und geben an, um wieviel das Rechteck oben (`Top`), unten (`Bottom`), links (`Left`) oder rechts (`Right`) verkleinert wird. Sind alle Parameter gleich Null, dann bleibt die Region unverändert.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu transformierende Regionen.
- ▷ **`RegionClipped`** (output_object) ..... region(-array) ⇝ *Hobject*
  Beschnittene Regionen.
- ▷ **`Top`** (input_control) ..... integer ⇝ *integer*
  Anzahl Zeilen die „oben“ abgeschnitten werden.
  Defaultwert : 1
  Wertevorschläge : `Top` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich : 0 ≤ `Top` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Bottom`** (input_control) ..... integer ⇝ *integer*
  Anzahl Zeilen die „unten“ abgeschnitten werden.
  Defaultwert : 1
  Wertevorschläge : `Bottom` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich : 0 ≤ `Bottom` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Left`** (input_control) ..... integer ⇝ *integer*
  Anzahl Spalten die „links“ abgeschnitten werden.
  Defaultwert : 1
  Wertevorschläge : `Left` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich : 0 ≤ `Left` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Right`** (input_control) ..... integer ⇝ *integer*
  Anzahl Zeilen die „rechts“ abgeschnitten werden.
  Defaultwert : 1
  Wertevorschläge : `Right` ∈ {0, 1, 2, 3, 4, 5, 7, 10, 20, 30, 50}
  Typischer Wertebereich : 0 ≤ `Right` (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Ergebnis

`clip_region_rel` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system ('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`clip_region_rel` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`smallest_rectangle1`, `intersection`, `gen_rectangle1`, `clip_region`

Modul

Region processing

`connection ( Region : ConnectedRegions : : )`

Berechnung von zusammenhängenden Regionen.

`connection` bestimmt die Zusammenhangskomponenten der Regionen in `Region`, zerlegt also die Regionen in zusammenhängende Bereiche. Die Nachbarschaft wird mit `set_system('neighborhood',<4/8>)` eingestellt. Voreingestellt ist die 8-ter Nachbarschaft, die für Vordergrund sinnvoll eingesetzt werden kann. Die maximale Anzahl von Zusammenhangskomponenten, die `connection` zurückliefert, kann mittels `set_system('max_connection',<Num>)` festgelegt werden. Die Voreinstellung von **0** bewirkt, daß alle Zusammenhangskomponenten zurückliefert werden. Die Umkehrung von `connection` für eine zusammenhängende Region ist `union1`.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu zerlegende Region.
- ▷ **`ConnectedRegions`** (output_object) ..... region-array ⇝ *Hobject*
  Zusammenhängende Regionen.

Beispiel

```
read_image(Image,'affe')
set_colored(WindowHandle,12)
threshold(Image,Light,150.0,255.0)
count_obj(Light,Number1)
fwrite_string('Nummber of regions after threshold = '+Number1)
fnew_line()
disp_region(Light,WindowHandle)
connection(Light,Many)
count_obj(Many,Number2)
fwrite_string('Nummber of regions after threshold = '+Number2)
fnew_line()
disp_region(Many,WindowHandle).
```

Komplexität

Sei F die Fläche der Eingaberegion und N die Anzahl der daraus erzeugten Zusammenhangskomponenten, dann ist die Laufzeitkomplexität: $O(\sqrt{F} * \sqrt{N})$.

Ergebnis

`connection` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`connection` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`auto_threshold`, `threshold`, `dyn_threshold`, `erosion1`

Mögliche Nachfolgerfunktionen

`select_shape`, `select_gray`, `shape_trans`, `set_colored`, `dilation1`, `count_obj`, `reduce_domain`, `add_channels`

Alternativen

`background_seg`

Siehe auch

`set_system`, `union1`

Modul

Region processing

`distance_transform` (`Region` : `DistanceImage` : `Metric`, `Foreground`, `Width`, `Height` :)

Distanztransformation einer Region.

`distance_transform` berechnet für jeden Punkt der Eingaberegion `Region` (bzw. deren Komplement) den Abstand zum Rand der Region. Die Auswahl der Pixel, für die der Abstand berechnet werden soll, kann mit dem Parameter `Foreground` gesteuert werden. Falls `Foreground` = **'true'**, werden die Abstände nur für die Pixel innerhalb der Region berechnet. Falls `Foreground` = **'false'**, werden die Abstände der Pixel außerhalb der Regionen zu deren Rand berechnet. Die Abstände werden für alle Punkte des Ausgabebildes `DistanceImage` berechnet. Die Größe des Ausgabebildes wird durch `Width` und `Height` bestimmt. Die Eingaberegion wird auf die Größe des Ausgabebildes beschnitten. Falls es wichtig ist, daß die Abstände innerhalb der gesamten Region berechnet werden, sollte die Region so verschoben werden (siehe `move_region`), daß sie nur positive Koordinaten besitzt und die Breite und Höhe des Ausgabebildes sollten groß genug gewählt werden, daß die Region vollständig darin enthalten ist. Die entsprechende Information kann mit `smallest_rectangle1` bestimmt werden.

Welche Metrik zur Berechnung des Abstandes verwendet wird, wird mit Hilfe des Parameters `Metric` festgelegt. Falls `Metric` = **'city-block'** wird der Abstand aus dem kürzesten Pfad eines Punktes zum Rand der Region berechnet, wobei nur horizontale und vertikale „Bewegungen“ erlaubt sind. Diese werden mit einer Distanz von 1 gewichtet. Falls `Metric` = **'chessboard'** wird der Abstand aus dem kürzesten Pfad eines Punktes zum Rand der Region berechnet, wobei horizontale, vertikale und diagonale „Bewegungen“ erlaubt sind. Diese werden mit einer Distanz von 1 gewichtet. Falls `Metric` = **'octagonal'** wird eine Kombination der beiden Verfahren verwendet, so daß diagonale Pfade ein etwas größeres Gewicht erhalten. Falls `Metric` = **'chamfer-3-4'** werden horizontale und vertikale Bewegungen mit einem Gewicht von 3 und diagonale Bewegungen mit einem Gewicht von 4 bewertet. Zur Normalisierung wird das resultierende Distanzbild durch 3 geteilt. Da diese Normierung Zeit kostet und man normalerweise nur an den relativen Abständen mehrerer Punkte interessiert ist, kann die Normierung mit Falls `Metric` = **'chamfer-3-4-unnormalized'** unterdrückt werden. Schließlich wird für `Metric` = **'euclidean'** eine annähernd euklidische Distanz berechnet.

Parameter

- ▷ **`Region`** (input_object) ..... region(-array) ⇝ *Hobject*
  Region, zu deren Rand die Distanz berechnet werden soll.
- ▷ **`DistanceImage`** (output_object) ..... image ⇝ *Hobject* : int4
  Rückgabebild, das die Distanz-Information enthält.
- ▷ **`Metric`** (input_control) ..... string ⇝ *string*
  Metrik, die bei der Distanztransformation verwendet werden soll.
  Defaultwert : "city-block"
  Werteliste : `Metric` ∈ {"city-block", "chessboard", "octagonal", "chamfer-3-4", "chamfer-3-4-unnormalized", "euclidean"}
- ▷ **`Foreground`** (input_control) ..... string ⇝ *string*
  Berechnung der Distanzen innerhalb (**true**) oder ausserhalb (**false**) der Eingabregion.
  Defaultwert : 'true'
  Werteliste : `Foreground` ∈ {'true', 'false'}

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des Ausgabebildes.
Defaultwert : 640
Wertevorschläge : Width ∈ {160, 192, 320, 384, 640, 768}
Typischer Wertebereich : 1 ≤ Width

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des Ausgabebildes.
Defaultwert : 480
Wertevorschläge : Height ∈ {120, 144, 240, 288, 480, 576}
Typischer Wertebereich : 1 ≤ Height

Beispiel

```
/* 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)
```

Komplexität

Die Laufzeit-Komplexität ist O(Width ∗ Height).

Ergebnis

distance_transform liefert H_MSG_2 (H_MSG_TRUE) zurück, wenn alle Parameter korrekt sind.

Parallelisierungsinformation

distance_transform ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, dyn_threshold, regiongrowing

Mögliche Nachfolgerfunktionen

threshold

Siehe auch

skeleton

Literatur

P. Soille: “Morphological Image Analysis, Principles and Applications”; Springer Verlag Berlin Heidelberg New York, 1999.

G. Borgefors: “Distance Transformations in Arbitrary Dimensions”; Computer Vision, Graphics, and Image Processing, Vol. 27, pages 321–345, 1984.

P.E. Danielsson: “Euclidean Distance Mapping”; Computer Graphics and Image Processing, Vol. 14, pages 227–248, 1980.

Modul

Region processing

eliminate_runs (Region : RegionClipped : ElimShorter, ElimLonger :)

Eliminieren von Lauflängen vorgegebener Länge.

eliminate_runs Löscht alle Sehnen der Lauflängenkodierung der Eingaberegionen, die kürzen als ElimShorter oder länger als ElimLonger sind.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
Zu transformierende Regionen.

▷ **RegionClipped** (output_object) ..... region(-array) ⇝ *Hobject*
Beschnittene Regionen.

▷ **ElimShorter** (input_control) ..... integer ⇝ *integer*
Alle Sehnen die kürzer sind werden unterdrückt.
Defaultwert : 3
Wertevorschläge : ElimShorter ∈ {2, 3, 4, 5, 6, 8, 10, 12, 15}
Typischer Wertebereich : 1 ≤ ElimShorter ≤ 500 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **ElimLonger** (input_control) ..... integer ⇝ *integer*
Alle Sehnen die länger sind werden unterdrückt.
Defaultwert : 1000
Wertevorschläge : ElimLonger ∈ {50, 100, 200, 500, 1000, 2000}
Typischer Wertebereich : 1 ≤ ElimLonger ≤ 10000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

Ergebnis

eliminate_runs liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system ('empty_region_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

eliminate_runs ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, connection, regiongrowing, pouring

Mögliche Nachfolgerfunktionen

erosion1, dilation1, disp_region

Alternativen

shape_trans

Modul

Region processing

expand_region (Regions, ForbiddenArea : RegionExpanded : Iterations, Mode :)

Füllt „Lücken" zwischen Regionen auf oder trennt überlappende Regionen.

expand_region dient dazu, Lücken zwischen den Regionen, wie sie z.B. durch Unterdrückung zu kleiner Regionen nach einer Bildsegmentation entstehen, zu schließen (Modus 'image') oder überlappende Eingaberegionen zu trennen (Modus 'region'). Beide Effekte beruhen auf der Expansion von Regionen. Dabei wird bei jeder Iteration der Expansion ein 1-Pixel breiter Streifen um die Region zu dieser hinzugefügt bzw. aus ihr entfernt.

Expandiert wird dabei nur in Bildbereiche, die nicht als „verbotene Bereiche" (Parameter ForbiddenArea) ausgewiesen sind. Die Zahl der Iterationen wird mit dem Parameter Iterations festgelegt. Die Übergabe des strings **'maximal'** veranlaßt expand_region, das Verfahren so lange zu iterieren, bis es konvergiert, also keine Änderungen mehr auftreten. Eine 0 auf dieser Parameterposition bewirkt die Ausgabe der nicht-überlappenden Teilregionen. Im Detail unterscheiden sich die beiden Modi **'image'** und **'region'** wie folgt:

'image' Die Eingaberegionen werden iterativ so lange ausgedehnt, bis sie eine andere Region oder einen Bildrand berühren. Da expand_region alle Regionen simultan bearbeitet, werden die „Lücken" zwischen den Regionen „gerecht" auf diese verteilt. Überlappende Regionen werden getrennt, indem ihre gemeinsamen Teilregionen (wiederum „gerecht") auf sie verteilt werden.

'region' Es wird keine Expansion der Eingaberegionen durchgeführt, sondern nur überlappende Regionen getrennt, indem die gemeinsamen Teilregionen „gerecht" auf die Regionen aufgeteilt werden. Da nach dem dafür benötigten „Negativwachstum" der Schnitt mit den Originalregionen gebildet wird, kann es in diesem Modus zu „Lücken" in den Ausgaberegionen kommen, die Segmentation der Bildebene ist also nicht total. Diesem Problem kann mit einem zweiten Aufruf von expand_region mit dem Komplement der ursprünglichen Eingaberegionen als „forbidden area" begegnet werden.

Parameter

- ▷ **Regions** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Regionen, zwischen denen Lücken geschlossen oder die getrennt werden sollen.
- ▷ **ForbiddenArea** (input_object) . . . . . region ⇝ *Hobject*
  In diesen Bereich darf nicht expandiert werden.
- ▷ **RegionExpanded** (output_object) . . . . . region(-array) ⇝ *Hobject*
  Expandierte oder getrennte Bildregionen.
- ▷ **Iterations** (input_control) . . . . . integer ⇝ *integer* / string
  Zahl der Iterationen.
  Defaultwert : 'maximal'
  Wertevorschläge : `Iterations` ∈ {'maximal', 0, 1, 2, 3, 5, 7, 10, 15, 20, 30, 50, 70, 100, 200}
  Typischer Wertebereich : 0 ≤ `Iterations` ≤ 1000 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Gewünschter Modus.
  Defaultwert : 'image'
  Werteliste : `Mode` ∈ {'image', 'region'}

Beispiel

```
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)
```

Ergebnis

`expand_region` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`expand_region` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`pouring`, `threshold`, `dyn_threshold`, `regiongrowing`

Alternativen

`dilation1`

Siehe auch

`expand_gray`, `interjacent`, `skeleton`

Modul

Region processing

fill_up (Region : RegionFillUp : :)

Auffüllen von Hohlflächen.

`fill_up` füllt Hohlflächen in Regionen aus, falls solche vorhanden sind. Die Anzahl der Regionen wird dabei nicht verändert. Die Art der Nachbarschaft wird mit `set_system('neighborhood',<4/8>)` festgelegt (Defaulteinstellung ist 8-ter Nachbarschaft).

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Eingaberegion(en) mit evtl. vorhandenen Hohlflächen.
- ▷ **RegionFillUp** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen, deren Hohlflächen aufgefüllt sind.

Ergebnis

`fill_up` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fill_up` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`fill_up_shape`

Siehe auch

`boundary`

Modul

Region processing

fill_up_shape (Region : RegionFillUp : Feature, Min, Max :)

Auffüllen von Hohlflächen mit vorgegebenen Formeigenschaften.

`fill_up_shape` füllt lediglich diejenigen Hohlflächen in `Region` auf, die vorgegebenen Formeigenschaften entsprechen. Dabei gibt der Parameter `Feature` den Namen des Formmerkmals vor. Mit `Min` und `Max` wird das Intervall festgelegt, in dem der Merkmalswert der Hohlfläche liegen muß, damit sie ausgefüllt wird.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Eingaberegion(en).
- ▷ **RegionFillUp** (output_object) ..... region(-array) ⇝ *Hobject*
  Ausgaberegion(en) mit weniger Hohlflächen.
- ▷ **Feature** (input_control) ..... string ⇝ *string*
  Formmerkmal.
  Defaultwert : ’area’
  Werteliste : `Feature` ∈ {’area’, ’compactness’, ’convexity’, ’anisometry’, ’phi’, ’ra’, ’rb’, ’inner_circle’, ’outer_circle’}
- ▷ **Min** (input_control) ..... number ⇝ *real* / integer
  Mindestwert für Merkmal Feature.
  Defaultwert : 1.0
  Wertevorschläge : `Min` ∈ {0.0, 1.0, 10.0, 50.0, 100.0, 500.0, 1000.0, 10000.0}
  Typischer Wertebereich : 0.0 ≤ `Min`
- ▷ **Max** (input_control) ..... number ⇝ *real* / integer
  Maximalwert für Merkmal Feature.
  Defaultwert : 100.0
  Wertevorschläge : `Max` ∈ {10.0, 50.0, 100.0, 500.0, 1000.0, 10000.0, 100000.0}
  Typischer Wertebereich : 0.0 ≤ `Max`

Beispiel

```
read_image(&Image,"affe");
threshold(Image,&Seg,120.0,255.0);
fill_up_shape(Seg,&Filled,"area",0.0,200.0);
```

Ergebnis

`fill_up_shape` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fill_up_shape` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`fill_up`

Siehe auch

`select_shape`, `connection`, `area_center`

Modul

Region processing

`hamming_change_region` (InputRegion : OutputRegion : Width, Height, Distance :)

Erzeugen einer Region mit vorgegebenem Hamming-Abstand.

`hamming_change_region` verändert die Regionen in dem linken oberen Bildausschnitt (der Größe `Width` x `Height`) so, daß die entstehenden Regionen zu den Eingabedaten den Hamming-Abstand `Distance` haben. Dazu werden `Distance` Punkte zur jeweiligen Eingaberegion hinzugefügt bzw. aus ihr entfernt.

Achtung

Wenn `Width` und `Height` zu groß gewählt werden, benötigt die erzeugte Region viel Speicherplatz.

Parameter

- ▷ **`InputRegion`** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu modifizierenden Regionen.
- ▷ **`OutputRegion`** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die den angegebenen Hamming-Abstand haben.
- ▷ **`Width`** (input_control) ..... extent.x ⇝ *integer*
  Horizontale Ausdehnung des zu ändernden Bereichs.
  Defaultwert : 100
  Wertevorschläge : `Width` ∈ {64, 128, 256, 512}
  Typischer Wertebereich : 1 ≤ `Width` ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : `Width` > `0`
- ▷ **`Height`** (input_control) ..... extent.y ⇝ *integer*
  Vertikale Ausdehnung des zu ändernden Bereichs.
  Defaultwert : 100
  Wertevorschläge : `Height` ∈ {64, 128, 256, 512}
  Typischer Wertebereich : 1 ≤ `Height` ≤ 512 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : `Height` > `0`

▷ **Distance** (input_control) ..... integer ⇝ *integer*
Hamming-Abstand zwischen den alten und den neuen Regionen.
Defaultwert : 1000
Wertevorschläge : Distance ∈ {100, 500, 1000, 5000, 10000}
Typischer Wertebereich : 0 ≤ Distance ≤ 10000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : (Distance ≥ 0) ∧ (Distance < (Width · Height))

Komplexität

Speicherplatzkomplexität der erzeugten Region in Byte (worst case): $O(2 * \texttt{Width} * \texttt{Height})$.

Ergebnis

hamming_change_region liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hamming_change_region ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

connection, regiongrowing, pouring, class_ndim_norm

Mögliche Nachfolgerfunktionen

select_shape

Siehe auch

hamming_distance

Modul

Region processing

interjacent (Region : RegionInterjacent : Mode :)

Unterteilen der Bildebene mittels vorgegebener Regionen.

interjacent unterteilt die Bildebene mittels der Regionen in Region. Als Ergebnis wird eine Ausgaberegion mit den ermittelten Trennlinien zurückgeliefert. Es stehen dafür drei verschiedene Modi zur Verfügung:

'medial_axis' Dieser Modus wird für Eingaberegionen verwendet, die sich nicht berühren. Die Prozedur findet dann Trennlinien zwischen den Regionen, die den Hintergrund „gerecht“ zwischen den Eingaberegionen aufteilt. Ihre Anwendung entspricht der Ausführung folgender Befehlssequenz:

```
complement('full',Region,Tmp) skeleton(Tmp,Result)
```

'border' Für einander nicht berührende Eingaberegionen entspricht die Prozedur in diesem Modus boundary (Region,Result), ersetzt also jeder Eingaberegion durch ihre Kontur. Einander berührende Eingaberegionen werden zu einer Region zusammengefaßt. Die zugehörige Ausgaberegion umfaßt dann deren Kontur und die (ein Pixel breite) Trennlinie zwischen den ursprünglichen Regionen. interjacent führt in diesem Fall also folgende Einzelschritte durch:

```
boundary(Region,Tmp1,'inner') union1(Tmp1,Tmp2)
  skeleton(Tmp2,Result)
```

'mixed' In diesem Modus verhält sich interjacent für einander nicht berührende Eingaberegionen wie im Modus 'medial_axis'. Kommt es zu einer Berührung, werden zwar wiederum Trennlinien zwischen den Eingaberegionen als Ausgaberegionen erzeugt, nun aber inklusive der „Berührungslinie“ der Regionen. Einander berührende Regionen werden also (durch eine Linie in der Ausgaberegion) „getrennt“. Dies entspricht der Anweisungssequenz:

```
erosion1(Region,Mask,Tmp1,1) union1(Tmp1,Tmp2)
  complement(full,Tmp2,Tmp3) skeleton(Tmp3,Result)
```

wobei Mask folgende „Kreuzmaske“ bezeichnet:

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Eingaberegionen, zwischen deren Regionen Trennlinien gefunden werden sollen.
- ▷ **RegionInterjacent** (output_object) ..... region ⇝ *Hobject*
  Eine Ausgaberegion mit den ermittelten Trennlinien als Region.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Gewünschter Modus.
  Defaultwert : ’mixed’
  Werteliste : `Mode` ∈ {’medial_axis’, ’border’, ’mixed’}

Beispiel

```
read_image(Image,'wald1_rot')
mean(Image,Mean,31,31)
dyn_threshold(Mean,Seg,20)
interjacent(Seg,Graph,'medial_axis')
disp_region(Graph,WindowHandle)
```

Ergebnis

`interjacent` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`interjacent` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Siehe auch

`expand_region`, `junctions_skeleton`, `boundary`

Modul

Region processing

junctions_skeleton (Region : EndPoints, JuncPoints : :)

Kreuzungs- und Endpunkte in einem Skelett.

`junctions_skeleton` liefert Kreuzungs- und Endpunkte in einem Skelett (siehe `skeleton`). Die Kreuzungspunkte in einer der Eingaberegionen aus `Region` werden als Region in `JuncPoints`, die Endpunkte entsprechend als Region in `EndPoints` gespeichert.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Die Skelette.
- ▷ **EndPoints** (output_object) ..... region(-array) ⇝ *Hobject*
  Endpunkte.
  Parameteranzahl : `EndPoints` = `Region`
- ▷ **JuncPoints** (output_object) ..... region(-array) ⇝ *Hobject*
  Kreuzungspunkte.
  Parameteranzahl : `JuncPoints` = `Region`

Beispiel

```
/* non-connected branches of a skeleton */
skeleton(Region,Skeleton)
junctions_skeleton(Skeleton,EPoints,JPoints)
difference(S,JPoints,Rows)
set_system('heighbourhood',4)
connection(Rows,Parts).
```

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität: $O(F)$.

Ergebnis

`junctions_skeleton` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`junctions_skeleton` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`skeleton`

Mögliche Nachfolgerfunktionen

`area_center`, `connection`, `get_region_points`, `difference`

Siehe auch

`pruning`, `split_skeleton_region`

Modul

Region processing

`merge_regions_line_scan` (`CurrRegions`, `PrevRegions` : `CurrMergedRegions`, `PrevMergedRegions` : `ImageHeight`, `MergeBorder`, `MaxImagesRegion` :)

Zusammenfügen von Regionen aus Zeilenkamerabildern.

Der Operator `merge_regions_line_scan` verbindet Regionen, die aus räumlich aneinandergrenzenden Bildern mit der Höhe `ImageHeight` segmentiert wurden und die, wenn das alte Bild verschoben wird, an den Bildrändern zusammenstoßen. Die Bilder können beispielsweise von einer Zeilenkamera nacheinander aufgenommen worden sein. Dabei wird davon ausgegangen, dass `CurrRegions` Regionen des aktuellen Bildes enthält und `PrevRegions` Regionen des vorherigen Bildes, welches die räumliche Fortsetzung des aktuellen Bildes darstellt.

Mit Hilfe des Parameters `MergeBorder` kann angegeben werden, ob die Oberkante des aktuellen Bildes an die Unterkante des vorherigen Bildes stößt (**’top’**) oder die Unterkannte des aktuellen Bildes an die Oberkante des vorherigen Bildes (**’bottom’**).

Der Parameter `MaxImagesRegion` bestimmt bei der rekursiven Anwendung des Operators `merge_regions_line_scan`, wieviel Bilder eine aktuelle Ausgangsregion maximal zurückreichen kann. Der Regionenteil, der über diese Bilder hinausreicht, wird abgeschnitten.

Der Operator `merge_regions_line_scan` liefert zwei Regionenarrays zurück. `PrevMergedRegions` enthält alle Regionen, die ausschließlich im alten Bild liegen und nicht mit einer aktuellen Region verbunden werden konnten. In `CurrMergedRegions` werden dagegen alle aktuellen Regionen, gegebenenfalls vergrößert um die angrenzenden Regionen aus `PrevRegions`, eingetragen. Die so verbundenen Regionen erscheinen als eine einzige neue Region in `CurrMergedRegions`, wobei die angefügten Teile um die Bildhöhe nach oben (`MergeBorder`=**’top’**) oder nach unten (`MergeBorder`=**’bottom’**) verschoben werden und in der Ergebnisregion in `CurrMergedRegions` über den Bildrand hinausgehen. Dafür wird durch `merge_regions_line_scan` der Systemparameter **’clip_region’** (vgl. `set_system`) auf **’false’** gesetzt.

Parameter

- ▷ **CurrRegions** (input_object) ..... region(-array) ⇝ *Hobject*
  Aktuelle Eingaberegionen.
- ▷ **PrevRegions** (input_object) ..... region(-array) ⇝ *Hobject*
  Im vorhergehenden Zyklus zusammengefügte Regionen.
- ▷ **CurrMergedRegions** (output_object) ..... region(-array) ⇝ *Hobject*
  Aktuelle Regionen, die ggf. mit den alten Regionen verbunden wurden.
- ▷ **PrevMergedRegions** (output_object) ..... region(-array) ⇝ *Hobject*
  Alte Regionen, die nicht mit den aktuellen verbunden werden konnten.
- ▷ **ImageHeight** (input_control) ..... integer ⇝ *integer*
  Höhe der Ausgangsbilder.
  Defaultwert : 512
  Werteliste : `ImageHeight` ∈ {240, 480, 512}
- ▷ **MergeBorder** (input_control) ..... string ⇝ *string*
  Im aktuellen Bild die Zeile, die mit dem vorhergehenden Bild zusammenstößt.
  Defaultwert : ”top”
  Werteliste : `MergeBorder` ∈ {”top”, ”bottom”}
- ▷ **MaxImagesRegion** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl der Bilder, über die sich eine Region erstrecken darf.
  Defaultwert : 3
  Wertevorschläge : `MaxImagesRegion` ∈ {1, 2, 3, 4, 5}

Ergebnis

`merge_regions_line_scan` liefert den Wert 2 (H_MSG_TRUE), falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`merge_regions_line_scan` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Region processing

partition_dynamic (Region : Partitioned : Distance, Percent :)

Horizontales Aufteilen einer Region in Rechtecke.

`partition_dynamic` zerlegt die Eingaberegion in Rechtecke mit der mittleren Breite `Distance`. Die Region wird nicht genau in n Teile zerlegt sondern es wird, ausgehend von dem berechneten Auftrennpunkt, eine Stelle mit der geringsten Anzahl von Punkten in vertikaler Richtung gesucht. Der Suchbereich ist maximal plus/minus die Breite eines Rechtecks. Falls `Percent` mit 100 angegeben wird, dann wird dieser Bereich voll ausgenutzt. Bei einem Wert von Null wird keine Suche ausgeführt.

Wenn die Region kleiner als die angegebene Größe ist, bleibt sie unverändert. Eine Zerlegung findet erst statt, wenn die Größe mindestens 1.5 mal so groß ist wie durch die Parameter vorgegeben.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Region die zerlegt werden soll.
- ▷ **Partitioned** (output_object) ..... region-array ⇝ *Hobject*
  Zerlegte Region.
- ▷ **Distance** (input_control) ..... real ⇝ *real*
  Breite der einzelnen Rechtecke.
- ▷ **Percent** (input_control) ..... real ⇝ *real*
  Prozentuale Verschiebung des Auftrennpunktes.
  Defaultwert : 20
  Wertevorschläge : `Percent` ∈ {0, 10, 20, 30, 40, 50, 70, 90, 100}
  Typischer Wertebereich : 0 ≤ `Percent` ≤ 100

Ergebnis

`partition_dynamic` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe

(keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), das bei leerer Ergebnisregion mit set_system('store_empty_region',<true/false>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

partition_dynamic ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, connection

Alternativen

partition_rectangle

Siehe auch

intersection, smallest_rectangle1, shape_trans, clip_region

Modul

Region processing

partition_rectangle (Region : Partitioned : Width, Height :)

Aufteilen einer Region in Rechtecke gleicher Größe.

partition_rectangle zerlegt die Eingaberegion in Rechtecke der Größe Width mal Height. Die Region wird immer in gleich große Rechtecke zerlegt. Deshalb werden Width und Height auf die aktuelle Größe der Region angepaßt. Wenn die Region kleiner als die angegebene Größe ist, bleibt sie unverändert. Eine Zerlegung findet erst statt, wenn die Größe mindestens 1.5 mal so groß ist wie durch die Parameter vorgegeben.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Region die zerlegt werden soll.
- ▷ **Partitioned** (output_object) ..... region(-array) ⇝ *Hobject*
  Zerlegte Region.
- ▷ **Width** (input_control) ..... real ⇝ *real*
  Breite der einzelnen Rechtecke.
- ▷ **Height** (input_control) ..... real ⇝ *real*
  Höhe der einzelnen Rechtecke.

Ergebnis

partition_rectangle liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), das bei leerer Ergebnisregion mit set_system('store_empty_region',<true/false>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

partition_rectangle ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

threshold, connection

Alternativen

partition_dynamic

Siehe auch

intersection, smallest_rectangle1, shape_trans, clip_region

Modul

Region processing

rank_region (Region : RegionCount : Width, Height, Number :)

Rangoperator für Regionen.

rank_region berechnet den binären Rangoperator. Es wird eine Filtermaske (Rechteck: Height x Width) verwendet. Dabei wird für jede Position die Anzahl der Punkte aus Region gezählt, die in der Rechteckmaske liegen. Ist die Anzahl größer oder gleich Number, dann wird der Schwerpunkt des Rechtecks in der Ergebnisregion eingetragen. Wählt man

$$\texttt{Number} = \frac{\texttt{Height} * \texttt{Width}}{2},$$

so erhält man den Median Operator.

Achtung

Als Werte für Height und Width sind nur ungerade Zahlen > 3 zugelassen. Werden andere Werte eingegeben, so werden diese automatisch (ohne Fehlermeldung) entsprechend modifiziert (nächstgrößere ungerade Zahl).

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu transformierende Region(en).
- ▷ **RegionCount** (output_object) ..... region(-array) ⇝ *Hobject*
  Ergebnisregion(en)
- ▷ **Width** (input_control) ..... extent.x ⇝ *integer*
  Breite der Filtermaske.
  Defaultwert : 15
  Wertevorschläge : Width $\in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}$
  Typischer Wertebereich : $3 \leq$ Width ≤ 511 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : (Width ≥ 3) $\wedge$ odd(Width)
- ▷ **Height** (input_control) ..... extent.y ⇝ *integer*
  Höhe der Filtermaske.
  Defaultwert : 15
  Wertevorschläge : Height $\in \{3, 5, 7, 9, 11, 13, 15, 17, 19, 21\}$
  Typischer Wertebereich : $3 \leq$ Height ≤ 511 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : (Height ≥ 3) $\wedge$ odd(Height)
- ▷ **Number** (input_control) ..... integer ⇝ *integer*
  Zahl der Punkte innerhalb der Filtermaske >= Number: Punkt wird in Ausgaberegion aufgenommen.
  Defaultwert : 70
  Wertevorschläge : Number $\in \{5, 10, 20, 40, 60, 80, 90, 120, 150, 200\}$
  Typischer Wertebereich : $1 \leq$ Number ≤ 1000 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
  Restriktion : Number > 0

Beispiel

```
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).
```

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität: $O(F * 8)$.

Ergebnis

`rank_region` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`rank_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`select_shape`, `disp_region`

Alternativen

`closing_rectangle1`, `expand_region`

Siehe auch

`rank_image`, `mean_image`

Modul

Region processing

remove_noise_region (InputRegion : OutputRegion : Type :)

Beseitigung von Rauschen bei Regionen.

`remove_noise_region` dient zum Beseitigen von Rauschen bei Region. Bei dem Modus 'n_4' wird ein strukturierendes Element erzeugt, das aus den vier 4-er Nachbarn eines Punktes besteht. Mit diesem strukturierenden Element wird eine Dilatation und anschliesend der Durchschnitt mit den Eingabedaten durchgeführt. Hierdurch werden alle Punkte, die keinen 4-er Nachbarn haben, gelöscht.

Parameter

- ▷ **InputRegion** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen die modifiziert werden.
- ▷ **OutputRegion** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen mit weniger Rauschen.
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Modus der Rauschunterdrückung.
  Defaultwert : 'n_4'
  Werteliste : Type ∈ {'n_4', 'n_8', 'n_48'}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität $O(\sqrt{F} * 4)$.

Ergebnis

`remove_noise_region` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`remove_noise_region` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`select_shape`

Siehe auch

`dilation1`, `intersection`, `gen_region_points`

Modul

Region processing

shape_trans (Region : RegionTrans : Type :)

Transformation der Form von Regionen.

shape_trans dient zur Transformation der Form von Regionen in Abhängigkeit von Type:

'convex' Konvexe Hülle.

'ellipse' Ellipse mit den gleichen Momenten und Fläche wie die Eingaberegion

'outer_circle' Kleinster umschließender Kreis.

'inner_circle' Größter Inkreis.

'rectangle1' Kleinstes umschließendes Rechteck parallel zu den Koordinatenachsen.

'rectangle2' Kleinstes umschließendes Rechteck.

'inner_center' Liefert als Ausgaberegion den Punkt auf dem Skelett der Eingaberegion, der am nächsten zum Schwerpunkt der Eingaberegion liegt.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die transformiert werden.
- ▷ **RegionTrans** (output_object) ..... region(-array) ⇝ *Hobject*
  Transformierte Regionen.
- ▷ **Type** (input_control) ..... string ⇝ *string*
  Art der Transformation.
  Defaultwert : 'convex'
  Werteliste : Type ∈ {'convex', 'ellipse', 'outer_circle', 'inner_circle', 'rectangle1', 'rectangle2', 'inner_center'}

Komplexität

Sei F die Fläche einer Eingaberegion, dann ist die Laufzeitkomplexität $O(F)$.

Ergebnis

shape_trans liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

shape_trans ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

connection, regiongrowing

Mögliche Nachfolgerfunktionen

disp_region, regiongrowing_mean, area_center

Siehe auch

convexity, elliptic_axis, area_center, smallest_rectangle1, smallest_rectangle2, set_shape, select_shape, inner_circle

Modul

Region processing

skeleton (Region : Skeleton : :)

Verdünnung von Regionen.

skeleton berechnet das Skelett der Eingaberegionen.

Parameter

▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
Zu verdünnende Regionen.

▷ **Skeleton** (output_object) ..... region(-array) ⇝ *Hobject*
Skelett-Region.

Parameteranzahl : `Skeleton` = `Region`

Komplexität

Sei F die Fläche des umschließenden Rechtecks, dann ist die Laufzeitkomplexität: $O(F)$ (pro Region).

Ergebnis

`skeleton` liefert den Wert 2 (H_MSG_TRUE), falls die Parameterwerte korrekt sind. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system ('empty_region_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`skeleton` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`sobel_amp`, `edges_image`, `bandpass_image`, `threshold`, `hysteresis_threshold`

Mögliche Nachfolgerfunktionen

`junctions_skeleton`, `pruning`

Alternativen

`morph_skeleton`, `thinning`

Siehe auch

`gray_skeleton`, `sobel_amp`, `edges_image`, `roberts`, `bandpass_image`, `threshold`

Literatur

Eckardt, U. “Verdünnung mit Perfekten Punkten”, Proceedings 10. DAGM-Symposium, IFB 180, Zurich, 1988

Modul

Region processing

sort_region (Regions : SortedRegions : SortMode, Order, RowOrCol :)

Sortieren von Regionen aufgrund ihrer relativen Lage.

`sort_region` ordnet die Regionen bezüglich ihrer relativen Lage an. Alle Sortierungen mit Ausnahme von **’character’** verwenden einen Punkt der Region. Diese Punkte werden bei `RowOrCol` = **’row’** zuerst bzgl. der Zeile dann bzgl. der Spalte sortiert; bei **’column’** wird zuerst der Spaltenwert verwendet. Der Parameter `SortMode` kann mit folgenden Werten belegt werden:

’character’ Die Regionen werden wie Buchstaben in einer Zeile aufgefäßt und entsprechend der Anordnung in der Schriftzeile angeordnet: Wenn sich zwei Regionen horizontal überlappen, werden sie über den Spaltenwert sortiert, sonst über den Zeilenwert.

’first_point’ Der Punkt mit dem kleinsten Spaltenwert in der ersten Zeile der Region.

’last_point’ Der Punkt mit dem größten Spaltenwert in der letzten Zeile der Region.

’upper_left’ Linkes oberes Eck des umschließenden Rechtecks.

’upper_right’ Rechtes oberes Eck des umschließenden Rechtecks.

’lower_left’ Linkes unteres Eck des umschließenden Rechtecks.

’lower_right’ Rechtes unteres Eck des umschließenden Rechtecks.

Der Parameter `Order` legt fest, ob auf- oder absteigend sortiert wird. Bei **’true’** wird aufsteigend bei **’false’** absteigend sortiert.

Parameter

- ▷ **Regions** (input_object) ..... region-array ⇝ *Hobject*
  Anzuordnende Regionen.
- ▷ **SortedRegions** (output_object) ..... region-array ⇝ *Hobject*
  Angeordnete Regionen.
- ▷ **SortMode** (input_control) ..... string ⇝ *string*
  Art der Sortierung.
  Defaultwert : 'first_point'
  Werteliste : SortMode ∈ {'character', 'first_point', 'last_point', 'upper_left', 'lower_left', 'upper_right', 'lower_right'}
- ▷ **Order** (input_control) ..... string ⇝ *string*
  Aufsteigend oder absteigend sortieren.
  Defaultwert : 'true'
  Werteliste : Order ∈ {'true', 'false'}
- ▷ **RowOrCol** (input_control) ..... string ⇝ *string*
  Anordnung erst nach Zeile oder Spalte.
  Defaultwert : 'row'
  Werteliste : RowOrCol ∈ {'row', 'column'}

Ergebnis

Sind die Parameter korrekt, dann liefert sort_region den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

sort_region ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

do_ocr_multi, do_ocr_single

Modul

Region processing

split_skeleton_lines (SkeletonRegion : : MaxDistance : BeginRow, BeginCol, EndRow, EndCol)

Auftrennen von Linien (codiert durch 1 Pixel breite unverzweigte Linien).

split_skeleton_lines trennt Eingabelinien (codiert durch 1 Pixel breite unverzweigte Regionen) gemäß ihrer Krümmung in Teillinien auf. Eine Linie wird dabei aufgetrennt, wenn der maximale Abstand eines Linienpunktes von der Verbindungsgeraden zwischen Anfangs- und Endpunkt der Linie größer als MaxDistance ist (Split & Merge Algorithmus). Zurückgeliefert wird dann die Geradenapproximation (Anfangs- und Endpunkte) an die Eingabelinie.

Achtung

Die Eingaberegionen müssen unverzweigte Linien codieren, also einzelne Äste des Skeletts.

Parameter

- ▷ **SkeletonRegion** (input_object) ..... region-array ⇝ *Hobject*
  Eingabelinien (codiert als 1 Pixel breite unverzweigte Regionen).
- ▷ **MaxDistance** (input_control) ..... integer ⇝ *integer*
  Krümmungsmaß (max. Abstand der Linienpunkte von der Verbindungsgeraden zwischen Anfangs- und Endpunkt).
  Defaultwert : 3
  Wertevorschläge : MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
  Typischer Wertebereich : 1 ≤ MaxDistance ≤ 500 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **BeginRow** (output_control) ..... line.begin.y-array ⇝ *integer*
  Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.
- ▷ **BeginCol** (output_control) ..... line.begin.x-array ⇝ *integer*
  Spaltenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **EndRow** (output_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **EndCol** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Ausgabelinien.

Beispiel

```
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_lines (SelectedBranches, 3, BeginRow, BeginCol, EndRow,
                      EndCol).
```

Ergebnis

`split_skeleton_lines` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`split_skeleton_lines` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`connection`, `select_shape`, `skeleton`, `junctions_skeleton`, `difference`

Mögliche Nachfolgerfunktionen

`select_lines`, `partition_lines`, `disp_line`

Siehe auch

`split_skeleton_region`, `detect_edge_segments`

Modul

Region processing

split_skeleton_region (SkeletonRegion : RegionLines : MaxDistance :)

Auftrennen von Linien (codiert durch 1 Pixel breite unverzweigte Regionen).

`split_skeleton_region` trennt Eingabelinien (codiert durch 1 Pixel breite unverzweigte Regionen) gemäß ihrer Krümmung in Teillinien auf. Eine Linie wird dabei aufgetrennt, wenn der maximale Abstand eines Linienpunktes von der Verbindungsgeraden zwischen Anfangs- und Endpunkt der Linie größer als `MaxDistance` ist (Split & Merge Algorithmus). Zurückgeliefert wird jedoch keine Geradenapproximation an die Eingabelinien, sondern die Originallinien aufgeteilt auf mehrere Ausgaberegionen.

Achtung

Die Eingaberegionen müssen unverzweigte Linien codieren, also einzelne Äste des Skeletts.

Parameter

▷ **SkeletonRegion** (input_object) ..... region(-array) ⇝ *Hobject*
Eingabelinien (codiert als 1 Pixel breite unverzweigte Regionen).

▷ **RegionLines** (output_object) ..... region-array ⇝ *Hobject*
Aufgespaltene Ausgabelinien.

▷ **MaxDistance** (input_control) . . . . . integer ⇝ *integer*
Krümmungsmaß (max. Abstand der Linienpunkte von der Verbindungsgeraden zwischen Anfangs- und Endpunkt).
Defaultwert : 3
Wertevorschläge : MaxDistance ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Typischer Wertebereich : 1 ≤ MaxDistance ≤ 500 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Beispiel

```
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)
```

Ergebnis

split_skeleton_region liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), und das bei leerer Ergebnisregion mit set_system('store_empty_region',<true/false>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

split_skeleton_region ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

connection, select_shape, skeleton, junctions_skeleton, difference

Mögliche Nachfolgerfunktionen

count_obj, select_shape, select_obj, area_center, elliptic_axis, smallest_rectangle2, get_region_polygon, get_region_contour

Siehe auch

split_skeleton_lines, get_region_polygon, gen_polygons_xld

Modul

Region processing

9.6 Zugriff

get_region_chain (Region : : : Row, Column, Chain)

Kontur von einem Objekt als Kettencode.

get_region_chain gibt die Kontur einer Region aus. Eine Kontur ist eine Folge von Punkten, die die Umrandung der Region beschreiben. Die Kontur „liegt auf“ der Region. Sie beginnt bei der kleinsten Zeilennummer; in dieser Zeile bei dem Punkt mit dem größten Spaltenindex. Der Umlauf erfolgt im Uhrzeigersinn. Hohlflächen der Region werden ignoriert. Der Richtungscode (Kettencode) ist wie folgt definiert:

$$\begin{matrix} 3 & 2 & 1 \\ 4 & * & 0 \\ 5 & 6 & 7 \end{matrix}$$

get_region_chain liefert den Code in Form eines Tupels ab. Bei einer leeren Region sind die Parameter Row und Column gleich Null und Chain ist das leere Tupel.

Achtung

Hohlflächen der Region werden ignoriert. Es darf nur eine Region übergeben werden und die muß genau eine Zusammenhangskomponente haben.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Zu transformierende Region.
- ▷ **Row** (output_control) ..... chain.begin.y ⇝ *integer*
  Zeile des Startpunktes.
- ▷ **Column** (output_control) ..... chain.begin.x ⇝ *integer*
  Spalte des Startpunktes.
- ▷ **Chain** (output_control) ..... chain.code-array ⇝ *integer*
  Richtungscode der Kontur (ab Startpunkt).
  Typischer Wertebereich : $0 \leq$ Chain ≤ 7

Ergebnis

get_region_chain liefert normalerweise den Wert 2 (H_MSG_TRUE). Wird mehr als eine Zusammenhangskomponente übergeben, wird ein Exception ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei leerer Region (die Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_region_chain ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

sobel_amp, threshold, skeleton, edges_image, gen_rectangle1, gen_circle

Mögliche Nachfolgerfunktionen

approx_chain, approx_chain_simple

Siehe auch

copy_obj, get_region_contour, get_region_polygon

Modul

Region processing

get_region_contour (Region : : : Rows, Columns)

Zugriff auf die Kontur eines Objektes.

get_region_contour gibt die Kontur einer Region aus. Eine Kontur ist eine Folge von Zeilen- (Rows) und Spaltenkoordinaten (Columns), die die Umrandung der Region beschreiben. Die Kontur liegt auf der Region. Sie beginnt bei der kleinsten Zeilennummer. In dieser Zeile bei dem Punkt mit dem größten Spaltenindex. Die Umlaufrichtung ist der Uhrzeigersinn. Der erste Punkt der Kontur ist gleich dem letzten. Hohlflächen der Region werden ignoriert. get_region_contour liefert die Koordinaten in Form von Tupeln ab. Eine leere Region wird als leeres Tupel übergeben.

Achtung

Hohlflächen der Region werden ignoriert. Es darf nur eine Region übergeben werden und die muß genau eine Zusammenhangskomponente haben.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Auszugebende Region.
- ▷ **Rows** (output_control) ..... contour.y-array ⇝ *integer*
  Zeilennummern der Konturpunkte.
- ▷ **Columns** (output_control) ..... contour.x-array ⇝ *integer*
  Spaltennummern der Konturpunkte.
  Parameteranzahl : Columns = Rows

Ergebnis

`get_region_contour` liefert normalerweise den Wert 2 (H_MSG_TRUE). Wird mehr als eine Zusammenhangskomponente übergeben, wird eine Exception-Behandlung ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen.

Parallelisierungsinformation

`get_region_contour` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `threshold`, `skeleton`, `edges_image`, `gen_rectangle1`, `gen_circle`

Siehe auch

`copy_obj`, `get_region_chain`, `get_region_polygon`

Modul

Region processing

get_region_convex (Region : : : Rows, Columns)

Zugriff auf die konvexe Hülle als Kontur.

`get_region_convex` gibt die konvexe Hülle einer Region als Polygon aus. Das Polygon ist die minimale Folge von Zeilen- (`Rows`) und Spaltenkoordinaten (`Columns`), die die Hülle der Region beschreiben. Die Polygonpunkte liegen auf der Region. Das Polygon beginnt bei der kleinsten Zeilennummer; in dieser Zeile bei dem Punkt mit dem größten Spaltenindex. Die Umlaufrichtung ist der Uhrzeigersinn. Der erste Punkt des Polygons ist gleich dem letzten. `get_region_convex` liefert die Koordinaten in Form von Tupeln ab. Eine leere Region wird als leeres Tupel übergeben.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Auszugebende Region.
- ▷ **Rows** (output_control) ..... contour.y-array ⇝ *integer*
  Zeilennummern der Konturpunkte.
- ▷ **Columns** (output_control) ..... contour.x-array ⇝ *integer*
  Spaltennummern der Konturpunkte.
  Parameteranzahl : `Columns` = `Rows`

Ergebnis

`get_region_convex` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`get_region_convex` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `skeleton`, `dyn_threshold`

Mögliche Nachfolgerfunktionen

`disp_polygon`

Alternativen

`shape_trans`

Siehe auch

`select_obj`, `get_region_contour`

Modul

Region processing

get_region_points (Region : : : Rows, Columns)

Zugriff auf die Punkte einer Region.

get_region_points gibt die Regionendaten in Form von Koordinatenlisten aus. Die Koordinaten sind in folgender Ordnung sortiert:

$$(r_1, c_1) \leq (r_2, c_2) := (r_1 < r_2) \quad \vee \quad (r_1 = r_2) \wedge (c1 \leq c2)$$

get_region_points liefert die Koordinaten in Form von Tupeln ab. Eine leere Region wird als leeres Tupel übergeben.

Achtung

Es darf nur eine Region übergeben werden.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Auf diese Region wird zugegriffen.
- ▷ **Rows** (output_control) ..... coordinates.y-array ⇝ *integer*
  Zeilennummern der Punkte in der Region
- ▷ **Columns** (output_control) ..... coordinates.x-array ⇝ *integer*
  Spaltennummern der Punkte in der Region.
  Parameteranzahl : Columns = Rows

Ergebnis

get_region_points liefert normalerweise den Wert 2 (H_MSG_TRUE). Wird mehr als eine Region übergeben, wird ein Exception ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen.

Parallelisierungsinformation

get_region_points ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

sobel_amp, threshold, connection

Alternativen

get_region_runs

Siehe auch

copy_obj, gen_region_points

Modul

Region processing

get_region_polygon (Region : : Tolerance : Rows, Columns)

Polygonapproximation einer Region.

get_region_polygon berechnet ein Polygon, das den Rand einer Region annähern soll. Ein Polygon ist eine Folge von Zeilen- (Rows) und Spaltenkoordinaten (Columns), die die Kontur der Region beschreibt. Von dem Polygon werden nur die Stützpunkte ausgegeben. Der Parameter Tolerance gibt an, wie groß der maximale Abstand zwischen dem Polygon und dem Regionenrand sein darf. Hohlflächen der Region werden ignoriert. get_region_polygon liefert die Koordinaten in Form von Tupeln ab.

Achtung

Hohlflächen der Region werden ignoriert. Es darf nur eine Region übergeben werden und die muß genau eine Zusammenhangskomponente haben.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Zu approximierende Region.
- ▷ **Tolerance** (input_control) ..... number ⇝ *real* / integer
  Maximaler Abstand zwischen dem Polygon und dem Regionenrand.
  Defaultwert : 5.0
  Wertevorschläge : Tolerance ∈ {0.0, 2.0, 5.0, 10.0}
  Typischer Wertebereich : 0.0 ≤ Tolerance (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0

▷ **Rows** (output_control) ..... polygon.y-array ⇝ *integer*
Zeilennummern der Stützpunkte der Kontur.

▷ **Columns** (output_control) ..... polygon.x-array ⇝ *integer*
Spaltennummern der Stützpunkte der Kontur.
Parameteranzahl : `Columns` = `Rows`

Ergebnis

`get_region_polygon` liefert normalerweise den Wert 2 (H_MSG_TRUE). Wird mehr als eine Zusammenhangskomponente übergeben, wird eine Exception-Behandlung ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen.

Parallelisierungsinformation

`get_region_polygon` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`sobel_amp`, `threshold`, `skeleton`, `edges_image`

Siehe auch

`copy_obj`, `gen_region_polygon`, `disp_polygon`, `get_region_chain`, `get_region_contour`, `set_line_approx`

Modul

Region processing

`get_region_runs` (Region : : : Row, ColumnBegin, ColumnEnd)

Zugriff auf die Lauflängenkodierung einer Region.

`get_region_runs` gibt die Regionendaten in Form von Sehnentupeln aus. Die Sehnendarstellung entsteht, indem man eine Region zeilenweise mit aufsteigender Zeilenzahl (= von „oben“ nach „unten“) untersucht. Jede Zeile wird von links nach rechts durchlaufen (aufsteigende Spaltenzahl). Dabei werden alle Anfangs- und Endpunkte von Regionenabschnitten (= Sehnen) gespeichert. Eine Region läßt sich somit durch eine Folge von Sehnen beschreiben, wobei eine Sehne durch Zeilennummer, Anfangs- und Endpunkte (Spaltennummer) definiert ist. `get_region_runs` liefert die drei Komponenten der Sehnen in Form von Tupeln. Bei einer leeren Region werden drei leere Tupel zurückgegeben.

Achtung

Es darf nur eine Region übergeben werden.

Parameter

▷ **Region** (input_object) ..... region ⇝ *Hobject*
Auszugebende Region.

▷ **Row** (output_control) ..... chord.y-array ⇝ *integer*
Zeilennummern der Sehnen.

▷ **ColumnBegin** (output_control) ..... chord.x1-array ⇝ *integer*
Spaltennummern der Anfangspunkte der Sehnen.
Parameteranzahl : `ColumnBegin` = `Row`

▷ **ColumnEnd** (output_control) ..... chord.x2-array ⇝ *integer*
Spaltennummern der Endpunkte der Sehnen.
Parameteranzahl : `ColumnEnd` = `Row`

Ergebnis

`get_region_runs` liefert normalerweise den Wert 2 (H_MSG_TRUE). Wird mehr als eine Region übergeben, wird eine Exception-Behandlung ausgelöst. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen.

Parallelisierungsinformation

`get_region_runs` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `connection`

Alternativen

`get_region_points`

Siehe auch

`copy_obj`, `gen_region_runs`

Modul

Region processing

Kapitel 10

Segmentation

auto_threshold (Image : Regions : Sigma :)

Segmentation mit Schwellen aus dem Histogramm.

auto_threshold segmentiert ein einkanaliges Bild mittels mehrfachem Thresholding. Als erstes wird dabei das relative Histogramm über den Grauwerten ermittelt. Dann werden im Histogramm relevante Minima gesucht, die schließlich nacheinander als Schwellen für das Thresholding dienen. Die Schwellen bestehen aus den Werte 0 und 255 und aus allen Minima des Histogramms (nach der Histogrammglättung). Für jedes Grauwertintervall wird *eine* Region erzeugt. Die Anzahl der Regionen ist die Anzahl der Minima + 1. Um so größer der Wert von Sigma ist, um so weniger Regionen werden extrahiert. Das Verfahren ist dann einsetzbar, wenn die gesuchten Regionen einheitliche Grauwerte ausweisen (homogene Regionen).

Parameter

- ▷ **Image** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte
  Zu segmentierendes Bild.
- ▷ **Regions** (output_object) . . . . . region-array ⇝ *Hobject*
  Regionen mit den Punkten innerhalb der Grauwertintervalle.
- ▷ **Sigma** (input_control) . . . . . number ⇝ *real* / integer
  Sigma für Gaußglättung des Histogramms.
  Defaultwert : 2.0
  Wertevorschläge : Sigma ∈ {0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich : 0.0 ≤ Sigma ≤ 100.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.3
  Restriktion : Sigma ≥ 0.0

Beispiel

```
#include  "HalconCpp.h"
#include  <iostream.h>

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
  }

  HImage       image (argv[1]),
               med;
  HWindow      w;
```

```
  w.SetDraw ("margin");
  w.SetColored (12);

  image.Display (w);

  med = image.MedianImage ("circle", 3, -3);
  med.Display (w);

  HRegionArray reg = med.AutoThreshold2 (2.0);
  HRegionArray con = reg.Connection ();

  cout << "Display image after AutoThreshold2 segmentation " << endl;
  con.Display (w);
  w.Click ();

  return (0);
}
```

Parallelisierungsinformation

auto_threshold ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

anisotrope_diff, median_image, illuminate

Mögliche Nachfolgerfunktionen

connection, select_shape, select_gray

Alternativen

gray_histo, smooth_funct_1d_gauss, threshold

Modul

Region processing

bin_threshold (Image : Region : :)

Schwellenwertsegmentation eines schwarz/weißen Bildes.

bin_threshold segmentiert ein einkanaliges Bild mittels eines automatisch bestimmten Schwellenwertes. Als erstes wird dabei das relative Histogramm über den Grauwerten ermittelt. Dann werden im Histogramm relevante Minima gesucht, die als Schwellen für das Thresholding dienen. Um die Anzahl der Minima zu reduzieren, wird das Histogramm wie bei auto_threshold mit einer Gaußmaske geglättet. Die Maske wird dabei so lange vergrößert, bis nur noch ein Minumum vorhanden ist. Ausgewählt werden die Grauwert von Null bis zu diesem Minimum. Das Verfahren ist besonders für die Segmentation von dunklen Buchstaben auf hellem Papier geeignet.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Zu segmentierendes Bild.
- ▷ **Region** (output_object) ..... region(-array) ⇝ *Hobject*
  Dunkle Bereiche des Bildes.

Beispiel

```
#include  "HalconCpp.h"
#include  <iostream.h>

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
```

```
  }

  HImage       image (argv[1]),
               med;
  HWindow      w;

  w.SetDraw ("margin");
  w.SetShape ("rectangle1");
  w.SetColored (12);

  image.Display (w);

  HRegionArray reg = image.BinThreshold();
  HRegionArray con = reg.Connection ();

  cout << "Display image after BinThreshold segmentation " << endl;
  con.Display (w);
  w.Click ();

  return (0);
}
```

Parallelisierungsinformation

`bin_threshold` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`anisotrope_diff`, `median_image`, `illuminate`

Mögliche Nachfolgerfunktionen

`connection`, `select_shape`, `select_gray`

Alternativen

`auto_threshold`, `char_threshold`, `gray_histo`, `smooth_funct_1d_gauss`, `threshold`

Modul

Region processing

`char_threshold` (`Image, HistoRegion : Characters : Sigma, Percent : Threshold`)

Schwellenwertsegmentation für Texte.

`char_threshold` ist darauf spezialisiert, in einkanaligen Bildern dunkle Schriftzeichen von hellem Hintergrund zu segmentieren. Die Vorgehensweise ist dabei wie folgt: Zuerst wird das Histogramm der Grauwerte des Bildes `Image` für die Punkte der Region `HistoRegion` ermittelt und zur Rauschunterdrückung mit einer Gaußmaske (Parameter `Sigma`) geglättet. In diesem Histogramm kennzeichnet das Maximum den Hintergrund (weißes Papier), während die Schriftzeichen ein kleines lokales Maximum bei niedrigen Grauwerten verursachen. Im Gegensatz zum Operator `bin_threshold` wird nun nicht nach dem Minimum zwischen den beiden Maxima gesucht; der Schwellenwert wird stattdessen in Relation zum Maximum des Histogramms, also zum Hintergrund, nach folgender Bedingung bestimmt:

$$Histogramm[\text{Schwellenwert}] * 100.0 < Histogramm[\text{Maximum}] * (100.0 - \texttt{Percent})$$

Falls z.B. `Percent` = 95 gewählt wird, sucht der Operator den Grauwert, dessen Häufigkeit höchstens 5 Prozent der maximalen Häufigkeit beträgt. Da `char_threshold` annimmt, daß die Schriftzeichen dunkler sind als der Hintergrund, wird der Schwellenwert links vom Maximum gesucht.

Im Vergleich zu `bin_threshold` sollte `char_threshold` dann eingesetzt werden, wenn kein deutliches Minimum zwischen den beiden zu den Schriftzeichen bzw. zum Hintergrund gehörenden Maxima besteht oder die Schriftzeichen gar kein lokales Maximum verursachen. Dies tritt z.B. bei ungleichmäßiger Beleuchtung auf, oder wenn das Bild nur wenige Schriftzeichen enthält.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Zu segmentierendes Bild.
- ▷ **`HistoRegion`** (input_object) ..... region ⇝ *Hobject*
  Region, in der das Histogramm berechnet wird.
- ▷ **`Characters`** (output_object) ..... region(-array) ⇝ *Hobject*
  Dunkle Bereiche des Bildes (Buchstaben).
- ▷ **`Sigma`** (input_control) ..... number ⇝ *real*
  Sigma für Glättung des Histogramms.
  Defaultwert : 2.0
  Wertevorschläge : `Sigma` ∈ {0.0, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich : 0.0 ≤ `Sigma` ≤ 50.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.2
- ▷ **`Percent`** (input_control) ..... number ⇝ *real* / integer
  Prozentwert für Grauwertunterschied.
  Defaultwert : 95
  Wertevorschläge : `Percent` ∈ {90, 92, 95, 96, 97, 98, 99, 99.5, 100}
  Typischer Wertebereich : 0.0 ≤ `Percent` ≤ 100.0 (lin)
  Minimale Schrittweite : 0.1
  Empfohlene Schrittweite : 0.5
- ▷ **`Threshold`** (output_control) ..... integer(-array) ⇝ *integer*
  Verwendeter Schwellenwert.

Beispiel

```
#include  "HalconCpp.h"
#include  <iostream.h>

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
  }

  HImage       image (argv[1]),
               med;
  HWindow      w;

  w.SetDraw ("margin");
  w.SetShape ("rectangle1");
  w.SetColored (12);

  image.Display (w);

  HRegionArray reg = image.CharThreshold(0,5,&Threshold);
  HRegionArray con = reg.Connection ();

  cout << "Display image after CharThreshold segmentation " << endl;
  con.Display (w);
  w.Click ();

  return (0);
}
```

Parallelisierungsinformation

`char_threshold` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`anisotrope_diff`, `median_image`, `illuminate`

Mögliche Nachfolgerfunktionen

`connection`, `select_shape`, `select_gray`

Alternativen

`bin_threshold`, `auto_threshold`, `gray_histo`, `smooth_funct_1d_gauss`, `threshold`

Modul

Region processing

`check_difference` (`Image, Pattern` : `Selected` : `Mode, DiffLowerBound, DiffUpperBound, GrayOffset, AddRow, AddCol` :)

Pixelweiser Vergleich zweier Bilder.

`check_difference` wählt aus dem Eingabebild die Bildpunkte ($g_o = g_{\texttt{Image}}$) aus, deren Grauwertabweichung von den entsprechenden Pixel in `Pattern` abhängig vom Parameter `Mode` innerhalb bzw. außerhalb des vorgegebenen Werteintervalls [`DiffLowerBound`, `DiffUpperBound`] liegt. Sei g_p der Grauwert aus `Pattern`, der gegenüber g_o um den Vektor (`AddRow`, `AddCol`) verschoben ist.

Dann werden im Modus 'diff_inside' die Pixel g_o selektiert mit

$$g\_o - g\_p - \texttt{GrayOffset} > \texttt{DiffLowerBound} \quad \text{und}$$
$$g\_o - g\_p - \texttt{GrayOffset} < \texttt{DiffUpperBound}$$

und im Modus 'diff_outside' die g_o mit

$$g\_o - g\_p - \texttt{GrayOffset} \leq \texttt{DiffLowerBound} \quad \text{oder}$$
$$g\_o - g\_p - \texttt{GrayOffset} \geq \texttt{DiffUpperBound}$$

Dieser Test wird für alle Bildpunkte aus dem Definitionsbereich von `Image`, geschnitten mit dem verschobenen Definitionsbereich von `Pattern`, durchgeführt. Alle Punkte, die die obige Bedingung erfüllen, werden in der Ausgaberegion gespeichert. Die beiden Bilder können unterschiedliche Größe haben. Typischerweise ist `Pattern` kleiner als `Image`.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Bild, das unterscht werden soll.
- ▷ **`Pattern`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Vergleichsbild.
- ▷ **`Selected`** (output_object) ..... region(-array) ⇝ *Hobject*
  Punkte, in denen sich die Bilder ähneln/unterscheiden.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Modus: bestimme äehnliche oder verschiedene Pixel.
  Defaultwert : 'diff_outside'
  Wertevorschläge : `Mode` ∈ {'diff_inside', 'diff_outside'}
- ▷ **`DiffLowerBound`** (input_control) ..... number ⇝ *integer*
  Untere Grenze der erlaubten Grauwertdifferenz.
  Defaultwert : -5
  Wertevorschläge : `DiffLowerBound` ∈ {0, -1, -2, -3, -5, -7, -10, -12, -15, -17, -20, -25, -30}
  Typischer Wertebereich : -255 ≤ `DiffLowerBound` ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2
  Restriktion : (`-255` ≤ `DiffLowerBound`) ∧ (`DiffLowerBound` ≤ `255`)

▷ **DiffUpperBound** (input_control) ..... number ⇝ *integer*
Obere Grenze der erlaubten Grauwertdifferenz.
Defaultwert : 5
Wertevorschläge : DiffUpperBound ∈ {0, 1, 2, 3, 5, 7, 10, 12, 15, 17, 20, 25, 30}
Typischer Wertebereich : -255 ≤ DiffUpperBound ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2
Restriktion : (-255 ≤ DiffUpperBound) ∧ (DiffUpperBound ≤ 255)

▷ **GrayOffset** (input_control) ..... number ⇝ *integer*
Korrekturgrauwert, der von Image abgezogen wird.
Defaultwert : 0
Wertevorschläge : GrayOffset ∈ {-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}
Typischer Wertebereich : -255 ≤ GrayOffset ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 2
Restriktion : (-255 ≤ GrayOffset) ∧ (GrayOffset ≤ 255)

▷ **AddRow** (input_control) ..... point.y ⇝ *integer*
Zeilenwert, um den Pattern verschoben wird.
Defaultwert : 0
Wertevorschläge : AddRow ∈ {-200, -100, -20, -10, 0, 10, 20, 100, 200}
Typischer Wertebereich : -32000 ≤ AddRow ≤ 32000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **AddCol** (input_control) ..... point.x ⇝ *integer*
Spaltenwert, um den Pattern verschoben wird.
Defaultwert : 0
Wertevorschläge : AddCol ∈ {-200, -100, -20, -10, 0, 10, 20, 100, 200}
Typischer Wertebereich : -32000 ≤ AddCol ≤ 32000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

Komplexität

Sei F die Anzahl der gültigen Pixel, dann ist die Laufzeitkomplexität: $O(F)$.

Ergebnis

check_difference liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

check_difference ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

connection, select_shape, reduce_domain, select_gray, rank_region, dilation1, opening

Alternativen

sub_image, dyn_threshold

Modul

Region processing

class_2dim_sup (ImageCol, ImageRow, FeatureSpace : RegionClass2Dim : :)

Segmentation mit 2-dimensionaler Pixelklassifikation.

class_2dim_sup klassifiziert Bildpunkte mit Hilfe von zweikanaligen Bildern und einem zweidimensionalen Merkmalsraum. Für jeden Bildpunkt werden zwei Grauwerte (aus jedem Bild einer) als Merkmale verwendet. Der Merkmalsraum wird durch eine Region repräsentiert. Die Klassifikation wird wie folgt durchgeführt:

Ein Bildpunkt aus der Eingaberegion wird genau dann übernommen, wenn der Punkt (g_c, g_l), der durch die Grauwerte festgelegt wird, in der Region FeatureSpace enthalten ist. Dabei ist g_l ein Grauwert aus dem Bild ImageRow und g_c ein Grauwert aus dem Bild ImageCol.

Sei P ein Bildpunkt mit den Koordinaten $P = (L, C)$, g_l der Grauwert an der Position (L, C) in dem Bild ImageRow und g_c der Grauwert an der Position (L, C) in dem Bild ImageCol, dann wird er in die Ergebnisregion (RegionClass2Dim) übernommen, falls:

$$(g_c, g_l) \in \texttt{FeatureSpace}$$

g_l wird also als Zeilen- und g_c als Spaltenkoordinate interpretiert.

Für die Erzeugung von FeatureSpace siehe histo_2dim. Der Merkmalsraum kann vor der Anwendung von class_2dim_sup noch mit Regionentransformationen wie rank_region, dilation1, shape_trans, elliptic_axis, etc. bearbeitet werden.

Die Parameter ImageCol und ImageRow müssen gleich viele Bilder mit jeweils gleicher Größe enthalten. Die Bildpunkte werden aus dem Durchschnitt der beiden Definitionsbereiche entnommen (siehe reduce_domain).

Parameter

- ▷ **ImageCol** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int1 / cyclic / direction
  Bild mit Grauwerten für ersten Kanal.
- ▷ **ImageRow** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Bild mit Grauwerten für zweiten Kanal.
- ▷ **FeatureSpace** (input_object) ..... region(-array) ⇝ *Hobject*
  Region, die den Merkmalsraum festlegt.
- ▷ **RegionClass2Dim** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen, die das Klassifikationskriterium erfüllen.

Beispiel

```
#include  "HalconCpp.h"
#include  <iostream.h>

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
  }

  HRegion   feats, cd2reg;
  HImage    image (argv[1]),
            text1, text2,
            mean1, mean2,
            histo;

  HWindow   win;
  long      nc;

  if ((nc = image.CountChannels ()) != 3)
  {
    cout << argv[1] << " is not a rgb-image " << endl;
    return (-2);
  }

  image.Display (win);

  win.SetColor ("green");
  cout << "Draw the region of interrest " << endl;
```

```
  HRegion  region = win.DrawRegion ();

  text1 = image.TextureLaws ("el", 2, 5);
  mean1 = text1.MeanImage (21, 21);
  text2 = mean1.TextureLaws ("es", 2, 5);
  mean2 = text2.MeanImage (21, 21);

  histo = region.Histo2dim (mean1, mean2);
  feats = histo.Threshold (1.0, 1000000.0);

  win.SetDraw ("fill");
  win.SetColor ("red");

  feats.Display (win);

  cout << "Charakteristics area in red" << endl;

  cd2reg = mean1.Class2dimSup (mean2, feats);

  win.SetColor ("blue");
  cd2reg.Display (win);

  cout << "Result of classification in blue " << endl;
  win.Click ();
  return (0);
}
```

Komplexität

Sei F die Fläche der Eingaberegion dann ist die Laufzeitkomplexität $O(256^2 + F)$.

Ergebnis

`class_2dim_sup` liefert den Wert 2 (H_MSG_TRUE). Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`class_2dim_sup` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`histo_2dim`, `threshold`, `draw_region`, `dilation1`, `opening`, `shape_trans`

Mögliche Nachfolgerfunktionen

`connection`, `select_shape`, `select_gray`

Alternativen

`class_ndim_norm`, `class_ndim_box`, `threshold`, `histo_2dim`

Modul

Region processing

class_2dim_unsup (Image1, Image2 : Classes : Threshold, NumClasses :)

Segmentation von zwei Bildern mittels Clustering.

`class_2dim_unsup` führt eine Klassifikation mit zwei einkanaligen Bildern durch. Als erstes wird ein zweidimensionales Histogramm über die 2 gegebenen Bilder aufgestellt (`histo_2dim`). In diesem zweidimensionalen Histogramm wird das erste Maximum gesucht. Es stellt das erste Clusterzentrum dar. Das Histogramm wird mit den Bildpunkten berechnet, die in beiden Bildern definiert sind (siehe `reduce_domain`). Danach werden im Bild alle Pixel gesucht, die sich höchstens um `Threshold` von den Grauwerten der des Clusterzentrums unterscheiden (Maximumsabstand vom Zentrum). Die gefundenen Pixel bilden eine Region. Anschließend werden die

zu den gefundenen Pixel gehörenden Knoten im Histogramm gelöscht, damit sie bei der nächsten Klassenbildung nicht mehr berücksichtigt werden müssen. In diesem modifizierten Histogramm wird wiederum das Maximum gesucht; es bildet das neue Clusterzentrum. Die eben genannten Schritte werden wiederholt. Insgesamt werden die Schritte `NumClasses`-mal, wiederholt; es ergeben sich also `NumClasses`-viele Ausgaberegionen. Es werden nur Punkte ausgegeben die in beiden Bildern definiert sind.

Achtung

Die beiden Eingabebilder müssen die gleiche Größe haben.

Parameter

- ▷ **`Image1`** (input_object) ..... image ⇝ *Hobject* : byte
  Erstes Eingabebild.
- ▷ **`Image2`** (input_object) ..... image ⇝ *Hobject* : byte
  Zweites Eingabebild.
- ▷ **`Classes`** (output_object) ..... region-array ⇝ *Hobject*
  Ergebnis der Segmentation.
- ▷ **`Threshold`** (input_control) ..... integer ⇝ *integer*
  Schwellenwert (maximaler Abstand vom Clusterzentrum).
  Defaultwert : 15
  Wertevorschläge : `Threshold` ∈ {0, 2, 5, 8, 12, 17, 20, 30, 50, 70}
- ▷ **`NumClasses`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Klassen (Clusertzentren).
  Defaultwert : 5
  Wertevorschläge : `NumClasses` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 30, 40, 50}

Beispiel

```
#include  "HalconCpp.h"
#include  <iostream.h>

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " 'image' " << endl;
    return (-1);
  }

  HImage   colimg (argv[1]),
           green, blue;

  HWindow  w;
  long     nc;

  if ((nc = colimg.CountChannels ()) != 3)
  {
    cout << argv[1] << " is not a rgb-image " << endl;
    return (-2);
  }

  colimg.Display (w);

  HImage        red = colimg.Decompose3 (&green, &blue);
  HRegionArray  seg = red.Class2dimUnsup (green, 15, 5);

  w.SetDraw ("margin");
  w.SetColored (12);
  seg.Display (w);
  w.Click ();
```

```
  return (0);
}
```

Ergebnis

`class_2dim_unsup` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`class_2dim_unsup` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`decompose2`, `decompose3`, `median_image`, `anisotrope_diff`, `reduce_domain`

Mögliche Nachfolgerfunktionen

`select_shape`, `select_gray`, `connection`

Alternativen

`threshold`, `histo_2dim`, `class_2dim_sup`, `class_ndim_norm`, `class_ndim_box`

Modul

Region processing

`class_ndim_box` (MultiChannelImage : Regions : ClassifHandle :)

Klassifikation von Bildpunkten durch Hyperquader.

`class_ndim_box` klassifiziert die Bildpunkte von mehrkanaligen Bildern. Es wird hierzu der Klassifikator `ClassifHandle` verwendet, der mit `create_class_box` erzeugt wurde. Trainiert werden kann der Klassifikator mit `class_ndim_box` oder wie bei `create_class_box` beschrieben. Siehe auch `class_ndim_norm`.

`MultiChannelImage` ist ein mehrkanaliges Bild, dessen Bildpunkte klassifiziert werden sollen. D.h. es werden die Bildpunkte ausgewählt, die dem Klassifikationskriterium genügen.

Parameter

- ▷ **`MultiChannelImage`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic
  Mehrkanaliges Bild.
- ▷ **`Regions`** (output_object) ..... region-array ⇝ *Hobject*
  Ergebnis der Segmentation.
- ▷ **`ClassifHandle`** (input_control) ..... class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.

Beispiel

```
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).
```

Komplexität

Sei N die Anzahl der Hyperquader und F die Fläche der Eingaberegion(en), dann ist die Laufzeitkomplexität $O(N * F)$.

Ergebnis

`class_ndim_box` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`class_ndim_box` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`create_class_box`, `learn_class_box`, `median_image`, `compose2`, `compose3`, `compose4`

Alternativen

`class_ndim_norm`, `class_2dim_sup`

Siehe auch

`class_ndim_box`, `descript_class_box`, `create_class_box`

Modul

Region processing

`class_ndim_norm` (MultiChannelImage : Regions : Metric, SingleMultiple, Radius, Center :)

Pixelklassifikation mit Hyperkugeln oder Hyperwürfeln.

`class_ndim_norm` klassifiziert die Pixel der in `MultiChannelImage` enthaltenen Bilder. Als Ergebnis werden in `Regions` für jedes Klassifizierungsobjekt ein (oder mehrere) Bild(er) ausgegeben (s.u.). Die verwendete Metrik ('euclid' oder 'maximum') wird mit `Metric` festgelegt. Es ist darauf zu achten, daß dieser Parameter genauso wie bei `learn_ndim_norm` besetzt wird. Mit `SingleMultiple` wird festgelegt, ob als Ergebnis (in `Regions`) eine Region ('single') oder für jedes Cluster eine eigene Region ('multiple') erzeugt werden soll. Der Parameter `Radius` gibt die Clusterradien bzw. die halbe Clusterkantenlänge an. `Center` enthält die Koordinaten aller Clusterzentren.

Parameter

- ▷ **`MultiChannelImage`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Mehrkanaliges Bild.
- ▷ **`Regions`** (output_object) ..... region-array ⇝ *Hobject*
  Ergebnis der Segmentation.
- ▷ **`Metric`** (input_control) ..... string ⇝ *string*
  Verwendete Metrik.
  Defaultwert : 'euclid'
  Werteliste : `Metric` ∈ {'euclid', 'maximum'}
- ▷ **`SingleMultiple`** (input_control) ..... string ⇝ *string*
  Als Ergebnis in Result eine Region oder für jedes Cluster eine eigene Region.
  Defaultwert : 'single'
  Werteliste : `SingleMultiple` ∈ {'single', 'multiple'}
- ▷ **`Radius`** (input_control) ..... number(-array) ⇝ *real* / integer
  Clusterradien bzw. halbe Clusterkantenlängen (von `learn_ndim_norm` geliefert).

▷ **Center** (input_control) ..... number(-array) ⇝ *real* / integer
Koordinaten aller Clusterzentren (von learn_ndim_norm geliefert).

Beispiel

```
#include  "HalconCpp.h"
#include  <iostream.h>

int main ()
{
  HImage   image ("meer"),
           t1, t2, t3,
           m1, m2, m3, m;

  HWindow  w;

  w.SetColor ("green");
  image.Display (w);

  cout << "Draw your region of interest " << endl;

  HRegion testreg = w.DrawRegion ();

  t1 = image.TextureLaws ("el", 2, 5);     m1 = t1.MeanImage (21, 21);
  t2 = image.TextureLaws ("es", 2, 5);     m2 = t2.MeanImage (21, 21);
  t3 = image.TextureLaws ("le", 2, 5);     m3 = t3.MeanImage (21, 21);

  m  = m1.Compose3 (m2, m3);

  Tuple Metric = "euclid";
  Tuple Radius = 20.0;
  Tuple MinNum = 5;
  Tuple NbrCha = 3;

  HRegion empty;
  Tuple cen, t;

  Radius = testreg.LearnNdimNorm (empty, m, Metric, Radius,
                                  MinNum, NbrCha, &cen, &t);
  Tuple RegMod = "multiple";

  HRegionArray reg = m.ClassNdimNorm (Metric, RegMod, Radius, cen, NbrCha);

  w.SetColored (12);
  reg.Display (w);
  cout << "Result of classification" << endl;
  return (0);
}
```

Komplexität

Sei N die Anzahl der Cluster (= Länge der Tupel Radius und Center) und F die Fläche der Eingaberegion(en), dann ist die Laufzeitkomplexität $O(N * F)$.

Ergebnis

class_ndim_norm liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

class_ndim_norm ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`learn_ndim_norm`, `compose2`, `compose3`, `compose4`

Mögliche Nachfolgerfunktionen

`connection`, `select_shape`, `reduce_domain`, `select_gray`

Alternativen

`class_ndim_box`, `class_2dim_sup`

Siehe auch

`disp_circle`, `disp_rectangle1`

Modul

Region processing

`detect_edge_segments` (`Image` : : `SobelSize`, `MinAmplitude`, `MaxDistance`, `MinLength` : `BeginRow`, `BeginCol`, `EndRow`, `EndCol`)

Detektion gerader Kantenstücke.

`detect_edge_segments` detektiert gerade Kantenstücke in einem Graustufenbild. Die gefundenen Kantensegmente werden als Strecken Linien) mit einem Anfangs-(`BeginRow`,`BeginCol`) und einem Endpunkt (`EndRow`,`EndCol`) zurückgeliefert. Die Kantendetektion basiert auf dem Sobelfilter (Variante 'sum_abs', Filtergröße, gemäß `SobelSize`). Auf dessen Filterantwort bezieht sich auch der Parameter `MinAmplitude`: Nur Pixel mit Intensität größer oder gleich `MinAmplitude` im gefilterten Eingangsbild werden als Kandidaten für Kantenpunkte verwendet. Die Rohkanten werden verdünnt und in gerade Teilstücke zerlegt. Aus technischen Gründen gehen dabei Kantenpunkte, in denen sich mehrere Kanten treffen, verloren. Somit liefert `detect_edge_segments` im allgemeinen keine geschlossenen Objektkonturen (bzw. lineare Approximationen daran). Der Parameter `MaxDistance` steuert die Zerlegung von Kanten in gerade Teilstücke. Er gibt die maximale Abweichung eines Kantenpunktes von der approximierenden Geraden an. Aus Laufzeitgründen wird dabei nicht der euklidische Abstand, sondern die Betragssumme der Koordinatendifferenzen verwendet. `MinLength` gibt die gewünschte Mindestlänge der Ausgabelinien an. Kürzere Linien werden unterdrückt.

Parameter

- ▷ **`Image`** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **`SobelSize`** (input_control) ..... integer ⇝ *integer*
  Filtergröße für Sobeloperator.
  Defaultwert : 5
  Werteliste : `SobelSize` ∈ {3, 5, 7, 9, 11, 13}
- ▷ **`MinAmplitude`** (input_control) ..... integer ⇝ *integer*
  Mindestamplitude für Kanten.
  Defaultwert : 32
  Wertevorschläge : `MinAmplitude` ∈ {10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 80, 90, 100, 110}
  Typischer Wertebereich : 1 ≤ `MinAmplitude` ≤ 255
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `MinAmplitude` ≥ `0`
- ▷ **`MaxDistance`** (input_control) ..... integer ⇝ *integer*
  Maximale Abweichung der approximierenden Geraden von der ursprünglichen Kante.
  Defaultwert : 3
  Wertevorschläge : `MaxDistance` ∈ {2, 3, 4, 5, 6, 7, 8}
  Typischer Wertebereich : 1 ≤ `MaxDistance` ≤ 30
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `MaxDistance` ≥ `0`

▷ **MinLength** (input_control) ..... integer ⇝ *integer*
Mindestlänge der Ausgabelinien.
Defaultwert : 10
Wertevorschläge : MinLength ∈ {3, 5, 7, 9, 11, 13, 16, 20}
Typischer Wertebereich : 1 ≤ MinLength ≤ 500
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : MinLength ≥ 0

▷ **BeginRow** (output_control) ..... line.begin.y-array ⇝ *integer*
Zeilenkoordinaten der Anfangspunkte der Ausgabelinien.

▷ **BeginCol** (output_control) ..... line.begin.x-array ⇝ *integer*
Spaltenkoordinaten der Anfgangspunkte der Ausgabelinien.

▷ **EndRow** (output_control) ..... line.end.y-array ⇝ *integer*
Zeilenkoordinaten der Endpunkte der Ausgabelinien.

▷ **EndCol** (output_control) ..... line.end.x-array ⇝ *integer*
Spaltenkoordinaten der Endpunkte der Ausgabelinien.

Beispiel

```
Htuple  SobelSize,MinAmplitude,MaxDistance,MinLength;
Htuple  RowBegin,ColBegin,RowEnd,ColEnd;

create_tuple(&SobelSize,1);
set_i(SobelSize,5,0);
create_tuple(&MinAmplitude,1);
set_i(MinAmplitude,32,0);
create_tuple(&MaxDistance,1);
set_i(MaxDistance,3,0);
create_tuple(&MinLength,1);
set_i(MinLength,10,0);
T_detect_edge_segments(Image,SobelSize,MinAmplitude,MaxDistance,MinLength,
                &RowBegin,&ColBegin,&RowEnd,&ColEnd);
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert detect_edge_segments den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

detect_edge_segments ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

sigma_image, median_image

Mögliche Nachfolgerfunktionen

select_lines, partition_lines, select_lines_longest, line_position, line_orientation

Alternativen

sobel_amp, threshold, skeleton

Modul

Image filters

dual_threshold (Image : RegionCrossings : MinSize, MinGray, Threshold :)

Schwellenwertoperator für Bilder mit Vorzeichen.

dual_threshold segmentiert das Eingabebild in Teilregionen mit Grauwerten $\geq$ Threshold („positive“ Bildbereiche) und eine mit Grauwerten $\leq$ -Threshold („negative“ Bildbereiche). „Positive“ oder „negative“ Teilbereiche mit weniger als MinSize Fläche werden dabei ebenso unterdrückt wie Teilbereiche, deren maximaler Grauwert betragsmäßig kleiner als MinGray ist.

Die durchgeführte Segmentation ist nicht vollständig, d.h. die „positiven“ und „negativen“ Ergebnisregionen überdecken zusammengenommen nicht unbedingt das ganze Eingabebild. Es bleiben alle Bildbereiche unberücksichtigt, deren Grauwerte zwischen -Threshold und Threshold (bzw. -MinGray und MinGray) liegen.

dual_threshold schließt sich typischerweise an die Anwendung von Laplace-Operatoren wie (laplace, derive_gauss oder diff_of_gauss) oder eine Bildsubtraktion (sub_image) an. In den Nulldurchgängen des Laplace-Operators spiegeln sich Bildkanten wider, die sich als Trennlinien zwischen den „positiven“ und „negativen“ Bildbereichen mittels dual_threshold bestimmen lassen (Threshold = 1). Der Parameter MinGray steuert dabei die Rauschinvarianz, MinSize das Auflösungsvermögen der Kantendetektion.

Bei Pixeltypen ohne Vorzeichen (byte) wird nur die positive Hälfte der Segmentation durchgeführt. dual_threshold verhält sich also wie eine einfaches Schwellenwertverfahren (threshold) mit nachgeschaltetem connection und select_gray.

Parameter

- ▷ **Image** (input_object) ..... image(-array) $\rightsquigarrow$ *Hobject* : byte / int2 / int4 / real
  Eingabebild mit Laplace-Bild.
- ▷ **RegionCrossings** (output_object) ..... region-array $\rightsquigarrow$ *Hobject*
  „Positive“ und „negative“ Teilregionen.
- ▷ **MinSize** (input_control) ..... integer $\rightsquigarrow$ *integer*
  Teilregionen mit weniger als MinSize Fläche werden unterdrückt.
  Defaultwert : 20
  Wertevorschläge : MinSize $\in \{0, 10, 20, 50, 100, 200, 500, 1000\}$
  Typischer Wertebereich : $0 \leq$ MinSize ≤ 10000 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **MinGray** (input_control) ..... real $\rightsquigarrow$ *real*
  Teilregionen, deren max. Grauwert betragsmäßig kleiner als MinGray ist, werden unterdrückt.
  Defaultwert : 5.0
  Wertevorschläge : MinGray $\in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0\}$
  Typischer Wertebereich : $0.001 \leq$ MinGray ≤ 10000.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : MinGray > 0
- ▷ **Threshold** (input_control) ..... real $\rightsquigarrow$ *real*
  Zurückgeliefert werden Teilregionen, deren Grauwerte größer gleich (bzw. kleiner gleich) Threshold (-Threshold) sind.
  Defaultwert : 2.0
  Wertevorschläge : Threshold $\in \{1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 9.0, 11.0, 15.0, 20.0\}$
  Typischer Wertebereich : $0.001 \leq$ Threshold ≤ 10000.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
  Restriktion : (Threshold ≥ 1) $\wedge$ (Threshold $\leq$ MinGray)

Beispiel

```
/* 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).
```

Ergebnis

dual_threshold liefert normalerweise den Wert 2 (H_MSG_TRUE). Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

dual_threshold ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

min_max_gray, sobel_amp, gauss_image, reduce_domain, diff_of_gauss, sub_image, derivate_gauss

Mögliche Nachfolgerfunktionen

connection, dilation1, erosion1, opening, closing, rank_region, shape_trans, skeleton

Alternativen

threshold, connection, select_shape, select_gray, dyn_threshold, check_difference

Siehe auch

laplace, diff_of_gauss, expand_region

Modul

Region processing

dyn_threshold (OrigImage, ThresholdImage : RegionDynThresh : Offset, LightDark :)

Segmentation mit lokaler Schwelle.

dyn_threshold wählt aus den Eingabebildern die Bildpunkte ($g_o = g_{\texttt{OrigImage}}$) aus, die der Schwellenwertbedingung genügen. Sei $g_m = g_{\texttt{ThresholdImage}}$ dann lautet diese Bedingung bei LightDark = 'light':

$$g_o \geq g_m + \texttt{Offset}$$

Bei LightDark = 'dark' lautet sie:

$$g_o \leq g_m - \texttt{Offset}$$

Bei LightDark = 'equal':

$$g_m - \texttt{Offset} \leq g_o \leq g_m + \texttt{Offset}$$

Und bei LightDark = 'not_equal':

$$g_m - \texttt{Offset} > g_o \vee g_o > g_m + \texttt{Offset}$$

für alle Bildpunkte aus den Regionen mit Grauwert g. D.h. alle Bildpunkte, deren Grauwerte in der Komponente OrigImage größer oder gleich den Grauwerten in der Vergleichskomponente ThresholdImage plus einer Konstante (Offset) sind, werden als Ergebnispunkte übernommen.

Typischerweise werden als Vergleichskomponenten das Original und eine geglättete Version eines Bildes verwendet (z.B. mean_image, gauss_image, etc.). Dann entspricht dyn_threshold in seiner Wirkung etwa der Anwendung von threshold auf ein hochpaßgefiltertes Bild (highpass_image).

Es lassen sich Objektkanten finden, wobei die Größe (Durchmesser) der gesuchten Objekte durch die Maskengröße des Tiefpaßfilters und die Steigung, bzw. Amplitude der Kante festgelegt wird.

Je größer der Maskendurchmesser gewählt wird, desto größer sind die gefundenen Regionen. Als Faustregel sollte die Maskengröße etwa doppelt so groß wie der Durchmesser der gesuchten Objekte sein. Es ist wichtig, den Parameter `Offset` nicht auf Null zu setzen, da dann zu viele kleine Regionen gefunden werden (Rauschen). Sinnvoll sind Werte, die betragsmäßig zwischen 5 und 40 liegen. Je größer `Offset` gewählt wird, desto weniger Punkte werden gefunden und um so kleiner werden die Regionen.

Alle Punkte eines Eingabebildes, die die obige Bedingung erfüllen, werden gemeinsam als eine Region abgespeichert. Die Zusammenhangskomponenten können gegebenenfalls mit `connection` berechnet werden.

Achtung

Wird `Offset` auf $-1 \ldots 1$ gesetzt, dann wird i.a. eine sehr verrauschte Region erzeugt, die viel Speicherplatz benötigt. Wird `Offset` zu groß gewählt (etwa > 60), dann kann es vorkommen, daß kein Punkt die Schwellenwertbedingung erfüllt (also eine leere Region berechnet wird). Wird `Offset` zu klein gewählt (etwa $<$ -60) dann können alle Punkte die Schwellenwertbedingung erfüllen. Es wird also eine maximale Region erzeugt.

Parameter

- ▷ **`OrigImage`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2 / int4 / real
  Bild das segmentiert werden soll.
- ▷ **`ThresholdImage`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int2 / int4 / real
  Bild mit den Schwellenwerten.
- ▷ **`RegionDynThresh`** (output_object) ..... region(-array) ⇝ *Hobject*
  Punkte, die der Schwellenwertbedingung genügen.
- ▷ **`Offset`** (input_control) ..... number ⇝ *real* / integer
  Wird auf die Grauwerte von ThresholdImage für Vergleich aufaddiert.
  Defaultwert : 5.0
  Wertevorschläge : `Offset` ∈ {1.0, 3.0, 5.0, 7.0, 10.0, 20.0, 30.0}
  Typischer Wertebereich : -255.0 ≤ `Offset` ≤ 255.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 5
  Restriktion : (`-255` < `Offset`) ∧ (`Offset` < `255`)
- ▷ **`LightDark`** (input_control) ..... string ⇝ *string*
  Helle,dunkle oder ähnliche Bereiche suchen.
  Defaultwert : 'light'
  Werteliste : `LightDark` ∈ {'dark', 'light', 'equal', 'not_equal'}

Beispiel

```
/* 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).
```

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität $O(F)$

Ergebnis

`dyn_threshold` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dyn_threshold` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`mean_image`, `smooth_image`, `gauss_image`

Mögliche Nachfolgerfunktionen

`connection`, `select_shape`, `reduce_domain`, `select_gray`, `rank_region`, `dilation1`, `opening`

Alternativen

`check_difference`, `highpass_image`, `sub_image`, `threshold`

Siehe auch

`mean_image`, `smooth_image`, `gauss_image`, `connection`, `rank_region`, `dilation1`

Modul

Region processing

`expand_gray` (Regions, Image, ForbiddenArea : RegionExpand : Iterations, Mode, Threshold :)

Füllt „Lücken“ zwischen Regionen (abhängig vom Grau- oder Farbwert) auf oder trennt überlappende Regionen.

`expand_gray` dient dazu, Lücken zwischen den Regionen der Eingabebilder, wie sie z.B. durch Unterdrückung zu kleiner Regionen nach einer Bildsegmentation entstehen, zu schließen (Modus **’image’**) oder überlappende Regionen zu trennen (Modus **’region’**). Beide Effekte beruhen auf der Expansion von Regionen. Dabei wird bei jeder Iteration der Expansion die Bildpunkte innerhalb eines 1-Punkt breiten Streifens um die Region zu dieser hinzugefügt, deren Grau- bzw. Farbwerte sich von den Werten der benachbarten Randpunkte der Region um höchstens `Threshold` (je Kanal) unterscheiden. Bei Bildern vom Typ **’cyclic’** (z.B. Richtungsbilder) werden auch Bildpunkte mit einer Grauwertdifferenz von mindestens $255 - $ `Threshold` zur Region hinzugenommen.

Expandiert wird dabei nur in Bildbereiche, die nicht als „verbotene Bereiche“ (Parameter `ForbiddenArea`) ausgewiesen sind. Die Zahl der Iterationen wird mit dem Parameter `Iterations` festgelegt. Die Übergabe des strings **’maximal’** veranlaßt bei `expand_gray`, das Verfahren so lange zu iterieren, bis es konvergiert, also keine Änderungen mehr auftreten. Eine 0 auf dieser Parameterposition bewirkt die Elimination der überlappenden Bildbereiche der Eingaberegionen. Im Detail unterscheiden sich die beiden Modi **’image’** und **’region’** wie folgt:

’image’ Die Eingaberegionen werden iterativ so lange ausgedehnt, bis sie eine andere Region oder einen Bildrand berühren (bzw. das Wachstum aufgrund zu hoher Grauwertdifferenzen zum Stillstand kommt). Da `expand_gray_ref` alle Regionen simultan bearbeitet, werden die „Lücken“ zwischen den Regionen „gerecht“ auf Regionen mit ähnlichem Grau- oder Farbwert verteilt. Überlappende Regionen werden getrennt, indem ihre gemeinsamen Teilregionen (wiederum „gerecht“) auf sie verteilt werden.

’region’ Es wird keine Expansion der Eingaberegionen durchgeführt, sondern es werden nur überlappende Bildregionen getrennt, indem die gemeinsamen Teilregionen „gerecht“ auf die bzgl. Grau- bzw. Farbwert passende Regionen aufgeteilt werden.

Achtung

Da Regionen nur in Bereiche mit passenden Grau- bzw. Farbwerten expandiert werden, bleiben im allgemeinen Lücken zwischen den Ausgaberegionen bestehen. Die Segmentation der Bildebene ist also nicht total.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, zwischen denen Lücken geschlossen oder die getrennt werden sollen.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Bild (evtl. mehrkanalig) für Graustufen- bzw. Farbtest.
- ▷ **`ForbiddenArea`** (input_object) ..... region ⇝ *Hobject*
  In diesen Bereich darf nicht expandiert werden.
- ▷ **`RegionExpand`** (output_object) ..... region(-array) ⇝ *Hobject*
  Expandierte oder getrennte Regionen.
- ▷ **`Iterations`** (input_control) ..... string ⇝ *string* / integer
  Zahl der Iterationen.
  Defaultwert : ’maximal’
  Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ’maximal’}
  Typischer Wertebereich : $1 \leq$ `Iterations` ≤ 500 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Gewünschter Modus.
  Defaultwert : ’image’
  Werteliste : `Mode` ∈ {’image’, ’region’}

▷ **Threshold** (input_control) . . . . . integer(-array) ⇝ *integer*
Maximalabweichung zwischen Randpixeln und Kandidaten für die Expansion.
Defaultwert : 32
Wertevorschläge : Threshold ∈ {5, 10, 15, 20, 25, 30, 40, 50}
Typischer Wertebereich : 1 ≤ Threshold ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  HImage   image (argv[1]);
  HRegion  empty_region;
  HWindow  win;

  image.Display (win);

  HRegionArray seg = (image >= 100).Connection ();

  seg.Display (win);
  HRegionArray exp = seg.ExpandGray1 (image, empty_region,
                                      "maximal", "image", 32);
  win.SetDraw ("margin");
  win.SetColored (12);
  exp.Display (win);
  win.Click ();

  return (0);
}
```

Ergebnis

expand_gray liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system('no_object_result',<Result>), das bei leerer Region mit set_system('empty_region_result',<Result>), und das bei leerer Ergebnisregion mit set_system('store_empty_region',<true/false>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

expand_gray ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

connection, regiongrowing, pouring, class_ndim_norm

Mögliche Nachfolgerfunktionen

select_shape

Siehe auch

expand_gray_ref, expand_region

Modul

Region processing

expand_gray_ref (Regions, Image, ForbiddenArea : RegionExpand : Iterations, Mode, RefGray, Threshold :)

Füllt „Lücken“ zwischen Regionen (abhängig vom Grau- oder Farbwert) auf oder trennt überlappende Regionen.

`expand_gray_ref` dient dazu, Lücken zwischen den Eingaberegionen, wie sie z.B. durch Unterdrückung zu kleiner Regionen nach einer Bildsegmentation entstehen, zu schließen (Modus **'image'**) oder überlappende Regionen zu trennen (Modus **'region'**). Beide Effekte beruhen auf der Expansion von Regionen. Dabei werden bei jeder Iteration der Expansion die Bildpunkte innerhalb eines 1-Punkt breiten Streifens um die Region zu dieser hinzugefügt, deren Grau- bzw. Farbwerte sich vom übergebenen Referenzgrauwert der Region um höchstens `Threshold` (je Kanal) unterscheiden. Bei Bildern vom Typ **'cyclic'** (z.B. Richtungsbilder) werden auch Bildpunkte mit einer Grauwertdifferenz von mindestens $255 - $ `Threshold` zur Region hinzugenommen.

Expandiert wird dabei nur in Bildbereiche, die nicht als „verbotene Bereiche" (Parameter `ForbiddenArea`) ausgewiesen sind. Die Zahl der Iterationen wird mit dem Parameter `Iterations` festgelegt. Die Übergabe des strings **'maximal'** veranlaßt bei `expand_gray_ref`, das Verfahren so lange zu iterieren, bis es konvergiert, also keine Änderungen mehr auftreten. Eine 0 auf dieser Parameterposition bewirkt die Ausgabe der nicht-überlappenden Teilregionen. Im Detail unterscheiden sich die beiden Modi **'image'** und **'region'** wie folgt:

'image' Die Eingaberegionen werden iterativ so lange ausgedehnt, bis sie eine andere Region oder einen Bildrand berühren (bzw. das Wachstum aufgrund zu hoher Grauwertdifferenzen zum Stillstand kommt). Da `expand_gray_ref` alle Regionen simultan bearbeitet, werden die „Lücken" zwischen den Regionen „gerecht" auf Regionen mit ähnlichem Grau- oder Farbwert verteilt. Überlappende Regionen werden getrennt, indem ihre gemeinsamen Teilregionen (wiederum „gerecht") auf sie verteilt werden.

'region' Es wird keine Expansion der Eingaberegionen durchgeführt, sondern es werden nur überlappende Bildregionen getrennt, indem die gemeinsamen Teilregionen „gerecht" auf die bzgl. Grau- bzw. Farbwert passende Regionen aufgeteilt werden.

Achtung

Da Regionen nur in Bereiche mit passenden Grau- bzw. Farbwerten expandiert werden, bleiben im allgemeinen Lücken zwischen den Ausgaberegionen bestehen. Die Segmentation der Bildebene ist also nicht total.

Parameter

- ▷ **`Regions`** (input_object) ..... region(-array) ⇝ *Hobject*
  Regionen, zwischen denen Lücken geschlossen oder die getrennt werden sollen.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Bild (evtl. mehrkanalig) für Graustufen- bzw. Farbtest.
- ▷ **`ForbiddenArea`** (input_object) ..... region ⇝ *Hobject*
  In diesen Bereich darf nicht expandiert werden.
- ▷ **`RegionExpand`** (output_object) ..... region(-array) ⇝ *Hobject*
  Expandierte oder getrennte Regionen.
- ▷ **`Iterations`** (input_control) ..... string ⇝ *string* / integer
  Zahl der Iterationen.
  Defaultwert : 'maximal'
  Wertevorschläge : `Iterations` ∈ {'maximal', 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 50, 70, 100, 150, 200, 300, 500}
  Typischer Wertebereich : $1 \leq$ `Iterations` ≤ 500 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Gewünschter Modus.
  Defaultwert : 'image'
  Werteliste : `Mode` ∈ {'image', 'region'}
- ▷ **`RefGray`** (input_control) ..... integer(-array) ⇝ *integer*
  Referenzgrauwert.
  Defaultwert : 128
  Wertevorschläge : `RefGray` ∈ {1, 10, 20, 50, 100, 128, 200, 255}
  Typischer Wertebereich : $1 \leq$ `RefGray` ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10

▷ **`Threshold`** (input_control) . . . . . integer(-array) ⇝ *integer*
Maximalabweichung zwischen Referenzgrauwert und Kanditaten für die Expansion.
Defaultwert : 32
Wertevorschläge : `Threshold` ∈ {4, 10, 15, 20, 25, 30, 40}
Typischer Wertebereich : 1 ≤ `Threshold` ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  HImage   image (argv[1]);
  HRegion  empty_region;
  HWindow  win;

  win.SetDraw ("margin");
  win.SetColored (12);

  image.Display (win);

  HRegionArray seg = (image >= 100).Connection ();
  seg.Display (win);

  Tuple iter = "maximal";
  Tuple mode = "image";
  Tuple refg = 128;
  Tuple thrs = 32;

  HRegionArray exp = seg.ExpandGrayRef (image, empty_region,
                                        iter, mode, refg, thrs);
  exp.Display (win);
  win.Click ();

  return (0);
}
```

Ergebnis

`expand_gray_ref` liefert normalerweise den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)`, das bei leerer Region mit `set_system('empty_region_result',<Result>)`, und das bei leerer Ergebnisregion mit `set_system('store_empty_region',<true/false>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`expand_gray_ref` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`connection`, `regiongrowing`, `pouring`, `class_ndim_norm`

Mögliche Nachfolgerfunktionen

`select_shape`

Siehe auch

`expand_gray`, `expand_region`

Modul

Region processing

expand_line (Image : RegionExpand : Index, ExpandType, RowColumn, Threshold :)

Expandieren ausgehend von einer Linie.

expand_line erzeugt eine Region durch Expansion, ausgehend von einer Line (Zeile oder Spalte).

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Zu segmentierendes Eingabebild.
- ▷ **RegionExpand** (output_object) ..... region(-array) ⇝ *Hobject*
  Gefundenes Segmente.
- ▷ **Index** (input_control) ..... integer ⇝ *integer*
  Index der Zeile oder Spalte.
  Defaultwert : 256
  Wertevorschläge : Index ∈ {16, 64, 128, 200, 256, 300, 400, 511}
  Restriktion : Index ≥ 0
- ▷ **ExpandType** (input_control) ..... string ⇝ *string*
  Kriterium für Abbruch.
  Defaultwert : 'gradient'
  Werteliste : ExpandType ∈ {'gradient', 'mean'}
- ▷ **RowColumn** (input_control) ..... string ⇝ *string*
  Segmentationsmodus (Zeile oder Spalte).
  Defaultwert : 'row'
  Werteliste : RowColumn ∈ {'row', 'column'}
- ▷ **Threshold** (input_control) ..... number ⇝ *real* / integer
  Schwellenwert für Ausdehnung.
  Defaultwert : 3.0
  Wertevorschläge : 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}
  Typischer Wertebereich : 1.0 ≤ Threshold ≤ 255.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 1.0
  Restriktion : (Threshold ≥ 0.0) ∧ (Threshold ≤ 255.0)

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  HImage   image (argv[1]),
           gauss;
  HWindow  win;

  win.SetDraw ("margin");
  win.SetColored (12);

  image.Display (win);

  gauss = image.GaussImage (5);

  HRegionArray reg = gauss.ExpandLine (100, "mean", "row", 5.0);

  reg.Display (win);
  win.Click ();

  return (0);
}
```

Parallelisierungsinformation

`expand_line` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `smooth_image`, `anisotrope_diff`, `median_image`, `affine_trans_image`, `rotate_image`

Mögliche Nachfolgerfunktionen

`intersection`, `opening`, `closing`

Alternativen

`regiongrowing_mean`, `expand_gray`, `expand_gray_ref`

Modul

Region processing

`fast_threshold` (Image : Region : MinGray, MaxGray, MinHeight :)

Schnelle Selektion aller Grauwerte innerhalb eines Intervalls.

`fast_threshold` wählt aus den Eingabebildern die Bildpunkte aus, deren Grauwerte g der Schwellenwertbedingung

$$\texttt{MinGray} \leq g \leq \texttt{MaxGray}$$

genügen. Die Auswahl der Punkt erfolgt, zur Verkürzung der Bearbeitungszeit, in zwei Schritten: Zunächst werden nur die Punkte in Zeilen mit Abständen `MinHeight` untersucht. Danach wird in der Umgebung aller so gefundenen Punkte eine genauere Segmentation durchgeführt.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / direction / cyclic
  Zu segmentierendes Bild.
- ▷ **`Region`** (output_object) ..... region(-array) ⇝ *Hobject*
  Region mit Punkten die die Grauwertbedingung erfüllen.
- ▷ **`MinGray`** (input_control) ..... number ⇝ *real* / integer
  Untere Schwelle für die Grauwerte.
  Defaultwert : 128
  Wertevorschläge : `MinGray` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
  Typischer Wertebereich : 0.0 ≤ `MinGray` ≤ 255.0 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5.0
- ▷ **`MaxGray`** (input_control) ..... number ⇝ *real* / integer
  Obere Schwelle für die Grauwerte.
  Defaultwert : 255.0
  Wertevorschläge : `MaxGray` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
  Typischer Wertebereich : 0.0 ≤ `MaxGray` ≤ 255.0 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5.0
- ▷ **`MinHeight`** (input_control) ..... number ⇝ *integer*
  Mindesthöhe eines Objektes, damit es gefunden wird.
  Defaultwert : 20
  Wertevorschläge : `MinHeight` ∈ {5, 10, 15, 20, 25, 30, 40, 50, 60, 70, 100}
  Typischer Wertebereich : 2 ≤ `MinHeight` ≤ 200 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 2

Komplexität

Sei F die Fläche der Ausgaberegion und $height$ die Höhe von `Image`, dann ist die Laufzeitkomplexität $O(F + height/\texttt{MinHeight})$.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `fast_threshold` den Wert 2 (H_MSG_TRUE). Für das Verhalten

bzgl. der Ein- und Ausgabebilder sind die Flags 'no_object_result', 'empty_region_result' und 'store_empty_region' einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fast_threshold` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`histo_to_thresh`, `min_max_gray`, `sobel_amp`, `gauss_image`, `reduce_domain`, `fill_interlace`

Mögliche Nachfolgerfunktionen

`connection`, `dilation1`, `erosion1`, `opening`, `closing`, `rank_region`, `shape_trans`, `skeleton`

Alternativen

`threshold`

Siehe auch

`class_2dim_sup`, `hysteresis_threshold`, `dyn_threshold`

Modul

Region processing

`histo_to_thresh ( : : Histogramm, Sigma : MinThresh, MaxThresh )`

Berechnung von Grauwertschwellen aus einem Histogramm.

`histo_to_thresh` berechnet aus einem Histogramm Schwellen für eine Segmentation mit `threshold`. Als Schwellen werden die Werte 0 und 255 und alle Minima des Histogramms verwendet. Vor der Berechnung der Schwellen wird das Histogramm mit einer Gaußfunktion geglättet.

Parameter

- ▷ **`Histogramm`** (input_control) ..... histogram-array ⇝ *integer* / real
  Grauwerthistogramm.
- ▷ **`Sigma`** (input_control) ..... number ⇝ *real*
  Sigma für Glättung des Histogramms.
  Defaultwert : 2.0
  Wertevorschläge : `Sigma` ∈ {0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich : 0.1 ≤ `Sigma` ≤ 30.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.2
- ▷ **`MinThresh`** (output_control) ..... integer-array ⇝ *integer*
  Minimale Schwellen.
- ▷ **`MaxThresh`** (output_control) ..... integer-array ⇝ *integer*
  Maximale Schwellen.

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]),
           Smoothed;
  HWindow  win;
```

```
  Tuple MinThres, MaxThres,
        HistoAbs, HistoRel,
        size   = 10,
        iter   = 3,
        thresh = 0.0;

  HRegionArray reg = image.GetDomain ();

  HistoAbs = reg.GrayHisto (image, &HistoRel);
  Smoothed = HistoAbs.FunctionSmoothMean (size, iter);
  MinThres = Smoothed.HistoToThresh (thresh, &MaxThres);

  HRegionArray seg = image.Threshold (MinThres, MaxThres);
  HRegionArray con = seg.Connection ();

  /* Alternativkonstrukt fuer Threshold() in
     Aufrufkombination mit Connection()
     -------------------------------------------------------
     HRegionArray con = ((image >= MinThres) &
                         (image <= MaxThres)).Connection ();
     ------------------------------------------------------- */

  con.Display (win);
  win.Click ();

  return (0);
}
```

Parallelisierungsinformation

`histo_to_thresh` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gray_histo`

Mögliche Nachfolgerfunktionen

`threshold`

Alternativen

`auto_threshold`

Modul

Region processing

hysteresis_threshold (Image : RegionHysteresis : Low, High, MaxLength :)

Auswahl von Bildpunkten durch 2 Schwellenwerte.

`hysteresis_threshold` führt eine Schwellenwertoperation mit Hysterese (nach Canny) durch. Dabei werden alle Punkte im Eingabebild `Image` größer oder gleich der oberen Schwelle `High` sofort in die Ausgaberegion(en) übernommen („sichere“ Punkte). Umgekehrt werden alle Punkte mit Grauwerten echt kleiner als die untere Schwelle `Low` zurückgewiesen. „Potentielle“ Punkte mit Grauwerten zwischen den beiden Schwellen schließlich werden dann übernommen, wenn sie durch einen Pfad mit maximaler Länge `MaxLength` von „potentiellen“ Punkten mit einem „sicheren“ Punkt verbunden sind. Die „sicheren“ Punkte strahlen also auf ihre Umgebung aus, sie „wirken nach“ (Hysterese). Die Grauwerte der Eingabebilder bleiben unverändert, lediglich die Regionen werden eventuell verkleinert.

Parameter

▷ **Image** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte
Bild, das segmentiert werden soll.

▷ **RegionHysteresis** (output_object) . . . . . region(-array) ⇝ *Hobject*
Ergebnis der Segmentation.

▷ **Low** (input_control) . . . . . integer ⇝ *integer*
Untere Schwelle für die Grauwerte.
Defaultwert : 30
Wertevorschläge : Low ∈ {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100}
Typischer Wertebereich : 0 ≤ Low ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : (0 < Low) ∧ (Low < 255)

▷ **High** (input_control) . . . . . integer ⇝ *integer*
Obere Schwelle für die Grauwerte.
Defaultwert : 60
Wertevorschläge : High ∈ {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130}
Typischer Wertebereich : 0 ≤ High ≤ 255 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : ((0 < High) ∧ (High < 255)) ∧ (High > Low)

▷ **MaxLength** (input_control) . . . . . integer ⇝ *integer*
Maximale Länge eines Pfades „potentieller“ Punkte zu einem „sicheren“ Punkt hin.
Defaultwert : 10
Wertevorschläge : MaxLength ∈ {1, 2, 3, 5, 7, 10, 12, 14, 17, 20, 25, 30, 35, 40, 50}
Typischer Wertebereich : 1 ≤ MaxLength ≤ 1000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : MaxLength > 1

Ergebnis

hysteresis_threshold liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **’no_object_result’**, **’empty_region_result’** und **’store_empty_region’** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hysteresis_threshold ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Alternativen

dyn_threshold, threshold, class_2dim_sup

Siehe auch

edges_image, sobel_amp, background_seg

Literatur

J. Canny, „Finding Edges and Lines in Images“; Report, AI-TR-720, M.I.T. Artificial Intelligence Lab., Cambridge, MA, 1983.

Modul

Region processing

learn_ndim_box (Foreground, Background, MultiChannelImage : : ClassifHandle :)

Trainieren des aktuellen Klassifikators mit mehrkanaligen Objekten.

learn_ndim_box trainiert den Klassifikator ClassifHandle (Type 2) mit den Bildpunkten der mehrkanaligen Bildern in Foreground. Die Punkte in Background sollen von dem Klassifikator zurückgewiesen werden. Der so trainierte Klassifikator kann für learn_ndim_box zur Segmentation von Bildern verwendet werden. Foreground soll gefunden werden, Background sind die Bildteile, die nicht gefunden werden sollen.

Beim Trainingsvorgang wird jedes Pixel einmal trainiert. Für Bildpunkte aus Foreground wird die Klasse „0“, für Background die Klasse „1“ verwendet. Es wird abwechselnd mit einem Bildpunkt aus Foreground und

einem Background trainiert. Falls eine Region kleiner ist, wird zyklisch von vorne begonnen, bis die andere abgearbeitet ist. learn_ndim_box akzeptiert später bei der Segmentation nur die Bildpunkte, die der Klasse „0“ zugeordnet werden.

Achtung

Alle Kanäle müssen von dem gleichen Typ sein und die gleiche Größe haben.

Parameter

- ▷ **Foreground** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu trainierende Vordergrundpunkte.
- ▷ **Background** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu trainierende Hintergrundpunkte (Rückweisungsklasse).
- ▷ **MultiChannelImage** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real / direction / cyclic
  Mehrkanaliges Grauwertbild.
- ▷ **ClassifHandle** (input_control) ..... class_box ⇝ *integer*
  Nummer des zu verwendenen Klassifikators.

Komplexität

Sei N die Anzahl der erzeugten Hyperquader und F die Fläche der Eingaberegion(en), dann ist die Laufzeitkomplexität $O(N * F)$.

Ergebnis

class_ndim_box liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind und ein Klassifikator aktiv ist. Für das Verhalten bzgl. der Eingabebilder sind die Flags **’no_object_result’** und **’empty_region_result’** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

learn_ndim_box ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

create_class_box, draw_region

Mögliche Nachfolgerfunktionen

class_ndim_box, descript_class_box

Alternativen

learn_class_box

Modul

Region processing

learn_ndim_norm (Foreground, Background, Image : : Metric, Distance, MinNumberPercent : Radius, Center, Quality)

Konstruktion von Clustern für class_ndim_norm.

learn_ndim_norm erzeugt aus den in Foreground enthaltenen Bilder Klassifikationscluster, die in der Prozedur class_ndim_norm verwendet werden. Mit Background wird ein Bild angegeben, das bei der Klassifikation (class_ndim_norm) nicht gefunden werden soll. Dieser Parameter darf auch leer sein (leeres Tupel).

Mit dem Parameter Distance wird der maximale Radius für die Cluster festgelegt. Er beschreibt den minimalen Abstand zweier Clusterzentren. Wird der Parameter Distance klein gewählt, so können die (kleinen) Hyperkugeln (-Würfel) den Merkmalsraum gut approximieren. Gleichzeitig steigt jedoch der Rechenaufwand beim Klassifizieren.

Das Verhältnis aus Anzahl der Pixel in einem Cluster zu der Gesamtzahl (in Prozent) muß über dem Wert von MinNumberPercent liegen, ansonsten wird dieser Cluster nicht ausgegeben. Der Parameter MinNumberPercent dient dazu, Ausreißer in der Trainingsmenge zu eliminieren. Wird er zu groß gewählt, dann werden zu viele Cluster unterdrückt.

Es können zwei verschiedene Verfahren verwendet werden: Das Minimum-Distance-Verfahren (n-dimensionale Kugeln) und das Maximum-Verfahren (n-dimensionale Würfel) für die Beschreibung der Pixel des zu klassifizierenden Bildobjekts im n-dimensionalen Histogramm (Parameter Metric). Die euklid’sche Metrik liefert i.a.

die besseren Ergebnisse, benötigt jedoch auch mehr Rechenzeit. Der Parameter `Quality` gibt die Qualität der Clusterbildung an. Dabei wird die Überschneidung der Abweisungsobjekte mit den Klassifizierungsobjekten berechnet. Null bedeutet maximale Überschneidung, Werte größer Null geben das entsprechende Verhältnis der Überlappung an. Wird kein Zurückweisungsobjekt angegeben, so ist dieser Wert 1. Die Objekte in `Background` haben jedoch keinen Einfluß auf die Konstruktion der Cluster. Sie dienen nur zur Kontrolle der zu erwartenden Ergebnisse.

Parameter

- ▷ **`Foreground`** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Vordergrundregion
- ▷ **`Background`** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Hintergrundregion (Rückweisungsklasse)
- ▷ **`Image`** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Bild mit zugehörigen Grauwerten.
- ▷ **`Metric`** (input_control) . . . . . string ⇝ *string*
  Verwendete Metrik
  Defaultwert : 'euclid'
  Werteliste : `Metric` ∈ {'euclid', 'maximum'}
- ▷ **`Distance`** (input_control) . . . . . number ⇝ *real* / integer
  Maximaler Radius für die Cluster.
  Defaultwert : 10.0
  Wertevorschläge : `Distance` ∈ {1.0, 2.0, 3.0, 4.0, 6.0, 8.0, 10.0, 13.0, 17.0, 24.0, 30.0, 40.0}
  Typischer Wertebereich : 0.0 ≤ `Distance` ≤ 511.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0
  Restriktion : `Radius` > `0.0`
- ▷ **`MinNumberPercent`** (input_control) . . . . . number ⇝ *real* / integer
  Das Verhältnis aus Anzahl der Pixel in einem Cluster zur Gesamtzahl (in Prozent) muß über dem Wert von MinNumberPercent liegen (sonst wird der Cluster nicht ausgegeben).
  Defaultwert : 0.01
  Wertevorschläge : `MinNumberPercent` ∈ {0.001, 0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0}
  Typischer Wertebereich : 0.0 ≤ `MinNumberPercent` ≤ 100.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : (`0` ≤ `MinNumberPercent`) ∧ (`MinNumberPercent` ≤ `100`)
- ▷ **`Radius`** (output_control) . . . . . real-array ⇝ *real*
  Clusterradien bzw. halbe Clusterkantenlängen.
- ▷ **`Center`** (output_control) . . . . . real-array ⇝ *real*
  Koordinaten aller Clusterzentren.
- ▷ **`Quality`** (output_control) . . . . . real ⇝ *real*
  Überschneidung der Abweisungsobjekte mit den Klassifizierungsobjekten (1: keine Überschneidung).
  Zusicherung : (`0` ≤ `Quality`) ∧ (`Quality` ≤ `1`)

Ergebnis

`learn_ndim_norm` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder sind die Flags **'no_object_result'** und **'empty_region_result'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`learn_ndim_norm` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`min_max_gray`, `sobel_amp`, `gauss_image`, `reduce_domain`, `diff_of_gauss`

Mögliche Nachfolgerfunktionen

`class_ndim_norm`, `connection`, `dilation1`, `erosion1`, `opening`, `closing`, `rank_region`, `shape_trans`, `skeleton`

Siehe auch

`class_ndim_norm`, `class_ndim_box`, `histo_2dim`

Literatur

P. Haberäcker, „Digitale Bildverarbeitung“; Hanser-Studienbücher, München, Wien, 1987

Modul

Region processing

local_max (Image : LocalMaxima : :)

Detektion aller Punkte die lokale Maxima sind.

local_max wählt all die Punkte aus, deren Grauwert größer ist als die Grauwerte aller benachbarten Pixel. Die zu berücksichtigende Umgebung kann mit set_system(::'neighborhood',<4/8>) eingestellt werden.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Zu verarbeitendes Bild.
- ▷ **LocalMaxima** (output_object) ..... region(-array) ⇝ *Hobject*
  Gefundene lokale Maxima als Regionen.

  Parameteranzahl : LocalMaxima = Image

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray maxi = cres.LocalMax ();

  win.SetColored (12);
  maxi.Display (win);
  win.Click ();

  return (0);
}
```

Parallelisierungsinformation

local_max ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, smooth_image

Mögliche Nachfolgerfunktionen

get_region_points, connection

Alternativen

gray_skeleton, nonmax_suppression_amp, plateaus, plateaus_center

Siehe auch

monotony, topographic_sketch, corner_response, texture_laws

Modul

Region processing

nonmax_suppression_amp (ImgAmp : ImageResult : Mode :)

Unterdrücken von nicht maximalen Punkten auf einer Kante.

nonmax_suppression_amp unterdrückt Punkte aus dem Definitionsbereich von ImgAmp, deren Grauwerte keine (gerichteten) Maxima sind. Im Gegensatz zu nonmax_suppression_dir wird dabei kein explizites Richtungsbild benötigt. Es stehen zwei Modi zur Verfügung:

'hvnms' Ein Punkt gilt hier als Maximum, wenn sein Grauwert entweder horizontal oder vertikal in einem Suchbereich von $\pm$ 5 Pixeln grösser oder gleich den Grauwerten seiner Nachbarpunkte ist. Nicht-Maximum Punkte werden aus der Region entfernt, die Grauwerte bleiben unverändert.

'loc_max' Ein Punkt gilt hier als Maximum, wenn sein Grauwert größer oder gleich den Grauwerten seiner acht Nachbarpunkte ist.

Parameter

- ▷ **ImgAmp** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Amplitudenbild.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *Hobject* : byte
  Bilder, deren Kantenregion verdünnt ist.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Horizontale/vertikale NMS oder ungerichtete NMS.
  Defaultwert : 'hvnms'
  Werteliste : Mode ∈ {'hvnms', 'loc_max'}

Ergebnis

nonmax_suppression_amp liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

nonmax_suppression_amp ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

sobel_amp

Mögliche Nachfolgerfunktionen

threshold, hysteresis_threshold

Alternativen

gray_skeleton, local_max, gray_dilation_rect

Siehe auch

skeleton

Literatur

S.Lanser: „Detektion von Stufenkanten mittels rekursiver Filter nach Deriche"; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: „Finding Edges and Rows in Images"; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge, MA; 1983.

Modul

Region processing

nonmax_suppression_dir (ImgAmp, ImgDir : ImageResult : Mode :)

Unterdrücken von nicht maximalen Punkten auf einer Kante.

nonmax_suppression_dir unterdrückt Punkte aus den Bildregionen von ImgAmp, deren Grauwerte keine (gerichteten) Maxima sind. ImgDir ist dabei ein Richtungsbild (Einheit: 2 Grad, d.h. 50 Grad sind z.B. mit Wert 25 codiert), wie es z.B. von edges_image geliefert wird. Es stehen zwei Modi zur Verfügung:

'nms' Für jeden Bildpunkt wird getestet, ob sein Grauwert senkrecht zu seiner Richtung maximal ist. Im Modus 'nms' werden dazu nur die beiden der Senkrechten am nächsten liegenden Nachbarpunkte für die Entscheidung herangezogen. Ist der Grauwert eines dieser Nachbarn echt grösser als der Grauwert des zu testenden Punktes, wird dieser unterdrückt (d.h. aus der Region entfernt, die Grauwerte bleiben unverändert).

'inms' Wie 'nms', die beiden benötigten Vergleichswerte werden jedoch durch Interpolation der Grauwerte von vier Nachbarpunkten gewonnen.

Parameter

- ▷ **ImgAmp** (input_object) ..... image(-array) ⇝ *Hobject*
  Amplitudenbild.
- ▷ **ImgDir** (input_object) ..... image(-array) ⇝ *Hobject*
  Richtungsbild.
- ▷ **ImageResult** (output_object) ..... image(-array) ⇝ *Hobject*
  Kantenregion, die verdünnt ist.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Non-Maximum Suppression, interpolierende NMS.
  Defaultwert : 'nms'
  Werteliste : Mode ∈ {'nms', 'inms'}

Ergebnis

nonmax_suppression_dir liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

nonmax_suppression_dir ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Kanal-Ebene*).

Mögliche Vorgängerfunktionen

edges_image, sobel_dir

Mögliche Nachfolgerfunktionen

threshold, hysteresis_threshold

Alternativen

nonmax_suppression_amp, gray_skeleton, gray_dilation_rect

Siehe auch

skeleton

Literatur

S.Lanser: „Detektion von Stufenkanten mittels rekursiver Filter nach Deriche"; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

J.Canny: „Finding Edges and Rows in Images"; Report, AI-TR-720; M.I.T. Artificial Intelligence Lab., Cambridge; 1983.

Modul

Region processing

plateaus (Image : Plateaus : :)

Detektion aller Grauwert-Plateaus.

plateaus wählt all die Punkte aus, deren Grauwert nicht kleiner als der Grauwert ihrer Nachbarn (8er Nachbarschaft) ist. Jedes Maximum ergibt eine eigene Region.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Zu verarbeitendes Bild.
- ▷ **Plateaus** (output_object) ..... region-array ⇝ *Hobject*
  Gefundenen Plateaus als Regionen (für jedes Plateau eine Region).

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray maxi = cres.Plateaus ();

  win.SetColored (12);
  maxi.Display (win);
  win.Click ();

  return (0);
}
```

Parallelisierungsinformation

`plateaus` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`area_center`, `get_region_points`, `select_shape`

Alternativen

`plateaus_center`, `gray_skeleton`, `nonmax_suppression_amp`, `local_max`

Siehe auch

`monotony`, `topographic_sketch`, `corner_response`, `texture_laws`

Modul

Region processing

plateaus_center (Image : Plateaus : :)

Detektion der Schwerpunkte aller Grauwert-Plateaus.

`plateaus_center` wählt all die Punkte aus, deren Grauwert nicht kleiner als der Grauwert ihrer Nachbarn (8-er Nachbarschaft) ist. Wenn mehrere solcher Punkte zusammenhängen (Plateau), wird der Schwerpunkt verwendet. Jedes Maximum ergibt eine eigene Region.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Zu verarbeitendes Bild.
- ▷ **Plateaus** (output_object) ..... region-array ⇝ *Hobject*
  Schwerpunkte der gefundenen Plateaus als Regionen (für jedes Plateau eine Region).

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  if (argc != 2)
  {
    cout << "Usage : " << argv[0] << " <name of image>" << endl;
    return (-1);
  }

  HImage   image (argv[1]);
  HWindow  win;

  image.Display (win);

  HImage       cres = image.CornerResponse (5, 0.04);
  HRegionArray maxi = cres.PlateausCenter ();

  win.SetColored (12);
  maxi.Display (win);
  win.Click ();

  return (0);
}
```

Parallelisierungsinformation

`plateaus_center` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `smooth_image`

Mögliche Nachfolgerfunktionen

`area_center`, `get_region_points`, `select_shape`

Alternativen

`plateaus`, `gray_skeleton`, `nonmax_suppression_amp`, `local_max`

Siehe auch

`monotony`, `topographic_sketch`, `corner_response`, `texture_laws`

Modul

Region processing

pouring (Image : Regions : Mode, MinGray, MaxGray :)

Segmentation nach dem „Giesskannen“-Prinzip.

`pouring` sieht die Werte des Eingabebildes als topologisches (Grauwert-)Gebirge an, das von oben betrachtet wird. Dabei werden große (= helle) Grauwerte als höher gelegene Punkte interpretiert, wogegen kleine (= dunkle) Grauwerte entsprechend die „Täler“ bilden. `pouring` segmentiert die Eingabebilder in mehreren Teilschritten. Zuerst werden die relativ höchsten Grauwerte — im folgenden als „lokale Maxima“ bezeichnet — ermittelt, d.h. die Pixel, die entweder allein oder in Form eines zusammenhängenden Plateaus größere Grauwerte besitzen, als

die unmittelbaren Nachbarn. Die Bestimmung der lokalen Maxima basiert dabei auf Vergleichen in der 4-er Nachbarschaft. Im nächsten Teilschritt bilden die so gefundenen lokalen Maxima den Ausgangspunkt für die Bereichszuordnung, bei der sich die Segmente von den Maxima ausgehend bis in die „Talsohlen“ ausdehnen, d.h. solange absteigende Grauwertketten existieren (wie Wasser, das aus einer Gießkanne über die Kuppen gegossen wurde und nun in alle Richtungen nach unten abläuft). Auch hier wird die 4-er Nachbarschaft zugrunde gelegt, jedoch mit abgeschwächter Bedingung (kleiner gleich). Dabei kann es zur Überschneidungen in den „Talsohlen“ kommen, wenn die Bedingung für mehr als ein Segment erfüllt ist. Diese Schnittmengen werden vorerst keinem Segment zugeordnet, sondern im letzten Teilschritt unter den konkurrierenden Segmenten aufgeteilt. Die Aufteilung erfolgt durch pixelweises gleichmäßiges Ausdehnen aller beteiligten Segmente, bis alle Pixel eindeutig zugeordnet wurden. Mit Hilfe des Parameters `Mode` kann der Programmablauf gesteuert werden. In Abhängigkeit von diesem Eingabeparameter läuft entweder der gesamte Algorithmus ab, oder wird nach Vollendung von Teilschritten — z.B. dem Auffinden aller lokalen Maxima oder der isolierten Segmente ohne Aufteilung der Schnittmengen — beendet. Die möglichen Werte für `Mode` werden nachfolgend erläutert.

’all’ Diese Parameterbelegung stellt den normalen Modus dar. Es wird die komplette Segmentationsroutine gestartet, bei der alle lokalen Maxima mit den zuzuordnenden Bereichen ermittelt werden. Überschneidungsbereiche werden aufgeteilt.

’maxima’ Die Segmentationsroutine terminiert, sobald die lokalen Maxima der Eingabebilder ermittelt worden sind. Eine Segmentation in zugehörige Bereiche wird dann nicht mehr durchgeführt. Die Ausgabe besteht aus den verschiedenen Maxima.

’regions’ Bei dieser Parameterbelegung führt die Segmentationsroutine die Suche nach allen lokalen Maxima sowie den zugehörigen Bereichen, die eindeutig zugeordnet werden können, durch. Die Teilbereiche, in den Überschneidungen auftreten, d.h. die mehr als einem Maximum zugeordnet werden können, werden nicht weiter bearbeitet, also nicht aufgeteilt. Die Ausgabe besteht aus allen ermittelten isolierten Segmenten.

Um bei Ablauf der vollständigen Segmentation die Aufteilung eines gleichmäßigen, vom restlichen Eingabebild verschiedenen Hintergrundes zu vermeiden, bieten die Parameter `MinGray` und `MaxGray` die Möglichkeit, Grauwertschwellen vorzugeben, so daß der Hintergrund ausmaskiert wird. Ein gewünschter unterer Grauwert wird in `MinGray` angegeben. Alle Bildpunkte, die einen kleineren Grauwert besitzen, werden dann weder zur Bestimmung der Maxima, noch zur weiteren Segmentierung herangezogen. Entsprechendes gilt für die Angabe einer oberen Grauwertschwelle in MaxGray: alle Pixel mit einem größeren Grauwert werden ebenfalls einfach ausmaskiert. Für eine komplette Segmentation aller Bildpunkte des Eingabebildes setzt man `MinGray` auf 0 und `MaxGray` auf 255 und erreicht so eine Bearbeitung des kompletten Grauwertspektrums. Auf `MinGray` $<$ `MaxGray` ist zu achten.

Parameter

▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte
Zu segmentierendes Eingabebild.

▷ **`Regions`** (output_object) ..... region-array ⇝ *Hobject*
Ergebnis der Segmentation.

▷ **`Mode`** (input_control) ..... string ⇝ *string*
Gewünschter Modus für den Programmablauf.
Defaultwert : ’all’
Werteliste : `Mode` $\in$ {’all’, ’maxima’, ’regions’}

▷ **`MinGray`** (input_control) ..... integer ⇝ *integer*
Alle Grauwerte, die echt unterhalb dieses Schrankenwertes liegen, werden ausmaskiert.
Defaultwert : 0
Wertevorschläge : `MinGray` $\in$ {0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110}
Typischer Wertebereich : $0 \leq$ `MinGray` ≤ 256 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : `MinGray` $\geq$ `0`

▷ **`MaxGray`** (input_control) ..... integer ⇝ *integer*
Alle Grauwerte, die echt oberhalb dieses Schrankenwertes liegen, werden ausmaskiert.
Defaultwert : 255
Wertevorschläge : `MaxGray` $\in$ {100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 210, 220, 230, 240,

250, 255}
Typischer Wertebereich : 0 ≤ MaxGray ≤ 256 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10
Restriktion : (MaxGray ≤ 255) ∧ (MaxGray > MinGray)

Beispiel

```
/* 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,'affe')
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).
```

Komplexität

Sei N die Anzahl der Bildpunkte des Eingabebildes und M die Anzahl der gefundenen Segmente, wobei das umschreibende Rechteck für ein Segment i insgesamt m_i Pixel beinhalte. Sei außerdem noch K_i die Komplexität der Lauflängenkodierung von Segment i, so beträgt die Laufzeitkomplexität

$$O(3 * N + sum_M(3 * m_i) + sum_M(K_i)) \ .$$

Ergebnis

`pouring` liefert normalerweise den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`pouring` wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`gauss_image`, `smooth_image`

Alternativen

`watersheds`, `local_max`

Siehe auch

`histo_2dim`, `expand_region`, `expand_gray`, `expand_gray_ref`

Modul

Region processing

regiongrowing (Image : Regions : Row, Column, Tolerance, MinSize :)

Segmentation mit Hilfe von Flächenwachstum.

`regiongrowing` liefert Bilder mit „gleicher“ Intensität — gerastert in Rechtecken der Größe `Row` × `Column`. Zur Entscheidung, ob benachbarte Rasterfelder zum selben Bild gehören oder nicht, wird nur die Differenz der Grauwerte ihrer Zentren herangezogen. Ist die Grauwertdifferenz kleiner oder gleich `Tolerance`, werden die Rasterfelder zu einer Region verschmolzen.

Sind g_1 und g_2 zwei zu untersuchende Grauwerte, dann gehören sie zur gleichen Region, falls:

$$|g_1 - g_2| < \texttt{Tolerance}$$

Bei Pixeln vom Typ cyclic wird folgende Formel verwendet:

$$(|g_1 - g_2| < \texttt{Tolerance}) \quad \wedge \quad (|g_1 - g_2| \leq 127)$$

$$(256 - |g_1 - g_2| < \texttt{Tolerance}) \quad \wedge \quad (|g_1 - g_2| > 127)$$

Bei Schrittweiten größer als 1 ist ein typisches Vorgehen, die Eingabeobjekte vor dem Aufruf von `regiongrowing` mit einem Tiefpaß der Größe `Row` × `Column` zu glätten (damit die ausgewerteten Schwerpunkte der Rasterfelder auch wirklich repräsentativ für das gesamte Rasterfeld sind). Ist das Bildmaterial nicht stark verrauscht und die Schrittweite klein, dann kann die vorherige Glättung in vielen Fällen auch entfallen. Dies führt natürlich zu einer Beschleunigung des Gesamtverfahrens.

In den Ergebnisbildern werden bei Schrittweiten größer als 1 nicht nur die untersuchten Punkte gesetzt, sondern es wird um jeden Punkt ein Rechteck der Größe `Row` × `Column` gesetzt. Es werden nur Regionen, die mindestens `MinSize` Punkte umfassen, ausgegeben.

Regiongrowing ist ein sehr schnelles Verfahren und ist daher für zeitkritische Anwendungen zu empfehlen.

Achtung

`Column` und `Row` werden automatisch auf ungerade Werte konvertiert.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int1 / int2 / int4 / cyclic / real
  Zu Segmentierendes Bild.
- ▷ **`Regions`** (output_object) ..... region-array ⇝ *Hobject*
  Gefundene Segmente.
- ▷ **`Row`** (input_control) ..... extent.y ⇝ *integer*
  Vertikaler Abstand zwischen den Testpunkten (Höhe des Rasters).
  Defaultwert : 3
  Wertevorschläge : `Row` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
  Typischer Wertebereich : 1 ≤ `Row` ≤ 99 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : (`Row` ≥ `1`) ∧ odd(`Row`)
- ▷ **`Column`** (input_control) ..... extent.x ⇝ *integer*
  Horizontaler Abstand zwischen den Testpunkten (Breite des Rasters).
  Defaultwert : 3
  Wertevorschläge : `Column` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21}
  Typischer Wertebereich : 1 ≤ `Column` ≤ 99 (lin)
  Minimale Schrittweite : 2
  Empfohlene Schrittweite : 2
  Restriktion : (`Column` ≥ `1`) ∧ odd(`Column`)
- ▷ **`Tolerance`** (input_control) ..... number ⇝ *real* / integer
  Punkte mit Grauwertdifferenz kleiner oder gleich Tolerance werden zum selben Objekt gezählt.
  Defaultwert : 6.0
  Wertevorschläge : `Tolerance` ∈ {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}
  Typischer Wertebereich : 1.0 ≤ `Tolerance` ≤ 127.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0
  Restriktion : (`0` ≤ `Tolerance`) ∧ (`Tolerance` < `127`)

▷ **MinSize** (input_control) ..... integer ⇝ *integer*
Mindestgrösse der Ausgaberegionen.
Defaultwert : 100
Wertevorschläge : MinSize ∈ {1, 5, 10, 20, 50, 100, 200, 500, 1000}
Typischer Wertebereich : 1 ≤ MinSize ≤ 1000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Restriktion : MinSize ≥ 1

Beispiel

```
read_image(Image,'fabrik')
mean_image(Image,Mean,Row,Column)
regiongrowing(Mean,Result,Row,Column,6.0,100).
```

Komplexität

Sei N die Anzahl der gefundenen Regionen und M die Anzahl der untersuchten Punkte in einer solchen Region, dann ist die Laufzeitkomplexität $O(N * log(M) * M)$.

Ergebnis

regiongrowing liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

regiongrowing ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

mean_image, gauss_image, smooth_image, median_image, anisotrope_diff

Mögliche Nachfolgerfunktionen

select_shape, reduce_domain, select_gray

Alternativen

regiongrowing_n, regiongrowing_mean, label_to_region

Modul

Region processing

regiongrowing_mean (Image : Regions : StartRows, StartColumns, Tolerance, MinSize :)

Flächenwachstum mit Mittelwertbildung.

regiongrowing_mean führt ein Flächenwachstum mit Mittelwertbildung ausgehend von vorgegebenen Startpunkten aus. StartRows und StartColumns geben die Startpunkte vor, von denen aus Regionen erzeugt werden. Zu jedem Zeitpunkt des Wachstumsprozesses ist der Mittelwert der bisherigen Region bekannt. Die Grauwerte der Randpunkte der Region werden mit diesem Wert verglichen und zu der Region hinzugefügt, falls der Grauwert nur wenig abweicht (Differenz < Tolerance). Regionen die zu klein sind (Fläche < MinSize) werden unterdrückt.

Werden keine Startpunkte vorgegeben (leere Tupel), dann beginnt der Expansionsprozeß bei dem ersten Bildpunkt (links oben) und wird nach jeder neu entstandenen Region mit dem ersten, bisher unbearbeiteten Bildpunkt, fortgesetzt.

Parameter

▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : byte / int4
Zu segmentierendes Eingabebild.

▷ **Regions** (output_object) ..... region-array ⇝ *Hobject*
Gefundene Segmente.

▷ **StartRows** (input_control) ..... point.y(-array) ⇝ *integer*
Zeilenkoordinaten für Startpunkte.
Defaultwert : '[]'
Typischer Wertebereich : $0 \leq$ StartRows ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **StartColumns** (input_control) ..... point.x(-array) ⇝ *integer*
Spaltenkoordinaten für Startpunkte.
Defaultwert : '[]'
Typischer Wertebereich : $0 \leq$ StartColumns ≤ 511 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1

▷ **Tolerance** (input_control) ..... number ⇝ *real*
Maximale Abweichung vom Mittelwert.
Defaultwert : 5.0
Wertevorschläge : 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}
Typischer Wertebereich : $0.1 \leq$ Tolerance ≤ 100.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 1.0
Restriktion : Tolerance > 0.0

▷ **MinSize** (input_control) ..... integer ⇝ *integer*
Mindestgröße einer Region.
Defaultwert : 100
Wertevorschläge : MinSize $\in$ {0, 10, 30, 50, 100, 500, 1000, 2000}
Typischer Wertebereich : $0 \leq$ MinSize ≤ 50000 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 100
Restriktion : MinSize ≥ 0

Ergebnis

regiongrowing_mean liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

regiongrowing_mean ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

gauss_image, sigma_image, anisotrope_diff, median_image

Mögliche Nachfolgerfunktionen

select_shape, reduce_domain, opening, expand_region

Alternativen

regiongrowing, regiongrowing_n

Modul

Region processing

regiongrowing_n (MultiChannelImage : Regions : Metric, MinTolerance, MaxTolerance, MinSize :)

Flächenwachstumverfahren für mehrkanalige Bilder.

regiongrowing_n führt ein mehrkanaliges Regiongrowing durch. Die n Kanäle liefern in jedem Bildpunkt einen n-dimensionalen Merkmalsvektor. Benachbarte Punkte werden zu einer Ausgaberegion zusammengefaßt, wenn die Differenz ihrer Merkmalsvektoren bezüglich der vorgegebenen Metrik im Intervall [MinTolerance, MaxTolerance] liegen. Untersucht werden direkte Nachbarn in der 4-er Nachbarschaft. Dabei stehen folgende Metriken zur Verfügung:

g_A bezeichne einen Grauwert im Vektor A, dem Merkmalsvektor in einem Punkt a, g_B die entsprechenden Grauwerte im Vektor B, dem Merkmalsvektor in einem Nachbarpunkt b. $g(d)$ ist der Grauwert mit Index d. $MinT$ steht für `MinTolerance`, $MaxT$ für `MaxTolerance`.

'1-norm': Betragssumme

$$MinT \leq \frac{1}{n} \sum |g_A - g_B| \leq MaxT$$

'2-norm': Euklidischer Abstand

$$MinT \leq \sqrt{\frac{\sum (g_A - g_B)^2}{n}} \leq MaxT$$

'3-norm': p - Norm mit p = 3

$$MinT \leq \sqrt[3]{\frac{\sum (g_A - g_B)^3}{n}} \leq MaxT$$

'4-norm': p - Norm mit p = 4

$$MinT \leq \sqrt[4]{\frac{\sum (g_A - g_B)^4}{n}} \leq MaxT$$

'n-norm': Minkowsky-Distanz

$$MinT \leq \sqrt[n]{\frac{\sum (g_A - g_B)^n}{n}} \leq MaxT$$

'max-diff': Supremumsdistanz

$$MinT \leq \max \{|g_A - g_B|\} \leq MaxT$$

'min-diff': Infimumsdistanz

$$MinT \leq \min \{|g_A - g_B|\} \leq MaxT$$

'variance': Varianz der Grauwertdifferenzen

$$MinT \leq Var(g_A - g_B) \leq MaxT$$

'dot-product':

$$MinT \leq \frac{1}{n} \sqrt{\sum (g_A g_B)} \leq MaxT$$

'correlation':

$$m_A = \frac{1}{n} \sum g_A$$

$$Var_A = \frac{1}{n} \sqrt{\sum (g_A - m_A)^2}$$

$$m_B = \frac{1}{n} \sum g_B$$

$$Var_B = \frac{1}{n} \sqrt{\sum (g_B - m_B)^2}$$

$$MinT \leq \frac{1}{n^2} \sum \frac{(g_A - m_A)(g_B - m_B)}{(Var_A Var_B)} \leq MaxT$$

'mean-diff': Differenz der arithmetischen Mittel

$$a = \frac{1}{n} \sum g_A$$

$$b = \frac{1}{n} \sum g_B$$

$$MinT \leq |a - b| \leq MaxT$$

'mean-ratio': Quotient der arithmetischen Mittel

$$a = \frac{1}{n} \sum g_A$$

$$b = \frac{1}{n} \sum g_B$$

$$MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT$$

'length-diff': Differenz der Vektorlängen

$$a = \sqrt{\frac{\sum g_A^2}{n}}$$

$$b = \sqrt{\frac{\sum g_B^2}{n}}$$

$$MinT \leq |a - b| \leq MaxT$$

'length-ratio': Quotient der Vektorlängen

$$a = \sqrt{\frac{\sum g_A^2}{n}}$$

$$b = \sqrt{\frac{\sum g_B^2}{n}}$$

$$MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT$$

'n-norm-ratio': Quotient der Vektorlängen bzgl. der p-Norm mit p = n

$$a = \sqrt[n]{\frac{\sum g_A^n}{n}}$$

$$b = \sqrt[n]{\frac{\sum g_B^n}{n}}$$

$$MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT$$

'gray-max-diff': Differenz der Maxima

$$a = \max \{|g_A|\}$$

$$b = \max \{|g_B|\}$$

$$MinT \leq |a - b| \leq MaxT$$

'gray-max-ratio': Quotient der Maxima

$$a = \max \{|g_A|\}$$

$$b = \max \{|g_B|\}$$

$$MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT$$

'gray-min-diff': Differenz der Minima

$$a = \min \{|g_A|\}$$

$$b = \min \{|g_B|\}$$

$$MinT \leq |a - b| \leq MaxT$$

'gray-min-ratio': Quotient der Minima

$$a = \min \{|g_A|\}$$

$$b = \min \{|g_B|\}$$

$$MinT \leq \min \left\{ \frac{a}{b}, \frac{b}{a} \right\} \leq MaxT$$

'variance-diff': Differenz der Varianzen über allen Grauwerten (Kanälen)

$$MinT \leq |Var(g_A) - Var(g_B)| \leq MaxT$$

'variance-ratio': Quotient der Varianzen über allen Grauwerten (Kanälen)

$$MinT \leq \frac{Var(g_B)}{Var(g_A)} \leq MaxT$$

'mean-abs-diff': Differenz der Betragssummen über allen Grauwerten (Kanälen)

$$a = \sum_{d,k,k<d} |g_A(d) - g_A(k)|$$

$$b = \sum_{d,k,k<d} |g_B(d) - g_B(k)|$$

$$MinT \leq \frac{|a-b|}{\text{Anzahl der Summen}} \leq MaxT$$

'mean-abs-ratio': Quotient der Betragssummen über allen Grauwerten (Kanälen)

$$a = \sum_{d,k,k<d} |g_A(d) - g_A(k)|$$

$$b = \sum_{d,k,k<d} |g_B(d) - g_B(k)|$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'max-abs-diff': Differenz der maximalen Abstände der Komponenten

$$a = \max\{g_A(d), g_A(k)\}$$

$$b = \max\{g_B(d), g_B(k)\}$$

$$MinT \leq |a-b| \leq MaxT$$

'max-abs-ratio': Quotient der maximalen Abstände der Komponenten

$$a = \max\{g_A(d), g_A(k)\}$$

$$b = \max\{g_B(d), g_B(k)\}$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'min-abs-diff': Differenz der minimalen Abstände der Komponenten

$$a = \min\{g_A(d), g_A(k)\}, k < d$$

$$b = \min\{g_B(d), g_B(k)\}, k < d$$

$$MinT \leq |a-b| \leq MaxT$$

'min-abs-ratio': Quotient der minimalen Abstände der Komponenten

$$a = \min\{g_A(d), g_A(k)\}, k < d$$

$$b = \min\{g_B(d), g_B(k)\}, k < d$$

$$MinT \leq \min\left\{\frac{a}{b}, \frac{b}{a}\right\} \leq MaxT$$

'plane': Für alle d_1,$d_2 \in [1, n]$ muß gelten:

$$g_A(d_1) > g_A(d_2) \Rightarrow g_B(d_1) > g_B(d_2)$$

$$g_A(d_1) < g_A(d_2) \Rightarrow g_B(d_1) < g_B(d_2)$$

Regionen mit einer Fläche geringer als `MinSize` werden unterdrückt.

Parameter

- ▷ **MultiChannelImage** (input_object) . . . . . image(-array) ⇝ *Hobject* : byte
  Zu segmentierendes Bild mit mehrkanaligen Bilddaten.
- ▷ **Regions** (output_object) . . . . . region-array ⇝ *Hobject*
  Ergebnis der Segmentation.
- ▷ **Metric** (input_control) . . . . . string ⇝ *string*
  Metrik für Distanz der Merkmalsvektoren.
  Defaultwert : '2-norm'
  Werteliste : Metric ∈ {'1-norm', '2-norm', '3-norm', '4-norm', 'n-norm', 'max-diff', 'min-diff', 'variance', 'dot-product', 'correlation', 'mean-diff', 'mean-ratio', 'length-diff', 'length-ratio', 'n-norm-ratio', 'gray-max-diff', 'gray-max-ratio', 'gray-min-diff', 'gray-min-ratio', 'variance-diff', 'variance-ratio', 'mean-abs-diff', 'mean-abs-ratio', 'max-abs-diff', 'max-abs-ratio', 'min-abs-diff', 'min-abs-ratio', 'plane'}
- ▷ **MinTolerance** (input_control) . . . . . number ⇝ *real* / integer
  Untere Schwelle für den Merkmalsabstand.
  Defaultwert : 0.0
  Wertevorschläge : MinTolerance ∈ {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}
  Typischer Wertebereich : 0.0 ≤ MinTolerance ≤ 255.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0
- ▷ **MaxTolerance** (input_control) . . . . . number ⇝ *real* / integer
  Obere Schwelle für den Merkmalsabstand.
  Defaultwert : 20.0
  Wertevorschläge : MaxTolerance ∈ {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}
  Typischer Wertebereich : 0.0 ≤ MaxTolerance ≤ 255.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 1.0
- ▷ **MinSize** (input_control) . . . . . integer ⇝ *integer*
  Mindestgröße der Ausgaberegionen.
  Defaultwert : 30
  Wertevorschläge : MinSize ∈ {1, 10, 25, 50, 100, 200, 500, 1000}
  Typischer Wertebereich : 1 ≤ MinSize ≤ 10000 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 5

Ergebnis

regiongrowing_n liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe set_system). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

regiongrowing_n ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

compose2, compose3

Alternativen

class_2dim_sup, class_ndim_norm, class_ndim_box

Siehe auch

regiongrowing

Modul

Region processing

threshold (Image : Region : MinGray, MaxGray :)

Selektion aller Grauwerte innerhalb eines Intervalls.

`threshold` wählt aus den Eingabebildern die Bildpunkte aus, deren Grauwerte g der Schwellenwertbedingung

$$\texttt{MinGray} \leq g \leq \texttt{MaxGray}$$

genügen.

Alle Punkte eines Eingabebildes, die die Bedingung erfüllen, werden gemeinsam als eine neue Region abgespeichert. Wird mehr als ein Grauwertbereich übergeben (Tupel von Werten für `MinGray` und `MaxGray`), dann wird für jedes dieser Intervalle eine Region erzeugt.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject* : byte / direction / cyclic / int2 / int4 / real
  Zu segmentierendes Bild.
- ▷ **`Region`** (output_object) ..... region(-array) ⇝ *Hobject*
  Region mit Punkten die die Grauwertbedingung erfüllen.
- ▷ **`MinGray`** (input_control) ..... number(-array) ⇝ *real* / integer
  Untere Schwelle für die Grauwerte.
  Defaultwert : 128.0
  Wertevorschläge : `MinGray` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
  Typischer Wertebereich : 0.0 ≤ `MinGray` ≤ 255.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 5.0
- ▷ **`MaxGray`** (input_control) ..... number(-array) ⇝ *real* / integer
  Obere Schwelle für die Grauwerte.
  Defaultwert : 255.0
  Wertevorschläge : `MaxGray` ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}
  Typischer Wertebereich : 0.0 ≤ `MaxGray` ≤ 255.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 5.0
  Restriktion : `MaxGray` ≥ `MinGray`

Beispiel

```
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).
```

Komplexität

Sei F die Fläche der Eingaberegion, dann ist die Laufzeitkomplexität $O(F)$.

Ergebnis

`threshold` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`threshold` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`histo_to_thresh`, `min_max_gray`, `sobel_amp`, `gauss_image`, `reduce_domain`, `fill_interlace`

Mögliche Nachfolgerfunktionen

`connection`, `dilation1`, `erosion1`, `opening`, `closing`, `rank_region`, `shape_trans`, `skeleton`

Alternativen

`class_2dim_sup`, `hysteresis_threshold`, `dyn_threshold`

Siehe auch

dual_threshold, zero_crossing, background_seg, regiongrowing

Modul

Region processing

threshold_sub_pix (Image : Border : Threshold :)

Subpixel-genaue Extraktion von Grauwert-Höhenlinien in einem Bild

threshold_sub_pix extrahiert die Grauwert-Höhenlinien der Höhe Threshold des Eingabebildes Image subpixel-genau. Die extrahierten Höhenlinien werden als XLD-Konturen in Border zurückgegeben. Im Gegensatz zum Operator threshold liefert threshold_sub_pix keine Fläche zurück, sondern die Linien, die Bereiche mit Grauwert kleiner Threshold von Bereichen mit Grauwert größer Threshold trennen.

Bei der Extraktion wird das Eingabebild als Oberfläche interpretiert, wobei zwischen den Pixelmittelpunkten bilinear interpoliert wird. Konsistent mit der so entstehenden Oberfläche werden einzelne Grauwert-Höhenlinien für jedes Pixel extrahiert und zu topologisch „sauberen“ Konturen verknüpft. Das bedeutet, daß die Höhenlinienkonturen an Kreuzungspunkten sauber aufgetrennt werden. Falls im Bild flächenhafte Bereiche mit einem konstanten Grauwert von Threshold auftreten, wird nur der Rand solcher Gebiete als Höhenlinie zurückgeliefert.

Parameter

- ▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte / int1 / int2 / int4 / real
  Eingabebild.
- ▷ **Border** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Extrahierte Grauwert-Höhenlinien.
- ▷ **Threshold** (input_control) ..... number ⇝ *real* / integer
  Schwellenwert für die Grauwert-Höhenlinien.
  Defaultwert : 128
  Wertevorschläge : Threshold ∈ {0.0, 10.0, 30.0, 64.0, 128.0, 200.0, 220.0, 255.0}

Beispiel

```
read_image(Image,'fabrik')
threshold_sub_pix(Image,Border,35)
disp_xld(Border,WindowHandle)
```

Ergebnis

threshold_sub_pix liefert normalerweise den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

threshold_sub_pix ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

threshold

Siehe auch

zero_crossing_sub_pix

Modul

Sub-pixel operators

watersheds (GrayImage : Basins, Watersheds : :)

Segmentation nach dem „Wasserscheiden“-Prinzip.

watersheds zerlegt das Bild aufgrund der Topologie der Grauwerte. Diese werden als relative Höhenwerte interpretiert. Große Werte entsprechen „hohen Bereichen“ niedrige Werte entsprechen „tiefen Bereichen“. In dem hieraus entstehenden dreidimensionalen Grauwertgebirge werden Wasserscheiden gesucht. Wasserscheiden sind

hellen Grate zwischen dunklen Becken. Der Parameter `Basins` enthält diese dunklen Becken. `Watersheds` enthält die Punkte der Wasserscheiden die mindestens 1 Pixel breit sind (Punkte auf dem Grat, die ein Plateau bilden). `Watersheds` ist immer eine Region pro Eingabebild, `Basins` dagegen enthält für jedes Becken eine eigene Region. Es ist günstig vorher einen Glättungsfilter zu verwenden um die Anzahl der Regionen zu reduzieren (z.B. `gauss_image`).

Achtung

Wenn das Bild feine Strukturen enthält oder verrauscht ist, entstehen sehr viele Regionen und damit steigt die Laufzeit deulich an.

Parameter

- ▷ **`GrayImage`** (input_object) ..... image(-array) ⇝ *Hobject* : byte
  Zu segmentierende Eingabebilder.
- ▷ **`Basins`** (output_object) ..... region-array ⇝ *Hobject*
  Gefundene Segmente (dunkle Becken).
- ▷ **`Watersheds`** (output_object) ..... region(-array) ⇝ *Hobject*
  Wasserscheiden zwischen den Becken.

Beispiel

```
#include  <iostream.h>
#include  "HalconCpp.h"

int main (int argc, char *argv[])
{
  HImage   image (argv[1]),
             invert, gauss;
  HWindow  win;

  cout << "Gauss of original " << endl;
  gauss  = image.GaussImage (9);
  image.Display (win);

  cout << "Invert of Gauss " << endl;
  invert = gauss.InvertImage ();
  invert.Display (win);

  HRegion        watersheds;
  HRegionArray   bassins = invert.Watersheds (&watersheds);

  win.SetColored (12);
  bassins.Display (win);
  win.Click ();

  return (0);
}
```

Ergebnis

`watersheds` liefert normalerweise den Wert 2 (H_MSG_TRUE). Für das Verhalten bzgl. der Eingabebilder und Ausgaberegionen sind die Flags **'no_object_result'**, **'empty_region_result'** und **'store_empty_region'** einstellbar (siehe `set_system`). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`watersheds` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`gauss_image`, `smooth_image`, `invert_image`

Mögliche Nachfolgerfunktionen

`expand_region`, `select_shape`, `reduce_domain`, `opening`

Alternativen

`pouring`

Modul

Region processing

zero_crossing (Image : RegionCrossing : :)

Nulldurchgänge im angegebenen Bild.

zero_crossing liefert als Ausgaberegion die Nulldurchgänge im übergebenen Eingabebild zurück. Ein Pixel wird dabei als Nulldurchgang markiert (d.h. in die Ausgaberegion RegionCrossing aufgenommen), wenn sein Grauwert (in Image) gleich Null ist oder mindestens einer seiner Nachbarn in 4er Nachbarschaft ein anderes Vorzeichen aufweist.

Diese Routine wird sinnvollerweise nach einer Kantenfilterung, die die zweite Ableitung bestimmt (z.B. laplace_of_gauss), gegebenenfalls gefolgt von einer Glättungsoperation aufgerufen. In diesem Fall markieren die Nulldurchgänge Kanten(kandidaten).

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject* : int2 / int4 / real
  Eingabebild.
- ▷ **RegionCrossing** (output_object) ..... region(-array) ⇝ *Hobject*
  Nulldurchgänge (als Region).

Ergebnis

zero_crossing liefert normalerweise den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

zero_crossing ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

laplace, laplace_of_gauss, derivate_gauss

Mögliche Nachfolgerfunktionen

connection, skeleton, boundary, select_shape, fill_up

Alternativen

threshold, dual_threshold

Modul

Region processing

zero_crossing_sub_pix (Image : ZeroCrossings : :)

Subpixel-genaue Extraktion von Nulldurchgängen in einem Bild

zero_crossing_sub_pix extrahiert die Nulldurchgänge des Eingabebildes Image subpixel-genau. Die extrahierten Nulldurchgänge werden als XLD-Konturen in ZeroCrossings zurückgegeben. zero_crossing_sub_pix kann somit zur subpixel-genauen Kantenextraktion verwendet werden, wenn als Eingabebild ein Laplace-gefiltertes Bild übergeben wird (siehe laplace, laplace_of_gauss, derivate_gauss).

Bei der Extraktion wird das Eingabebild als Oberfläche interpretiert, wobei zwischen den Pixelmittelpunkten bilinear interpoliert wird. Konsistent mit der so entstehenden Oberfläche werden einzelne Nulldurchgangslinien für jedes Pixel extrahiert und zu topologisch „sauberen“ Konturen verknüpft. Das bedeutet, daß die Nulldurchgangskonturen an Kreuzungspunkten sauber aufgetrennt werden. Falls im Bild flächenhafte Bereiche mit einem konstanten Grauwert von 0 auftreten, wird nur der Rand solcher Gebiete als Nulldurchgang zurückgeliefert.

Parameter

- ▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : int1 / int2 / int4 / real
  Eingabebild.
- ▷ **ZeroCrossings** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Extrahierte Nulldurchgänge.

Beispiel

```
/* 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')
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)
```

Ergebnis

`zero_crossing_sub_pix` liefert normalerweise den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`zero_crossing_sub_pix` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`laplace`, `laplace_of_gauss`, `diff_of_gauss`, `derivate_gauss`

Alternativen

`zero_crossing`

Modul

Sub-pixel operators

Kapitel 11

System

11.1 Betriebssystem

count_seconds (: : : Seconds)

Seit dem letzten Aufruf von count_seconds *vergangene Prozeßzeit.*

count_seconds dient der Zeitmessung. Der erste Aufruf der Prozedur liefert i.A. eine Null, jeder weitere Aufruf dann die seither vergangene Prozeßzeit in Sekunden. Zur Messung wird die C-Funktion „clock“ verwendet.

Achtung

Die Zeitmessung ist nicht exakt und hängt von der Auslastung des Rechners ab.

Parameter

▷ **Seconds** (output_control) . . . . . real ⇝ *real*
Prozeßzeit seit dem Programmstart.

Beispiel

```
count_seconds(:::Start) >
/* aktion to be measured */
count_seconds(:::End) >
eval(::End - Start:RunTime).
```

Ergebnis

count_seconds liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

count_seconds ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

System

system_call (: : Command :)

Ausführung eines Systemaufrufs.

system_call führt einen Systemaufruf (C-Prozedur „system“) mit den Befehlen in Command aus. Wird der leere String übergeben, dann wird eine interaktive Shell gestartet (’csh -i’).

Parameter

▷ **Command** (input_control) . . . . . string ⇝ *string*
Befehl für das Betriebssystem.
Defaultwert : ’ls’

Ergebnis

Ist der eingegebene Befehl vom Betriebssystem ausführbar, dann liefert `system_call` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung ausgeführt.

Parallelisierungsinformation

`system_call` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`count_seconds`

Siehe auch

`wait_seconds`, `count_seconds`

Modul

System

`wait_seconds` (: : Seconds :)

Verzögerung der Programmausführung.

`wait_seconds` verzögert die Programmausführung um `Seconds` Sekunden.

Parameter

▷ **`Seconds`** (input_control) . . . . . real ⇝ *real*
Anzahl Sekunden, um die die Programmausführung ausgesetzt wird.
Defaultwert : 10
Restriktion : `Seconds` ≥ `0`

Ergebnis

`wait_seconds` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`wait_seconds` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`system_call`

Siehe auch

`system_call`, `count_seconds`

Modul

System

11.2 Datenbank

`count_relation` (: : RelationName : NumOfTuples)

Anzahl der Einträge in der HALCON-Datenbank.

`count_relation` zählt die Anzahl der Einträge in einer der vier Relationen der HALCON-Datenbank. Die HALCON-Datenbank ist folgendermaßen aufgebaut:

Es gibt zwei Basisrelationen für Regionendaten und Bildmatrizen. Aus Elementen dieser beiden Relationen werden die HALCON-Objekte Region und Bild aufgebaut: Eine Region besteht aus einem Zeiger auf ein Tupel in der Regionendatenrelation. Ein Bild besteht aus einem Zeiger auf eine Tupel in der Regionendatenrelation (wie eine Region) und zusätzlich aus einem oder mehreren Zeigern auf Tupel der Matrizrelation. Sind mehrer Zeiger vorhanden, wird ein solches Bild als mehrkanaliges Bild bezeichnet.

Regionen und Bilder werden gemeinsam als Objekte bezeichnet. Man kann eine Region als den Spezialfall eines Bildes auffassen, das keine Bildmatrizen hat. Die Tupel der Regionendatenrelation und der Bildmatrixrelation werden aus Speicherplatzgründen von mehreren Objekten gemeinsam verwendet. Es kann also z.B. mehr Bilder als Bildmatrizen geben. Relevant für den Speicherbedarf sind nur die beiden lowlevel Relationen. Bildobjekte (Regionen und Bilder) bestehen nur aus Verweisen auf Regionen- und Matrixdaten und benötigen deshalb nur einige Byte Speicher.

Mögliche Werte für `RelationName`:

'image': Bildermatrizen. Dabei kann eine Matrix durchaus Bestandteil mehrerer Bilder sein (keine redundante Speicherung).

'region':

'XLD': eXtended Line Description: Konturen, Polygone, Parallele, Linen etc. XLD Daten haben keine Grauwerte und sind subpixel genau. Regionen (immer vorhanden sind die volle und die leere Region). Dabei kann eine Region natürlich Bestandteil mehrerer Bildobjekte sein (keine redundante Speicherung).

'object': Bildobjekte. Aufgebaut aus einer Region (Region genannt) und optional Bildmatrizen (Bild genannt).

'tuple': Im Kompaktmodus werden Tupel von Bildobjekten unter einem Surrogat in dieser Relation gespeichert. Statt mit den einzelnen Objektschlüsseln wird dann nur mehr mit diesem Schlüßel gearbeitet. Es hängt von der Wirtssprache ab, ob die Objekte einzeln (Prolog und C++) oder als Tupel (C, Smalltalk, Lisp, OPS-5) übergeben werden.

Gewisse Datenbank-Objekte werden bereits von `reset_obj_db` angelegt und müßen daher immer vorhanden sein (die undefinierte Grauwertkomponente, die Objekte 'full' (FULL_REGION in HALCON/C) und 'empty' (EMPTY_REGION in HALCON/C) sowie die darin enthaltene leere und volle Region). `reset_obj_db` erscheint dementsprechend bei einem Aufruf von `get_channel_info` auch als 'creator' der vollen und leeren Region. Die Prozedur dient beispielsweise dazu, den die Vollständigkeit von `clear_obj` Operation zu überprüfen.

Parameter

▷ **`RelationName`** (input_control) . . . . . string ⇝ *string*
Interessierende Relation der HALCON-Datenbank.
Defaultwert : 'object'
Werteliste : `RelationName` ∈ {'image', 'region', 'XLD', 'object', 'tuple'}

▷ **`NumOfTuples`** (output_control) . . . . . integer ⇝ *integer*
Anzahl der Tupel in der Relation.

Beispiel

```
reset_obj_db(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.
*/
```

Ergebnis

Ist der Parameter korrekt, dann liefert `count_relation` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung ausgeführt.

Parallelisierungsinformation

`count_relation` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`reset_obj_db`

Siehe auch

`clear_obj`

Modul

System

get_modules (: : : UsedModules, ModuleKey)

Abfrage der verwendeten Module und des Modulschlüssels.

`get_modules` fragt die Modulnummern aller bis zu diesem Zeitpunkt aufgerufenen Module ab. Jeder Operator ist einem Module zugeordnet (maximal 32), deren Namen in `UsedModules` übergeben werden. Aus den verwendeten Modulen wird der Modulschlüssel erzeugt, der in `ModuleKey` übergeben wird. Dieser Schlüssel wird für den Lizenz-Manager benötigt. `get_modules` wird normalerweise am Ende eines Programms aufgerufen um alle verwendeten Module abzufragen.

Parameter

- ▷ **UsedModules** (output_control) ..... string-array ⇝ *string*
  Namen der verwendeten Module.
- ▷ **ModuleKey** (output_control) ..... integer ⇝ *integer*
  Schlüssel für Lizenz-Manager.

Parallelisierungsinformation

`get_modules` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Operators not requiring licensing

reset_obj_db (: : DefaultImageWidth, DefaultImageHeight, DefaultChannels :)

Initialisierung des HALCON-Systems.

`reset_obj_db` initialisiert das HALCON-System. Dabei legt die Prozedur die vier Relationen (Grauwertdaten, Regionendaten, Bildobjekte und ObjektTupel) an (siehe auch `count_relation`), die für die Bildverarbeitung mit HALCON nötig sind. Sind die Relationen bereits vorhanden, dann werden alle Tupel in den Relationen gelöscht!

Die `DefaultImageWidth` und `DefaultImageHeight` geben die Startwerte für das globale maximale Bildformat an. Wenn das erste erzeugte Objekt ein Bild ist (z.B. `read_image`), dann werden in (Standard-)Halcon die angegebenen Werte mit der Größe dieses Bildes überschrieben. In Parallel HALCON werden die Werte stattdessen nur dann mit der neuen Bildgröße überschrieben, falls diese die initialen Werte übersteigt. Wird statt einem Bild zuerst eine Region erzeugt, dann werden sowohl in Standard- also auch in Parallel HALCON die Werte nur verändert, wenn das neue Bild größer als der Vorgabewert ist. Dies gilt aber nicht nur bei dem ersten Bild, das erzeugt, bzw. eingelesen wird: Das globale Bildformat wird immer vergrößert wenn größere Bilder erzeugt werden.

Das globale Bildformat hat Bedeutung beim Öffnen von Fenstern (`open_window`) und beim Clipping von Regionen. Regionen werden, sofern der Clip-Modus eingeschaltet ist (`set_system('clip_region','true')`) an dem globalen Bildformat geclippt. Dies kann dann zu Problemen führen, wenn Bilder unterschiedlicher Größe verwendet werden. In diesem Fall ist nur garantiert, daß eine Region kleiner oder gleich dem Bildformat der größten Bildes ist.

Der Parameter `DefaultChannels` gibt die häufigste Anzahl von Kanälen eines Bildobjektes an. Dieser Wert kann auch auf 0 gesetzt werden, wenn viele Regionen verwendet werden. Wenn mehr Kanäle für ein Bild nötig sind, als bei der Initialisierung angegeben wurde, dann wird für dieses Bild die Anzahl dynamisch erweitert. Wenn weniger Kanäle benötigt werden als bei der Initialisierung angegeben, dann werden die überzähligen Kanäle auf

undefiniert gesetzt. Dies wirkt für den Anwender, als wären sie nicht vorhanden; es wird jedoch unnötiger Speicher allokiert.

Die Parameterwerte können mit `get_system` abgefragt werden.

Achtung

Wird `reset_obj_db` am Anfang einer HALCON-Sitzung nicht aufgerufen, dann wird HALCON automatisch mit `reset_obj_db(128,128,0)` initialisiert. Bei erneutem Aufruf von `reset_obj_db` werden alle Bildobjekte in der Datenbank gelöscht.

Parameter

- ▷ **`DefaultImageWidth`** (input_control) . . . . . integer ⇝ *integer*
  Default-Bildbreite (in Pixeln).
  Defaultwert : 128
  Wertevorschläge : `DefaultImageWidth` ∈ {64, 128, 256, 512, 525, 1024}
- ▷ **`DefaultImageHeight`** (input_control) . . . . . integer ⇝ *integer*
  Default-Bildhöhe (in Pixeln).
  Defaultwert : 128
  Wertevorschläge : `DefaultImageHeight` ∈ {64, 128, 256, 512, 768, 1024}
- ▷ **`DefaultChannels`** (input_control) . . . . . integer ⇝ *integer*
  Übliche Anzahl von Kanälen, durch die die Systemkonstante 'max_channels' beschränkt ist.
  Defaultwert : 0
  Wertevorschläge : `DefaultChannels` ∈ {0, 1, 2, 3, 4, 5, 6, 7}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `reset_obj_db` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exeptionbehandlung durchgeführt.

Parallelisierungsinformation

`reset_obj_db` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

`get_channel_info`, `count_relation`

Modul

Operators not requiring licensing

11.3 Fehlerbehandlung

`get_check` (: : : `Check`)

Status der HALCON-Kontrollmodi.

Mit `get_check` kann abgefragt werden, welche Überprüfungen gerade eingeschaltet und welche ausgeschaltet sind. `Check` liefert ein Tupel mit den Namen der Kontrollmodi (vgl. `set_check`), denen jeweils eine Tilde (~, z.B. '~data') vorangestellt ist, falls die betreffende Kontrolle ausgeschaltet ist.

Parameter

- ▷ **`Check`** (output_control) . . . . . string-array ⇝ *string*
  Tupel der aktuellen Kontrollmodi.

Ergebnis

`get_check` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`get_check` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`set_check`

Siehe auch

`set_check`

Modul

System

get_error_text (: : ErrorNumber : ErrorText)

Fehlertext zu einer HALCON-Fehlernummer abfragen.

get_error_text gibt zu einer HALCON-Fehlernummer den zugehörigen Fehlertext aus. Es handelt sich hierbei um denselben Text, der bei einer Exception-Behandlung ausgegeben wird. get_error_text ist z.B. dann nützlich, wenn die Fehler-Behandlung selbst programmiert wird (siehe: set_check(::'~give_error':)).

Achtung

Bei unbekannten Fehlernummern wird eine Standardmeldung ausgegeben.

Parameter

- ▷ **ErrorNumber** (input_control) ..... integer ⇝ *integer*
  Nummer des HALCON-Fehlers.
  Restriktion : $(1 \leq \texttt{ErrorNumber}) \wedge (\texttt{ErrorNumber} \leq 36000)$
- ▷ **ErrorText** (output_control) ..... string ⇝ *string*
  Zugehöriger Fehlertext.

Beispiel

```
Herror   err;
char       message[MAX_STRING];

set_check("~give_error");
err = send_region(region,socket_id);
set_check("give_error");
if (err != MESS_TRUE) {
get_error_text((long)err,message);
  fprintf(stderr,"my error message: %s\n",message);
  exit(1);
}
```

Ergebnis

get_error_text liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

get_error_text ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

set_check

Siehe auch

set_check

Modul

System

get_spy (: : Class : Value)

Aktuelle Konfiguration des HALCON Debugging-Tool.

get_spy liefert die aktuelle Konfiguration von spy, dem HALCON Debugging Tool. Die vorhandenen Überwachungsmodi (Wahlmöglichkeiten für Class) sowie die zugehörigen Einstellmöglichkeiten (mögliche Werte für Value) können mittels query_spy abgefragt werden. Eine detaillierte Beschreibung findet sich bei set_spy.

Parameter

- ▷ **Class** (input_control) ..... string ⇝ *string*
  Überwachungsmodus
  Defaultwert : 'mode'
  Werteliste : Class ∈ {'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) . . . . . string ⇝ *string* / integer / real
Status des Überwachungsmodus.

Ergebnis

get_spy liefert den Wert 2 (H_MSG_TRUE), falls der Parameter Class korrekt ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_spy ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

reset_obj_db

Siehe auch

set_spy, query_spy

Modul

System

query_spy (: : : Classes, Values)

Einstellmöglichkeiten des HALCON Debugging-Tool abfragen.

query_spy liefert alle Einstellmöglichkeiten von spy, dem HALCON Debugging Tool, also die vorhandenen Überwachungsmodi (Classes) und die zugehörigen Einstellmöglichkeiten (Values). Eine detaillierte Beschreibung von spy findet sich bei set_spy.

Achtung

Die Werte von Values können nicht direkt als Eingabe für set_spy dienen, da sie als symbolische Konstante übergeben werden.

Parameter

▷ **Classes** (output_control) . . . . . string-array ⇝ *string*
Vorhandene Überwachungsmodi (siehe set_spy).

▷ **Values** (output_control) . . . . . string-array ⇝ *string*
Zugehöriger Status der Überwachungsmodi.

Ergebnis

query_spy liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

query_spy ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

reset_obj_db

Siehe auch

set_spy, get_spy

Modul

System

set_check (: : Check :)

Ein- und Ausschalten von HALCON-Kontrollen.

Mit set_check kann man verschiedene automatische Kontrollen des HALCON-Systems ein- bzw. ausschalten. Ist ein bestimmter Kontrollmodus eingeschaltet, werden zur Laufzeit Überprüfungen von Parametern etc. vorgenommen. Wird dabei eine Inkonsistenz festgestellt, so wird der Programmlauf mit einer Exception-Behandlung abgebrochen. Typischerweise sollte man bei der Programmentwicklung die Kontrollen eingeschaltet lassen und sie erst nach abgeschlossener Testphase ausschalten. Ist der Kontrollmodus ausgeschaltet und tritt währenddessen ein Fehler auf, kann es zu unvorhersehbarem Verhalten des Systems kommen. Das HALCON-System verfügt über verschiedene Kontrollmöglichkeiten, die jeweils getrennt gesetzt oder ausgeschaltet werden können. Das

Einschalten erfolgt durch den Aufruf von set_check mit dem Namen (Check) der gewünschten Kontrolle; ausgeschaltet wird sie durch Angabe ihres Namens mit vorangestellter Tilde (~, z.B. '~data').

Kontrollmodi:

'color': Ist diese Kontrolle eingeschaltet, dürfen nur Farben verwendet werden, die das Display für das aktuelle Fenster unterstützt. Anderenfalls wird eine Fehlermeldung ausgegeben.
Ist diese Kontrolle ausgeschaltet, wird für nicht existente Farben eine *Näherungsfarbe* verwendet (siehe: set_color, set_gray, set_rgb).

'text': Ist diese Kontrolle eingeschaltet, werden die Koordinaten beim Setzen des Textcursors sowie bei der Ausgabe von Zeichenreihen (write_string) dahingehend überprüft, ob ein Teil eines Zeichens außerhalb des Fensters zu liegen kommt (was vom System nicht grundsätzlich verboten wird).
Ist der Modus ausgeschaltet, dann wird der Text an den Fensterrändern geclippt.

'parameter': (Nur für HALCON/PRO) Ist diese Kontrolle eingeschaltet, dürfen Ausgabeparameter beim Aufruf einer Prozedur noch nicht instantiiert sein. Sonst wird der normale Unifikationsmechanismus verwendet.

'data': (Für Programmentwicklung) Überprüfung der Konsistenz von Bildobjekten (Regionen und Grauwertkomponenten).

'interface': Ist diese Kontrolle eingeschaltet, wird die Schnittstelle zwischen Wirtssprache und den HALCON-Prozeduren zur Laufzeit überwacht (z.B. Typisierung und Anzahl der Werte).

'database': Konsistenzprüfungen in der Datenbank (z.B. existiert ein Objekt, das gelöscht werden soll, überhaupt?)

'give_error': Legt fest, ob Fehler „Exceptions" auslösen oder nicht. Ist diese Kontrolle ausgeschaltet, muß sich das Anwenderprogramm selbst um eine geeignete Fehlerbehandlung kümmern. Es sei insbesondere darauf hingewiesen, daß nicht gemeldete Fehler meist zu nicht besetzten Ausgabeparametern führen, die ein unvorhersehbares Verhalten des Programms verursachen können.

'father': Ist diese Kontrolle eingeschaltet, erlaubt HALCON als „Vater" eines neuen Fensters bei open_window nur die Nummer eines anderen HALCON-Fensters, sonst auch andere X-Window IDs.

'region': (für Programmentwicklung) Überprüfung der Konsistenz von Sehnendaten (bedeutet mitunter starke Verlangsamung der Routinen).

'clear': Soll ein Tupel von Objekten mittels clear_obj gelöscht werden, wird normalerweise eine Exception ausgelöst, wenn einzelne Objekte gar nicht (mehr) existieren. Ist 'clear' eingeschaltet, werden solche Objekte ignoriert.

'all': Schaltet alle Kontrollmodi ein.

'none': Schaltet alle Kontrollmodi aus.

'default': Voreinstellung.

Parameter

▷ **Check** (input_control) ..... string(-array) ⇝ *string*
Gewünschter Kontrollmodus.
Defaultwert : 'default'
Werteliste : Check ∈ {'color', 'text', 'database', 'data', 'interface', 'give_error', 'father', 'region', 'clear', 'all', 'none', 'default'}

Ergebnis

set_check liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_check ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Siehe auch

get_check, set_color, set_rgb, set_hsi, write_string

Modul

System

set_spy (: : Class, Value :)

Steuern des HALCON Debugging Tools.

set_spy ist das HALCON Debugging Tool. Es erlaubt die flexible Überwachung der Ein- und Ausgabedaten von HALCON-Operatoren - sowohl graphisch als auch textuell. Aktiviert wird die Datenkonrolle mittels

```
set_spy(::'mode','on':),
```

ausgeschaltet mittels

```
set_spy(::'mode','off':).
```

Außerdem kann das Debugging Tool mit der Environmentvariable HALCONSPY aktiviert werden: Wenn diese Variable definiert ist, entspricht das dem Aufruf mit **'mode'** und **'on'**.

Folgende Überwachungsmodi können (natürlich in beliebiger Kombination) über Class/Value eingestellt werden:

Class Bedeutung / Value

'operator' Bei Aufruf einer Routine werden deren Name und die Namen ihrer Parameter (in $\backslash TRIAS$ Notation) ausgegeben.
Value: 'on' oder 'off'
default: 'off'

'input_control' Bei Aufruf einer Routine werden die Namen und Werte der Eingabesteuerparameter ausgegeben.
Value: 'on' oder 'off'
default: 'off'

'output_control' Bei Aufruf einer Routine werden die Namen und Werte der Ausgabesteuerparameter ausgegeben.
Value: 'on' oder 'off'
default: 'off'

'db' Informationen über die 4 Relationen in der HALCON-Datenbank. Dies ist insbesondere bei der Suche nach vergessenen clear_obj zu suchen.
Value: 'on' oder 'off'
default: 'off'

'parameter_values' Ergänzung zu 'input_control' und 'output_control': Gibt an, wieviele Werte pro Parameter maximal ausgegeben werden sollen (maximale Tupellänge für Ausgabe).
Value: Tupellänge (integer)
default: 4

'input_gray_window' Bei jedem lesenden Zugriff auf eine Grauwertkomponente eines (Eingabe-)Bildobjektes, wird diese auf dem angegebenen Fenster ausgegeben (Window-ID; 'none' schaltet diese Überwachung aus).
Value: Window-ID (integer) oder 'none'
default: 'none'

'output_gray_window' Sobald eine Grauwertkomponente eines (Ausgabe-) Bildobjektes gesetzt wird, gibt spy diese auf dem angegebenen Fenster aus (Window-ID; 'none' schaltet diese Überwachung aus).
Value: Window-ID (integer) oder 'none'
default: 'none'

'input_region_window' Bei jedem lesenden Zugriff auf die Region eines (Eingabe-)Bildobjektes wird diese auf dem angegebenen Fenster ausgegeben (Window-ID; 'none' schaltet diese Überwachung aus).
Value: Window-ID (integer) oder 'none'
default: 'none'

'output_region_window' Sobald die Region eines (Ausgabe-)Bildobjektes gesetzt wird, gibt spy diese auf dem angegebenen Fenster aus (Window-ID; 'none' schaltet diese Überwachung aus).
Value: Window-ID (integer) oder 'none'
default: 'none'

'time' Rechenzeit für die Ausführung des Operators
Value: 'on' oder 'off'
default: 'off'

'halt' Legt fest, ob nach jeder Einzelaktion ('multiple') oder nur jeweils am Operatorende ('single') gewartet werden soll. Der Parameter ist nur von Bedeutung, wenn mit 'timeout' oder 'button_window' das Warten eingeschaltet wurde.
Value: 'single' oder 'multiple'
default: 'multiple'
Value: 'single' oder 'multiple'
default: 'multiple'

'timeout' Nach jeder Ausgabe wird die angegebene Anzahl von Sekunden gewartet.
Value: Sekunden (real)
default 0.0

'button_window' Alternative zu 'timeout': Nach jeder Ausgabe wartet spy, bis mit der Maus in das angegebene Fenster gezeigt ('button_click' = 'false') oder geklickt ('button_click' = 'true') wird (Window-ID; 'none' schaltet diese Überwachung aus).
Value: Window-ID (integer) oder 'none'
default: 'none'

'button_click' Ergänzung zu 'button_window': Legt fest, ob nach einer Ausgabe auf einen Mausklick gewartet werden soll oder nicht.
Value: 'on' oder 'off'
default: 'off'

'button_notify' Ist 'button_notify' eingeschaltet, erzeugt spy nach jeder Ausgabe einen Piepston. Dies ist in Kombination mit 'button_window' sinnvoll.
Value: 'on' oder 'off'
default: 'off'

'log_file' Umleitung der Textausgabe von spy in ein File, das mit open_file geöffnet wurde.
Value: ein File Handle (siehe `open_file`)

'error' Ist 'error' eingeschaltet, gibt spy beim Auftreten eines internen Fehlers die Prozeduren (Datei/Zeilennummer) aus, bei denen der Fehler auftritt.
Value: 'on' oder 'off'
default: 'off'

'internal' Ist 'internal' eingeschaltet, gibt spy vor dem Aufruf eines internen HALCON-Operators den Prozedurname mit ihren Parametern (Datei/Zeilennummer) aus.
Value: 'on' oder 'off'
default: 'off'

Parameter

▷ **`Class`** (input_control) ..... string ⇝ *string*
Überwachungsmodus
Defaultwert : 'mode'
Werteliste : `Class` ∈ {'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 ⇝ *string* / integer / real
Einzustellender Status des Überwachungsmodus.
Defaultwert : 'on'
Wertevorschläge : `Value` ∈ {'on', 'off', 1, 2, 3, 4, 5, 10, 50, 0.0, 1.0, 2.0, 5.0, 10.0}

Beispiel

```
/* init spy: Setting of the wished control modi */
set_spy("mode","on");
set_spy("operator","on");
set_spy("input_control","on");
set_spy("output_control","on");
/* calling of program section, that will be examined */
set_spy("mode","off");
```

Ergebnis

`set_spy` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_spy` ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`reset_obj_db`

Siehe auch

`get_spy`, `query_spy`

Modul

System

11.4 Information

`disp_info` (: : `ProcName, Machine, DispProgram, Extension` :)

Ausgabe der Manualinformation einer Prozedur.

`disp_info` gibt mit dem Previewer `DispProgram` den Manualeintrag der angegebenen Prozedur auf dem Bildschirm aus. Die einzelnen Dateien (`ProcName.Extension`) befinden sich in dem HALCON-Directory „doc/ps/reference/LANGUAGE/*“. Der Wert von LANGUAGE wird durch die aktuell verwendete Wirtssprache festgelegt. Das Directory kann auch mit `set_system(::'reference_dir','Pfad':)` eingestellt werden.

Parameter

- **`ProcName`** (input_control) ..... proc_name ⇝ *string*
  Name der gesuchten Prozedur.
  Defaultwert : 'read_image'
- **`Machine`** (input_control) ..... string ⇝ *string*
  Name des Rechners auf dem die Daten ausgegeben werden sollen oder leerer String.
  Defaultwert : ''
- **`DispProgram`** (input_control) ..... string ⇝ *string*
  Name des Programms das den Hilfstext ausgeben soll.
  Defaultwert : 'ghostview'
  Wertevorschläge : `DispProgram` ∈ {'gs', 'ghostview'}
- **`Extension`** (input_control) ..... string ⇝ *string*
  Extension der Hilfsdatei.
  Defaultwert : 'ps'

Ergebnis

Sind die Parameterwerte korrekt, die Datei und das Programm verfügbar, dann liefert `disp_info` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`disp_info` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

`open_window`

Modul

Image / region / XLD management

`get_chapter_info` (: : `Chapter` : `Info`)

Informationen zu den Prozedurkapiteln.

`get_chapter_info` liefert Informationen zu den Prozedurkapiteln. Wird für `Chapter` der leere String übergeben, liefert die Routine in `Info` die Namen aller Kapitel zurück. Spezifiziert man hingegen ein bestimmtes Kapitel bzw. Kapitel + Unterkapitel (durch ein Tupel von Namen), werden die zugehörigen Unterkapitel oder - falls es keine weiteren Unterkapitel mehr gibt - die Namen der zugehörigen Prozeduren ausgegeben. Die Einteilung der Prozedurkapitel entspricht dabei der Kapitel-/ Unterkapiteleinteilung im HALCON-Manual. Hinweis: Die Prozedurkapitel bzw. -unterkapitel einer konkreten Prozedur kann mittels `get_operator_info`

(::<Name>,'chapter',Info:) abgefragt werden. Die Online-Texte werden den Dateien german.hlp, german.sta, german.num und german.idx entnommen, die HALCON im aktuellen Directory und im Directory 'help_dir' (siehe get_system und set_system) sucht.

Parameter

- **Chapter** (input_control) . . . . . string(-array) ⇝ *string*
  Interessierende Prozedurklasse bzw. -unterklasse.
  Defaultwert : ''
- **Info** (output_control) . . . . . string-array ⇝ *string*
  Prozedurklassen (Chapter = '') oder Prozedurunterklassen bzw. Prozeduren.

Ergebnis

Sind die Parameterwerte korrekt und ist die Hilfedateien verfügbar, dann liefert get_chapter_info den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_chapter_info wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

get_system, set_system

Siehe auch

get_operator_info, get_system, set_system

Modul

Image / region / XLD management

get_keywords (: : ProcName : Keywords)

Schlüßelwörter, die Prozeduren zugeordnet sind, abfragen.

get_keywords liefert alle Schlüßelwörter in den Online-Texten zu den Prozeduren, die den vorgegebenen Teilstring ProcName in ihrem Namen haben. Wird für ProcName der leere String übergeben, gibt get_keywords sämtliche Schlüßelwörter aus. Die Schlüßelwörter einer einzigen Prozedur können auch mit get_operator_info abgefragt werden. Die Online-Texte werden den Dateien **german.hlp**, **german.sta**, **german.num**, **german.key** und **german.idx** entnommen, die HALCON im aktuellen Directory und im Directory 'help_dir' (siehe get_system und set_system) sucht.

Parameter

- **ProcName** (input_control) . . . . . proc_name ⇝ *string*
  Teilstring in den Namen der Prozeduren, zu denen Schlüßelwörter benötigt werden.
  Defaultwert : 'get_keywords'
- **Keywords** (output_control) . . . . . string-array ⇝ *string*
  Schlüßelwörter zu den Prozeduren.

Ergebnis

get_keywords liefert den Wert 2 (H_MSG_TRUE), wenn die Parameter korrekt sind und die Hilfedateien zur Verfügung stehen. Sonst wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_keywords wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

get_chapter_info

Alternativen

get_operator_info

Siehe auch

get_operator_name, search_operator, get_param_info

Modul

Image / region / XLD management

get_operator_info (: : ProcName, Slot : Information)

Informationen zu einer HALCON-Prozedur.

get_operator_info dient zur Abfrage der Online-Texte zu einer vorgegebenen Prozedur (siehe auch get_operator_name). Die Art der Information (Slot), die für jede Prozedur zur Verfügung stehen, lassen sich mittels query_operator_info abfragen. Derzeit werden folgende Slots angeboten:

'short': Kurzbeschreibung der Prozedur.

'abstract': Beschreibung der Prozedur.

'procedure_class': Kapitelname(n) in der Prozedurhierarchie (Kapitel, Unterkapitel im HALCON-Manual).

'functionality': Funktionalität, ist gleich der Objektklasse, der die Prozedur zugeordnet werden kann.

'keywords': Schlüsselwörter zu der Prozedur (optional).

'example': Beispiel für die Anwendung der Prozedur (optional). Mit 'example.SPRACHE' (SPRACHE $\in$ {c,c++,smalltalk,trias}) können Beispiele für eine bestimmte Sprache abgefragt werden, insofern sie vorhanden sind. Wird die Sprache nicht mit angegeben oder ist in dieser Sprache kein Beispiel vorhandenen, wird das TRIAS-Beispiel ausgegeben.

'complexity': Komplexität der Prozedur (optional).

'effect': Momentan nicht genutzt.

'alternatives': Alternativen zur Prozedur (optional).

'see_also': Prozeduren mit weitergehenden Informationen (optional).

'predecessor': Mögliche und sinnvolle Vorgängerprozedur

'successor': Mögliche und sinnvolle Nachfolgerprozedur

'result_state': Rückgabewert der Prozedur (TRUE, FALSE, FAIL, VOID oder EXCEPTION).

'attention': Einschränkungen und Hinweise zur korrekten Anwendung der Prozedur (optional).

'parameter': Namen der Parameter der Prozedur (siehe get_param_info).

'references': Literaturverweise (optional).

'module': Das Modul, zu dem der Operator gehört.

'html_path': Das Verzeichnis, in dem die HTML-Dokumentation des Operators steht.

'parameter_relations': Zusicherungen zu den Parametern. (optional).

Die Texte werden den Dateien german.hlp, german.sta, german.key, german.num und german.idx entnommen, die HALCON im aktuellen Directory und im Directory 'help_dir' (bzw. 'user_help_dir') (siehe get_system und set_system) sucht. Bei Slots mit textuellen Informationen kann mit '.latex' nach dem Slotnamen der Text in LaTeX Notation geholt werden.

Parameter

- ▷ **ProcName** (input_control) ..... proc_name ⇝ *string*
  Name der Prozedur, zu der Information benötigt werden.
  Defaultwert : 'get_operator_info'
- ▷ **Slot** (input_control) ..... string ⇝ *string*
  Gewünschte Information.
  Defaultwert : 'abstract'
  Werteliste : Slot $\in$ {'short', 'abstract', 'procedure_class', 'functionality', 'effect', 'complexity', 'predecessor', 'successor', 'alternatives', 'see_also', 'keywords', 'example', 'attention', 'result_state', 'return_value', 'references', 'sourcefiles', 'deffile', 'module', 'html_path'}
- ▷ **Information** (output_control) ..... string(-array) ⇝ *string*
  Information (leer, falls keine Information vorhanden ist)

Ergebnis

get_operator_info liefert den Wert 2 (H_MSG_TRUE), wenn die Parameter korrekt sind und die Hilfsdateien verfügbar sind. Sonst wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_operator_info wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

get_keywords, search_operator, get_operator_name, query_operator_info, query_param_info, get_param_info

Mögliche Nachfolgerfunktionen

get_param_names, get_param_num, get_param_types

Alternativen

get_param_names

Siehe auch

query_operator_info, get_param_info, get_operator_name, get_param_num, get_param_types

Modul

Image / region / XLD management

get_operator_name (: : Pattern : ProcNames)

Prozeduren, die eine Zeichenreihe als Teilstring im Namen haben.

get_operator_name nimmt eine Zeichenreihe (Pattern) als Eingabe und sucht alle HALCON-Prozeduren, die diese Zeichenreihe als Teilstring im Namen enthalten. Bei Eingabe des leeren Strings werden die Namen aller verfügbaren Prozeduren ausgegeben.

Parameter

- ▷ **Pattern** (input_control) ..... string ⇝ *string*
  Teilstring des gesuchten Namens (leer <=> alle Namen).
  Defaultwert : 'info'
- ▷ **ProcNames** (output_control) ..... string-array ⇝ *string*
  Gefundene Prozedurnamen.

Ergebnis

get_operator_name liefert den Wert 2 (H_MSG_TRUE) falls die Hilfedateien verfügbar sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_operator_name ist *wiedereintrittsfähig* („reentrant"), *lokal* auszuführen („local") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

get_operator_info, get_param_names, get_param_num, get_param_types

Alternativen

search_operator

Siehe auch

get_operator_info, get_param_names, get_param_num, get_param_types

Modul

Image / region / XLD management

get_param_info (: : ProcName, ParamName, Slot : Information)

Informationen zu Prozedurparametern.

get_param_info dient zur Abfrage der Online-Texte zu einem Parameter einer vorgegebenen Prozedur. Die Art der Information (Slot), die für jeden Parameter zur Verfügung steht, läßt sich mittels query_param_info abfragen. Derzeit werden folgende Slots angeboten:

'description': Beschreibung des Parameters.

'description.latex': Beschreibung des Parameters in LaTeX Notation.

'parameter_class': Parameterklasse: 'input_object', 'output_object', 'input_control' oder 'output_control'.

'type_list': Zulässige(r) Datentyp(en) für Parameterwerte (nur bei Steuerparametern). Werte: 'real', 'integer' oder 'string'.

'default_type': Default-Typ für Parameterwerte (nur bei Steuerparametern). Dies ist der Parametertyp der bei HALCON/C im „simple Mode" verwendet wird. Falls 'none' angegeben wird, dann muß der „Tupel Modus" verwendet werden. Werte: 'real', 'integer', 'string' oder 'none'.

'sem_type': Semantischer Typ des Parameters. Das ist wichtig für die Zuordnung der Parameter zu Objektklassen in objektorientierten Sprachen (C++, Smalltalk). Gehören mehrere Parameter semantisch zu einem Typ, so wird dies dabei ebenfalls bezeichnet. Bisher werden die folgenden Objekte Unterstützt:
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-Wert für den Parameter (nur bei Eingabe-Steuerparametern). Hierbei handelt es sich um eine reine Information (der Parameterwert muß in jedem Fall explizit übergeben werden, auch wenn der default-Wert verwendet werden soll). Dieser Eintrag dient nur als Hinweis, als Ausgangspunkt für eigene Experimente. Die Werte sind so gewählt, daß sie normalerweise keine Fehler verursachen und etwas sinnvolles bewirken.

'multi_value': 'true', falls an dieser Parameterposition mehrere Werte, also Tupel von Werten stehen dürfen, sonst 'false'.

'multichannel': 'true', falls ein Eingabebildobjekt mehrkanalig sein darf.

'mixed_type': Nur bei Steuerparametern und auch nur dann, wenn Wertetupel ('multivalue'-'true') und verschiedene Datentypen für Parameterwerte ('type_list' mehr als ein Wert) zulässig sind. In diesem Fall gibt `Slot` an, ob in einem Tupel auch Werte verschiedener Typen gemischt auftreten dürfen ('true' oder 'false').

'values': Werteauswahl (optional).

'value_list': Falls ein Parameter nur eine begrenzte Anzahl von Werten annehmen kann, werden diese explizit aufgeführt (optional).

'valuemin': Minimalwert eines Werteintervalls.

'valuemax': Maximalwert eines Werteintervalls.

'valuefunction': Funktion des Werteverlaufs einer Testreihe (lin, log, quadr, ...).

'steprec': Empfohlene Schrittweite der Parameterwerte in einer Testreihe.

'steprec': Minimale Schrittweite der Parameterwerte in einer Testreihe.

'valuenumber': Aussage über die Anzahl der Parameter absolut oder relativ zu anderen Parametern

'assertion': Aussage über die Parameterwerte absolut oder relativ zu anderen Parametern.

Die Online-Texte werden den Dateien Halcon.hlp, Halcon.sta und Halcon.idx entnommen, die HALCON im aktuellen Directory und im Directory 'help_dir' (siehe `get_system` und `set_system`) sucht.

Parameter

▷ **`ProcName`** (input_control) . . . . . proc_name ⇝ *string*
Name der Prozedur, zu deren Parametern Information benötigt wird.
Defaultwert : 'get_param_info'

▷ **`ParamName`** (input_control) . . . . . string ⇝ *string*
Name des Parameters, zu dem Information benötigt wird.
Defaultwert : 'Slot'

▷ **Slot** (input_control) ..... string ⇝ *string*
Gewünschte Information.
Defaultwert : 'description'
Werteliste : 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) ⇝ *string*
Information (leer, falls keine Informationen vorhanden sind).

Ergebnis

The operator get_param_info returns the value 2 (H_MSG_TRUE) if the parameters are correct and the helpfiles are available. Otherwise an exception handling is raised.

Parallelisierungsinformation

get_param_info wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

get_keywords, search_operator

Alternativen

get_param_names, get_param_num, get_param_types

Siehe auch

query_param_info, get_operator_info, get_operator_name

Modul

Image / region / XLD management

get_param_names (: : ProcName : InpObjPar, OutpObjPar, InpCtrlPar, OutpCtrlPar)

Zugriff auf die Parameternamen einer HALCON-Prozedur.

get_param_names liefert für die in ProcName angegebene HALCON-Prozedur die Namen der Eingabeobjekte, der Ausgabeobjekte, der Eingabesteuerparameter sowie der Ausgabesteuerparameter.

Parameter

▷ **ProcName** (input_control) ..... proc_name ⇝ *string*
Name der Prozedur.
Defaultwert : 'get_param_names'

▷ **InpObjPar** (output_control) ..... string-array ⇝ *string*
Namen der Eingabeobjekte.

▷ **OutpObjPar** (output_control) ..... string-array ⇝ *string*
Namen der Ausgabeobjekte.

▷ **InpCtrlPar** (output_control) ..... string-array ⇝ *string*
Namen der Eingabesteuerparameter.

▷ **OutpCtrlPar** (output_control) ..... string-array ⇝ *string*
Namen der Ausgabesteuerparameter.

Ergebnis

get_param_names liefert den Wert 2 (H_MSG_TRUE), falls der Prozedurname existiert und die Hilfedateien verfügbar sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_param_names ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_keywords, search_operator, get_operator_name, get_operator_info

Mögliche Nachfolgerfunktionen

get_param_num, get_param_types

Alternativen

get_operator_info, get_param_info

Siehe auch

get_param_num, get_param_types, get_operator_name

Modul

Image / region / XLD management

get_param_num (: : ProcName : CName, InpObjPar, OutpObjPar, InpCtrlPar, OutpCtrlPar, Type)

Stelligkeit einer HALCON-Prozedur.

get_param_num bestimmt für die angegebene HALCON-Prozedur die Stelligkeit der Ein- und Ausgabeobjektparameter sowie der Ein- und Ausgabesteuerparameter. Ausserdem erhält man den Namen der C-Funktion (CName), die von der Prozedur aufgerufen wird. Der Ausgabeparameter Type gibt an, ob es sich um eine System- oder eine Benutzer-Prozedur handelt.

Parameter

- **ProcName** (input_control) ..... proc_name ⇝ *string*
  Name der Prozedur.
  Defaultwert : 'get_param_num'
- **CName** (output_control) ..... string ⇝ *string*
  Name der aufgerufenen C-Funktion.
- **InpObjPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Eingabeobjektparameter.
- **OutpObjPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Ausgabeobjektparameter.
- **InpCtrlPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Eingabesteuerparameter.
- **OutpCtrlPar** (output_control) ..... integer ⇝ *integer*
  Anzahl der Ausgabesteuerparameter.
- **Type** (output_control) ..... string ⇝ *string*
  Systemprozedur oder Benutzerprozedur.
  Wertevorschläge : Type ∈ {'system', 'user'}

Ergebnis

get_param_num liefert den Wert 2 (H_MSG_TRUE), falls der Prozedurname existiert. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_param_num ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_keywords, search_operator, get_operator_name, get_operator_info

Mögliche Nachfolgerfunktionen

get_param_types

Alternativen

get_operator_info, get_param_info

Siehe auch

get_param_names, get_param_types, get_operator_name

Modul

Image / region / XLD management

get_param_types (: : ProcName : InpCtrlParType, OutpCtrlParType)

Default-Datentyp für die Steuerparameter einer HALCON-Prozedur.

get_param_types gibt für jeden Eingabe- und jeden Ausgabesteuerparameter dessen Default-Datentyp an. Ein Default-Typ eines Parameters ist der Typ, der in HALCON/C in „simple Mode" verwendet wird. Dies ist für

Parameter relevant, die mehr als einen Typ zulassen wie z.B. `write_string`. Dabei werden die Typen der Eingabeparameter in der Variablen `InpCtrlParType` zusammengefaßt, die Typen der Ausgabeparameter in der Variablen `OutpCtrlParType`. Mögliche Werte der Typen sind:

'integer': Eine ganze Zahl.

'integer tuple': Eine ganze Zahl oder Tupel ganzer Zahlen.

'real': Eine Gleitpunktzahl.

'real tuple': Eine Gleitpunktzahl oder Tupel von Gleitpunktzahlen.

'string': Eine Zeichenreihe.

'string tuple': Eine Zeichenreihe oder Tupel von Zeichenreihen.

'no_default': Einzelner Wert, dessen Typ nicht festgelegt werden kann.

'no_default tuple': Einzelner Wert oder Tupel von Werten, deren Typ nicht festgelegt werden kann.

'default': Einzelner Wert mit unbekanntem Typ, System nimmt 'integer' an.

Parameter

- **`ProcName`** (input_control) ..... proc_name ⇝ *string*
  Name der Prozedur.
  Defaultwert : 'get_param_types'
- **`InpCtrlParType`** (output_control) ..... string-array ⇝ *string*
  Default-Typen der Eingabesteuerparameter.
- **`OutpCtrlParType`** (output_control) ..... string-array ⇝ *string*
  Default-Typen der Ausgabesteuerparameter.

Ergebnis

The operator `get_param_types` returns the value 2 (H_MSG_TRUE) if the indicated procedure name exists. Otherwise an exception handling is raised.

Parallelisierungsinformation

`get_param_types` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`get_keywords`, `search_operator`, `get_operator_name`, `get_operator_info`

Alternativen

`get_param_info`

Siehe auch

`get_param_names`, `get_param_num`, `get_operator_info`, `get_operator_name`

Modul

Image / region / XLD management

`query_operator_info ( : : : Slots )`

Slots der Informationen für `get_operator_info`.

`query_operator_info` liefert die Namen der Online-Texte (`Slots`), die für jede Prozedur online zur Verfügung stehen. Die Informationen selbst können mit

```
get_operator_info(::<ProcName>,<Slot>:<Information>)
```

abgefragt werden.

Parameter

- **`Slots`** (output_control) ..... string-array ⇝ *string*
  Namen der Slots von `get_operator_info`.

Ergebnis

`query_operator_info` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`query_operator_info` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Nachfolgerfunktionen

get_operator_info

Siehe auch

get_operator_info

Modul

Image / region / XLD management

query_param_info (: : : Slots)

Slots der Online-Informationen für get_param_info.

query_param_info liefert die Namen der Informationen (Slots), die für jeden Parameter einer Prozedur online zur Verfügung stehen (Online-Texte). Die Online-Texte selbst können mit

```
get_param_info(::<Procedurname>,<Parametername>,<Slot>:<Information>)
```

abgefragt werden.

Parameter

- **Slots** (output_control) . . . . . string-array ⇝ *string*
  Namen der Slots bei get_param_info.

Ergebnis

query_param_info liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

query_param_info ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

get_param_info

Siehe auch

get_param_info

Modul

Image / region / XLD management

search_operator (: : Keyword : ProcNames)

Namen aller Prozeduren zu einem Schlüßelwort.

search_operator liefert die Namen aller Prozeduren, deren Online-Texte das Schlüßelwort Keyword enthalten (siehe get_operator_info). Alle verfügbaren Schlüßelwörter liefert der Aufruf

```
get_keywords(::'':  <Schlüßelwörter>)
```

.

Die Online-Texte werden den Dateien german.hlp, german.sta, german.key, german.num und german.idx entnommen, die HALCON im aktuellen Directory und im Directory 'help_dir' (siehe (siehe get_system und get_system) sucht.

Parameter

- **Keyword** (input_control) . . . . . string ⇝ *string*
  Schlüßelwort, zu dem Prozeduren gesucht werden sollen.
  Defaultwert : 'Information'
- **ProcNames** (output_control) . . . . . string-array ⇝ *string*
  Prozeduren, deren Slot 'keyword' das Schlüßelwort enthalten.

Ergebnis

search_operator liefert den Wert 2 (H_MSG_TRUE), wenn die Parameter korrekt sind und die Hilfsdateien verfügbar sind. Sonst wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

search_operator wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

get_keywords

Siehe auch

get_keywords, get_operator_info, get_param_info

Modul

Image / region / XLD management

11.5 Parallelisierung

check_par_hw_potential (: : AllInpPars :)

Überprüfen der Hardware bezüglich des Potentials zur Parallelverarbeitung.

check_par_hw_potential wird verwendet, um die automatische Parallelisierung von HALCON-Operatoren zu unterstützen. HALCON verwendet intern einen Parallelisierungsmechanismus, um Mehrprozessorarchitekturen besser zu nutzen und die Bearbeitungsdauer von Operatoren zu verkürzen. Da die Parallelisierung automatisch geschieht, muss der Anwender programmtechnisch keine Vorarbeiten für die Parallelverarbeitung leisten. Das bedeutet insbesondere, dass alle Programme, die auf HALCON-Operatoren basieren, direkt bzw. unverändert auf Parallel-Hardware verwendet werden können und trotzdem deren erweitertes Potential mittels Parallelverarbeitung nutzen können. check_par_hw_potential untersucht eine gegebene Hardware auf ihre Eignung zur parallelen Bearbeitung von HALCON Operatoren. Hierfür wird jeder Operator, der sich prinzipiell für eine parallele Bearbeitung im Zuge der automatischen Paralleliserung eignet, sowohl sequentiell als auch parallel mehrfach bearbeitet, wobei die Eingabeparameter in mehreren Durchläufen verschieden belegt werden, um zum Beispiel Abhängigkeiten zwischen der Größe von Eingabebildern und der Rentabilität der Parallelverarbeitung zu testen. Hierbei wird der Eingabeparameter AllInpPars berücksichtigt: Normalerweise, das heißt, wenn AllInpPars mit dem Standardwert 0 (“false”) belegt ist, werden ausschließlich Abhängigkeiten zu bestimmten Eingabeparametern untersucht, bei denen eine Abhängigkeit zur Bearbeitungsdauer “wahrscheinlich” ist. Alle anderen Eingabeparameter bleiben unberücksichtigt, was den gesamten Vorgang merklich beschleunigt. In seltenen Fällen kann es jedoch vorkommen, dass ein Operator für eine neue HALCON-Release intern umgeschrieben wurde. Dann kann es sein, dass ein Steuerparameter, der in den bisherigen HALCON-Versionen keine direkte Abhängigkeit zur Operatorbearbeitungsdauer aufgewiesen hat, nun solch eine Abhängigkeit besitzt. In diesem Fall, lohnt es sich, den AllInpPars auf 1 (“true”) zu setzen, um die Betrachtung aller Eingabeparameter zu erzwingen. Im Zweifel findet sich ein entsprechender Hinweis in den “Release Notes” einer HALCON-Release. Insgesamt werden innerhalb der Testläufe von check_par_hw_potential viele einzelne Rechner-spezifische Informationen gesammelt, die es HALCON ermöglichen, Operatoren optimal parallel zu bearbeiten. Die gesammelten Daten werden resident gespeichert und stehen allen weiteren HALCON-Programmläufen zur Verfügung. check_par_hw_potential muss also nur ein einziges Mal auf jedem Mehrprozessorrechner, auf dem die automatische Parallelisierung genutzt werden soll, ausgeführt werden. Natürlich sollte check_par_hw_potential erneut aufgerufen werden, sobald ein Rechner zum Beispiel durch den Einbau einer neuen CPU verändert wird, oder sobald das Betriebssystem des Rechners ausgetauscht wird. Dies gilt auch, falls sich der Rechnername (“host name”) ändert, da HALCON die rechner-spezifische Information anhand des “host names” identifiziert. Falls derselbe Mehrprozessorrechner unter verschiedenen Betriebssystemen (z.B. Windows und Linux) genutzt wird, sollte check_par_hw_potential jeweils einmal für jedes Betriebssystem ausgeführt werden, um auch den starken Einfluss des Betriebssystems auf die Nutzung paralleler Hardware korrekt messen zu können. Unter Windows speichert HALCON das zum jeweiligen Rechner gehörende Parallelisierungswissen in dessen Registry und zwar in einem Rechner-spezifischen Registry-Schlüssel, der von verschiedenen Benutzern gleichermaßen benutzt werden kann. Unter Windows NT kann der betreffende Registry-Schlüssel im Normalfall durch jeden Benutzer verändert werden. Bei Windows 2000 ist dies jedoch normalerweise nur Benutzern erlaubt, die Administratorrechte besitzen oder zumindest der Gruppe “Hauptbenutzer” angehören. Bei allen anderen Benutzern zeigt der Aufruf von check_par_hw_potential keine Auswirkung (liefert jedoch auch keinen Fehler). Bei Linux-/UNIX-Systemen wird das Parallelisierungswissen in einer Datei direkt im HALCON-Installationsverzeichnis ($HALCONROOT) abgespeichert. Folglich sollte auch hier der Operator check_par_hw_potential durch einen entsprechend autorisierten Benutzer ausgeführt werden, d.h. durch einen Benutzer, der die Schreibrechte auf das besagte Verzeichnis besitzt. Wird HALCON unter Linux/UNIX in einem Netzwerk verwendet, so werden in der Datei die Daten für alle Rechner, auf denen ein Hardware-Check ausgeführt wurde, gesammelt.

Achtung

Da check_par_hw_potential zu Testzwecken jeden zu parallelisierenden Operator mehrfach aufrufen und

bearbeiten muss, kann die Ausführungsdauer von `check_par_hw_potential` verhältnismäßig lange dauern. `check_par_hw_potential` verwendet die automatische Parallelisierung von Operatoren, die ausschließlich durch Parallel HALCON unterstützt wird. Deshalb liefert `check_par_hw_potential` in einer nicht parallelen HALCON-Version stets einen entsprechenden Fehler zurück. Der Operator `check_par_hw_potential` muss durch entsprechend autorisierte Benutzer aufgerufen werden, damit die gesammelten Informationen dauerhaft gespeichert werden können (Details hierzu finden sich in der obigen Operatorbeschreibung).

Parameter

▷ **`AllInpPars`** (input_control) . . . . . integer ⇝ *integer*
Sämtliche Eingabeparameter untersuchen?
Defaultwert : 0
Werteliste : `AllInpPars` ∈ {0, 1}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `check_par_hw_potential` den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`check_par_hw_potential` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`store_par_knowledge`

Siehe auch

`store_par_knowledge`, `load_par_knowledge`

Modul

System

`load_par_knowledge ( : : FileName : )`

Lade das Wissen zur automatischen Parallelisierung aus Datei.

`load_par_knowledge` wird verwendet, um die automatische Parallelisierung von HALCON-Operatoren zu unterstützen. HALCON verwendet intern einen Parallelisierungsmechanismus, um Mehrprozessorarchitekturen besser zu nutzen und die Bearbeitungsdauer von Operatoren zu verkürzen. Um diese Parallelisierung automatisch durchführen zu können, benötigt HALCON spezielles Wissen über die verwendete Hardware bzw. über deren Potential, die verschiedenen HALCON-Operatoren parallel zu verarbeiten. Dieses Wissen ist Rechner-spezifisch und kann mit dem Operator `check_par_hw_potential` trainiert bzw. erlernt werden. HALCON speichert dieses Wissen in der “Registry” (unter Windows) bzw. in einer Datei im HALCON-Installationsverzeichnis (unter Linux/UNIX), so dass das einmal erworbene Wissen bei jeder weiteren Verwendung von HALCON auf demselben Rechner zur Verfügung steht. `load_par_knowledge` ermöglicht es nun, dieses intern in HALCON verwendete Wissen aus einer speziellen ASCII-Datei einzulesen, die zuvor mit dem Operator `store_par_knowledge` geschrieben wurde. Der Name der Datei (inkl. Pfad und “Extension”) muss beim Aufruf von `load_par_knowledge` in `FileName` übergeben werden. Beim Einlesen der Datei wird überprüft, ob die Informationen in der Datei auf dem gegenwärtig verwendeten Rechner erzeugt wurden und ob sie für die gegenwärtig verwendete HALCON Version (und Revision) gelten. Ist dies der Fall, wird die Information übernommen und resident gespeichert. Anderenfalls wird die Information ignoriert und eine entsprechende Fehlermeldung zurückgegeben.

Parameter

▷ **`FileName`** (input_control) . . . . . filename.named ⇝ *string*
Name der Datei des Parallelisierungswissens.
Defaultwert : ”

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `load_par_knowledge` den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`load_par_knowledge` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`store_par_knowledge`

Siehe auch

store_par_knowledge, check_par_hw_potential

Modul

System

store_par_knowledge (: : FileName :)

Abspeichern des Wissens zur automatischen Parallelisierung in Datei.

store_par_knowledge wird verwendet, um die automatische Parallelisierung von HALCON-Operatoren zu unterstützen. HALCON verwendet intern einen Parallelisierungsmechanismus, um Mehrprozessorarchitekturen besser zu nutzen und die Bearbeitungsdauer von Operatoren zu verkürzen. Um diese Parallelisierung automatisch durchführen zu können, benötigt HALCON spezielles Wissen über die verwendete Hardware bzw. über deren Potential, die verschiedenen HALCON-Operatoren parallel zu verarbeiten. Dieses Wissen ist Rechner-spezifisch und kann mit dem Operator check_par_hw_potential trainiert bzw. erlernt werden. HALCON speichert dieses Wissen in der "Registry" (unter Windows) bzw. in einer Datei im HALCON-Installationsverzeichnis (unter Linux/UNIX), so dass das einmal erworbene Wissen bei jeder weiteren Verwendung von HALCON auf demselben Rechner zur Verfügung steht. store_par_knowledge ermöglicht es nun, dieses intern in HALCON verwendete Wissen als ASCII-Datei abzuspeichern. Der Name der Datei (inkl. Pfad und "Extension") muss beim Aufruf von store_par_knowledge in FileName übergeben werden. Diese Datei kann dann später mit dem Operator load_par_knowledge wieder eingelesen werden.

Parameter

▷ **FileName** (input_control) . . . . . filename.named ⇝ *string*
Name der Datei für das Parallelisierungswissen.
Defaultwert : "

Ergebnis

Sind die Parameterwerte korrekt, dann liefert store_par_knowledge den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

store_par_knowledge ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

check_par_hw_potential

Mögliche Nachfolgerfunktionen

load_par_knowledge

Siehe auch

load_par_knowledge, check_par_hw_potential

Modul

System

11.6 Parameter

get_system (: : Query : Information)

Informationen über die aktuellen HALCON-Systemparameter.

get_system gibt Informationen über die aktuellen HALCON Systemparameter aus. Einige dieser Parameter können mit set_system dynamisch geändert werden (im folgenden mit + gekennzeichnet). Wird bei Query der String '?' übergeben, erhält man in Information die Namen aller Systemparameter.

Folgende Systemparameter können abgefragt werden:

Versionen

'version': Versionsnummer von HALCON, z.B.: 6.0

'last_update': Datum der Erzeugung der HALCON-Bibliothek

'revison': Unterversion von HALCON, z.B.: 1

Obergrenzen

'max_contour_length': maximale Anzahl von Kontur- bzw. Polygonstützpunkten einer Region.

'max_images': maximale Gesamtzahl von Bildern.

'max_channels': maximale Anzahl von Kanälen eines Bildes

'max_obj_per_par': maximale Anzahl von Bildobjekten, die bei einem Aufruf pro Parameter verwendet werden dürfen.

'max_inp_obj_par': Maximale Anzahl von Eingabeparametern.

'max_outp_obj_par': Maximale Anzahl von Ausgabeparametern.

'max_inp_ctrl_par': Maximale Anzahl von Eingabe-Steuerparametern.

'max_outp_ctrl_par': Maximale Anzahl von Ausgabe-Objektparametern.

'max_window': Maximale Anzahl von Fenstern.

'max_window_types': Maximale Anzahl von Window-Systemen.

'max_proc': Maximale Anzahl von HALCON-Prozeduren (System + benutzerdefiniert).

Grafik

+'flush_graphic': Festlegung, ob Flushoperation nach jeder HALCON-Prozedur ein oder ausgeschaltet ist.

+'int2_bits': Anzahl signifikanter Bits bei int2 Bildern.

+'int_zooming': Zoomen von Bildern mit Integer- oder mit Floatingpoint-Arithmetik

+'draw_mode': (keine Beschreibung vorhanden)

'ignore_colormap': (keine Beschreibung vorhanden)

+'backing_store': Sichern der Fensterinhalte bei Überlappung.

+'icon_name': (keine Beschreibung vorhanden)

+'window_name': (keine Beschreibung vorhanden)

+'default_font': Name des Fonts der beim Öffnen eines Fensters gesetzt wird.

+'single_lut': (keine Beschreibung vorhanden)

+'update_lut': (keine Beschreibung vorhanden)

+'x_package': Anzahl Bytes die pro Datentransfer an den X-Server übertragen werden.

+'num_gray_4': Anzahl der unter X-Window reservierten Farben für die Ausgabe von Graustufen (disp_image) auf einer Maschine mit 4 Bitplanes (16 Farben).

+'num_gray_6': Anzahl der unter X-Window reservierten Farben für die Ausgabe von Graustufen (disp_image) auf einer Maschine mit 6 Bitplanes (64 Farben).

+'num_gray_8': Anzahl der unter X-Window reservierten Farben für die Ausgabe von Graustufen (disp_image) auf einer Maschine mit 8 Bitplanes (256 Farben).

+'num_gray_percentage': HALCON reserviert unter X-Window einen Teil der verfügbaren Farben für die Darstellung von Graustufen (disp_image). Dabei wird versucht, andere X-Applikationen möglichst wenig zu beeinträchtigen. Gelingt es HALCON allerdings nicht, mindestens 'num_gray_percentage' Prozent der benötigten Farben vom X-Server zu erhalten, wird einfach ein bestimmter Teil der Lookup-Table ohne Rücksicht auf Verluste für die HALCON Graustufen reserviert.
In dem Fall kann es zu unerwünschten Farbverschiebungen kommen, wenn zwischen HALCON Fenstern und den Fenstern anderer Applikationen gewechselt oder (außerhalb von HALCON) ein Window-Dump erzeugt wird. Die Zahl der zu reservierenden realen Graustufen ist abhängig von der Anzahl der Bitplanes auf der Ausgabemaschine (siehe 'num_gray_*'). Auf S/W-Maschinen werden naturgemäß keine Farben reserviert - Graustufen werden vielmehr bei der Ausgabe 'gedithert'. Maschinen mit 2 oder 3 Bitplanes werden wie S/W-Maschinen behandelt, Maschinen mit 5 (7) Bitplanes wie Maschinen mit 4 (6) Bitplanes und Maschinen mit mehr als 8 Bitplanes wie Maschinen mit 8 Bitplanes. Eine Sonderstellung nehmen Maschinen mit 24 Bit Display (Echtfarbenmaschinen) ein. Graustufenausgabe.
Hinweis: Vor dem Öffnen des ersten Fensters auf einer Maschine mit x Bitplanes enthält num_gray_x die Anzahl der zu reservierenden Farben für die Graustufenausgabe, nach dem Öffnen des ersten Fensters hingegen die Anzahl der tatsächlich reservierten Farben.

+'num_gray_percentage': (keine Beschreibung vorhanden)

+'num_graphic_2': Anzahl der unter X-Window reservierten HALCON Graphikfarben (für `disp_region` etc.) auf einer Maschine mit 2 Bitplanes (4 Farben).

+'num_graphic_4': Anzahl der unter X-Window reservierten HALCON Graphikfarben (für `disp_region` etc.) auf einer Maschine mit 4 Bitplanes (16 Farben).

+'num_graphic_6': Anzahl der unter X-Window reservierten HALCON Graphikfarben (für `disp_region` etc.) auf einer Maschine mit 6 Bitplanes (64 Farben).

+'num_graphic_8': Anzahl der unter X-Window reservierten HALCON Graphikfarben (für `disp_region` etc.) auf einer Maschine mit 8 Bitplanes (256 Farben).

Bildverarbeitung

+'neighborhood': Verwendung der 4er oder 8er Nachbarschaft.

+'only_lines': (keine Beschreibung vorhanden)

+'init_new_image': Initialisierung von Bildern vor der Anwendung von Grauwerttransformationen.

+'no_object_result': Verhalten bei leeren Objektlisten.

+'empty_region_result': Verhalten von Prozeduren bei Eingabeobjekten mit leeren Regionen, die für solche Objekte eigentlich nicht sinnvoll sind (z.B. diverse Regionenmerkmale, Segmentation, etc.). Mögliche Rückgabewerte:
'true': Der Fehler wird so gut es geht übergangen
'false': Prozedur liefert FALSE
'fail': Prozedur liefert FAIL
'void': Prozedur liefert VOID
'exception': Fehlerbehandlung

+'store_empty_region': Abspeichern von Objekten mit leerer Region.

+'clip_region': Clippen von Ausgaberegionen auf das globale Bildformat.

'width': Maximale Bildbreite - in Standard-HALCON enthält dieser Wert die maximale Bildbreite aller HALCON-Bildobjekte, die sich momentan im Speicher befinden. In Parallel HALCON enthält dieser Wert die maximale Bildbreite aller HALCON-Bildobjekte, die sich seit dem Start der aktuellen HALCON-Anwendung im Speicher befinden oder befunden haben (auch wenn sie zwischenzeitlich schon gelöscht wurden).

'height': Maximale Bildhöhe - in Standard-HALCON enthält dieser Wert die maximale Bildhöhe aller HALCON-Bildobjekte, die sich momentan im Speicher befinden. In Parallel HALCON enthält dieser Wert die maximale Bildhöhe aller HALCON-Bildobjekte, die sich seit dem Start der aktuellen HALCON-Anwendung im Speicher befinden oder befunden haben (auch wenn sie zwischenzeitlich schon gelöscht wurden).

'obj_images': aktuelle Anzahl von Grauwertkomponenten pro Bildobjekt.

+'current_runlength_number': Aktuell verwendete Anzahl von Sehnen, die zur Kodierung von Regionen verwendet werden können.

Parallelisierung

+'parallelize_operators': Gibt an, ob Parallel HALCON eine automatische Parallelisierung einsetzt, um die Bearbeitung von Operatoren auf Mehrprozessorrechnern zu beschleunigen.

+'reentrant': Zeigt an, ob Parallel HALCON zum gegenwärtigen Zeitpunkt reentrente Operatorzugriffe unterstützt oder nicht. Reentrente Zugriffe sind für die automatische Parallelisierung in Parallel HALCON notwendig.

'processor_num': Gibt die Anzahl der Prozessoren an, die von Parallel HALCON in der verwendeten Hardware gefunden wurde. Dies ist gleichzeitig auch die Prozessorenzahl, die als Grundlage der automatischen Parallelisierung dient.

Datei

'flush_file': Pufferung der Dateiausgabe.

Verzeichnisse

+'image_dir': Verzeichnispfade, in denen nach Bilddatei gesucht wird (siehe: `read_image`).

+'lut_dir': Defaultverzeichnispfad für Farbtabellen (siehe: `set_lut`).

+'reference_dir': Verzeichnispfad für die Postscript-Dokumentation von HALCON.

+'help_dir': Defaultverzeichnispfad mit den help-Dateien für die Online-Hilfe: {german,english}.{hlp,sta,idx,num,key}.

Sonstiges

+'do_low_error': Flag, ob low-level-Fehlermeldungen ausgegeben werden sollen.

'num_proc': Gesamtzahl der verfügbaren HALCON-Prozeduren ('num_sys_proc' + 'num_user_proc').

'num_sys_proc': Anahl der Systemprozeduren (unterstützte Prozeduren).

'num_user_proc': Anzahl der benutzerdefinierten Prozeduren (siehe Handbuch 'Extension Packages').

'byte_order': Byte-Order des Prozessors (**'msb_first'** oder **'lsb_first'**).

'operating_system': Name des Betriebssystems des Rechners, auf dem der HALCON-Prozeß ausgeführt wird.

'operating_system_version': Versionsnummer des Betriebssystems des Rechners, auf dem der HALCON-Prozeß ausgeführt wird.

+'max_connection': Maximale Anzahl von Regionen, die von `connection` zurückgeliefert wird.

+'alloctmp_single_block': Flag für Art der temporären Speicherverwaltung.

+'alloctmp_max_blocksize': Maximale Größe zu allokierender Speicherblöcke innerhalb temporärer Speicherverwaltung. (Ohne Effekt, falls 'alloctmp_max_blocksize' == -1 oder 'alloctmp_single_block' == 'true')

+'extern_alloc_funct': Pointer auf externe Funktion zur Speicherallokation von Ergebnisbildern.

+'extern_free_funct': Pointer auf externe Funktion zur Speicherfreigabe von Ergebnisbildern.

'temp_mem': Bedarf an temporärem Speicher des zuletzt ausgeführten Operators. Diese Option wird nur in Standard-HALCON unterstützt. In Parallel HALCON wird stattdessen der Bedarf des Operators `get_system` an temporärem Speicher zurückgegeben.

+'language': Sprache, in der die Fehlermeldungen ausgegeben werden ('english' oder 'german').

Parameter

▷ **`Query`** (input_control) ..... string(-array) ⇝ *string*
Gewünschter Systemparameter.
Defaultwert : 'width'
Werteliste : `Query` ∈ {"?", 'version', 'last_update', 'revision', 'max_images', 'max_channels', 'max_obj_per_par', 'max_inp_obj_par', 'max_outp_obj_par', 'max_inp_ctrl_par', 'max_outp_ctrl_par', 'max_window', 'max_window_types', 'max_proc', 'flush_graphic', 'int2_bits', 'int_zooming', 'draw_mode', 'ignore_colormap', 'backing_store', 'icon_name', 'window_name', 'default_font', 'single_lut', 'update_lut', 'x_package', 'num_gray_4', 'num_gray_6', 'num_gray_8', 'num_gray_percentage', 'num_graphic_2', 'num_graphic_4', 'num_graphic_6', 'num_graphic_8', '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', 'image_dir', 'lut_dir', 'reference_dir', 'help_dir', 'num_proc', 'num_sys_proc', 'num_user_proc', 'temp_mem', 'alloctmp_single_block', 'alloctmp_max_blocksize', 'extern_alloc_funct', 'extern_free_funct', 'do_low_error', 'reentrant', 'parallelize_operators', 'processor_num', 'max_connection', 'byte_order', 'operating_system', 'operating_system_version', 'language'}

▷ **`Information`** (output_control) ..... integer(-array) ⇝ *integer* / real / string
Aktueller Wert des Systemparameters.

Ergebnis

`get_system` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_system` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`reset_obj_db`

Mögliche Nachfolgerfunktionen

`set_system`

Siehe auch

`set_system`

Modul

System

`set_system ( : : Systemparameter, Value : )`

Verändern von HALCON-Systemparametern.

set_system ermöglicht es, verschiedene Systemparameter zur Laufzeit zu verändern.

Mögliche Systemparameter:

'neighborhood': Verwendung bei allen Prozeduren, bei denen Nachbarschaftsbeziehungen untersucht werden: connection, get_region_contour, get_region_chain, get_region_polygon, get_region_thickness, boundary, paint_region, disp_region, fill_up, contlength, shape_histo_all.
Value: 4 oder 8
default: 8

'default_font': Beim Öffnen eines Fensters wird automatisch ein Font für die Textausgabe gesetzt. Dabei wird der 'default_font' verwendet. Wird der voreingestellte Font nicht gefunden, so kann vor dem Öffnen des Fensters ein anderer Fontname angegeben werden.
Value: Dateiname des Fonts
default: fixed

'single_lut': (keine Beschreibung vorhanden) default: 'false'

'update_lut' Legt fest, ob die HALCON-Farbtabellen an die Umgebung angepäßt werden.
Value: 'true' oder 'false'
default: 'false'

'image_dir': Bilddaten (z.B. read_image und read_sequence) werden im aktuellen Directory und im 'image_dir' gesucht (sofern keine absoluten Pfade angegeben werden). Man kann mehrere Directories angegeben (Suchpfad), jeweils getrennt durch ein Semikolon (Windows NT) bzw. einen Doppelpunkt (Unix). Der Pfad kann auch mit der Environmentvariablen HALCONIMAGES festgelegt werden.
Value: Name des Dateipfades
default: '$HALCONROOT/images' bzw. '%HALCONROOT%/images'

'lut_dir': Farbtabellen (set_lut), die als ASCII-Datei realisiert sind, werden im aktuellen Directory und im 'lut_dir' gesucht (sofern keine absoluten Pfade angegeben werden). Falls HALCONROOT gesetzt ist, sucht HALCON die Farbtabellen im Unter-Directory „lut".
Value: Name des Dateipfades
default: '$HALCONROOT/lut' bzw. '%HALCONROOT%/lut'

'reference_dir': HTML und Postscript-Quellen der HALCON-Dokumentation (insbesondere das HALCON-Manual selbst) werden im aktuellen Directory und im 'reference_dir' gesucht. Dieser Systemparameter wird z.B. für disp_info benötigt. Dieser Parameter kann auch vor dem Aufruf von HALCON durch die Environmentvariable HALCONROOT gesetzt werden. Dabei muß die Variable auf das Directory oberhalb des Hilfe-Directories zeigen (das ist das HALCON-Homedirectory): z.B.: '/usr/local/halcon'
Value: Name des Dateipfades
default: '$HALCONROOT/doc/ps/reference/c' bzw. '%HALCONROOT%/doc/ps/reference/c'

'help_dir': Die Online-Textdatein german oder english.hlp, .key, .sta, .num und .idx werden im aktuellen Directory und im 'help_dir' gesucht. Dieser Systemparameter wird u.a. für get_operator_info und get_param_info benötigt. Dieser Parameter kann auch vor dem Aufruf von HALCON durch die Environmentvariable HALCONROOT gesetzt werden. Dabei muß die Variable auf das Directory oberhalb des Hilfe-Directories zeigen (das ist das HALCON-Homedirectory): z.B.: '/usr/local/halcon'
Value: Name des Dateipfades
default: '$HALCONROOT/help' bzw. '%HALCONROOT%/help'

'only_lines': (keine Beschreibung vorhanden) default: 'true'

'init_new_image': legt fest, ob neue Bilder vor der Anwendung von Filtern auf 0 gesetzt werden sollen. Dies ist nicht noetig, falls immer das ganze Bild gefiltert wird oder die Daten von nicht gefilterten Bildbereichen nicht von Bedeutung sind.
Value: 'true' oder 'false'
default: 'true'

'no_object_result': legt fest, wie sich Operationen, die Bilder verarbeiten, bei leeren Bildern (= keine Bilder) als Eingabe verhalten sollen. Mögliche Werte für Value:
'true': Fehler wird übergangen
'false': Prozedur liefert FALSE
'fail': Prozedur liefert FAIL
'void': Prozedur liefert VOID
'exception': Fehlerbehandlung
default: 'true'

'empty_region_result': steuert das Verhalten von Prozeduren auf leere Eingaberegionen, die für solche Regionen eigentlich nicht sinnvoll sind (z.B. diverse Regionenmerkmale, Segmentation, etc.). Mögliche Werte für `Value`:
'true': Der Fehler wird so gut es geht übergangen
'false': Prozedur liefert FALSE
'fail': Prozedur liefert FAIL
'void': Prozedur liefert VOID
'exception': Fehlerbehandlung
default: 'true'

'store_empty_region': Bei einer Reihe von Operationen kann es vorkommen, daß Objekte mit leerer Region (= keine Bildpunkte) erzeugt werden (z.B. `intersection`, `threshold`, etc.). Dieser Parameter legt fest ob diese Operatoren ein Objekt mit leerer Region als Ergebnis ausgegeben (**'true'**) oder ob die Regionen unterdrückt also nicht ausgegeben werden sollen (**'false'**).
`Value`: 'true' oder 'false'
default: 'true'

'backing_store': legt fest, ob der Inhalt von Fenstern, falls diese sich überlappt hatten, automatisch regeneriert wird. Einige Implementierungen von X-Window sind fehlerhaft; um diesen Fehler zu umgehen, kann das Sichern des Inhalts abgeschaltet werden. Es kann in einigen Fällen sinnvoll sein, den Sicherungsrungsmechanismus auszuschalten, wenn es auf Laufzeit / Speicherplatz ankommt.
`Value`: true oder false
default: true

'flush_graphic': Nach jeder HALCON-Prozedur, die eine Graphik-Ausgabe erzeugt, wird eine Flush-Operation ausgeführt, um die Daten sofort sichtbar zu machen. Bei einigen Programmen ist dies jedoch nicht nötig (z.B. wenn immer mit der Maus gearbeitet) wird. In diesen Fällen kann 'flush_graphic' auf 'false' gesetzt werden, um die Laufzeit zu verbessern.
`Value`: 'true' oder 'false'
default: 'true'

'flush_file': Mit diesem Parameter wird festgelegt, ob Ausgaben auf Datei (auch Terminal) gepuffert werden oder nicht. Falls gepuffert wird, werden die Daten auf dem Terminal in der Regel erst nach `fnew_line` sichtbar.
`Value`: 'true' oder 'false'
default: 'true'

'x_package': Bei der Ausgabe von Bilddaten über das Netz kann es wegen starker Auslastung des Rechners oder des Netzes zu Fehlübertragungen kommen. Um dies zu vermeiden, werden die Daten in kleinen Paketen transportiert. Wird lokal am Rechner gearbeitet, so können diese Einheiten beliebig vergrößert werden, was zu einer wesentlichen Verbesserung der Ausgabeleistung führt.
`Value`: Paketgröße (in Bytes)
default: 20480

'int2_bits': Anzahl der signifikanten Bits bei int2 images. Diese Anzahl wird für die Darstellung von int2 Bilder verwendet. Wenn der Wert -1 ist, werden die Grauwerte automatisch maximal skaliert.
`Value`: -1 oder 9..15
default: -1

'num_gray_4': Anzahl der unter X-Window zu reservierendenen Farben für die Ausgabe von Graustufen (disp_channel) auf einer Maschine mit 4 Bitplanes (16 Farben).
Achtung! Dieser Wert darf nur geändert werden, bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 2 - 12
default: 8

'num_gray6_': Anzahl der unter X-Window zu reservierendenen Farben für die Ausgabe von Graustufen (disp_channel) auf einer Maschine mit 6 Bitplanes (64 Farben).
Achtung! Dieser Wert darf nur geändert werden, bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 2 - 62
default: 50

'num_gray_8': Anzahl der unter X-Window zu reservierendenen Farben für die Ausgabe von Graustufen (disp_image) auf einer Maschine mit 8 Bitplanes (256 Farben).
Achtung! Dieser Wert darf nur geändert werden, bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 2 - 254
default: 140

'num_gray_percentage': HALCON reserviert unter X-Window einen Teil der verfügbaren Farben für die Darstellung von Graustufen (disp_image). Dabei wird versucht, andere X-Applikationen möglichst wenig zu beeinträchtigen. Gelingt es HALCON allerdings nicht, mindestens 'num_gray_percentage' Prozent der benötigten Farben vom X-Server zu erhalten, wird einfach ein bestimmter Teil der Lookup-Table ohne Rücksicht auf Verluste für die HALCON Graustufen reserviert. In dem Fall kann es zu unerwüenschten Farbverschiebungen kommen, wenn zwischen HALCON Fenstern und den Fenstern anderer Applikationen gewechselt oder (außerhalb von HALCON) ein Window-Dump erzeugt wird. Die Zahl der zu reservierenden realen Graustufen ist abhängig von der Anzahl der Bitplanes auf der Ausgabemaschine (siehe 'num_gray_*'). Auf S/W-Maschinen werden naturgemäß keine Farben reserviert - Graustufen werden vielmehr bei der Ausgabe 'gedithert'. Bei Verwendung von Graustufen-Displays werden nur Grautöne ('black', 'white', 'gray' etc.) verwendet. Maschinen mit 2 oder 3 Bitplanes werden wie S/W-Maschinen behandelt, Maschinen mit 5 (7) Bitplanes wie Maschinen mit 4 (6) Bitplanes und Maschinen mit mehr als 8 Bitplanes wie Maschinen mit 8 Bitplanes. Eine Sonderstellung nehmen Maschinen mit 24 Bit Display (Echtfarbenmaschinen) ein. Hier entfällt natürlich die Reservierung von Farben für die Graustufenausgabe.
Achtung! Dieser Wert darf nur geändert werden, bevor das erste Fenster auf dieser Maschine geoeffnet wurde. Auf einer Maschine mit x Bitplanes enthält num_gray_x dabei vor dem Öffnen des ersten Fensters die Anzahl der zu reservierenden Farben für die Graustufenausgabe, nach dem Öffnen des ersten Fensters hingegen die Anzahl der tatsächlich reservierten Farben.
`Value`: 0 - 100
default: 30

'num_graphic_percentage': (keine Beschreibung vorhanden) default: 60

'draw_mode': (keine Beschreibung vorhanden) default: 'complement'

'int_zooming': (keine Beschreibung vorhanden) default: 'true'

'ignore_colormap': (keine Beschreibung vorhanden) default: 'false'

'icon_name': (keine Beschreibung vorhanden) default: 'default'

'num_graphic_2': Anzahl der unter X-Window zu reservierenden HALCON Graphikfarben (für disp_region etc.) auf einer Maschine mit 2 Bitplanes (4 Farben).
Achtung! Dieser Wert darf nur geändert werden, bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 0 - 2
default: 2

'num_graphic_4': Anzahl der unter X-Window zu reservierenden HALCON Graphikfarben (für disp_region etc.) auf einer Maschine mit 4 Bitplanes (16 Farben).
Achtung! Dieser Wert darf nur geändert werden, bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 0 - 14
default: 5

'num_graphic_6': Anzahl der unter X-Window zu reservierenden HALCON Graphikfarben (für disp_region etc.) auf einer Maschine mit 6 Bitplanes (64 Farben).
Achtung! Dieser Wert darf nur geändert werden bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 0 - 62
default: 10

'num_graphic_8': Anzahl der unter X-Window zu reservierenden HALCON Graphikfarben (für disp_region etc.) auf einer Maschine mit 8 Bitplanes (256 Farben).
Achtung! Dieser Wert darf nur geändert werden bevor das erste Fenster auf dieser Maschine geöffnet wurde.
`Value`: 0 - 64
default: 20

'graphic_colors' HALCON reserviert die ersten num_graphic_x Farben aus dieser Liste von X-Window Farbnamen als Graphikfarben. Per Default verwendet HALCON dabei dieselbe Liste, die auch beim Aufruf von query_all_colors geliefert wird. Die Liste kann allerdings individuell angepaßt werden: Dazu wird hier ein Tupel von Farbnamen als Value uebergeben. Sinnvollerweise sollte ein solches Tupel immer die Farben 'black' und 'white', eventuell auch 'red', 'green' und 'blue' enthalten. Wird als Value 'default' übergeben, kehrt HALCON zur Voreinstellung zurück.
Hinweis: Bei Graustufenmaschinen werden nicht die ersten x Farben, sondern die ersten x Grautöne aus der Liste reserviert.
Achtung! Diese Liste darf nur vor dem Öffnen des ersten Fensters veräendert werden.
`Value`: Tupel von X-Window Farbnamen
default: siehe query_all_colors

'current_runlength_number': Regionen werden intern in einer Lauflängenkodierung abgelegt. Mit diesem Parameter kann die maximale Anzahl der Sehnen festgelegt werden, die für die Darstellung einer Region verwendet werden können. Es ist zu beachten, daß einige Prozeduren selbständig die Anzahl hochsetzen, falls dies notwendig ist. Der Wert kann sowohl vergrößert als auch verkleinert werden.
`Value`: maximale Sehnenzahl
default: 50000

'clip_region': Legt fest, ob die Regionen in der HALCON-Datenbank auf das aktuelle Bildformat beschnitten („geclippt") werden oder nicht. Dies ist bei Prozeduren wie z.B. bei `gen_circle`, `gen_rectangle1` oder `dilation1` der Fall.
`Value`: 'true' oder 'false'
default: 'true'

'do_low_error' Legt fest, ob HALCON low-level-Fehler ausgibt oder nicht.
`Value`: 'true' or 'false'
default: 'false'

'reentrant' Gibt an, ob HALCON innerhalb einer parallel arbeitenden Programmumgebung eingesetzt wird, oder nicht. Dieser Parameter ist ausschliesslich für Parallel HALCON von Bedeutung, das die gleichzeitige Abarbeitung mehrerer HALCON-Operatoren erlaubt. Folgerichtig wird er in der sequentiell arbeitenden HALCON-Version ignoriert. Ist der Parameter auf 'true' gesetzt, so verwendet Parallel HALCON interne Synchronisierungsmechanismen, um gemeinsam benutzte Datenobjekte vor parallelen Zugriffen zu schützen. So notwendig diese Maßnahme in einer echt parallel arbeitenden Anwendung sein mag, so unerwünscht ist der dadurch entstehende Overhead, falls die Anwendung alle HALCON-Operatoren rein sequentiell aufruft. Für den letzteren Fall kann man dem System die sequentielle Arbeitsweise durch das Setzen auf 'false' signalisieren. Hierauf werden die internen Synchronisierungsmechanismen ausser Kraft gesetzt, so dass sich der Overhead verringert. Als Folge ergibt sich natürlich, dass Parallel HALCON dann nicht mehr thread-safe ist. Hierdurch ergibt sich als Seiteneffekt, dass durch das Setzen von 'reentrant' auf 'false' auch die interne Parallelisierung ausser Kraft gesetzt wird, die Parallel HALCON normalerweise einsetzt, um die Abarbeitung von Operatoren auf Mehrprozessormaschinen zu beschleunigen. Wird 'reentrant' hingegen auf 'true' gesetzt, so geht Parallel HALCON in den Standardbetrieb über, das heißt, es arbeitet thread-safe und es setzt die automatische Parallelisierung von Operatoren ein.
`Value`: 'true' or 'false'
default: Parallel HALCON: 'true', sonst: 'false'

'parallelize_operators' Gibt an, ob Parallel HALCON eine automatische Parallelisierung einsetzt, um die Abarbeitung von Operatoren auf Mehrprozessormaschinen zu beschleunigen. Durch das Setzen auf 'false' wird die automatische Parallelisierung abgeschaltet, wobei Parallel HALCON jedoch auch weiterhin nicht nur thread-safe bleibt, sondern auch die gleichzeitige Abarbeitung mehrerer HALCON-Operatoren erlaubt. Das Setzen auf 'false' ist zum Beispiel dann sinnvoll, wenn HALCON-Operatoren innerhalb eines parallelen Programms aufgerufen werden, das auch das Scheduling bzw. eine geeignete Lastbalancierung der Operatoren und Daten selbst festlegt. In diesem Fall kann es wünschenswert sein, dass HALCON nicht auch noch zusätzliche Parallelisierungsschritte selbst durchführt, die dem Lastbalancierungskonzept der Anwendung in die Quere kommen könnten. Der Parameter 'parallelize_operators' wird ausschließlich von Parallel HALCON unterstützt und wird von der sequentiell arbeitenden HALCON-Version ignoriert.
`Value`: 'true' or 'false'
default: Parallel HALCON: 'true', otherwise: 'false'

'max_connection' Legt die maximale Anzahl von Regionen fest, die von `connection` zurückgeliefert wird. Für `Value`=**0** werden alle Regionen zurückgeliefert.

'alloctmp_single_block' Art der Verwaltung des temporären Speichers. `Value`: 'true' or 'false'
default: 'false'

'alloctmp_max_blocksize' Maximale Größe zu allokierender Speicherblöcke innerhalb temporärer Speicherverwaltung. (Ohne Effekt, falls 'alloctmp_max_blocksize' == -1 oder 'alloctmp_single_block' == 'true') `Value`: -1 oder ¿= 0
default: -1

'extern_alloc_funct' Pointer auf externe Funktion zur Speicherallokation von Ergebnisbildern. default: 0

'extern_free_funct' Pointer auf externe Funktion zur Speicherfreigabe von Ergebnisbildern. default: 0

'language' Sprache, in der die Fehlermeldungen ausgegeben werden. `Value`: 'english' oder 'german'. default: ' english'

Parameter

▷ **Systemparameter** (input_control) ..... string(-array) ⇝ *string*
Name des zu verändernden Systemparameters.
Defaultwert : 'image_dir'
Werteliste : Systemparameter ∈ {'neighborhood', 'default_font', 'single_lut', 'update_lut', 'image_dir', 'lut_dir', 'reference_dir', 'help_dir', 'only_lines', 'init_new_image', 'no_object_result', 'empty_region_result', 'store_empty_region', 'backing_store', 'flush_graphic', 'flush_file', 'x_package', 'int2_bits', 'icon_name', 'num_gray_4', 'num_gray_6', 'num_gray_8', 'num_gray_percentage', 'num_graphic_2', 'num_graphic_4', 'num_graphic_6', 'num_graphic_8', 'num_graphic_percentage', 'draw_mode', 'int_zooming', 'ignore_colormap', 'graphic_colors', 'current_runlength_number', 'clip_region', 'update_lut', 'do_low_error', 'reentrant', 'parallelize_operators', 'max_connection', 'alloctmp_single_block', 'alloctmp_max_blocksize', 'extern_alloc_funct', 'extern_free_funct', 'language'}

▷ **Value** (input_control) ..... string(-array) ⇝ *string* / integer / real
Neuer Wert des Systemparameters.
Defaultwert : 'true'
Wertevorschläge : Value ∈ {'true', 'false', 0, 4, 8, 100, 140, 255}

Ergebnis

set_system liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

set_system ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

reset_obj_db, get_system, set_check

Siehe auch

get_system, set_check

Modul

System

11.7 Seriell

clear_serial (: : SerialHandle, Channel :)

Löschen des Puffers einer seriellen Verbindung.

clear_serial löscht Daten, die an das serielle Gerät, das durch SerialHandle bezeichnet wird, geschickt, aber noch nicht übertragen worden sind (Channel = **'output'**), oder löscht Daten, die bereits empfangen, aber noch nicht gelesen worden sind (Channel = **'input'**), oder führt beide Operationen gleichzeitig aus (Channel = **'in_out'**).

Parameter

▷ **SerialHandle** (input_control) ..... serial_id ⇝ *integer*
Handle der seriellen Schnittstelle.

▷ **Channel** (input_control) ..... string ⇝ *string*
Zu löschender Puffer.
Defaultwert : 'input'
Werteliste : Channel ∈ {'input', 'output', 'in_out'}

Ergebnis

Wenn die Parameter korrekt sind und die Puffer des seriellen Geräts gelöscht werden konnten, liefert clear_serial den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

clear_serial ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_serial

Mögliche Nachfolgerfunktionen

`read_serial`, `write_serial`

Siehe auch

`read_serial`

Modul

System

`close_all_serials` (: : :)

Schließen aller seriellen Geräte.

`close_all_serials` schließt alle seriellen Geräte, die mit `open_serial` geöffnet wurde.

Ergebnis

`close_all_serials` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`close_all_serials` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_serial`

Alternativen

`close_serial`

Siehe auch

`open_serial`, `close_file`

Modul

System

`close_serial` (: : SerialHandle :)

Schließen eines seriellen Gerätes.

`close_serial` schließt ein serielles Gerät, das mit `open_serial` geöffnet wurde.

Parameter

▷ **`SerialHandle`** (input_control) . . . . . serial_id ⇝ *integer*
Handle der seriellen Schnittstelle.

Ergebnis

Wenn die Parameter korrekt sind und das angegebene Gerät geschlossen werden konnte, liefert `close_serial` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_serial` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_serial`

Siehe auch

`open_serial`, `close_file`

Modul

System

`get_serial_param` (: : SerialHandle : BaudRate, DataBits, FlowControl, Parity, StopBits, TotalTimeOut, InterCharTimeOut)

Auslesen der Parameter eines seriellen Geräts.

get_serial_param liefert die aktuellen Parameter des seriellen Geräts, das in SerialHandle übergeben wird, zurück. Für eine Beschreibung der Parameter der seriellen Schnittstelle siehe set_serial_param.

Parameter

- ▷ **SerialHandle** (input_control) ..... serial_id ⇝ *integer*
  Handle der seriellen Schnittstelle.
- ▷ **BaudRate** (output_control) ..... integer ⇝ *integer*
  Übertragungsgeschwindigkeit der seriellen Schnittstelle.
- ▷ **DataBits** (output_control) ..... integer ⇝ *integer*
  Anzahl Datenbits der seriellen Schnittstelle.
- ▷ **FlowControl** (output_control) ..... string ⇝ *string*
  Datenflußkontrolle der seriellen Schnittstelle.
- ▷ **Parity** (output_control) ..... string ⇝ *string*
  Parität der seriellen Schnittstelle.
- ▷ **StopBits** (output_control) ..... integer ⇝ *integer*
  Anzahl Stopbits der seriellen Schnittstelle.
- ▷ **TotalTimeOut** (output_control) ..... integer ⇝ *integer*
  Gesamt-Timeout der seriellen Schnittstelle in ms.
- ▷ **InterCharTimeOut** (output_control) ..... integer ⇝ *integer*
  Inter-character Timeout der seriellen Schnittstelle in ms.

Ergebnis

Wenn die Parameter korrekt sind und die Parameter des Gerätes gelesen werden konnten, liefert get_serial_param den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

get_serial_param ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_serial

Mögliche Nachfolgerfunktionen

get_serial_param, read_serial, write_serial

Siehe auch

set_serial_param

Modul

System

open_serial (: : PortName : SerialHandle)

Öffnen eines seriellen Gerätes.

open_serial öffnet ein serielles Gerät. Der Name des Geräts wird in PortName. Er ist Betriebssystem-abhängig: auf Windows NT Rechnern wird typischerweise **’COM1’-’COM4’** verwendet, während die seriellen Schnittstellen unter Unix normalerweise **’/dev/tty*’** heißen. Die Parameter der seriellen Schnittstelle, wie die Übertragungsgeschwindigkeit oder die Anzahl der Datenbits, sind nach dem Öffnen auf die Standardwerte des jeweiligen Gerätes gesetzt. Sie können mit set_serial_param gesetzt oder verändert werden.

Parameter

- ▷ **PortName** (input_control) ..... filename.named ⇝ *string*
  Name der seriellen Schnittstelle.
  Defaultwert : ”COM1”
  Wertevorschläge : PortName ∈ {”COM1”, ”COM2”, ”COM3”, ”COM4”, ”/dev/ttya”, ”/dev/ttyb”, ”/dev/tty00”, ”/dev/tty01”, ”/dev/ttyd1”, ”/dev/ttyd2”, ”/dev/cua0”, ”/dev/cua1”}
- ▷ **SerialHandle** (output_control) ..... serial_id ⇝ *integer*
  Handle der seriellen Schnittstelle.

Ergebnis

Wenn die Parameter korrekt sind und das angegebene Gerät geöffnet werden konnte, liefert open_serial den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`open_serial` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`set_serial_param`, `read_serial`, `write_serial`, `close_serial`

Siehe auch

`set_serial_param`, `get_serial_param`, `open_file`

Modul

System

read_serial (: : SerialHandle, NumCharacters : Data)

Lesen von einem seriellen Gerät.

`read_serial` versucht, `NumCharacters` von dem seriellen Gerät, das durch `SerialHandle` angegeben wird, zu lesen. Die gelesenen Daten werden in `Data` als Tupel von Integer-Werten zurückgegeben. Dies erlaubt das Lesen von NUL-Zeichen, die sonst als String-Ende interpretiert würden. Falls der Timeout des Gerätes mit `set_serial_param` auf einen Wert größer als 0 gesetzt wurde, wird so lange auf die Ankunft des ersten Zeichens gewartet, wie durch den Timeout angegeben ist, ansonsten kehrt die Funktion sofort zurück. In jedem Fall werden die zur Zeit der Rückkehr verfügbaren Zeichen zurückgeliefert. D.h., es können weniger Zeichen als angefordert zurückgeliefert werden. Dies kann über die Länge des Tupels `Data` überprüft werden.

Parameter

- ▷ **SerialHandle** (input_control) ..... serial_id ⇝ *integer*
  Handle der seriellen Schnittstelle.
- ▷ **NumCharacters** (input_control) ..... integer ⇝ *integer*
  Anzahl der zu lesenden Zeichen.
  Defaultwert : 1
  Wertevorschläge : `NumCharacters` ∈ {1, 2, 3, 4, 5, 10, 20, 40, 100}
- ▷ **Data** (output_control) ..... integer(-array) ⇝ *integer*
  Gelesene Zeichen (als Integer-Tupel).

Ergebnis

Wenn die Parameter korrekt sind und das Lesen vom seriellen Gerät erfolgreich war, liefert `read_serial` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_serial` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_serial`

Siehe auch

`write_serial`

Modul

System

set_serial_param (: : SerialHandle, BaudRate, DataBits, FlowControl, Parity, StopBits, TotalTimeOut, InterCharTimeOut :)

Setzen der Parameter eines seriellen Gerätes.

Mit `set_serial_param` können die Parameter eines seriellen Gerätes gesetzt werden. Der Parameter `BaudRate` gibt die Ein- und Ausgabegeschwindigkeit des Gerätes an. Es ist zu beachten, daß nicht jedes Gerät alle Geschwindigkeiten unterstützt. Die Anzahl der übertragenen Datenbits wird mit `DataBits` festgelegt. Der Parameter `FlowControl` legt fest, ob und welche Art von Datenflußkontrolle verwendet werden. Im letzteren Fall kann zwischen Software-Kontrolle (**’xon_xoff’**) und Hardware-Kontrolle (**’cts_rts’**, **’dtr_dsr’** gewählt werden. Ob und welche Art von Paritätskontrolle der übertragenen Daten verwendet wird, läßt sich mit `Parity` festlegen. Die Anzahl der übertragenen Stop-Bits kann mit `StopBits` gesetzt werden. Schließlich können noch

zwei Timeouts für das Lesen von dem seriellen Gerät gesetzt werden. Der Parameter `TotalTimeOut` bezieht sich auf die Zeit, die bei `read_serial` bis zum Eintreffen des ersten Zeichens, unabhängig von der Anzahl der insgesamt zu lesenden Zeichen, höchstens vergehen darf. Der Parameter `InterCharTimeOut` gibt die Zeit an, die zwischen dem Lesen einzelner Zeichen verstreichen darf, falls mehrer Zeichen mit `read_serial` gelesen werden sollen. Falls einer der beiden Timeouts auf **-1** gesetzt wird, so wird beliebig lange auf das Eintreffen eines Zeichens gewartet. Auf Windows-NT-Systemen ergibt sich so ein maximaler Timout von `TotalTimeOut` $+ n$`InterCharTimeOut`, wenn n Zeichen gelesen werden sollen. Auf Unix-Systemen kann systembedingt nur einer der beiden Timeouts gesetzt werden. Wenn beide Timeouts größer als -1 übergeben werden, wird nur der Total-Timeout beachtet. Die Einheit der beiden Timeouts ist Millisekunden. Für jeden dieser Parameter kann die aktuelle Einstellung beibehalten werden, indem **'unchanged'** übergeben wird.

Parameter

- ▷ **`SerialHandle`** (input_control) . . . . . serial_id ⇝ *integer*
  Handle der seriellen Schnittstelle.
- ▷ **`BaudRate`** (input_control) . . . . . integer ⇝ *integer* / string
  Übertragungsgeschwindigkeit der seriellen Schnittstelle.
  Defaultwert : 'unchanged'
  Werteliste : `BaudRate` ∈ {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 ⇝ *integer* / string
  Anzahl Datenbits der seriellen Schnittstelle.
  Defaultwert : 'unchanged'
  Werteliste : `DataBits` ∈ {5, 6, 7, 8, 'unchanged'}
- ▷ **`FlowControl`** (input_control) . . . . . string ⇝ *string*
  Datenflußkontrolle der seriellen Schnittstelle.
  Defaultwert : 'unchanged'
  Werteliste : `FlowControl` ∈ {'none', 'xon_xoff', 'cts_rts', 'dtr_dsr', 'unchanged'}
- ▷ **`Parity`** (input_control) . . . . . string ⇝ *string*
  Parität der seriellen Schnittstelle.
  Defaultwert : 'unchanged'
  Werteliste : `Parity` ∈ {'none', 'odd', 'even', 'unchanged'}
- ▷ **`StopBits`** (input_control) . . . . . integer ⇝ *integer* / string
  Anzahl Stopbits der seriellen Schnittstelle.
  Defaultwert : 'unchanged'
  Werteliste : `StopBits` ∈ {1, 2, 'unchanged'}
- ▷ **`TotalTimeOut`** (input_control) . . . . . integer ⇝ *integer* / string
  Gesamt-Timeout der seriellen Schnittstelle in ms.
  Defaultwert : 'unchanged'
  Wertevorschläge : `TotalTimeOut` ∈ {-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 'unchanged'}
- ▷ **`InterCharTimeOut`** (input_control) . . . . . integer ⇝ *integer* / string
  Inter-character Timeout der seriellen Schnittstelle in ms.
  Defaultwert : 'unchanged'
  Wertevorschläge : `InterCharTimeOut` ∈ {-1, 0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 'unchanged'}

Ergebnis

Wenn die Parameter korrekt sind und die Parameter des Gerätes gesetzt werden konnten, liefert `set_serial_param` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`set_serial_param` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_serial`, `get_serial_param`

Mögliche Nachfolgerfunktionen

`read_serial`, `write_serial`

Siehe auch

`get_serial_param`

Modul

System

write_serial (: : SerialHandle, Data :)

Schreiben in eine seriellen Verbindung.

write_serial schreibt die Zeichen, die in Data übergeben werden, auf das serielle Gerät, das durch SerialHandle angegeben wird. Die zu schreibenden Daten werden als Tupel von Integer-Werten übergegeben. Dies erlaubt das Schreiben von NUL-Zeichen, die sonst als String-Ende interpretiert würden. Beim Schreiben wird immer gewartet, bis die Übertragung erfolgt ist, d.h., es kann kein Timeout gesetzt werden.

Parameter

- ▷ **SerialHandle** (input_control) . . . . . serial_id ⇝ *integer*
  Handle der seriellen Schnittstelle.
- ▷ **Data** (input_control) . . . . . integer(-array) ⇝ *integer*
  Zu schreibende Zeichen (als Integer-Tupel).

Ergebnis

Wenn die Parameter korrekt sind und das Schreiben zum seriellen Gerät erfolgreich war, liefert write_serial den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_serial ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_serial

Siehe auch

read_serial

Modul

System

11.8 Sockets

close_socket (: : Socket :)

Schließen einer Socket-Verbindung.

close_socket schließt eine Socket-Verbindung, die zuvor mit open_socket_accept, open_socket_connect oder socket_accept_connect geöffnet worden ist. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

- ▷ **Socket** (input_control) . . . . . socket_id ⇝ *integer*
  Nummer des Sockets.

Parallelisierungsinformation

close_socket ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

open_socket_accept, open_socket_connect, socket_accept_connect

Modul

System

get_next_socket_data_type (: : Socket : DataType)

Abfrage des nächsten an einem Socket anliegenden HALCON-Datentyps.

`get_next_socket_data_type` liefert den Datentyp der nächsten am Socket `Socket` anliegenden Daten in `DataType` zurück. Die möglichen Werte für `DataType` sind:

'no_data': Es liegen keine Daten an.

'no_halcon_data': Es liegen Daten an, sie sind aber keine HALCON-Daten.

'tuple': Die nächsten Daten sind ein Tupel.

'region': Die nächsten Daten sind ein Regionenobjekt.

'image': Die nächsten Daten sind ein Bildobjekt.

'xld_cont': Die nächsten Daten sind XLD-Konturen.

'xld_poly': Die nächsten Daten sind XLD-Polygone.

'xld_para': Die nächsten Daten sind XLD-Parallelen.

'xld_mod_para': Die nächsten Daten sind modifizierte XLD-Parallelen.

'xld_ext_para': Die nächsten Daten sind erweiterte XLD-Parallelen.

Parameter

- ▷ **`Socket`** (input_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.
- ▷ **`DataType`** (output_control) ..... string ⇝ *string*
  Anliegender HALCON-Datentyp.

Parallelisierungsinformation

`get_next_socket_data_type` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Siehe auch

`send_image`, `receive_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`

Modul

System

`open_socket_accept ( : : Port : AcceptingSocket )`

Öffnen eines Sockets, der Verbindungsanfragen zuläßt.

`open_socket_accept` öffnet einen Socket, der ankommende Verbindungsanfragen von anderen HALCON-Prozessen akzeptiert. Dieser Operator ist der notwendige erste Schritt, um einen Kommunikationskanal zwischen zwei HALCON-Prozessen aufzubauen. Der Socket horcht nach ankommenden Verbindungsanfragen auf dem durch `Port` bestimmten Port. Der akzeptierende Socket wird in `AcceptingSocket` zurückgegeben. `open_socket_accept` kehrt sofort zurück, ohne auf eine Verbindungsanfrage eines anderen Prozesses zu warten, die durch Aufruf von `open_socket_connect` im anderen Prozeß erfolgt. Dies ermöglicht es mehreren Prozessen, mit dem Prozeß, der `open_socket_accept` aufruft, eine Verbindung herzustellen. Um eine ankommende Verbindungsanfrage zu akzeptieren, muß `socket_accept_connect` aufgerufen werden, nachdem der andere Prozeß `open_socket_connect` ausgeführt hat.

Parameter

- ▷ **`Port`** (input_control) ..... integer ⇝ *integer*
  Nummer des Ports.
  Defaultwert : 3000
  Typischer Wertebereich : $1024 \leq$ `Port` ≤ 65535
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **`AcceptingSocket`** (output_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.

Beispiel

```
/* 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)
receive_region (ConnectedRegions, 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)
```

Parallelisierungsinformation

`open_socket_accept` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`socket_accept_connect`

Siehe auch

`open_socket_connect`, `close_socket`, `send_image`, `receive_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`

Modul

System

`open_socket_connect` (: : HostName, Port : Socket)

Öffnen einer Socket-Verbindung zu einem bestehenden Socket.

`open_socket_connect` öffnet eine Socket-Verbindung zu einem akzeptierenden Socket auf dem Rechner mit dem Namen `HostName`, der auf dem Port `Port` nach Verbindungsanfragen horcht. Der akzeptierende Socket muß zuvor in dem anderen HALCON-Prozeß mit `open_socket_accept` erzeugt worden sein. Der so erzeugte Socket wird in `Socket` zurückgegeben. Um eine Verbindung herzustellen, muß der HALCON-Prozeß, in dem der akzeptierende Socket läuft, `socket_accept_connect` aufrufen. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Parameter

- ▷ **HostName** (input_control) ..... string ⇝ *string*
  Name des Rechners, zu dem die Verbindung aufgebaut werden soll.
  Defaultwert : 'localhost'
- ▷ **Port** (input_control) ..... integer ⇝ *integer*
  Nummer des Ports.
  Defaultwert : 3000
  Typischer Wertebereich : $1024 \leq$ Port ≤ 65535
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
- ▷ **Socket** (output_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.

Parallelisierungsinformation

open_socket_connect ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple

Siehe auch

open_socket_accept, socket_accept_connect, close_socket

Modul

System

receive_image (: Image : Socket :)

Empfangen von Bildern über eine Socket-Verbindung.

receive_image liest ein Bildobjekt, das von einem anderen HALCON-Prozeß über die Socket-Verbindung Socket mit dem Operator send_image geschickt worden ist. Wenn noch kein Bild verschickt worden sind, wird der Prozeß, der receive_image aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

- ▷ **Image** (output_object) ..... image(-array) ⇝ *Hobject*
  Empfangenes Bild.
- ▷ **Socket** (input_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.

Parallelisierungsinformation

receive_image ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_socket_connect, socket_accept_connect

Siehe auch

send_image, send_region, receive_region, send_tuple, receive_tuple, get_next_socket_data_type

Modul

System

receive_region (: Region : Socket :)

Empfangen von Regionen über eine Socket-Verbindung.

receive_region liest ein Regionenobjekt, das von einem anderen HALCON-Prozeß über die Socket-Verbindung Socket mit dem Operator send_region geschickt worden ist. Wenn noch keine Regionen verschickt worden sind, wird der Prozeß, der receive_region aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

▷ **Region** (output_object) . . . . . region(-array) ⇝ *Hobject*
Empfangene Regionen.
▷ **Socket** (input_control) . . . . . socket_id ⇝ *integer*
Nummer des Sockets.

Parallelisierungsinformation

`receive_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_connect`, `socket_accept_connect`

Siehe auch

`send_region`, `send_image`, `receive_image`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

System

receive_tuple (: : Socket : Tuple)

Empfangen eines Tupels über eine Socket-Verbindung.

`receive_tuple` liest ein Tupel, das von einem anderen HALCON-Prozeß über die Socket-Verbindung `Socket` mit dem Operator `send_tuple` geschickt worden ist. Wenn noch kein Tupel verschickt worden ist, wird der Prozeß, der `receive_tuple` aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Parameter

▷ **Socket** (input_control) . . . . . socket_id ⇝ *integer*
Nummer des Sockets.
▷ **Tuple** (output_control) . . . . . string ⇝ *string*
Empfangenes Tupel.

Parallelisierungsinformation

`receive_tuple` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_connect`, `socket_accept_connect`

Siehe auch

`send_tuple`, `send_image`, `receive_image`, `send_region`, `receive_region`, `get_next_socket_data_type`

Modul

System

receive_xld (: XLD : Socket :)

Empfangen von XLD-Objekten über eine Socket-Verbindung.

`receive_xld` liest ein XLD-Objekt, das von einem anderen HALCON-Prozeß über die Socket-Verbindung `Socket` mit dem Operator `send_xld` geschickt worden ist. Wenn noch kein XLD-Objekt verschickt worden sind, wird der Prozeß, der `receive_xld` aufgerufen hat, blockiert, bis genug Daten eingetroffen sind. Für ein ausführliches Beispiel siehe `send_xld`.

Parameter

▷ **XLD** (output_object) . . . . . xld(-array) ⇝ *Hobject*
Empfangenes XLD-Objekt.
▷ **Socket** (input_control) . . . . . socket_id ⇝ *integer*
Nummer des Sockets.

Parallelisierungsinformation

`receive_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_connect`, `socket_accept_connect`

Siehe auch

`send_xld`, `send_image`, `receive_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

System

send_image (Image : : Socket :)

Senden von Bildern über eine Socket-Verbindung.

`send_image` verschickt ein Bildobjekt über die durch `Socket` bestimmte Socket-Verbindung. Der empfangende HALCON-Prozeß muß `receive_image` aufrufen, um das Bild vom Socket auszulesen. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Zu schickendes Bild.
- ▷ **Socket** (input_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.

Parallelisierungsinformation

`send_image` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_connect`, `socket_accept_connect`

Siehe auch

`receive_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

System

send_region (Region : : Socket :)

Senden von Regionen über eine Socket-Verbindung.

`send_region` verschickt ein Regionenobjekt über die durch `Socket` bestimmte Socket-Verbindung. Der empfangende HALCON-Prozeß muß `receive_region` aufrufen, um die Regionen vom Socket auszulesen. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Parameter

- ▷ **Region** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu schickende Regionen.
- ▷ **Socket** (input_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.

Parallelisierungsinformation

`send_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_connect`, `socket_accept_connect`

Siehe auch

`receive_region`, `send_image`, `receive_image`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

System

send_tuple (: : Socket, Tuple :)

Senden eines Tupels über eine Socket-Verbindung.

send_tuple verschickt ein Tupel über die durch Socket bestimmte Socket-Verbindung. Der empfangende HALCON-Prozeß muß receive_tuple aufrufen, um das Tupel vom Socket auszulesen. Für ein ausführliches Beispiel siehe open_socket_accept.

Parameter

- ▷ **Socket** (input_control) . . . . . socket_id ⇝ *integer*
  Nummer des Sockets.
- ▷ **Tuple** (input_control) . . . . . string ⇝ *string*
  Zu sendendes Tupel.

Parallelisierungsinformation

send_tuple ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

open_socket_connect, socket_accept_connect

Siehe auch

receive_tuple, send_image, receive_image, send_region, receive_region, get_next_socket_data_type

Modul

System

send_xld (XLD : : Socket :)

Senden von XLD-Objekten über eine Socket-Verbindung.

send_xld verschickt ein XLD-Objekt über die durch Socket bestimmte Socket-Verbindung. Der empfangende HALCON-Prozeß muß receive_xld aufrufen, um das XLD-Objekt vom Socket auszulesen.

Parameter

- ▷ **XLD** (input_object) . . . . . xld(-array) ⇝ *Hobject*
  Zu schickendes XLD-Objekt.
- ▷ **Socket** (input_control) . . . . . socket_id ⇝ *integer*
  Nummer des Sockets.

Beispiel

```
/* 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)
receive_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)
```

Parallelisierungsinformation

`send_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_connect`, `socket_accept_connect`

Siehe auch

`receive_xld`, `send_image`, `receive_image`, `send_region`, `receive_region`, `send_tuple`, `receive_tuple`, `get_next_socket_data_type`

Modul

System

socket_accept_connect (: : AcceptingSocket, Wait : Socket)

Annehmen einer Verbindungsanfrage auf einem akzeptierenden Socket.

`socket_accept_connect` nimmt eine ankommende Verbindungsanfrage, die mit `open_socket_connect` von einem anderen HALCON-Prozeß erzeugt worden ist, auf dem akzeptierenden Socket `AcceptingSocket` an. Der akzeptierende Socket muß zuvor mit `open_socket_accept` erzeugt worden sein. Falls `Wait`=**'true'**, wartet `socket_accept_connect`, bis eine Verbindung zustande kommt. Falls `Wait`=**'false'**, kehrt `socket_accept_connect` mit der Fehlermeldung FAIL zurück, falls derzeit keine Verbindungsanfrage eines anderen Prozesses vorliegt. Das Ergebnis von `socket_accept_connect` ist ein weiterer Socket `Socket`, der für eine bidirektionale Kommunikation mit einem anderen HALCON-Prozeß verwendet werden kann. Nachdem die Verbindung hergestellt ist, können Daten zwischen den zwei Prozessen durch Aufruf der entsprechenden Sende- und Empfangsoperatoren ausgetauscht werden. Für ein ausführliches Beispiel siehe `open_socket_accept`.

Parameter

- ▷ **AcceptingSocket** (input_control) ..... socket_id ⇝ *integer*
  Nummer des akzeptierenden Sockets.
- ▷ **Wait** (input_control) ..... string ⇝ *string*
  Soll auf eine Verbindungsanfrage gewartet werden?
  Werteliste : `Wait` ∈ {'true', 'false'}
- ▷ **Socket** (output_control) ..... socket_id ⇝ *integer*
  Nummer des Sockets.

Parallelisierungsinformation

`socket_accept_connect` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`open_socket_accept`

Mögliche Nachfolgerfunktionen

send_image, receive_image, send_region, receive_region, send_tuple, receive_tuple

Siehe auch

open_socket_connect, close_socket

Modul

System

Kapitel 12

Tools

12.1 Affine-Abbildungen

affine_trans_point_2d (: : HomMat2D, Px, Py : Qx, Qy)

Wende eine homogene 2D-Transformations-Matrix auf Punkte an.

affine_trans_point_2d berechnet das Ergebnis der Anwendung der affinen Abbildung, die durch die homogene 2D-Transformations-Matrix HomMat2D gegeben ist, auf die Eingabepunkte (Px,Py). Die Ergebnispunkte werden in (Qx,Qy) zurückgeliefert.

Wenn es sich bei den Koordinaten (Px,Py) um Bildkoordinaten handelt, ist zu beachten, daß die Einträge der homogenen Transformationsmatrix werden so interpretiert werden sollten, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Parameter

- ▷ **HomMat2D** (input_control) . . . . . affine2d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 6
- ▷ **Px** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
  Defaultwert : 64
  Wertevorschläge : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Px ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Py** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
  Defaultwert : 64
  Wertevorschläge : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Py ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Qx** (output_control) . . . . . point.x(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).
- ▷ **Qy** (output_control) . . . . . point.y(-array) ⇝ *real*
  Ausgabepunkt (y-Koordinate).

Ergebnis

affine_trans_point_2d liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`affine_trans_point_2d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`

Mögliche Nachfolgerfunktionen

`hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`

Modul

Basic operators

affine_trans_point_3d (: : HomMat3D, Px, Py, Pz : Qx, Qy, Qz)

Wende eine homogene 3D-Transformations-Matrix auf Punkte an.

`affine_trans_point_3d` berechnet das Ergebnis der Anwendung der affinen Abbildung, die durch die homogene 3D-Transformations-Matrix `HomMat3D` gegeben ist, auf die Eingabepunkte (`Px`,`Py`,`Pz`). Die Ergebnispunkte werden in (`Qx`,`Qy`,`Qz`) zurückgeliefert.

`affine_trans_point_3d` dient somit zur Transformation eines 3D-Punktes (`Px`, `Py`, `Pz`) im Ausgangskoordinatensystem in das Zielkoordinatensystem. Die hierzu notwendige Translation und Rotation ist durch die 4x3 Transformations-Matrix `HomMat3D` bestimmt. Die Angabe der Werte erfolgt dabei zeilenweise, d.h. die jeweils ersten 3 Werte der drei Zeilen entsprechen der Rotationsmatrix $\mathcal{R}$ und die jeweils letzten Werte der drei Zeilen entsprechen dem Translationsvektor $\mathcal{T}$. Bei der Transformation wird der Punkt zunächst rotiert und dann wird der gedrehte Punkt transliert:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathcal{R} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}$$

Parameter

- ▷ **HomMat3D** (input_control) ..... affine3d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 12
- ▷ **Px** (input_control) ..... point3d.x(-array) ⇝ *real* / integer
  Eingabepunkt (x-Koordinate).
  Defaultwert : 64
  Wertevorschläge : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Px ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Py** (input_control) ..... point3d.y(-array) ⇝ *real* / integer
  Eingabepunkt (y-Koordinate).
  Defaultwert : 64
  Wertevorschläge : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Py ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Pz** (input_control) ..... point3d.z(-array) ⇝ *real* / integer
  Eingabepunkt (z-Koordinate).
  Defaultwert : 64
  Wertevorschläge : Pz ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Pz ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Qx** (output_control) ..... point3d.x(-array) ⇝ *real*
  Ausgabepunkt (x-Koordinate).

▷ **Qy** (output_control) ..... point3d.y(-array) ⇝ *real*
Ausgabepunkt (y-Koordinate).
▷ **Qz** (output_control) ..... point3d.z(-array) ⇝ *real*
Ausgabepunkt (z-Koordinate).

Ergebnis

`affine_trans_point_3d` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`affine_trans_point_3d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`hom_mat3d_translate`, `hom_mat3d_scale`, `hom_mat3d_rotate`, `pose_to_hom_mat3d`

Mögliche Nachfolgerfunktionen

`hom_mat3d_translate`, `hom_mat3d_scale`, `hom_mat3d_rotate`, `project_3d_point`

Modul

Basic operators

dvf_to_hom_mat2d (VectorField : : : HomMat2D)

Näherung eines Verschiebungsvektorfeldes durch eine Affine Abbildung.

`dvf_to_hom_mat2d` berechnet aus dem Verschiebungsvektorfeld `VectorField` eine affine Abbildung, die die unterliegende Transformation möglichst gut annähert. Diese wird in `HomMat2D` zurückgeliefert.

Wenn das Verschiebungsvektorfeld aus einem Originalbild I_{orig} und einem zweiten Bild I_{res} berechnet worden ist, so enthält die intern gespeicherte Transformationsmatrix (siehe `affine_trans_image`) eine Abbildung, die beschreibt, wie das erste Bild I_{orig} in das zweite Bild I_{res} transformiert werden kann.

Parameter

▷ **VectorField** (input_object) ..... singlechannel-image ⇝ *Hobject* : dvf
Eingabebild.
▷ **HomMat2D** (output_control) ..... affine2d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 6

Parallelisierungsinformation

`dvf_to_hom_mat2d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`optical_flow_match`

Mögliche Nachfolgerfunktionen

`affine_trans_image`

Alternativen

`vector_to_hom_mat2d`

Modul

Basic operators

hom_mat2d_compose (: : HomMat2DSecond, HomMat2DFirst : HomMat2DCompose)

Setze zwei homogene 2D-Transformations-Matrizen zusammen.

`hom_mat2d_compose` fügt zwei homogene 2D-Transformations-Matrizen zu einer neuen Matrix zusammen. Dabei wird die Abbildung, die durch `HomMat2DFirst` gegeben ist, zuerst ausgeführt und dann die Abbildung, die durch `HomMat2DSecond` gegeben ist, d.h. mit `HomMat2DFirst` = $\mathcal{H}_1$, `HomMat2DSecond` = $\mathcal{H}_2$ und `HomMat2DCompose` = $\mathcal{H}_c$ gilt:

$\mathcal{H}_c = \mathcal{H}_2 \cdot \mathcal{H}_1$

Parameter

- ▷ **HomMat2DSecond** (input_control) . . . . . affine2d-array ⇝ *real*
  Zweite Eingabe-Transformations-Matrix.
  Parameteranzahl : 6
- ▷ **HomMat2DFirst** (input_control) . . . . . affine2d-array ⇝ *real*
  Erste Eingabe-Transformations-Matrix.
  Parameteranzahl : 6
- ▷ **HomMat2DCompose** (output_control) . . . . . affine2d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 6

Ergebnis

hom_mat2d_compose liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat2d_compose ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Mögliche Nachfolgerfunktionen

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Basic operators

hom_mat2d_identity (: : : HomMat2DIdentity)

Erzeuge die homogene Transformations-Matrix der identischen 2D-Abbildung.

hom_mat2d_identity erzeugt die homogene 2×3 Transformations-Matrix HomMat2DIdentity, die die identische 2D-Abbildung beschreibt.

Parameter

- ▷ **HomMat2DIdentity** (output_control) . . . . . affine2d-array ⇝ *real*
  Transformations-Matrix.
  Parameteranzahl : 6

Ergebnis

hom_mat2d_identity liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat2d_identity ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Basic operators

hom_mat2d_invert (: : HomMat2D : HomMat2DInvert)

Invertiere eine homogene 2D-Transformations-Matrix.

hom_mat2d_invert invertiert die homogene 2D-Transformations-Matrix, die in HomMat2D übergeben wird. Die resultierende Matrix wird in HomMat2DInvert zurückgeliefert.

Parameter

▷ **HomMat2D** (input_control) . . . . . affine2d-array ⇝ *real*
Eingabe-Transformations-Matrix.
Parameteranzahl : 6

▷ **HomMat2DInvert** (output_control) . . . . . affine2d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 6

Ergebnis

`hom_mat2d_invert` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabematrix invertierbar ist.

Parallelisierungsinformation

`hom_mat2d_invert` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`

Mögliche Nachfolgerfunktionen

`hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`

Modul

Basic operators

hom_mat2d_rotate (: : HomMat2D, Phi, Px, Py : HomMat2DRotate)

Füge eine Rotation zu einer homogenen 2D-Transformations-Matrix hinzu.

`hom_mat2d_rotate` fügt zu einer homogenen 2D-Transformations-Matrix `HomMat2D` eine Rotation um den Winkel `Phi` hinzu. Der Punkt (`Px`,`Py`) ist dabei der Fixpunkt der Rotation. Die resultierende Matrix wird in `HomMat2DRotate` zurückgeliefert.

Parameter

▷ **HomMat2D** (input_control) . . . . . affine2d-array ⇝ *real*
Eingabe-Transformations-Matrix.
Parameteranzahl : 6

▷ **Phi** (input_control) . . . . . angle.rad ⇝ *real* / integer
Rotationswinkel.
Defaultwert : 0.78
Wertevorschläge : `Phi` ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
Typischer Wertebereich : 0 ≤ `Phi` ≤ 6.28318530718
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **Px** (input_control) . . . . . point.x ⇝ *real* / integer
Fixpunkt der Abbildung (x-Koordinate).
Defaultwert : 0
Wertevorschläge : `Px` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ `Px` ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Py** (input_control) . . . . . point.y ⇝ *real* / integer
Fixpunkt der Abbildung (y-Koordinate).
Defaultwert : 0
Wertevorschläge : `Py` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ `Py` ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **HomMat2DRotate** (output_control) . . . . . affine2d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 6

Ergebnis

`hom_mat2d_rotate` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`hom_mat2d_rotate` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`hom_mat2d_identity`, `hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`

Mögliche Nachfolgerfunktionen

`hom_mat2d_translate`, `hom_mat2d_scale`, `hom_mat2d_rotate`, `hom_mat2d_slant`

Modul

Basic operators

`hom_mat2d_scale ( : : HomMat2D, Sx, Sy, Px, Py : HomMat2DScale )`

Füge eine Skalierung zu einer homogenen 2D-Transformations-Matrix hinzu.

`hom_mat2d_scale` fügt zu einer homogenen 2D-Transformations-Matrix `HomMat2D` eine Skalierung um die Skalierungsfaktoren `Sx` und `Sy` hinzu. Der Punkt (`Px`,`Py`) ist dabei der Fixpunkt der Skalierung. Die resultierende Matrix wird in `HomMat2DScale` zurückgeliefert.

Parameter

▷ **`HomMat2D`** (input_control) . . . . . affine2d-array ⇝ *real*
Eingabe-Transformations-Matrix.
Parameteranzahl : `6`

▷ **`Sx`** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in x-Richtung.
Defaultwert : 2
Wertevorschläge : `Sx` ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 16}
Typischer Wertebereich : 0 ≤ `Sx` ≤ 1024
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.125
Restriktion : `Sx` ≠ `0`

▷ **`Sy`** (input_control) . . . . . number ⇝ *real* / integer
Skalierungsfaktor in y-Richtung.
Defaultwert : 2
Wertevorschläge : `Sy` ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 16}
Typischer Wertebereich : 0 ≤ `Sy` ≤ 1024
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.125
Restriktion : `Sy` ≠ `0`

▷ **`Px`** (input_control) . . . . . point.x ⇝ *real* / integer
Fixpunkt der Abbildung (x-Koordinate).
Defaultwert : 0
Wertevorschläge : `Px` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ `Px` ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **`Py`** (input_control) . . . . . point.y ⇝ *real* / integer
Fixpunkt der Abbildung (y-Koordinate).
Defaultwert : 0
Wertevorschläge : `Py` ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ `Py` ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **HomMat2DScale** (output_control) . . . . . affine2d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 6

Ergebnis

hom_mat2d_scale liefert den Wert 2 (H_MSG_TRUE), falls beide Skalierungsfaktoren verschieden von 0 sind.

Parallelisierungsinformation

hom_mat2d_scale ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Mögliche Nachfolgerfunktionen

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Basic operators

hom_mat2d_slant (: : HomMat2D, Theta, Axis, Px, Py : HomMat2DSlant)

Füge eine Scherung zu einer homogenen 2D-Transformations-Matrix hinzu.

hom_mat2d_slant fügt zu einer homogenen 2D-Transformations-Matrix HomMat2D eine Scherung um den Winkel Theta hinzu. Eine Scherung ist eine affine Abbildung, bei der eine Koordinatenachse festgehalten wird, während die andere Koordinatenachse um den Winkel Theta im Gegenuhrzeigersinn gedreht wird. Der Parameter Axis bestimmt, für welche Koordinatenachse die Scherung ausgeführt werden soll. Für Axis = **'x'** wird die x-Achse geschert, für Axis = **'y'** die y-Achse. Der Punkt (Px,Py) ist dabei der Fixpunkt der Scherung. Die resultierende Matrix wird in HomMat2DSlant zurückgeliefert.

Parameter

▷ **HomMat2D** (input_control) . . . . . affine2d-array ⇝ *real*
Eingabe-Transformations-Matrix.
Parameteranzahl : 6

▷ **Theta** (input_control) . . . . . angle.rad ⇝ *real* / integer
Winkel der Scherung.
Defaultwert : 0.78
Wertevorschläge : Theta ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
Typischer Wertebereich : 0 ≤ Theta ≤ 6.28318530718
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1

▷ **Axis** (input_control) . . . . . string ⇝ *string*
Koordinatenachse, die geschert wird.
Defaultwert : 'x'
Werteliste : Axis ∈ {'x', 'y'}

▷ **Px** (input_control) . . . . . point.x ⇝ *real* / integer
Fixpunkt der Abbildung (x-Koordinate).
Defaultwert : 0
Wertevorschläge : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ Px ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Py** (input_control) . . . . . point.y ⇝ *real* / integer
Fixpunkt der Abbildung (y-Koordinate).
Defaultwert : 0
Wertevorschläge : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ Py ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **HomMat2DSlant** (output_control) ..... affine2d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 6

Ergebnis

hom_mat2d_slant liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat2d_slant ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Mögliche Nachfolgerfunktionen

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Basic operators

hom_mat2d_to_affine_par (: : HomMat2D : Sx, Sy, Phi, Theta, Tx, Ty)

Berechnung der affinen Abbildungsparameter aus einer homogenen 2D-Transformations-Matrix.

hom_mat2d_to_affine_par berechnet aus einer homogenen 2D-Transformations-Matrix HomMat2D die zugehörigen affinen Abbildungsparameter. Die Parameter Sx und Sy geben an, wie stark die ursprünglichen x- und y-Achsen von der Transformation skaliert werden. Die beiden Skalierungsfaktoren sind immer positv. Der Winkel Theta beschreibt, wie stark die transformierten Koordinatenachsen von der Orthogonalität abweichen. Für Theta = 0 sind die beiden Koordinatenachsen orthogonal. Falls Theta $> \pi/2$, enthält die Transformation eine Spiegelung. Der Winkel Phi beschreibt die Rotation der x-Achse des transformierten Koordinatensystems gegenüber der x-Achse des ursprünglichen Koordinatensystems. Die Parameter Tx und Ty beschreiben die Verschiebung der beiden Koordinatensysteme. Die Matrix HomMat2D läßt sich mit folgender Operatorsequenz aus den sechs Abbildungsparametern erzeugen:

```
hom\_mat2d\_identity (HomMat2DIdentity)
hom\_mat2d\_scale (HomMat2DIdentity, SxRef, SyRef, 0, 0, HomMat2DScale)
hom\_mat2d\_slant (HomMat2DScale, ThetaRef, 'y', 0, 0, HomMat2DSlant)
hom\_mat2d\_rotate (HomMat2DSlant, PhiRef, 0, 0, HomMat2DRotate)
hom\_mat2d\_translate (HomMat2DRotate, TxRef, TyRef, HomMat2D)
```

Parameter

▷ **HomMat2D** (input_control) ..... affine2d-array ⇝ *real*
Eingabe-Transformations-Matrix.
Parameteranzahl : 6

▷ **Sx** (output_control) ..... real ⇝ *real*
Skalierung in x-Richtung.

▷ **Sy** (output_control) ..... real ⇝ *real*
Skalierung in y-Richtung.

▷ **Phi** (output_control) ..... angle.rad ⇝ *real*
Rotationswinkel.

▷ **Theta** (output_control) ..... angle.rad ⇝ *real*
Scherungswinkel.

▷ **Tx** (output_control) ..... point.x ⇝ *real*
Translation in x-Richtung.

▷ **Ty** (output_control) ..... point.y ⇝ *real*
Translation in y-Richtung.

Ergebnis

Falls die Matrix HomMat2D nicht ausgeartet ist, liefert hom_mat2d_to_affine_par den Wert 2 (H_MSG_TRUE) zurück. Ansonsten wird eine Ausnahmebehandlung durchgeführt.

Parallelisierungsinformation

hom_mat2d_to_affine_par ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

vector_to_hom_mat2d, vector_to_rigid, vector_to_similarity

Mögliche Nachfolgerfunktionen

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Basic operators

hom_mat2d_translate (: : HomMat2D, Tx, Ty : HomMat2DTranslate)

Füge eine Translation zu einer homogenen 2D-Transformations-Matrix hinzu.

hom_mat2d_translate fügt zu einer homogenen 2D-Transformations-Matrix HomMat2D eine Translation um den Vektor (Tx,Ty) hinzu. Die resultierende Matrix wird in HomMat2DTranslate zurückgeliefert.

Parameter

- ▷ **HomMat2D** (input_control) ..... affine2d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 6
- ▷ **Tx** (input_control) ..... point.x ⇝ *real* / integer
  Verschiebung in x-Richtung.
  Defaultwert : 64
  Wertevorschläge : Tx ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Tx ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Ty** (input_control) ..... point.y ⇝ *real* / integer
  Verschiebung in y-Richtung.
  Defaultwert : 64
  Wertevorschläge : Ty ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Ty ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **HomMat2DTranslate** (output_control) ..... affine2d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 6

Ergebnis

hom_mat2d_translate liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat2d_translate ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Mögliche Nachfolgerfunktionen

hom_mat2d_translate, hom_mat2d_scale, hom_mat2d_rotate, hom_mat2d_slant

Modul

Basic operators

hom_mat3d_compose (: : HomMat3DSecond, HomMat3DFirst : HomMat3DCompose)

Setze zwei homogene 3D-Transformations-Matrizen zusammen.

hom_mat3d_compose fügt zwei homogene 3D-Transformations-Matrizen zu einer neuen Matrix zusammen. Dabei wird die Abbildung, die durch HomMat3DFirst gegeben ist, zuerst ausgeführt und dann die Abbildung, die durch HomMat3DSecond gegeben ist, d.h. mit HomMat3DFirst = $\mathcal{H}_1$, HomMat3DSecond = $\mathcal{H}_2$ und HomMat3DCompose = $\mathcal{H}_c$ gilt:

$\mathcal{H}_c = \mathcal{H}_2 \cdot \mathcal{H}_1$

Für die Anwendung der homogenen 4x3 Transformations-Matrix auf die Transformation eines 3D-Punktes aus einem Ausgangskoordinatensystem in ein Zielkoordinatensystem bedeutet dies:

$$\begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} = \mathcal{H}_c \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \mathcal{H}_2 \cdot \mathcal{H}_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$
$$= \mathcal{H}_2 \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}$$

mit:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathcal{H}_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \mathcal{R}_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}_1$$

es gilt daher:

$$\begin{pmatrix} x_3 \\ y_3 \\ z_3 \end{pmatrix} = \mathcal{R}_2 \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + \mathcal{T}_2 = \mathcal{R}_2 \cdot \left(\mathcal{R}_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}_1 \right) + \mathcal{T}_2$$
$$= \mathcal{R}_2 \cdot \mathcal{R}_1 \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{R}_2 \cdot \mathcal{T}_1 + \mathcal{T}_2 = \mathcal{R}_c \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}_c$$

mit:

Rotation: $\mathcal{R}_c = \mathcal{R}_2 \cdot \mathcal{R}_2$
Translation: $\mathcal{T}_c = \mathcal{R}_2 \cdot \mathcal{T}_1 + \mathcal{T}_2$

Parameter

- ▷ **HomMat3DSecond** (input_control) ..... affine3d-array ⇝ *real*
  Zweite Eingabe-Transformations-Matrix.
  Parameteranzahl : 12
- ▷ **HomMat3DFirst** (input_control) ..... affine3d-array ⇝ *real*
  Erste Eingabe-Transformations-Matrix.
  Parameteranzahl : 12
- ▷ **HomMat3DCompose** (output_control) ..... affine3d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 12

Beispiel

```
/* read camera pose */
read_pose('campose.dat',CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat1)
/* multiply transformation matrices */
hom_mat3d_compose(HomTransMat2,HomTransMat1,HomTransMatMult).
```

Ergebnis

hom_mat3d_compose liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat3d_compose ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat3d_identity, hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate, pose_to_hom_mat3d

Mögliche Nachfolgerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Siehe auch

affine_trans_point_3d, hom_mat3d_identity, hom_mat3d_rotate, hom_mat3d_translate, pose_to_hom_mat3d, hom_mat3d_to_pose

Modul

Basic operators

hom_mat3d_identity (: : : HomMat3DIdentity)

Erzeuge die homogene Transformations-Matrix der identischen 3D-Abbildung.

hom_mat3d_identity erzeugt die homogene 3×4 Transformations-Matrix HomMat3DIdentity, die die identische 3D-Abbildung beschreibt.

Parameter

▷ **HomMat3DIdentity** (output_control) ..... affine3d-array ⇝ *real*
Transformations-Matrix.
Parameteranzahl : 12

Ergebnis

hom_mat3d_identity liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat3d_identity ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Alternativen

pose_to_hom_mat3d

Modul

Basic operators

hom_mat3d_invert (: : HomMat3D : HomMat3DInvert)

Invertiere eine homogene 3D-Transformations-Matrix.

hom_mat3d_invert invertiert die homogene 3D-Transformations-Matrix, die in HomMat3D übergeben wird. Die resultierende Matrix wird in HomMat3DInvert zurückgeliefert.

Für die Anwendung der homogenen 4x3 Transformations-Matrix auf die Transformation eines 3D-Punktes aus einem Ausgangskoordinatensystem in ein Zielkoordinatensystem bedeutet dies: Die invertierte Transformations-Matrix HomMat3DInvert beschreibt dann die umgekehrte Transformation, also vom ursprünglichen Zielkoordinatensystem in das ursprüngliche Ausgangskoordinatensystem. Die Transformation eines 3D-Punktes im Ausgangskoordinatensystem in ein Zielkoordinatensystem ist betimmt durch eine Rotation $\mathcal{R}$ und eine Translation $\mathcal{T}$ (siehe affine_trans_point_3d). Es gilt daher:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathcal{R}_{org} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}_{org}$$

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = {\mathcal{R}_{org}}^{-1} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} - \mathcal{T}_{org}$$

mit $\mathcal{R}_{inv} = {\mathcal{R}_{org}}^{-1}$ und $\mathcal{T}_{inv} = -\mathcal{T}_{org}$:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = \mathcal{R}_{inv} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + \mathcal{T}_{inv}$$

Parameter

▷ **HomMat3D** (input_control) ..... affine3d-array ⇝ *real*
Eingabe-Transformations-Matrix.
Parameteranzahl : 12

▷ **HomMat3DInvert** (output_control) ..... affine3d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 12

Beispiel

```
/* read camera pose */
read_pose('campose.dat',CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat)
/* invert transformation matrix */
hom_mat3d_invert(HomTransMat,HomTransMatInv).
```

Ergebnis

hom_mat3d_invert liefert den Wert 2 (H_MSG_TRUE), falls die Eingabematrix invertierbar ist.

Parallelisierungsinformation

hom_mat3d_invert ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate, pose_to_hom_mat3d

Mögliche Nachfolgerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate, hom_mat3d_to_pose

Siehe auch

affine_trans_point_3d, hom_mat3d_identity, hom_mat3d_rotate, hom_mat3d_translate, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Basic operators

hom_mat3d_rotate (: : HomMat3D, Phi, Axis, Px, Py, Pz : HomMat3DRotate)

Füge eine Rotation zu einer homogenen 3D-Transformations-Matrix hinzu.

hom_mat3d_rotate fügt zu einer homogenen 3D-Transformations-Matrix HomMat3D eine Rotation um die Axis-Achse um den Winkel Phi hinzu. Der Punkt (Px,Py,Pz) ist dabei der Fixpunkt der Rotation. Die resultierende Matrix wird in HomMat3DRotate zurückgeliefert.

Für die Anwendung der homogenen 4x3 Transformations-Matrix auf die Transformation eines 3D-Punktes aus einem Ausgangskoordinatensystem in ein Zielkoordinatensystem bedeutet dies (ohne Fixpunkt verschiebung): Mit hom_mat3d_rotate wird aus der Transformations-Matrix HomMat3D eine weitere Transformations-Matrix HomMat3DRotate generiert, wobei die Lage des Zielkoordinatensystems um die Axis-Achse gegen den Uhrzeigersinn um den Winkel Phi gedreht ist, bei Sicht in die positive Achsenrichtung.

Soll dagegen die Lage des Ausgangskoordinatensystems gedreht werden, so muß die Transformations-Matrix zunächst mit hom_mat3d_invert invertiert werden, dann die Drehung mit hom_mat3d_rotate ausgeführt werden und anschließend wieder invertiert werden.

Die Transformation eines 3D-Punktes im Ausgangskoordinatensystem in ein Zielkoordinatensystem ist betimmt durch eine Rotation $\mathcal{R}$ und eine Translation $\mathcal{T}$ (siehe affine_trans_point_3d). Es gilt daher:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathcal{R}_{rot} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}_{rot}$$
$$= (\mathcal{R}_{\phi} \cdot \mathcal{R}_{org}) \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + \mathcal{T}_{org}$$

mit

$$\mathcal{R}_\phi^x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix}, \quad \mathcal{R}_\phi^y = \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix}, \quad \mathcal{R}_\phi^z = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Parameter

- ▷ **HomMat3D** (input_control) ..... affine3d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 12
- ▷ **Phi** (input_control) ..... angle.rad ⇝ *real* / integer
  Rotationswinkel.
  Defaultwert : 0.78
  Wertevorschläge : Phi ∈ {0.1, 0.2, 0.3, 0.4, 0.78, 1.57, 3.14}
  Typischer Wertebereich : 0 ≤ Phi ≤ 6.28318530718
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
- ▷ **Axis** (input_control) ..... string ⇝ *string*
  Achse, um die gedreht wird.
  Defaultwert : ”x”
  Wertevorschläge : Axis ∈ {”x”, ”y”, ”z”}
- ▷ **Px** (input_control) ..... point3d.x ⇝ *real* / integer
  Fixpunkt der Abbildung (x-Koordinate).
  Defaultwert : 0
  Wertevorschläge : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Px ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Py** (input_control) ..... point3d.y ⇝ *real* / integer
  Fixpunkt der Abbildung (y-Koordinate).
  Defaultwert : 0
  Wertevorschläge : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Py ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Pz** (input_control) ..... point3d.z ⇝ *real* / integer
  Fixpunkt der Abbildung (z-Koordinate).
  Defaultwert : 0
  Wertevorschläge : Pz ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Pz ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **HomMat3DRotate** (output_control) ..... affine3d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 12

Beispiel

```
/* read camera pose */
read_pose('campose.dat',CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat)
/* rotate destination coordinate system around x-axis by 10 degree */
hom_mat3d_rotate(HomTransMat,10,'x',0,0,0,HomTransMatRot)
/* rotate source coordinate system around y-axis by 270 degree */
hom_mat3d_invert(HomTransMat,HomTransMatInv)
hom_mat3d_rotate(HomTransMatInv,270,'y',0,0,0,HomTransMatRot2)
hom_mat3d_invert(HomTransMatRot2,HomTransMat2).
```

Ergebnis

hom_mat3d_rotate liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat3d_rotate ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat3d_identity, hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Mögliche Nachfolgerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Siehe auch

hom_mat3d_invert, hom_mat3d_identity, hom_mat3d_rotate, hom_mat3d_translate, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Basic operators

hom_mat3d_scale (: : HomMat3D, Sx, Sy, Sz, Px, Py, Pz : HomMat3DScale)

Füge eine Skalierung zu einer homogenen 3D-Transformations-Matrix hinzu.

hom_mat3d_scale fügt zu einer homogenen 3D-Transformations-Matrix HomMat3D eine Skalierung um die Skalierungsfaktoren Sx und Sy hinzu. Der Punkt (Px,Py,Pz) ist dabei der Fixpunkt der Skalierung. Die resultierende Matrix wird in HomMat3DScale zurückgeliefert.

Parameter

- **HomMat3D** (input_control) ..... affine3d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 12
- **Sx** (input_control) ..... number ⇝ *real* / integer
  Skalierungsfaktor in x-Richtung.
  Defaultwert : 2
  Wertevorschläge : Sx ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  Typischer Wertebereich : 0 ≤ Sx ≤ 1024
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.125
  Restriktion : Sx ≠ 0
- **Sy** (input_control) ..... number ⇝ *real* / integer
  Skalierungsfaktor in y-Richtung.
  Defaultwert : 2
  Wertevorschläge : Sy ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  Typischer Wertebereich : 0 ≤ Sy ≤ 1024
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.125
  Restriktion : Sy ≠ 0
- **Sz** (input_control) ..... number ⇝ *real* / integer
  Skalierungsfaktor in z-Richtung.
  Defaultwert : 2
  Wertevorschläge : Sz ∈ {0.125, 0.25, 0.5, 1, 2, 4, 8, 112}
  Typischer Wertebereich : 0 ≤ Sz ≤ 1024
  Minimale Schrittweite : 0.001
  Empfohlene Schrittweite : 0.125
  Restriktion : Sz ≠ 0

▷ **Px** (input_control) . . . . . point3d.x ⇝ *real* / integer
Fixpunkt der Abbildung (x-Koordinate).
Defaultwert : 0
Wertevorschläge : Px ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ Px ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Py** (input_control) . . . . . point3d.y ⇝ *real* / integer
Fixpunkt der Abbildung (y-Koordinate).
Defaultwert : 0
Wertevorschläge : Py ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ Py ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **Pz** (input_control) . . . . . point3d.z ⇝ *real* / integer
Fixpunkt der Abbildung (z-Koordinate).
Defaultwert : 0
Wertevorschläge : Pz ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
Typischer Wertebereich : 0 ≤ Pz ≤ 1024
Minimale Schrittweite : 1
Empfohlene Schrittweite : 10

▷ **HomMat3DScale** (output_control) . . . . . affine3d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 12

Ergebnis

hom_mat3d_scale liefert den Wert 2 (H_MSG_TRUE), falls alle drei Skalierungsfaktoren verschieden von 0 sind.

Parallelisierungsinformation

hom_mat3d_scale ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat3d_identity, hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Mögliche Nachfolgerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Modul

Basic operators

hom_mat3d_translate (: : HomMat3D, Tx, Ty, Tz : HomMat3DTranslate)

Füge eine Translation zu einer homogenen 3D-Transformations-Matrix hinzu.

hom_mat3d_translate fügt zu einer homogenen 3D-Transformations-Matrix HomMat3D eine Translation um den Vektor (Tx,Ty,Tz) hinzu. Die resultierende Matrix wird in HomMat3DTranslate zurückgeliefert.

Für die Anwendung der homogenen 4x3 Transformations-Matrix auf die Transformation eines 3D-Punktes aus einem Ausgangskoordinatensystem in ein Zielkoordinatensystem bedeutet dies: Mit hom_mat3d_translate wird aus der Transformations-Matrix HomMat3D eine weitere Transformations-Matrix HomMat3DTranslate generiert, wobei die Lage des Ursprungs des Zielkoordinatensystems um die Beträge der Translation (Tx,Ty,Tz) in jeweils die entsprechenden negativen Achsenrichtungen verschoben ist.

Soll dagegen die Lage des Ausgangskoordinatensystems transliert werden, so muß die Transformations-Matrix zunächst mit hom_mat3d_invert invertiert werden, dann die Translation mit hom_mat3d_translate ausgeführt werden und anschließend wieder invertiert werden.

Die Transformation eines 3D-Punktes im Ausgangskoordinatensystem in ein Zielkoordinatensystem ist betimmt durch eine Rotation $\mathcal{R}$ und eine Translation $\mathcal{T}$ (siehe affine_trans_point_3d). Es gilt daher:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathcal{R}_{trans} \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} + \mathcal{T}_{trans}$$
$$= \mathcal{R}_{org} \cdot \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} + (\mathcal{T}_{org} + \mathcal{T}_{\delta})$$

mit

$$\mathcal{T}_{\delta} = \begin{pmatrix} t\_x \\ t\_y \\ t\_z \end{pmatrix}$$

Parameter

- ▷ **HomMat3D** (input_control) ..... affine3d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 12
- ▷ **Tx** (input_control) ..... point3d.x ⇝ *real* / integer
  Verschiebung in x-Richtung.
  Defaultwert : 64
  Wertevorschläge : Tx ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Tx ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Ty** (input_control) ..... point3d.y ⇝ *real* / integer
  Verschiebung in y-Richtung.
  Defaultwert : 64
  Wertevorschläge : Ty ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Ty ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **Tz** (input_control) ..... point3d.z ⇝ *real* / integer
  Verschiebung in z-Richtung.
  Defaultwert : 64
  Wertevorschläge : Tz ∈ {0, 16, 32, 64, 128, 256, 512, 1024}
  Typischer Wertebereich : 0 ≤ Tz ≤ 1024
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 10
- ▷ **HomMat3DTranslate** (output_control) ..... affine3d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 12

Beispiel

```
/* read camera pose */
read_pose('campose.dat',CamPose)
/* transform pose to transformation matrix */
pose_to_hom_mat3d(CamPose,HomTransMat)
/* translate destination coordinate along x-axis by 0.4 m */
hom_mat3d_translate(HomTransMat,-0.4,0,0,HomTransMatTransl)
/* translate source coordinate along y-axis by -0.03 m */
hom_mat3d_invert(HomTransMat,HomTransMatInv)
hom_mat3d_translate(HomTransMatInv,0,0.03,0,HomTransMatTransl2)
hom_mat3d_invert(HomTransMatTransl2,HomTransMat2).
```

Ergebnis

hom_mat3d_translate liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

hom_mat3d_translate ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat3d_identity, hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Mögliche Nachfolgerfunktionen

hom_mat3d_translate, hom_mat3d_scale, hom_mat3d_rotate

Siehe auch

hom_mat3d_invert, hom_mat3d_identity, hom_mat3d_rotate, hom_mat3d_translate, pose_to_hom_mat3d, hom_mat3d_to_pose, hom_mat3d_compose

Modul

Basic operators

vector_angle_to_rigid (: : Row1, Column1, Angle1, Row2, Column2, Angle2 : HomMat2D)

Berechnung einer starren affinen Abbildung aus Punkten und Winkeln.

vector_angle_to_rigid berechnet aus einer Punktkorrespondenz und zwei zugehörigen Winkeln eine starre affine Abbildung. Die Koordinaten des ursprünglichen Punktes werden in (Row1,Column1) übergeben, der zugehörige Winkel in Angle1. Die Koordinaten des transformierten Punktes werden in (Row2,Column2) übergeben, der zugehörige Winkel in Angle2. Der Operator vector_angle_to_rigid ist insbesondere sinnvoll, um aus den Ergebnissen von Pattern-Matching-Operatoren (best_match_rot und best_match_rot_mg) eine starre Abbildung zu konstruieren, die ein Referenzbild in das aktuelle Bild transformiert oder (wenn die Parameter in der umgekehrten Reihenfolge übergeben werden) das aktuelle Bild in das Referenzbild transformiert. Die Abbildung wird in HomMat2D zurückgeliefert und kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Parameter

- ▷ **Row1** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des Ausgangspunkts.
- ▷ **Column1** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des Ausgangspunkts.
- ▷ **Angle1** (input_control) ..... angle.rad ⇝ *real* / integer
  Winkel des Ausgangspunkts.
- ▷ **Row2** (input_control) ..... point.y ⇝ *real* / integer
  Zeilenkoordinate des transformierten Punkts.
- ▷ **Column2** (input_control) ..... point.x ⇝ *real* / integer
  Spaltenkoordinate des transformierten Punkts.
- ▷ **Angle2** (input_control) ..... angle.rad ⇝ *real* / integer
  Winkel des transformierten Punkts.
- ▷ **HomMat2D** (output_control) ..... affine2d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 6

Beispiel

```
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)
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)
```

Parallelisierungsinformation

`vector_angle_to_rigid` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`best_match_rot_mg`, `best_match_rot`

Mögliche Nachfolgerfunktionen

`hom_mat2d_invert`, `affine_trans_image`, `affine_trans_region`, `affine_trans_contour_xld`, `affine_trans_polygon_xld`, `affine_trans_point_2d`

Alternativen

`vector_to_rigid`

Siehe auch

`dvf_to_hom_mat2d`

Modul

Basic operators

vector_to_hom_mat2d (: : Rows1, Columns1, Rows2, Columns2 : HomMat2D)

Näherung einer affinen Abbildung aus Punktkorrespondenzen.

`vector_to_hom_mat2d` erzeugt aus (mindestens drei) Punktkorrespondenzen eine Näherung für eine affine Abbildung. Die Punktkorrespondenzen werden in den Tupeln (`Rows1`,`Columns1`) und (`Rows2`,`Columns2`) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist überbestimmt, falls mehr als drei Punktkorrespondenzen übergeben werden. In diesem Fall ist die zurückgelieferte Abbildung diejenige, die die Abstände zwischen den Eingabepunkten (`Rows1`,`Columns1`) und den Zielpunkten (`Rows2`,`Columns2`) minimiert. Die Abbildung wird in `HomMat2D` zurückgeliefert und kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. `affine_trans_image`, verwendet werden.

Parameter

- ▷ **Rows1** (input_control) ..... point.y-array ⇝ *real*
  Zeilenindizes der Startpunkte.
- ▷ **Columns1** (input_control) ..... point.x-array ⇝ *real*
  Spaltenindizes der Startpunkte.
- ▷ **Rows2** (input_control) ..... point.y-array ⇝ *real*
  Zeilenindizes der Endpunkte.
- ▷ **Columns2** (input_control) ..... point.x-array ⇝ *real*
  Spaltenindizes der Endpunkte.
- ▷ **HomMat2D** (output_control) ..... affine2d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.

Parameteranzahl : 6

Parallelisierungsinformation

`vector_to_hom_mat2d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`affine_trans_image`

Alternativen

`dvf_to_hom_mat2d`

Siehe auch

`affine_trans_image`, `optical_flow_match`

Modul

Basic operators

vector_to_rigid (: : Rows1, Columns1, Rows2, Columns2 : HomMat2D)

Näherung einer starren affinen Abbildung aus Punktkorrespondenzen.

vector_to_rigid erzeugt aus (mindestens zwei) Punktkorrespondenzen eine Näherung für eine starre affine Abbildung. Die Punktkorrespondenzen werden in den Tupeln (Rows1,Columns1) und (Rows2,Columns2) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist durch die mindestens zwei Punktkorrespondenzen immer überbestimmt. Deswegen ist die zurückgelieferte Abbildung diejenige, die die Abstände zwischen den Eingabepunkten (Rows1,Columns1) und den Zielpunkten (Rows2,Columns2) minimiert. Die Abbildung wird in HomMat2D zurückgeliefert und kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Parameter

- ▷ **Rows1** (input_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der originalen Punkte.
- ▷ **Columns1** (input_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der originalen Punkte.
- ▷ **Rows2** (input_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der transformierten Punkte.
- ▷ **Columns2** (input_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der transformierten Punkte.
- ▷ **HomMat2D** (output_control) ..... affine2d-array ⇝ *real*
  Ausgabe-Transformations-Matrix.
  Parameteranzahl : 6

Parallelisierungsinformation

vector_to_rigid ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

affine_trans_image, affine_trans_region, affine_trans_contour_xld, affine_trans_polygon_xld, affine_trans_point_2d

Alternativen

vector_to_hom_mat2d, vector_to_similarity

Siehe auch

dvf_to_hom_mat2d

Modul

Basic operators

vector_to_similarity (: : Rows1, Columns1, Rows2, Columns2 : HomMat2D)

Näherung einer Ähnlichkeitsabbildung aus Punktkorrespondenzen.

vector_to_similarity erzeugt aus (mindestens zwei) Punktkorrespondenzen eine Näherung für eine Ähnlichkeitsabbildung. Die Punktkorrespondenzen werden in den Tupeln (Rows1,Columns1) und (Rows2,Columns2) übergeben, wobei korrespondierende Punkte an denselben Indexpositionen stehen müssen. Die Transformation ist überbestimmt, falls mehr als zwei Punktkorrespondenzen übergeben werden. In diesem Fall ist die zurückgelieferte Abbildung diejenige, die die Abstände zwischen den Eingabepunkten (Rows1,Columns1) und den Zielpunkten (Rows2,Columns2) minimiert. Die Abbildung wird in HomMat2D zurückgeliefert und kann direkt mit den Operatoren, die Daten mit affinen Abbildungen transformieren, z.B. affine_trans_image, verwendet werden.

Parameter

- ▷ **Rows1** (input_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinaten der originalen Punkte.
- ▷ **Columns1** (input_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinaten der originalen Punkte.

▷ **Rows2** (input_control) . . . . . point.y-array ⇝ *real*
Zeilenkoordinaten der transformierten Punkte.

▷ **Columns2** (input_control) . . . . . point.x-array ⇝ *real*
Spaltenkoordinaten der transformierten Punkte.

▷ **HomMat2D** (output_control) . . . . . affine2d-array ⇝ *real*
Ausgabe-Transformations-Matrix.
Parameteranzahl : 6

Parallelisierungsinformation

`vector_to_similarity` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`affine_trans_image`, `affine_trans_region`, `affine_trans_contour_xld`, `affine_trans_polygon_xld`, `affine_trans_point_2d`

Alternativen

`vector_to_hom_mat2d`, `vector_to_rigid`

Siehe auch

`dvf_to_hom_mat2d`

Modul

Basic operators

12.2 Barcode

decode_1d_bar_code (: : BarCodeElements, BarCodeDescr : Characters, Reference, IsCorrect)

Decodieren einer Sequenz an Barcode-Elementen.

`decode_1d_bar_code` decodiert eine Sequenz von Strichen und Lücken (Elemente) die mit `find_1d_bar_code` oder `get_1d_bar_code` gefunden wurden. Als Eingabe dient die Dicke aller Elemente (in Pixeln) in nicht diskretisierter Form. Es kann jedoch auch die diskrete Form der Elemente verwendet werden wie sie von `discrete_1d_bar_code` erzeugt wird.

Das Ergebnis ist die Sequenz der Zeichen `Characters` und der Referenznummern `Reference`. Zusätzlich wird der Parity-Test für die gefundenen Zeichen durchgeführt und das Ergebnis in `IsCorrect` übergeben. Für den Fall das alle Zeichen als Nutzzeichen verwendet werden, ist der Wert von `IsCorrect` zu ignorieren.

Achtung

Bei Barcodes vom Typ Pharmacode kann die Leserichtung nicht aus den Barcodedaten bestimmt werden, da der Barcode keine Extrazeichen (wie Start- oder Stop- oder Prüfzeichen) spezifiziert, die eine Bestimmung der Leserichtung ermöglichen, so daß der Barcode immer in beiden Richtungen gelesen werden kann. Für PharmaCodes werden deshalb die Ergebnisse der Dekodierung in beide Leserichtungen als zwei Werte sowohl in `Characters` als auch in `Reference` zurückgeliefert. Die Entscheidung, welches Element die richtige Leserichtung beinhaltet muß vom Aufrufer anhand der Orientierung des Barcodes (wie sie im Parameter `Orientation` bei `find_1d_bar_code` und `find_1d_bar_code_region` zurückgeliefert wird) entschieden werden. Dabei entspricht das jeweils erste Element von `Characters` und `Reference` der Standardleserichtung in der Orientierung, die durch `Orientation` kodiert wird, und das jeweils zweite Element der entgegengesetzten Leserichtung. Falls z.B. die Orientierung 0 ist, entspricht der erste Wert der Leserichtung von rechts nach links. Falls die Orientierung $\pi/2$ ist, entspricht der erste Wert der Leserichtung von oben nach unten.

Parameter

▷ **BarCodeElements** (input_control) . . . . . number-array ⇝ *real*
Breite der Elemente des Barcodes.

▷ **BarCodeDescr** (input_control) . . . . . string-array ⇝ *string* / integer / real
Beschreibung eines Barcode-Typs

▷ **Characters** (output_control) . . . . . string-array ⇝ *string*
Decodierte Zeichen in Standard-Interpretation.

▷ **Reference** (output_control) . . . . . integer-array ⇝ *integer*
Decodierte Zeichen als Zahlenwerte.

▷ **IsCorrect** (output_control) ..... integer ⇝ *integer*
Information, ob es sich um einen korrekten Barcode handelt.
Werteliste : IsCorrect ∈ {0, 1}

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
HTuple   BarcodeFound,Elements,Orientation;
HTuple   Characters,Reference,IsCorrect;
Hobject  Image,CodeRegion;

gen_1d_bar_code_descr("EAN 13",13,13,&BarCodeDescr);
find_1d_bar_code(Image,&CodeRegion,BarCodeDescr,empty,empty,
                 &BarcodeFound,&Elements,&Orientation);
if (BarcodeFound[0].l)
{
  decode_1d_bar_code(Elements,BarCodeDescr,
                     &Characters,&Reference,&IsCorrect);
  if (IsCorrect[0].l)
    for (int i=0; i<Characters.Num(); i++)
    {
      char *value = Characters[i];
    }
}
```

Ergebnis

decode_1d_bar_code liefert den Wert 2 (H_MSG_TRUE), sofern die übergebene Barcodebeschreibung korrekt ist und eine interpretierbare Elementliste übergeben wird.

Parallelisierungsinformation

decode_1d_bar_code ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

find_1d_bar_code, get_1d_bar_code

Modul

Barcode reader

decode_2d_bar_code (: : BarCodeDescr, BarCodeDimension, BarCodeData : SymbolCharacters, CorrSymbolData, DecodedData, DecodingError, StructuredAppend)

Decodieren von 2D-Barcode-Daten.

decode_2d_bar_code dekodiert binäre 2D-Barcode-Daten, die mit Hilfe von get_2d_bar_code oder get_2d_bar_code_pos gelesen wurden. Der Parameter BarCodeData enthält die Datenwerte, BarCodeDimension die Breite und Höhe des Datenfeldes und einen Symbol-Index, BarCodeDescr die Beschreibung der Barcodeklasse.

Zuerst werden die binären Daten des Symbols vollständig in einen Strom von 8-Bit-Zeichen umgewandelt (SymbolCharacters). Dieser Datenstrom besteht aus den eigentlichen (möglicherweise fehlerbehafteten) Daten sowie zusätzlichen Zeichen für die Fehlerkorrektur. Konnte aus diesem Datenstrom ein fehlerfreier Datensatz rekonstruiert werden, steht dieser, immer noch in kodierter Form, in CorrSymbolData. DecodingError enthält die Anzahl der korrigierten Fehler oder einen negativen Fehlercode, falls eine Korrektur nicht möglich war.

In einem letzten Schritt werden die Daten dekodiert und als Tupel von ASCII-Zeichen in DecodedData abgelegt. Ist das untersuchte Symbol Bestandteil einer Gruppe von zusammengehörigen Symbolen (ECC 200: “Structured Append”), enthält der Parameter StructuredAppend die Nummer der Symbols in der Reihe, die Anzahl zusammengehöriger Symbole sowie eine Zahl als Identifikator der Gruppe. Ansonsten steht in den ersten beiden Feldern von StructuredAppend jeweils eine **1**.

Parameter

- ▷ **BarCodeDescr** (input_control) . . . . . string-array ⇝ *string* / integer / real
  Beschreibung der 2D-Barcodeklasse
- ▷ **BarCodeDimension** (input_control) . . . . . integer-array ⇝ *integer*
  Tupel mit der Dimension des untersuchten Symbols. Bei ECC 200: Datenfeldbreite, -höhe, Symbol-Index.
- ▷ **BarCodeData** (input_control) . . . . . integer-array ⇝ *integer*
  Tupel mit der Datenwerten des untersuchten Symbols.
- ▷ **SymbolCharacters** (output_control) . . . . . string-array ⇝ *string*
  Symbol-Rohdaten (Daten- und Fehlerzeichen).
- ▷ **CorrSymbolData** (output_control) . . . . . integer-array ⇝ *integer*
  Fehlerbereinigte Symboldaten.
- ▷ **DecodedData** (output_control) . . . . . integer-array ⇝ *integer*
  Dekodierte Zeichen als Zahlenwerte.
- ▷ **DecodingError** (output_control) . . . . . integer ⇝ *integer*
  Anzahl der Fehler bei der Dekodierung.
- ▷ **StructuredAppend** (output_control) . . . . . integer-array ⇝ *integer*
  Bei Zugehörigkeit des Symbols zu einer Gruppe von Symbolen: Position in der Gruppe, Anzahl der Symbole, Gruppen-ID.

Ergebnis

Der Operator `decode_2d_bar_code` signalisiert über den Fehlercode sowohl, daß inkorrekte Parameter übergeben wurden, als auch, daß ein Fehler beim Dekodieren des Datenstroms aufgetreten ist. Der Fehlercode 8812 bedeutet hierbei, qdaß der dekodierte Datenstrom ein ungültiges Datenwort enthielt. Das Auftreten benutzerdefinierter Steuerwörter führt zu einem Abbruch mit dem Fehlercode 8813.

Parallelisierungsinformation

`decode_2d_bar_code` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`get_2d_bar_code`

Modul

Barcode reader

`discrete_1d_bar_code ( : : Elements, BarCodeDescr : DisrecteBarCode )`

Aus den Breiten einzelner Elemente einen diskreten Barcode erzeugen.

`discrete_1d_bar_code` wandelt eine Liste von Elementdicken, wie sie von `find_1d_bar_code` und `get_1d_bar_code` gewonnen werden, in einen diskreten Barcode um. Für jedes Element des Barcodes wird also die Anzahl von Modulen bestimmt.

Dieser Operator wird verwendet, falls der Barcode-Typ bei `decode_1d_bar_code` nicht bekannt ist und der Anwender selbst die Dekodierung durchführen möchte. In diesem Fall wird `gen_1d_bar_code_descr_gen` zur Erzeugung der Barcode-Beschreibung und `find_1d_bar_code` zum Finden des Barcodes und zur Erzeugung der Elementliste verwendet. Mit `discrete_1d_bar_code` kann dann die Liste der Elementbreiten in die diskrete Form umgesetzt werden.

Parameter

- ▷ **Elements** (input_control) . . . . . number-array ⇝ *real*
  Liste der Elementdicken des Barcodes
- ▷ **BarCodeDescr** (input_control) . . . . . string-array ⇝ *string* / integer / real
  Beschreibung eines Barcode-Typs
- ▷ **DisrecteBarCode** (output_control) . . . . . number-array ⇝ *integer*
  Dicke der Elemente als Vielfache eines Moduls.

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
```

```
HTuple   BarcodeFound,Elements,Orientation;
HTuple   DiscreteBarCode;
Hobject  Image,CodeRegion;

gen_1d_bar_code_descr_gen(20,40,2,empty,empty,-1.0,"false",&BarCodeDescr);
find_1d_bar_code(Image,&CodeRegion,BarCodeDescr,empty,empty,
                 &BarcodeFound,&Elements,&Orientation);
if (BarcodeFound[0].l)
{
  discrete_1d_bar_code(Elements,BarCodeDescr,&DiscreteBarCode);
  for (int i=0; i<DiscreteBarCode.Num(); i++)
  {
    int NumModules = DiscreteBarCode[i];
  }
}
```

Ergebnis

`discrete_1d_bar_code` liefert den Wert 2 (H_MSG_TRUE), sofern die übergebene Barcodebeschreibung korrekt ist und eine umsetzbare Elementliste verwendet wird.

Parallelisierungsinformation

`discrete_1d_bar_code` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`find_1d_bar_code`, `get_1d_bar_code`

Siehe auch

`decode_1d_bar_code`

Modul

Barcode reader

find_1d_bar_code (Image : CodeRegion : BarCodeDescr, GenericName, GenericValue : BarcodeFound, BarCodeElements, Orientation)

Einen Barcode in einem Bild finden.

`find_1d_bar_code` sucht einen Barcode in einem Bild. Übergeben wird die Barcode Beschreibung die mit `gen_1d_bar_code_descr` oder `gen_1d_bar_code_descr_gen` erzeugt wurde. Hierdurch wird festgelegt, welche Art von Barcode gefunden werden soll.

Als Ergebnis werden die Dicken der Elemente und die Orientierung des Barcodes zurückgegeben. Zusätzlich erhält man die Region, in der der Barcode liegt. Falls ein Barcode gefunden wurde, liefert `BarcodeFound` den Wert 1, sonst 0.

Zur Steuerung der internen Bildverarbeitung können über die Parameter `GenericName` und `GenericValue` gezielt Steuerparameter übergeben werden. Dies geschieht, indem bei `GenericName` die Liste der Namen der Parameter als Strings übergeben werden. Bei `GenericValue` werden die zugehörigen Werte an den gleichen Indexpositionen übergeben. Normalerweise ist es nicht nötig, die Werte zu ändern. Dies ist nur bei schwierigen Aufnahmebedingungen oder bei speziellen Barcodes empfohlen. Im Einzelnen handelt es sich hierbei um folgende Werte:

’amplitude_sobel’ Mindestamplitude für die Kantensuche: Die Barcode-Suche basiert im ersten Schritt auf dem Sobelfilter. Dies hat gegenüber einem Schwellenwertverfahren den Vorteil, daß das Bild ungleichmäßig ausgeleuchtet sein kann. Im Falle eines sehr schlechten Kontrastes (Unterschied zwischen hellen und dunklen Elementen des Barcodes) kann dieser Wert kleiner gewählt werden, um alle Teile zu finden. Falls der Kontrast sehr gut ist, kann der Wert auch größer gewählt werden, um die Laufzeit zu verbessern. Die Wirkung kann mit dem Operator `sobel_dir` mit dem Kantentyp ’sum_abs’ getestet werden.

Vorbesetzung: 70

Verwendung: Suche der Barcode-Region

'min_size_element' Bei der Analyse von Objekten, die (Teile von) Elemente(n) sein können, wird dieser Parameter zum Unterdrücken von sehr kleinen Objekten verwendet. Es ist zu beachten, daß bei schlechter Bildqualität ein Element zunächst in mehrere Bruchstücke zerfallen kann. Der Parameter muß hier größer als das kleinste Bruchstück gewählt werden. Bei guter Bildqualität kann der Wert erhöht werden. Dies verbessert die Laufzeit. Der Wert muß aber immer kleiner als die Länge eines Elementes (in Pixeln) sein. Ist der Wert zu groß, kann der gefundene Barcode unvollständig sein.

Vorbesetzung: 30

Verwendung: Suche der Barcode-Region

'max_size_element' Analog zur Mindestgröße von Barcode-Teilen werden mit diesem Parameter zu große Teile unterdrückt. Es ist zu beachten, daß bei geringer Auflösung mehrere Elemente bei der Vorverarbeitung verschmelzen können. Der Wert ist also entsprechend groß zu wählen. Weiterhin sollte der Wert kleiner sein als die Größe des gesamten Barcodes.

Vorbesetzung: 15000

Verwendung: Suche der Barcode-Region

'angle_range' Dieser Parameter hat die gleiche Funktion wie der Parameter 'sum_angles', nur wird er bei der Suche nach Elementen des Barcodes verwendet. Er hat somit Einfluß auf die Schätzung der Ausrichtung der Elemente. Bei schlechten Aufnahmen muß der Wert groß sein. Kleinere Werte haben eine kürzere Laufzeit zur Folge. Der Wert wird in Grad angegeben. Dieser Parameter braucht nicht geändert zu werden.

Vorbesetzung: 24

Verwendung: Suche der Barcode-Region

'correct_angle' Dieser Wert hat Einfluß auf die Entscheidung, ob eine Region ein Barcode-Element sein kann. Hierzu wird die Orientierung aller Randpunkte eines potentiellen Elementes bestimmt. Wenn genügend Punkte die gleiche Orientierung haben wird die Region als Elementkandidat akzeptiert. Eine Erhöhung des Wertes schränkt die Auswahl stärker ein und verbessert damit die Laufzeit. Um so schlechter die Bildqualität ist, um so kleiner muß der Wert sein.

Vorbesetzung: 0.5 (entspricht 50 Prozent)

Verwendung: Suche der Barcode-Region

'dilation_factor' Zunächst werden die einzelnen Elemente des Barcodes gesucht, die dann anschließend zu dem gesamten Barcode (`CodeRegion`) zusammengefaßt werden. Hierzu wird der morphologische Operator Closing verwendet. Das Maß der Dilatation wird automatisch geschätzt. Sollte diese Schätzung zu hoch oder zu niedrig sein, kann dies mit dem Parameter korrigiert werden. Dies kann bei Barcodes mit großen Lücken (Wert erhöhen) oder bei Barcodes die zu dicht an anderen Objekten liegen (Wert erniedrigen) nötig sein. Der Parameter ist unabhängig von der Bildqualität.

Vorbesetzung: 1

Verwendung: Suche der Barcode-Region

'sum_angles' Dieser Parameter wird für die abschließende Bestimmung der Orientierung des Barcodes verwendet (Wert von `Orientation`). Hierzu werden alle Kantenrichtungen, die innerhalb des Barcodes auftreten, bestimmt. Zunächst wird die häufigste Orientierung bestimmt. Unter Miteinbeziehung der Nachbarorientierungen wird das Ergebnis durch Mittelung berechnet. Mit dem Parameter wird der Winkelbereich der mit einbezogenen Nachbarorientierungen festlegt. Der Wert wird in Grad angegeben. Dieser Parameter braucht nicht geändert zu werden.

Vorbesetzung: 40

Verwendung: Suche der Barcode-Region

'min_area_bar_code' Dieser Wert dient zur abschließenden Beurteilung der potentiellen Barcode-Region. Der Parameter gibt die Mindestfläche der Gesamtregion vor. Werden zu kleine Regionen gefunden, kann dieser Wert erhöht werden. Der Parameter ist dann von Bedeutung, wenn kleine Regionen mit gleicher Ausrichtung wie die Barcode-Elemente im Bild vorhanden sind. Der Parameter ist unabhängig von der Bildqualität.

Vorbesetzung: 1000

Verwendung: Suche der Barcode-Region

'sigma_project' Für die Extraktion der Barcode-Elemente werden die Grauwerte der Barcode-Region in Richtung der Elemente projeziert. Die so erhaltenen Daten werden vor der Dickenbestimmung geglättet. Bei großen Abständen zwischen den Elementen kann die Glättung erhöht werden, um Scheinstrukturen zu unterdrücken. Der Wert sollte auf jeden Fall größer als 0.5 sein.

Vorbesetzung: 0.7

Verwendung: Extraktion der Elementdicken

'amplitude_project' Bei der Bestimmung der Elementdicken aus den Grauwertprojektionen bestimmt dieser Parameter die Mindestamplitude (Grauwertunterschiede) bei den Übergängen von Hell nach Dunkel und Dunkel nach Hell. Bei starken Störungen (Flecken) kann dieser Wert erhöht werden, um Scheinstrukturen zu unterdrücken. Bei sehr schwachen Kontrasten (z.B. bei einer geringen Auflösung und sehr dünnen Elementen) muß der Wert klein sein.

Vorbesetzung: 3

Verwendung: Extraktion der Elementdicken

'width_project' Die Grauwertprojektionen werden nur in dem inneren Bereich der Barcode-Region bestimmt. Der Parameter gibt die halbe Dicke dieses Bereiches an. Falls der Winkel der Barcode-Region nicht gut geschätzt wird, sollte dieser Wert klein sein. Falls Störungen im Barcode auftreten (z.B. Kratzer, Flecken), kann dieser Wert erhöht werden, um diese Störungen zu überbrücken. Bei sehr schmalen Barcodes sollte der Wert gesenkt werden.

Vorbesetzung: 25

Verwendung: Extraktion der Elementdicken

'add_length_project' Da die Barcode-Region manchmal zu kurz extrahiert wird, verlängert dieser Parameter die Region in beide Richtungen um den angegebenen Betrag in Pixeln. Falls andere Objekte in der Nähe des Barodes liegen, kann dies zu einer Extraktion von falschen Elementen führen. In diesem Fall sollte der Wert verkleinert werden. Werden die Anfangs- und Endelemente nicht gefunden, ist der Wert zu erhöhen.

Vorbesetzung: 5

Verwendung: Extraktion der Elementdicken

'interpolation_project' Mit diesem Parameter wird die Art der Grauwertinterpolation bei der Grauwertprojektion festgelegt. Bei einem Wert von 1 wird eine bilineare Interpolation verwendet. Bei dem Wert 0 wird keine Interpolation verwendet. Ohne Interpolation ist die Berechnung schneller, jedoch ist die Qualität nicht so gut und es kann zu Fehlmessungen kommen.

Vorbesetzung: 1

Verwendung: Extraktion der Elementdicken

'max_extra_elements' Dieser Wert dient dazu, eventuell in der Umbebung des Barcodes extrahierte, zu den Strichen des Barcodes parallele Elemente zu eliminieren, und so eine korrekte Dekodierung zu ermöglichen. Der Wert gilt für beide Enden des Barcodes getrennt, so daß insgesamt 2*max_extra_elements Elemente (Striche und Lücken) eliminiert werden können. Auf jeder Seite des Barcodes können also bis zu max_extra_elements/2 Striche eliminiert werden.

Vorbesetzung: 6

Verwendung: Extraktion der Elementdicken

Es ist zu beachten, daß die beschriebenen Parameter im Normalfall nicht verändert werden sollten. Eine Veränderung erfordert Erfahrung in der Bildverarbeitung.

Parameter

- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Bild, das Barcode enthält.
- ▷ **`CodeRegion`** (output_object) ..... region ⇝ *Hobject*
  Region des Barcode.
- ▷ **`BarCodeDescr`** (input_control) ..... string-array ⇝ *string* / integer / real
  Beschreibung eines Barcode-Typs
- ▷ **`GenericName`** (input_control) ..... string(-array) ⇝ *string*
  Namen der optionalen Steuerparameter.
  Defaultwert : '[]'
  Werteliste : `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', 'max_extra_elements'}
- ▷ **`GenericValue`** (input_control) ..... number(-array) ⇝ *real* / integer
  Wert der optionalen Steuerparameter.
  Defaultwert : '[]'
- ▷ **`BarcodeFound`** (output_control) ..... integer ⇝ *integer*
  Information ob der Barcode gefunden wurde.
  Werteliste : `BarcodeFound` ∈ {0, 1}
- ▷ **`BarCodeElements`** (output_control) ..... number-array ⇝ *real*
  Breiten der Elemente.
- ▷ **`Orientation`** (output_control) ..... angle.rad ⇝ *real*
  Orientierung des Barcodes.

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
HTuple   BarcodeFound,Elements,Orientation;
HTuple   Characters,Reference,IsCorrect;
Hobject  Image,CodeRegion;

gen_1d_bar_code_descr("EAN 13",13,13,&BarCodeDescr);
find_1d_bar_code(Image,&CodeRegion,BarCodeDescr,empty,empty,
                 &BarcodeFound,&Elements,&Orientation);
if (BarcodeFound[0].l)
{
  decode_1d_bar_code(Elements,BarCodeDescr,
                     &Characters,&Reference,&IsCorrect);
  if (IsCorrect[0].l)
    for (int i=0; i<Characters.Num(); i++)
    {
      char *value = Characters[i];
    }
}
```

Ergebnis

`find_1d_bar_code` liefert den Wert 2 (H_MSG_TRUE), sofern die die Parameter korrekt sind.

Parallelisierungsinformation

`find_1d_bar_code` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_1d_bar_code_descr`, `gen_1d_bar_code_descr_gen`

Mögliche Nachfolgerfunktionen

decode_1d_bar_code, discrete_1d_bar_code

Alternativen

find_1d_bar_code_region

Siehe auch

sobel_dir, gray_projections, gen_measure_rectangle2

Modul

Barcode reader

find_1d_bar_code_region (Image : CodeRegion : BarCodeDescr, GenericName, GenericValue : Orientation)

Mehrere Barcode-Regionen in einem Bild finden.

find_1d_bar_code_region dient zur Suche mehrerer Barcodes in einem Bild. Im Gegensatz zu find_1d_bar_code wird dieser Operator verwendet, wenn mehr als ein Barcode im Bild vorhanden ist. Hierbei werden nur die Regionen des Barcodes, nicht aber die Dicken der Elemente bestimmt. Für jede Region wird auch deren Orientierung in Bogenmaß ermittelt.

Die Steuerung der Bildverarbeitung erfolgt genauso wie bei find_1d_bar_code. Die Beschreibungen der Parameter GenericName und GenericValue sind bei diesem Oprator zu finden.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Bild, das Barcodes enthält.
- ▷ **CodeRegion** (output_object) ..... region(-array) ⇝ *Hobject*
  Regionen der Barcodes.
- ▷ **BarCodeDescr** (input_control) ..... string-array ⇝ *string* / integer / real
  Beschreibung eines Barcode-Typs
- ▷ **GenericName** (input_control) ..... string(-array) ⇝ *string*
  Namen der optionalen Steuerparameter.
  Defaultwert : '[]'
  Werteliste : GenericName ∈ {'amplitude_sobel', 'min_size_element', 'max_size_element', 'angle_range', 'correct_angle', 'dilation_factor', 'sum_angles'}
- ▷ **GenericValue** (input_control) ..... number(-array) ⇝ *real* / integer / string
  Werte der optionalen Steuerparameter.
  Defaultwert : '[]'
- ▷ **Orientation** (output_control) ..... real(-array) ⇝ *real*
  Orientierung des Barcodes.

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
HTuple   Orientations, Elements;
HTuple   Characters,Reference,IsCorrect;
Hobject  Image,CodeRegions,CodeRegion,GrayRegion;
long     num;

gen_1d_bar_code_descr("code 39",4,15,&BarCodeDescr);
find_1d_bar_code_region(Image,&CodeRegion,BarCodeDescr,empty,empty,
                        &Orientations);
count_obj(CodeRegions,&num);
for (long i=0; i<num; i++)
{
  select_obj(CodeRegions,&CodeRegion,i);
  reduce_domain(Image,CodeRegion,GrayRegion)\:
  get_1d_bar_code(GrayRegion,BarCodeDescr,empty,empty,Orientations[i],
```

```
                  &Elements);
  decode_1d_bar_code(Elements,BarCodeDescr,
                     &Characters,&Reference,&IsCorrect);
}
```

Ergebnis

`find_1d_bar_code_region` liefert den Wert 2 (H_MSG_TRUE), sofern die Parameter korrekt sind und mindestens ein Barcode gefunden wird.

Parallelisierungsinformation

`find_1d_bar_code_region` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_1d_bar_code_descr`, `gen_1d_bar_code_descr_gen`

Mögliche Nachfolgerfunktionen

`get_1d_bar_code`, `count_obj`, `select_obj`, `reduce_domain`

Alternativen

`find_1d_bar_code`

Siehe auch

`sobel_dir`

Modul

Barcode reader

find_2d_bar_code (Image : CodeRegion : BarCodeDescr, GenParamNames, GenParamValues : CodeRegDescr)

Suche nach Regionen, die einen 2D-Barcode enthalten könnten.

`find_2d_bar_code` sucht im Bild `Image` nach Regionen, die einen 2D-Barcode enthalten könnten. Vielversprechende Regionenkandidaten werden als Tupel von Regionen in `CodeRegion` zurückgeliefert. Ob eine solche Region tatsächlich einen lesbaren Barcode enthält, wird erst beim nachfolgenden Auslesen des Barcodes mit Hilfe des Operators `get_2d_bar_code` (bzw. `get_2d_bar_code_pos`) festgestellt.

Neben den Regionen, die mit hoher Wahrscheinlichkeit einen Barcode enthalten, reicht der Operator `find_2d_bar_code` an den nachfolgenden Operator für die Datenextraktion (`get_2d_bar_code` bzw. `get_2d_bar_code_pos`) bestimmte interne Informationen über die Regionen weiter. Diese Informationen werden in einen Regionen-Descriptor geschrieben und im Parameter `CodeRegDescr` für die weitere Verarbeitung zurückgeliefert.

Für die Suche der Barcode-Region können in Abhängigkeit von der Druckmethode, mit der der Barcode erzeugt wurde (**’mode’**), unterschiedliche Verfahren zum Einsatz kommen. Diese Druckmethode ist zusammen mit anderen Informationen über den Barcode Bestandteil des Descriptors, der mit Hilfe des Operators `gen_2d_bar_code_descr` erzeugt und im Parameter `BarCodeDescr` übergeben wird.

Bei schwierigen Bedingungen können mit Hilfe der (optionalen) generischen Parameter `GenParamNames` und `GenParamValues` zusätzliche Parameter zur Steuerung der Suchmethode in Form von Bezeichner-Wert-Paaren übergeben.

Die Einflußnahme kann auf zwei Ebenen erfolgen: Eine Klasse von Parametern beschreibt die zu suchenden Barcode-Symbole und kann somit direkt aus dem zur Verfügung stehenden Bildmaterial abgeleitet werden. Führt ein Anpassen dieser Parameter nicht zum Erfolg, kann ein Teil der intern verwendeten Bildverarbeitungsoperatoren auch direkt beeinflußt werden. Da in Abhängigkeit von der Druckmethode unterschiedliche Bildverarbeitungsverfahren zum Einsatz kommen, stehen auch unterschiedliche Parameter zur Verfügung.

- Allgemeine Parameter zur Beschreibung des Symbols:

  ’module_width’ Mittlere Größe eines Moduls (einzelnes Datenelement) in Pixeln.

  Aus dieser Größe werden eine Reihe weiterer Parameter abgeleitet. Ein Modul sollte in der Größenordnung von 4 bis 16 Pixeln liegen, um erfolgreich erkannt zu werden.

`GenParamValues`:	> 0
default:	10

- Bildverarbeitungsparameter für die Druckmethode **'printed'**:

  Die folgenden Parameter werden dazu verwendet, die Menge interessanter Regionen mit Hilfe bestimmter Regionenmerkmale einzuschränken. Um die Verwendung eines Merkmals bei der Segmentation explizit zu verhindern, ist der entsprechende Parameter auf **-1** zu setzen.

  'anisometry_max' Maximale Anisometrie (vgl. `eccentricity`).
  `GenParamValues`: > 1 (-1: Merkmal nicht verwenden)
  default: 1.45

  'compactness_min' Minimale Kompaktheit (vgl. `compactness`).
  `GenParamValues`: > 1 (-1: Merkmal nicht verwenden)
  default: 1.2

  'compactness_max' Maximale Kompaktheit.
  `GenParamValues`: $\geq$ **'compactness_min'** (-1: Merkmal nicht verwenden)
  default: 3.0

  'circularity_max' Maximale Zirkularität (vgl. `circularity`).
  `GenParamValues`: ≤ 1.0 (-1: Merkmal nicht verwenden)
  default: 0.8

  'circularity_min' Minimale Zirkularität.
  `GenParamValues`: $\leq$ **'circularity_max'** (-1: Merkmal nicht verwenden)
  default: 0.45

  'deviation_min' Minimale Grauwert-Abweichung (vgl. `intensity`).
  `GenParamValues`: > 1.0 (-1: Merkmal nicht verwenden)
  default: 20.0

  Die folgenden Parameter werden automatisch aus der Modulgröße **'module_width'** und Parametern des Barcode-Descriptors `BarCodeDescr` bestimmt (siehe Operator `gen_2d_bar_code_descr`). Sie können jedoch auch individuell angepaßt werden. ACHTUNG: Wird die Modulgröße zusammen mit anderen Parametern gesetzt, muß sie VOR den anderen Parametern angegeben werden.

  'mean_mask_size_1' Maskengröße für die erste Bildmittelung.
  `GenParamValues`: ≥ 3
  default: $2.5 * \text{'module\_width'}$

  'mean_mask_size_2' Maskengröße für die zweite Bildmittelung.
  `GenParamValues`: ≥ 5
  default: $10.0 * \text{'module\_width'}$

  'area_min' Minimale Symbolgröße in Pixel*Pixel.
  `GenParamValues`: > 0
  default: $0.25 * \text{'module\_width'}^2 * \text{'columns\_min'} * \text{'rows\_min'}$

  'area_max' Maximale Symbolgröße in Pixel*Pixel.
  `GenParamValues`: $\geq$ 'area_min'
  default: $4.00 * \text{'module\_width'}^2 * \text{'columns\_min'} * \text{'rows\_min'}$

  'closing_mask_rad' Maskengröße für `closing_circle` über der Testregion.
  `GenParamValues`: > 0
  default: **'module_width'**

- Bildverarbeitungsparameter für die Druckmethode **'engraved_darkfield'**:

  'compactness_min' Minimale Kompaktheit (vgl. `compactness`).
  `GenParamValues`: > 1.0 (-1: Merkmal nicht verwenden)
  default: 1.2

'compactness_max' Maximale Kompaktheit.
GenParamValues: $\geq$ **'compactness_min'** (-1: Merkmal nicht verwenden)
default: 3.0

'edge_thresh' Schwellwert im Kantenbild.
GenParamValues: 0 ... 255
default: 120

'region_rect2_rel' Flächenverhältnis zwischen Region und dem kleinsten umschließenden Rechteck.
GenParamValues: < 1.0
default: 0.7

'median_mask_rad' Maskengröße für initiale Medianfilterung (vgl. Operator median_image).
GenParamValues: > 0
default: $0.1 * \mathbf{'module\_width'} + 0.5$

- Bildverarbeitungsparameter für die Druckmethode **'engraved_lightfield'**:

'anisometry_max' Maximale Anisometrie (vgl. eccentricity).
GenParamValues: > 1 (-1: Merkmal nicht verwenden)
default: 1.45

'compactness_min' Minimale Kompaktheit (vgl. compactness).
GenParamValues: > 1 (-1: Merkmal nicht verwenden)
default: 1.2

'compactness_max' Maximale Kompaktheit.
GenParamValues: $\geq$ **'compactness_min'** (-1: Merkmal nicht verwenden)
default: 3.0

'circularity_max' Maximale Zirkularität (vgl. circularity).
GenParamValues: ≤ 1.0 (-1: Merkmal nicht verwenden)
default: 0.8

'circularity_min' Minimale Zirkularität.
GenParamValues: $\leq$ **'circularity_max'** (-1: Merkmal nicht verwenden)
default: 0.45

'deviation_min' Minimale Grauwert-Abweichung (vgl. intensity).
GenParamValues: > 1.0 (-1: Merkmal nicht verwenden)
default: 20.0

'mean_mask_size' Maskengröße für die Bildmittelung.
GenParamValues: ≥ 3
default: $10 * \mathbf{'module\_width'}$

'area_min' Minimale Symbolgröße in Pixel*Pixel.
GenParamValues: > 0
default: $0.25 * \mathbf{'module\_width'}^2 * \mathbf{'columns\_min'} * \mathbf{'rows\_min'}$

'area_max' Maximale Symbolgröße in Pixel*Pixel.
GenParamValues: $\geq$ 'area_min'
default: $4.00 * \mathbf{'module\_width'}^2 * \mathbf{'columns\_min'} * \mathbf{'rows\_min'}$

'closing_mask_rad' Maskengröße für closing_circle über der Testregion.
GenParamValues: > 0
default: **'module_width'**

'opening_mask_rad' Maskengröße für opening_circle über der Testregion.
GenParamValues: > 0
default: $1.5 * \mathbf{'module\_width'}$

Parameter

- ▷ **Image** (input_object) . . . . . image ⇝ *Hobject*
  Bild, das Barcode(s) enthält.
- ▷ **CodeRegion** (output_object) . . . . . region(-array) ⇝ *Hobject*
  Regionen, die einen Barcode enthalten könnten.
- ▷ **BarCodeDescr** (input_control) . . . . . string-array ⇝ *string* / integer / real
  Beschreibung der gesuchten 2D-Barcodeklasse
- ▷ **GenParamNames** (input_control) . . . . . string(-array) ⇝ *string*
  Liste von Namen (optionaler) generischer Parameter für die Steuerung der Bildverarbeitung.
  Defaultwert : '[]'
  Werteliste : `GenParamNames` ∈ {'module_width', 'anisometry_max', 'compactness_min', 'compactness_max', 'circularity_min', 'circularity_max', 'deviation_min', 'mean_mask_size', 'mean_mask_size_1', 'mean_mask_size_2', '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) ⇝ *real* / integer / string
  Liste von Werten der generischen Parameter für die Steuerung der Bildverarbeitung.
  Defaultwert : '[]'
  Werteliste : `GenParamValues` ∈ {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-array ⇝ *string* / integer / real
  Zusätzliche Parameter, die die Barcoderegionen beschreiben und für die Extraktion der Datenwerte verwendet werden können (siehe `decode_2d_bar_code`).

Ergebnis

`find_2d_bar_code` liefert den Wert 2 (H_MSG_TRUE), falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`find_2d_bar_code` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_2d_bar_code_descr`

Mögliche Nachfolgerfunktionen

`get_2d_bar_code`, `get_2d_bar_code_pos`

Modul

Barcode reader

gen_1d_bar_code_descr (: : CodeName, MinCharacters, MaxCharacters : BarCodeDescr)

Beschreibung eines 1D Barcodes erzeugen.

`gen_1d_bar_code_descr` erzeugt eine Beschreibung eines eindimensionalen Barcodes. Diese Beschreibung wird für die Suche (`find_1d_bar_code` oder `find_1d_bar_code_region`) und die Dekodierung eines Barcodes (`decode_1d_bar_code`) und andere Barcode-Operationen benötigt. `gen_1d_bar_code_descr` ist also der erste Operator der in einer Programmsequenz für die Barcode-Verarbeitung aufgerufen wird. `gen_1d_bar_code_descr` muß nur einmal zu Beginn eines Programms aufgerufen werden. Der Wert von `BarCodeDescr` kann beliebig oft verwendet werden. Soll mehr als eine Art von Barcode verarbeitet werden, dann ist `gen_1d_bar_code_descr` für jeden Barcode-Typ einmal aufzurufen.

Es ist zu beachten, daß die Beschreibung nur die wichtigsten Informationen des Barcodes enthält. Dies hat insbesondere zur Folge, daß praktisch alle Barcodes mit einer beliebigen Beschreibung *gefunden* werden können. Hingegen wird eine spezifische Beschreibung benötigt, um einen speziellen Barcode-Typ zu *decodieren*.

Parameter

- ▷ **CodeName** (input_control) . . . . . string ⇝ *string*
  Names des Barcodes
  Defaultwert : ”EAN 13”
  Werteliste : CodeName ∈ {”2/5 Industrial”, ”2/5 Interleaved”, ”Code 39”, ”Codabar”, ”Code 128”, ”Code 93”, ”EAN 13”, ”EAN 13 Add-On 2”, ”EAN 13 Add-On 5”, ”EAN 8”, ”EAN 8 Add-On 2”, ”EAN 8 Add-On 5”, ”UPC-A”, ”UPC-A Add-On 2”, ”UPC-A Add-On 5”, ”UPC-E”, ”UPC-E Add-On 2”, ”UPC-E Add-On 5”, ”PharmaCode”}
- ▷ **MinCharacters** (input_control) . . . . . integer ⇝ *integer*
  Minimale Anzahl von Zeichen (falls wählbar)
  Defaultwert : 6
  Wertevorschläge : MinCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30}
- ▷ **MaxCharacters** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Zeichen (falls wählbar)
  Defaultwert : 10
  Wertevorschläge : MaxCharacters ∈ {-1, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 14, 16, 18, 20, 25, 30, 35, 40, 50}
- ▷ **BarCodeDescr** (output_control) . . . . . string-array ⇝ *string* / integer / real
  Beschreibung eines Barcode-Typs

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
HTuple   BarcodeFound,Elements,Orientation;
HTuple   Characters,Reference,IsCorrect;
Hobject  Image,CodeRegion;

gen_1d_bar_code_descr("EAN 13",13,13,&BarCodeDescr);
find_1d_bar_code(Image,&CodeRegion,BarCodeDescr,empty,empty,
                 &BarcodeFound,&Elements,&Orientation);
if (BarcodeFound[0].l)
{
  decode_1d_bar_code(Elements,BarCodeDescr,
                     &Characters,&Reference,&IsCorrect);
  if (IsCorrect[0].l)
    for (int i=0; i<Characters.Num(); i++)
    {
      char *value = Characters[i];
    }
}
```

Ergebnis

gen_1d_bar_code_descr liefert den Wert 2 (H_MSG_TRUE), sofern die übergebenen Barcode-Name korrekt ist.

Parallelisierungsinformation

gen_1d_bar_code_descr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

find_1d_bar_code, find_1d_bar_code_region

Alternativen

gen_1d_bar_code_descr_gen

Modul

Barcode reader

gen_1d_bar_code_descr_gen (: : MinCodeLength, MaxCodeLength, ElementSizes, StartElement, StopElement, MaxSizeRatio, DiscreteCode : BarCodeDescr)

Generische Beschreibung eines 1D Barcodes erzeugen.

gen_1d_bar_code_descr_gen erzeugt eine generische Beschreibung eines eindimensionalen Barcodes. Dieser Operator wird verwendet, wenn der gewünschte Barcode bei gen_1d_bar_code_descr nicht verfügbar ist aber trotzdem mit find_1d_bar_code oder find_1d_bar_code_region gefunden werden soll. Hierbei ist jedoch zu beachten, daß der so gefundene Barcode nicht mit decode_1d_bar_code gelesen werden kann. Die Dekodierung ist in diesem Fall vom Anwender selbst durchzuführen.

Von den Parametern muß auf jeden Fall für DiscreteCode und ElementSizes der korrekte Wert angegeben werden. Die übrigen Parameter sind für die Suche des Barcodes nicht so wichtig und können vom System dynamisch ermittelt werden.

Parameter

- ▷ **MinCodeLength** (input_control) ..... integer ⇝ *integer*
  Minimale Länge des Codes in Modulen (einschließlich Start- und Stopzeichen).
  Defaultwert : 30
  Werteliste : MinCodeLength ∈ {-1, 30, 60, 90, 110, 130, 150, 200}
- ▷ **MaxCodeLength** (input_control) ..... integer ⇝ *integer*
  Maximale Länge des Codes in Modulen (einschließlich Start- und Stopzeichen).
  Defaultwert : 30
  Werteliste : MaxCodeLength ∈ {-1, 30, 60, 90, 110, 130, 150, 200}
- ▷ **ElementSizes** (input_control) ..... integer ⇝ *integer*
  Anzahl verschiedener Elementgrößen.
  Defaultwert : 2
  Werteliste : ElementSizes ∈ {1, 2, 3, 4, 5, 6, 7}
- ▷ **StartElement** (input_control) ..... integer(-array) ⇝ *integer*
  Liste der Elemente des Startzeichens. Die Breite eines Elements wird als Anzahl von Modulen angegeben. Lücken werden durch negative Zahlen markiert.
  Defaultwert : '[1,-1]'
- ▷ **StopElement** (input_control) ..... integer(-array) ⇝ *integer*
  Liste der Elemente des Stopzeichens. Die Breite eines Elements wird als Anzahl von Modulen angegeben. Lücken werden durch negative Zahlen markiert.
  Defaultwert : '[1,-1]'
- ▷ **MaxSizeRatio** (input_control) ..... number ⇝ *real*
  Maximales Verhältnis Länge zu Höhe
  Defaultwert : 2.5
  Werteliste : MaxSizeRatio ∈ {-1, 2, 3, 4, 5, 6}
- ▷ **DiscreteCode** (input_control) ..... string ⇝ *string*
  Diskreter Code (ignoriere Lücken)
  Defaultwert : 'false'
  Werteliste : DiscreteCode ∈ {'true', 'false'}
- ▷ **BarCodeDescr** (output_control) ..... string-array ⇝ *string* / integer / real
  Beschreibung eines Barcode-Typs

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
HTuple   BarcodeFound,Elements,Orientation;
HTuple   DiscreteBarCode;
Hobject  Image,CodeRegion;

gen_1d_bar_code_descr_gen(20,40,2,empty,empty,-1.0,"false",&BarCodeDescr);
find_1d_bar_code(Image,&CodeRegion,BarCodeDescr,empty,empty,
                 &BarcodeFound,&Elements,&Orientation);
if (BarcodeFound[0].l)
```

```
{
  discrete_1d_bar_code(Elements,BarCodeDescr,&DiscreteBarCode);
  for (int i=0; i<DiscreteBarCode.Num(); i++)
  {
    int NumModules = DiscreteBarCode[i];
  }
}
```

Ergebnis

`gen_1d_bar_code_descr_gen` liefert den Wert 2 (H_MSG_TRUE), sofern die übergebenen Werte korrekt sind.

Parallelisierungsinformation

`gen_1d_bar_code_descr_gen` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`find_1d_bar_code`, `find_1d_bar_code_region`

Alternativen

`gen_1d_bar_code_descr`

Modul

Barcode reader

gen_2d_bar_code_descr (: : CodeType, GenParamNames, GenParamValues : BarCodeDescr)

Erzeugen einer generischen Beschreibung einer 2D-Barcodeklasse.

`gen_2d_bar_code_descr` erzeugt eine generische Beschreibung für einen zweidimensionalen Barcode. Zu den Bestandteilen einer solchen Beschreibung gehören der Barcodetyp sowie weitere charakteristische Merkmale der gesuchten Codes.

Der Barcodetyp wird im Parameter `CodeType` übergeben. Zur Zeit werden die folgenden 2D-Barcodes unterstützt:

’Data Matrix ECC 200’ Data Matrix mit ECC 200 Kodierung.

Die Beschreibung der Barcodes erfolgt mit Hilfe generischer Parameter. Dafür können in den Parametern `GenParamNames` und `GenParamValues` Paare von Descriptoren und Werten übergeben werden. Die folgenden generischen Parameter beschreiben die Größe und Form des Symbols:

’columns’ Exakte Anzahl der Spalten des Symbols.

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’ Mindestanzahl an Spalten.

`GenParamValues`:	10...144 (gerade)
default:	10

’columns_max’ Maximale Anzahl an Spalten.

`GenParamValues`:	10...144 (gerade)

’rows’ Exakte Anzahl der Zeilen des Symbols.

`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’ Mindestanzahl an Zeilen.

`GenParamValues`:	8...144 (gerade)
default:	8

'rows_max' Maximale Anzahl an Zeilen.

GenParamValues:	8...144 (gerade)	
default:	144	

'shape' Form des Symbols.

GenParamValues:	'square':	quadratisch
	'rectangle':	rechteckig
	'any':	beliebig
default:	'any'	

Die folgenden generischen Parameter beschreiben den Druck bzw. das Erscheinungsbild des Barcodes:

'foreground' Färbung gesetzter Bits (Vordergrund).

2D-Barcodes können sowohl schwarz-auf-weiß als auch umgekehrt gedruckt werden. Ist bekannt, ob gesetzte Bits hell oder dunkel sind, kann durch Angabe dieser Information Rechenzeit gespart werden, insbesondere beim Suchen der Symbolkandidaten (Operator `find_2d_bar_code`).

GenParamValues:	'any':	beliebig
	'dark','black':	Dunkler Druck auf hellem Hintergrund (schwarz auf weiß)
	'light','white':	heller Druck auf dunklem Hintergrund (weiß auf schwarz)
default:	'any'	

'mode' Methode beim Druck des Barcodes.

Durch Angabe dieses Parameters wird ein für eine spezielle Druckmethode optimiertes Bildverarbeitungsverfahren ausgewählt.

GenParamValues:	'printed':	Normaler Schwarz-Weiß-Druck. Module erscheinen als Quadrate; benachbarte Module mit gleichem Wert verlaufen zu einer Region.
	'engraved_darkfield':	Die Bits werden z.B. mit Laser in die Oberfläche gebrannt. Bei Dunkelfeld-Beleuchtung erscheinen Module mit Wert **1** als dunkle runde Punkte mit weißer Korona; benachbarte Module mit dem Wert **1** werden also durch Zwischenräume getrennt.
	'engraved_lightfield':	Wie 'engraved_darkfield', aber mit Hellfeld-Beleuchtung. Module mit Wert **1** erscheinen als helle Punkte mit dunkler Korona; benachbarte Module mit dem Wert **1** werden durch Zwischenräume getrennt.
default:	'printed'	

Parameter

▷ **`CodeType`** (input_control) . . . . . string ⇝ *string*
Typ des 2D-Barcodes.
Defaultwert : "Data Matrix ECC 200"
Werteliste : `CodeType` ∈ {"Data Matrix ECC 200"}

▷ **`GenParamNames`** (input_control) . . . . . string-array ⇝ *string*
Liste von Namen generischer Parameter zur Beschreibung der gesuchten 2D-Barcodeklasse.
Defaultwert : '[]'
Werteliste : `GenParamNames` ∈ {'columns', 'columns_min', 'columns_max', 'rows', 'rows_min', 'rows_max', 'foreground', 'shape', 'mode'}

▷ **`GenParamValues`** (input_control) . . . . . integer-array ⇝ *integer* / string / real
Liste von Werten der generischen Parameter zur Beschreibung der gesuchten 2D-Barcodeklasse.
Defaultwert : '[]'
Werteliste : `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, 'dark', 'light', 'square', 'rectangle', 'printed', 'engraved_darkfield', 'engraved_lightfield', 'any'}

▷ **`BarCodeDescr`** (output_control) . . . . . string-array ⇝ *string* / integer / real
Beschreibung der 2D-Barcodeklasse

Beispiel

```
gen_2d_bar_code_descr ('Data Matrix ECC 200', ['mode','foreground'],
                       ['printed','dark'], BarCodeDescr)
```

Ergebnis

gen_2d_bar_code_descr liefert den Wert 2 (H_MSG_TRUE), falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_2d_bar_code_descr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

find_2d_bar_code

Modul

Barcode reader

get_1d_bar_code (BarCodeRegion : : BarCodeDescr, GenericName, GenericValue, Orientation : BarCodeElements)

Bestimmt die Dicke der Elemente innerhalb einer Barcode-Region.

get_1d_bar_code extrahiert die Elementdicke innerhalb der vorgegebenen (Barcode-)Region.

Die Steuerung der Verarbeitung erfolgt genauso wie bei find_1d_bar_code. Die Beschreibung der Parameter GenericName und GenericValue sind bei diesem Operator zu finden.

Parameter

- ▷ **BarCodeRegion** (input_object) ..... image ⇝ *Hobject*
  Region des Barcodes.
- ▷ **BarCodeDescr** (input_control) ..... string-array ⇝ *string* / integer / real
  Beschreibung eines Barcode-Typs
- ▷ **GenericName** (input_control) ..... string(-array) ⇝ *string*
  Namen der optionalen Steuerparameter.
  Defaultwert : ’[]’
  Werteliste : GenericName ∈ {’sigma_project’, ’amplitude_project’, ’width_project’, ’add_length_project’, ’interpolation_project’, ’max_extra_elements’}
- ▷ **GenericValue** (input_control) ..... number(-array) ⇝ *real* / integer
  Werte der optionalen Steuerparameter.
  Defaultwert : ’[]’
- ▷ **Orientation** (input_control) ..... angle.rad ⇝ *real*
  Orientierung des Barcodes.
- ▷ **BarCodeElements** (output_control) ..... number-array ⇝ *real*
  Dicke der einzelnen Elemente.

Beispiel

```
HTuple   empty;   // empty list of values
HTuple   BarCodeDescr;
HTuple   Orientations, Elements;
HTuple   Characters,Reference,IsCorrect;
Hobject  Image,CodeRegions,CodeRegion,GrayRegion;
long     num;

gen_1d_bar_code_descr("code 39",4,15,&BarCodeDescr);
find_1d_bar_code_region(Image,&CodeRegion,BarCodeDescr,empty,empty,
                        &Orientations);
count_obj(CodeRegions,&num);
for (long i=0; i<num; i++)
{
  select_obj(CodeRegions,&CodeRegion,i);
  reduce_domain(Image,CodeRegion,GrayRegion)\:
```

```
  get_1d_bar_code(GrayRegion,BarCodeDescr,empty,empty,Orientations[i],
                  &Elements);
  decode_1d_bar_code(Elements,BarCodeDescr,
                     &Characters,&Reference,&IsCorrect);
}
```

Ergebnis

`get_1d_bar_code` liefert den Wert 2 (H_MSG_TRUE), sofern die Parameter korrekt sind.

Parallelisierungsinformation

`get_1d_bar_code` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`find_1d_bar_code_region`, `select_obj`, `reduce_domain`

Mögliche Nachfolgerfunktionen

`decode_1d_bar_code`

Alternativen

`find_1d_bar_code`

Modul

Barcode reader

get_2d_bar_code (BarCodeRegion, Image : : BarCodeDescr, CodeRegDescr, GenParamNames, GenParamValues : BarCodeDimension, BarCodeData)

Bestimmen der Werte der Datenelemente (in ECC 200: “Module”) innerhalb einer Barcode-Region (“Data Matrix Symbol”).

`get_2d_bar_code` extrahiert aus einem Symbolkandidaten `BarCodeRegion` und den dazugehörigen Grauwerten `Image` die einzelnen Datenwerte (Bit gesetzt oder nicht). `BarCodeDescr` enthält den mit `gen_2d_bar_code_descr` erzeugten Barcode-Descriptor, `CodeRegDescr` den von `find_2d_bar_code` erzeugten Regionen-Descriptor.

Wenn die übergebene Region `BarCodeRegion` einen Barcode enthält, werden die Daten zeilenweise in ein Tupel geschrieben und im Parameter `BarCodeData` zurückgeliefert. Dabei stehen positive Werte für eine logische Eins (Bit gesetzt) und negative Werte für eine logische Null (Bit nicht gesetzt). Der Wert Null bedeutet, daß das Modul nicht eindeutig klassifiziert werden konnte.

In `BarCodeDimension` wird die Größe des Datenfeldes zurückgegeben. Diese ist nicht mit der Symbolgröße zu verwechseln, zu der zusätzliche Such- und Alignmentmuster mitzählen. Die ersten beiden Werte von `BarCodeDimension` enthalten die Breite und die Höhe des Datenfelds, der dritte Wert entspricht dem Index des Symbols.

Falls kein Barcode extrahiert werden konnte, wird dies durch einen Fehlercode im Rückgabewert signalisiert.

Um auch in schwierigen Fällen noch eine Extraktion der Datenwerte zu ermöglichen, kann mit Hilfe der (optionalen) generischen Parameter `GenParamNames` und `GenParamValues` gezielt Einfluß auf die Bildverarbeitungsverfahren genommen werden. Welche das im einzelnen sind, hängt von der Druckform des Barcodes ab, die im Barcode-Descriptor `BarCodeDescr` abgelegt ist (siehe `gen_2d_bar_code_descr`). Die Breite eines einzelnen Moduls (**’module_width’**) braucht an dieser Stelle nicht mehr spezifiziert zu werden, da sie Bestandteil des von `find_2d_bar_code` gelieferten Regionen-Descriptors `CodeRegDescr` ist.

Die Bildverarbeitung gliedert sich grob in die folgenden Schritte: Zuerst wird die übergebene Region `BarCodeRegion` mittels Schwellwert-Operator in eine helle und eine dunkle Region unterteilt. Da um einen 2D-Barcode ein “leerer” Sicherheitsbereich zu liegen hat, kann in diesem Bereich bestimmt werden, ob das Symbol schwarz-auf-weiß oder umgekehrt gedruckt ist. Damit können die “dunklen” und “hellen” Regionen den logischen Datenwerten zugeordnet werden. `BarCodeRegion` wird dann durch ein Rechteck approximiert. Die Randbereiche des Rechtecks werden auf die in der 2D-Barcode-Spezifikation festgelegten Muster hin untersucht. Daraus ergibt sich automatisch die Größe der Datenmatrix und die Größe und Position der Module, deren Datenwert daraufhin ermittelt wird.

- Parameter für die Bestimmung der Datenwerte eines Moduls:

'module_rad_ratio' Bereich des Moduls für die Klassifikation der Daten.

Dieser Parameter legt den Bereich des Moduls (in Prozent) fest, in dem die Klassifikation der Daten (gesetzt oder nicht gesetzt) erfolgen soll. 'module_rad_ratio' = 1.0 bedeutet, daß das Modul in seiner gesamten Größe untersucht werden soll, bei sehr kleinem 'module_rad_ratio' wird dagegen nur ein Punkt in der Mitte getestet.

GenParamValues:	$0 <$ 'module_rad_ratio' ≤ 1	
default:	Druckmethode **'printed'**:	0.3
	Druckmethode **'engraved_lightfield'**:	0.3
	Druckmethode **'engraved_darkfield'**:	0.7

'use_grayvals' Verfahren zur Bestimmung des Datenwerts (Nur wählbar bei den Druckmethoden **'printed'** und **'engraved_lightfield'**; bei **'engraved_darkfield'** wird immer anhand der Grauwerte klassifiziert!).

Mit dem Parameter 'use_grayvals' wird festgelegt, mit welchem Verfahren der Datenwert eines Moduls ermittelt werden soll. Ist 'use_grayvals' = 'no', wird getestet, ob der mit **'module_rad_ratio'** festgelegte Bereich innerhalb der Region liegt, die dem Datenwert **1** zugeordnet wurde. Liegt mehr als die Hälfte des Moduls innerhalb dieser Region, wird das Modul als gesetztes Bit interpretiert.
Ist 'use_grayvals' = 'yes', werden innerhalb des Testbereichs die Grauwerte untersucht. Dafür wird zuvor aus dem bekannten Symbolrand ein Klassifikator erzeugt, der anhand des Minimums, des Maximums und des Wertebereichs der Grauwerte den Testbereich als 'gesetzt' oder 'nicht gesetzt' klassifiziert. Einzelne Grauwert-Merkmale werden ausgeschlossen, wenn sie die Testregionen nicht eindeutig trennen können. Ist die Klassifikation nicht eindeutig, entscheidet die Mehrheit. Ist keines der Grauwertmerkmale geeignet, die Testregionen zu trennen, wird automatisch die regionenbasierte Methode verwendet. Zu erkennen ist dies anhand der in BarCodeData zurückgelieferten Werte: Bei der regionenbasierten Methode werden Vielfache von 4 zurückgeliefert, ansonsten Werte zwischen -3 und +3.
Bei der Druckmethode **'engraved_darkfield'** wird immer anhand der Grauwerte klassifiziert. Die Klassifizierbarkeit läßt sich hierbei oft durch eine günstige Wahl von **'module_rad_ratio'** verbessern.

GenParamValues:	'yes', 'no'
default:	'yes'

'use_grayval_min' Klassifikation anhand des Grauwert-Minimums.

GenParamValues:	'yes', 'no'
default:	'yes'

'use_grayval_max' Klassifikation anhand des Grauwert-Maximums.

GenParamValues:	'yes', 'no'	
default:	**'printed'**, **'engraved_lightfield'**:	'yes'
	'engraved_darkfield':	'no'

'use_grayval_range' Klassifikation anhand des Grauwertbereichs.

GenParamValues:	'yes', 'no'
default:	'yes'

- Bei den Druckmethoden **'printed'** und **engraved_lightfield** können zusätzlich die folgenden Parameter gesetzt werden:

'enlarge_region_rad' Vergrößerung der Symbolregion.

GenParamValues:	> 0 (≤ 0: keine Vergrößerung erlaubt)
default:	2.5

'thresh_percent' Histogrammbereich zum Suchen des optimalen Schwellwerts.

Der Schwellwert zum Trennen der hellen und dunklen Bereiche der Region wird in den mittleren 'thresh_percent' Prozent des Grauwerthistogramms gesucht. Bei 'thresh_percent' = 0 liegt der Schwellwert genau in der Mitte zwischen dem hellsten und dem dunkelsten Pixel. Bei 'thresh_percent' = 100 wird der gesamte Histogrammbereich durchsucht. Der Schwellwert beeinflußt neben der Größe der Datenregion auch die Position des umschließenden Vierecks des Symbols und somit auch geringfügig die angenommene Lage der Module.

GenParamValues:	0 ... 100
default:	50

'thresh_step_width' Schrittweite zum Suchen des optimalen Schwellwerts.

Der mit 'thresh_percent' abgesteckte Bereich wird mit der Schrittweite 'thresh_step_width' solange durchsucht, bis eine optimale Trennung zwischen den hellen und dunklen Bereichen (Maximum der Summe aus hellen und dunklen Regionen) gefunden wurde. Je kleiner 'thresh_step_width' ist, um so genauer kann das Optimum bestimmt werden, um so länger wird aber auch gesucht.

GenParamValues: > 0
default: 10

'smooth_cont' Parameter für die Approximation der Region durch ein Polygon (Rechteck).

Der Parameter 'smooth_cont' bestimmt, über wieviele Punkte die Kontur geglättet wird, bevor die Polygonapproximation durchgeführt wird (vgl. Operator segment_contours_xld).

GenParamValues: 0,3,5,7,9,...
default: 5

'max_line_dist1', 'max_line_dist2' Abstand der Kontur von der approximierten Linie.

Die Parameter 'max_line_dist1' und 'max_line_dist2' bestimmen den maximalen Abstand zwischen der Kontur und der approximierten Linie (vgl. Operator segment_contours_xld).

GenParamValues: ≥ 0
default: 4

'min_cont_len' Minimale Länge der Konturelemente in Pixel.

Konturelemente, die kürzer als 'min_cont_len' sind, werden beim Bestimmen des umgebenden Vierecks der Region ignoriert. Bei sehr kleinen Symbolen oder bei geringer Auflösung kann es nötig sein, diesen Wert zu verkleinern.

GenParamValues: > 0
default: 50

'border_width_min' Minimale Breite des Randbereichs.

Im Randbereich der übergebenen Region muß das sogenannte Suchmuster liegen, bestehend aus zwei kontinuierlichen Linien mit gesetzten Bits und zwei alternierenden Linien. Ausgehend von 'border_width_min' wird der Suchbereich solange vergrößert, bis ein gültiger Symbolrand gefunden oder **'border_width_max'** erreicht wurde.

GenParamValues: > 0
default: 2

'border_width_max' Maximale Breite des Randbereichs.

GenParamValues: $\geq$ **'border_width_min'**
default: 5

'border_width' Exakte Breite des Randbereichs.

Als Alternative kann im Parameter 'border_width' die exakte Breite des Suchbereichs übergeben werden. Damit werden automatisch **'border_width_min'** und **'border_width_max'** gesetzt.

GenParamValues: > 0
default: -

Bei der Druckmethode **'engraved_darkfield'** können die folgenden Parameter gesetzt werden:

'median_mask_rad' Maskengröße für initiale Medianfilterung (vgl. Operator median_image).

GenParamValues: 3,5,7,...,501
default: **'module_width'***0.1 + 0.5

'gray_erosion_size' Maskengröße für Grauwerterosion (vgl. Operator gray_erosion_rect).

GenParamValues: 3,5,7,...,511
default: **'module_width'***0.18 + 3.5

'measure_sigma' Parameter für die Extraktion gerader Kantenpaare (vgl. Operator measure_pairs).

GenParamValues: 0.4...100
default: **'module_width'** < 20: 0.7
'module_width' ≥ 20: 1.4

'measure_thresh' Parameter für die Extraktion gerader Kantenpaare (vgl. Operator `measure_pairs`).

`GenParamValues`: 1...255
default: 2.0

Achtung

Der Operator `get_2d_bar_code` signalisiert über den Fehlercode, daß die übergebene Region keinen gültigen Barcode enthält. Da die vom Operator `find_2d_bar_code` extrahierten Regionenkandidaten aber durchaus Regionen ohne Barcode enthalten können, sollte jeder Aufruf von `get_2d_bar_code` in eine eigene Fehlerbehandlung eingeschlossen werden (siehe Beispiel).

Parameter

- ▷ **`BarCodeRegion`** (input_object) ..... region ⇝ *Hobject*
  Region, die einen 2D-Barcode enthalten könnte.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Zugehöriges Bild.
- ▷ **`BarCodeDescr`** (input_control) ..... string-array ⇝ *string* / integer / real
  Beschreibung der 2D-Barcodeklasse
- ▷ **`CodeRegDescr`** (input_control) ..... string-array ⇝ *string* / integer / real
  Zusätzliche Parameter, die die Barcoderegion beschreiben und für die Extraktion der Datenwerte verwendet werden können.
- ▷ **`GenParamNames`** (input_control) ..... string-array ⇝ *string*
  Liste von Namen (optionaler) generischer Steuerparameter.
  Defaultwert : '[]'
- ▷ **`GenParamValues`** (input_control) ..... number-array ⇝ *real* / integer
  Liste der Werte der generischen Steuerparameter.
  Defaultwert : '[]'
- ▷ **`BarCodeDimension`** (output_control) ..... integer-array ⇝ *integer*
  Tupel mit der Dimension des extrahierten Symbols. Bei ECC 200: Datenfeldbreite, -höhe, Symbol-Index.
- ▷ **`BarCodeData`** (output_control) ..... integer-array ⇝ *integer*
  Tupel mit den Datenwerten des extrahierten Symbols. Wert > 0: logische 1, Wert < 0: logische 0, Wert = 0: Modul nicht klassifizierbar.

Beispiel

```
dev_error_var (ErrorCode, 1)
gen_2d_bar_code_descr ('Data Matrix ECC 200', ['mode'], ['printed'],
                       BarCodeDescr)
read_image (Image, 'bar2d.tif')
find_2d_bar_code (Image, CodeRegion, BarCodeDescr,
                  ['module_width'], [10], CodeRegDescr)
SymbolNum := |CodeRegion|
for i := 1 to SymbolNum by 1
  ObjectSelected := CodeRegion[i]
  dev_set_check ('~give_error')
  get_2d_bar_code (ObjectSelected, Image, BarCodeDescr, CodeRegDescr,
                   [], [], BarCodeDimension, BarCodeData)
  err := ErrorCode
  dev_set_check ('give_error')
  if (err = 2)
    decode_2d_bar_code (BarCodeDescr, BarCodeDimension, BarCodeData,
                        SymbolCharacters, CorrSymbolData, DecodedData,
                        DecodingError, StructuredAppend)
    if (|DecodedData|)
      tuple_chrt (DecodedData, String)
    endif
  endif
endfor
```

Ergebnis

Der Operator `get_2d_bar_code` signalisiert über den Fehlercode sowohl, daß inkorrekte Parameter übergeben wurden als auch, daß die Extraktion eines Barcodes aus der übergebenen Region aus bestimmten Gründen scheiterte. Konnte z.B. kein umschließendes Viereck um die Region bestimmt werden, wird der Operator mit dem Fehlercode 8808 abgebrochen. Liegt das Viereck z.T. außerhalb des Bildes, wird mit dem Fehlercode 8807 abgebrochen. Wurde innerhalb des Randbereichs kein Suchmuster gefunden, wird der Operator mit dem Fehlercode 8810 abgebrochen. Wird dagegen der durchgezogener Symbolrand an zwei aneinanderstoßenden Seiten ('L') nicht gefunden, wird mit dem Fehlercode 8809 abgebrochen. Kann bei der Druckmethode **'engraved_darkfield'** keines der Grauwertmerkmale die Testregionen trennen, bricht der Operator mit dem Fehlercode 8811 ab.

Parallelisierungsinformation

`get_2d_bar_code` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`find_2d_bar_code`

Mögliche Nachfolgerfunktionen

`decode_2d_bar_code`

Alternativen

`get_2d_bar_code_pos`

Modul

Barcode reader

get_2d_bar_code_pos (BarCodeRegion, Image : : BarCodeDescr, CodeRegDescr, GenParamNames, GenParamValues : BarCodeDimension, BarCodeData, DataElementRow, DataElementCol)

Bestimmen der Datenwerte der Elemente (in ECC 200: "Module") innerhalb einer Barcode-Region ("Data Matrix Symbol") und ihrer Positionen im Bild.

`get_2d_bar_code_pos` extrahiert aus einem Symbolkandidaten `BarCodeRegion` und den dazugehörigen Grauwerten `Image` die einzelnen Datenwerte (Bit gesetzt oder nicht). Im Gegensatz zu `get_2d_bar_code` werden zusätzlich die Positionen der einzelnen Datenelemente im Bild in den Parametern `DataElementRow` und `DataElementCol` zurück geliefert. Diese können dann zur Kontrolle der Funktion des Operators zum Einzeichnen der Daten in das Ausgangsbild verwendet werden. Für die Beschreibung der anderen Parameter siehe `get_2d_bar_code`.

Achtung

Der Operator `get_2d_bar_code_pos` signalisiert über den Fehlercode, daß die übergebene Region keinen gültigen Barcode enthält. Da die vom Operator `find_2d_bar_code` extrahierten Regionenkandidaten aber durchaus Regionen ohne Barcode enthalten können, sollte jeder Aufruf von `get_2d_bar_code` in eine eigene Fehlerbehandlung eingeschlossen werden (siehe Beispiel).

Parameter

- ▷ **BarCodeRegion** (input_object) ..... region ⇝ *Hobject*
  Region, die einen Barcode enthalten könnte.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Zugehöriges Bild.
- ▷ **BarCodeDescr** (input_control) ..... string-array ⇝ *string* / integer / real
  Beschreibung der 2D-Barcodeklasse
- ▷ **CodeRegDescr** (input_control) ..... string-array ⇝ *string* / integer / real
  Zusätzliche Parameter, die die Barcoderegionen beschreiben und für die Extraktion der Datenwerte verwendet werden können.
- ▷ **GenParamNames** (input_control) ..... string-array ⇝ *string*
  Liste von Namen (optionaler) generischer Steuerparameter.
  Defaultwert : '[]'
- ▷ **GenParamValues** (input_control) ..... number-array ⇝ *real* / integer
  Liste der Werte der generischen Steuerparameter.
  Defaultwert : '[]'

▷ **BarCodeDimension** (output_control) ..... integer-array ⇝ *integer*
Tupel mit der Dimension des extrahierten Symbols. Bei ECC 200: Datenfeldbreite, -höhe, Symbol-Index.

▷ **BarCodeData** (output_control) ..... integer-array ⇝ *integer*
Tupel mit den Datenwerten des extrahierten Symbols. Wert > 0: logische 1, Wert < 0: logische 0, Wert = 0: Modul nicht klassifizierbar.

▷ **DataElementRow** (output_control) ..... real-array ⇝ *real*
Tupel mit den Zeilenpositionen der Datenelemente des extrahierten Symbols im Bild.

▷ **DataElementCol** (output_control) ..... real-array ⇝ *real*
Tupel mit den Spaltenpositionen der Datenwerte des extrahierten Symbols.

Beispiel

```
dev_set_check (’~give_error’)
get_2d_bar_code_pos (ObjectSelected, Image, BarCodeDescr,
                     CodeRegDescr, [], [], BarCodeDimension, BarCodeData,
                     DataElementRow, DataElementCol)
err := ErrorCode
dev_set_check (’give_error’)
if (err = 2)
  idx := 0
  dev_update_pc (’off’)
  dev_update_time (’off’)
  dev_update_var (’off’)
  for m := 0 to BarCodeDimension[0]-1 by 1
    for n := 0 to BarCodeDimension[1]-1 by 1
      if (BarCodeData[idx]>0)
        set_color (WindowHandle, ’green’)
      else
        if (BarCodeData[idx]<0)
          set_color (WindowHandle, ’red’)
        else
          set_color (WindowHandle, ’blue’)
        endif
      endif
      disp_circle (WindowHandle, DataElementRow[idx],
                   DataElementCol[idx], 1)
      idx := idx+1
    endfor
  endfor
  dev_update_pc (’on’)
  dev_update_time (’on’)
  dev_update_var (’on’)
  decode_2d_bar_code (BarCodeDescr, BarCodeDimension, BarCodeData,
                      SymbolCharacters, CorrSymbolData, DecodedData,
                      DecodingError, StructuredAppend)
endif
```

Ergebnis

Der Operator `get_2d_bar_code_pos` signalisiert über den Fehlercode sowohl, daß inkorrekte Parameter übergeben wurden als auch, daß die Extraktion eines Barcodes aus der übergebenen Region aus bestimmten Gründen scheiterte. Konnte z.B. kein umschließendes Viereck um die Region bestimmt werden, wird der Operator mit dem Fehlercode 8808 abgebrochen. Liegt das Viereck z.T. außerhalb des Bildes, wird mit dem Fehlercode 8807 abgebrochen. Wurde innerhalb des Randbereichs kein Suchmuster gefunden, wird der Operator mit dem Fehlercode 8810 abgebrochen. Wird dagegen der durchgezogener Symbolrand an zwei aneinanderstoßenden Seiten (’L’) nicht gefunden, wird mit dem Fehlercode 8809 abgebrochen. Kann bei der Druckmethode **’engraved_darkfield’** keines der Grauwertmerkmale die Testregionen trennen, bricht der Operator mit dem Fehlercode 8811 ab.

Parallelisierungsinformation

`get_2d_bar_code_pos` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

find_2d_bar_code

Mögliche Nachfolgerfunktionen

decode_2d_bar_code

Alternativen

get_2d_bar_code

Siehe auch

gen_2d_bar_code_descr

Modul

Barcode reader

12.3 Fourier-Deskriptor

```
abs_invar_fourier_coeff ( : :  RealInvar, ImaginaryInvar, CoefP, CoefQ,
AZInvar :  RealAbsInvar, ImaginaryAbsInvar )
```

Normalisiert die Fourierkoeffizienten bzgl. der Aufpunktverschiebungen.

abs_invar_fourier_coeff normalisiert die Fourierkoeffizienten bzgl. der Aufpunktverschiebungen. Diese kommen zustande, wenn ein Objekt gedreht wird. Der Konturverfolger get_region_contour beginnt mit der Aufzeichnung der Kontur im linken oberen Eck der Region und fährt die Kontur im Uhrzeigersinn entlang. Wird das Objekt nun gedreht, dann ändert sich der Startwert der Konturpunktkette, was zu einer Phasenverschiebung im Frequenzraum führt. Hierbei kann zwischen zwei Normalisierungsarten gewählt werden:

abs_amount: Phaseninformation wird eliminiert; Normalisierung ist nicht strukturerhaltend, d.h. bei Rücktransformation der AZ-Invarianten ist keine Ähnlichkeit mit der Musterregion mehr zu erkennen.

az_invar1: AZ-Invarianten 1. Ordnung normalisieren bzgl. der Aufpunktverschiebungen strukturerhaltend, sind jedoch anfälliger gegenüber lokalen und globalen Störungen, insbesondere gegenüber projektiven Verzerrungen.

Parameter

- ▷ **RealInvar** (input_control) . . . . . real-array ⇝ *real*
  Realteile der normalisierten Fourierkoeffizienten.
- ▷ **ImaginaryInvar** (input_control) . . . . . real-array ⇝ *real*
  Imaginärteile der normalisierten Fourierkoeffizienten.
- ▷ **CoefP** (input_control) . . . . . integer ⇝ *integer*
  Normalisierungskoeffizienten p.
  Defaultwert : 1
  Wertevorschläge : CoefP ∈ {1, 2}
  Restriktion : CoefP ≥ 1
- ▷ **CoefQ** (input_control) . . . . . integer ⇝ *integer*
  Normalisierungskoeffizienten q.
  Defaultwert : 1
  Wertevorschläge : CoefQ ∈ {1, 2}
  Restriktion : (CoefQ ≥ 1) ∧ (CoefQ ≠ CoefP)
- ▷ **AZInvar** (input_control) . . . . . string ⇝ *string*
  Ordnung der AZ-Invarianten.
  Defaultwert : 'abs_amount'
  Werteliste : AZInvar ∈ {'abs_amount', 'az_invar1'}
- ▷ **RealAbsInvar** (output_control) . . . . . real-array ⇝ *real*
  Realteile der normalisierten Fourierkoeffizienten.
- ▷ **ImaginaryAbsInvar** (output_control) . . . . . real-array ⇝ *real*
  Imaginärteile der normalisierten Fourierkoeffizienten.

Beispiel

```
get_region_contour(single,&row,&col);
length_of_contour = length_tuple(row);
move_contour_orig(row,col,&trow,&tcol);
prep_contour_fourier(trow,tcol,"unsigned_area",&param_scale);
fourier_1dim(trow,tcol,param_scale,50,&frow,&fcol);
invar_fourier_coeff(frow,fcol,1,"affine_invar",&invrow,&invcol);
abs_invar_fourier_coeff(invrow,invcol,1,2,"az_invar1",&absrow,&abscol);
fourier_1dim_inv(absrow,abscol,length_of_contour,&fsynrow,&fsyncol);
```

Parallelisierungsinformation

`abs_invar_fourier_coeff` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`invar_fourier_coeff`

Mögliche Nachfolgerfunktionen

`fourier_1dim_inv`, `match_fourier_coeff`

Modul

Fourier descriptors

`fourier_1dim` (: : `Rows, Columns, ParContour, MaxCoef` : `RealCoef, ImaginaryCoef`)

Berechnet die Fourierkoeffizienten einer parametrisierten Kontur.

`fourier_1dim` berechnet die Fourierkoeffizienten einer parametrisierten Kontur unter Verwendung einer gültigen Parameterskala. Diese kann z.B. mit der Prozedur `prep_contour_fourier` gewonnen werden. Es handelt sich bei dieser Funktion um die Berechnung der Fourierkoeffizienten geschlossener Konturen, die als komplexwertige Kurven aufgefaßt werden. Daher wird bei der Ermittlung der Fourierkoeffizienten die Fouriertransformation periodischer Funktionen verwendet. Der Parameter `MaxCoef` bestimmt hierbei den $Absolutwert + 1$ der maximalen Anzahl der Fourierkoeffizienten, d.h. werden n Koeffizienten angegeben, so berechnet die Prozedur Koeffizienten von $-n$ bis n. Die Kontur wird verlustfrei approximiert, wenn $n = AnzahlderKonturpunkte$, während bereits $n = 100$ die Kontur so gut nähert, daß ein Fehler kaum mehr zu erkennen ist; $n \in [40, 50]$ ist jedoch für die meisten Anwendungen ausreichend. Gibt man für `MaxCoef` den Wert 0 an, so werden alle Koeffizienten bestimmt.

Parameter

- ▷ **`Rows`** (input_control) ..... contour.y-array ⇝ *integer*
  Zeilenkoordinaten der Kontur.
- ▷ **`Columns`** (input_control) ..... contour.x-array ⇝ *integer*
  Spaltenkoordinaten der Kontur.
- ▷ **`ParContour`** (input_control) ..... real-array ⇝ *real*
  Parameterskala.
- ▷ **`MaxCoef`** (input_control) ..... integer ⇝ *integer*
  Gewünschten Anzahl von Fourierkoeffizienten oder alle (0).
  Defaultwert : 50
  Wertevorschläge : `MaxCoef` ∈ {0, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 400}
  Restriktion : `MaxCoef` ≥ `0`
- ▷ **`RealCoef`** (output_control) ..... real-array ⇝ *real*
  Realteile der Fourierkoeffizienten.
- ▷ **`ImaginaryCoef`** (output_control) ..... real-array ⇝ *real*
  Imaginärteile der Fourierkoeffizienten.

Beispiel

```
get_region_contour(single,&row,&col);
move_contour_orig(row,col,&trow,&tcol);
```

```
prep_contour_fourier(trow,tcol,"unsigned_area",&param_scale);
fourier_1dim(trow,tcol,param_scale,&frow,&fcol);
invar_fourier_coeff(frow,fcol,1,"affine_invar",&invrow,&invcol);
abs_invar_fourier_coeff(invrow,invcol,1,2,"az_invar1",&absrow,&abscol);
```

Parallelisierungsinformation

`fourier_1dim` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`prep_contour_fourier`

Mögliche Nachfolgerfunktionen

`invar_fourier_coeff`, `disp_polygon`

Modul

Fourier descriptors

`fourier_1dim_inv ( : : RealCoef, ImaginaryCoef, MaxCoef : Rows, Columns )`

Eindimensionale Fouriersynthese (inverse Fouriertransformation).

Rücktransformation der Fourierkoeffizienten, bzw. Fourierdeskriptoren. Die Anzahl der rückzutransformierenden Werte sollte die Länge der transformierten Kontur nicht übersteigen.

Parameter

- ▷ **`RealCoef`** (input_control) ..... real-array ⇝ *real*
  Realteile.
- ▷ **`ImaginaryCoef`** (input_control) ..... real-array ⇝ *real*
  Imaginärteile.
- ▷ **`MaxCoef`** (input_control) ..... integer ⇝ *integer*
  Eingabe der Rücktransformationsschritte.
  Defaultwert : 100
  Wertevorschläge : `MaxCoef` ∈ {5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 400}
  Restriktion : `MaxCoef` ≥ `1`
- ▷ **`Rows`** (output_control) ..... contour.y-array ⇝ *real*
  Zeilenkoordinaten.
- ▷ **`Columns`** (output_control) ..... contour.x-array ⇝ *real*
  Spaltenkoordinaten.

Beispiel

```
get_region_contour(single,&row,&col);
length_of_contour = row.Num();
move_contour_orig(row,col,&trow,&tcol);
prep_contour_fourier(trow,tcol,"unsigned_area",&param_scale);
fourier_1dim(trow,tcol,param_scale,50,&frow,&fcol);
invar_fourier_coeff(frow,fcol,1,"affine_invar",&invrow,&invcol);
abs_invar_fourier_coeff(invrow,invcol,1,2,"az_invar1",&absrow,&abscol);
fourier_1dim_inv(absrow,abscol,length_of_contour,&fsynrow,&fsyncol);
```

Parallelisierungsinformation

`fourier_1dim_inv` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`invar_fourier_coeff`, `fourier_1dim`

Mögliche Nachfolgerfunktionen

`disp_polygon`

Modul

Fourier descriptors

invar_fourier_coeff (: : RealCoef, ImaginaryCoef, NormPar, InvarType : RealInvar, ImaginaryInvar)

Normalisierung der Fourierkoeffizienten.

Eliminieren affiner Information aus den Fourierkoeffizienten, ermitteln der Affin-Invarianten. Die Fourierkoeffizienten werden geeignet normalisiert, so daß alle zueinander affinen Konturen auf diesselbe Kontur abgebildet werden. Stufen der affinen Abbildungen sind:

1. Translationen (InvarType = 'transl_invar')
2. + Rotationen (InvarType = 'congr_invar')
3. + Skalierungen (InvarType = 'simil_invar')
4. + Scherungen (InvarType = 'affine_invar')

Mit dem Steuerparameter InvarType wird angegeben, bzgl. welcher Schicht affiner Abbildungen normalisiert werden soll. Hierbei ist zu beachten, das bei Angabe einer Schicht bzgl. aller darunterliegenden Schichten normalisiert wird. Eine nachfolgende Aufpunktnormalisierung ist für die meisten Anwendungen zu empfehlen!

Parameter

- ▷ **RealCoef** (input_control) ..... real-array ⇝ *real*
  Realteile der Fourierkoeffizienten.
- ▷ **ImaginaryCoef** (input_control) ..... real-array ⇝ *real*
  Imaginärteile der Fourierkoeffizienten.
- ▷ **NormPar** (input_control) ..... integer ⇝ *integer*
  Eingabe des Normalisierungskoeffizienten
  Defaultwert : 1
  Wertevorschläge : NormPar ∈ {1, 2}
  Restriktion : NormPar ≥ 1
- ▷ **InvarType** (input_control) ..... string ⇝ *string*
  Angabe der Stufe der affinen Abbildungen.
  Defaultwert : 'affine_invar'
  Werteliste : InvarType ∈ {'affine_invar', 'simil_invar', 'congr_invar', 'transl_invar'}
- ▷ **RealInvar** (output_control) ..... real-array ⇝ *real*
  Realteile der normalisierten Fourierkoeffizienten.
- ▷ **ImaginaryInvar** (output_control) ..... real-array ⇝ *real*
  Imaginärteile der normalisierten Fourierkoeffizienten.

Beispiel

```
prep_contour_fourier(trow,tcol,"unsigned_area",&param_scale);
fourier_1dim(trow,tcol,param_scale,&frow,&fcol);
invar_fourier_coeff(frow,fcol,1,"affine_invar",&invrow,&invcol);
abs_invar_fourier_coeff(invrow,invcol,1,2,"az_invar1",&absrow,&abscol);
```

Parallelisierungsinformation

invar_fourier_coeff ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

fourier_1dim

Mögliche Nachfolgerfunktionen

invar_fourier_coeff

Modul

Fourier descriptors

match_fourier_coeff (: : RealCoef1, ImaginaryCoef1, RealCoef2, ImaginaryCoef2, MaxCoef, Damping : Distance)

Ähnlichkeit zweier Konturen.

match_fourier_coeff berechnet den euklidischen Abstand zweier Konturen, die als Fourierkoeffizienten vorliegen. Um zu vermeiden, daß die höheren Frequenzen zu stark mit eingehen, können folgende Dämpfungen verwendet werden:

none: Keine Dämpfung.

1/index: Absolutbeträge der Fourierkoeffizienten werden durch ihren Index geteilt.

1/index*index: Absolutbeträge der Fourierkoeffizienten werden durch ihren Quadratindex geteilt.

Je höher der Ergebniswert ist, desto unterschiedlicher sind Muster- und Testkontur. Unterscheidet sich die Anzahl der Koeffizienten, so werden nur die ersten n Koeffizienten verglichen. MaxCoef gibt die Anzahl der Koeffizienten vor, die zum Vergleich verwendet werden. Wird Null angegeben, so werden alle Koeffizienten verwendet.

Parameter

- ▷ **RealCoef1** (input_control) ..... real-array ⇝ *real*
  Realteile der Muster-Fourierkoeffizienten.
- ▷ **ImaginaryCoef1** (input_control) ..... real-array ⇝ *real*
  Imaginärteile der Muster-Fourierkoeffizienten.
- ▷ **RealCoef2** (input_control) ..... real-array ⇝ *real*
  Realteile der Vergleichs-Fourierkoeffizienten.
- ▷ **ImaginaryCoef2** (input_control) ..... real-array ⇝ *real*
  Imaginärteile der Vergleichs-Fourierkoeffizienten.
- ▷ **MaxCoef** (input_control) ..... integer ⇝ *integer*
  Anzahl der Fourierkoeffizienten
  Defaultwert : 50
  Wertevorschläge : MaxCoef ∈ {0, 5, 10, 15, 20, 30, 40, 50, 70, 100, 200, 400}
  Restriktion : MaxCoef ≥ 0
- ▷ **Damping** (input_control) ..... string ⇝ *string*
  Art der Dämpfung.
  Defaultwert : ”1/index”
  Wertevorschläge : Damping ∈ {’none’, ”1/index”, ”1/(index*index)”}
- ▷ **Distance** (output_control) ..... real ⇝ *real*
  Ähnlichkeit der Konturen.

Beispiel

```
prep_contour_fourier(trow,tcol,"unsigned_area",&param_scale);
fourier_1dim(trow,tcol,param_scale,50,&frow,&fcol);
invar_fourier_coeff(frow,fcol,1,"affine_invar",&invrow,&invcol);
abs_invar_fourier_coeff(invrow,invcol,1,2,
                        "az_invar1",&absrow,&abscol);
match_fourier_coeff(contur1_row, contur1_col,
                    contur2_row, contur2_col, 50,
                    "1/index", &Distance_wert);
```

Parallelisierungsinformation

match_fourier_coeff ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

invar_fourier_coeff

Modul

Fourier descriptors

move_contour_orig (: : Rows, Columns : RowsMoved, ColumnsMoved)

Transformation des Koordinatenursprunges in den Flächenschwerpunkt.

move_contour_orig verschiebt die eingegebene Kontur so, daß der Koordinatenursprung im Flächenschwerpunkt liegt.

Parameter

- ▷ **Rows** (input_control) . . . . . contour.y-array ⇝ *integer*
  Zeilenkoordinaten der Kontur.
- ▷ **Columns** (input_control) . . . . . contour.x-array ⇝ *integer*
  Spaltenkoordinaten der Kontur.
- ▷ **RowsMoved** (output_control) . . . . . contour.y-array ⇝ *integer*
  Zeilenkoordinaten der verschobenen Kontur.
- ▷ **ColumnsMoved** (output_control) . . . . . contour.x-array ⇝ *integer*
  Spaltenkoordinaten der verschobenen Kontur.

Parallelisierungsinformation

`move_contour_orig` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`get_region_contour`

Mögliche Nachfolgerfunktionen

`prep_contour_fourier`

Modul

Fourier descriptors

prep_contour_fourier (: : Rows, Columns, TransMode : ParContour)

Berechnet eine Parameterskala der übergebenen Kontur.

`prep_contour_fourier` parametrisiert die übergebene Kontur, um diese für die eindimensionale Fouriertransformation vorzubereiten. Die Kontur muß hierbei geschlossen vorliegen. Es stehen drei Parameterfunktionen über den Steuerparameter `TransMode` zur Auswahl:

arc: Parametrisierung mit dem *Bogenmaß*.

signed_area: Parametrisierung mit der *vorzeichenbehafteten Fläche*.

unsigned_area: Parametrisierung mit der *absoluten Fläche*.

Hierbei ist zu beachten, daß das Bogenmaß, im Gegensatz zu der vorzeichenbehafteten bzw. vorzeichenfreien Fläche, unter den affinen Abbildungen nicht linear transformiert wird.

Parameter

- ▷ **Rows** (input_control) . . . . . contour.y-array ⇝ *integer*
  Zeilenindizes der Kontur
- ▷ **Columns** (input_control) . . . . . contour.x-array ⇝ *integer*
  Spaltenindizes der Kontur
- ▷ **TransMode** (input_control) . . . . . string ⇝ *string*
  Art der Parametrisierung
  Defaultwert : ’signed_area’
  Wertevorschläge : `TransMode` ∈ {’arc’, ’unsigned_area’, ’signed_area’}
- ▷ **ParContour** (output_control) . . . . . real-array ⇝ *real*
  Parametrisierte Kontur

Beispiel

```
get_region_contour(single,&row,&col);
move_contour_orig(row,col,&trow,&tcol);
prep_contour_fourier(trow,tcol,"unsigned_area",&param_scale);
fourier_1dim(trow,tcol,param_scale,&frow,&fcol);
```

Parallelisierungsinformation

`prep_contour_fourier` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

move_contour_orig

Mögliche Nachfolgerfunktionen

fourier_1dim

Modul

Fourier descriptors

12.4 Funktion

abs_funct_1d (: : Function : FunctionAbsolute)

Absolutbetrag der y-Werte.

abs_funct_1d berechnet die Absolutbeträge der y-Werte der Funktion Function.

Parameter

- ▷ **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Eingabefunktion.
- ▷ **FunctionAbsolute** (output_control) ..... function_1d-array ⇝ *real* / integer
  Funktion mit den Absolutbeträgen der y-Werte.

Parallelisierungsinformation

abs_funct_1d ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_funct_1d_pairs, create_funct_1d_array

Modul

Tools

create_funct_1d_array (: : YValues : Function)

Erzeugen einer Funktion aus einer Folge von y-Werten.

create_funct_1d_array erzeugt aus einer Folge von y-Werten eine eindimensionale Funktion. Diese Funktion kann dann mit den Operatoren für Funktionen verarbeitet und analysiert werden. Die eingegebenen Werte werden in der folgenden Weise interpretiert: Der erste Wert von YValues ist der y-Wert an der Stelle Null, der zweite Wert an der Stelle Eins usw. Die Werte erzeugen also eine Funktion aufgebaut aus äquidistanten Stützstellen (mit Abstand 1) die bei dem x-Wert Null beginnt.

Alternativ kann eine Funktion auch mit dem Operator create_funct_1d_pairs erzeugt werden, der durch die explizite Angabe der x-Werte prinzipiell auch nicht-äquidistante Stützstellen ermöglicht. Um dieselbe Spezifikation zu erhalten, wie sie bei create_funct_1d_array definiert wird, müßten die x-Werte bei create_funct_1d_pairs daher als ein Tupel übergeben werden, das beginnend bei Null um 1 aufsteigende Werte enthält. Es ist aber zu beachten, daß die Verwendung von create_funct_1d_pairs zu einer anderen internen Darstellung der Funktion führt, die mehr Speicher bedarf (da alle Paare gespeichert werden) und die in der Verarbeitung etwas langsamer sein kann.

Um aus einer mittels create_funct_1d_array erzeugten Funktion mit äquidistanten Stützstellen des Abstandes 1 eine neue zu generieren, die einen anderen Abstand bei den äquidistanten Stützstellen aufweist, kann der Operator transform_funct_1d verwendet werden.

Parameter

- ▷ **YValues** (input_control) ..... number(-array) ⇝ *real* / integer
  X-Werte der Stützstellen.
- ▷ **Function** (output_control) ..... function_1d-array ⇝ *real* / integer
  Erzeugte Funktion.

Parallelisierungsinformation

create_funct_1d_array ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

write_funct_1d, gnuplot_plot_funct_1d, y_range_funct_1d, get_pair_funct_1d, transform_funct_1d

Alternativen

create_funct_1d_pairs, read_funct_1d

Siehe auch

funct_1d_to_pairs

Modul

Tools

create_funct_1d_pairs (: : XValues, YValues : Function)

Erzeugen einer Funktion aus Paaren von (x,y)-Werten.

create_funct_1d_pairs erzeugt aus Paaren von (x,y)-Werten eine eindimensionale Funktion. Die x-Werte XValues der Funktion müssen aufsteigend sortiert sein. Die resultierende Funktion kann dann mit den Operatoren für Funktionen verarbeitet und analysiert werden.

Alternativ kann eine Funktion mit dem Operator create_funct_1d_array erzeugt werden. Im Gegensatz zu diesem Operator können hier die Stützpunkte jedoch beliebige Abstände haben. create_funct_1d_pairs ist also der allgemeinere Operator. Dabei ist jedoch zu beachten, daß die Verarbeitung einer mit create_funct_1d_pairs erzeugten Funktion aufgrund dieser Allgemeinheit oftmals nicht so effizient ausgeführt werden kann. Insbesondere sind nicht alle Verarbeitungen implementiert. Gegebenenfalls kann die Funktion mit dem Operator sample_funct_1d in eine äquidistante Darstellung umgewandelt werden.

Parameter

- ▷ **XValues** (input_control) ..... number(-array) ⇝ *real* / integer
  X-Werte der Stützstellen.
- ▷ **YValues** (input_control) ..... number(-array) ⇝ *real* / integer
  Y-Werte der Stützstellen.
- ▷ **Function** (output_control) ..... function_1d-array ⇝ *real* / integer
  Erzeugte Funktion.

Parallelisierungsinformation

create_funct_1d_pairs ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

write_funct_1d, gnuplot_plot_funct_1d, y_range_funct_1d, get_pair_funct_1d

Alternativen

create_funct_1d_array, read_funct_1d

Siehe auch

funct_1d_to_pairs

Modul

Tools

distance_funct_1d (: : Function1, Function2, Mode, Sigma : Distance)

Abstand zweier Funktionen.

distance_funct_1d berechnet den Abstand zweier Funktionen. Die beiden Funktionen können verschieden lang sein.

Parameter

- ▷ **Function1** (input_control) ..... function_1d-array ⇝ *real* / integer
  Eingabefunktion 1.

▷ **Function2** (input_control) . . . . . function_1d-array ⇝ *real* / integer
Eingabefunktion 2.

▷ **Mode** (input_control) . . . . . string(-array) ⇝ *string*
Art der Invarianten.
Defaultwert : 'length'
Werteliste : Mode ∈ {'length', 'mean'}

▷ **Sigma** (input_control) . . . . . number(-array) ⇝ *real*
Standardabweichung der optionalen Gaußglättung
Defaultwert : 0.0
Wertevorschläge : Sigma ∈ {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}

▷ **Distance** (output_control) . . . . . real-array ⇝ *real* / integer
Abstand der Funktionen.

Parallelisierungsinformation

distance_funct_1d ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Modul

Tools

funct_1d_to_pairs (: : Function : XValues, YValues)

Zugriff auf die x/y-Werte einer Funktion.

funct_1d_to_pairs zerlegt die Eingabefunktion Function in Tupel von x- und y- Werten.

Parameter

▷ **Function** (input_control) . . . . . function_1d-array ⇝ *real* / integer
Eingabefunktion.

▷ **XValues** (output_control) . . . . . number-array ⇝ *real* / integer
X-Werte der Funktion.

▷ **YValues** (output_control) . . . . . number-array ⇝ *real* / integer
Y-Werte der Funktion.

Parallelisierungsinformation

funct_1d_to_pairs ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_funct_1d_pairs, create_funct_1d_array

Modul

Tools

get_pair_funct_1d (: : Function, Index : X, Y)

Zugriff auf Funktionswerte über den Index der Stützstelle.

get_pair_funct_1d greift auf einen Funktionswert der Funktion Function zu. Hierzu wird der Index einer oder mehrerer Stützstellen angegeben.

Parameter

▷ **Function** (input_control) . . . . . function_1d-array ⇝ *real* / integer
Eingabefunktion.

▷ **Index** (input_control) . . . . . integer(-array) ⇝ *integer*
Index der Stützstelle(n).

▷ **X** (output_control) . . . . . number(-array) ⇝ *real*
X-Wert an der vorgegebenen Stützstelle.

▷ **Y** (output_control) . . . . . number(-array) ⇝ *real*
Y-Wert an der vorgegebenen Stützstelle.

Parallelisierungsinformation

`get_pair_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Tools

integrate_funct_1d (: : Function : Positive, Negative)

Berechnung der positiven und negativen Flächen unter einer Funktion.

`integrate_funct_1d` integriert die Funktion`Function` (siehe `create_funct_1d_array` und `create_funct_1d_pairs`) und gibt das Integral der positiven und negativen Teile der Funktion in `Positive` bzw. `Negative` zurück. Das Integral der Funktion ist also die Differenz `Positive` - `Negative`. Die Integration wird über das Intervall, auf dem die Funktion definiert ist, ausgeführt. Zur Integration wird die Funktion linear interpoliert.

Parameter

- ▷ **Function** (input_control) ..... function_1d-array ⇝ *real*
  Eingabefunktion.
- ▷ **Positive** (output_control) ..... number ⇝ *real*
  Fläche unter dem positiven Teil der Funktion.
- ▷ **Negative** (output_control) ..... number-array ⇝ *real*
  Fläche unter dem negativen Teil der Funktion.

Parallelisierungsinformation

`integrate_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Siehe auch

`create_funct_1d_array`, `create_funct_1d_pairs`

Modul

Tools

match_funct_1d_trans (: : Function1, Function2, Border, ParamsConst, UseParams : Params, ChiSquare, Covar)

Berechnung der Transformationsparameter zwischen zwei Funktionen.

`match_funct_1d_trans` berechnet die Transformationsparameter zwischen zwei Funktionen, die als die Tupel `Function1` und `Function2` übergeben werden (siehe `create_funct_1d_array` und `create_funct_1d_pairs`). Es wird folgendes Modell der Transformation zwischen den zwei Funktionen verwendet:

$$y_1(x) = a_1 y_2(a_3 x + a_4) + a_2 \ .$$

Die Transformationsparameter werden durch eine Ausgleichsrechnung berechnet, indem die folgende Funktion minimiert wird:

$$\sum_{i=0}^{n-1} \left(y_1(x_i) - a_1 y_2(a_3 x_i + a_4) + a_2 \right)^2 \ .$$

Dabei werden die Funktionswerte von y_2 linear interpoliert. Der Parameter `Border` entscheidet, welchen Wert die Funktion `Function2` außerhalb des gültigen Bereichs hat. Für `Border`=**’zero’** wird der Wert auf 0 gesetzt, für `Border`=**’constant’** auf den jeweiligen Randwert, für `Border`=**’mirror’** werden die Funktionswerte am Rand

gespiegelt und für Border=**'cyclic'** werden sie zyklisch fortgesetzt. Die berechneten Transformationsparameter werden als Tupel der Länge 4 in Params zurückgeliefert. Falls einige der Parameter bekannte Werte besitzen, so kann der jeweilige Parameter von der Ausgleichsrechnung ausgeschlossen werden, indem an der entsprechenden Stelle im Tupel UseParams der Wert **'false'** eingetragen wird. In diesem Fall muß im Tupel ParamsConst der bekannte Wert des Parameters eingetragen werden. Wenn der Parameter für die Ausgleichung verwendet wird (UseParams = **'true'**), wird der entsprechende Parameter in ParamsConst ignoriert. Als Ausgabe liefert match_funct_1d_trans weiterhin die Summe der quadratischen Fehler ChiSquare der Ergebnisfunktion, also der mit den Transformationsparametern transformierten Funktion, sowie die Kovariazmatrix Covar der Transformationsparameter Params zurück. Diese Werte können verwendet werden, um festzustellen, ob ein erfolgreiches Matching der Funktionen möglich war.

Parameter

- ▷ **Function1** (input_control) . . . . . function_1d-array ⇝ *real* / integer
  Funktion 1.
- ▷ **Function2** (input_control) . . . . . function_1d-array ⇝ *real* / integer
  Funktion 2.
- ▷ **Border** (input_control) . . . . . string ⇝ *string*
  Randbehandlung für Funktion 2.
  Defaultwert : 'constant'
  Werteliste : Border ∈ {'zero', 'constant', 'mirror', 'cyclic'}
- ▷ **ParamsConst** (input_control) . . . . . number-array ⇝ *real*
  Werte der Parameter, die konstant gehalten werden.
  Defaultwert : '[1.0,0.0,1.0,0.0]'
  Parameteranzahl : 4
- ▷ **UseParams** (input_control) . . . . . string-array ⇝ *string*
  Soll ein Parameter angepaßt werden?
  Defaultwert : '['true','true','true','true']'
  Werteliste : UseParams ∈ {'true', 'false'}
  Parameteranzahl : 4
- ▷ **Params** (output_control) . . . . . number-array ⇝ *real*
  Transformationsparameter zwischen den Funktionen.
  Parameteranzahl : 4
- ▷ **ChiSquare** (output_control) . . . . . number ⇝ *real*
  Quadratischer Fehler der Ausgabefunktion.
- ▷ **Covar** (output_control) . . . . . number-array ⇝ *real*
  Kovarianzmatrix der Transformationsparameter
  Parameteranzahl : 16

Parallelisierungsinformation

match_funct_1d_trans ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_funct_1d_array, create_funct_1d_pairs

Siehe auch

gray_projections

Modul

Tools

negate_funct_1d (: : Function : FunctionInverted)

Negation aller y-Werte.

negate_funct_1d negiert alle y-Werte der Funktion Function.

Parameter

- ▷ **Function** (input_control) . . . . . function_1d-array ⇝ *real* / integer
  Eingabefunktion.

▷ **FunctionInverted** (output_control) . . . . . function_1d-array ⇝ *real* / integer
Funktion mit den negierten y-Werten.

Parallelisierungsinformation

`negate_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Tools

num_points_funct_1d (: : Function : Length)

Anzahl der Stützstellen der Funktion.

`num_points_funct_1d` bestimmt die Anzahl von Stützstellen der Funktion `Function`.

Parameter

▷ **Function** (input_control) . . . . . function_1d-array ⇝ *real* / integer
Eingabefunktion.
▷ **Length** (output_control) . . . . . integer ⇝ *integer*
Anzahl der Stützstellen.

Parallelisierungsinformation

`num_points_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Tools

read_funct_1d (: : FileName : Function)

Eine Funktion von Datei lesen.

`read_funct_1d` liest den Inhalt der Datei `FileName` und wandelt diesen in die Funktion `Function`, dargestellt als Tupel, um. Die Datei muß mit `write_funct_1d` erzeugt worden sein.

Parameter

▷ **FileName** (input_control) . . . . . filename ⇝ *string*
Name der zu lesenden Datei.
▷ **Function** (output_control) . . . . . function_1d(-array) ⇝ *real* / integer
Eingelesene Funktion.

Ergebnis

Sind die Parameter korrekt, dann liefert `read_funct_1d` den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, dann liefert `read_funct_1d` den Wert 5 (H_MSG_FAIL). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`fread_string`, `read_tuple`

Siehe auch

`write_funct_1d`, `gnuplot_plot_ctrl`, `write_image`, `write_region`, `open_file`

Modul

Basic operators

sample_funct_1d (: : Function, XMin, XMax, XDist, Border : SampledFunction)

Taste eine Funktion in einem Intervall äquidistant ab.

sample_funct_1d tastet die Eingabefunktion Function im Intervall [XMin,XMax] an äquidistanten Stützstellen mit dem Abstand XDist ab. Die letzte Stützstelle liegt im Intervall, falls XMax-XMin kein ganzzahliges Vielfaches von XDist ist. Bei der Abtastung wird die Eingabefunktion linear interpoliert. Der Parameter Border entscheidet, welchen Wert die Funktion Function außerhalb des gültigen Bereichs hat. Für Border=**'zero'** wird der Wert auf 0 gesetzt, für Border=**'constant'** auf den jeweiligen Randwert, für Border=**'mirror'** werden die Funktionswerte am Rand gespiegelt und für Border=**'cyclic'** werden sie zyklisch fortgesetzt.

Parameter

- ▷ **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Eingabefunktion.
- ▷ **XMin** (input_control) ..... number ⇝ *real* / integer
  Minimaler x-Wert der Ausgabefunktion.
- ▷ **XMax** (input_control) ..... number ⇝ *real* / integer
  Maximaler x-Wert der Ausgabefunktion.
  Restriktion : XMax > XMin
- ▷ **XDist** (input_control) ..... number ⇝ *real* / integer
  Abstand der Abtastwerte.
  Restriktion : XDist > 0
- ▷ **Border** (input_control) ..... string ⇝ *string*
  Randbehandlung für die Eingabefunction.
  Defaultwert : 'constant'
  Werteliste : Border ∈ {'zero', 'constant', 'mirror', 'cyclic'}
- ▷ **SampledFunction** (output_control) ..... function_1d-array ⇝ *real*
  Abgetastete Funktion.

Parallelisierungsinformation

sample_funct_1d ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

transform_funct_1d, create_funct_1d_array, create_funct_1d_pairs

Modul

Tools

scale_y_funct_1d (: : Function, Mult, Add : FunctionScaled)

Multiplizieren und Addieren der y-Werte der Funktion.

scale_y_funct_1d multipliziert und addiert die y-Werte der Funktion Function mit den Parametern Mult und Add.

Parameter

- ▷ **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Eingabefunktion.
- ▷ **Mult** (input_control) ..... number ⇝ *real*
  Faktor für Skalierung der y-Werte.
  Defaultwert : 2
  Wertevorschläge : Mult ∈ {0.1, 0.3, 0.5, 1, 2, 5, 10}
- ▷ **Add** (input_control) ..... number ⇝ *real*
  Konstante, die auf die y-Werte addiert wird.
  Defaultwert : 0
  Wertevorschläge : Add ∈ {-10, -5, 1, 0, 5, 10}
- ▷ **FunctionScaled** (output_control) ..... function_1d-array ⇝ *real* / integer
  Transformierte Funktion.

Parallelisierungsinformation

`scale_y_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Tools

smooth_funct_1d_gauss (: : Function, Sigma : SmoothedFunction)

Glättung einer äquidistanten Funktion mit einer Gaußfunktion.

`smooth_funct_1d_gauss` führt die Glättung einer eindimensionalen Funktion durch die Anwendung einer Gaußfunktion durch.

Parameter

- ▷ **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Zu glättende Funktion.
- ▷ **Sigma** (input_control) ..... number ⇝ *real*
  Sigma der Gaußfunktion für die Glättung.
  Defaultwert : 2.0
  Wertevorschläge : `Sigma` ∈ {0.5, 1.0, 2.0, 3.0, 4.0, 5.0}
  Typischer Wertebereich : 0.1 ≤ `Sigma` ≤ 50.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.2
- ▷ **SmoothedFunction** (output_control) ..... function_1d-array ⇝ *real*
  Geglättete Funktion.

Parallelisierungsinformation

`smooth_funct_1d_gauss` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Mögliche Nachfolgerfunktionen

`match_funct_1d_trans`, `distance_funct_1d`

Modul

Tools

smooth_funct_1d_mean (: : Function, SmoothSize, Iterations : SmoothedFunction)

Glättung einer 1D Funktion durch Mittelung.

`smooth_funct_1d_mean` führt die Glättung einer 1-dimensionalen Funktion mit Hilfe der mehrmaligen Anwendung eines Mittelwertfilters durch.

Parameter

- ▷ **Function** (input_control) ..... function_1d-array ⇝ *integer* / real
  1D Funktion.
- ▷ **SmoothSize** (input_control) ..... integer ⇝ *integer*
  Größe der Glättungsmaske.
  Defaultwert : 10
  Wertevorschläge : `SmoothSize` ∈ {1, 3, 5, 7, 9, 11, 13, 15, 21, 31, 51}
  Typischer Wertebereich : 1 ≤ `SmoothSize` ≤ 1000 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Restriktion : `SmoothSize` > `0`

▷ **Iterations** (input_control) ..... integer ⇝ *integer*
Iterationen der Glättung.
Defaultwert : 3
Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9}
Typischer Wertebereich : 1 ≤ `Iterations` ≤ 100 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Restriktion : `Iterations` ≥ 1

▷ **SmoothedFunction** (output_control) ..... function_1d-array ⇝ *real*
Geglättete Funktion.

Parallelisierungsinformation

`smooth_funct_1d_mean` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_array`

Alternativen

`smooth_funct_1d_gauss`

Modul

Tools

transform_funct_1d (: : Function, Params : TransformedFunction)

Transformiere eine Funktion mit gegebenen Transformationsparametern.

`transform_funct_1d` transformiert die Eingabefunktion `Function` mit den Transformationsparametern, die in `Params` übergeben werden. Die Funktion `Function` wird dabei als Tupel übergeben (siehe `create_funct_1d_array` und `create_funct_1d_pairs`). Es wird folgendes Modell der Transformation zwischen den zwei Funktionen verwendet (siehe `match_funct_1d_trans`):

$$y_t(x) = a_1 y(a_3 x + a_4) + a_2 \ .$$

Die Ausgabefunktion `TransformedFunction` entsteht dadurch, daß die x- und y-Werte der Eingabefunktion getrennt nach obiger Formel transformiert werden, d.h. die Ausgabefunktion wird nicht neu abgetastet. Hierfür steht der Operator `sample_funct_1d` zur Verfügung.

Parameter

▷ **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
Eingabefunktion.

▷ **Params** (input_control) ..... number-array ⇝ *real*
Transformationsparameter zwischen den Funktionen.
Parameteranzahl : 4

▷ **TransformedFunction** (output_control) ..... function_1d-array ⇝ *real* / integer
Transformierte Funktion.

Parallelisierungsinformation

`transform_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`, `match_funct_1d_trans`

Modul

Tools

write_funct_1d (: : Function, FileName :)

Eine Funktion auf Datei schreiben.

write_funct_1d schreibt den Inhalt von Function auf Datei. Es handelt sich hierbei um ein ASCII-Format, d.h. die Daten sind zwischen unterschiedlichen Rechnertypen austauschbar. Die Daten können mit dem Operator read_funct_1d wieder eingelesen werden. Für den Dateiname ist keine Extension festgelegt.

Parameter

- **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Funktion die geschrieben werden soll.
- **FileName** (input_control) ..... filename ⇝ *string*
  Name der zu schreibenden Datei.

Ergebnis

Sind die Parameter korrekt, dann liefert write_funct_1d den Wert 2 (H_MSG_TRUE). Kann die Datei nicht geöffnet werden, dann liefert write_funct_1d den Wert 5 (H_MSG_FAIL). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_funct_1d ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_funct_1d_pairs, create_funct_1d_array

Alternativen

write_tuple, fwrite_string

Siehe auch

read_funct_1d, write_image, write_region, open_file

Modul

Basic operators

x_range_funct_1d (: : Function : XMin, XMax)

Kleinster und größter x-Wert der Funktion.

x_range_funct_1d bestimmt den kleinsten und den größten x-Wert der Funktion Function.

Parameter

- **Function** (input_control) ..... function_1d-array ⇝ *real* / integer
  Eingabefunktion.
- **XMin** (output_control) ..... number ⇝ *real*
  Kleinster x-Wert.
- **XMax** (output_control) ..... number ⇝ *real*
  Größter x-Wert.

Parallelisierungsinformation

x_range_funct_1d ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_funct_1d_pairs, create_funct_1d_array

Modul

Tools

y_range_funct_1d (: : Function : YMin, YMax)

Kleinster und größter y-Wert der Funktion.

y_range_funct_1d bestimmt den kleinsten und den größten y-Wert der Funktion Function.

Parameter

- ▷ **Function** (input_control) . . . . . function_1d-array ⇝ *real* / integer
  Eingabefunktion.
- ▷ **YMin** (output_control) . . . . . number ⇝ *real*
  Kleinster y-Wert.
- ▷ **YMax** (output_control) . . . . . number ⇝ *real*
  Größter y-Wert.

Parallelisierungsinformation

`y_range_funct_1d` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_funct_1d_pairs`, `create_funct_1d_array`

Modul

Tools

12.5 Geometrie

angle_ll (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : Angle)

Berechnung des Winkels zwischen zwei Geraden.

`angle_ll` berechnet den Winkel zwischen zwei Geraden. Als Eingabe werden die Zeilen- und Spaltenwerte der ersten Gerade (`RowA1`,`ColumnA1`, `RowA2`,`ColumnA2`) und der zweiten Gerade (`RowB1`,`ColumnB1`, `RowB2`,`ColumnB2`) erwartet. Die Berechnung des Winkels geschieht wie folgt: Man interpretiert die beiden Geraden als Vektoren, wobei `RowA1`,`ColumnA1` bzw. `RowB1`,`ColumnB1` die Startpunkte und `RowA2`,`ColumnA2` bzw. `RowB2`,`ColumnB2` die Endpunkte sind. Dreht man nun den Vektor A gegen den Uhrzeigersinn auf den Vektor B (der Drehpunkt ist der Schnittpunkt der Geraden) so ist der benötigte Rotationswinkel der gesuchte Wert. Der Winkel ist also abhängig von der Reihenfolge der Koordinaten und der Reihenfolge der beiden Geraden. Das Ergebnis, d.h. der Winkel in Bogenmaß, wird im Parameter `Angle` übergeben. Die Winkel liegen im Bereich von $-\pi \leq$ `Angle` $\leq \pi$.

Parameter

- ▷ **RowA1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der ersten Gerade.
- ▷ **ColumnA1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der ersten Gerade.
- ▷ **RowA2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der ersten Gerade.
- ▷ **ColumnA2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der ersten Gerade.
- ▷ **RowB1** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der zweiten Gerade.
- ▷ **ColumnB1** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der zweiten Gerade.
- ▷ **RowB2** (input_control) . . . . . point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der zweiten Gerade.
- ▷ **ColumnB2** (input_control) . . . . . point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der zweiten Gerade.
- ▷ **Angle** (output_control) . . . . . number(-array) ⇝ *real*
  Winkel zwischen den Geraden.

Beispiel

```
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(rad(i)) * 200
  ColumnB2 := 255 + cos(rad(i)) * 200
  disp_line (WindowHandle, RowB1, ColumnB1, RowB2, ColumnB2)
  angle_ll (RowA1, ColumnA1, RowA2, ColumnA2,
            RowB1, ColumnB1, RowB2, ColumnB2, Angle)
endfor
```

Ergebnis

`angle_ll` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`angle_ll` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`angle_lx`

Modul

Basic operators

angle_lx (: : Row1, Column1, Row2, Column2 : Angle)

Berechnung des Winkels zwischen einer Gerade und der x-Achse.

`angle_lx` berechnet den Winkel zwischen einer Gerade und der x-Achse (Horizontalen). Als Eingabe werden die Zeilen- und Spaltenwerte der Gerade (`Row1`,`Column1`, `Row2`,`Column2`) erwartet. Die Berechnung des Winkels geschieht wie folgt: Man interpretiert die Gerade als einen Vektor, wobei `Row1`,`Column1` der Startpunkt und `Row2`,`Column2` der Endpunkt sind. Dreht man nun den Vektor gegen den Uhrzeigersinn auf die x-Achse (der Drehpunkt ist der Schnittpunkt der Geraden mit der Achse) so ist der benötigte Rotationswinkel der gesuchte Wert. Der Winkel ist also abhängig von der Reihenfolge der Koordinaten. Das Ergebnis, d.h. der Winkel in Bogenmaß, wird im Parameter `Angle` übergeben. Der Winkel liegt im Bereich von $-\pi \leq$ `Angle` $\leq \pi$.

Parameter

- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Gerade.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Gerade.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Gerade.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Gerade.
- ▷ **Angle** (output_control) ..... number(-array) ⇝ *real*
  Winkel zwischen der Gerade und der x-Achse.

Beispiel

```
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
```

Ergebnis

angle_lx liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

angle_lx ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

angle_ll

Modul

Basic operators

distance_lr (Region : : Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnung des Abstandes zwischen einer Gerade und einer Region.

distance_lr berechnet den orthogonalen Abstand zwischen einer Gerade und einer Region. Als Eingabe werden die Koordinaten von zwei Punkten (Row1,Column1, Row2,Column2), die die Gerade darstellen, und eine Region erwartet. Das Ergebnis wird dann in den Parametern DistanceMin und DistanceMax übergeben.

Achtung

Aus Laufzeitgründen werden in distance_lr Hohlflächen nicht berücksichtigt. Weiterhin kann beim Schnitt der Geraden mit der Region ein Wert größer als 0.5 geliefert werden.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Eingaberegion.
- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Gerade.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Gerade.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Gerade.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Gerade.
- ▷ **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen der Gerade und der Region
- ▷ **DistanceMax** (output_control) ..... number(-array) ⇝ *real*
  Maximaler Abstand zwischen der Gerade und der Region

Beispiel

```
dev_close_window ()
read_image (Image, 'fabrik')
dev_open_window (0, 0, 512, 512, 'white', WindowHandle)
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
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
```

Ergebnis

distance_lr liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

distance_lr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

distance_pr, distance_sr, diameter_region

Siehe auch

hamming_distance, select_region_point, test_region_point, smallest_rectangle2

Modul

Basic operators

distance_pl (: : Row, Column, Row1, Column1, Row2, Column2 : Distance)

Berechnung des Abstandes zwischen einem Punkt und einer Gerade.

distance_pl berechnet den orthogonalen Abstand zwischen einem Punkt (Row,Column) und einer Gerade, gegeben durch zwei beliebige Punkte auf der Gerade. Das Ergebnis wird im Parameter Distance übergeben.

Parameter

- **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des Punktes.
- **Column** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert Punktes.
- **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Gerade.
- **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Gerade.
- **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Gerade.
- **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Gerade.
- **Distance** (output_control) ..... number(-array) ⇝ *real*
  Abstand zwischen den Punkten

Beispiel

```
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
```

Ergebnis

`distance_pl` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`distance_pl` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`distance_ps`

Siehe auch

`distance_pp`, `distance_pr`

Modul

Basic operators

distance_pp (: : Row1, Column1, Row2, Column2 : Distance)

Berechnung des Abstandes zwischen zwei Punkten.

`distance_pp` berechnet den Abstand zwischen Paaren von Punkten. Der Abstand berechnet sich wie folgt:

$$\texttt{Distance} = \sqrt{((\texttt{Row1} - \texttt{Row2})^2 + (\texttt{Column1} - \texttt{Column2})^2)}$$

Das Ergebnis wird im Parameter `Distance` übergeben.

Parameter

- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes.
- ▷ **Distance** (output_control) ..... number(-array) ⇝ *real*
  Abstand zwischen den Punkten

Beispiel

```
dev_close_window ()
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)
RowPoint := 80
ColPoint := 250
NumberTuple := |Rows|
dev_set_color ('red')
set_draw (WindowHandle, 'margin')
disp_circle (WindowHandle, RowPoint, ColPoint, 10)
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_pp (RowPoint, ColPoint, Rows[i], Columns[i], Distance)
endfor
```

Ergebnis

`distance_pp` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`distance_pp` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Alternativen

`distance_ps`

Siehe auch

`distance_pl`, `distance_pr`

Modul

Basic operators

distance_pr (Region : : Row, Column : DistanceMin, DistanceMax)

Berechnung des Abstandes zwischen einem Punkt und einer Region.

`distance_pr` berechnet den Abstandes zwischen einem Punkt und einer Region. Als Eingabe werden die Spalten- und Zeilenwerte des Punktes (`Row`,`Column`) und eine Region erwartet. Falls der Punkt innerhalb der Region liegt, ist der minimale Abstand Null. Das Ergebnis wird in den Parametern `DistanceMin` und `DistanceMax` übergeben.

Parameter

- ▷ **`Region`** (input_object) ..... region ⇝ *Hobject*
  Eingaberegion.
- ▷ **`Row`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des Punktes.
- ▷ **`Column`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des Punktes.
- ▷ **`DistanceMin`** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen dem Punkt und der Region
- ▷ **`DistanceMax`** (output_control) ..... number(-array) ⇝ *real*
  Maximaler Abstand zwischen dem Punkt und der Region

Beispiel

```
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
  Column2 := 255 + cos(rad(i)) * 200
  disp_line (WindowHandle, Row1, Column1, Row2, Column2)
  distance_pr (SelectedRegions, Row2, Column2,
               DistanceMin, DistanceMax)
endfor
```

Ergebnis

distance_pr liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

distance_pr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

distance_lr, distance_sr, diameter_region

Siehe auch

hamming_distance, select_region_point, test_region_point, smallest_rectangle2

Modul

Basic operators

distance_ps (: : Row, Column, Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnung der Abständen zwischen einem Punkt und einer Strecke.

distance_ps berechnet den minimalen und den maximalen Abstand zwischen einem Punkt (Row,Column) und einer Strecke. Die Strecke wird durch ihre Anfangs- (Row1,Column1) und Endkoordinate (Row2,Column2) beschrieben. DistanceMax ist der maximale Abstand zu den beiden Endpunkten der Strecke. DistanceMin liefert den Abstand senkrecht zur Strecke (analog zu distance_pl), falls der Punkt „zwischen“ den Endpunkten liegt. Ansonsten wird der minimale Anstand zu einem der Endpunkte verwendet.

Parameter

- ▷ **Row** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes.
- ▷ **Column** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes.
- ▷ **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Strecke.
- ▷ **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Strecke.
- ▷ **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Strecke.
- ▷ **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Strecke.
- ▷ **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen Punkt und Strecke
- ▷ **DistanceMax** (output_control) ..... number(-array) ⇝ *real*
  Maximaler Abstand zwischen Punkt und Strecke

Beispiel

```
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 := 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
```

Ergebnis

`distance_ps` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`distance_ps` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`distance_pl`

Siehe auch

`distance_pp`, `distance_pr`

Modul

Basic operators

distance_rr_min (Regions1, Regions2 : : : MinDistance, Row1, Column1, Row2, Column2)

Minimaler Abstand zwischen den Konturpunkten von je zwei Regionen.

`distance_rr_min` berechnet den minimalen Abstand von Paaren von Regionen. Falls mehrere Regionen in `Regions1` und `Regions2` übergeben werden, wird der Abstand zwischen den Konturpunkten der jeweils i-ten Elementen berechnet. Dieser bildet dann den i-ten Eintrag im Ausgabeparameter `MinDistance`. Die Berechnung erfolgt durch den Vergleich aller Konturpunkte. Es wird der euklidische Abstand berechnet. Die Parameter (`Row1`,`Column1`) bzw. (`Row2`, `Column2`) geben die Position auf der Kontur von `Regions1` bzw. `Regions2` an, zwischen denen der Abstand minimal ist.

Achtung

Jede Region muß aus genau einer Zusammenhangskomponente bestehen. Beide Eingabeparameter müssen die gleiche Anzahl von Regionen enthalten. Die Regionen dürfen nicht leer sein. Falls sich die Regionen überlappen, wird der Abstand mit 0.0 angegeben. In diesem Fall sind die Positionen nicht zuverlässig.

Parameter

- ▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.
- ▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu untersuchende Regionen.

▷ **MinDistance** (output_control) ..... real(-array) ⇝ *real*
Minimaler Abstand zwischen Konturen der Regionen.
Zusicherung : `0` ≤ `MinDistance`

▷ **Row1** (output_control) ..... point.y(-array) ⇝ *integer*
Zeilenindex auf Kontur in `Regions1`.

▷ **Column1** (output_control) ..... point.x(-array) ⇝ *integer*
Spaltenindex auf Kontur in `Regions1`.

▷ **Row2** (output_control) ..... point.y(-array) ⇝ *integer*
Zeilenindex auf Kontur in `Regions2`.

▷ **Column2** (output_control) ..... point.x(-array) ⇝ *integer*
Spaltenindex auf Kontur in `Regions2`.

Komplexität

Seien $N1$,$N2$ die Längen der Konturen, dann beträgt die Laufzeitkomplexität $O(N1 * N2)$.

Ergebnis

`distance_rr_min` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`distance_rr_min` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`threshold`, `regiongrowing`, `connection`

Alternativen

`distance_rr_min_dil`, `dilation1`, `intersection`

Modul

Region processing

distance_rr_min_dil (Regions1, Regions2 : : : MinDistance)

Minimaler Abstand zwischen je zwei Regionen mit Hilfe der Dilatation.

`distance_rr_min_dil` berechnet den minimalen Abstand von Paaren von Regionen. Falls mehrere Regionen in `Regions1` und `Regions2` übergeben werden, wird der Abstand zwischen den jeweils i-ten Elementen berechnet. Dieser bildet dann den i-ten Eintrag im Ausgabeparameter `MinDistance`. Die Berechnung erfolgt mit Hilfe der Dilatation mit dem Golay-Element ’h’. Das Ergebnis ist:

$$AnzahlIterationen * 2 - 1$$

.

Die Maske ’h’ bewirkt, daß gerade die Maximums-Metrik berechnet wird..

Achtung

Beide Parameter müßen die gleiche Anzahl von Regionen enthalten. Die Regionen dürfen nicht leer sein.

Parameter

▷ **Regions1** (input_object) ..... region(-array) ⇝ *Hobject*
Zu untersuchende Regionen.

▷ **Regions2** (input_object) ..... region(-array) ⇝ *Hobject*
Zu untersuchende Regionen.

▷ **MinDistance** (output_control) ..... integer(-array) ⇝ *integer*
Minimale Abstände der Regionen.
Zusicherung : `-1` ≤ `MinDistance`

Ergebnis

`distance_rr_min_dil` liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`distance_rr_min_dil` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, regiongrowing, connection

Alternativen

distance_rr_min, dilation1, intersection

Modul

Region processing

distance_sl (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : DistanceMin, DistanceMax)

Berechnung der Abständen zwischen einer Strecke und einer Gerade.

distance_sl berechnet den minimalen und den maximalen orthogonalen Abstand zwischen einer Strecke und einer Gerade. Als Eingabe werden die Spalten- und Zeilewerte der Strecke (RowA1,ColumnA1,RowA2,ColumnA2) und der Gerade (RowB1,ColumnB1,RowB2,ColumnB2) erwartet. Das Ergebnis, d.h. minimaler und maximaler Abstand, wird dann im Parameter DistanceMin und DistanceMax übergeben. Falls sich die beiden Strecken schneiden, dann ist DistanceMin gleich Null.

Parameter

- ▷ **RowA1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Strecke.
- ▷ **ColumnA1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Strecke.
- ▷ **RowA2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Strecke.
- ▷ **ColumnA2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Strecke.
- ▷ **RowB1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Gerade.
- ▷ **ColumnB1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert es ersten Punktes der Gerade.
- ▷ **RowB2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Gerade.
- ▷ **ColumnB2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Gerade.
- ▷ **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen Strecke und Gerade
- ▷ **DistanceMax** (output_control) ..... number(-array) ⇝ *real*
  Maximaler Abstand zwischen Strecke und Gerade

Beispiel

```
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')
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
```

Ergebnis

`distance_sl` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`distance_sl` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`distance_pl`

Siehe auch

`distance_ps`, `distance_pp`

Modul

Basic operators

distance_sr (Region : : Row1, Column1, Row2, Column2 : DistanceMin, DistanceMax)

Berechnung des Abstandes zwischen einer Strecke und einer Region.

`distance_sr` berechnet den Abstandes zwischen einer Strecke und einer Region. Als Eingabe werden die Anfangs- und Endpunkte der Strecke (`Row1`,`Column1`,`Row2`,`Column2`) und eine Region erwartet. Das Ergebnis wird in den Parametern `DistanceMin` und `DistanceMax` übergeben.

Achtung

Aus Laufzeitgründen werden in `distance_sr` Hohlflächen nicht berücksichtigt. Weiterhin kann beim Schnitt der Geraden mit der Region ein Wert größer als 0.5 geliefert werden.

Parameter

- **Region** (input_object) ..... region ⇝ *Hobject*
  Eingaberegion.
- **Row1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Strecke.
- **Column1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Strecke.
- **Row2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Strecke.
- **Column2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Strecke.
- **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen der Strecke und der Region
- **DistanceMax** (output_control) ..... number(-array) ⇝ *real*
  Maximaler Abstand zwischen der Strecke und der Region

Beispiel

```
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
```

Ergebnis

distance_sr liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

distance_sr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

distance_lr, distance_pr, diameter_region

Siehe auch

hamming_distance, select_region_point, test_region_point, smallest_rectangle2

Modul

Basic operators

distance_ss (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : DistanceMin, DistanceMax)

Berechnung der Abständen zwischen zwei Strecken.

distance_ss berechnet den minimalen und den maximalen Abstand zwischen den Anfangs- und Endkoordinaten von zwei Strecken. Als Eingabe werden die Spalten und Zeilen (RowA1,ColumnA1,RowA2,ColumnA2) der ersten Strecke und der zweiten Strecke (RowB1,ColumnB1,RowB2,ColumnB2) erwartet. Das Ergebnis, d.h. minimaler und maximaler Abstand, wird dann im Parameter DistanceMin und DistanceMax übergeben. Falls sich die beiden Strecken schneiden, dann ist DistanceMin gleich Null.

Parameter

- ▷ **RowA1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der ersten Strecke.
- ▷ **ColumnA1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der ersten Strecke.
- ▷ **RowA2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der ersten Strecke.
- ▷ **ColumnA2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der ersten Strecke.
- ▷ **RowB1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der zweiten Strecke.
- ▷ **ColumnB1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der zweiten Strecke.
- ▷ **RowB2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der zweiten Strecke.
- ▷ **ColumnB2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der zweiten Strecke.
- ▷ **DistanceMin** (output_control) ..... number(-array) ⇝ *real*
  Minimaler Abstand zwischen den Strecken
- ▷ **DistanceMax** (output_control) ..... number(-array) ⇝ *real*
  Maximaler Abstand zwischen den Strecken

Beispiel

```
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
```

Ergebnis

`distance_ss` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`distance_ss` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Alternativen

`distance_pp`

Siehe auch

`distance_pl`, `distance_ps`

Modul

Basic operators

get_points_ellipse (: : Angle, Row, Column, Phi, Radius1, Radius2 : RowPoint, ColPoint)

Ellipsenpunkte gemäß Winkel.

`get_points_ellipse` liefert die Punkte (`RowPoint`,`ColPoint`) auf der übergebenen Ellipse, die zu den Winkeln in `Angle`, bezogen auf die Hauptachse der Ellipse, korrespondieren. Die Ellipse selbst wird beschrieben durch ihren Mittelpunkt (`Row`, `Column`), die Orientierung der Hauptachse `Phi` und die Länge der großen `Radius1` bzw. kleinen Halbachse `Radius2`.

Parameter

- ▷ **Angle** (input_control) ..... real(-array) ⇝ *real*
  Winkel der gesuchten Punkte auf der Ellipse [rad].
  Defaultwert : 0
  Restriktion : (Angle ≥ 0) ∧ (Angle ≤ 6.283185307)
- ▷ **Row** (input_control) ..... ellipse.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Column** (input_control) ..... ellipse.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Phi** (input_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der Hauptachse in Bogenmaß.
  Restriktion : (Phi ≥ 0) ∧ (Phi ≤ 6.283185307)
- ▷ **Radius1** (input_control) ..... ellipse.radius1 ⇝ *real*
  Länge der großen Halbachse.
  Restriktion : Radius1 > 0
- ▷ **Radius2** (input_control) ..... ellipse.radius2 ⇝ *real*
  Länge der kleinen Halbachse.
  Restriktion : Radius2 ≥ 0
- ▷ **RowPoint** (output_control) ..... point.row(-array) ⇝ *real*
  Zeilenkoordinaten der Ellipsenpunkte.
- ▷ **ColPoint** (output_control) ..... point.column(-array) ⇝ *real*
  Spaltenkoordinate der Ellipsenpunkte.

Beispiel

```
draw_ellipse(WindowHandle,Row,Column,Phi,Radius1,Radius2)
get_points_ellipse([0,3.14],Row,Column,Phi,Radius1,Radius2,RowPoint,ColPoint)
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_points_ellipse` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_points_ellipse` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`fit_ellipse_contour_xld`, `draw_ellipse`, `gen_ellipse_contour_xld`

Siehe auch

`gen_ellipse_contour_xld`

Modul

Basic operators

intersection_ll (: : RowA1, ColumnA1, RowA2, ColumnA2, RowB1, ColumnB1, RowB2, ColumnB2 : Row, Column, IsParallel)

Berechnung des Schnittpunktes zwischen zwei Geraden.

`intersection_ll` berechnet die Zeilen- und Spaltenwerte des Schnittpunktes zwischen zwei Geraden. Als Eingabe werden die Spalten- und Zeilewerte der Geraden (`RowA1`,`ColumnA1`, `RowA2`,`ColumnA2`) und (`RowB1`,`ColumnB1`, `RowB2`,`ColumnB2`) erwartet. Das Ergebnis wird dann im Parameter `Row` und `Column` übergeben. Falls die Geraden parallel zueinander sind, liefert `IsParallel` den Wert 1 zurück, sonst 0. Außerdem sind die Werte von `Row` und `Column` undefiniert.

Achtung

Falls die Geraden parallel zueinander sind, dann sind die Werte von `Row` und `Column` undefiniert.

Parameter

- **RowA1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der ersten Gerade.
- **ColumnA1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der ersten Gerade.
- **RowA2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der ersten Gerade.
- **ColumnA2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der ersten Gerade.
- **RowB1** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der zweiten Gerade.
- **ColumnB1** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert es ersten Punktes der zweiten Gerade.
- **RowB2** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der zweiten Gerade.
- **ColumnB2** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der zweiten Gerade.
- **Row** (output_control) ..... point.y(-array) ⇝ *real*
  Zeilenwert des Schnittpunktes
- **Column** (output_control) ..... point.x(-array) ⇝ *real*
  Spaltenwert des Schnittpunktes
- **IsParallel** (output_control) ..... number(-array) ⇝ *integer*
  Sind die zwei Geraden parallel?

Beispiel

```
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,
                   RowLine2, ColLine2, Row, Column, IsParallel)
  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
```

Ergebnis

`intersection_ll` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`intersection_ll` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Basic operators

`projection_pl` (: : `Row, Column, Row1, Column1, Row2, Column2` : `RowProj, ColProj`)

Berechnung der Projektion von einem Punkt auf eine Gerade.

`projection_pl` berechnet die Projektion von einem Punkt (`Row`,`Column`) auf eine Gerade. Die Gerade wird durch seine Anfangs- (`Row1`,`Column1`) und Endkoordinaten (`Row2`,`Column2`) beschrieben. `RowProj` ist der Zeilenwert vom Projektionspunkt. `ColProj` ist der Spaltenwert vom Projektionspunkt.

Parameter

- **`Row`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des Punktes.
- **`Column`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des Punktes.
- **`Row1`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des ersten Punktes der Gerade.
- **`Column1`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des ersten Punktes der Gerade.
- **`Row2`** (input_control) ..... point.y(-array) ⇝ *real* / integer
  Zeilenwert des zweiten Punktes der Gerade.
- **`Column2`** (input_control) ..... point.x(-array) ⇝ *real* / integer
  Spaltenwert des zweiten Punktes der Gerade.
- **`RowProj`** (output_control) ..... number(-array) ⇝ *real*
  Zeilenwert des Projektionspunkt
- **`ColProj`** (output_control) ..... number(-array) ⇝ *real*
  Spaltenwert des Projektionspunkt

Beispiel

```
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
```

Ergebnis

`projection_pl` liefert den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`projection_pl` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Basic operators

12.6 Hintergrundschätzer

`close_all_bg_esti` (: : :)

Löscht alle Hintergrundschätzer-Datensätze.

`close_all_bg_esti` gibt den Speicherplatz aller Hintergrundschätzer-Datensätze frei.

Achtung

Da alle Hintergrundschätzer geschlossen werden, sind alle vorhandenen Handle fortan ungültig.

Ergebnis

Gelingt es, die Hintergrundschätzer zu schließen, liefert `close_all_bg_esti` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_all_bg_esti` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Alternativen

`close_bg_esti`

Siehe auch

`create_bg_esti`

Modul

Background estimation

`close_bg_esti` (: : BgEstiHandle :)

Löscht Hintergrundschätzer Datensatz.

close_bg_esti gibt den Speicherplatz des Hintergrundschätzer Datensatzes frei.

Parameter

▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *integer*
ID des BgEsti-Datensatzes.

Beispiel

```
/* 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. */
/* - end of background estimation - */
/* close the dataset: */
close_bg_est(BgEstiHandle).
```

Ergebnis

close_bg_esti liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind.

Parallelisierungsinformation

close_bg_esti ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

run_bg_esti

Siehe auch

create_bg_esti

Modul

Background estimation

create_bg_esti (InitializeImage : : Syspar1, Syspar2, GainMode, Gain1, Gain2, AdaptMode, MinDiff, StatNum, ConfidenceC, TimeC : BgEstiHandle)

Erzeugt Datensatz für Hintergrundschätzer und belegt diesen mit Parametern.

create_bg_esti legt einen neuen Datensatz für die Hintergrundschätzung an und belegt diesen mit den übergebenen Parametern. Zu diesem Datensatz gehört auch das geschätzte Hintergrundbild. Der erzeugte Datensatz wird automatisch der aktuelle Datensatz.

Mit InitializeImage wird eine Anfangsvorhersage für das Hintergrundbild festgelegt. Es ist sinnvoll eine Aufnahme der Szene ohne bewegte Objekte zu übergeben, so daß eine geringe Adaption für den Vordergrund gewählt werden kann. Falls keine Aufnahme der leeren Szene zur Verfügung steht, kann z.B. ein Graubild übergeben werden. Es muß dann allerdings eine schnellere Adaption für den Vordergrund gewählt werden, da alle Bildbereiche zunächst als Vordergrund erkannt werden. Das Initialisierungsbild muß ein Bild vom Typ **byte** oder

'real' sein. Ferner werden nur einkanalige Bilder verarbeitet, daher muß für jeden Kanal ein eigener Datensatz aufgebaut werden. Mit dem `InitializeImage` wird die Größe und der aktuelle Ausschnitt (Region) für alle Hintergrundschätzungen (`run_bg_esti`), die mit diesem Datensatz durch geführt werden, festgelegt.

`Syspar1` und `Syspar2` sind die Parameter der Systemmatrix. Die Systemmatrix beschreibt das System der Grauwertveränderungen entsprechend der Kalmanfilter-Theorie. Im Hintergrundschätzer ist für jedes Pixel ein solches System realisiert.

Mit `GainMode` wird bestimmt, ob ein festes Kalman-Gain für die Schätzung verwendet werden soll oder ob das Gain in Abhängigkeit der Grauwertdifferenz zwischen Vorhersage und Meßwert verwendet werden soll. Wird `GainMode` = **'fixed'** gewählt, so ist `Gain1` der Kalmangain, der bei der Schätzung für Pixel, die Vordergrund detektiert wurden, verwendet wird und `Gain2` ist der Kalmangain, der bei der Schätzung für Pixel, die als Hintergrund detektiert wurden, verwendet wird. `Gain1` ist sinnvollerweise kleiner als `Gain2`, denn die vollständige Adaption des Vordergrundes sollte langsamer als die Adaption des Hintergrundes erreicht sein. Feste Gains sind sinnvoll, wenn mit dem Hintergrundschätzer Regionen mit Bewegung gefunden werden sollen und somit `Gain1` sehr klein oder **0.0** gewählt wurde. `Gain1` und `Gain2` sollten hier zwischen **0.0** und **1.0** gewählt werden.

Wird `GainMode` = **'frame'** gewählt, so wird für die Vordergrundschätzung, sowie für die Hintergrundschätzung eine Tabelle mit Kalmangains für alle 256 möglichen Grauwertdifferenzen bestimmt. Mit `Gain1` und `Gain2` wird dann die Anzahl der Frames angegeben, die benötigt werden, um eine Abweichung zwischen Vorhersagewert und Meßwert zu adaptieren. Es ist einleuchtend, daß bei gleicher Adaptionszeit (Anzahl der Frames) bei einer großen Differenz ein größeres Kalmangain benötigt wird als bei einer kleinen Differenz. `Gain1` ist hier sinnvollerweise größer als `Gain2`, denn die Adaptionszeit für die Vordergrundadaption sollte länger sein, als die der Hintergrundadaption. Unterschiedliche Gains für verschiedene Grauwertdifferenzen sind sinnvoll, wenn der Hintergrundschätzer zur Aufnahme der 'leeren Szene' verwendet wird, d.h. wenn sich im Aufnahmebereich ständig Objekte bewegen und ein Bild des reinen Hintergrundes erstellt werden muß. Hierzu darf dann die Adaptionszeit für den Vordergrund (`Gain1`) nicht zu groß gewählt werden. `Gain1` und `Gain2` sollten hier größer als **1.0** gewählt werden.

Mit `AdaptMode` wird festgelegt, ob der Schwellenwert, der auf die Grauwertdifferenz zwischen Meßwert und Schätzwert angewendet wird, fest ist oder entsprechend der Streuung der Grauwerte der als Hintergrund detektieten Pixel verändert wird.

Wenn `AdaptMode` = **'off'** gewählt wird, wird mit `MinDiff` die absolute Schwelle angegeben.Die Parameter `StatNum`, `ConfidenceC` und `TimeC` sind hier ohne Bedeutung.

Wenn `AdaptMode` = **'on'** gewählt wird, wird mit `MinDiff` eine Basisschwelle angegeben, auf die, entsprechend der statistischen Auswertung der Grauwertverteilung über die Zeit in jedem Bildpunkt, ein Offset hinzu addiert wird. Mit `StatNum` gibt man die Anzahl der statistischen Datensätze an, d.h. wie viele der vergangenen Frames zur Berechnung der Varianz verwendet werden sollen (FIR-Filter). Mit `ConfidenceC` gibt man die Konfidenzkonstante an, die zur Bestimmung des Konfidenzintervalles dient. Mit dem Konfidenzintervall werden die Werte für die Hintergrundstatistik nachgeführt, wenn der Bereich durch ein Objekt im Vordergrund verdeckt ist, d.h. das Pixel als Vordergrund detektiert ist. Entsprechend der Students t-Verteilung ist die Konfidenzkonstante **4.30** (**3.25**, **2.82**, **2.26**) für ein **99,8%** (**99,0%**, **98,0%**, **95,0%**) Konfidenzintervall. Mit `TimeC` gibt man eine Zeitkonstante für die e-Funktion an, mit der die Schwelle im Fall der Vordergrunddetektion angehoben werden soll. Dies bedeutet, daß die Schwelle in den Bereichen, in denen Bewegung im Vordergrund detektiert wurde, angehoben wird, damit bei anschließendem wieder freigegebenem Blick auf den Hintergrund die Toleranz (Schwelle) für Beleuchtungsänderungen gestiegen ist. Dies ist nötig, da man in der Zeit, in der der Hintergrund verdeckt ist keine Aussage über die Beleuchtungsänderungen im Hintergrund, machen kann und somit das geschätzte Hintergrundbild auch nicht adaptieren kann.

Achtung

Wenn für `GainMode` = **'frame'** gewählt wurde, kann bei der Wahl von großen Werten für `Gain1` oder `Gain2` die Laufzeit sehr groß werden, da die Werte für die Gaintabelle mit einem eindimensionalen Optimierer (binäre Suche) bestimmt werden.

Parameter

▷ **`InitializeImage`** (input_object) . . . . . image ⇝ *Hobject* : byte / real
Initialisierungsbild.

▷ **`Syspar1`** (input_control) . . . . . real ⇝ *real*
1. Parameter der Systemmatrix.
Defaultwert : 0.7
Wertevorschläge : `Syspar1` ∈ {0.65, 0.7, 0.75}
Typischer Wertebereich : 0.05 ≤ `Syspar1` ≤ 1.0
Empfohlene Schrittweite : 0.05

▷ **`Syspar2`** (input_control) . . . . . real ⇝ *real*
2. Parameter der Systemmatrix.
Defaultwert : 0.7
Wertevorschläge : `Syspar2` ∈ {0.65, 0.7, 0.75}
Typischer Wertebereich : 0.05 ≤ `Syspar2` ≤ 1.0
Empfohlene Schrittweite : 0.05

▷ **`GainMode`** (input_control) . . . . . string ⇝ *string*
Art der Gains.
Defaultwert : 'fixed'
Werteliste : `GainMode` ∈ {'fixed', 'frame'}

▷ **`Gain1`** (input_control) . . . . . real ⇝ *real*
Kalmangain / Adaptionszeit für Vordergrund.
Defaultwert : 0.002
Wertevorschläge : `Gain1` ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
Restriktion : `0.0` ≤ `Gain1`

▷ **`Gain2`** (input_control) . . . . . real ⇝ *real*
Kalmangain / Adaptionszeit für Hintergrund.
Defaultwert : 0.02
Wertevorschläge : `Gain2` ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
Restriktion : `0.0` ≤ `Gain2`

▷ **`AdaptMode`** (input_control) . . . . . string ⇝ *string*
Adaption der Schwelle.
Defaultwert : 'on'
Werteliste : `AdaptMode` ∈ {'on', 'off'}

▷ **`MinDiff`** (input_control) . . . . . real ⇝ *real*
Schwelle, für Vordergrund / Hintergrund.
Defaultwert : 7.0
Wertevorschläge : `MinDiff` ∈ {3.0, 5.0, 7.0, 9.0, 11.0}
Empfohlene Schrittweite : 0.2

▷ **`StatNum`** (input_control) . . . . . integer ⇝ *integer*
Anzahl statistischer Datensätze.
Defaultwert : 10
Wertevorschläge : `StatNum` ∈ {5, 10, 20, 30}
Typischer Wertebereich : 1 ≤ `StatNum`
Empfohlene Schrittweite : 5

▷ **`ConfidenceC`** (input_control) . . . . . real ⇝ *real*
Konfidenzkonstante.
Defaultwert : 3.25
Wertevorschläge : `ConfidenceC` ∈ {4.30, 3.25, 2.82, 2.62}
Empfohlene Schrittweite : 0.01
Restriktion : `0.0` < `ConfidenceC`

▷ **`TimeC`** (input_control) . . . . . real ⇝ *real*
Abklingkonstante.
Defaultwert : 15.0
Wertevorschläge : `TimeC` ∈ {10.0, 15.0, 20.0}
Empfohlene Schrittweite : 5.0
Restriktion : `0.0` < `TimeC`

▷ **BgEstiHandle** (output_control) ..... bg_estimation ⇝ *integer*
ID des BgEsti-Datensatzes.

Beispiel

```
/* 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).
```

Ergebnis

create_bg_esti liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind.

Parallelisierungsinformation

create_bg_esti ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

run_bg_esti

Siehe auch

set_bg_esti_params, close_bg_esti

Modul

Background estimation

get_bg_esti_params (: : BgEstiHandle : Syspar1, Syspar2, GainMode, Gain1, Gain2, AdaptMode, MinDiff, StatNum, ConfidenceC, TimeC)

Gibt die Parameter des Datensatzes aus.

Mit get_bg_esti_params werden die Parameter des Datensatz ausgegeben. Die zurückgegebenen Parameter sind identisch mit denen für create_bg_esti und set_bg_esti_params (Erklärung siehe dort).

Parameter

▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *integer*
ID des BgEsti-Datensatzes.

▷ **Syspar1** (output_control) ..... real ⇝ *real*
1. Parameter der Systemmatrix.

▷ **Syspar2** (output_control) ..... real ⇝ *real*
2. Parameter der Systemmatrix.

▷ **GainMode** (output_control) ..... string ⇝ *string*
Art der Gains.

▷ **Gain1** (output_control) ..... real ⇝ *real*
Kalmangain / Adaptionszeit für Vordergrund.

▷ **Gain2** (output_control) ..... real ⇝ *real*
Kalmangain / Adaptionszeit für Hintergrund.

▷ **AdaptMode** (output_control) ..... string ⇝ *string*
Adaption der Schwelle.

▷ **MinDiff** (output_control) ..... real ⇝ *real*
Schwelle, für Vordergrund / Hintergrund.

▷ **StatNum** (output_control) ..... integer ⇝ *integer*
Anzahl statistischer Datensätze.

▷ **ConfidenceC** (output_control) ..... real ⇝ *real*
Konfidenzkonstante.

▷ **TimeC** (output_control) . . . . . real ⇝ *real*
Abklingkonstante.

Beispiel

```
/* 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 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. */
```

Ergebnis

`get_bg_esti_params` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind.

Parallelisierungsinformation

`get_bg_esti_params` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_bg_esti`

Mögliche Nachfolgerfunktionen

`run_bg_esti`

Siehe auch

`set_bg_esti_params`

Modul

Background estimation

give_bg_esti (: BackgroundImage : BgEstiHandle :)

Liefert geschätztes Hintergrundbild.

`give_bg_esti` liefert das im aktuellen BgEsti-Datensatz gespeicherte Hintergrundbild. Das Hintergrundbildes ist vom gleichen Typ und gleicher Größe, wie das bei `create_bg_esti` angegebene Initialisierungsbild.

Parameter

- ▷ **BackgroundImage** (output_object) ..... image ⇝ *Hobject* : byte / real
  geschätztes Hintergrundbild aus aktuellem BgEsti-Datensatz.
- ▷ **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *integer*
  ID des BgEsti-Datensatzes.

Beispiel

```
/* 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).
```

Ergebnis

give_bg_esti returns 2 (H_MSG_TRUE) if all parameters are correct.

Parallelisierungsinformation

give_bg_esti ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

run_bg_esti

Mögliche Nachfolgerfunktionen

run_bg_esti, create_bg_esti, update_bg_esti

Siehe auch

run_bg_esti, update_bg_esti, create_bg_esti

Modul

Background estimation

run_bg_esti (PresentImage : ForegroundRegion : BgEstiHandle :)

Schätzt den Hintergrund und liefert Vordergrund-Region.

run_bg_esti adaptiert mit einem Kalman-Filter das im angegebenen Datensatz liegende Hintergrundbild und liefert eine Region des Vordergrundes (Bereich bewegter Objekte) zurück.

Es wird für jedes Pixel anhand der Daten im aktuellen Datensatz eine Vorhersage des Grauwertes bestimmt und diese dann mit dem Wert im aktuellen Bild (PresentImage) verglichen. Mit dem Schwellenwert (fest oder adaptiv, siehe create_bg_esti) werden die Pixel in Vordergrund oder Hintergrund klassifiziert.

Es werden nur einkanalige Bilder verarbeitet, daher muß der Hintergrund für jeden Kanal einzeln adaptiert werden. Hierzu müssen mehrere Datensätze mit create_bg_esti aufgebaut werden.

Empfehlenswert ist es, den Hintergrundschätzer nur auf der Hälfte oder sogar nur auf einem Viertel der Originalbildgröße anzuwenden. Hierzu muß das Originalbild zunächst mit zoom_image_factor verkleinert werden (dies gilt auch für das Initialisierungsbild bei create_bg_esti). Der Vorteil ist auf der einen Seite die reduzierte Laufzeit und auf der anderen Seite ist das Bild durch die Verkleinerung Tiefpaß-gefiltert. Dadurch sind

hochfrequente Störungen, wie Kamerarauschen herausgefiltert und stören somit auch nicht bei der Adaption des Hintergrundbildes. Es ist somit möglich die Schwelle (siehe `create_bg_esti`) geringer zu wählen. Die von `run_bg_esti` zurückgelieferte `ForegroundRegion` muß dann natürlich für die Auswertung (Bereiche mit Bewegung) mit `zoom_region` wieder um den gleichen Faktor vergrößert werden.

Achtung

Das übergebene Bild (`PresentImage`) muß von gleichem Typ und gleicher Größe sein, wie das im aktuellen Datensatz abgelegte Hintergrundbild (festgelegt mit `create_bg_esti`).

Parameter

- ▷ **`PresentImage`** (input_object) ..... image ⇝ *Hobject* : byte / real
  Aktuelles Bild.
- ▷ **`ForegroundRegion`** (output_object) ..... region ⇝ *Hobject*
  Region der als Vordergrund detektierten Bereiche.
- ▷ **`BgEstiHandle`** (input_control) ..... bg_estimation ⇝ *integer*
  ID des BgEsti-Datensatzes.

Beispiel

```
/* 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. */
```

Ergebnis

`run_bg_esti` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind.

Parallelisierungsinformation

`run_bg_esti` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_bg_esti`, `update_bg_esti`

Mögliche Nachfolgerfunktionen

`run_bg_esti`, `give_bg_esti`, `update_bg_esti`

Siehe auch

`set_bg_esti_params`, `create_bg_esti`, `update_bg_esti`, `give_bg_esti`

Modul

Background estimation

`set_bg_esti_params` `( : : BgEstiHandle, Syspar1, Syspar2, GainMode, Gain1, Gain2, AdaptMode, MinDiff, StatNum, ConfidenceC, TimeC : )`

Ändert im Datensatz die Parameter.

Mit `set_bg_esti_params` werden die Parameter im Datensatz verändert. Die übergebenen Parameter sind identisch mit denen für `create_bg_esti` (Erklärung siehe dort).

Das Bildformat kann jedoch nicht geändert werden, hierfür muß ein neuer Datensatz angelegt werden, bei dem das Initialisierungsbild das gewünschte Format hat.

Soll das Hintergrundbild komplett ausgetauscht werden, so muß `update_bg_esti` verwendet werden. Es muß dann sowohl für das Eingabebild und auch für die Up-Date-Region das aktuelle Bild angegeben werden.

Achtung

Wenn `GainMode` = **'frame'** gewählt wurde, kann bei der Wahl von großen Werten für `Gain1` oder `Gain2` die Laufzeit sehr groß werden, da die Werte für die Gaintabelle mit einem eindimensionalen Optimierer (binäre Suche) bestimmt werden.

Parameter

- ▷ **`BgEstiHandle`** (input_control) . . . . . bg_estimation ⇝ *integer*
  ID des BgEsti-Datensatzes.
- ▷ **`Syspar1`** (input_control) . . . . . real ⇝ *real*
  1. Parameter der Systemmatrix.
  Defaultwert : 0.7
  Wertevorschläge : `Syspar1` ∈ {0.65, 0.7, 0.75}
  Typischer Wertebereich : 0.05 ≤ `Syspar1` ≤ 1.0
  Empfohlene Schrittweite : 0.05
- ▷ **`Syspar2`** (input_control) . . . . . real ⇝ *real*
  2. Parameter der Systemmatrix.
  Defaultwert : 0.7
  Wertevorschläge : `Syspar2` ∈ {0.65, 0.7, 0.75}
  Typischer Wertebereich : 0.05 ≤ `Syspar2` ≤ 1.0
  Empfohlene Schrittweite : 0.05
- ▷ **`GainMode`** (input_control) . . . . . string ⇝ *string*
  Art der Gains.
  Defaultwert : 'fixed'
  Werteliste : `GainMode` ∈ {'fixed', 'frame'}
- ▷ **`Gain1`** (input_control) . . . . . real ⇝ *real*
  Kalmangain / Adaptionszeit für Vordergrund.
  Defaultwert : 0.002
  Wertevorschläge : `Gain1` ∈ {10.0, 20.0, 50.0, 0.1, 0.05, 0.01, 0.005, 0.001}
  Restriktion : `0.0` ≤ `Gain1`
- ▷ **`Gain2`** (input_control) . . . . . real ⇝ *real*
  Kalmangain / Adaptionszeit für Hintergrund.
  Defaultwert : 0.02
  Wertevorschläge : `Gain2` ∈ {2.0, 4.0, 8.0, 0.5, 0.1, 0.05, 0.01}
  Restriktion : `0.0` ≤ `Gain2`
- ▷ **`AdaptMode`** (input_control) . . . . . string ⇝ *string*
  Adaption der Schwelle.
  Defaultwert : 'on'
  Werteliste : `AdaptMode` ∈ {'on', 'off'}
- ▷ **`MinDiff`** (input_control) . . . . . real ⇝ *real*
  Schwelle, für Vordergrund / Hintergrund.
  Defaultwert : 7.0
  Wertevorschläge : `MinDiff` ∈ {3.0, 5.0, 7.0, 9.0, 11.0}
  Empfohlene Schrittweite : 0.2
- ▷ **`StatNum`** (input_control) . . . . . integer ⇝ *integer*
  Anzahl statistischer Datensätze.
  Defaultwert : 10
  Wertevorschläge : `StatNum` ∈ {5, 10, 20, 30}
  Typischer Wertebereich : 1 ≤ `StatNum`
  Empfohlene Schrittweite : 5

▷ **ConfidenceC** (input_control) . . . . . real ⇝ *real*
Konfidenzkonstante.
Defaultwert : 3.25
Wertevorschläge : ConfidenceC ∈ {4.30, 3.25, 2.82, 2.62}
Empfohlene Schrittweite : 0.01
Restriktion : 0.0 < ConfidenceC

▷ **TimeC** (input_control) . . . . . real ⇝ *real*
Abklingkonstante.
Defaultwert : 15.0
Wertevorschläge : TimeC ∈ {10.0, 15.0, 20.0}
Empfohlene Schrittweite : 5.0
Restriktion : 0.0 < TimeC

Beispiel

```
/* 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. */
```

Ergebnis

set_bg_esti_params liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind.

Parallelisierungsinformation

set_bg_esti_params ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_bg_esti

Mögliche Nachfolgerfunktionen

run_bg_esti

Siehe auch

update_bg_esti

Modul

Background estimation

update_bg_esti (PresentImage, UpDateRegion : : BgEstiHandle :)

Verändert geschätztes Hintergrundbild.

update_bg_esti überschreibt das im BgEsti-Datensatz gespeicherte Hintergrundbild in den durch UpDateRegion festgelegten Bereichen durch die Grauwerte aus PresentImage. Dies kann zur „harten“ Adaption verwendet werden, d.h. Bildbereiche, in denen sich der Hintergrund stark verändert hat, können hiermit schnell adaptiert werden.

Achtung

Das übergebene Bild (PresentImage) muß vom gleichen Typ und gleicher Größe sein, wie das im aktuellen Datensatz abgelegte Hintergrundbild (festgelegt mit create_bg_esti).

Parameter

- **PresentImage** (input_object) ..... image ⇝ *Hobject* : byte / real
  Aktuelles Bild.
- **UpDateRegion** (input_object) ..... region ⇝ *Hobject*
  Region, die zu verändernde Bereiche beschreibt.
- **BgEstiHandle** (input_control) ..... bg_estimation ⇝ *integer*
  ID des BgEsti-Datensatzes.

Beispiel

```
/* 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. */
```

Ergebnis

update_bg_esti liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind.

Parallelisierungsinformation

update_bg_esti ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

run_bg_esti

Mögliche Nachfolgerfunktionen

run_bg_esti

Siehe auch

run_bg_esti, give_bg_esti

Modul

Background estimation

12.7 Hough

hough_circle_trans (Region : HoughImage : Radius :)

Liefert die Hough-Transformation für Kreise mit dem angegebenen Radius.

hough_circle_trans berechnet für die in Region übergebenen Regionen die Hough-Transformation für Kreise mit einem bestimmten Radius. Dabei werden im Parameterraum (bzw. Hough- oder Akkumulatorraum) die Mittelpunkte aller möglichen Kreise für jeden Punkt im Bildraum akkumuliert. Kreise, die durch viele Punkte der Eingabe-Region gestützt werden, erzeugen so im Ergebnisbild (HoughImage) ein Maximum an der den Kreismittelpunkt beschreibenden Stelle. Aus den Koordinaten dieser Maxima ergeben sich durch Subtraktion des Radius die Mittelpunkte der Kreise im Bildraum. Werden mehrere Radien übergeben, sind alle Hough-Bilder um den maximalen Radius verschoben.

Parameter

- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Binäres Kantenbild, in dem Kreise gefunden werden sollen.
- ▷ **HoughImage** (output_object) ..... image(-array) ⇝ *Hobject* : int2
  Hough-Transformierte für Kreise mit dem angegebenen Radius.
  Parameteranzahl : HoughImage = Radius
- ▷ **Radius** (input_control) ..... integer(-array) ⇝ *integer*
  Radius des Kreises, nach dem im Bild gesucht werden soll.
  Defaultwert : 12
  Typischer Wertebereich : 3 ≤ Radius (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1
  Parameteranzahl : (1 ≤ Radius) ≤ 500

Ergebnis

hough_circle_trans liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hough_circle_trans ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Region processing

hough_circles (RegionIn : RegionOut : Radius, Percent, Mode :)

Mittelpunkt von Kreisen für einem bestimmten Radius.

hough_circle_trans findet den Mittelpunkt von Kreisen in Regionen mit Hilfe der Hough-Transformation für Kreise mit einem bestimmten Radius.

Parameter

- ▷ **RegionIn** (input_object) ..... region ⇝ *Hobject*
  Binäres Kantenbild, in dem Kreise gefunden werden sollen.
- ▷ **RegionOut** (output_object) ..... region(-array) ⇝ *Hobject*
  Mittelpunkte der Kreise, die mit Percent Prozent im Kantenbild enthalten sind.
  Parameteranzahl : RegionOut = ((Radius · Percent) · Mode)

▷ **Radius** (input_control) . . . . . integer(-array) ⇝ *integer*
Radius des Kreises, nach dem im Bild gesucht werden soll.
Defaultwert : 12
Typischer Wertebereich : 2 ≤ Radius ≤ 500 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 1
Parameteranzahl : (1 ≤ Radius) ≤ 500

▷ **Percent** (input_control) . . . . . integer(-array) ⇝ *integer*
Gibt an, wieviel Prozent des (idealen) Kreises im Kantenbild RegionIn enthalten sein müssen (ca.-Angabe).
Defaultwert : 60
Typischer Wertebereich : 10 ≤ Percent ≤ 100 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 5
Parameteranzahl : (1 ≤ Percent) ≤ 100

▷ **Mode** (input_control) . . . . . integer(-array) ⇝ *integer*
Der Modus definiert die Lage der gesuchten Kreise:
0 - Radius entspricht dem äußerem Rand der gesetzten Pixel.
1 - Radius entspricht den Mittelpunkten der Pixel der Kreislinie.
2 - 0 und 1 zusammen (etwas unschärfer, dadurch aber auch sicherer gegenüber etwas anders gesetzten Kreisen, 50 % höherer Rechenaufwand als 0 und 1).
Werteliste : Mode ∈ {0, 1, 2}
Parameteranzahl : (1 ≤ Mode) ≤ 3

Ergebnis

hough_circles liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hough_circles ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Region processing

hough_line_trans (Region : HoughImage : AngleResolution :)

Liefert die Hough-Transformation für Linien in Regionen.

hough_line_trans berechnet für die in Region übergebenen Regionen die Hough-Transformation für Linien. Dabei werden im Parameterraum (bzw. Hough- oder Akkumulatorraum) Winkel und Länge der Normalenvektoren der Linien eingetragen, d.h. die Parametrisierung der Linien erfolgt in der Hesseschen Normalform.

Das Ergebnis wird in ein neu erzeugtes Int2-Image (HoughImage) eingetragen, bei dem die X-Achse dem Winkel des Normalenvektors zur X-Achse (im Ausgangsbild) und die Y-Achse dem Abstand der Linie vom Ursprung entspricht.

Der Winkel läuft von -90 Grad bis 180 Grad, und wird mit einer Auflösung von 1/AngleResolution eingetragen, das bedeutet, daß ein Pixel in X-Richtung 1/AngleResolution Grad entspricht und daß das HoughImage 270 ∗ AngleResolution + 1 Pixel breit ist. Die Höhe von HoughImage entspricht der Diagonalen des umschließenden Rechtecks der Eingabe-Region.

Die Maxima im Ergebnisbild entsprechen den Parameterwerten der Linien im Ausgangsbild.

Parameter

▷ **Region** (input_object) . . . . . region ⇝ *Hobject*
Binäres Kantenbild, in dem Linien gefunden werden sollen.

▷ **HoughImage** (output_object) . . . . . image ⇝ *Hobject* : int2
Hough-Transformierte für Linien.

▷ **AngleResolution** (input_control) . . . . . integer ⇝ *integer*
Einstellen der Auflösung im Winkelbereich.
Defaultwert : 4
Werteliste : AngleResolution ∈ {1, 2, 4, 8}

Ergebnis

hough_line_trans liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hough_line_trans ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, skeleton

Mögliche Nachfolgerfunktionen

threshold, local_max

Siehe auch

hough_circle_trans, gen_region_hline

Modul

Region processing

hough_lines (RegionIn : : AngleResolution, Threshold, AngleGap, DistGap : Angle, Dist)

Findet Linien in Kantenbildern mit Hilfe der Hough-Transformation und liefert sie in Hessescher Normalform zurück.

Mit Hilfe von hough_lines können in einer Region linienhafte Strukturen selektiert werden. Dabei müssen die einzelnen Punkte einer Linie nicht zusammenhängen. Das Verfahren basiert auf der Hough-Transformation. Linien werden in Hessescher Normalform, d.h durch Richtung und Länge ihres Normalenvektors repräsentiert.

Mit AngleResolution wird die Genauigkeit bei der Bestimmung der Winkel definiert. Sie beträgt 1/AngleResolution Grad. Der Parameter Threshold bestimmt, durch wieviel Punkte der Ausgangsregion eine Linienhypothese wenigstens gestützt werden muß, um in die Ausgabe zu gelangen. AngleGap und DistGap definieren eine Umgebung um die Punkte im Hough-Bild für die Bestimmung der lokalen Maxima. Zurückgeliefert werden die Linien in Hessesscher Normalform.

Parameter

▷ **RegionIn** (input_object) . . . . . region ⇝ *Hobject*
Binäres Kantenbild, in dem Linien gefunden werden sollen.

▷ **AngleResolution** (input_control) . . . . . integer ⇝ *integer*
Einstellen der Auflösung im Winkelbereich.
Defaultwert : 4
Werteliste : AngleResolution ∈ {1, 2, 4, 8}

▷ **Threshold** (input_control) . . . . . integer ⇝ *integer*
Schwellenwert im Hough-Bild.
Defaultwert : 100
Typischer Wertebereich : 1 ≤ Threshold

▷ **AngleGap** (input_control) . . . . . integer ⇝ *integer*
Minimaler Abstand zwischen zwei Maxima im Hough-Bild (in Angle-Richtung).
Defaultwert : 5
Typischer Wertebereich : 0 ≤ AngleGap

▷ **DistGap** (input_control) . . . . . integer ⇝ *integer*
Minimaler Abstand zwischen zwei Maxima im Hough-Bild (in Distance-Richtung).
Defaultwert : 5
Typischer Wertebereich : 0 ≤ DistGap

▷ **Angle** (output_control) ..... hesseline.angle.rad-array ⇝ *real*
Winkel (im Bogenmaß) der Normalen-Vektoren der gefundenen Linien.
Typischer Wertebereich : -1.5707963 ≤ Angle ≤ 3.1415927

▷ **Dist** (output_control) ..... hesseline.distance-array ⇝ *real*
Abstände der gefundenen Linien vom Koordinatenursprung.
Typischer Wertebereich : -1.5707963 ≤ Dist ≤ 3.1415927
Parameteranzahl : Dist = Angle

Ergebnis

hough_lines liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system ('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hough_lines ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, skeleton

Mögliche Nachfolgerfunktionen

select_matching_lines

Siehe auch

hough_line_trans, gen_region_hline, hough_circles

Modul

Region processing

select_matching_lines (RegionIn : RegionLines : AngleIn, DistIn, LineWidth, Thresh : AngleOut, DistOut)

Wählt aus einer Menge von Linien (Hessesche Normalform) die Linien aus, die am besten in eine Eingaberegion passen.

Mit select_matching_lines können aus einer Menge von Linien, die in Hessescher Normalform vorliegen zu einer ebenfalls als Parameter übergebenen Region (RegionIn) die Linien ausgewählt werden, die am besten in die Region hineinpassen. Mit LineWidth kann die Breite der Linien angegeben werden. Die selektierten Linien werden sowohl in Hessescher Normalform als auch als Regionen zurückgeliefert.

Die Auswahl der Linien erfolgt iterativ: In einer Schleife wird als erstes die Linie aus der Menge der Eingabelinien ausgewählt, die die größte Überschneidung mit der Eingaberegion besitzt. Diese Linie wird in die Ausgabemenge übernommen, alle Punkte, die zu dieser Linie gehören, werden im Folgenden nicht mehr für die Bestimmung von Überschneidungen berücksichtigt. Unterschreitet, die maximale Überschneidung zwischen der Region und den Linien einen Schwellenwert (Thresh), wird die Schleife verlassen. Die ausgewählten Linien werden sowohl als Regionen, als auch in Hessescher Normalform zurückgegeben.

Parameter

▷ **RegionIn** (input_object) ..... region ⇝ *Hobject*
Region, in denen die Linien gematcht werden sollen.

▷ **RegionLines** (output_object) ..... region-array ⇝ *Hobject*
Regionen-Array, mit den gematchten Linien.

▷ **AngleIn** (input_control) ..... hesseline.angle.rad-array ⇝ *real*
Winkel (im Bogenmaß) der Normalen-Vektoren der Eingabe-Linien.
Typischer Wertebereich : -1.5707963 ≤ AngleIn ≤ 3.1415927

▷ **DistIn** (input_control) ..... hesseline.distance-array ⇝ *real*
Abstände der Eingabe-Linien vom Koordinatenursprung.
Typischer Wertebereich : -1.5707963 ≤ DistIn ≤ 3.1415927
Parameteranzahl : DistIn = AngleIn

▷ **LineWidth** (input_control) . . . . . integer ⇝ *integer*
Breite der Linien.
Defaultwert : 7
Typischer Wertebereich : 1 ≤ LineWidth

▷ **Thresh** (input_control) . . . . . integer ⇝ *integer*
Schwellenwert für die Anzahl der Linienpunkte in der Region.
Defaultwert : 100
Typischer Wertebereich : 1 ≤ Thresh

▷ **AngleOut** (output_control) . . . . . hesseline.angle.rad-array ⇝ *real*
Winkel (im Bogenmas) der Normalen-Vektoren der selektierten Linien.
Typischer Wertebereich : -1.5707963 ≤ AngleOut ≤ 3.1415927
Parameteranzahl : AngleOut ≤ AngleIn

▷ **DistOut** (output_control) . . . . . hesseline.distance-array ⇝ *real*
Abstände der selektierten Linien vom Koordinatenursprung.
Typischer Wertebereich : -1.5707963 ≤ DistOut ≤ 3.1415927
Parameteranzahl : DistOut = AngleOut

Ergebnis

select_matching_lines liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe (keine Eingaberegionen vorhanden) läßt sich mittels set_system('no_object_result',<Result>) festlegen. Das Verhalten bei einer leeren Region (Region ist die leere Menge) wird mit set_system('empty_region_result',<Result>) bestimmt. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

select_matching_lines ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hough_lines

Modul

Region processing

12.8 Kalibrierung

caltab_points (: : CalTabDescrFile : X, Y, Z)

Lesen der Markenmittelpunkte aus Eichkörperbeschreibungsdatei.

caltab_points liest die Markenmittelpunkte aus der Eichkörperbeschreibungsdatei CalTabDescrFile aus. Die Markenmittelpunkte liegen als 3D-Koordinaten im Eichkörperkoordinatensystem vor und stellen damit das 3D-Modell des Eichkörpers dar. Der Operator camera_calibration projiziert diese Modellpunkte in das Bild. Durch Minimierung der Distanz zwischen den projizierten Modellpunkten und den im Bild beobachteten 2D-Koordinaten (vgl. find_marks_and_pose) werden so die genauen Werte für die gesuchten inneren und äußeren Kameraparameter ermittelt.

Parameter

▷ **CalTabDescrFile** (input_control) . . . . . string ⇝ *string*
Dateiname der Eichkörperbeschreibungsdatei.
Defaultwert : ’caltab.descr’

▷ **X** (output_control) . . . . . real-array ⇝ *real*
X-Koordinaten der Markenmittelpunkte.

▷ **Y** (output_control) . . . . . real-array ⇝ *real*
Y-Koordinaten der Markenmittelpunkte.

▷ **Z** (output_control) . . . . . real-array ⇝ *real*
Z-Koordinaten der Markenmittelpunkte.

Beispiel

```
// read_image(Image1,'calib-01.tiff')
```

```
// find calibration pattern */
find_caltab(Image1,Caltab1,'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
// read 3D positions of calibration marks */
caltab_points('caltab.descr',NX,NY,NZ) >
//  camera calibration
camera_calibration(NX,NY,NZ,RCoord1,CCoord1,
  [0.008,0.0,0.000011,0.000011,384,288,768,576],
  StartPose1,11,CamParam,FinalPose,Errors)
//  visualize calibration result
disp_image(Image1,WindowHandle)
set_color(WindowHandle,'red')
disp_caltab('caltab.descr',CamParam,FinalPose,1.0).
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei `CalTabDescrFile` erfolgreich gelesen werden, dann liefert `caltab_points` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`caltab_points` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`camera_calibration`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `project_3d_point`, `get_line_of_sight`, `create_caltab`

Modul

Camera calibration

camera_calibration (: : NX, NY, NZ, NRow, NCol, StartCamParam, NStartPose, EstimateParams : CamParam, NFinalPose, Errors)

Bestimmung aller Kameraparameter durch simultane Ausgleichsrechnung.

`camera_calibration` führt die eigentliche Kalibrierung der Kamera durch. Dazu werden die bekannten 3D-Modellpunkte (mit Koordinaten `NX`, `NY`, `NZ`) ins Bild projiziert und die Summe der Abstandsquadrate zwischen den so gewonnenen Projektionen und ihren korrespondierenden Bildpunkten (mit Koordinaten `NRow`, `NCol`) minimiert. Konvergiert dieses Bündelausgleichsverfahren, so sind damit die exakten inneren (`CamParam`) und äußeren (`NFinalPose`) Kameraparameter der Kamera bestimmt worden. Als Startwerte für das Ausgleichsverfahren dienen die Parameter `StartCamParam` und `NStartPose`. Da mit Hilfe dieses Operators Bild-Modell-Korrespondenzen simultan abgeglichen werden können, die aus unterschiedlichen Bildern stammen, wird dieses Verfahren auch als **Multibildkalibrierung** bezeichnet.

Ganz allgemein bedeutet Kamerakalibrierung, diejenigen Parameter exakt zu bestimmen, die die (optische) Abbildung eines beliebigen 3D-Weltpunktes P_W im Raum auf ein (Sub-)Pixel [r,c] auf dem CCD-Sensor der Kamera modellieren. Dies ist insbesondere von Bedeutung, wenn aus dem Kamerabild die ursprüngliche räumliche Lage von irgendwelchen Gegenständen im Bild bestimmt werden soll, beispielsweise bei der Vermessung von Werkstücken.

Als Kameramodell wird das einer **Lochkamera mit radialer Verzerrung** verwendet, falls die Kammerkonstante in `StartCamParam` mit einem Wert größer als 0 übergeben wird. Es beschreibt die Abbildung eines 3D-Weltpunktes P_W in ein Pixel [r,c] des Videobildes durch folgende Transformationen:

$$P_C = [x \;\; y \;\; z]^T = \mathcal{R} \cdot P_W + \mathcal{T}$$

$$u = \text{Focus} \cdot \frac{x}{z} \quad \text{bzw.} \quad v = \text{Focus} \cdot \frac{y}{z}$$

$$\tilde{u} = \frac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \quad \text{bzw.} \quad \tilde{v} = \frac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}}$$

$$c = \frac{\tilde{u}}{S_x} + C_x \text{ bzw. } r = \frac{\tilde{v}}{S_y} + C_y$$

Falls die Kammerkonstante in `StartCamParam` als 0 übergeben wird, wird als Kameramodell eine **Telezentrische Kamera mit radialer Verzerrung** verwendet, d.h. es wird angenommen, daß die Optik des Kameraobjektives eine Parallelprojektion der Weltpunkte durchführt. Die entsprechenden Gleichungen sind dann:

$$\begin{aligned} P_C = [x\ \ y\ \ z]^T &= \mathcal{R} \cdot P_W + \mathcal{T} \\ u = x \text{ bzw. } v &= y \\ \tilde{u} = \frac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \text{ bzw. } \tilde{v} &= \frac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}} \\ c = \frac{\tilde{u}}{S_x} + C_x \text{ bzw. } r &= \frac{\tilde{v}}{S_y} + C_y \end{aligned}$$

Diese Gleichungen umfassen eine Koordinatentransformation vom Weltkoordinatensystem ins Kamerakoordinatensystem, eine perspektivische bzw. parallele Projektion in die Bildebene, eine radiale Verzerrung dieses projizierten Punktes und schließlich eine Diskretisierung und Hauptpunktverschiebung.

Innerhalb der insgesamt 15 Kameraparameter unterscheidet man zwischen inneren und äußeren Kameraparametern:

Innere Kameraparameter: Sie beschreiben die Beschaffenheit der verwendeten Kamera und beschreiben somit vor allem die interne CCD-Sensorgröße und die Abbildungseigenschaften der verwendeten Kombination von Objektiv, Kamera und Framegrabber. Bei dem oben beschriebenen Kameramodell sind das die folgenden 8 Parameter:

- **Focus:** Kammerkonstante des Objektivs (entspricht realer Brennweite). 0 für telezentrische Objektive.
- **Kappa (κ):** Verzerrungskoeffizient zur Modellierung der kissen- bzw. tonnenförmigen Verzerrung durch das Objektiv.
- **Sx:** Skalierungsfaktor, entspricht dem horizontalen Abstand zweier benachbarter Zellen auf dem CCD-Sensor. Vorsicht: Bei Unterabtastung des Bildes erhöht sich dieser Wert!
- **Sy:** Skalierungsfaktor, entspricht dem vertikalen Abstand zweier benachbarter Zellen auf dem CCD-Sensor. Da in der Regel das Bildsignal zeilensynchron ausgelesen wird, ist dieser Wert durch die Größe des CCD-Sensors exakt festgelegt und muß daher bei Lochkameras nicht durch die Kalibrierung bestimmt werden. Vorsicht: Bei Unterabtastung des Bildes erhöht sich dieser Wert!
- **Cx:** Spaltenkoordinate des Kamerahauptpunktes (optisches Zentrum der radialen Verzerrung).
- **Cy:** Zeilenkoordinate des Kamerahauptpunktes (optisches Zentrum der radialen Verzerrung).
- **ImageWidth:** Breite des abgetasteten Videobildes. Vorsicht: Bei Unterabtastung des Bildes erniedrigt sich dieser Wert!
- **ImageHeight:** Höhe des abgetasteten Videobildes. Vorsicht: Bei Unterabtastung des Bildes erniedrigt sich dieser Wert!

Äußere Kameraparameter: Diese 7 Parameter beschreiben, wo die Kamera positioniert ist und welche Orientierung sie aufweist. Daher werden die äußeren Kameraparameter auch als Kameralage (engl.: camera pose) bezeichnet. Die relative Position der Kamera bzgl. einem Welt- bzw. Objektkoordinatensystem ist durch einen 3D-Translationsvektor festgelegt (Parameter 1,2 und 3). Die Orientierung der Kamera wird in der Regel durch die drei Rotationswinkel um die drei Koordinatenachsen angegeben (Parameter 4,5 und 6).

Der siebte Parameter innerhalb der Kameralage gibt die Codierung des Darstellungstyps der 3D-Transformation an: Z.B. ist der Wert '0' der Code des Typs 1 und bedeutet, daß die Transformation eines Punktes beschrieben ist, hiebei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die neue Y-Achse und schließlich um die nach der zweiten Drehung entstandene X-Achse. `camera_calibration` verarbeitet jedoch alle verschieden Darstellungstypen für `NStartPose`. Die Bedeutung der anderen Darstellungstypen der 3D-Transformation ist bei `create_pose` beschrieben. Während die inneren Kameraparameter für alle Positionen und Orientierungen der Kamera identisch sind, ändern sich die äußeren Kameraparameter natürlich mit der veränderten Kameralage.

Die folgenden Abschnitte befassen sich jeweils mit einzelnen Fragen, die sich bei der Verwendung von `camera_calibration` stellen und sollen daher sowohl dem Verständnis als auch als Leitfaden für die eigene Anwendung dienen.

Wie erzeuge ich einen geeigneten Eichkörper? Die wohl einfachste Methode, die inneren Kameraparameter einer CCD-Kamera zu ermitteln, ist die Verwendung des planaren Standardeichkörpers, wie ihn der Operator `create_caltab` erzeugt. Für den Nahbereich der Kamera genügt es vielleicht sogar, diesen Eichkörper auf einem DIN A4 Laserdrucker auszugeben und auf einen festen Pappkarton aufzuziehen. Für den Fernbereich – insbesondere bei der Verwendung eines Weitwinkelobjektivs – wird so ein Eichkörper zu klein sein. Hier bietet es sich an, die PostScript-Datei auf einem großformatigen Tintenstrahldrucker ausgeben und anschließend z.B. auf Aluminium aufziehen zu lassen. Wichtig ist dabei vor allem, daß die Koordinaten der Marken in der Eichkörperbeschreibungsdatei so genau wie möglich mit dem wirklichen Eichkörper übereinstimmen. Daher sollte unbedingt der gedruckte Eichkörper nachgemessen und die Eichkörperbeschreibungsdatei entsprechend modifiziert werden!

Wie nehme ich eine Menge von geeigneten Bildern auf? Ist ein planarer Standardeichkörper vorhanden, so kann folgendermaßen vorgegangen werden: Mit der zu kalibrierenden Kombination aus Objektiv (mit definierter Entfernungseinstellung), Kamera und Framegrabber wird der Eichkörper aufgenommen, vgl. `open_framegrabber` bzw. `grab_image`. Dabei sollte folgendes berücksichtigt werden:

- Insgesamt sollten mindestens 10 bis 20 Bilder zur Verfügung stehen.
- Der Eichkörper muß jeweils komplett (inkl. Rand!) im Bild sein.
- Es sollten keine Reflexionen o.ä. auf dem Eichkörper sein.
- Innerhalb der Menge von Bildern sollte der Eichkörper jeweils unterschiedlich im Bild erscheinen: Mal links im Bild, mal rechts, mal (links bzw. rechts) unten, mal (links bzw. rechts) oben, und das ganze aus verschiedenen Entfernungen. Dabei sollte der Eichkörper am besten jeweils leicht bis mittel verdreht um seine X- und/oder Y-Achse gehalten werden, so daß die perspektivische Verzerrung der Eichkörpermarken gut sichtbar ist. Die jeweiligen äußeren Kameraparameter (Lage der Kamera bzgl. dem Eichkörper) sollen also viele verschiedene Werte annehmen!
- Der Eichkörper sollte jeweils mindestens etwa ein Viertel des gesamten Bildes ausfüllen, damit die Marken robust detektiert werden können.

Wie extrahiere ich die Eichkörpermarken aus den Bildern? Im Falle der Verwendung des planaren Standardeichkörpers können mit Hilfe der Operatoren `find_caltab` und `find_marks_and_pose` für jedes einzelne aufgenommene Bild die Koordinaten der Eichkörpermarkenmittelpunkte sowie eine grobe Schätzung der äußeren Kameraparameter bestimmt werden. Die Konkatenation dieser Werte kann dann direkt als Startwert für die äußeren Kameraparameter (`NStartPose`) für `camera_calibration` verwendet werden.

Diejenigen Bilder, auf denen die Suche nach dem Eichkörper (`find_caltab`) gescheitert ist bzw. für die der Operator `find_marks_and_pose` nicht die Markenpunkte detektieren konnte, sollten natürlich nicht in die Eingabeparameter von `camera_calibration` eingehen.

Woher nehme ich geeignete Startwerte für die inneren Kameraparameter? Der Operator `find_marks_and_pose` zur Ermittlung von Startwerten für die äußeren Kameraparameter (`NStartPose`) sowie der Operator `camera_calibration` benötigen Startwerte für die inneren Kameraparameter. Diese können in einem geeigneten Format als Datei zur Verfügung stehen (siehe `read_cam_par`). Eine solche Datei kann entweder durch den Operator `write_cam_par` erzeugt oder auch per Hand editiert werden. Für die Startwerte der einzelnen Parameter gilt folgendes:

Focus: Startwert ist gleich der nominalen Brennweite des verwendeten Objektivs, also beispielsweise 0.008 m.

Kappa: Als Startwert den Wert 0.0 verwenden. Der kalibrierte Wert liegt dann in der Regel je nach Objektiv zwischen -1000.0 und -50000.0 $1/m^2$.

Sx: Der Startwert für den horizontalen Abstand zweier benachbarter CCD-Zellen hängt von der Größe des CCD-Chips in der Kamera ab (vgl. technische Daten der Kamera). Man unterscheidet dabei zwischen 1/3Chips (z.B. SONY XC-73, SONY XC-777), 1/2Chips (z.B. SONY XC-999, Panasonic WV-CD50) und 2/3Chips (z.B. SONY DXC-151, SONY XC-77). Hinweise: Bei Unterabtastung des Bildes erhöht sich der Wert für Sx entsprechend! Geeignete Startwerte sind:

	Vollbild (768*576)	Unterabtastung (384*288)
1/3"-Chip	0.0000055 m	0.0000110 m
1/2"-Chip	0.0000086 m	0.0000172 m
2/3"-Chip	0.0000110 m	0.0000220 m

Der Wert für Sx wird kalibriert, da das Videosignal einer Videokamera in der Regel nicht pixelsynchron abgetastet wird.

Sy: Da die handelsüblichen Kameras annähernd quadratische Pixel haben, gelten für Sy dieselben Werte wie für Sx. Der Wert für Sy wird normalerweisefür Lochkameras NICHT kalibriert, da das Videosignal einer Videokamera in der Regel zeilensynchron abgetastet wird und somit der Startwert gleich dem gesuchten endgültigen Wert entspricht. Geeignete Startwerte sind:

	Vollbild (768*576)	Unterabtastung (384*288)
1/3"-Chip	0.0000055 m	0.0000110 m
1/2"-Chip	0.0000086 m	0.0000172 m
2/3"-Chip	0.0000110 m	0.0000220 m

Cx und Cy: Startwerte für die Koordinaten des Kamerahauptpunktes sind die halbe Bildbreite bzw. -hoehe. Hinweis: Bei Unterabtastung des Bildes verringern sich die Werte entsprechend! Geeignete Startwerte sind:

	Vollbild (768*576)	Unterabtastung (384*288)
Cx	384.0	192.0
Cy	288.0	144.0

ImageWidth und ImageHeight: Diese beiden Parameter werden durch die Framegrabberanbindung gesetzt und werden daher natürlich nicht kalibriert. Geeignete Startwerte sind:

	Vollbild (768*576)	Unterabtastung (384*288)
ImageWidth	768	384
ImageHeight	576	288

Welche der Kameraparameter werden geschätzt? Mit dem Eingabeparameter `EstimateParams` kann man angeben, welche der gesuchten Kameraparameter geschätzt werden sollen. Üblicherweise ist dieser Wert gleich **'all'**, d.h. alle 6 äußeren Kameraparameter (Translation und Rotation) und alle inneren Kameraparameter werden bestimmt. Andernfalls enthält `EstimateParams` ein Tupel von Strings, die die Kombination von Parametern, die geschätzt werden soll, enthält.

Wie sieht die Reihenfolge der einzelnen Parameter aus? Die Länge des Tupels `NStartPose` korrespondiert zu der Anzahl der Kalibrierungsbilder, d.h. werden beispielsweise 15 Aufnahmen eines Eichkörpers verwendet, so ergibt sich die Länge des Tupels `NStartPose` zu a$15 \cdot 7 = 105$ (15 mal je 7 äußere Kameraparameter). Die ersten 7 Werte entsprechen daher der Kameralage zum 1. Bild, die nächsten 7 derjenigen des 2. Bildes usw. .

Die dadurch festgelegte Anzahl Kalibrierungsbilder muß auch bei den Tupeln mit den Koordinaten der 3D-Modellmarkenpunkte bzw. der extrahierten 2D-Markenpunkte berücksichtigt werden. Bei einer Anzahl von 15 Kalibrierungsbildern müssen daher die Tupel `NRow` und `NCol` 15mal so viele Werte aufweisen wie die Tupel mit den 3D-Modellmarkenpunkten (`NX`, `NY` und `NZ`). Sind beispielsweise 49 Markenpunkte pro Bild vorhanden, so haben die Tupel `NX`, `NY` und `NZ` jeweils die Länge $15 \cdot 49 = 735$, die Tupel `NRow` und `NCol` haben dagegen die Länge 49. Die Reihenfolge der Werte in `NRow` und `NCol` erfolgt dabei bilderweise, d.h. bei 49 Marken korrespondiert der erste 3D-Modellpunkt mit dem 1., 50., 99., 148., 197., 246. etc. extrahierten 2D-Markenpunkt.

Die 3D-Modellpunkte können mit dem Operator `caltab_points` aus einer Eichkörperbeschreibungsdatei gelesen werden. Startwerte für die Kameralage können für jedes betrachtete Bild mit dem Operator `find_marks_and_pose` ermittelt werden. Das Tupel `NStartPose` ergibt sich dann als Konkatenation dieser einzelnen Kameralagen.

Was bedeuten die Ausgabeparameter? Falls die Kamerakalibrierung erfolgreich verlaufen ist, d.h. das Bündelausgleichsverfahren konvergiert ist, sind in den Ausgabeparametern `CamParam` und `NFinalPose` die auf diese Weise ermittelten genauen Werte für die inneren und äußeren Kameraparameter enthalten. Die Länge des Tupels `NFinalPose` entspricht dann derjenigen von `NStartPose`.

Die Darstellungstypen von `NFinalPose` entsprechen dem Darstellungstyp der des ersten Tupels aus `NStartPose`. Der Darstellungstyp kann jedoch mit `convert_pose_type` gewandelt werden. Die Beschreibung der verschiedenen Darstellungstypen und deren Umwandlung ist bei `create_pose` zu finden.

Anhand des durchschnittlichen Fehlers (`Errors`) kann die erzielte Genauigkeit der Kalibrierung abgeschätzt werden. Die Fehler (Abweichungen in X- und Y-Koordinate) werden in Pixel angegeben.

Muß man einen planaren Eichkörper verwenden? Nein. Der Operator `camera_calibration` ist so ausgelegt, daß er mit den Eingabetupeln `NX`, `NY`, `NZ`, `NRow` und `NCol` beliebige 3D/2D-Korrespondenzen erhält, vgl. obigen Abschnitt über die Länge von `NStartPose`.

Es ist daher prinzipiell egal, wie man auf die nötigen 3D-Modellmarkenpunkte und die korrespondierenden extrahierten 2D-Markenpunkte ermittelt hat. Somit ist es einerseits möglich, auch einen 3D-Eichkörper zu verwenden. Andererseits kann man auch beliebige markante Punkte (natürliche Landmarken) verwenden, deren Position in der Welt bekannt sind. Falls man dann `EstimateParams` auf 6 setzt, kann man diese Weise mit `camera_calibration` die Lage der Kamera in der Welt ermitteln! Hierzu sind dann mindestens drei 3D/2D-Korrespondenzen als Eingabe anzugeben. Zum Erzeugen von `NStartPose` sind die homogenen Transformationsmatrizen hilfreich, vgl. Programmbeispiel bei `hom_mat3d_to_pose`. `NStartPose` kann jedoch auch direkt über `create_pose` generiert werden, vgl. auch Programmbeispiel dort.

Achtung

Die Ausgleichsrechnung der Kalibrierung hängt von den Startwerten für die innneren (`StartCamParam`) und äußeren (`NStartPose`) Kameraparameter ab. Anhand der zurückgegebenen durchschnittlichen Fehler (`Errors`) kann die Genauigkeit der Kalibrierung abgeschätzt werden. Die Fehler (Abweichungen in X- und Y-Koordinate) werden in Pixel angegeben.

Parameter

- ▷ **`NX`** (input_control) ..... real-array ⇝ *real*
  Geordnetes Tupel mit allen X-Koordinaten der Kalibriermarken (in Meter).
- ▷ **`NY`** (input_control) ..... real-array ⇝ *real*
  Geordnetes Tupel mit allen Y-Koordinaten der Kalibriermarken (in Meter).
- ▷ **`NZ`** (input_control) ..... real-array ⇝ *real*
  Geordnetes Tupel mit allen Z-Koordinaten der Kalibriermarken (in Meter).
- ▷ **`NRow`** (input_control) ..... real-array ⇝ *real*
  Geordnetes Tupel mit allen Zeilen-Koordinaten der extrahierten Kalibriermarken (in Pixel).
- ▷ **`NCol`** (input_control) ..... real-array ⇝ *real*
  Geordnetes Tupel mit allen Spalten-Koordinaten der extrahierten Kalibriermarken (in Pixel).
- ▷ **`StartCamParam`** (input_control) ..... real-array ⇝ *real* / integer
  Startwerte für die inneren Kameraparameter.
- ▷ **`NStartPose`** (input_control) ..... pose-array ⇝ *real* / integer
  Geordnetes Tupel mit allen Startwerten der äußeren Kameraparameter.
- ▷ **`EstimateParams`** (input_control) ..... string ⇝ *string* / integer
  Zu schätzenden Kameraparameter.
  Defaultwert : 'all'
  Werteliste : `EstimateParams` ∈ {'all', 'alpha', 'beta', 'gamma', 'transx', 'transy', 'transz', 'focus', 'kappa', 'cx', 'cy', 'sx', 'sy'}
- ▷ **`CamParam`** (output_control) ..... number-array ⇝ *real* / integer
  Innere Kameraparameter.
- ▷ **`NFinalPose`** (output_control) ..... pose-array ⇝ *real* / integer
  Geordnetes Tupel mit allen äußeren Kameraparametern.
- ▷ **`Errors`** (output_control) ..... real-array ⇝ *real*
  Durchschnittlicher Fehler in Pixel.

Beispiel

```
//  read calibration images
read_image(Image1,'calib-01.tiff')
read_image(Image2,'calib-02.tiff')
read_image(Image3,'calib-03.tiff')
//  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],11,CamParam,NFinalPose,Errors)
//  write internal camera parameters to file
write_cam_par(CamParam,'campar.dat').
```

Ergebnis

Sind die Parameterwerte korrekt und konnten die gesuchten Kameraparameter durch das Bündelausgleichsverfahren bestimmt werden, dann liefert `camera_calibration` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt

Parallelisierungsinformation

`camera_calibration` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`find_marks_and_pose`, `caltab_points`, `read_cam_par`

Mögliche Nachfolgerfunktionen

`write_pose`, `pose_to_hom_mat3d`, `disp_caltab`, `sim_caltab`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `disp_caltab`, `sim_caltab`, `write_cam_par`, `read_cam_par`, `create_pose`, `convert_pose_type`, `write_pose`, `read_pose`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`, `caltab_points`, `create_caltab`

Modul

Camera calibration

`change_radial_distortion_cam_par ( : : Mode, CamParIn, Kappa : CamParOut )`

Bestimmung neuer Kameraparameter gemäß eines vorgebenen radialen Verzerrungskoeffizienten.

`change_radial_distortion_cam_par` modifiziert die inneren Kameraparameter gemäß des vorgebenen radialen Verzerrungskoeffizienten `Kappa`. Über den Prameter `Mode` werden dabei folgende Modi unterschieden:

- **’fixed’**: Die Kameraparameter bleiben bis auf den radialen Verzerrungskoeffizienten κ unverändert. Dies bewirkt i.a. eine Änderung des Bildausschnittes.
- **’fullsize’**: Die Skalierungsfaktoren S_x und S_y und der Hauptpunkt $[C_x, C_y]^T$ werden so angepaßt, daß der Bildausschnitt unverändert bleibt. Konkret wird dabei sichergestellt, daß alle Bildpunkte im Originalkamerabild auch im modifizierten (entzerrten) Kamerabild abgebildet werden. Im allgemeinen entstehen dabei undefinierte Pixel im modifizierten Bild.
- **’adaptive’**: Ein Kompromiß zwischen den beiden anderen Modi: Der Bildausschnitt wird etwas verkleinert, um undefinierte Bildbereiche zu vermeiden. Auch hier werden neben κ die Skalierungsfaktoren S_x und S_y und der Hauptpunkt $[C_x, C_y]^T$ modifiziert.

In allen Fällen ist der Verzerrungskoeffizient κ in den Ausgabekameraparametern `CamParOut` gleich `Kappa`. Die Rückrechnung der Pixel im modifizierten Bild in die Bildebene gemäß der neuen inneren Kameraparameter ergibt die gleichen Punkte wie die Rückrechnung der korrespondierenden Subpixel im Originalbild gemäß der alten inneren Kameraparameter `CamParIn`.

Parameter

- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Modus
  Defaultwert : ’adaptive’
  Wertevorschläge : `Mode` ∈ {’fullsize’, ’adaptive’, ’fixed’}
- ▷ **`CamParIn`** (input_control) ..... real-array ⇝ *real*
  Innere Kameraparameter (Original).

▷ **Kappa** (input_control) ..... real ⇝ *real*
Gewuenschter radialer Verzerrungskoeffizient.
Defaultwert : 0.0

▷ **CamParOut** (output_control) ..... real-array ⇝ *real*
Innere Kameraparameter (modifiziert).

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `change_radial_distortion_cam_par` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`change_radial_distortion_cam_par` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`camera_calibration`, `read_cam_par`

Mögliche Nachfolgerfunktionen

`change_radial_distortion_image`, `change_radial_distortion_contours_xld`

Siehe auch

`camera_calibration`, `read_cam_par`, `change_radial_distortion_image`, `change_radial_distortion_contours_xld`

Modul

Camera calibration

change_radial_distortion_contours_xld (Contours : ContoursRectified : CamParIn, CamParOut :)

Radiale Verzerrung von Konturen verändern.

`change_radial_distortion_contours_xld` verändert die radiale Verzerrung der Konturen `Contours` gemäß den übergebenen inneren Kameraparametern `CamParIn` bzw. `CamParOut`. Dazu wird jedes Subpixel der Eingabekonturen vermöge `CamParIn` in die Bildebene zurückgerechnet und dann vermöge `CamParOut` in ein Subpixel der entsprechenden Ausgabekontur überführt.

Wird zur Bestimmung der modifizierten Kameraparameter `CamParOut` die Routine `change_radial_distortion_cam_par` verwendet, entsprechen die Ausgabekonturen `ContoursRectified` den Eingabekonturen bei einer Aufnahmeoptik mit veränderter radialer Verzerrung κ . Für $\kappa = 0$ werden die Konturen radial entzerrt. Eine etwaige Lagebestimmung für die Kamera (externe Kameraparameter) wird dadurch nicht beeinflußt.

Parameter

▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
Eingabekonturen.

▷ **ContoursRectified** (output_object) ..... xld_cont(-array) ⇝ *Hobject*
Ausgabekonturen mit veränderter radialer Verzerrung.

▷ **CamParIn** (input_control) ..... real-array ⇝ *real*
Innere Kameraparameter für `Contours`.

▷ **CamParOut** (input_control) ..... real-array ⇝ *real*
Innere Kameraparameter für `ContoursRectified`.

Parallelisierungsinformation

`change_radial_distortion_contours_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`change_radial_distortion_cam_par`, `gen_contours_skeleton_xld`, `edges_sub_pix`, `smooth_contours_xld`

Mögliche Nachfolgerfunktionen

`gen_polygons_xld`, `smooth_contours_xld`

Siehe auch

`change_radial_distortion_cam_par`, `camera_calibration`, `read_cam_par`, `change_radial_distortion_image`

Modul

Camera calibration

change_radial_distortion_image (Image, Region : ImageRectified : CamParIn, CamParOut :)

Radiale Verzerrung eines Bildes verändern.

change_radial_distortion_image verändert die radiale Verzerrung des Eingabebild Image gemäß den übergebenen inneren Kameraparametern CamParIn bzw. CamParOut. Dazu wird (bei unveränderter Bildgröße) jedes Pixel im Ausgabebild vermöge CamParOut in die Bildebene zurückgerechnet und dann vermöge CamParIn in ein Subpixel im Eingabebild überführt. Der resultierende Grauwert wird aus dem Eingabebild durch bi-lineare Interpolation abgeleitet. Liegt das Subpixel ausserhalb des Eingabebildes, wird das Pixel im Ausgabebild auf 'schwarz' gesetzt und aus der Ausgaberegion entfernt.

Wird zur Bestimmung der modifizierten Kameraparameter CamParOut die Routine change_radial_distortion_cam_par verwendet, entspricht das Ausgabebild ImageRectified dem Eingabebild bei einer Aufnahmeoptik mit geänderter radialer Verzerrung κ . Für $\kappa = 0$ wird das Bild radial entzerrt. Eine etwaige Lagebestimmung für die Kamera (externe Kameraparameter) wird dadurch nicht beeinflußt.

Parameter

- ▷ **Image** (input_object) ..... (multichannel-)image(-array) ⇝ *Hobject*
  Originalbild.
- ▷ **Region** (input_object) ..... region ⇝ *Hobject*
  Interessierender Bildbereich in Image.
- ▷ **ImageRectified** (output_object) ..... (multichannel-)image(-array) ⇝ *Hobject*
  Ausgabebild mit veränderter radialer Verzerrung.
- ▷ **CamParIn** (input_control) ..... real-array ⇝ *real*
  Innere Kameraparameter für Image.
- ▷ **CamParOut** (input_control) ..... real-array ⇝ *real*
  Innere Kameraparameter für ImageRectified.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert change_radial_distortion_image den Wert 2 (H_MSG_TRUE). Bei leerer Eingabe wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

change_radial_distortion_image ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

change_radial_distortion_cam_par, read_image, grab_image

Mögliche Nachfolgerfunktionen

edges_image, threshold

Siehe auch

change_radial_distortion_cam_par, camera_calibration, read_cam_par, change_radial_distortion_contours_xld

Modul

Camera calibration

contour_to_world_plane_xld (Contours : ContoursTrans : CamParam, CamPose, Scale :)

Transformiert eine XLD-Kontur in eine Ebene des Weltkoordinatensystems, die durch die externen Kameraparameter gegeben ist.

contour_to_world_plane_xld transformiert die Konturen Contours in eine Ebene im Weltkoordinatensystem. Letztere ist entweder identisch mit der Ebene der Kalibrierplatte oder — nach Korrektur der Plattendicke mit dem Operator set_origin_pose — parallel zu dieser. Dabei wird zunächst aus den inneren Kameraparametern (CamParam) der Sehstrahl vom Projektionszentrum der Kamera zum Bildpunkt unter Berücksichtigung

der radialen Verzeichnung im System der Kamera berechnet. Anschließend wird dieser Sehstrahl mit den äußeren Kameraparametern (CamPose) in das System der Ebene transformiert. Der Schnittpunkt von Ebene (Z=0) und Sehstrahl entspricht dann den Pseudo-3D-Koordinaten, deren X- und Y- Koordinaten dann in der Ausgabekontur ContoursTrans gespeichert werden. Die erhaltenen Koordinaten können abschließend mit dem Parameter Scale beliebig skaliert werden. Dies ist insbesondere dann sinnvoll, wenn die Koordinaten in einem Bild dargestellt werden sollen. Der Parameter Scale ist dann in $[Einheit/Pixel]$ anzugeben. Die ursprüngliche Einheit ist durch die Koordinaten der Kalibrierpunkte gegeben. In vielen Fällen werden letztere in der Einheit 'Meter' in die Kalibrierung eingeführt, woraus folgt, daß ein Meter einem Pixel im Bild entspricht. In diesem Fall ist es auch möglich, die Pixelgröße direkt durch **'m'**, **'cm'**, **'mm'** oder **'μm'** anzugeben.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
  Zu transformierende Eingabe-XLD-Konturen.
- ▷ **ContoursTrans** (output_object) ..... xld_cont(-array) ⇝ *Hobject*
  Transformierte XLD-Konturen.
- ▷ **CamParam** (input_control) ..... number-array ⇝ *real* / integer
  Innere Kameraparameter.
  Parameteranzahl : 8
- ▷ **CamPose** (input_control) ..... pose-array ⇝ *real* / integer
  Äußere Kameraparameter
  Parameteranzahl : 7
- ▷ **Scale** (input_control) ..... number ⇝ *string* / integer / real
  Maßstab oder Dimension.
  Defaultwert : 'm'
  Wertevorschläge : Scale ∈ {'m', 'cm', 'mm', 'microns', 'μm', 1.0, 0.01, 0.001, '1e-6', 0.0254, 0.3048, 0.9144}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert contour_to_world_plane_xld den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

contour_to_world_plane_xld ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration, set_origin_pose

Siehe auch

image_points_to_world_plane

Modul

Camera calibration

convert_pose_type (: : PoseIn, OrderOfTransform, OrderOfRotation, ViewOfTransform : PoseOut)

Ändert Darstellungstyp von 3D-Lageparametern.

convert_pose_type dient zum Konvertieren von 3D-Lageparameter PoseIn in 3D-Lageparameter PoseOut mit einem anderen Darstellungstypen.

Durch 3D-Lageparameter sind 3D-Transformationen definiert, die aus einer Translation und einer Rotation bestehen. Halcon unterstützt verschiedene Darstellungstypen von solchen Transformationen. Dies sind z.B. die äußeren Kameraparameter, gegeben durch den 3D-Translationsvektor (in Meter), die drei Werte der Rotationswinkel und die Codierung des Darstellungstyps der 3D-Transformation: Beispielsweise ist der Wert '0' der Code des Typs 1 und bedeutet, daß die Transformation eines Punktes beschrieben ist, hierbei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die Y-Achse und schließlich um die X-Achse. Die Bedeutung der anderen Darstellungstypen der 3D-Transformation ist bei create_pose beschrieben.

Mit ViewOfTransform gibt man an, ob die neue Transformations-Vorschrift PoseOut die Transformation eines Punktes (**'point'**) oder die Transformation eines Koordinatensystems (**'coordinate_system'**) beschreiben

soll. `OrderOfTransform` gibt an, ob bei der Transformation zuerst die Rotation (**'Rp+T'**) oder zuerst die Translation (**'R(p-T)'**) durchgeführt wird. Die Bedeutung der drei Rotationswerte wird mit `OrderOfRotation` angegeben. Hierbei bedeuten die Werte **'gba'** und **'abg'**, daß die drei Rotationswerte der 3D-Transformations-Vorschrift die Drehwinkel α (um die x-Achse), β (um die y-Achse) und γ (um die z-Achse) angeben. Bei **'gba'** ist die Drehreihenfolge: γ , β , α und bei **'abg'**: α , β , γ . Ist für `OrderOfRotation` **'rodriguez'** angeben, so werden die Rotationsparameter als Rodriguez-Rotationsvektor ausgewertet.

Parameter

- ▷ **`PoseIn`** (input_control) . . . . . pose-array ⇝ *real* / integer
  3D-Transformations-Vorschrift.
  Parameteranzahl : 7
- ▷ **`OrderOfTransform`** (input_control) . . . . . string ⇝ *string*
  Reihenfolge von Rotation und Translation.
  Defaultwert : "Rp+T"
  Wertevorschläge : `OrderOfTransform` ∈ {"Rp+T", "R(p-T)"}
- ▷ **`OrderOfRotation`** (input_control) . . . . . string ⇝ *string*
  Bedeutung der Rotationswerte.
  Defaultwert : "gba"
  Wertevorschläge : `OrderOfRotation` ∈ {"gba", "abg", "rodriguez"}
- ▷ **`ViewOfTransform`** (input_control) . . . . . string ⇝ *string*
  Sichtweise der Transformation.
  Defaultwert : "point"
  Wertevorschläge : `ViewOfTransform` ∈ {"point", "coordinate_system"}
- ▷ **`PoseOut`** (output_control) . . . . . pose-array ⇝ *real* / integer
  3D-Transformations-Vorschrift.
  Parameteranzahl : 7

Beispiel

```
//  get pose (external camera parameters):
read_pose ('campose.dat', Pose)
//  convert pose to a pose with desired semantic
convert_pose_type (Pose, 'Rp+T', 'abg', 'coordinate_system', Pose2).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `convert_pose_type` den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`convert_pose_type` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_pose`, `hom_mat3d_to_pose`, `camera_calibration`, `hand_eye_calibration`

Mögliche Nachfolgerfunktionen

`write_pose`

Siehe auch

`create_pose`, `get_pose_type`, `write_pose`, `read_pose`

Modul

Camera calibration

`create_caltab` (: : Width, CalTabDescrFile, CalTabFile :)

Erzeugt Eichkörperbeschreibungsdatei und passende PostScript-Datei.

`create_caltab` erzeugt die Beschreibung eines planaren Eichkörpers. Der Eichkörper hat 49 quadratisch angeordnete, schwarze, runde Marken auf weißem Grund, die von einem schwarzen Rahmen umrandet sind. Mit `Width` wird die Breite (gleich der Höhe) des gesamten Eichkörpers in Metern angegeben. Bei einer gewählten Breite von 0.8 m ergibt sich daher der Abstand zwischen zwei benachbarten Marken zu 10 cm, während der Markenradius sowie die Breite des schwarzen Rahmens dann einen Wert von 2.5 cm haben.

Die Datei CalTabDescrFile enthält die eigentliche Eichkörperbeschreibung, d.h. die Anzahl der Zeilen und Spalten des Eichkörpers, die Geometrie des Rahmens (vgl. find_caltab) und die Koordinaten und den Radius aller Eichkörpermarken im Eichkörperkoordinatensystem. Eine solche Datei hat z.B. folgendes Aussehen (Kommentare werden durch ein '#' am Zeilenanfang markiert):

```
#
# Description of the standard calibration table
# used for the CCD-camera calibration in Halcon
# (generated by create\_caltab())
#
# Halcon version 4.11 --  Fri Feb  7 16:13:56 1997
#

# 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 table
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
0.2 0 0.025
0.3 0 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 = 0.2 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 = 0.3 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
```

Die Datei `CalTabFile` enthält die korrespondierende PostScript-Beschreibung des Eichkörpers.

Achtung

Je nach der erzielten Genauigkeit des verwendeten Ausgabegeräts (z.B. Laserdrucker) stimmt der ausgedruckte Eickkörper nicht genau mit der Eichkörperbeschreibungsdatei `CalTabDescrFile` überein. Es müßen daher ggf. die Koordinaten der Eichkörpermarken in der Eichkörperbeschreibungsdatei an die realen Maße angepaßt werden!

Parameter

▷ **`Width`** (input_control) . . . . . real ⇝ *real*
Breite des Eichkörpers in Meter.
Defaultwert : 0.8
Wertevorschläge : `Width` ∈ {1.2, 0.8, 0.6, 0.4, 0.2, 0.1}
Empfohlene Schrittweite : 0.1
Restriktion : `0.0` < `Width`

▷ **`CalTabDescrFile`** (input_control) . . . . . string ⇝ *string*
Dateiname der Eichkörperbeschreibungsdatei.
Defaultwert : ’caltab.descr’

▷ **`CalTabFile`** (input_control) . . . . . string ⇝ *string*
Dateiname der PostScript-Datei.
Defaultwert : ’caltab.ps’

Beispiel

```
//  create calibration table with width = 80 cm
create_caltab(0.8,'caltab.descr','caltab.ps').
```

Ergebnis

Sind die Parameterwerte korrekt und konnten beide Dateien erfolgreich geschrieben werden, dann liefert `create_caltab` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`create_caltab` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`read_cam_par`, `caltab_points`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`

Modul

Camera calibration

create_pose (: : TransX, TransY, TransZ, Rot1, Rot2, Rot3, OrderOfTransform, OrderOfRotation, ViewOfTransform : Pose)

Erzeugt 3D-Lageparameter.

`create_pose` dient zum Erzeugen von 3D-Lageparametern `Pose`, die eine 3D-Transformations-Vorschrift definieren. Die Translation und die Rotation werden jeweils durch 3 Werte (`TransX`, `TransY`, `TransZ` und `Rot1`, `Rot2`, `Rot3`) angeben. Mit `ViewOfTransform` gibt man an, ob die Transformations-Vorschrift die Transformation eines Punktes (**’point’**) oder die Transformation eines Koordinatensystems (**’coordinate_system’**) beschreibt. `OrderOfTransform` gibt an, ob bei der Transformation zuerst die Rotation (**’Rp+T’**) oder zuerst die Tranlation (**’R(p-T)’**) durchgeführt wird. Die Bedeutung der drei Rotationswerte wird mit `OrderOfRotation` angegeben. Hierbei bedeuten die Werte **’gba’** und **’abg’**, daß mit `Rot1` der Drehwinkel α (um die x-Achse), mit `Rot2` der Drehwinkel β (um die y-Achse) und mit `Rot3` der Drehwinkel γ (um die z-Achse) angegeben ist. Bei **’gba’** ist die Drehreihenfolge: γ , β , α und bei **’abg’**: α , β , γ . Ist für `OrderOfRotation` **’rodriguez’** angeben, so werden die Rotationsparameter (`Rot1`, `Rot2`, `Rot3`) als Rodriguez-Rotationsvektor ausgewertet.

`Pose` ist ein Tuple der Länge 7. In den ersten drei Werten wird der Translationsvektor [`TransX`, `TransY`, `TransZ`] abgelegt, in den folgenden drei Werten `Rot1`, `Rot2` und `Rot3`. Der 7. Wert ist der sog. Darstellungstyp der 3D-Lageparameter und der somit definierten 3D-Transformations-Vorschrift.

Aus den Kombinationsmöglichkeiten von `ViewOfTransform`, `OrderOfTransform` und `OrderOfRotation` ergeben sich 12 verschiedene Darstellungstypen für die 3D-Transformationen. Die Typen sind im einzelnen:

Nr.	OrderOfTransform	OrderOfRotation	ViewOfTransform	Code
1	**’Rp+T’**	**’gba’**	**’point’**	0
2	**’Rp+T’**	**’abg’**	**’point’**	2
3	**’Rp+T’**	**’rodriguez’**	**’point’**	4
4	**’Rp+T’**	**’gba’**	**’coordinate_system’**	1
5	**’Rp+T’**	**’abg’**	**’coordinate_system’**	3
6	**’Rp+T’**	**’rodriguez’**	**’coordinate_system’**	5
7	**’R(p-T)’**	**’gba’**	**’point’**	8
8	**’R(p-T)’**	**’abg’**	**’point’**	10
9	**’R(p-T)’**	**’rodriguez’**	**’point’**	12
10	**’R(p-T)’**	**’gba’**	**’coordinate_system’**	9
11	**’R(p-T)’**	**’abg’**	**’coordinate_system’**	11
12	**’R(p-T)’**	**’rodriguez’**	**’coordinate_system’**	13

Dabei ist zu beachten, daß im allgemeinen die Symbole **R** und **T** im Parameter `OrderOfTransform` nicht identisch mit denen sind, die innerhalb der üblichen Notation einer homogenen Transformationsmatrix verwendet werden.

Den Typ einer 3D-Transformations-Vorschrift kann man mit `get_pose_type` abfragen. Man erhält jedoch nicht den Code der Nummer des Typs, sondern die Sicht der Transformation, die Reihenfolge der Transformation und die Reihenfolge der Rotation, entsprechend den Angaben die bei `create_pose` angegeben werden. Die Umwandlung zwischen den einzelnen Typen kann mit `convert_pose_type` vorgenommen werden. Die Beschreibung der Umwandlungen ist weiter unten aufgeführt.

Der Darstellungstyp Nr. 1 mit dem Code '0' entspricht der Transformation mit homogenen Transformations-Matrizen, vgl. `pose_to_hom_mat3d`. Denn bei der Multiplikation eines 3D-Vektors mit der homogenen Transformations-Matrix wird auch zuerst die Rotation und dann die Translation bestimmt, es wird ein Punkt transformiert und die Rotationsreihenfolge ist durch `pose_to_hom_mat3d` auf γ - β - α festgelegt, vgl. auch `affine_trans_point_3d`. Die drei Drehwinkel α , β und γ werden mit den Parametern `Rot1`, `Rot2`, `Rot3` angegeben. Es gilt:

$$\begin{aligned} P_2 &= [x\ \ y\ \ z]^T = \mathcal{R}^{(1)} \cdot P_1 + \mathcal{T}^{(1)} \\ &= \mathcal{R}_x(\alpha) \cdot \mathcal{R}_y(\beta) \cdot \mathcal{R}_z(\gamma) \cdot P_1 + \mathcal{T}^{(1)} \end{aligned}$$

mit:

$$\mathcal{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \quad \text{Rotation um x-Achse}$$

$$\mathcal{R}_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \quad \text{Rotation um y-Achse}$$

$$\mathcal{R}_z(\gamma) = \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{Rotation um z-Achse}$$

$\Rightarrow$

$$\mathcal{R}^{(1)} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

mit:

$$\begin{aligned} r_{11} &= cos(\beta)cos(\gamma) \\ r_{12} &= -cos(\beta)sin(\gamma) \\ r_{13} &= sin(\beta) \\ r_{21} &= sin(\alpha)sin(\beta)cos(\gamma) + cos(\alpha)sin(\gamma) \\ r_{22} &= -sin(\alpha)sin(\beta)sin(\gamma) + cos(\alpha)cos(\gamma) \\ r_{23} &= -sin(\alpha)cos(\beta) \\ r_{31} &= -cos(\alpha)sin(\beta)cos(\gamma) + sin(\alpha)sin(\gamma) \\ r_{32} &= cos(\alpha)sin(\beta)sin(\gamma) + sin(\alpha)cos(\gamma) \\ r_{33} &= cos(\alpha)cos(\beta) \end{aligned}$$

Der Darstellungstyp Nr. 1 ist der Referenztyp für alle Kalibrieroperatoren in Halcon, daher soll hier jeweils die Umwandlungen aller anderen Typen in den Typ 1 angegeben werden, die Umrechnungen sind für die umgekehrten Umwandlungen entsprechend umkehrbar. Bei den Umwandlungen ergeben sich sog. Umwandlungsgrundtypen, dies sind die Umwandlung aus den Typen 2, 3, 4 und 7. Alle anderen Umwandlungen stützen sich auf die Umwandlungsgrundtypen ab. Die einzelnen Umwandlungen sind im folgenden beschrieben:

Typ 2: (**'Rp+T'**, **'abg'**, **'point'**) Gegenüber dem Typ 1 ist die Drehreihenfolge vertauscht. Es wird zuerst um die x-, dann um die neue y-Achse und dann erst um die nach der 2. Drehung entstandene z-Achse gedreht. Mit den Parametern `Rot1`, `Rot2`, `Rot3` werden auch hier die drei Drehwinkel α, β und γ angegeben. Der Translationsvektor [`TransX`, `TransY`, `TransZ`] ist beim Typ 1 und dem Typ 2 identisch. Für die Umrechnung gilt:

$$\begin{aligned} P_2 &= [x\ \ y\ \ z]^T = \mathcal{R} \cdot P_1 + \mathcal{T} \\ &= \mathcal{R}_z(\gamma^{(2)}) \cdot \mathcal{R}_y(\beta^{(2)}) \cdot \mathcal{R}_x(\alpha^{(2)}) \cdot P_1 + \mathcal{T}^{(2)} \\ &= \mathcal{R}_x(\alpha^{(1)}) \cdot \mathcal{R}_y(\beta^{(1)}) \cdot \mathcal{R}_z(\gamma^{(1)}) \cdot P_1 + \mathcal{T}^{(1)} \\ \Rightarrow& \\ \mathcal{R}^{(1)} = \mathcal{R}^{(2)} &= \mathcal{R}_x(\alpha^{(1)}) \cdot \mathcal{R}_y(\beta^{(1)}) \cdot \mathcal{R}_z(\gamma^{(1)}) \cdot P_1 + \mathcal{T}^{(1)} \\ &= \mathcal{R}_z(\gamma^{(2)}) \cdot \mathcal{R}_y(\beta^{(2)}) \cdot \mathcal{R}_x(\alpha^{(2)}) \cdot P_1 + \mathcal{T}^{(2)} \end{aligned}$$

mit:

$$\mathcal{R}^{(2)} = \begin{pmatrix} r_{11}^{(2)} & r_{12}^{(2)} & r_{13}^{(2)} \\ r_{21}^{(2)} & r_{22}^{(2)} & r_{23}^{(2)} \\ r_{31}^{(2)} & r_{32}^{(2)} & r_{33}^{(2)} \end{pmatrix}$$

wobei:

$$\begin{aligned}
r_{11}^{(2)} &= cos(\gamma^{(2)})cos(\beta^{(2)}) \\
r_{12}^{(2)} &= cos(\gamma^{(2)})sin(\beta^{(2)})sin(\alpha^{(2)}) - sin(\gamma^{(2)})cos(\alpha^{(2)}) \\
r_{13}^{(2)} &= cos(\gamma^{(2)})sin(\beta^{(2)})cos(\alpha^{(2)}) + sin(\gamma^{(2)})sin(\alpha^{(2)}) \\
r_{21}^{(2)} &= sin(\gamma^{(2)})cos(\beta^{(2)}) \\
r_{22}^{(2)} &= sin(\gamma^{(2)})sin(\beta^{(2)})sin(\alpha^{(2)}) + cos(\gamma^{(2)})cos(\alpha^{(2)}) \\
r_{23}^{(2)} &= sin(\gamma^{(2)})sin(\beta^{(2)})cos(\alpha^{(2)}) - cos(\gamma^{(2)})sin(\alpha^{(2)}) \\
r_{31}^{(2)} &= -sin(\beta^{(2)}) \\
r_{32}^{(2)} &= cos(\beta^{(2)})sin(\alpha^{(2)}) \\
r_{33}^{(2)} &= cos(\beta^{(2)})cos(\alpha^{(2)})
\end{aligned}$$

$$\begin{aligned}
cos(\beta^{(1)}) &= \sqrt{{r_{11}^{(2)}}^2 + {r_{12}^{(2)}}^2} \\
&\Rightarrow \\
\gamma^{(1)} &= atan2\left(\frac{R_{23}^{(2)}}{cos(\beta^{(1)})}, \frac{R_{33}^{(2)}}{cos(\beta^{(1)})}\right) \\
\beta^{(1)} &= atan2\left(R_{13}^{(2)}, cos(\beta^{(1)})\right) \\
\alpha^{(1)} &= atan2\left(\frac{R_{12}^{(2)}}{cos(\beta^{(1)})}, \frac{R_{11}^{(2)}}{cos(\beta^{(1)})}\right) \\
\mathcal{T}^{(1)} &= \mathcal{T}^{(2)}
\end{aligned}$$

$atan2(y, x)$ steht für $arctan(\frac{x}{y})$ unter Beachtung der Vorzeichen von x und y zur Bestimmung des richtigen Quadranten. Im pathologischen Fall von $cos(\beta^{(1)}) = 0$ ist das Problem nicht lösbar, denn die Repräsentation einer Rotation mit drei Parametern weist eine Singularität auf. In diesem Fall wird die Rotationsmatix minimal gestört, wodurch die Singularität vermieden wird. (Der Winkel β wird hierbei jeweils in 0.00001 Grad-Schritten verändert.)

Typ 3: (**'Rp+T'**, **'rodriguez'**, **'point'**) Der Rodriguez-Vektor ist eine alternative Darstellung einer Rotation im Raum. Die Richtung des Vektors definiert die Drehachse, um die gedreht wird. Die Länge des Vektors bestimmt normalerweise den Drehwinkel im mathematisch positiven Sinn. Hier wird eine Variation des Rodriguez-Vektors verwendet, wobei die Länge den Tangenz des halben Drehwinkel beschreibt.

Die Parameter `Rot1`, `Rot2` und `Rot3` sind dann keine Winkelangaben, sondern entsprechen den drei Werten des Vektor-Tupels $[r_x r_y, r_z]$. Der Translationsvektor [`TransX`, `TransY`, `TransZ`] ist beim Typ 1 und dem Typ 3 identisch. Für die Umwandlung Rotationsteils des Typ 3 in den Typ 1 sei hier die Umrechnung des Rodrigues-Vektors in die Rotationsmatrix $R^{(1)}$ angegeben:

$$
\begin{aligned}
h &= \frac{2}{1+r_x^2+r_y^2+r_z^2} \\
r_{11} &= 1-h(r_y^2+r_z^2) \\
r_{12} &= h(r_x r_y - r_z) \\
r_{13} &= h(r_x r_z + r_y) \\
r_{21} &= h(r_y r_x + r_z) \\
r_{22} &= 1-h(r_z^2+r_x^2) \\
r_{23} &= h(r_y r_z - r_x) \\
r_{31} &= h(r_z r_x - r_y) \\
r_{32} &= h(r_z r_y + r_x) \\
r_{33} &= 1-h(r_x^2+r_y^2)
\end{aligned}
$$

Typ 4: (**'Rp+T'**, **'gba'**, **'coordinate_system'**) Gegenüber dem Typ 1 wird hier nicht die Transformation eines Punktes, sondern die Transformation eines Koordinatensystems beschrieben. Mit den Parametern `Rot1`, `Rot2`, `Rot3` werden auch hier die drei Drehwinkel α, β und γ angegeben. Der Translationsvektor [`TransX`, `TransY`, `TransZ`] ist beim Typ 1 und dem Typ 4 identisch. Es werden lediglich die Drehwinkel negiert. Für die Umrechnung gilt:

$$
\begin{aligned}
P_2 &= [x\ \ y\ \ z]^T = \mathcal{R}\cdot P_1 + \mathcal{T} \\
&= \mathcal{R}_x(\alpha^{(4)})\cdot\mathcal{R}_y(\beta^{(4)})\cdot\mathcal{R}_z(\gamma^{(4)})\cdot P_1 + \mathcal{T}^{(4)} \\
&= \mathcal{R}_x(\alpha^{(1)})\cdot\mathcal{R}_y(\beta^{(1)})\cdot\mathcal{R}_z(\gamma^{(1)})\cdot P_1 + \mathcal{T}^{(1)}
\end{aligned}
$$

mit:

$$
\begin{aligned}
\mathcal{T}^{(1)} &= \mathcal{T}^{(4)} \\
\alpha^{(1)} &= -\alpha^{(4)} \\
\beta^{(1)} &= -\beta^{(4)} \\
\gamma^{(1)} &= -\gamma^{(4)}
\end{aligned}
$$

Typ 5: (**'Rp+T'**, **'abg'**, **'coordinate_system'**) Bei der Umwandlung von Typ 5 in den Typ 1 kann zunächst in den Typ 2 gewandelt werden und dann vom Typ 2 in Typ 1, vgl. die Erklärungen dort. Typ 5 und Typ 2 unterscheiden sich durch die Sichtweise der Transformation (`ViewOfTransform`). Es muß daher wie bei der Umwandlung von Typ 4 in den Typ 1 die Vorzeichen der Drehwinkel geändert werden. Mit den Parametern `Rot1`, `Rot2`, `Rot3` werden auch hier die drei Drehwinkel α, β und γ angegeben. Der Translationsvektor [`TransX`, `TransY`, `TransZ`] ist beim Typ 1 und dem Typ 5 identisch. Für die Umwandlung in den Typ 2 gilt:

$$
\begin{aligned}
P_2 &= [x\ \ y\ \ z]^T = \mathcal{R}\cdot P_1 + \mathcal{T} \\
&= \mathcal{R}_z(\gamma^{(5)})\cdot\mathcal{R}_y(\beta^{(5)})\cdot\mathcal{R}_x(\alpha^{(5)})\cdot P_1 + \mathcal{T}^{(5)} \\
&= \mathcal{R}_z(\gamma^{(2)})\cdot\mathcal{R}_y(\beta^{(2)})\cdot\mathcal{R}_x(\alpha^{(2)})\cdot P_1 + \mathcal{T}^{(2)}
\end{aligned}
$$

mit:

$$
\begin{aligned}
\mathcal{T}^{(2)} &= \mathcal{T}^{(5)} \\
\alpha^{(2)} &= -\alpha^{(5)} \\
\beta^{(2)} &= -\beta^{(5)} \\
\gamma^{(2)} &= -\gamma^{(5)}
\end{aligned}
$$

Typ 6: (**'Rp+T'**, **'rodriguez'**, **'coordinate_system'**) Beim Typ 6 ist die Rotation, wie beim Typ 3, durch einen Rodrigues-Vektors angegeben. Die Parameter `Rot1`, `Rot2` und `Rot3` sind dann hier keine Winkelangaben, sondern entsprechen den drei Werten des Vektor-Tupels [r_x, r_y, r_z]. Der Translationsvektor (`TransX`, `TransY`, `TransZ`) ist beim Typ 1 und dem Typ 6 identisch. Die Umwandlung des Rotationsteils in den Typ 1 erfolgt auch hier in zwei Schritten: Zunächst wird aus dem Vektor-Tupel [r_x, r_y, r_x] die Rotationsmatrix erstellt, damit erhält man eine Darstellung im Typ 4. Vgl. hierzu die Erläuterungen der Umwandlung von Typ 3 in Typ 1.

Im zweiten Schritt wird von Typ 4 in Typ 1 gewandelt, hierzu sind dann noch die Drehwinkel zu negieren. Vgl. die Eräuterungen zur Umwandlung von Typ 4 in Typ 1.

Typ 7: (**'R(p-T)'**, **'gba'**, **'point'**) Im Unterschied zum Typ 1 wird mit dem Typ 7 eine Transformation beschrieben, bei der zuerst die Translation und dann erst die Rotation ausgeführt wird. Mit den Parametern `Rot1`, `Rot2` und `Rot3` werden hier ebenfalls die drei Drehwinkel α, β und γ angegeben. Für die Transformation gilt:

$$
\begin{aligned}
P_2 &= [x\ \ y\ \ z]^T = \mathcal{R}^{(1)}\cdot P_1 + \mathcal{T}^{(1)} \\
&= \mathcal{R}^{(7)}\cdot\left(P_1 - \mathcal{T}^{(7)}\right) \\
&= \mathcal{R}^{(7)}\cdot P_1 - \mathcal{R}^{(7)}\cdot\mathcal{T}^{(7)} \\
\Rightarrow
\end{aligned}
$$

$$\begin{aligned} \mathcal{R}^{(1)} &= \mathcal{R}^{(7)} \\ \mathcal{T}^{(1)} &= -\mathcal{R}^{(7)} \cdot \mathcal{T}^{(7)} \end{aligned}$$

Typ 8: (**'R(p-T)'**, **'abg'**, **'point'**) Die Umwandlung des Typ 8 in den Typ 1 erfolgt wieder in 2 Schritten: Zunächst eine Umwandlung in den Typ 7, vgl. die Erläuterungen bei der Umwandlung von Typ 2 in Typ 1. Im zweiten Schritt wird dann eine Umwandlung von Typ 7 in Typ 1 vorgenommen, vgl. die Erläuterungen dort. Mit den Parametern `Rot1`, `Rot2` und `Rot3` werden hier ebenfalls die drei Drehwinkel α, β und γ angegeben.

Typ 9: (**'R(p-T)'**, **'rodriguez'**, **'point'**) Die Umwandlung des Typ 9 in den Typ 1 erfolgt wieder in 2 Schritten: Zunächst eine Umwandlung in den Typ 7, vgl. die Erläuterungen bei der Umwandlung von Typ 3 in Typ 1. Im zweiten Schritt wird dann eine Umwandlung von Typ 7 in Typ 1 vorgenommen, vgl. die Erläuterungen dort. Die Parameter `Rot1`, `Rot2` und `Rot3` sind dann hier keine Winkelangaben, sondern entsprechen den drei Werten des Rodriguez-Vektor-Tupels $[r_x, r_y, r_z]$.

Typ 10: (**'R(p-T)'**, **'gba'**, **'coordinate_system'**) Die Umwandlung des Typ 10 in den Typ 1 erfolgt wieder in 2 Schritten: Zunächst eine Umwandlung in den Typ 7, vgl. die Erläuterungen bei der Umwandlung von Typ 4 in Typ 1. Im zweiten Schritt wird dann eine Umwandlung von Typ 7 in Typ 1 vorgenommen, vgl. die Erläuterungen dort. Mit den Parametern `Rot1`, `Rot2` und `Rot3` werden hier ebenfalls die drei Drehwinkel α, β und γ angegeben.

Typ 11: (**'R(p-T)'**, **'abg'**, **'coordinate_system'**) Die Umwandlung des Typ 11 in den Typ 1 erfolgt in 3 Schritten: Zunächst eine Umwandlung in den Typ 8, vgl. die Erläuterungen bei der Umwandlung von Typ 4 in Typ 1. Im zweiten Schritt wird dann eine Umwandlung von Typ 8 in Typ 7 vorgenommen, vgl. die Erläuterungen zur Umwandlung von Typ 2 in Typ 1. Im dritten Schritt wird dann schließlich die Umwandlung von Typ 7 in Typ 1 vorgenommen, vgl. die Erläuterungen dort. Mit den Parametern `Rot1`, `Rot2` und `Rot3` werden hier ebenfalls die drei Drehwinkel α, β und γ angegeben.

Typ 12: (**'R(p-T)'**, **'rodriguez'**, **'coordinate_system'**) Die Umwandlung des Typ 12 in den Typ 1 erfolgt auch in 3 Schritten: Zunächst eine Umwandlung in den Typ 10, vgl. die Erläuterungen bei der Umwandlung von Typ 3 in Typ 1. Im zweiten Schritt wird dann eine Umwandlung von Typ 10 in Typ 7 vorgenommen, vgl. die Erläuterungen zur Umwandlung von Typ 4 in Typ 1. Im dritten Schritt wird dann schließlich die Umwandlung von Typ 7 in Typ 1 vorgenommen, vgl. die Erläuterungen dort. Die Parameter `Rot1`, `Rot2` und `Rot3` sind hier keine Winkelangaben, sondern entsprechen den drei Werten des Rodriguez-Vektor-Tupels $[r_x, r_y, r_z]$.

Als Tip: Bei der Bescheibung der Lage eines zweiten Koordinatensystems in einem Basiskoordinatensystem ist es am einleuchtendsten, wenn man zunächst davon ausgeht, daß die Koordinatensysteme identisch sind, also exakt aufeinander liegen. Danach verschiebt man das zweite Koordinatensystem bis der Ursprung in der entsprechenden Position zu liegen kommt. Der Vektor, der diese Translation beschreibt, ist der Translations-Vektor, der mit [`TransX`, `TransY`, `TransZ`] anzugeben ist. Danach werden die Rotationen ausgeführt bis die gewünschte Orientierung des zweiten Koordinatensystem im Basiskoordinatensystem erreicht ist. Wählt man als Drehreihenfolge γ - β - α , so dreht man zunächst um die z-Achse im mathematisch positiven Sinn. Danach um die neu entstandene y-Achse und schließlich um die nach der zweiten Drehung entstandene x-Achse. Der Drehwinkel um die x-Achse enstspricht α und muß als `Rot1` angegeben werden, der Drehwinkel um die y-Achse entspricht β und wird als `Rot2` angegeben. Der Drehwinkel um die z-Achse entspricht γ und ist als `Rot3` anzugeben. Diese Beschreibung der 3D-Transformation entspricht dem Typ 10 mit dem Code '9', daher ist `ViewOfTransform` = **'coordinate_system'**, `OrderOfTransform` = **'R(p-T)'** und `OrderOfRotation` = **'gba'** zu setzen.

Im angegebenen Programmbeispiel ist gezeigt, wie man die initiale Kameralage für einen anschließenden Aufruf von `camera_calibration` bestimmen kann, wenn man die Kamera grob relativ zum Weltkoordinatensystem vermißt. Dies ist insbesondere von Interesse, falls nicht der planare Standardeichkörper (vgl. `create_caltab`) verwendet wird und somit der Operator `find_marks_and_pose` nicht angewandt werden kann. Stattdessen können natürlich oder künstliche Landmarken mit bekannter Weltposition verwendet werden.

Sei beispielsweise die relative Kameraposition vom Ursprung des Weltkörperkoordinatensystems etwa die folgende: x = 1.08 m, y = 0.25 m, z = 0.62 m. Das heißt, daß der Ursprung des Kamerakoordinatensystems im Weltkoordinaten die Koordinaten [1.08, 0.25, 0.62] besitzt. Das Kamerakoordinatensystem ist so orientiert, daß der Sichtstrahl der Kamera der positiven Z-Achse entspricht. Die Orientierung des Kamerakoordinatensystems relativ zum Weltkoordinatensystem kann durch drei Rotationswinkel angegeben werden. Im Beispiel wird das Kamerakoordinatensystem erst um 100 Grad um die z-Achse des Weltkoordinatensystems gedreht und anschließend um -120 Grad um die neue (!!!) x-Achse gedreht. Vgl. auch das Beispiel bei `hom_mat3d_to_pose`.

Parameter

▷ **TransX** (input_control) . . . . . real ⇝ *real*
Translation in X-Achsenrichtung.
Defaultwert : 0.1
Wertevorschläge : TransX ∈ {-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}
Typischer Wertebereich : 0.05 ≤ TransX
Empfohlene Schrittweite : 0.05

▷ **TransY** (input_control) . . . . . real ⇝ *real*
Translation in Y-Achsenrichtung.
Defaultwert : 0.1
Wertevorschläge : TransY ∈ {-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}
Typischer Wertebereich : 0.05 ≤ TransY
Empfohlene Schrittweite : 0.05

▷ **TransZ** (input_control) . . . . . real ⇝ *real*
Translation in Z-Achsenrichtung.
Defaultwert : 0.1
Wertevorschläge : TransZ ∈ {-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}
Typischer Wertebereich : 0.05 ≤ TransZ
Empfohlene Schrittweite : 0.05

▷ **Rot1** (input_control) . . . . . real ⇝ *real*
1. Rotationswert.
Defaultwert : 90
Wertevorschläge : Rot1 ∈ {90, 180, 270}
Typischer Wertebereich : 0 ≤ Rot1 ≤ 360
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.2

▷ **Rot2** (input_control) . . . . . real ⇝ *real*
2. Rotationswert.
Defaultwert : 90
Wertevorschläge : Rot2 ∈ {90, 180, 270}
Typischer Wertebereich : 0 ≤ Rot2 ≤ 360
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.2

▷ **Rot3** (input_control) . . . . . real ⇝ *real*
3. Rotationswert.
Defaultwert : 90
Wertevorschläge : Rot3 ∈ {90, 180, 270}
Typischer Wertebereich : 0 ≤ Rot3 ≤ 360
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.2

▷ **OrderOfTransform** (input_control) . . . . . string ⇝ *string*
Reihenfolge von Rotation und Translation.
Defaultwert : ”Rp+T”
Wertevorschläge : OrderOfTransform ∈ {”Rp+T”, ”R(p-T)”}

▷ **OrderOfRotation** (input_control) . . . . . string ⇝ *string*
Bedeutung der Rotationswerte.
Defaultwert : ”gba”
Wertevorschläge : OrderOfRotation ∈ {”gba”, ”abg”, ”rodriguez”}

▷ **ViewOfTransform** (input_control) . . . . . string ⇝ *string*
Sichtweise der Transformation.
Defaultwert : ”point”
Wertevorschläge : ViewOfTransform ∈ {”point”, ”coordinate_system”}

▷ **Pose** (output_control) . . . . . pose-array ⇝ *real* / integer
3D-Transformations-Vorschrift.
Parameteranzahl : 7

Beispiel

```
//  get internal camera parameters:
read_cam_par ('campar.dat',CamParam)
//  read 3D world points [WorldPointsX,WorldPointsY,WorldPointsZ]
//  extract correspondent 2D image points [PixelsRow,PixelsColumn]
//  get rough camera start pose from relative camera pose:
create_pose (1.08, 0.25, 0.62, -120, 0, 100, "R(p-T)", "gba",
             "coordinate_system", StartPose)
//  calibration of external camera params:
camera_calibration (WorldPointsX, WorldPointsY, WorldPointsZ,
                    PixelsRow, PixelsColumn, CamParam, StartPose, 6,
                    Dummy, FinalPose, Errors).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `create_pose` den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`create_pose` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`pose_to_hom_mat3d`, `write_pose`, `camera_calibration`, `hand_eye_calibration`

Alternativen

`read_pose`, `hom_mat3d_to_pose`

Siehe auch

`convert_pose_type`, `get_pose_type`, `hom_mat3d_to_pose`, `pose_to_hom_mat3d`, `write_pose`, `read_pose`

Modul

Camera calibration

`disp_caltab` (: : WindowHandle, CalTabDescrFile, CamParam, CamPose, ScaleFac :)

Projektion und Visualisierung des 3D-Eichkörpermodells ins Bild.

`disp_caltab` visualisiert im aktuellen Ausgabefenster die Marken und die Verbindungslinien zwischen den Marken des durch `CalTabDescrFile` beschriebenen Eichkörpers. Dazu wird das 3D-Modell des Eichkörpers mittels der inneren Kameraparameter (`CamParam`) und der Kameralage (äußere Kameraparameter `CamPose`) auf die Bildebene projiziert. Das verwendete Kameramodell (Lochkamera mit radialer Verzerrung) wird bei `write_cam_par` beschrieben.

Typischerweise wird `disp_caltab` verwendet, um das Ergebnis der Kamerakalibrierung (vgl. `camera_calibration`) durch Überblendung auf das originale Bild zu verifizieren. Die aktuelle Strichstärke kann mit `set_line_width`, die aktuelle Zeichenfarbe mit `set_color` gesetzt werden.

Der Parameter `ScaleFac` beeinflußt die Anzahl der Stützstellen, mit der die ellipsenförmigen Konturen der Eichkörpermarken approximiert werden. Die Erhöhung der Stützstellenanzahl ist sinnvoll, wenn der Bildbereich im aktuellen Ausgabefenster vergrößert dargestellt wird (vgl. `set_part`).

Parameter

- ▷ **`WindowHandle`** (input_control) ..... window ⇝ *integer*
  Fenster_id.
- ▷ **`CalTabDescrFile`** (input_control) ..... string ⇝ *string*
  Dateiname der Eichkörperbeschreibungsdatei.
  Defaultwert : ’caltab.descr’
- ▷ **`CamParam`** (input_control) ..... number-array ⇝ *real* / integer
  Innere Kameraparameter.
  Parameteranzahl : 8
- ▷ **`CamPose`** (input_control) ..... pose-array ⇝ *real* / integer
  Äußere Kameraparameter.
  Parameteranzahl : 7

▷ **ScaleFac** (input_control) . . . . . real ⇝ *real*
Skalierungsfaktor für Darstellung.
Defaultwert : 1.0
Wertevorschläge : ScaleFac ∈ {0.5, 1.0, 2.0, 3.0}
Empfohlene Schrittweite : 0.05
Restriktion : 0.0 < ScaleFac

Beispiel

```
// read calibration image
read_image(Image1,'calib-01.tiff')
// find calibration pattern
find_caltab(Image1,Caltab1,'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
// read 3D positions of calibration marks
caltab_points('caltab.descr',NX,NY,NZ)
// camera calibration
camera_calibration(NX,NY,NZ,RCoord1,CCoord1,
  [0.008,0.0,0.000011,0.000011,384,288,768,576],
  StartPose1,11,CamParam,FinalPose,Errors)
//  visualize calibration result
disp_image(Image1,WindowHandle)
set_color(WindowHandle,'red')
disp_caltab('caltab.descr',CamParam,FinalPose,1.0).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert disp_caltab den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

disp_caltab ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

camera_calibration, read_cam_par, read_pose

Siehe auch

find_marks_and_pose, camera_calibration, sim_caltab, write_cam_par, read_cam_par, create_pose, write_pose, read_pose, project_3d_point, get_line_of_sight

Modul

Camera calibration

find_caltab (Image : Caltab : CalTabDescrFile, SizeGauss, MarkThresh, MinDiamMarks :)

Segmentation der Eichkörperregion im Bild.

find_caltab bestimmt die Region eines planaren Eichkörpers mit runden Marken im übergebenen Bild Image. Dazu wird das Bild zuerst geglättet (vgl. gauss_image); die Größe der verwendeten Filtermaske wird durch SizeGauss eingestellt. Anschließend wird ein Schwellenwertoperator (vgl. threshold) mit minimalem Grauwert MarkThresh und maximalem Grauwert 255 angewandt. Unter den extrahierten zusammenhängenden Regionen wird nun eine möglichst konvexe Region mit annähernd korrekter Anzahl von Löchern (entsprechend den dunklen Marken) gesucht. Um Rauschen zu unterdrücken, werden kleine Löcher mit geringerem Durchmesser als die erwartete Markengröße MinDiamMarks zuvor eliminiert. Die Anzahl der Eichkörpermarken wird der Beschreibungsdatei CalTabDescrFile des Eichkörpers entnommen. Die komplette Erklärung dieser Datei ist bei der Beschreibung von create_caltab zu finden.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **Caltab** (output_object) ..... region ⇝ *Hobject*
  Ausgaberegion.
- ▷ **CalTabDescrFile** (input_control) ..... string ⇝ *string*
  Dateiname der Eichkörperbeschreibungsdatei.
  Defaultwert : 'caltab.descr'
- ▷ **SizeGauss** (input_control) ..... integer ⇝ *integer*
  Filtergröße.
  Defaultwert : 3
  Werteliste : SizeGauss ∈ {0, 3, 5, 7, 9, 11}
- ▷ **MarkThresh** (input_control) ..... integer ⇝ *integer*
  Schwellenwert zur Markenextraktion.
  Defaultwert : 112
  Werteliste : MarkThresh ∈ {48, 64, 80, 96, 112, 128, 144, 160}
- ▷ **MinDiamMarks** (input_control) ..... integer ⇝ *integer*
  Erwarteter Mindestdurchmesser der Eichkörpermarken.
  Defaultwert : 5
  Werteliste : MinDiamMarks ∈ {3, 5, 9, 15, 30, 50, 70}

Beispiel

```
// read calibration image
read_image(Image,'calib-01.tiff')
// find calibration pattern
find_caltab(Image,Caltab,'caltab.descr',3,112,5).
```

Ergebnis

Sind die Parameterwerte korrekt und findet find_caltab eine passende Bildregion, dann liefert find_caltab den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (kein Eingabebild vorhanden) läßt sich mittels set_system(::'no_object_result',<Result>:) und das bei leerer Ergebnisregion mit set_system(::'store_empty_region',<true/false>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

find_caltab ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_image

Mögliche Nachfolgerfunktionen

find_marks_and_pose

Siehe auch

find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab, caltab_points, create_caltab

Modul

Camera calibration

find_marks_and_pose (Image, CalTabRegion : : CalTabDescrFile, StartCamParam, StartThresh, DeltaThresh, MinThresh, Alpha, MinContLength, MaxDiamMarks : RCoord, CCoord, StartPose)

Extraktion der 2D-Kalibriermarken aus dem Videobild sowie Bestimmung der Startwerte für die äußeren Kameraparameter.

find_marks_and_pose dient zur Berechnung der notwendigen Eingabedaten für die eigentliche Kalibrierung (vgl. camera_calibration): Zum einen werden die 2D-Mittelpunkte [RCoord,CCoord] der Eichkörpermarken innerhalb der Region CalTabRegion im Bild Image extrahiert und geordnet. Außerdem wird eine

grobe Schätzung für die äußeren Kameraparameter (StartPose) berechnet, d.h. für die Lage der Kamera im Eichkörperkoordinatensystem.

Innerhalb der Region CalTabRegion, die beispielsweise von dem Operator find_caltab detektiert wurde, im Eingabebild Image wird ein Kantenoperator angewendet, vgl. Operator edges_image, Modus 'lanser2'. Der Filterparameter für diese Kantendetektion ist über Parameter Alpha einstellbar. Im Kantenbild werden geschlossene Konturen gesucht, deren Anzahl gerade derjenigen aus der Eichkörperbeschreibungsdatei CalTabDescrFile entspricht und die ein ellipsenförmiges Aussehen besitzen. Konturen, die kürzer als MinContLength sind, werden dabei verworfen, ebenso Konturen, die eine Bildregion mit einem Durchmesser größer als MaxDiamMarks umschließen (also z.B. die Umrandung des Eichkörpers selbst). Zur Kontursuche im Kantenbild wird ein Schwellenwertoperator angewandt, der die hellen Bereiche im Amplitudenbild des Kantenoperators segmentiert. Zunächst wird der Schwellenwert auf den Wert StartThresh gesetzt. Schlägt die Suche nach den Konturen bzw. die nachfolgende Schätzung der Kameralage fehl, wird der Schwellenwert sukzessive um DeltaThresh bis minimal auf den Wert MinThresh erniedrigt.

Jede der gefundenen Konturen wird subpixel-genau verfeinert (vgl. edges_sub_pix) und anschließend vermöge eines Ausgleichsverfahrens durch eine Ellipse approximiert. Die Mittelpunkte dieser Ellipsen stellen eine gute Näherung and die gesuchten 2D-Bildkoordinaten [RCoord,CCoord] der Eichkörpermarkenmittelpunkte dar. Die Reihenfolge der einzelnen Werte innerhalb dieser beiden Tupel erfolgt zeilenweise von links oben nach rechts unten im Bild und muß unbedingt mit der Reihenfolge der 3D-Koordinaten der Marken in der Eichkörperbeschreibungsdatei CalTabDescrFile übereinstimmen, da so die Korrespondenzen zwischen den extrahierten Bild- und den bekannten Modellmarken festgelegt werden!

Aus den Ellipsenparametern der einzelnen Marken wird abschließend noch eine grobe Schätzung für die äußeren Kameraparameter berechnet. Dazu werden die aufgestellten Korrespondenzen zwischen den extrahierten Bild- und den bekannten Modellmarken verwendet, die sich durch die Reihenfolge der Markenpunkte in [RCoord,CCoord] bzw. in der Datei CalTabDescrFile ergeben. Die Schätzung StartPose beschreibt also gerade die Lage der Kamera im Eichkörperkoordinatensystem (vgl. create_pose), wie sie als Startwert für die eigentliche Kamerakalibrierung durch den Operator camera_calibration benötigt wird.

Parameter

- ▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **CalTabRegion** (input_object) ..... region ⇝ *Hobject*
  Eichkörperregion.
- ▷ **CalTabDescrFile** (input_control) ..... string ⇝ *string*
  Dateiname der Eichkörperbeschreibung.
  Defaultwert : 'caltab.descr'
- ▷ **StartCamParam** (input_control) ..... number-array ⇝ *real* / integer
  Startwerte für die inneren Kameraparameter.
- ▷ **StartThresh** (input_control) ..... number ⇝ *integer*
  Startschwellwert zur Konturdetektion.
  Defaultwert : 128
  Werteliste : StartThresh ∈ {80, 96, 112, 128, 144, 160}
- ▷ **DeltaThresh** (input_control) ..... number ⇝ *integer*
  Schleifenwert zur Verkleinerung von StartThresh.
  Defaultwert : 10
  Werteliste : DeltaThresh ∈ {6, 8, 10, 12, 14, 16, 18, 20, 22}
- ▷ **MinThresh** (input_control) ..... number ⇝ *integer*
  Minimaler Schwellwert zur Konturdetektion.
  Defaultwert : 18
  Werteliste : MinThresh ∈ {8, 10, 12, 14, 16, 18, 20, 22}
- ▷ **Alpha** (input_control) ..... real ⇝ *real*
  Filterparameter zur Konturdetektion, vgl. edges_image.
  Defaultwert : 0.9
  Wertevorschläge : Alpha ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1}
  Typischer Wertebereich : 0.2 ≤ Alpha ≤ 50.0
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Alpha > 0.0

▷ **MinContLength** (input_control) ..... real ⇝ *real*
Minimale Länge der Markenkonturen.
Defaultwert : 15.0
Wertevorschläge : MinContLength ∈ {10.0, 15.0, 20.0, 30.0, 40.0, 100.0}
Restriktion : MinContLength > 0.0

▷ **MaxDiamMarks** (input_control) ..... real ⇝ *real*
Maximal erwarteter Markendurchmesser.
Defaultwert : 100.0
Wertevorschläge : MaxDiamMarks ∈ {50.0, 100.0, 150.0, 200.0, 300.0}
Restriktion : MaxDiamMarks > 0.0

▷ **RCoord** (output_control) ..... real-array ⇝ *real*
Tupel aller detektierten Zeilen-Koordinaten (in Pixel).

▷ **CCoord** (output_control) ..... real-array ⇝ *real*
Tupel aller detektierten Spalten-Koordinaten (in Pixel).

▷ **StartPose** (output_control) ..... pose-array ⇝ *real* / integer
Schätzung für die äußeren Kameraparameter.
Parameteranzahl : 7

Beispiel

```
//  read calibration image
read_image(Image,'calib-01.tiff')
//  find calibration pattern
find_caltab(Image,Caltab1,'caltab.descr',3,112,5)
//  find calibration marks and start pose
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).
```

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Schätzung für die äußeren Kameraparameter bestimmt werden, dann liefert find_marks_and_pose den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

find_marks_and_pose ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

find_caltab

Mögliche Nachfolgerfunktionen

camera_calibration

Siehe auch

find_caltab, camera_calibration, disp_caltab, sim_caltab, read_cam_par, read_pose, create_pose, pose_to_hom_mat3d, caltab_points, create_caltab, edges_sub_pix, edges_image

Modul

Camera calibration

get_line_of_sight (: : Row, Column, CamParam : PX, PY, PZ, QX, QY, QZ)

Berechnung des zu einem Bildpunkt gehörigen Sichtstrahls.

get_line_of_sight berechnet für ein Pixel im Rechnerkoordinatensystem (Row, Column) den zugehörigen Sichtstrahl. Der Sichtstrahl ist eine Gerade im Kamerakoordinatensystem, die durch zwei auf ihr liegende Punkte (PX,PY,PZ) und (QX,QY,QZ) beschrieben wird. Es wird ein Loch- oder Telezentrisches Kameramodell mit radialer Verzerrung zugrundegelegt, das durch die inneren Kameraparameter CamParam beschrieben wird (siehe camera_calibration). Falls eine Lochkamera verwendet wird, liegt der zweite Punkt in der Brennpunktebene, d.h. der Ausgabeparameter QZ entspricht gerade der Brennweite der Kamera. Die Geradengleichung des Sichtstrahls ist gegeben durch

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} p\_x \\ p\_y \\ p\_z \end{pmatrix} + t \begin{pmatrix} q\_x - p\_x \\ q\_y - p\_y \\ q\_z - p\_z \end{pmatrix} .$$

Der Vorteil der Darstellung der Geraden durch zwei Punkte ist, daß es hiermit einfacher wird, die Gerade im Raum zu transformieren. Es muß lediglich der Operator `affine_trans_point_3d` auf die beiden Punkte angewandt werden.

Parameter

- ▷ **Row** (input_control) ..... real-array ⇝ *real*
  Zeilennummer des Pixels.
- ▷ **Column** (input_control) ..... real-array ⇝ *real*
  Spaltennummer des Pixels.
- ▷ **CamParam** (input_control) ..... number-array ⇝ *real* / integer
  Innere Kameraparameter.
  Parameteranzahl : 8
- ▷ **PX** (output_control) ..... real-array ⇝ *real*
  X-Koordinate des ersten Punktes auf dem Sichtstrahl.
- ▷ **PY** (output_control) ..... real-array ⇝ *real*
  Y-Koordinate des ersten Punktes auf dem Sichtstrahl.
- ▷ **PZ** (output_control) ..... real-array ⇝ *real*
  Z-Koordinate des ersten Punktes auf dem Sichtstrahl.
- ▷ **QX** (output_control) ..... real-array ⇝ *real*
  X-Koordinate des zweiten Punktes auf dem Sichtstrahl.
- ▷ **QY** (output_control) ..... real-array ⇝ *real*
  Y-Koordinate des zweiten Punktes auf dem Sichtstrahl.
- ▷ **QZ** (output_control) ..... real-array ⇝ *real*
  Z-Koordinate des zweiten Punktes auf dem Sichtstrahl.

Beispiel

```
//  get internal 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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `get_line_of_sight` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`get_line_of_sight` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_cam_par`, `camera_calibration`

Mögliche Nachfolgerfunktionen

`affine_trans_point_3d`

Siehe auch

`camera_calibration`, `disp_caltab`, `read_cam_par`, `project_3d_point`, `affine_trans_point_3d`

Modul

Camera calibration

get_pose_type (: : Pose : OrderOfTransform, OrderOfRotation, ViewOfTransform)

Fragt Darstellungstyp von 3D-Lageparametern ab.

Mit get_pose_type wird der Darstellungstyp von 3D-Lageparametern Pose abgefragt.

Durch 3D-Lageparameter sind 3D-Transformationen definiert, die aus einer Translation und einer Rotation bestehen. Halcon unterstützt verschiedene Darstellungstypen von solchen Transformationen. Dies sind z.B. die äußeren Kameraparameter, gegeben durch den 3D-Translationsvektor (in Meter), die drei Werte der Rotationswinkel und die Codierung des Darstellungstyps der 3D-Transformation: Z.B. ist der Wert '0' der Code des Typs 1 und bedeutet, daß die Transformation eines Punktes beschrieben ist, hierbei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die neue Y-Achse und schließlich um die nach der zweiten Drehung entstandene X-Achse. Die Bedeutung der anderen Darstellungstypen der 3D-Transformation ist bei create_pose beschrieben.

Der mit ViewOfTransform zurückgegebene Wert besagt, ob die Transformations-Vorschrift die Transformation eines Punktes (**'point'**) oder die Transformation eines Koordinatensystems (**'coordinate_system'**) beschreibt. OrderOfTransform gibt an, ob bei der Transformation zuerst die Rotation (**'Rp+T'**) oder zuerst die Translation (**'R(p-T)'**) durchgeführt wird. Die Bedeutung der drei Rotationswerte wird mit OrderOfRotation angegeben. Hierbei bedeuten die Werte **'gba'** und **'abg'**, daß mit den Rotationswerten von Pose die Drehwinkel α (um die x-Achse), β (um die y-Achse) und γ (um die z-Achse) angegeben sind. Bei **'gba'** ist die Drehreihenfolge: γ , β , α und bei **'abg'**: α , β , γ . Ist OrderOfRotation = **'rodriguez'**, so werden die Rotationsparameter als Rodriguez-Rotationsvektor ausgewertet.

Parameter

- ▷ **Pose** (input_control) . . . . . pose-array ⇝ *real* / integer
  3D-Transformations-Vorschrift.
  Parameteranzahl : 7
- ▷ **OrderOfTransform** (output_control) . . . . . string ⇝ *string*
  Reihenfolge von Rotation und Translation.
- ▷ **OrderOfRotation** (output_control) . . . . . string ⇝ *string*
  Bedeutung der Rotationswerte.
- ▷ **ViewOfTransform** (output_control) . . . . . string ⇝ *string*
  Sichtweise der Transformation.

Beispiel

```
//  get pose (external camera parameters) */
read_pose ('campose.dat', CamPose)
//  get semantic type of pose
get_pose_type (CamPose, OrderT, OrderR, SightT).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert get_pose_type den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

get_pose_type ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration

Mögliche Nachfolgerfunktionen

convert_pose_type

Siehe auch

create_pose, convert_pose_type, write_pose, read_pose

Modul

Camera calibration

hand_eye_calibration (: : NX, NY, NZ, NRow, NCol, MPointsOfImage, MRelPoses, BaseStartPose, CamStartPose, CamParam, ToEstimate, StopCriterion, MaxIterations, MinError : BaseFinalPose, CamFinalPose, NumErrors)

Durchführen einer Hand-Auge-Kalibrierung.

Die Hand-Auge-Kalibrierung hand_eye_calibration dient zur Bestimmung des Versatzes zwischen dem Ursprung des Kamerakoordinatensystems und dem Bewegungsursprung bei einem aktiven Kamerasystem. Üblicherweise wird dies bei Robotern, die mit einer Kamera ausgestattet sind, verwendet. Hierbei muß der Versatz zwischen der Kamera (dem Auge) und dem Greifer (der Hand) bestimmt werden, damit Objekte videogestützt gegriffen werden können.

Der im folgenden beschriebene Fall geht davon aus, daß die Kamera fest am Endeffektor des Manipulators montiert ist (man spricht von hand-in-eye-configuration). In diesem Fall wird durch die Hand-Auge-Kalibrierung hauptsächlich der (konstante) Versatz zwischen Kamera- und Greiferkoordinationsystem bestimmt. Ist die Kamera dagegen stationär mit Blick auf den Manipulator angebracht, wird vor allem der Versatz zwischen Kamera- und Basiskoordinationsystem des Roboters gesucht. Dies kann ebenfalls durch Anwendung von hand_eye_calibration durchgeführt werden.

hand_eye_calibration arbeitet ähnlich wie die Kalibrierung der äußeren Kameraparameter (Kameralage), vgl. camera_calibration. Es wird hier jedoch nicht nur die Kameralage als einfache Abbildung bestimmt, sondern eine Abbildungskette vom Objektkoordinatensystem, über das Koordinatensystem in der Basis des aktiven Systems, über das Koordinatensystem in der Bewegungseinheit (Manipulator), bis hin zum Kamerakoordinatensystem bestimmt. Es ergibt sich somit für die Transformation eines Punktes P_W des Welt(Objekt)-Koordinatensystems in das Kamerakoordinatensystem:

$$\begin{aligned} P_C &= [x\ \ y\ \ z]^T = \mathcal{R} \cdot P_W + \mathcal{T} \\ &= \mathcal{R}_c \left(\mathcal{R}_m \left(\mathcal{R}_v \cdot P_W + \mathcal{T}_v\right) + \mathcal{T}_m\right) + \mathcal{T}_c \end{aligned}$$

mit:

$\mathcal{R}_v$, $\mathcal{T}_v$:	Rotation und Translation vom Welt(Objekt)-Koordinatensystem in das Basiskoordinatensystem des aktiven Systems (vehicle)
$\mathcal{R}_m$, $\mathcal{T}_m$:	Rotation und Translation vom Basiskoordinatensystem in das Koordinatensystem der Bewegungseinheit (manipulator)
$\mathcal{R}_c$, $\mathcal{T}_c$:	Rotation und Translation vom Manipulatorkoordinatensystem in das Kamerakoordinatensystem (camera)

Von der Hand-Auge-Kalibrierung soll ($\mathcal{R}_c$, $\mathcal{T}_c$) bestimmt werden. Um die Notwendigkeit eines exact vermessenen Aufbaus zu vermeiden wird ein Verfahren mit Fehlerausgleichsrechnung verwendet, bei dem ($\mathcal{R}_v$, $\mathcal{T}_v$) mit bestimmt wird. Daher werden Meßdaten verwendet, die sich auf k verschiedene Manipulatorpositionen beziehen. Voraussetzung ist daher die Möglichkeit zur Bestimmung der Positionen des Manipulators. Diese ergeben sich bei einem Roboterarm aus den Winkelpositionen der einzelnen Gelenke. Es wird vorausgesetzt, daß sich der Manipulator mit entsprechender Genauigkeit positionieren läßt. Analog zur Bestimmung der inneren Kameraparameter (vgl. camera_calibration) wird eine große Anzahl von verschiedenen Positionen verwendet, um die Robustheit des Verfahrens zu steigern.

Das Verfahren minimiert eine Fehlerfunktion nach dem Newton-Verfahren mit Hilfe von Normalengleichungen. Daher sind für ($\mathcal{R}_c$, $\mathcal{T}_c$) und ($\mathcal{R}_v$, $\mathcal{T}_v$) geeignete Startwerte anzugeben.

Startwerte für ($\mathcal{R}_c$, $\mathcal{T}_c$) ergeben sich durch grobe Vermessung des Aufbaus. Die Startwerte für ($\mathcal{R}_v$, $\mathcal{T}_v$) können aus den Startwerten für ($\mathcal{R}_c$, $\mathcal{T}_c$), den äußeren Kameraparametern ($\mathcal{R}_a$, $\mathcal{T}_a$) (Lage der Kamera in der gesamten Abbildungskette; vgl. camera_calibration) und der dazugehörigen Lage des Manipulators ($\mathcal{R}_m$, $\mathcal{T}_m$) bestimmt werden. Denn es gilt:

$$\begin{aligned} P_C &= \mathcal{R}_a \cdot P_W + \mathcal{T}_a \\ &= \mathcal{R}_c \left(\mathcal{R}_m \left(\mathcal{R}_v \cdot P_W + \mathcal{T}_v\right) + \mathcal{T}_m\right) + \mathcal{T}_c \end{aligned}$$

mit:

$\mathcal{R}_a$, $\mathcal{T}_a$:	Rotation und Translation vom Welt(Objekt)-Koordinatensystem in das Kamerkoordinatensystem (Kameralage).

Die folgenden Abschnitte befassen sich jeweils mit einzelnen Fragen, die sich bei der Verwendung von hand_eye_calibration stellen und sollen daher sowohl dem Verständnis als auch als Leitfaden für die eigene Anwendung dienen.

Wie erhalte ich die 3D-Modellpunkte? Als Grundlage für die Hand-Auge-Kalibrierung dienen 3D-Modellpunkte im Objektkoordinatensystem (NX, NY, NY), mit zugehörigen Abbildungen im Kamerakoordinatensystem (NRow, NCol). Um eine erfolgreiche Hand-Auge-Kalibrierung durchführen zu

können, müssen 3D-Modellpunkte angegeben werden, die aus Systemkonfigurationen mit genügend vielen verschiedenen Positionen des Bewegungssystems (Manipulators) ermittelt worden sind.

Es können beliebige bekannte Punkte im Objektkoordinatensystem verwendet werden. Man muß lediglich die Projektionen im Bild kennen. Idealerweise verwendet man jedoch einen Standardeichkörper, wie er mit `create_caltab` erzeugt werden kann. Damit können dann mit `find_caltab` und `find_marks_and_pose` die Positionen der Marken des Kalibrierkörpers im Bild extrahiert werden.

Für die Beschreibung zur Extraktion des Eichkörpers und der 3D-Modellpunkte sei auf Erklärungen bei `camera_calibration` verwiesen.

Mit dem Parameter `MPointsOfImage` wird angegeben, wieviele 3D-Modellpunkte für jede Lage des Bewegungssystems, also damit für jedes Bild, verwendet werden. Damit ist die Zuordnung der in `NX`, `NY`, `NY` linearisiert angegebenen 3D-Modellpunkte und den Projektionen (`NRow`, `NCol`) zu den entsprechenden Konfigurationen des Systems (Positionen der Bewegungseinheit; `MRelPoses`) möglich.

Die 3D-Modellpunkte können für den Standardeichkörper mit dem Operator `caltab_points` aus einer Eichkörperbeschreibungsdatei gelesen werden.

Wie nehme ich eine Menge von geeigneten Bildern auf? Ist ein planarer Standardeichkörper vorhanden, so kann folgendermaßen vorgegangen werden: Mit der zu kalibrierenden Kombination aus Basis des Bewegungssystems (Fahrzeug bei mobilen Systemen, als feststehender Bezugspunkt für Bewegungseinheit), der Bewegungseinheit (Manipulator, Roboterarm oder Schwenk-/Neigekopf) und dem Kamerasystem wird der Eichkörper aufgenommen, vgl. `open_framegrabber` bzw. `grab_image`. Dabei sollte folgendes berücksichtigt werden:

- Insgesamt sollten mindestens 10 bis 20 Bilder aus verschiedenen Aufnahmepositionen zur Verfügung stehen, bei denen die Position der Kamera zum Eichkörper verschieden ist. Die Position des Eichkörpers darf jedoch nicht verändert werden!
- Der Eichkörper muß jeweils komplett (inkl. Rand!) im Bild sein.
- Es sollten keine Reflexionen o.ä. auf dem Eichkörper sein.
- Innerhalb der Menge von Bildern sollte der Eichkörper jeweils aus den unterschiedlichsten Positionen der Bewegungseinheit aufgenommen werden. Der Eichkörper kann und soll hierbei an den verschiedensten Positionen im Bild erscheinen. Dabei sollte der Eichkörper dann jeweils leicht bis mittel verdreht um seine X- und/oder Y-Achse im Bild erscheinen, so daß die perspektivische Verzerrung der Eichkörpermarken gut sichtbar ist. Die jeweiligen äußeren Kameraparameter (Lage der Kamera bzgl. dem Eichkörper) sollen also viele verschiedene Werte annehmen! Bedenken Sie, daß die Bewegungseinheit nur 6 verschiedene Freiheitsgrade hat (3 x Rotation und 3 x Translation), daß aber 12 verschiedene Freiheitsgrade (`BaseFinalPose` und `CamFinalPose`) geschätzt werden sollen. D.h., daß der Raum der Bewegungsfreiheitsgrade der Bewegungseinheit bei den Aufnahmen nach Möglichkeit voll ausgenutzt werden muß.
- Der Eichkörper sollte jeweils mindestens etwa ein Viertel des gesamten Bildes ausfüllen, damit die Marken robust detektiert werden können.
- Die inneren Kameraparameter der zu verwendenden Kameraeinheit müssen zuvor bestimmt sein und werden mit `CamParam` angegeben, vgl. `camera_calibration`. Beachten Sie, daß Veränderungen der Bildgröße, der Brennweite, der Blende oder des Fokus eine Veränderung der inneren Kameraparameter bedeuten.
- Die Kameraeinheit darf zwischen den Aufnahmen nicht verändert werden, d.h. weder Brennweite und Blende, noch Fokussierung darf verändert werden, denn für alle Aufnahmen gelten die gleichen inneren Kameraparameter. Achten Sie darauf, daß bei einer Änderung der Entfernung der Kameraeinheit zum Eichkörper, durch die Bewegungseinheit, die Fokussierung (Schärfe im Bild) noch ausreicht. Sorgen Sie daher bei den Aufnahmen für eine gute Ausleuchtung der Kalibrierplatte, denn die Tiefenschärfe nimmt mit kleiner Blende zu.

Woher nehme ich geeignete Startwerte? Die Startwerte für $(\mathcal{R}_c, \mathcal{T}_c)$ und $(\mathcal{R}_v, \mathcal{T}_v)$ werden wie die äußeren Kameraparameter als Lage (eng. pose) mit den Parametern `CamStartPose` und `BaseStartPose` angegeben.

`CamStartPose` erhält man durch Vermessen der Lage der Kamera im Koordinatensystem der Bewegungseinheit (Manipulator). Es ist das gleiche Vorgehen, wie bei der Bestimmung des Startwerts für die Kalibrierung der äußeren Kameraparameter ohne den Standardeichkörper. Verwenden Sie hierzu `create_pose`. Bei der Vermessung ist es einfacher, zuerst die Translation des Ursprungs des Kamerakoordinatensystems im Koordinatensystem der Bewegungseinheit zu ermitteln und danach erst die Orientierung zu bestimmen. Die Translation ist durch eine Verschiebung in Richtung der drei Achsen (Translations-Parameter von `create_pose`) betimmt.

Das Kamerakoordinatensystem ist so orientiert, daß der Sichtstrahl der Kamera der positiven Z-Achse entspricht. Die Orientierung des Kamerakoordinatensystems relativ zum Weltkoordinatensystem kann durch drei Rotationswinkel angegeben werden. Zur Ermittlung der Orientierung 'dreht' man z.B. zunächst um die z-Achse des Koordinatensystems der Bewegungseinheit, dann um die neue y-Achse und schließlich um die nach der 2. Drehung entstandene x-Achse. Die Drehwinkel um die x-, y- und z-Achse entsprechen dann den Rotations-Parametern von `create_pose`. Entsprechend dieser Beschreibung der Lage sind bei `create_pose` als Sichtweise der Transformationsvorschrift **'coordinate_system'** (man beschreibt die Transformation eines Koordinatensystems), für die Reihenfolge der Transformationen **'R(p-T)'** (zuerst die Translation und dann erst die Rotation) und für die Reihenfolge der Rotation **'gba'** (γ , β , α) anzugeben. Dies entspricht dem Darstellungstyp 10. Für weitergehende Informationen sei hier auf die Beschreibung von `create_pose` verwiesen.

Zur Bestimmung von `BaseStartPose` sollte zunächst für eine der aufgenommenen Bilder die Kalibrierung der äußeren Kameraparameter durchgeführt werden, vgl. `camera_calibration`. Aus der so ermittelten Lage des Kamerakoordinatensystems im Welt(Objekt)-Koordinatensystems und den anderen Positionen (Lagen) der Abbildungskette (Startwert für ($\mathcal{R}_c$, $\mathcal{T}_c$) und exakte Werte für die zugehörige Position des Manipulators ($\mathcal{R}_m$, $\mathcal{T}_m$)), läßt sich der Startwert sehr elegant und einfach durch die Verwendung von homogenen 3D-Transformations-Matrizen bestimmen, denn es gilt:

$$\begin{aligned} P_C &= \mathcal{R} \cdot P_W + \mathcal{T} \\ &= \mathcal{H} \cdot P_W \end{aligned}$$

und:

$$\mathcal{H}_v = {\mathcal{H}_m}^{-1} \cdot {\mathcal{H}_{c(start)}}^{-1} \cdot \mathcal{H}_a$$

mit:

$\mathcal{H}_v$	:	Transformation zwischen Welt- und Basiskoordinatensystem (Startwert)
$\mathcal{H}_m$	:	Transformation zwischen Basis- und Koodinatensystem der Bewegungseinheit (Manipulator)
$\mathcal{H}_{c(start)}$	:	Transformation zwischen Koodinatensystem der Bewegungseinheit und dem Kamerakoordinatensystem (Startwert, vermessen)
$\mathcal{H}_a$	:	Transformation zwischen Welt- und Kamerakoordinatensystem (ermittelt durch Kalibrierung der äußeren Kameraparameter).

Vgl. hierzu `pose_to_hom_mat3d`, `affine_trans_point_3d`, `hom_mat3d_invert`, `hom_mat3d_compose` und `hom_mat3d_to_pose`.

Wie erhalte ich die Lagen der Bewegungseinheit? Mit dem Parameter `MRelPoses` werden die Positionen der Bewegungseinheit angegeben. Falls m Positionen angefahren wurden, so müssen m Positionen linearisiert, also hintereinander, angegeben werden. Die Darstellung dieser 3D-Lageparameter entspricht der, der äußeren Kameraparameter, wie auch der, der Startwerte `CamStartPose` und `BaseStartPose`. 3D-Lageparameter sind Tupel der Länge 7, vgl. `create_pose`.

Es ist äußerst wichtig für das Verfahren der Hand-Auge-Kalibrierung, daß diese Werte sich exakt aus dem Bewegungssystem ablesen oder ausmessen lassen. Die Umsetzung der abgelesenen Winkel eines Roboterarms in die Notation der Lagedarstellung (Roation und Translation), erfolgt mit `create_pose`. Es empfiehlt sich das gleiche Vorgehen, wie bei der Bestimmung des Startwerts `CamStartPose` für die Kameralage im Koordinatensystem der Bewegungseinheit. Das Manipulatorkoordinatensystem so orientiert sein, daß es bei der Nullage des Manipulators mit dem Koordinatensystem in der Basis übereinstimmt oder zumindest parallel liegt. Zur Bestimmung der Lagen der einzelnen Manipulatorstellung ermittelt man wieder zunächst die 3 Translationen zwischen dem Ursprung des Basiskoordinatensystem und dem Ursprung des Manipulatorkoordinatensystems. Die Translationen sind in Richtung der Achsen des Basiskoordinatensystems bei der Generierung der Lagewerte mit `create_pose` anzugeben. Bei der Orientierung der Koordinatensysteme zueinander 'dreht' man nun um die z-Achse des verschobenen Koordinatensystems, dann um die neue y-Achse und schließlich um die nach der 2. Drehung entstandene x-Achse. Die drei Rotationswinkel werden ebenfalls als Parameter bei `create_pose` verwendet. Enstsprechend der so ermittelten Lage sind zur Definition des Darstellungstypen der Lage bei `create_pose` noch **'coordinate_system'**, **'R(p-T)'** und **'gba'** anzugeben, dies entspricht dem Darstellungstypen 10, siehe auch die Erklärungen bei `create_pose`.

Wie kann man einzelnen Lagewerte von der Schätzung ausschließen? `hand_eye_calibration` schätzt maximal 12 Lagewerte. Dies sind jeweils 3 Rotations- und 3 Translationswerte für die Lage des Basiskoordinatensystems im Welt-(Objekt-)Koordinatensystem und für die Lage des Kamerakoordinatensystems im Koordinatensystem der Bewegungseinheit (Manipulator). Es gibt jedoch die Möglichkeit einzelne dieser

12 Lagewerte beim Schätzen auszuschließen. Dies bedeutet, daß die als Startewert angegebenen Lagewerte unverändert bleiben und als konstant für die Schätzung aller anderer Lagewerte angenommen werden. Mit dem Parameter `ToEstimate` gibt man an, welche Lagewerte geschätzt werden sollen. In `ToEstimate` werden hintereinander Schlüßelwörter für die zu schätzenden Lagewerte angegeben. Dies sind:

Für die Lage des Basiskoordinatensystems im Welt-(Objekt-)Koordinatensystem:

'baseTx'	=	Translation in x-Richtung
'baseTy'	=	Translation in y-Richtung
'baseTz'	=	Translation in z-Richtung
'baseRa'	=	Rotation um x-Achse
'baseRb'	=	Rotation um y-Achse
'baseRg'	=	Rotation um z-Achse

Für die Lage des Kamerakoordinatensystems im Koordinatensystem der Bewegungseinheit (Manipulator):

'camTx'	=	Translation in x-Richtung
'camTy'	=	Translation in y-Richtung
'camTz'	=	Translation in z-Richtung
'camRa'	=	Rotation um x-Achse
'camRb'	=	Rotation um y-Achse
'camRg'	=	Rotation um z-Achse

Zum Schätzen aller 12 Lagewerte können entweder alle 12 Schlüßelwörter hintereinander angegeben werden oder das Schlüßelwort **'all'**.

Das Festhalten einzelner Lagewerte ist zum einen sinnvoll, wenn man teilweise exakt vermessenen Werte hat und diese daher nicht mitschätzen lassen will. Zum anderen kann man durch das Fixieren und der damit begründeten Reduzierung der Anzahl der Freiheitsgrade, die Robustheit des Verfahrens steigern, wenn die Bewegungseinheit nur wenige Bewegungsmöglichkeiten hat, wie z.B. ein Schwenk-/Neigekopf.

Welche Abbruchkriterien gibt es für das Fehlerminimierungsverfahren? Das Verfahren bricht entweder nach einer bestimmten Anzahl von Iterationen ab oder wenn ein bestimmter minimaler Fehler unterschritten wird. Mit dem Parameter `StopCriterion` wählt man dies aus. Mit **'CountIterations'** bricht das Verfahren nach der mit `MaxIterations` angegebenen maximalen Anzahl von Iterationen ab.

Wenn für `StopCriterion` **'MinError'** angegeben wird, so läuft das Verfahren, bis der minimale Fehler, der mit `MinError` angegeben wird, unterschritten wird. Werden allerdings dabei die in `MaxIterations` angegebene Anzahl der maximalen Iterationen überschritten, so bricht das Verfahren mit einer entsprechenden Fehlermeldung ab.

Wie sieht die Reihenfolge der einzelnen Parameter aus? Die Länge des Tupels `MPointsOfImage` entspricht der Anzahl der verschiedenen Positionen der Bewegungseinheit (Manipulator). Mit `MPointsOfImage` wird angegeben wieviele Modellpunkte pro Position verwendet werden. Wird der Standardeichkörper verwendet, so sind dies 49 Punkte pro Position und Bild. Wurden 15 Aufnahmen gemacht so ist `MPointsOfImage` ein Tupel der Länge 15, wobei alle Einträge '49' sind.

Die dadurch festgelegte Anzahl von Kalibrierungsbildern muß auch bei den Tupeln mit den Koordinaten der 3D-Modellmarkenpunkte bzw. der extrahierten 2D-Markenpunkte berücksichtigt werden. Bei einer Anzahl von 15 Kalibrierungsbildern mit jeweils 49 Markenpunkte müssen daher die Tupel `NRow`, `NCol`, `NX`, `NY` und `NZ` $15 \cdot 49 = 735$ Werte enthalten. Die Reihenfolge der Werte erfolgt dabei bilderweise. D.h. die ersten 49 Werte korrespondieren zu den 49 Marken, die zum ersten Bild gehören. Die Reihenfolge der 3D-Markenpunkte und den 2D-Markenpunkten ist hierbei gleich.

Die Länge des Tupels `MRelPoses` korrespondiert zu der Anzahl der Kalibrierungsbilder, d.h. werden beispielsweise 15 Aufnahmen mit verschiedenen Positionen verwendet, so ergibt sich die Länge des Tupels `MRelPoses` zu 15*7=105 (15 mal je 7 Lageparameter). Die ersten 7 Werte entsprechen daher der Lage der Bewegungeseinheit bei der ersten Aufnahme, die nächsten 7 denen bei der zweiten Aufnahme usw. .

Was bedeuten die Ausgabeparameter? Wenn `StopCriterion` = **'CountIterations'** gewählt wurde, so werden auch wenn das Verfahren nicht konvergiert die Ausgabeparameter ausgegeben. Ist jedoch `StopCriterion` = **'MinError'** gewählt, so muß das Verfahren den minimalen Fehler unterschreiten, um Ausgabeparameter auszugeben.

Die Ausgabewerte sind die Lagewerte für die Lage des Basiskoordinatensystems im Welt-(Objekt-)Koordinatensystem, `BaseFinalPose` und für die Lage des Kamerakoordinatensystems im Koordinatensystem des Bewegungseinheit, dem Hand-Auge-Versatz, `CamFinalPose`.

Der Darstellungstyp von `BaseFinalPose` und `CamFinalPose` entspricht den Darstellungstypen der entsprechenden Startwerte. Der Darstellungstyp kann jedoch mit `convert_pose_type` gewandelt werden.

Die Beschreibung der verschiedenen Darstellungstypen und deren Umwandlung ist bei `create_pose` zu finden.

In `NumErrors` wird eine Liste (als Tupel) von Fehlern jeder Iteration zurückgegeben. Anhand des Fehlerverlaufs kann erkannt werden, ob das Verahren für die angegebenen Startwerte konvergiert. Die Fehlerwerte sind 3D-Abweichungen in [m]. Der letzte Eintrag in der Fehlerliste entspricht damit einer Abschätzung der Genauigkeit für die zurückgegebenen Lagewerte.

Achtung

Die Güte der Kalibrierung ist von der Genauigkeit der Eingabeparameter (Positionen der Kalibrierpunkte und Startwerte für `BaseStartPose`, `CamStartPose`) abhängig. Anhand der zurückgegebenen Fehlermaße `NumErrors` kann erkannt werden, ob das Verfahren konvergiert. Ebenfalls kann die Genauigkeit abgeschätzt werden. Die Fehlermaße sind 3D-Abweichungen in [m].

Parameter

- ▷ **`NX`** (input_control) . . . . . real-array ⇝ *real*
  Linearisierte Liste, die alle X-Koordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **`NY`** (input_control) . . . . . real-array ⇝ *real*
  Linearisierte Liste, die alle Y-Koordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **`NZ`** (input_control) . . . . . real-array ⇝ *real*
  Linearisierte Liste, die alle Z-Koordinaten der Kalibrierpunkte enthält (in der Bildreihenfolge).
- ▷ **`NRow`** (input_control) . . . . . real-array ⇝ *real*
  Linearisierte Liste, die alle Markenabszissen enthält (in der Bildreihenfolge).
- ▷ **`NCol`** (input_control) . . . . . real-array ⇝ *real*
  Linearisierte Liste, die alle Markenordinaten enthält (in der Bildreihenfolge).
- ▷ **`MPointsOfImage`** (input_control) . . . . . integer-array ⇝ *integer*
  Für jedes Bild: Anzahl der Kalibrierpunkte.
- ▷ **`MRelPoses`** (input_control) . . . . . pose-array ⇝ *real* / integer
  Zu jedem Bild die gemessenen Werte für die relative Lage des Greifers.
- ▷ **`BaseStartPose`** (input_control) . . . . . pose-array ⇝ *real* / integer
  Startwert fuer Lage des Roboters im WKS.
- ▷ **`CamStartPose`** (input_control) . . . . . pose-array ⇝ *real* / integer
  Startwert fuer Lage der Kamera zum Greifer.
- ▷ **`CamParam`** (input_control) . . . . . number-array ⇝ *real* / integer
  Innere Kameraparameter.
- ▷ **`ToEstimate`** (input_control) . . . . . string-array ⇝ *string*
  Zu schaetzende Parameter (max. 12 Freiheitsgrade).
  Defaultwert : ”all”
  Werteliste : `ToEstimate` ∈ {”all”, ”baseTx”, ”baseTy”, ”baseTz”, ”baseRa”, ”baseRb”, ”baseRg”, ”camTx”, ”camTy”, ”camTz”, ”camRa”, ”camRb”, ”camRg”}
- ▷ **`StopCriterion`** (input_control) . . . . . string ⇝ *string*
  Art des Abbruchkriteriums.
  Defaultwert : ”CountIterations”
  Werteliste : `StopCriterion` ∈ {”CountIterations”, ”MinError”}
- ▷ **`MaxIterations`** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der maximal auszuführenden Iterationen.
  Defaultwert : 15
  Wertevorschläge : `MaxIterations` ∈ {10, 15, 20, 25, 30}
- ▷ **`MinError`** (input_control) . . . . . real ⇝ *real*
  Zu unterschreitender minimaler Fehler als Abbruchkriterium.
  Defaultwert : 0.0005
  Wertevorschläge : `MinError` ∈ {0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1}
- ▷ **`BaseFinalPose`** (output_control) . . . . . pose-array ⇝ *real* / integer
  Berechnete Lage des Basissystems zum Weltkoordinatensystem.
- ▷ **`CamFinalPose`** (output_control) . . . . . pose-array ⇝ *real* / integer
  Berechnete Lage der Kamera zum Greiferkoordinatensystem.
- ▷ **`NumErrors`** (output_control) . . . . . real(-array) ⇝ *real*
  Fehlermaße fuer jede Iteration.

Beispiel

```
HTuple    CamParam, RCoord, CCoord;
HTuple    ManuPose, NumMarker, X, Y, Z, XPositions, YPositions;
HTuple    ZPositions, RCoordTmp, CCoordTmp, StartPose;
HTuple    ManuPoseTmp, BaseStartPose, CamStartPose, BaseFinalPose;
HTuple    CamFinalPose, NumErrors;
Hobject   Image, Caltab;
char      Datname[256];
char      CaltabName[256];

sprintf(CaltabName, "Caltab.descr");
::read_cam_par("CamParam.cal",&CamParam);
RCoord = HTuple();
CCoord = HTuple();
ManuPose = HTuple();
NumMarker = HTuple();
::caltab_points(CaltabName,&X,&Y,&Z);
XPositions = HTuple();
YPositions = HTuple();
ZPositions = HTuple();
// find landmarks in every image
for (long i=0; i<=22; i++)
{
  sprintf(Datname,"Image.%03ld.tiff",i);
  HImage Image(Datname);
  HRegion Caltab = Image.FindCaltab(CaltabName,3,150,5);
  RCoordTmp = Image.FindMarksAndPose(Caltab,CaltabName,CamParam,128,10,
                                     &CCoordTmp,&StartPose);
  RCoord = ((HTuple)RCoord).Concat(RCoordTmp);
  CCoord = ((HTuple)CCoord).Concat(CCoordTmp);
  sprintf(Datname,"Pose.%03ld.cal",i);
  ::read_pose(Datname,&ManuPoseTmp);
  ManuPose = ((HTuple)ManuPose).Concat(ManuPoseTmp);
  XPositions = ((HTuple)XPositions).Concat(X);
  YPositions = ((HTuple)YPositions).Concat(Y);
  ZPositions = ((HTuple)ZPositions).Concat(Z);
  NumMarker = ((HTuple)NumMarker).Concat(RCoordTmp.Num());
}
// read start poses
::read_pose("BaseStartPose.cal",&BaseStartPose);
::read_pose("CameraStartPose.cal",&CamStartPose);
// H A N D - E Y E - C A L I B R A T I O N
::hand_eye_calibration(XPositions,YPositions,ZPositions,RCoord,CCoord,
                       NumMarker,ManuPose,BaseStartPose,CamStartPose,
                       CamParam,"all","CountIterations",20,0.000670,
                       &BaseFinalPose,&CamFinalPose,&NumErrors);
// save pose of difference between 'hand' and 'eye'
::write_pose(CamFinalPose,"CameraFinalPose.cal");
```

Ergebnis

Sind die Parameterwerte korrekt und konvergiert das Verfahren gegen den angegebenen minimalen Fehler (bei `StopCriterion` = **'MinError'**), dann liefert `hand_eye_calibration` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`hand_eye_calibration` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`find_marks_and_pose`

Mögliche Nachfolgerfunktionen

write_pose, convert_pose_type, pose_to_hom_mat3d, disp_caltab, sim_caltab

Siehe auch

find_caltab, find_marks_and_pose, disp_caltab, sim_caltab, write_cam_par, read_cam_par, create_pose, convert_pose_type, write_pose, read_pose, pose_to_hom_mat3d, hom_mat3d_to_pose, caltab_points, create_caltab

Modul

Camera calibration

hom_mat3d_to_pose (: : HomMat3D : Pose)

Konvertierung einer homogenen Transformations-Matrix in 3D-Lageparameter.

hom_mat3d_to_pose dient zum Konvertieren einer homogenen Transformations-Matrix in die korrespondierende Darstellung der Abbildung als Translationsvektor und als Rotationswinkel. Die Angabe der homogenen 4x3 Transformations-Matrix im Tupel HomMat3D erfolgt zeilenweise, d.h. die ersten 3 Werte entsprechen der ersten Zeile der Rotationsmatrix, danach folgt der X-Wert des Translationsvektors. Dann folgen die drei Werte der zweiten Zeile der Rotationsmatrix und der Y-Wert des Translationsvektors. Schließlich folgen entsprechend die Werte der dritten Zeile der Rotationsmatrix und der Z-Wert des Translationsvektors, vgl. auch affine_trans_point_3d.

Das Ausgabetupel Pose beschreibt 3D-Lageparameter und definiert somit eine 3D-Transformation. Dies sind z.B. die äußeren Kameraparameter, gegeben durch den 3D-Translationsvektor (in Meter), die drei Werte der Rotationswinkel und die Codierung des Darstellungstyps der 3D-Transformation. Der Darstellungstyp eines mit hom_mat3d_to_pose erzeugten 3D-Lageparameter hat immer den Typ 1 mit Code '0'. Dies bedeutet, daß die Transformation eines Punktes beschrieben ist, hierbei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die neue Y-Achse und schließlich um die nach der zweiten Drehung entstandene X-Achse. Dies entspricht der Anwendung der homogenen Transformations-Matrizen, daher gilt:

$$\begin{aligned} P_C = [x \;\; y \;\; z]^T &= \mathcal{H} \cdot P_W \\ &= \mathcal{R}^{(1)} \cdot P_W + \mathcal{T}^{(1)} \end{aligned}$$

Die Bedeutung der anderen Darstellungstypen für die 3D-Lageparameter ist bei create_pose beschrieben. Mit convert_pose_type läßt sich Pose in 3D-Lageparemeter eines anderen Typen umwandeln.

Im angegebenen Programmbeispiel ist gezeigt, wie man die initiale Kameralage für einen anschließenden Aufruf von camera_calibration bestimmen kann, wenn man die Kamera grob relativ zum Weltkoordinatensystem vermißt. Dies ist insbesondere von Interesse, falls nicht der planare Standardeichkörper (vgl. create_caltab) verwendet wird und somit der Operator find_marks_and_pose nicht angewandt werden kann. Stattdessen können natürliche oder künstliche Landmarken mit bekannter Weltposition verwendet werden.

Sei beispielsweise die relative Kameraposition vom Ursprung des Weltkörperkoordinatensystems etwa die folgende: x = 1.08 m, y = 0.25 m, z = 0.62 m. Das heißt, daß der Ursprung des Kamerakoordinatensystems im Weltkoordinatensystem die Koordinaten [1.08, 0.25, 0.62] besitzt. Das Kamerakoordinatensystem ist so orientiert, daß der Sichtstrahl der Kamera der positiven Z-Achse entspricht. Die Orientierung des Kamerakoordinatensystems relativ zum Weltkoordinatensystem kann durch drei Rotationswinkel angegeben werden. Im Beispiel wird das Kamerakoordinatensystem erst um 100 Grad um die z-Achse des Weltkoordinatensystems gedreht und anschließend um -120 Grad um die neue (!!!) x-Achse gedreht. Hinweis: Bei den beiden Aufrufen von hom_mat3d_rotate erfolgt daher die Angabe des jeweiligen Winkels negiert, da sich die Beschreibung der Transformationen von 3D-Punkten und die Beschreibung der Transformationen von Koordinatensystemen in den Vorzeichen der Rotationswinkel unterscheiden. Vgl. hierzu die Erläuterungen und das Beispiel bei create_pose.

Parameter

▷ **HomMat3D** (input_control) ..... affine3d-array ⇝ *real*
Homogene Transformations-Matrix.
Parameteranzahl : 12

▷ **Pose** (output_control) . . . . . pose-array ⇝ *real* / integer
Äußere Kameraparameter.
Parameteranzahl : 7

Beispiel

```
//  read internal camera parameters:
read_cam_par('campar.dat',CamParam)
//  read 3D world points [WorldPointsX,WorldPointsY,WorldPointsZ]
//  extract correspondent 2D image points [PixelsRow,PixelsColumn]
//  get rough camera pose from relative camera pose:
hom_mat3d_identity(HomMat3DIdent)
hom_mat3d_translate(HomMat3DIdent,-1.08,-0.25,-0.62,HomMat3DTrans)
hom_mat3d_rotate(HomMat3DTrans,-100*3.14159/180,'z',0,0,0,
                 HomMat3DRotZ)
hom_mat3d_rotate(HomMat3DRotZ,+120*3.14159/180,'x',0,0,0,
                 HomMat3DWorldStart)
//  convert transformation matrix to pose (rotation and translation)
hom_mat3d_to_pose(HomMat3DWorldStart,StartPose)
//  calibration of external camera params:
camera_calibration(WorldPointsX,WorldPointsY,WorldPointsZ,
                   PixelsRow,PixelsColumn,CamParam,StartPose,6,
                   Dummy,FinalPose,Errors).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert hom_mat3d_to_pose den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

hom_mat3d_to_pose ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

hom_mat3d_rotate, hom_mat3d_translate, hom_mat3d_invert

Mögliche Nachfolgerfunktionen

camera_calibration, write_pose, disp_caltab, sim_caltab

Siehe auch

create_pose, camera_calibration, disp_caltab, sim_caltab, write_pose, read_pose, pose_to_hom_mat3d, project_3d_point, get_line_of_sight, hom_mat3d_rotate, hom_mat3d_translate, hom_mat3d_invert, affine_trans_point_3d

Modul

Camera calibration

image_points_to_world_plane (: : CamParam, CamPose, Rows, Cols, Scale : X, Y)

Transformiert Bildpunkte in eine Ebene des Weltkoordinatensystems, die durch die externen Kameraparameter gegeben ist.

image_points_to_world_plane transformiert Bildpunkte, die in Rows und Cols gegeben sind, in eine Ebene im Weltkoordinatensystem. Letztere ist entweder identisch mit der Ebene der Kalibrierplatte oder – nach Korrektur der Plattendicke mit dem Operator set_origin_pose – parallel zu dieser. Dabei wird zunäechst aus den inneren Kameraparametern (CamParam) der Sehstrahl vom Projektionszentrum der Kamera zum Bildpunkt unter Berücksichtigung der radialen Verzeichnung im System der Kamera berechnet. Anschließend wird dieser Sehstrahl mit den äußeren Kameraparametern (CamPose) in das System der Ebene transformiert. Der Schnittpunkt von Ebene (Z=0) und Sehstrahl entspricht dann den Pseudo-3D-Koordinaten X und Y. Die erhaltenen Koordinaten können abschließend mit dem Parameter Scale beliebig skaliert werden. Dies ist insbesondere dann sinnvoll, wenn die Koordinaten in einem Bild dargestellt werden sollen. Der Parameter Scale ist dann in $[Einheit/Pixel]$ anzugeben. Die ursprüngliche Einheit ist durch die Koordinaten der Kalibrierpunkte gegeben. In vielen Fällen werden letztere in der Einheit 'Meter' in die Kalibrierung eingeführt, woraus folgt, daß ein Meter

einem Pixel im Bild entspricht. In diesem Fall ist es auch möglich, die Pixelgröße direkt durch **'m'**, **'cm'**, **'mm'** oder **'μm'** anzugeben.

Parameter

- ▷ **CamParam** (input_control) . . . . . number-array ⇝ *real* / integer
  Innere Kameraparameter.
  Parameteranzahl : 8
- ▷ **CamPose** (input_control) . . . . . pose-array ⇝ *real* / integer
  Äußere Kameraparameter
  Parameteranzahl : 7
- ▷ **Rows** (input_control) . . . . . coordinates.y-array ⇝ *real* / integer
  Zeilenindizes der zu transfomierenden Punkte.
  Defaultwert : 100.0
- ▷ **Cols** (input_control) . . . . . coordinates.x-array ⇝ *real* / integer
  Spaltenindizes der zu transfomierenden Punkte.
  Defaultwert : 100.0
- ▷ **Scale** (input_control) . . . . . number ⇝ *string* / integer / real
  Maßstab oder Dimension.
  Defaultwert : 'm'
  Wertevorschläge : Scale ∈ {'m', 'cm', 'mm', 'microns', 'μm', 1.0, 0.01, 0.001, '1e-6', 0.0254, 0.3048, 0.9144}
- ▷ **X** (output_control) . . . . . coordinates.x-array ⇝ *real*
  x-Koordinate im Weltkoordinatensystem.
- ▷ **Y** (output_control) . . . . . coordinates.y-array ⇝ *real*
  y-Koordinate im Weltkoordinatensystem.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert image_points_to_world_plane den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

image_points_to_world_plane ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_pose, hom_mat3d_to_pose, camera_calibration, hand_eye_calibration, set_origin_pose

Siehe auch

contour_to_world_plane_xld

Modul

Camera calibration

pose_to_hom_mat3d (: : Pose : HomMat3D)

Konvertierung von 3D-Lageparametern in eine homogene Transformations-Matrix.

pose_to_hom_mat3d dient zum Konvertieren von 3D-Lageparameter, wie den äußeren Kameraparameter Pose in eine äquivalente homogene 4x3 Transformations-Matrix HomMat3D.

Diese Transformations-Matrix besteht aus einer 3x3 Rotationsmatrix und dem Translationsvektor. Die Angabe der Werte erfolgt dabei zeilenweise, d.h. die ersten 3 Werte entsprechen der ersten Zeile der Rotationsmatrix, danach folgt der X-Wert des Translationsvektors. Dann folgen die drei Werte der zweiten Zeile der Rotationsmatrix und der Y-Wert des Translationsvektors. Schließlich folgen entsprechend die Werte der dritten Zeile der Rotationsmatrix und der Z-Wert des Translationsvektors, vgl. auch affine_trans_point_3d.

Das Eingabetupel Pose beschreibt 3D-Lageparameter und definiert somit eine 3D-Transformation. Dies sind z.B. die äußeren Kameraparameter, gegeben durch den 3D-Translationsvektor (in Meter), die drei Werte der Rotationswinkel und die Codierung des Darstellungstyps der 3D-Transformation: Z.B. ist der Wert '0' der Code des Typs 1 und bedeutet, daß die Transformation eines Punktes beschrieben ist, hierbei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die neue Y-Achse und schließlich

um die nach der zweiten Drehung entstandene X-Achse. pose_to_hom_mat3d verarbeitet jedoch alle verschieden Darstellungstypen für Pose. Die Bedeutung der anderen Darstellungstypen der 3D-Transformation ist bei create_pose beschrieben. Die 3D-Transformations-Vorschrift vom Typ 1 entspricht der 3D-Transformation mit homogenen Transformationsmatrizen, es gilt daher:

$$\begin{aligned} P_C = [x \;\; y \;\; z]^T &= \mathcal{R}^{(1)} \cdot P_W + \mathcal{T}^{(1)} \\ &= \mathcal{H} \cdot P_W \end{aligned}$$

Parameter

- ▷ **Pose** (input_control) . . . . . pose-array ⇝ *real* / integer
  Äußere Kameraparameter.
  Parameteranzahl : 7
- ▷ **HomMat3D** (output_control) . . . . . affine3d-array ⇝ *real*
  Homogene Transformations-Matrix.
  Parameteranzahl : 12

Beispiel

```
//  read internal camera parameters
read_cam_par('campar.dat'CamParam)
//  read initial camera pose
read_pose('campose.initial',StartCamPose)
//  read 3D world points [WorldPointsX,WorldPointsY,WorldPointsZ]
//  extract correspondent 2D image points [PixelsRow,PixelsColumn]
//  calibration of external camera parameters:
camera_calibration(WorldPointsX,WorldPointsY,WorldPointsZ,
                   PixelsRow,PixelsColumn,CamParam,StartCamPose,6,
                   Dummy,FinalCamPose,Errors)
//  transform FinalCamPose to 4x3 transformation matrix
pose_to_hom_mat3d(FinalCamPose,HomMat3DWorldCam).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert pose_to_hom_mat3d den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

pose_to_hom_mat3d ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

camera_calibration, read_pose

Mögliche Nachfolgerfunktionen

affine_trans_point_3d, hom_mat3d_invert, hom_mat3d_translate, hom_mat3d_rotate, hom_mat3d_to_pose

Siehe auch

create_pose, camera_calibration, write_pose, read_pose, hom_mat3d_to_pose, project_3d_point, get_line_of_sight, hom_mat3d_rotate, hom_mat3d_translate, hom_mat3d_invert, affine_trans_point_3d

Modul

Camera calibration

project_3d_point (: : X, Y, Z, CamParam : Row, Column)

Projektion von 3D-Punkten in Rechnerkoordinaten (Subpixel).

project_3d_point dient zur Projektion von einem oder mehreren 3D-Punkten mit den Koordinaten X, Y und Z in das Rechnerkoordinatensystem (in Pixel). Die Koordinaten X, Y und Z sind Kamerakoordinaten, entsprechen also der Position eines Punktes relativ zur Kamera.

Die inneren Kameraparameter `CamParam` (Focus, Sx, Sy, Cx, Cy, Kappa (κ), ImageWidth und ImageHeight) beschreiben dabei die Abbildungseigenschaften der Kamera. Als Kameramodell wird das einer **Lochkamera mit radialer Verzerrung** verwendet, falls die Kammerkonstante in `CamParam` mit einem Wert größer als 0 übergeben wird. Es beschreibt die Abbildung eines 3D-Punktes P_C in ein (Sub-)Pixel [r,c] des Videobildes durch folgende Transformationen:

$$\begin{aligned} P_C &= [x \;\; y \;\; z]^T \\ u = \text{Focus} \cdot \tfrac{x}{z} \;\; &\text{bzw.} \;\; v = \text{Focus} \cdot \tfrac{y}{z} \\ \tilde{u} = \frac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \;\; &\text{bzw.} \;\; \tilde{v} = \frac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}} \\ c = \tfrac{\tilde{u}}{S_x} + C_x \;\; &\text{bzw.} \;\; r = \tfrac{\tilde{v}}{S_y} + C_y \end{aligned}$$

Falls die Kammerkonstante in `CamParam` als 0 übergeben wird, wird als Kameramodell eine **Telezentrische Kamera mit radialer Verzerrung** verwendet, d.h. es wird angenommen, daß die Optik des Kameraobjektives eine Parallelprojektion der Weltpunkte durchführt. Die entsprechenden Gleichungen sind dann:

$$\begin{aligned} P_C = [x \;\; y \;\; z]^T &= \mathcal{R} \cdot P_W + \mathcal{T} \\ u = x \;\; &\text{bzw.} \;\; v = y \\ \tilde{u} = \frac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \;\; &\text{bzw.} \;\; \tilde{v} = \frac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}} \\ c = \tfrac{\tilde{u}}{S_x} + C_x \;\; &\text{bzw.} \;\; r = \tfrac{\tilde{v}}{S_y} + C_y \end{aligned}$$

Diese Gleichungen umfassen eine Koordinatentransformation vom Weltkoordinatensystem ins Kamerakoordinatensystem, eine perspektivische bzw. parallele Projektion in die Bildebene, eine radiale Verzerrung dieses projizierten Punktes und schließlich eine Diskretisierung und Hauptpunktverschiebung.

Parameter

- ▷ **X** (input_control) ..... real-array ⇝ *real*
  X-Koordinaten der zu projizierenden 3D-Punkte.
- ▷ **Y** (input_control) ..... real-array ⇝ *real*
  Y-Koordinaten der zu projizierenden 3D-Punkte.
- ▷ **Z** (input_control) ..... real-array ⇝ *real*
  Z-Koordinaten der zu projizierenden 3D-Punkte.
- ▷ **CamParam** (input_control) ..... number-array ⇝ *real* / integer
  Innere Kameraparameter.
  Parameteranzahl : 8
- ▷ **Row** (output_control) ..... real-array ⇝ *real*
  Zeilennummern der Pixel.
  Defaultwert : 'ProjectedRow'
- ▷ **Column** (output_control) ..... real-array ⇝ *real*
  Spaltennummern der Pixel.
  Defaultwert : 'ProjectedCol'

Beispiel

```
//  read camera pose
read_pose(campose.dat',CamPose)
//  transform camera pose to transformation matrix
pose_to_hom_mat3d(CamPose,HomMat3D)
//  transform 3D points from source into destination coordinate system
affine_trans_point_3d([3.0,3.2],[4.5,4.5],[5.8,6.2],HomMat3D,X,Y,Z)
//  read internal camera parameters
read_cam_par('campar.dat',CamParam)
// project 3D points into image
project_3d_point(X,Y,Z,CamParam,Row,Column).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `project_3d_point` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`project_3d_point` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`read_cam_par`, `affine_trans_point_3d`

Mögliche Nachfolgerfunktionen

`gen_region_points`, `gen_region_polygon`, `disp_polygon`

Siehe auch

`camera_calibration`, `disp_caltab`, `read_cam_par`, `get_line_of_sight`, `affine_trans_point_3d`

Modul

Camera calibration

read_cam_par (: : CamParFile : CamParam)

Lesen der inneren Kameraparameter aus Textdatei.

`read_cam_par` dient zum Einlesen der inneren Kameraparameter `CamParam` aus einer Textdatei mit dem Namen `CamParFile`.

Das Format der Textdatei ist eine (von Halcon unabhängige) generische Parameterbeschreibung, die beliebige Gruppen von Parametern zu hierarchisch organisierten Parametergruppen (Struktur ParGroup) zusammenfasst. Die Beschreibung eines Parameters innerhalb einer ParGroup enthält in 3 Zeilen folgende Einträge:

```
Name : Kurzname : aktueller Wert ;
 Typ : Untere Grenze (optional) : Obere Grenze (optional) ;
 Beschreibung (optional) ;
```

Der Halcon-Operator `read_cam_par` erwartet in der Datei `CamParFile` die Parametergruppe Camera: Parameter, in der die 8 Parameter Focus, Sx, Sy, Cx, Cy, Kappa (κ), ImageWidth und ImageHeight enthalten sind. Kommentare werden durch ein ’#’ am Zeilenanfang markiert. Eine solche Datei hat z.B. folgendes Aussehen:

```
# INTERNAL CAMERA PARAMETERS

ParGroup: Camera: Parameter;
    "Internal CCD-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 CCD-chip [meter]";

Sy:sy:  1.1e-05;
    DOUBLE:0.0:;
    "Height of a cell on the CCD-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]";
```

Als Kameramodell wird das einer **Loch- oder telezentrischen Kamera mit radialer Verzerrung** verwendet. Eine detaillierte Beschreibung dieser Kameramodelle steht bei der Beschreibung von write_cam_par.

Parameter

▷ **CamParFile** (input_control) ..... string ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert : 'campar.dat'
Werteliste : CamParFile ∈ {'campar.dat', 'campar.initial', 'campar.final'}

▷ **CamParam** (output_control) ..... number-array ⇝ *real* / integer
Innere Kameraparameter.
Parameteranzahl : 8

Beispiel

```
//  get internal camera parameters:
read_cam_par('campar.dat',CamParam).
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich gelesen werden, dann liefert read_cam_par den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

read_cam_par ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

find_marks_and_pose, sim_caltab, create_caltab, disp_caltab, camera_calibration

Siehe auch

find_caltab, find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab, write_cam_par, write_pose, read_pose, project_3d_point, get_line_of_sight

Modul

Camera calibration

read_pose (: : PoseFile : Pose)

Einlesen von 3D-Lageparametern aus einer Textdatei.

read_pose dient zum Einlesen der 3D-Lageparametern Pose aus einer Textdatei mit dem Namen PoseFile.

Das Ausgangstupel Pose beschreibt 3D-Lageparameter und definiert somit eine 3D-Transformation. Dies sind z.B. die äußeren Kameraparameter, gegeben durch den 3D-Translationsvektor (in Meter), die drei Werte der Rotationswinkel und die Codierung des Darstellungstyps der 3D-Transformation: Z.B. ist der Wert '0' der Code des Typs 1 und bedeutet, daß die Transformation eines Punktes beschrieben ist, hierbei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die neue Y-Achse und schließlich um die nach der zweiten Drehung entstandene X-Achse. Die Bedeutung der anderen Darstellungstypen für die 3D-Transformation ist bei create_pose beschrieben.

Eine solche Datei kann durch den Operator write_pose erzeugt werden und hat z.B. folgendes Aussehen:

```
# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
f 0

# Rotation angles [deg] or Rodriguez-vector:
r -17.8134 1.83816 0.288092

# Translational vector (x y z [m]):
t 0.280164 0.150644 1.7554
```

Parameter

▷ **PoseFile** (input_control) ..... filename ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert : 'campose.dat'
Werteliste : PoseFile ∈ {'campose.dat', 'campose.initial', 'campose.final'}

▷ **Pose** (output_control) ..... pose-array ⇝ *real* / integer
Äußeren Kameraparameter.
Parameteranzahl : 7

Beispiel

```
// get pose (external camera parameters):
read_pose('campose.dat',CamPose).
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich gelesen werden, dann liefert read_pose den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_pose ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_cam_par

Mögliche Nachfolgerfunktionen

pose_to_hom_mat3d, camera_calibration, disp_caltab, sim_caltab

Siehe auch

create_pose, find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab, write_pose, pose_to_hom_mat3d, hom_mat3d_to_pose

Modul

Camera calibration

set_origin_pose (: : PoseIn, DX, DY, DZ : PoseNewOrigin)

Verschiebt den Ursprung der externen Kameraparameter.

set_origin_pose verschiebt den Ursprung der externen Kameraparameter um den Vektor, der durch die Parameter DX, DY und DZ festgelegt ist. Dies kann z.B. dazu genutzt werden, die durch die Kamerakalibrierung erhaltenen externen Kameraparameter PoseIn um die Dicke der Kalibrierplatte zu korrigieren. Dazu muß für den Verschiebungsvektor (0,0,D) gewählt werden, wobei D der Dicke der Kalibrierplatte entspricht. Die dadurch erhaltenen neuen Kameraparameter PoseNewOrigin entsprechen somit dem Ergebnis einer Kalibrierung mit unendlich dünner Kalibrierplatte. Mit Hilfe von set_origin_pose ist es demnach z.B. mit dem Operatoren image_points_to_world_plane oder contour_to_world_plane_xld möglich, Messungen im Bild direkt auf die Meßebene (anstatt auf die Ebene der Kalibrierplatte) zu beziehen. In diesem Zusammenhang ist es außerdem realisierbar, negative Koordinaten im Weltkoordinatensystem zu vermeiden, indem zusätzlich Werte füer DX und DY angegeben werden.

Parameter

- ▷ **PoseIn** (input_control) . . . . . pose-array ⇝ *real* / integer
  Ursprüngliche 3D-Transformations-Vorschrift.
  Parameteranzahl : 7
- ▷ **DX** (input_control) . . . . . real ⇝ *real*
  Verschiebung des Ursprungs in x-Richtung.
  Defaultwert : 0
- ▷ **DY** (input_control) . . . . . real ⇝ *real*
  Verschiebung des Ursprungs in y-Richtung.
  Defaultwert : 0
- ▷ **DZ** (input_control) . . . . . real ⇝ *real*
  Verschiebung des Ursprungs in z-Richtung.
  Defaultwert : 0
- ▷ **PoseNewOrigin** (output_control) . . . . . pose-array ⇝ *real* / integer
  Neue 3D-Beschreibung nach Verschiebung des Ursprungs.
  Parameteranzahl : 7

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `set_origin_pose` den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`set_origin_pose` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`create_pose`, `hom_mat3d_to_pose`, `camera_calibration`, `hand_eye_calibration`

Mögliche Nachfolgerfunktionen

`write_pose`, `pose_to_hom_mat3d`, `image_points_to_world_plane`, `contour_to_world_plane_xld`

Modul

Camera calibration

sim_caltab (: SimImage : CalTabDescrFile, CamParam, CamPose, GrayBackground, GrayCaltab, GrayMarks, ScaleFac :)

Simulation eines Videobildes mit Eichkörper.

`sim_caltab` dient der Simulation eines Kalibrierungsbildes. Dazu wird die Eichkörperbeschreibung aus der Datei `CalTabDescrFile` eingelesen und die darin enthaltende 3D-Beschreibung mit Hilfe der angegebenen Kameraparameter (innere Kameraparameter `CamParam`, äußere Kameraparameter `CamPose`) auf die Bildebene projiziert (vgl. `project_3d_point`).

In dem simulierten Bild ist nur der Eichkörper abgebildet. Der Bildhintergrund wird mit dem Grauwert `GrayBackground`, der Hintergrund des Eichkörpers mit `GrayCaltab` und die Eichkörpermarken werden mit `GrayMarks` belegt. Der Parameter `ScaleFac` beeinflußt die Anzahl der Stützstellen, mit der die ellipsenförmigen Konturen der Eichkörpermarken approximiert werden (vgl. `disp_caltab`). Die Erhöhung der Stützstellenanzahl bewirkt daher eine genauere Bestimmung der Markenbegrenzung, erhöht aber gleichzeitig die Rechenzeit. Für jedes Pixel des simulierten Videobildes, das eine solche subpixel-genaue Markenbegrenzung berührt, wird der Grauwert linear zwischen `GrayMarks` und `GrayCaltab` anhand des Verhältnisses Innerhalb/Außerhalb gesetzt.

Mit dem Operator `sim_caltab` lassen sich synthetische Kalibrierbilder (mit bekannten Kameraparametern!) erzeugen, mit denen man die Güte des Kalibrieralgorithmus (vgl. `camera_calibration`) testen kann.

Parameter

- ▷ **SimImage** (output_object) . . . . . image ⇝ *Hobject* : byte
  Simuliertes Kalibrierbild.
- ▷ **CalTabDescrFile** (input_control) . . . . . string ⇝ *string*
  Dateiname der Eichkörperbeschreibungsdatei.
  Defaultwert : ’caltab.descr’
- ▷ **CamParam** (input_control) . . . . . number-array ⇝ *real* / integer
  Innere Kameraparameter.
  Parameteranzahl : 8

▷ **CamPose** (input_control) . . . . . pose-array ⇝ *real* / integer
Äußere Kameraparameter
Parameteranzahl : 7

▷ **GrayBackground** (input_control) . . . . . integer ⇝ *integer*
Hintergrundgrauwert des Bildes.
Defaultwert : 128
Wertevorschläge : `GrayBackground` ∈ {0, 32, 64, 96, 128, 160}
Restriktion : (`0` ≤ `GrayBackground`) ≤ `255`

▷ **GrayCaltab** (input_control) . . . . . integer ⇝ *integer*
Grauwert der Eichkörperplatte.
Defaultwert : 224
Wertevorschläge : `GrayCaltab` ∈ {144, 160, 176, 192, 208, 224, 240}
Restriktion : (`0` ≤ `GrayCaltab`) ≤ `255`

▷ **GrayMarks** (input_control) . . . . . integer ⇝ *integer*
Grauwert der Eichmarkierungen.
Defaultwert : 80
Wertevorschläge : `GrayMarks` ∈ {16, 32, 48, 64, 80, 96, 112}
Restriktion : (`0` ≤ `GrayMarks`) ≤ `255`

▷ **ScaleFac** (input_control) . . . . . real ⇝ *real*
Skalierungsfaktor zum Herabsetzen der Überabtastung.
Defaultwert : 1.0
Wertevorschläge : `ScaleFac` ∈ {1.0, 0.5, 0.25, 0.125}
Empfohlene Schrittweite : 0.05
Restriktion : `1.0` ≥ `ScaleFac`

Beispiel

```
// read calibration image
read_image(Image1,'calib-01.tiff')
//  find calibration pattern
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
// find calibration marks and initial pose
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)
//  read 3D positions of calibration marks
caltab_points('caltab.descr',NX,NY,NZ)
//  camera calibration
camera_calibration(NX,NY,NZ,RCoord1,CCoord1,
  [0.008,0.0,0.000011,0.000011,384,288,768,576],
  StartPose1,11,CamParam,FinalPose,Errors)
//  simulate calibration image
sim_caltab(Image1Sim,'caltab.descr',CamParam,FinalPose,128,224,80,1).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sim_caltab` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`sim_caltab` wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`camera_calibration`, `find_marks_and_pose`, `read_pose`, `read_cam_par`, `hom_mat3d_to_pose`

Mögliche Nachfolgerfunktionen

`find_caltab`

Siehe auch

`find_caltab`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `create_pose`, `hom_mat3d_to_pose`, `project_3d_point`, `create_caltab`

Modul

Camera calibration

write_cam_par (: : CamParam, CamParFile :)

Abspeichern der inneren Kameraparameter in eine Textdatei.

write_cam_par dient zum Abspeichern der inneren Kameraparameter CamParam in eine Textdatei mit dem Namen CamParFile.

Das Format der Textdatei ist eine (von Halcon unabhängige) generische Parameterbeschreibung, die beliebige Gruppen von Parametern zu hierarchisch organisierten Parametergruppen (Struktur ParGroup) zusammenfasst. Die Beschreibung eines Parameters innerhalb einer ParGroup enthält in 3 Zeilen folgende Einträge:

```
Name : Kurzname : aktueller Wert ;
 Typ : Untere Grenze (optional) : Obere Grenze (optional) ;
 Beschreibung (optional) ;
```

In diesem Format werden als Parametertypen BOOL, XBOOL (exklusives BOOL), INT, FLOAT, DOUBLE, STRING und FILE_SELECTOR unterstützt. Kommentare werden durch ein '#' am Zeilenanfang markiert.

Mit dem Operator write_cam_par wird in die Datei CamParFile die Parametergruppe Camera:Parameter geschrieben, in der die 8 Parameter Focus, Sx, Sy, Cx, Cy, Kappa (κ), ImageWidth und ImageHeight enthalten sind.

Als Kameramodell wird das einer **Lochkamera mit radialer Verzerrung** verwendet, falls die Kammerkonstante in CamParam mit einem Wert größer als 0 übergeben wird. Es beschreibt die Abbildung eines 3D-Punktes P_C in ein (Sub-)Pixel [r,c] des Videobildes durch folgende Transformationen:

$$\begin{aligned} P_C &= [x \;\; y \;\; z]^T \\ u = \text{Focus} \cdot \tfrac{x}{z} \;\; &\text{bzw.} \;\; v = \text{Focus} \cdot \tfrac{y}{z} \\ \tilde{u} = \tfrac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \;\; &\text{bzw.} \;\; \tilde{v} = \tfrac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}} \\ c = \tfrac{\tilde{u}}{S_x} + C_x \;\; &\text{bzw.} \;\; r = \tfrac{\tilde{v}}{S_y} + C_y \end{aligned}$$

Falls die Kammerkonstante in CamParam als 0 übergeben wird, wird als Kameramodell eine **Telezentrische Kamera mit radialer Verzerrung** verwendet, d.h. es wird angenommen, daß die Optik des Kameraobjektives eine Parallelprojektion der Weltpunkte durchführt. Die entsprechenden Gleichungen sind dann:

$$\begin{aligned} P_C = [x \;\; y \;\; z]^T &= \mathcal{R} \cdot P_W + \mathcal{T} \\ u = x \;\; &\text{bzw.} \;\; v = y \\ \tilde{u} = \tfrac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \;\; &\text{bzw.} \;\; \tilde{v} = \tfrac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}} \\ c = \tfrac{\tilde{u}}{S_x} + C_x \;\; &\text{bzw.} \;\; r = \tfrac{\tilde{v}}{S_y} + C_y \end{aligned}$$

Diese Gleichungen umfassen eine Koordinatentransformation vom Weltkoordinatensystem ins Kamerakoordinatensystem, eine perspektivische bzw. parallele Projektion in die Bildebene, eine radiale Verzerrung dieses projizierten Punktes und schließlich eine Diskretisierung und Hauptpunktverschiebung.

Die Parameter $\mathcal{R}$ und $\mathcal{T}$ beschreiben die 3D-Lage des Kamerakoordinatensystems im Weltkoordinatensystem und werden auch als **äußere Kameraparameter** bezeichnet, vgl. auch hom_mat3d_to_pose und create_pose.

Die inneren Kameraparameter bestehen aus der Brennweite (Focus), dem radialen Verzerrungskoeffizienten (κ), den Skalierungsfaktoren S_x und S_y (horizontaler und vertikaler Abstand zweier Zellen auf dem CCD-Chip), dem Kamerahauptpunkt $[C_x, C_y]^T$ und der verwendeten Bildbreite und Bildhöhe. Im Gegensatz zu den äußeren Kameraparametern sind die inneren Kameraparameter unabhängig von der konkreten Lage der CCD-Kamera. Sie beschreiben den Abbildungsvorgang der verwendeten Kombination von Kamera, Objektiv und Framegrabber und werden durch die Kamerakalibrierung bestimmt, vgl. camera_calibration.

Parameter

▷ **CamParam** (input_control) . . . . . number-array ⇝ *real* / integer
Innere Kameraparameter.
Parameteranzahl : 8

▷ **CamParFile** (input_control) . . . . . string ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert : 'campar.dat'
Werteliste : CamParFile ∈ {'campar.dat', 'campar.initial', 'campar.final'}

Beispiel

```
//  read calibration images
read_image(Image1,'calib-01.tiff')
read_image(Image2,'calib-02.tiff')
read_image(Image3,'calib-03.tiff')
//  find calibration pattern
find_caltab(Image1,Caltab1,'caltab.descr',3,112,5)
find_caltab(Image2,Caltab2,'caltab.descr',3,112,5)
find_caltab(Image3,Caltab,'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],11,CamParam,NFinalPose,Errors)
//  write internal camera parameters to file
write_cam_par(CamParam,'campar.dat').
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich geschrieben werden, dann liefert write_cam_par den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_cam_par ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

camera_calibration

Siehe auch

find_caltab, find_marks_and_pose, camera_calibration, disp_caltab, sim_caltab, read_cam_par, write_pose, read_pose, project_3d_point, get_line_of_sight

Modul

Camera calibration

write_pose (: : Pose, PoseFile :)

Abspeichern von 3D-Lageparametern in eine Textdatei.

write_pose dient zum Abspeichern von 3D-Lageparametern Pose in eine Textdatei mit dem Namen PoseFile.

Das Eingabetupel `Pose` beschreibt 3D-Lageparameter und definiert somit eine 3D-Transformation. Dies sind z.B. die äußeren Kameraparameter, gegeben durch den 3D-Translationsvektor (in Meter), die drei Werte der Rotationswinkel und die Codierung des Darstellungstyps der 3D-Transformation: Z.B. ist der Wert '0' der Code des Typs 1 und bedeutet, daß die Transformation eines Punktes beschrieben ist, hierbei die 3D-Rotation vor der Translation ausgeführt wird, daß die drei Rotationen als Winkel (in Grad) angegeben sind, und daß die Reihenfolge der drei Rotationen gleich γ-β-α ist, d.h. erst wird um die Z-Achse gedreht, dann um die neue Y-Achse und schließlich um die nach der zweiten Drehung entstandene X-Achse. `pose_to_hom_mat3d` verarbeitet jedoch alle verschieden Darstellungstypen für `Pose`. Die Bedeutung der anderen Darstellungstypen der 3D-Transformation ist bei `create_pose` beschrieben.

Eine solche Datei hat z.B. folgendes Aussehen:

```
# 3D POSE PARAMETERS: rotation and translation

# Used representation type:
f 0

# Rotation angles [deg] or Rodriguez-vector:
r -17.8134 1.83816 0.288092

# Translational vector (x y z [m]):
t 0.280164 0.150644 1.7554
```

Parameter

▷ **`Pose`** (input_control) . . . . . pose-array ⇝ *real* / integer
Äußere Kameraparameter.
Parameteranzahl : 7

▷ **`PoseFile`** (input_control) . . . . . filename ⇝ *string*
Dateiname der Kameraparameterdatei.
Defaultwert : 'campose.dat'
Werteliste : `PoseFile` ∈ {'campose.dat', 'campose.initial', 'campose.final'}

Beispiel

```
//  read calibration images
read_image(Image1,'calib-01.tiff')
read_image(Image2,'calib-02.tiff')
read_image(Image3,'calib-03.tiff')
//  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],11,CamParam,NFinalPose,Errors)
//  write external camera parameters of first calibration image
write_pose(NFinalPose[0],NFinalPose[1],NFinalPose[2],NFinalPose[3],
  NFinalPose[4],NFinalPose[5],NFinalPose[6],'campose.dat').
```

Ergebnis

Sind die Parameterwerte korrekt und konnte die Datei erfolgreich geschrieben werden, dann liefert `write_pose` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`write_pose` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`camera_calibration`, `hom_mat3d_to_pose`

Siehe auch

`create_pose`, `find_marks_and_pose`, `camera_calibration`, `disp_caltab`, `sim_caltab`, `read_pose`, `pose_to_hom_mat3d`, `hom_mat3d_to_pose`

Modul

Camera calibration

12.9 Kalmanfilter

```
filter_kalman ( : :  Dimension, Model, Measurement,
PredictionIn :  PredictionOut, Estimate )
```

Schätzung des aktuellen Zustands eines Systems mittels Kalman-Filterung.

`filter_kalman` liefert eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems. Kalman-Filter werden im Bereich der Bildverarbeitung insbesondere erfolgreich bei der Bildfolgenanalyse eingesetzt (Hintergrund-Erkennung, Fahrbahnverfolgung mittels Linienverfolgung oder mittels Regionen-Analyse etc.). Im folgenden wird zunächst eine kurze Einführung in die Theorie der Kalman-Filter gegeben. Im Anschluß daran wird dann die Routine `filter_kalman` selbst beschrieben.

KALMAN-FILTER: Ein diskretes, stochastisch gestörtes, lineares System wird durch folgende Kenngrößen charakterisiert:

- Status $x(t)$: Beschreibt den momentanen Zustand des Systems (Geschwindigkeiten, Temperaturen,...)
- Stellgröße $u(t)$: Eingaben von außen in das System
- Meßwerte $y(t)$: Meßwerte, die durch Beobachtung des Systems gewonnen werden. Sie spiegeln den Systemzustand (oder zumindest Teile davon) wider.
- Eine Ausgabefunktion, die die Abhängigkeit der Meßwerte vom Status beschreibt.
- Eine Übergangsfunktion, die angibt, wie sich der Status in Abhängigkeit von der Zeit, seinem momentanen Wert und den Stellgrößen verändert.

Die Ausgabefunktion und die Übergangsfunktion sind linear. Ihre Anwendung kann daher als Matrix-Multiplikation geschrieben werden.

Die Übergangsfunktion wird durch die Übergangsmatrix $A(t)$ und die Stellmatrix beschrieben, die Ausgangsfunktion durch die Meßmatrix $C(t)$. $A(t)$ charakterisiert dabei die Abhängigkeit des neuen vom alten Systemstatus, $G(t)$ die Abhängigkeit von den Stellgrößen. In der Praxis ist es selten möglich (oder zumindest zu aufwendig), ein reales System und sein Verhalten in sich geschlossen exakt zu beschreiben. Insbesondere beschränkt man sich in der Regel auf eine relativ kleine Zahl von Variablen zur Darstellung des Systemverhaltens. Daraus resultiert ein Fehler, der sogenannte Systemfehler (der auch Systemstörung genannt wird) $v(t)$. Auch die Ausgabefunktion ist im allgemeinen nicht exakt. Jede Messung ist fehlerbehaftet. Die Meßfehler seien mit $w(t)$ bezeichnet. Damit ergeben sich folgende Systemgleichungen:

$$x(t+1) = A(t)x(t) + G(t)u(t) + v(t)$$
$$y(t) = c(t)x(t) + w(t)$$

Die System- und Meßfehler $v(t)$ und $w(t)$ sind unbekannt. Sie werden bei den im Zusammenhang mit Kalman-Filterung betrachteten Systemen als gaußverteilte Zufallsvektoren betrachtet (daher die Bezeichnung „stochastisch gestörte Systeme“). Somit wird das System berechenbar, wenn die zu $v(t)$ und $w(t)$ gehörigen Erwartungswerte und Kovarianzmatrizen bekannt sind.

Die Schätzung des Systemzustandes erfolgt wie in der Gauss-Markov- Schätzung. Der Kalman-Filter ist jedoch ein rekursiver Algorithmus, der sich nur auf die aktuellen Meßwerte $y(t)$ und den letzten Status $x(t)$ stützt. Letzterer enthält implizit das Wissen um früher bereits angefallenen Meßwerte.

Für den Startwert $x(0)$ ist ein geeigneter Schätzwert x_0 anzugeben, der als Erwartungswert einer Zufallsvariable für $x(0)$ aufgefaßt wird. Deren Fehler habe die Erwartung 0 und die Kovarianzmatrix P_0, die ebenfalls anzugeben ist. Zum Zeitpunkt t sei der Erwartungswert der Störungen $v(t)$ und $w(t)$ jeweils 0 und ihre Kovarianzmatrizen seien $Q(t)$ und $R(t)$. $x(t)$, $v(t)$ und $w(t)$ werden meist als unkorreliert angenommen (es sind auch beliebige Rauschprozesse modellierbar - die Erstellung der benötigten Matrizen durch den Anwender ist dann allerdings deutlich aufwendiger). An die gesuchten Schätzwerte x^t werden folgende Bedingungen gestellt:

- Die Schätzwerte x^t hängen linear vom tatsächlichen Wert $x(t)$ und der Meßwertfolge $y(0), y(1), \cdots, y(t)$ ab.
- x^t sei erwartungstreu, d.h. $Ex^t = Ex(t)$.
- Als Gütekriterium erfülle x^t das Kriterium der minimalen Varianz, d.h. die Varianz des Schätzfehlers, definiert als $x(t) - x^t$, sei minimal.

Der Kalman-Filter führt nach der Initialisierung

$$\hat{x}(0) = x_0, \hat{P}(0) = P_0$$

zu jedem Zeitpunkt t folgende Berechnungsschritte aus:

$$\begin{array}{lrcl}
(K-III) & K(t) & = & \frac{\hat{P}(t)C'(t)}{C(t)\hat{P}(t)C'(t)+R(t)} \\
(K-IV) & x^t & = & \hat{x}(t) + K(t)(y(t) - C(t)\hat{x}(t)) \\
(K-V) & \tilde{P}(t) & = & \hat{P}(t) - K(t)C(t)\hat{P}(t) \\
(K-I) & \hat{x}(t+1) & = & A(t)x^t + G(t)u(t) \\
(K-II) & \hat{P}(t+1) & = & A(t)\tilde{P}(t)A'(t) + Q(t)
\end{array}$$

Dabei ist $\tilde{P}(t)$ die Kovarianzmatrix des Schätzfehlers, $\hat{x}(t)$ der Extrapolations- bzw. Prädiktionswert des Zustands, $\hat{P}(t)$ die Kovarianzen des Prädiktionsfehlers $\hat{x} - x$, K die Verstärkungsmatrix (das sogenannte Kalman-gain) und X' die Transponierte einer Matrix X. Hingewiesen sei an dieser Stelle auch auf die Möglichkeit der Vorhersage (Prädiktion) des Folgezustandes durch Gleichung (K-I). Dies ist z.B. in der Bildverarbeitung mitunter sehr nützlich, um „regions of interest“ im Folgebild zu bestimmen.

Wie oben erwähnt, ist die Modellierung beliebiger Rauschprozesse mitunter sehr aufwendig. Ist beispielsweise das System- und das Meßrauschen korelliert mit entsprechender Kovarianzmatrix L, sind die Gleichungen für das Kalman-gain und die Fehler-Kovarianzmatrix zu modifizieren:

$$\begin{array}{lrcl}
(K-III) & K(t) & = & \frac{\hat{P}(t)C'(t)+L(t)}{C(t)\hat{P}(t)+C(t)l(t)+L'C'(t)+R(t)} \\
(K-V) & \tilde{P}(t) & = & (\hat{P}(t) - K(t)C(t)\hat{P}(t))\hat{P}(t) - K(t)L(t)
\end{array}$$

Es ist nun Aufgabe des Benutzers, die linearen Systemgleichungen (K-I) bis (K-V) - in Abhängigkeit vom konkreten Problem - aufzustellen. Er muß also ein der Problemlösung zugrundeliegendes mathematisches Modell entwickeln. Statistische Größen, die die Ungenauigkeiten seines Systems und die die zu erwartenden Meßfehler beschreiben, müßen dabei geschätzt werden, wenn sie sich nicht exakt berechnen lassen. Im einzelnen sind also folgende Schritte nötig:

1. Aufstellen eines mathematischen Modells
2. Auswahl charakteristischer Zustandsvariablen
3. Aufstellen der Gleichungen, die die Aenderung dieser Zustandsvariablen beschreiben und Linearisierung derselben (Matrizen A und G)
4. Aufstellung der Gleichungen, die die Abhängigkeit der Meßwerte des Systems von den Zustandsvariablen beschreiben und Linearisierung derselben (Matrix C)
5. Aufstellung oder Schätzung statistischer Abhängigkeiten zwischen den Systemstörungen (Matrix Q)
6. Aufstellung oder Schätzung statistischer Abhängigkeiten zwischen den Meßfehlern (Matrix R)
7. Initialisierung des Anfangszustandes

Die Initialisierung des Systems (Punkt 7) verlangt dabei, wie oben erwähnt, die Angabe einer Schätzung x_0 des Systemzustandes zum Zeitpunkt 0 und der zugehörigen Kovarianzmatrix P_0. Ist der exakte Anfangszustand nicht bekannt, empfiehlt es sich, die Komponenten des Vektors x_0 auf die Mittelwerte der jeweiligen Wertebereiche zu setzen und hohe Werte in P_0 einzutragen (in der Größenordnung der Quadrate der Wertebereiche). Nach einigen Iterationen (wenn die Zahl der insgesamt akkumulierten Meßwerte größer geworden ist als die Zahl der Systemgrößen), erhält man auch hier brauchbare Werte. Ist umgekehrt der Anfangszustand exakt bekannt, sind alle Einträge von P_0 auf Null zu setzen - P_0 beschreibt ja die Kovarianzen des Fehlers zwischen dem Schätzwert x_0 und dem tatsächlichen Wert $x(0)$.

DIE FILTER-ROUTINE:

Eine Kalman-Filterung hängt von einer Reihe von Daten ab, die sich in vier Gruppen gliedern lassen:

Modellparameter: Übergangsmatrix A, Steuermatrix G mit der Stellgröße u und Meßmatrix C)

Modellstochastik: Systemfehler-Kovarianzmatrix Q, Systemfehler-Meßfehler- Kovarianzmatrix L und Meßfehler-Kovarianzmatrix R)

Meßvektor: y

Systemvergangenheit: Extrapolationsvektor $\hat{x}$ und Extrapolationsfehler-Kovarianzmatrix $\hat{P}$

Viele Systeme kommen dabei ohne Eingaben „von außen“ und damit ohne G und u aus. Außerdem sind im Normalfall System- und Meßfehler unkorreliert (L entfällt).

Konkret werden die benötigten Daten der Routine durch folgende Parameter übergeben:

`Dimension`: Dieser Parameter enthält die Dimensionen von Status-, Meß- und Stellvektor. `Dimension` ist somit ein Vektor **[n,m,p]**, wobei n die Anzahl der Zustandsvariablen, **m** die Anzahl der Meßwerte und **p** die Anzahl der Stellglieder ist. Für ein System ohne deterministische Steuerung (d.h. ohne Einfluß „von außen“) ist folglich **[n,m,0]** zu übergeben.

`Model`: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierten Matrizen (Vektoren) **A,C,Q,G,u** und (gegebenenfalls) **L**. `Model` ist also ein Vektor der Länge $n \times n + n \times m + n \times n + n \times p + p[+n \times m]$. Der letzte Summand entfällt, falls System- und Meßfehler unkorreliert sind, d.h. kein **L** anzugeben ist.

`Measurement`: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierte Matrix **R** und den Messvektor **y**. `Measurement` ist also ein Vektor der Dimension $m \times m + m$.

`PredictionIn` / `PredictionOut`: Diese Parameter enthalten hintereinandergehängt die zeilenweise linearisierte Matrix $\hat{P}$ (die Extrapolationsfehler-Kovarianzmatrix) und den Extrapolationsvektor $\hat{x}$, sind also Vektoren der Länge $n \times n + n$. `PredictionIn` ist ein Eingabeparameter, der $\hat{P}(t)$ und $\hat{x}(t)$ zum aktuellen Zeitpunkt t enthalten muß. In `PredictionOut` liefert die Routine dann die entsprechenden Vorhersagen $\hat{P}(t+1)$ und $\hat{x}(t+1)$.

`Estimate`: In Diesem Parameter liefert die Routine hintereinandergehängt die zeilenweise linearisierte Matrix $\tilde{P}$ (die Schätzfehler-Kovarianzmatrix) und den geschätzten Zustand $\tilde{x}$. `Estimate` ist daher ein Vektor der Länge $n \times n + n$.

Zu beachten ist dabei, daß Kovarianzmatrizen ($Q, R, \hat{P}, \tilde{P}$) naturgemäß symmetrisch sein müßen.

Parameter

▷ **`Dimension`** (input_control) ..... integer-array ⇝ *integer*
Die Dimensionen von Status-, Meß- und Stellvektor.
Defaultwert : '[3,1,0]'
Typischer Wertebereich : $0 \leq$ `Dimension` ≤ 30

▷ **`Model`** (input_control) ..... real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierten Matrizen A, C, Q, eventuell G und u und gegebenenfalls L.
Defaultwert : '[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]'
Typischer Wertebereich : $0.0 \leq$ `Model` ≤ 10000.0

▷ **`Measurement`** (input_control) ..... real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix R und der Meßvektor y.
Defaultwert : '[1.2,1.0]'
Typischer Wertebereich : $0.0 \leq$ `Measurement` ≤ 10000.0

▷ **PredictionIn** (input_control) ..... real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix $\hat{P}$ (die Extrapolationsfehler- kovarianzen) und der Extrapolationsvektor $\hat{x}$.
Defaultwert : '[0.0,0.0,0.0,0.0,180.5,0.0,0.0,0.0,100.0,0.0,100.0,0.0]'
Typischer Wertebereich : $0.0 \leq$ PredictionIn ≤ 10000.0

▷ **PredictionOut** (output_control) ..... real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix P$*$ (die Extrapolationsfehler- kovarianzen) und der Extrapolationsvektor $\hat{x}$

▷ **Estimate** (output_control) ..... real-array ⇝ *real*
Hintereinandergehängt die zeilenweise linearisierte Matrix $\tilde{P}$ (die Schätzfehlerkovarianzen) und der geschätzte Zustand $\tilde{x}$.

Beispiel

```
// Typical procedure:
// 1. To initialize the variables, which describe the model, e.g. with
read_kalman('kalman.init',Dim,Mod,Meas,Pred)
// 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.
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert filter_kalman den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

filter_kalman ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_kalman, sensor_kalman

Mögliche Nachfolgerfunktionen

update_kalman

Siehe auch

read_kalman, update_kalman, sensor_kalman

Literatur

W.Hartinger: „Entwurf eines anwendungsunabhängigen Kalman-Filters mit Untersuchungen im Bereich der Bildfolgenanalyse“; Diplomarbeit; Technische Universität München, Institut für Informatik, Lehrstuhl Prof. Radig; 1991.

R.E.Kalman: „A New Approach to Linear Filtering and Prediction Problems“; Transactions ASME, Ser.D: Journal of Basic Engineering; Vol. 82, S.34-45; 1960.

R.E.Kalman, P.l.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 Workshop, Florence, Italy, May, 29th - 31st, 1989; Elsevier, Amsterdam; 1990.

Modul

Tools

read_kalman (: : FileName : Dimension, Model, Measurement, Prediction)

Einlesen der Beschreibungsdatei eines Kalman-Filters.

read_kalman liest die Beschreibungsdatei FileName eines Kalman-Filters. Der Kalman-Filter liefert eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems. Sie werden im Bereich der Bildverarbeitung insbesondere erfolgreich bei der Bildfolgenanalyse eingesetzt. Eine Kalman-Filterung stützt sich dabei auf ein mathematisches Modell des zu untersuchenden Systems, das durch folgende Größen charakterisiert wird:

Modellparameter: Übergangsmatrix A, Steuermatrix G mit der Stellgröße u und Meßmatrix C

Modellstochastik: Systemfehler-Kovarianzmatrix Q, Systemfehler-Meßfehler-Kovarianzmatrix L und Meßfehler-Kovarianzmatrix R

Schätzung des Anfangszustandes des Systems: Zustand x_0 und zugehörige Kovarianzmatrix P_0

Viele Systeme kommen dabei ohne Eingaben „von außen“ und damit ohne G und u aus. Außerdem sind im Normalfall System- und Meßfehler unkorreliert (L entfällt).

Die oben genannten Kenngrößen können in einer ASCII-Datei mit folgendem Aufbau abgelegt und dann mittels read_kalman eingelesen werden:

Dimensionszeile
+ Inhaltszeile
+ Matrix A
+ Matrix C
+ Matrix Q
[+ Matrix G + Vektor u]
[+ Matrix L]
+ Matrix R
[+ Matrix P_0]
[+ Vektor x_0]

Die Dimensionszeile hat dabei die Form

n = <integer> m = <integer> p = <integer>

wobei n die Anzahl der Zustandsvariablen, m die Anzahl der Meßwerte und p die Anzahl der Stellglieder ist (siehe Parameter Dimension). Die maximale Dimension wird dabei durch eine Systemkonstante (derzeit 30) begrenzt.

Die Inhaltszeile ist von der Form

$$A * C * Q * G * u * L * R * P * x*$$

und beschreibt den weiteren Inhalt der Datei. Statt '$*$' ist dabei '+' (Parameter ist vorhanden) bzw. '-' (Parameter fehlt) einzusetzen. Zu beachten ist dabei, daß nur die oben durch [...] als optional gekennzeichneten Parameter in einer Beschreibungsdatei weggelassen werden dürfen. Fehlt die Anfangsschätzung a_0 (d.h. 'x-'), werden die Komponenten des Vektors als 0.0 angenommen. Fehlt die Kovarianzmatrix P_0 der Anfangsschätzung (d.h. 'P-'), wird angenommen, der Fehler sei sehr groß. Die Matrixelemente werden in diesem Fall auf 10000.0 gesetzt. Dieser Wert erscheint zwar hoch, ist aber nur dann ausreichend, wenn die Wertebereiche der Komponenten des Zustandsvektors x um Zehnerpotenzen kleiner sind. $(r \times s)$ Matrizen werden in der Form

$$\begin{array}{c} < Kommentar, d.h.\ string > \\ < a_{11} > \quad < a_{12} > \quad \cdots \quad < a_{1s} > \\ \vdots \qquad\qquad \ddots \qquad \vdots \\ < a_{r1} > \quad < a_{r2} > \quad \cdots \quad < a_{rs} > \end{array}$$

zeilenweise (mit beliebigen Zwischenräumen/Zeilenvorschüben) abgespeichert, Vektoren entsprechend in der Form

$$< Kommentar, d.h.\ string >$$
$$< a_1 > \quad \cdots \quad < a_k >$$

Zurückgeliefert werden von `read_kalman` folgende Parameterwerte:

`Dimension`: Dieser Parameter enthält die Dimensionen von Status-, Meß- und Stellvektor. Dimension ist somit ein Vektor **[n,m,p]**, wobei n die Anzahl der Zustandsvariablen, m die Anzahl der Meßwerte und p die Anzahl der Stellglieder ist. Für ein System ohne deterministische Steuerung (d.h. ohne Einfluß „von außen“) ist folglich `Dimension` = **[n,m,0]**.

`Model`: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierten Matrizen (Vektoren) **A,C,Q,G,u** und (gegebenenfalls) **L**. `Model` ist also ein Vektor der Länge $n \times n + n \times m + n \times n + n \times p + p[+n \times m]$. Der letzte Summand entfällt, falls System- und Meßfehler unkorreliert sind, d.h. kein **L** angegeben wurde.

`Measurement`: Dieser Parameter enthält die zeilenweise linearisierte Matrix R ist also ein Vektor der Dimension $m \times m$.

`Prediction`: Dieser Parameter enthält hintereinandergehängt die zeilenweise linearisierte Matrix P_0 (Fehler-Kovarianzmatrix der Anfangsschätzung) und die Anfangsschätzung x_0, ist also ein Vektor der Länge $n \times n + n$.

Parameter

- ▷ **`FileName`** (input_control) ..... filename ⇝ *string*
  Beschreibungsdatei für einen Kalman-Filter.
  Defaultwert : ”kalman.init”
- ▷ **`Dimension`** (output_control) ..... integer-array ⇝ *integer*
  Die Dimensionen von Status-, Meß- und Stellvektor.
- ▷ **`Model`** (output_control) ..... real-array ⇝ *real*
  Hintereinandergehängt die zeilenweise linearisierten Matrizen A, C, Q, eventuell G und u und gegebenenfalls L.
- ▷ **`Measurement`** (output_control) ..... real-array ⇝ *real*
  Die zeilenweise linearisierte Matrix R.
- ▷ **`Prediction`** (output_control) ..... real-array ⇝ *real*
  Hintereinandergehängt die zeilenweise linearisierte Matrix P_0 (die Fehler-Kovarianzmatrix der Anfangsschätzung) und die Anfangsschätzung x_0

Beispiel

```
%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 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
%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       = [1.0,1.0,0.5,0.0,1.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,
%                    54.3,37.9,48.0,37.9,34.3,42.5,48.0,42.5,43.7]
%Measurement = [1.2]
%Prediction  = [0.0,0.0,0.0,0.0,180.5,0.0,0.0,0.0,100.0,0.0,100.0,
%                          0.0].
```

Ergebnis

Ist die Beschreibungsdatei lesbar und korrekt, liefert `read_kalman` den Wert 2 (H_MSG_TRUE). Andernfalls wird eventuell eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`read_kalman` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`filter_kalman`

Siehe auch

`update_kalman`, `filter_kalman`, `sensor_kalman`

Modul

Tools

sensor_kalman (: : Dimension, MeasurementIn : MeasurementOut)

Interaktive Eingabe von Meßwerten für eine Kalman-Filterung.

`sensor_kalman` dient zur interaktiven Eingabe von Meßwerten für eine Kalman-Filterung. Kalman-Filter liefern eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems. Sie werden im Bereich der Bildverarbeitung insbesondere erfolgreich bei der Bildfolgenanalyse eingesetzt.

Jede Filterung basiert dabei auf Meßwerten. Wie diese aus Bildern oder Sensordaten gewonnen werden, ist von Anwendung zu Anwendung verschieden und muß daher dem Benutzer überlassen werden. Mit Hilfe von `sensor_kalman` ist jedoch eine interaktive Eingabe von (fiktiven) Meßwerten y und der zugehörigen Meßfehler-Kovarianzmatrix R möglich. Dies erleichtert insbesondere das Austesten von Kalman-Filtern während der Konstruktionsphase.

Die Parameter `MeasurementIn` und `MeasurementOut` enthalten dabei hintereinandergehängt die zeilenweise linearisierte Matrix R und den Meßvektor y, sind also Vektoren der Länge `Dimension` $\times$ `Dimension` $+$ `Dimension`

Parameter

- ▷ **Dimension** (input_control) . . . . . integer ⇝ *integer*
  Die Anzahl der Meßwerte.
  Defaultwert : 1
  Typischer Wertebereich : $0 \leq$ `Dimension` ≤ 30
- ▷ **MeasurementIn** (input_control) . . . . . real-array ⇝ *real*
  Hintereinandergehängt die zeilenweise linearisierte Matrix R und der Meßvektor y.
  Defaultwert : ’[1.2,1.0]’
  Typischer Wertebereich : $0.0 \leq$ `MeasurementIn` ≤ 10000.0
- ▷ **MeasurementOut** (output_control) . . . . . real-array ⇝ *real*
  Hintereinandergehängt die zeilenweise linearisierte Matrix R und der Meßvektor y.

Ergebnis

Sind die Parameter korrekt, liefert `sensor_kalman` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`sensor_kalman` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`filter_kalman`

Siehe auch

`filter_kalman`, `read_kalman`, `update_kalman`

Modul

Tools

update_kalman (: : FileName, DimensionIn, ModelIn, MeasurementIn : DimensionOut, ModelOut, MeasurementOut)

Einlesen einer Updatedatei eines Kalman-Filters.

`update_kalman` liest die Updatedatei `FileName` eines Kalman-Filters. Kalman-Filter liefern eine Schätzung des aktuellen Zustands (oder auch eine Prädiktion des künftigen Zustandes) eines diskreten, stochastisch gestörten, linearen Systems.

Eine Kalman-Filterung stützt sich dabei auf ein mathematisches Modell des zu untersuchenden Systems, das zu jedem Zeitpunkt durch folgende Größen charakterisiert wird:

Modellparameter: Übergangsmatrix A, Steuermatrix G mit der Stellgröße u und Meßmatrix C

Modellstochastik: Systemfehler-Kovarianzmatrix Q, Systemfehler-Meßfehler-Kovarianzmatrix L und Meßfehler-Kovarianzmatrix R

Meßvektor: y

Systemvergangenheit: Extrapolationsvektor $\hat{x}$ und Extrapolationsfehler-Kovarianzmatrix $\hat{P}$

Viele Systeme kommen dabei ohne Eingaben „von außen“ und damit ohne G und u aus. Außerdem sind im Normalfall System- und Meßfehler unkorreliert (L entfällt). Einige der oben genannten Größen können sich dynamisch (von Iteration zu Iteration) ändern. `update_kalman` dient zur Modifikation von Systemteilen gemäß einer Updatedatei (ASCII) mit folgendem Aufbau (vgl. auch `read_kalman`):

Dimensionszeile
+ Inhaltszeile
+ Matrix A
+ Matrix C
+ Matrix Q
+ Matrix G + Vektor u
+ Matrix L
+ Matrix R

Die Dimensionszeile hat dabei die Form

n = <integer> m = <integer> p = <integer>

wobei n die Anzahl der Zustandsvariablen, m die Anzahl der Meßwerte und p die Anzahl der Stellglieder ist (siehe Parameter `DimensionIn` / `DimensionOut`). Die maximale Dimension wird dabei durch eine Systemkonstante (derzeit 30) begrenzt. Da hier an einem gültigen Modell Veränderungen vorgenommen werden sollen, sind die Dimensionen n und m unveränderbar (und werden nur zu Kontrollzwecken angegeben). Die Inhaltszeile ist von der Form

$$A * C * Q * G * u * L * R*$$

und beschreibt den weiteren Inhalt der Datei. Statt ’$*$’ ist dabei ’+’ (Parameter ist vorhanden) bzw. ’-’ (Parameter fehlt) einzusetzen. Im Gegensatz zu Beschreibungsdateien für `read_kalman` muß die Systembeschreibung hier nicht vollständig sein. Anzugeben sind nur die sich ändernden Systemteile. Die Angabe von Schätzwerten entfällt, da diese gemäß Konstruktion des Filters von der jeweils letzten Filterung stammen müßen. $(r \times s)$ Matrizen werden in der Form

$$\begin{array}{cccc} < Kommentar, d.h.\ string > & & & \\ < a_{11} > & < a_{12} > & \cdots & < a_{1s} > \\ \vdots & & \ddots & \vdots \\ < a_{r1} > & < a_{r2} > & \cdots & < a_{rs} > \end{array}$$

zeilenweise (mit beliebigen Zwischenräumen/Zeilenvorschüben) abgespeichert, Vektoren entsprechend in der Form

$$\begin{array}{ccc} < Kommentar, d.h.\ string > & & \\ < a_1 > & \cdots & < a_k > \end{array}$$

Verändert werden von `read_kalman` folgende Parameterwerte:

`DimensionIn` / `DimensionOut`: Diese Parameter enthalten die Dimensionen von Status-, Meß- und Stellvektor und sind daher Vektoren **[n,m,p]**, wobei **n** die Anzahl der Zustandsvariablen, **m** die Anzahl der Meßwerte und **p** die Anzahl der Stellglieder ist. **n** und **m** sind für ein gegebenes System fest, dürfen also in der Updatedatei nicht von den entsprechenden Eingabewerten abweichen. Für ein System ohne deterministische Steuerung (d.h. ohne Einfluß „von außen“) ist **p = 0**.

`ModelIn` / `ModelOut`: Diese Parameter enthalten hintereinandergehängt die zeilenweise linearisierten Matrizen (Vektoren) **A,C,Q,G,u** und (gegebenenfalls) **L**. `ModelIn` / `ModelOut` sind also Vektoren der Länge $n \times n + n \times m + n \times n + n \times p + p[+n \times m]$. Der letzte Summand entfällt, falls System- und Meßfehler unkorreliert sind, d.h. kein **L** angegeben wurde.

`MeasurementIn` / `MeasurementOut`: Diese Parameter enthalten die zeilenweise linearisierte Matrix **R**, sind also Vektoren der Dimension $m \times m$.

Parameter

- ▷ **`FileName`** (input_control) ..... filename ⇝ *string*
  Updatedatei für einen Kalman-Filter.
  Defaultwert : ”kalman.updt”
- ▷ **`DimensionIn`** (input_control) ..... integer-array ⇝ *integer*
  Die Dimensionen von Status-, Meß- und Stellvektor.
  Defaultwert : ’[3,1,0]’
  Typischer Wertebereich : $0 \leq$ `DimensionIn` ≤ 30
- ▷ **`ModelIn`** (input_control) ..... real-array ⇝ *real*
  Hintereinandergehängt die zeilenweise linearisierten Matrizen A,C,Q, eventuell G und u und gegebenenfalls L.
  Defaultwert : ’[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]’
  Typischer Wertebereich : $0.0 \leq$ `ModelIn` ≤ 10000.0
- ▷ **`MeasurementIn`** (input_control) ..... real-array ⇝ *real*
  Die zeilenweise linearisierte Matrix R.
  Defaultwert : ’[1,2]’
  Typischer Wertebereich : $0.0 \leq$ `MeasurementIn` ≤ 10000.0
- ▷ **`DimensionOut`** (output_control) ..... integer-array ⇝ *integer*
  Die Dimensionen von Status-, Meß- und Stellvektor.
- ▷ **`ModelOut`** (output_control) ..... real-array ⇝ *real*
  Hintereinandergehängt die zeilenweise linearisierten Matrizen A,C,Q, eventuell G und u und gegebenenfalls L.
- ▷ **`MeasurementOut`** (output_control) ..... real-array ⇝ *real*
  Die zeilenweise linearisierte Matrix R.

Beispiel

```
%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]
```

Ergebnis

Ist die Updatedatei lesbar und korrekt, liefert `update_kalman` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`update_kalman` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`filter_kalman`

Siehe auch

`read_kalman`, `filter_kalman`, `sensor_kalman`

Modul

Tools

12.10 Matching

`clear_shape_model` (: : `ModelID` :)

Freigabe des Speichers eines Formmodells.

`clear_shape_model` gibt den Speicher eines Formmodells, das mit `create_shape_model` angelegt wurde, wieder frei. Das Modell kann nach dem Aufruf nicht mehr verwendet werden. Der Handle `ModelID` ist nach dem Aufruf ungültig.

Parameter

▷ **`ModelID`** (input_control) . . . . . shape_model ⇝ *integer*
Handle des Modells.

Ergebnis

Ist der Handle des Modells gültig, dann liefert `clear_shape_model` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`clear_shape_model` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`create_shape_model`, `read_shape_model`, `write_shape_model`

Modul

Template matching

create_shape_model (Template : : NumLevels, AngleStart, AngleExtent, AngleStep, Optimization, Metric, Contrast, MinContrast : ModelID)

Vorbereiten eines Formmodells für das Matching.

create_shape_model bereitet ein Muster, das als Bild Template übergeben wird, als Formmodell für das Matching vor.

Das Modell wird in auf mehreren Pyramidenebenen in mehereren Rotationen generiert und im Speicher abgelegt. Der Ausgabeparameter ModelID ist ein Handle für dises Modell, der in nachfolgenden Aufrufen von find_shape_model verwendet wird.

Die Anzahl der Pyramidenebenen wird mit NumLevels festgelegt. Sie sollte so groß wie möglich gewählt werden, da hierdurch das Auffinden des Modells erheblich beschleunigt wird. Bei der Wahl von NumLevels ist aber darauf zu achten, daß das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt (mindestens vier). Dies kann anhand der Ausgabe von inspect_shape_model überprüft werden. Falls nicht genügend Modellpunkte erzeugt werden, liefert create_shape_model eine Fehlermeldung zurück. Falls NumLevels als **0** übergeben wird, wählt create_shape_model die Anzahl der Pyramidenstufen automatisch. In seltenen Fällen kann es vorkommen, daß create_shape_model die Anzahl der Pyramidenstufen zu hoch oder zu niedrig bestimmt. Falls die Anzahl der Pyramidenstufen zu hoch gewählt wird, kann das dazu führen, daß das Modell im Bild nicht erkannt wird oder daß sehr niedrige Parameter für MinScore oder Greediness in find_shape_model selektiert werden müssen, damit das Modell gefunden wird. Falls die Anzahl der Pyramidenstufen zu niedrig gewählt wird, kann es zu erhöhten Laufzeiten in find_shape_model kommen. In diesen Fällen sollte die Anzahl der Pyramidenstufen mit Hilfe der Ausgabe von inspect_shape_model gewählt werden.

Die Parameter AngleStart und AngleExtent legen den Winkelbereich für die möglichen Rotationen des Modells im Bild fest. Das Modell kann also mit find_shape_model nur in diesem Winkelbereich gefunden werden. Der Parameter AngleStep gibt die Schrittweite der Winkel in dem gewählten Winkelbereich an. Falls bei find_shape_model keine Subpixelgenauigkeit spezifiziert wird, gibt AngleStep also die erreichbare Winkelgenauigkeit an. AngleStep sollte aufgrund der Größe des Objektes gewählt werden. Kleinere Modelle besitzen nur eine kleine Anzahl von verschiedenen diskreten Rotationen im Bild. Deshalb sollte AngleStep für kleinere Modelle größer gewählt werden. Falls AngleExtent kein ganzzahliges Vielfaches von AngleStep ist, wird AngleStep entsprechend angepaßt. Das Modell wird in dem gewählten Winkelbereich vorab erzeugt und im Speicher abgelegt. Der Speicherbedarf zur Speicherung des Modells ist also proportional zur Anzahl der Winkelschritte und der Anzahl der Punkte im Modell. Wenn also AngleStep zu klein bzw. AngleExtent zu groß gewählt wird, kann es vorkommen, daß das Modell nicht mehr in den (virtuellen) Speicher paßt. In diesem Fall muß entweder AngleStep größer oder AngleExtent kleiner gewählt werden. In jedem Fall ist es aus Laufzeitgründen vorteilhaft, wenn das Modell komplett in den Hauptspeicher paßt und somit ein Paging durch das Betriebssystem vermieden werden kann. Da die Möglichkeit zur subpixelgenauen Winkelbestimmung in find_shape_model gegeben ist, kann bei Modellen mit einem Durchmesser von ca. 300 Pixeln AngleStep $\geq$ 1 gewählt werden. Falls AngleStep $= 0$ übergeben wird, wählt create_shape_model automatisch eine Schrittweite basierend auf der Größe des Modells aus.

Bei besonders großen Modellen kann es auch sinnvoll sein, die Anzahl der Modellpunkte durch Setzen des Parameters Optimization auf einen Wert ungleich **'none'** zu setzen. Falls Optimization $=$ 'none', werden alle Modellpunkte abgespeichert. Ansonsten wird die Anzahl der Punkte entsprechend dem Parameter Optimization reduziert. Falls die Anzahl der Punkte reduziert wird, kann es bei find_shape_model notwendig werden, den Parameter **Greediness** auf einen kleineren Wert, z.B. 0.7 oder 0.8, zu setzen. Bei kleineren Modellen bewirkt die Reduktion der Anzahl der Punkte keine Beschleunigung, da dadurch typischerweise wesentlich mehr potentielle Instanzen des Modells untersucht werden müssen.

Der Parameter Contrast legt fest, welchen Grauwertkontrast die Punkte des Modells besitzen müssen. Der Kontrast ist ein Maß für die lokalen Grauwertdifferenzen zwischen dem Objekt und dem Hintergrund und zwischen verschiedenen Teilen des Objektes. Contrast sollte so gewählt werden, daß nur die signifikanten Merkmale des Musters für das Modell verwendet werden. Die Wirkung dieses Parameters kann auch vorab mit inspect_shape_model überprüft werden.

Mit MinContrast wird festgelegt, welchen Grauwertkontrast das Modell später bei der Erkennung mit find_shape_model im Bild mindestens besitzen muß. Mit anderen Worten stellt dieser Paremeter somit eine Abgrenzung des Musters von Rauschen im Bild dar. Eine gute Wahl ist deshalb der Bereich von Grauwertänderungen, der durch das Rauschen im Bild verursacht wird. Falls die Grauwerte z.B. in einem Bereich von 10 Graustufen durch Rauschen schwanken, sollte MinContrast auf 10 gesetzt werden. Offensichtlich muß MinContrast kleiner als Contrast sein. Falls das Modell später in sehr kontrastarmen Bildern erkannt werden soll, muß

MinContrast entsprechend klein gewählt werden. Falls das Modell mit erheblichen Verdeckungen erkannt werden soll, sollte MinContrast etwas größer als der Grauwertbereich, der durch das Rauschen verursacht wird, gewählt werden, um eine robuste und genaue Lageschätzung des verdeckten Modells zu gewährleisten.

Der Parameter Metric legt fest, unter welchen Bedingungen das Muster im Bild noch erkannt wird. Falls Metric = 'use_polarity', muß das Objekt im Bild dieselben Kontrasteigenschaften aufweisen wie das Modell. Wenn z.B. das Modell ein helles Objekt auf dunklem Hintergrund ist, wird das Objekt im Bild nur dann gefunden, wenn es auch heller als der Hintergrund ist. Falls Metric = 'ignore_global_polarity', wird das Objekt auch dann im Bild gefunden, wenn sich der Kontrast global umkehrt. Im obigen Beispiel würde das Objekt also auch dann gefunden, wenn es dunkler als der Hintergrund ist. Die Laufzeit von find_shape_model erhöht sich in diesem Fall geringfügig. Falls Metric = 'ignore_local_polarity', wird das Modell auch dann gefunden, wenn sich die Kontrastverhältnisse lokal ändern. Dieser Modus kann z.B. dann sinnvoll sein, wenn das Objekt aus einem Teil mittleren Grauwerts besteht, auf dem entweder dunkle oder helle Unterobjekte liegen können. Da sich in diesem Fall die Laufzeit von find_shape_model aber wesentlich erhöht, ist es in solchen Fällen meist sinnvoller, mehrere Modelle mit create_shape_model zu erzeugen und mit find_shape_model zu suchen.

Parameter

- ▷ **Template** (input_object) ..... image ⇝ *Hobject* : byte
  Eingabebild, dessen Definitionsbereich für das zum Aufbau des Modells verwendet wird.
- ▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Pyramidenebenen.
  Defaultwert : 0
  Werteliste : NumLevels ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
- ▷ **AngleStart** (input_control) ..... angle.rad ⇝ *real*
  Kleinste auftretende Rotation des Musters.
  Defaultwert : -0.39
  Wertevorschläge : AngleStart ∈ {-3.14, -1.57, -0.79, -0.39, -0.20, 0.0}
- ▷ **AngleExtent** (input_control) ..... angle.rad ⇝ *real*
  Ausdehnung des Winkelbereichs.
  Defaultwert : 0.79
  Wertevorschläge : AngleExtent ∈ {6.28, 3.14, 1.57, 0.79, 0.39}
  Restriktion : AngleExtent ≥ 0
- ▷ **AngleStep** (input_control) ..... angle.rad ⇝ *real*
  Schrittweite der Winkel (Auflösung).
  Defaultwert : 0
  Wertevorschläge : AngleStep ∈ {0, 0.0175, 0.0349, 0.0524, 0.0698, 0.0873}
  Restriktion : AngleStep ≥ 0
- ▷ **Optimization** (input_control) ..... string ⇝ *string*
  Art der Optimierung.
  Defaultwert : 'none'
  Werteliste : Optimization ∈ {'none', 'point_reduction_low', 'point_reduction_medium', 'point_reduction_high'}
- ▷ **Metric** (input_control) ..... string ⇝ *string*
  Art der zum Matchen verwendeten Metrik.
  Defaultwert : 'use_polarity'
  Werteliste : Metric ∈ {'use_polarity', 'ignore_global_polarity', 'ignore_local_polarity'}
- ▷ **Contrast** (input_control) ..... number ⇝ *integer*
  Kontrast des Objektes im Musterbild.
  Defaultwert : 30
  Wertevorschläge : Contrast ∈ {10, 20, 30, 40, 60, 80, 100, 120, 140, 160}
- ▷ **MinContrast** (input_control) ..... number ⇝ *integer*
  Minimaler Kontrast des Objektes in den Suchbildern.
  Defaultwert : 10
  Wertevorschläge : MinContrast ∈ {10, 20, 20, 40}
  Restriktion : MinContrast < Contrast
- ▷ **ModelID** (output_control) ..... shape_model ⇝ *integer*
  Handle des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert create_shape_model den Wert 2 (H_MSG_TRUE). Gegebe-

nenfalls wird eine Exception-Behandlung durchgeführt. Wenn die Parameter NumLevels und Contrast so gewählt worden sind, daß das Modell zu wenige Punkte besitzt, wird die Fehlermeldung 8510 zurückgeliefert.

Parallelisierungsinformation

create_shape_model wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

draw_region, reduce_domain, threshold

Mögliche Nachfolgerfunktionen

find_shape_model, clear_shape_model, write_shape_model

Alternativen

create_template_rot

Modul

Template matching

find_shape_model (Image : : ModelID, AngleStart, AngleExtent, MinScore, NumMatches, MaxOverlap, SubPixel, NumLevels, Greediness : Row, Column, Angle, Score)

Suche der besten Matches eines Formmodells in einem Bild.

find_shape_model findet die besten NumMatches Instanzen des Formmodells ModelID im Eingabebild Image. Das Modell muß zuvor mit create_shape_model erzeugt oder mit read_shape_model eingelesen worden sein.

Die Position und Rotation der gefundenen Instanzen des Modells wird in Row, Column und Angle zurückgeliefert. Zusätzlich wird in Score die Bewertung der gefundenen Instanzen zurückgegeben. Die Bewertung ist eine Zahl zwischen 0 und 1 und ist ein ungefähres Maß dafür, welcher Anteil des Modells im Bild zu sehen ist. Falls z.B. die Hälfte des Modells im Bild verdeckt ist, kann die Bewertung nicht größer als 0.5 sein.

Das Modell wird innerhalb des Definitionsbereiches des Eingabebildes nur an den Stellen gesucht, an denen das Modell vollständig in das Bild paßt. Das bedeutet, daß das Modell nicht gefunden werden kann, wenn es aus dem Bild herausragt, selbst wenn es eine Bewertung größer als MinScore erreichen würde (siehe unten). Die Parameter AngleStart und AngleExtent legen den Winkelbereich fest, in dem nach dem Modell gesucht wird. Der Winkelbereich wird gegebenenfalls auf den Bereich beschnitten, der bei der Erzeugung des Modells mit create_shape_model angegeben worden ist. Dies bedeutet insbesondere, daß die Winkelbereiche des Modells und der Suche sich tatsächlich überlappen müssen. Eine Änderung des Winkelbereichs bei der Suche modulo 2π erfolgt nicht. Zur Vereinfachung der Darstellung werden im Rest des Absatzes alle Winkel in Grad angegeben, obwohl sie in find_shape_model im Bogenmaß angegeben werden müssen. Falls das Modell also z.B. mit AngleStart $= -20$ [Grad] und AngleExtent $= 40$ [Grad] erzeugt worden ist und der Suchbereich in find_shape_model z.B. auf AngleStart $= 350$ [Grad] und AngleExtent $= 20$ [Grad] gesetzt wird, wird das Modell nicht gefunden, obwohl sich die Winkelbereiche bei Betrachtung modulo 360 [Grad] überlappen würden. Um das Modell zu finden, muß in diesem Beispiel AngleStart $= -10$ [Grad] gewählt werden.

Der Parameter MinScore legt fest, welche Bewertung ein potentieller Match mindestens besitzen muß, damit er als eine Instanz des Modells im Bild angesehen wird. Je größer der Wert von MinScore gewählt werden kann, desto schneller verläuft die Suche. Falls erwartet werden kann, daß das Modell niemals verdeckt wird, kann MinScore auf so hohe Werte wie 0.8 oder sogar 0.9 gesetzt werden.

Mit NumMatches kann angegeben werden, wieviele Instanzen des Modells im Bild höchstens gefunden werden sollen. Falls mehr als NumMatches Instanzen eine Bewertung größer als MinScore erreichen, werden nur die besten NumMatches Instanzen zurückgeliefert. Falls weniger als NumMatches Instanzen gefunden werden, werden nur diese Instanzen zurückgeliefert, d.h. der Parameter MinScore hat Vorrang vor NumMatches.

Falls das Modell Symmetrien aufweist, kann es vorkommen, daß mehrere Instanzen an ähnlichen Positionen im Bild, aber mit verschiedenen Rotationen gefunden werden. Mit dem Parameter MaxOverlap kann bestimmt werden, um welchen Anteil, ausgedrückt als Zahl zwischen 0 und 1, sich zwei Instanzen höchstens überlappen dürfen, damit sie als verschieden angesehen werden, und somit zurückgeliefert werden. Falls sich zwei Instanzen um mehr als MaxOverlap überlappen, wird nur die beste gefundene Instanz zurückgeliefert. Die Berechnung der Überlappung erfolgt anhand der kleinsten umschließenden Rechtecke beliebiger Orientierung der Konturen (siehe smallest_rectangle2). Bei MaxOverlap $= 0$ dürfen sich die gefundenen Instanzen nicht überlappen, bei MaxOverlap $= 1$ werden alle gefundenen Instanzen zurückgeliefert.

Der Parameter `SubPixel` gibt an, ob die Extraktion subpixelgenau erfolgen soll. Falls `SubPixel` auf **'true'** gesetzt wird, werden sowohl die Position als auch die Rotation subpixelgenau bestimmt.

Mit `NumLevels` wird die Anzahl der Pyramidenebenen festgelegt, die bei der Suche verwendet werden soll. Die Anzahl der Ebenen wird gegebenenfalls auf den bei der Erzeugung mit `create_shape_model` angebenen Bereich beschnitten. Falls `NumLevels` als **0** angegeben wird, wird die mit `create_shape_model` angegebene Anzahl verwendet.

Der Parameter `Greediness` bestimmt, wie „gierig" die Suche durchgeführt werden soll. Für `Greediness` $= 0$ wird eine sichere Suchheuristik verwendet, die das Modell, falls im Bild vorhanden, immer findet. Allerdings ist die Suche hiermit relativ zeitaufwendig. Für `Greediness` $= 1$ wird eine unsicherere Suchheuristik verwendet, bei der es in seltenen Fällen vorkommen kann, daß das Modell nicht gefunden wird, obwohl es im Bild sichtbar ist. Für `Greediness` $= 1$ wird die maximale Suchgeschwindigkeit erreicht. In den allermeisten wird das Template für `Greediness` $= 0.9$ immer sicher gefunden.

Parameter

▷ **`Image`** (input_object) ..... image ⇝ *Hobject* : byte
Eingabebild, in dem das Modell gefunden werden soll.

▷ **`ModelID`** (input_control) ..... shape_model ⇝ *integer*
Handle des Modells.

▷ **`AngleStart`** (input_control) ..... angle.rad ⇝ *real*
Kleinste auftretende Rotation des Modells.
Defaultwert : -0.39
Wertevorschläge : `AngleStart` ∈ {-3.14, -1.57, -0.78, -0.39, -0.20, 0.0}

▷ **`AngleExtent`** (input_control) ..... angle.rad ⇝ *real*
Ausdehnung des Winkelbereichs.
Defaultwert : 0.78
Wertevorschläge : `AngleExtent` ∈ {6.28, 3.14, 1.57, 0.78, 0.39, 0.0}
Restriktion : `AngleExtent` $\geq$ `0`

▷ **`MinScore`** (input_control) ..... real ⇝ *real*
Minimale Bewertung der zu findenden Instanzen des Modells.
Defaultwert : 0.5
Wertevorschläge : `MinScore` ∈ {0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich : $0 \leq$ `MinScore` ≤ 1
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.05

▷ **`NumMatches`** (input_control) ..... integer ⇝ *integer*
Anzahl der findenden Instanzen des Modells.
Defaultwert : 1
Wertevorschläge : `NumMatches` ∈ {0, 1, 2, 3, 4, 5, 10, 20}

▷ **`MaxOverlap`** (input_control) ..... real ⇝ *real*
Maximale Überlappung der zu findenden Instanzen des Modells.
Defaultwert : 0.5
Wertevorschläge : `MaxOverlap` ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich : $0 \leq$ `MaxOverlap` ≤ 1
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.05

▷ **`SubPixel`** (input_control) ..... string ⇝ *string*
Subpixelgenauigkeit falls **'true'**.
Defaultwert : 'true'
Werteliste : `SubPixel` ∈ {'true', 'false'}

▷ **`NumLevels`** (input_control) ..... integer ⇝ *integer*
Anzahl der verwendeten Pyramidenebenen.
Defaultwert : 0
Werteliste : `NumLevels` ∈ {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Greediness** (input_control) ..... real ⇝ *real*
„Gierigkeit“ der Suchheuristik (0: sicher aber langsam; 1: schnell aber Matches können „übersehen“ werden).
Defaultwert : 0.9
Wertevorschläge : Greediness ∈ {0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
Typischer Wertebereich : 0 ≤ Greediness ≤ 1
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.05

▷ **Row** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate der gefundenen Instanzen des Modells.

▷ **Column** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate der gefundenen Instanzen des Modells.

▷ **Angle** (output_control) ..... angle.rad-array ⇝ *real*
Rotationswinkel der gefundenen Instanzen des Modells.

▷ **Score** (output_control) ..... real-array ⇝ *real*
Bewertung der gefundenen Instanzen des Modells.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert find_shape_model den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels set_system ('no_object_result',<Result>) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

find_shape_model ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_shape_model, read_shape_model

Alternativen

best_match_rot_mg

Modul

Template matching

inspect_shape_model (Image : ModelImages, ModelRegions : NumLevels, Contrast :)

Repräsentation eines Formmodells.

inspect_shape_model erzeugt eine Repräsentation eines Formmodells. Der Operator ist insbesondere dazu nützlich, um die Parameter NumLevels und Contrast, die bei create_shape_model verwendet werden, einfach und schnell zu bestimmen. Die Repräsentation des Modells wird in auf mehreren Pyramidenebenen generiert, wobei die Anzahl der Pyramidenebenen durch NumLevels bestimmt wird. Im Gegensatz zu create_shape_model wird das Modell nur in der im Eingabebild Image vorliegenden Rotation, also 0°, erzeugt. Als Ausgabe erzeugt inspect_shape_model ein Bildobjekt ModelImages mit den Bildern der einzelnen Pyramidenstufen, sowie eine Region in ModelRegions für jede Pyramidenstufe, die das Modell auf dieser Pyramidenstufe repräsentiert. Auf die einzelnen Objekte der jeweiligen Pyramidenstufe kann mit select_obj zugegriffen werden. Wie bei create_shape_model beschrieben, sollten die Anzahl der Pyramidenebenen möglichst groß gewählt werden, wobei darauf zu achten ist, daß das Modell auf der obersten Pyramidenstufe noch erkennbar ist und genügend viele Punkte besitzt. Der Parameter Contrast sollte so gewählt werden, daß nur die signifikanten Merkmale des Musters für das Modell verwendet werden. In der normalen Verwendung wird inspect_shape_model mehmals interaktiv mit unterschiedlichen Werten für NumLevels und Contrast aufgerufen, bis ein zufriedenstellendes Modell entsteht. Hierauf wird create_shape_model mit den so bestimmten Parametern aufgerufen.

Parameter

▷ **Image** (input_object) ..... image ⇝ *Hobject* : byte
Eingabebild.

▷ **ModelImages** (output_object) ..... image-array ⇝ *Hobject* : byte
Bildpyramide des Eingagebildes.

▷ **ModelRegions** (output_object) ..... region-array ⇝ *Hobject*
Model-Regionen-Pyramide.

▷ **NumLevels** (input_control) ..... integer ⇝ *integer*
Anzahl von Pyramidenebenen.
Defaultwert : 4
Werteliste : NumLevels ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

▷ **Contrast** (input_control) ..... number ⇝ *integer*
Kontrast des Objektes im Bild.
Defaultwert : 30
Wertevorschläge : Contrast ∈ {10, 20, 30, 40, 60, 80, 100, 120, 140, 160}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert inspect_shape_model den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

inspect_shape_model ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

reduce_domain

Mögliche Nachfolgerfunktionen

select_obj

Siehe auch

create_shape_model

Modul

Template matching

read_shape_model (: : FileName : ModelID)

Einlesen eines Formmodells von Datei.

read_shape_model liest ein Formmodell, das mit write_shape_model geschrieben wurde, aus der Datei FileName ein.

Parameter

▷ **FileName** (input_control) ..... filename ⇝ *string*
Name der Datei.

▷ **ModelID** (output_control) ..... shape_model ⇝ *integer*
Handle des Modells.

Ergebnis

Ist der Dateiname korrekt, dann liefert read_shape_model den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_shape_model wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

find_shape_model

Siehe auch

create_shape_model, clear_shape_model

Modul

Template matching

write_shape_model (: : ModelID, FileName :)

Schreiben eines Formmodells auf Datei.

write_shape_model schreibt ein Formmodell in die Datei FileName. Das Modell kann mit read_shape_model wieder eingelesen werden.

Parameter

- ▷ **ModelID** (input_control) . . . . . shape_model ⇝ *integer*
  Handle des Modells.
- ▷ **FileName** (input_control) . . . . . filename ⇝ *string*
  Name der Datei.

Ergebnis

Ist der Dateiname korrekt (Schreiberlaubnis), dann liefert write_shape_model den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_shape_model ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_shape_model

Modul

Template matching

12.11 Measure

close_measure (: : MeasureHandle :)

Löschen eines Measure-Objektes.

close_measure löscht das Measure-Objekt, das in MeasureHandle übergeben wird. Der für das Measure-Objekt verwendete Speicher wird freigegeben.

Parameter

- ▷ **MeasureHandle** (input_control) . . . . . measure_id ⇝ *integer*
  Handle des Measure-Objekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert close_measure den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_measure ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_measure_rectangle2, measure_pos, measure_pairs

Modul

Sub-pixel operators

gen_measure_arc (: : CenterRow, CenterCol, Radius, AngleStart, AngleExtent, AnnulusRadius, Width, Height, Interpolation : MeasureHandle)

Vorbereitung der Extraktion gerader Kanten senkrecht zu einem Kreisring.

gen_measure_arc bereitet die Extraktion *gerader* Kanten, die *senkrecht* zu einem Kreisring liegen, vor. Mit Kreisring ist hier ein Kreisbogen mit zugehöriger Dicke gemeint. Der Mittelpunkt des Kreisbogens wird in den Parametern CenterRow und CenterCol übergeben, sein Radius in Radius, der Startwinkel in AngleStart und der Winkelbereich des Bogens relativ zum Startwinkel in AngleExtent. Falls AngleExtent > 0 wird ein Bogen entgegen dem Uhrzeigersinn erzeugt, ansonsten ein Bogen im Uhrzeigersinn. Der Radius des Rings, d.h. seine halbe Dicke, wird mit AnnulusRadius bestimmt.

Der Algorithmus zur Kantenextraktion ist bei measure_pos beschrieben. Wie dort dargestellt, können verschiedene Arten der Interpolation zur Berechnung des Grauwertprofils verwendet werden. Mit Interpolation

= **'nearest_neighbor'** werden die Grauwerte bei der Messung durch den Grauwert des nächstgelegenen Pixels, d.h. durch konstante Interpolation, bestimmt. Für `Interpolation` = **'bilinear'** wird bilineare Interpolation verwendet und für `Interpolation` = **'bicubic'** bikubische Interpolation.

Mit `gen_measure_arc` werden alle Berechnungen, die für mehrere Messungen verwendet werden können, aus dem eigentlichen Meß-Operator ausgelagert. Hierzu wird eine optimierte Datenstruktur, ein sogenanntes Measure-Objekt, aufgebaut, das in `MeasureHandle` zurückgegeben wird. Um die Messungen möglichst schnell ausführen zu können, muß bereits bei der Generierung des Measure-Objekts die später bei der Messung verwendete Bildgröße in `Width` und `Height` angegeben werden. Dieses Vorgehen erhöht die Geschwindigkeit der eigentlichen Messungen erheblich.

Der Systemparameter **'int_zooming'** (siehe `set_system`) beeinflußt die Genauigkeit und Geschwindigkeit der Berechnung des Measure-Objekts. Falls **'int_zooming'** auf **'true'** gesetzt wird, werden die internen Berechnungen in Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die geometrische Genauigkeit etwas geringer. Falls **'int_zooming'** auf **'false'** gesetzt wird, werden die internen Berechnungen in Gleitkommaarithmetik durchgeführt, was zur bestmöglichen geometrischen Genauigkeit führt, die Laufzeit allerdings signifikant erhöht.

Parameter

▷ **`CenterRow`** (input_control) . . . . . point.y ⇝ *real* / integer
Zeilenkoordinate des Mittelpunktes des Bogens.
Defaultwert : 100.0
Wertevorschläge : `CenterRow` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich : 0.0 ≤ `CenterRow` ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **`CenterCol`** (input_control) . . . . . point.x ⇝ *real* / integer
Spaltenkoordinate des Mittelpunktes des Bogens.
Defaultwert : 100.0
Wertevorschläge : `CenterCol` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich : 0.0 ≤ `CenterCol` ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **`Radius`** (input_control) . . . . . number ⇝ *real* / integer
Radius des Bogens.
Defaultwert : 50.0
Wertevorschläge : `Radius` ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich : 0.0 ≤ `Radius` ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **`AngleStart`** (input_control) . . . . . angle.rad ⇝ *real* / integer
Start des Bogens im Bogenmaß.
Defaultwert : 0.0
Wertevorschläge : `AngleStart` ∈ {-3.14159265359, -2.35619449019, -1.5707963268, -0.785398163398, 0.0, 0.785398163398, 1.5707963268, 2.35619449019, 3.14159265359}
Typischer Wertebereich : -3.14159265359 ≤ `AngleStart` ≤ 3.14159265359 (lin)
Minimale Schrittweite : 0.0314159265359
Empfohlene Schrittweite : 0.314159265359

▷ **`AngleExtent`** (input_control) . . . . . angle.rad ⇝ *real* / integer
Winkelausdehnung des Bogens im Bogenmaß.
Defaultwert : 6.28318530718
Wertevorschläge : `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}
Typischer Wertebereich : -6.28318530718 ≤ `AngleExtent` ≤ 6.28318530718 (lin)
Minimale Schrittweite : 0.0314159265359
Empfohlene Schrittweite : 0.314159265359
Restriktion : `AngleExtent` ≠ `0.0`

▷ **AnnulusRadius** (input_control) ..... number ⇝ *real* / integer
Radius (halbe Breite) des Rings.
Defaultwert : 10.0
Wertevorschläge : AnnulusRadius ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich : 0.0 ≤ AnnulusRadius ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : AnnulusRadius ≤ Radius

▷ **Width** (input_control) ..... extent.x ⇝ *integer*
Breite des später zu verarbeitenden Bildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
Typischer Wertebereich : 0 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16

▷ **Height** (input_control) ..... extent.y ⇝ *integer*
Höhe des später zu verarbeitenden Bildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}
Typischer Wertebereich : 0 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16

▷ **Interpolation** (input_control) ..... string ⇝ *string*
Art der zu verwendenden Interpolation.
Defaultwert : 'nearest_neighbor'
Werteliste : Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **MeasureHandle** (output_control) ..... measure_id ⇝ *integer*
Handle des Measure-Objekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_measure_arc den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_measure_arc ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

draw_circle

Mögliche Nachfolgerfunktionen

measure_pos, measure_pairs

Alternativen

edges_sub_pix

Modul

Sub-pixel operators

gen_measure_rectangle2 (: : Row, Column, Phi, Length1, Length2, Width, Height, Interpolation : MeasureHandle)

Vorbereitung der Extraktion gerader Kanten senkrecht zu einem Rechteck.

gen_measure_rectangle2 bereitet die Extraktion *gerader* Kanten, die *senkrecht* zur Hauptachse eines Rechtecks liegen, vor. Der Mittelpunkt des Rechtecks wird in den Parametern Row und Column übergeben, die Richtung der Hauptachse in Phi und die Längen der beiden Achsen, d.h. die halben Durchmesser des Rechtecks, in Length1 und Length2.

Der Algorithmus zur Kantenextraktion ist bei measure_pos beschrieben. Wie dort beschrieben, können verschiedene Arten der Interpolation zur Berechnung des Grauwertprofils verwendet werden. Mit Interpolation = **'nearest_neighbor'** werden die Grauwerte bei der Messung durch den Grauwert des nächstgelegenen Pixels, d.h. durch konstante Interpolation, bestimmt. Für Interpolation = **'bilinear'** wird bilineare Interpolation verwendet und für Interpolation = **'bicubic'** bikubische Interpolation.

Mit gen_measure_rectangle2 werden alle Berechnungen, die für mehrere Messungen verwendet werden können, aus dem eigentlichen Meß-Operator ausgelagert. Hierzu wird eine optimierte Datenstruktur, ein sogenanntes Measure-Objekt, aufgebaut, das in MeasureHandle zurückgegeben wird. Um die Messungen möglichst schnell ausführen zu können, muß bereits bei der Generierung des Measure-Objekts die später bei der Messung verwendete Bildgröße in Width und Height angegeben werden. Dieses Vorgehen erhöht die Geschwindigkeit der eigentlichen Messungen erheblich.

Der Systemparameter **'int_zooming'** (siehe set_system) beeinflußt die Genauigkeit und Geschwindigkeit der Berechnung des Measure-Objekts. Falls **'int_zooming'** auf **'true'** gesetzt wird, werden die internen Berechnungen in Festkommaarithmetik durchgeführt, was zu wesentlich kürzeren Laufzeiten führt. Allerdings ist hier die geometrische Genauigkeit etwas geringer. Falls **'int_zooming'** auf **'false'** gesetzt wird, werden die internen Berechnungen in Gleitkommaarithmetik durchgeführt, was zur bestmöglichen geometrischen Genauigkeit führt, die Laufzeit allerdings signifikant erhöht.

Parameter

▷ **Row** (input_control) . . . . . rectangle2.center.y ⇝ *real* / integer
Zeilenkoordinate des Mittelpunktes des Rechtecks.
Defaultwert : 50.0
Wertevorschläge : Row ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich : 0.0 ≤ Row ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Column** (input_control) . . . . . rectangle2.center.x ⇝ *real* / integer
Spaltenkoordinate des Mittelpunktes des Rechtecks.
Defaultwert : 100.0
Wertevorschläge : Column ∈ {10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0, 500.0}
Typischer Wertebereich : 0.0 ≤ Column ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Phi** (input_control) . . . . . rectangle2.angle.rad ⇝ *real* / integer
Winkel der Längsachse zur Horizontalen (Bogenmaß).
Defaultwert : 0.0
Wertevorschläge : Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Typischer Wertebereich : -1.178097 ≤ Phi ≤ 1.178097 (lin)
Minimale Schrittweite : 0.001
Empfohlene Schrittweite : 0.1
Restriktion : (−pi < Phi) ∧ (Phi ≤ pi)

▷ **Length1** (input_control) . . . . . rectangle2.hwidth ⇝ *real* / integer
Halbe Breite.
Defaultwert : 200.0
Wertevorschläge : Length1 ∈ {3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0, 300.0, 500.0}
Typischer Wertebereich : 0.0 ≤ Length1 ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0

▷ **Length2** (input_control) . . . . . rectangle2.hheight ⇝ *real* / integer
Halbe Höhe.
Defaultwert : 100.0
Wertevorschläge : Length2 ∈ {1.0, 2.0, 3.0, 5.0, 10.0, 15.0, 20.0, 50.0, 100.0, 200.0}
Typischer Wertebereich : 0.0 ≤ Length2 ≤ 511.0 (lin)
Minimale Schrittweite : 1.0
Empfohlene Schrittweite : 10.0
Restriktion : Length2 ≤ Length1

▷ **Width** (input_control) . . . . . extent.x ⇝ *integer*
Breite des später zu verarbeitenden Bildes.
Defaultwert : 512
Wertevorschläge : Width ∈ {128, 160, 192, 256, 320, 384, 512, 640, 768}
Typischer Wertebereich : 0 ≤ Width ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16

▷ **Height** (input_control) . . . . . extent.y ⇝ *integer*
Höhe des später zu verarbeitenden Bildes.
Defaultwert : 512
Wertevorschläge : Height ∈ {120, 128, 144, 240, 256, 288, 480, 512, 576}
Typischer Wertebereich : 0 ≤ Height ≤ 1024 (lin)
Minimale Schrittweite : 1
Empfohlene Schrittweite : 16

▷ **Interpolation** (input_control) . . . . . string ⇝ *string*
Art der zu verwendenden Interpolation.
Defaultwert : 'nearest_neighbor'
Werteliste : Interpolation ∈ {'nearest_neighbor', 'bilinear', 'bicubic'}

▷ **MeasureHandle** (output_control) . . . . . measure_id ⇝ *integer*
Handle des Measure-Objekts.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_measure_rectangle2 den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_measure_rectangle2 ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

draw_rectangle2

Mögliche Nachfolgerfunktionen

measure_pos, measure_pairs, measure_thresh

Alternativen

edges_sub_pix

Modul

Sub-pixel operators

measure_pairs (Image : : MeasureHandle, Sigma, Threshold, Transition, Select : RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, IntraDistance, InterDistance)

Extraktion gerader Kantenpaare senkrecht zu einem Rechteck.

measure_pairs dient zur Extraktion von *geraden* Kantenpaaren, die *senkrecht* zur Hauptachse eines Rechtecks liegen.

Der Algorithmus arbeitet analog zu measure_pos. Die Kanten werden jedoch zu Paaren mit unterschiedlichen Kantenübergängen (Vorzeichen) zusammengefaßt. Falls Transition = **'positive'** werden in RowEdgeFirst und ColumnEdgeFirst Kantenpunkte mit Übergang von dunkel nach hell und in RowEdgeSecond und ColumnEdgeSecond mit Übergang von hell nach dunkel übergeben. Im Modus **'negative'** ist dies genau umgekehrt. Im Modus **'all'** bestimmt die erste gefunden Kante die Art der Paarbildung. Treten Kanten mit gleichen Vorzeichen mehrfach hintereinander auf, so wird jeweils die erste Kante der Folge zur Paarbildung verwendet. Schließlich kann gewählt werden, welche der extrahierten Kantenpaare zurückgeliefert werden sollen. Wenn Select auf **'all'** gesetzt wird, werden alle gefundenen Paare zurückgegeben, falls Select auf **'first'** gesetzt wird, nur das erste Kantepaar, und falls Select auf **'last'** gesetzt wird, nur das letzte.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks liegen, zurückgegeben. Die zugehörigen Kantenamplituden werden in AmplitudeFirst und AmplitudeSecond zurückgeliefert. Zusätzlich wird noch der Abstand innerhalb eines Kantenpaares in IntraDistance und der Abstand von aufeinanderfolgenden Kantenpaaren in InterDistance zurückgeliefert. Dabei entspricht IntraDistance[i] dem Abstand von EdgeFirst[i] und EdgeSecond[i], während InterDistance[i] dem Abstand von EdgeSecond[i] und EdgeFirst[i+1] entspricht. D.h., das Tupel InterDistance enthält ein Element weniger als die Tupel der Kantenpaare.

Achtung

measure_pairs liefert nur dann brauchbare Ergebnisse, falls die Annahme, daß die Kanten gerade und senkrecht zur Hauptachse des Rechtecks sind, erfüllt ist. measure_pairs sollte daher z.B. nicht dazu verwendet

werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, daß das Rechteck möglichst senkrecht zu den Bildkanten liegt.

Parameter

- ▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **MeasureHandle** (input_control) ..... measure_id ⇝ *integer*
  Handle des Measure-Objekts.
- ▷ **Sigma** (input_control) ..... number ⇝ *real*
  Sigma der Gaußglättung.
  Defaultwert : 1.0
  Wertevorschläge : `Sigma` ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich : 0.4 ≤ `Sigma` ≤ 100 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Sigma` ≥ `0.4`
- ▷ **Threshold** (input_control) ..... number ⇝ *real*
  Minimale Amplitude einer Kante.
  Defaultwert : 30.0
  Wertevorschläge : `Threshold` ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
  Typischer Wertebereich : 1 ≤ `Threshold` ≤ 255 (lin)
  Minimale Schrittweite : 0.5
  Empfohlene Schrittweite : 2
- ▷ **Transition** (input_control) ..... string ⇝ *string*
  Hell/dunkel oder dunkel/hell Kante.
  Defaultwert : 'all'
  Werteliste : `Transition` ∈ {'all', 'positive', 'negative'}
- ▷ **Select** (input_control) ..... string ⇝ *string*
  Auswahl der Endpunkte.
  Defaultwert : 'all'
  Werteliste : `Select` ∈ {'all', 'first', 'last'}
- ▷ **RowEdgeFirst** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Kante.
- ▷ **ColumnEdgeFirst** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Kante.
- ▷ **AmplitudeFirst** (output_control) ..... real-array ⇝ *real*
  Kantenamplitude der Kanten (mit Vorzeichen).
- ▷ **RowEdgeSecond** (output_control) ..... point.y-array ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Kante.
- ▷ **ColumnEdgeSecond** (output_control) ..... point.x-array ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Kante.
- ▷ **AmplitudeSecond** (output_control) ..... real-array ⇝ *real*
  Kantenamplitude der Kanten (mit Vorzeichen).
- ▷ **IntraDistance** (output_control) ..... real-array ⇝ *real*
  Abstand zwischen Kanten eines Kantenpaars.
- ▷ **InterDistance** (output_control) ..... real-array ⇝ *real*
  Abstand zwischen aufeinanderfolgenden Kantenpaaren.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `measure_pairs` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`measure_pairs` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_measure_rectangle2`

Mögliche Nachfolgerfunktionen

`close_measure`

Alternativen

edges_sub_pix, measure_pos

Modul

Sub-pixel operators

measure_pos (Image : : MeasureHandle, Sigma, Threshold, Transition, Select : RowEdge, ColumnEdge, Amplitude, Distance)

Extraktion gerader Kanten senkrecht zu einem Rechteck.

measure_pos dient zur Extraktion von *geraden* Kanten, die *senkrecht* zur Hauptachse eines Rechtecks liegen.

Der Algorithmus erzeugt zunächst ein eindimensionales Grauwertprofil durch Mittelung der Grauwerte entlang von Geraden senkrecht zur Hauptachse des Rechtecks. Dabei wird das Eingabebild Image an Sub-Pixel-Positionen abgetastet, die einen ganzzahligen Zeilen- und Spalten-Abstand (im Koordinatensystem des Rechtecks) zum Mittelpunkt des Rechtecks haben. Weil die Abtastung des Bildes einige Berechnungen erfordert, die in mehreren Messungen verwendet werden können, wird der Operator gen_measure_rectangle2 verwendet, um diese Berechnungen nur einmal ausführen zu müssen, und somit die Geschwindigkeit von measure_pos signifikant zu erhöhen. Aufgrund der Tatsache, daß eine bessere Interpolation bei der Sub-Pixel-Abtastung der Grauwerte zu einer besseren Genauigkeit der extrahierten Kanten führt, aber auch die Laufzeit des Operators erhöht, können verschiedene Interpolationsverfahren in gen_measure_rectangle2 gewählt werden. (Die Interpolation beeinflußt nur Rechtecke, die nicht parallel zu den Koordinatenachsen des Bildes sind.) Das Measure-Objekt, das mit gen_measure_rectangle2 erzeugt wurde, wird in MeasureHandle übergeben.

Nach der Bestimmung des eindimensionalen Kantenprofils werden die sub-pixel-genauen Kantenpositionen durch Faltung des Profils mit den Ableitungen einer Gauß-Maske mit Standardabweichung Sigma berechnet. Auffällige Kanten können mit dem Parameter Threshold, der einen Schwellwert für die Kantenamplitude, d.h. den Absolutbetrag der ersten Ableitung, angibt, selektiert werden. Außerdem ist es möglich, nur positive Kanten, also solche, die einen Übergang von dunkel nach hell in der Richtung der Hauptachse des Rechtecks darstellen, (Transition = **'positive'**), nur negative Kanten, also Hell-Dunkel-Übergänge, (Transition = **'negative'**) oder alle Kanten zu selektieren (Transition = **'all'**). Schließlich kann gewählt werden, welche der extrahierten Kanten zurückgeliefert werden sollen. Wenn Select auf **'all'** gesetzt wird, werden alle gefundenen Kanten zurückgegeben, falls Select auf **'first'** gesetzt wird, nur die erste Kante, und falls Select auf **'last'** gesetzt wird, nur die letzte.

Die extrahierten Kanten werden als einzelne Punkte, die auf der Hauptachse des Rechtecks liegen, in (RowEdge,ColumnEdge) zurückgegeben. Die zugehörigen Kantenamplituden werden in Amplitude zurückgeliefert. Zusätzlich wird noch der Abstand von aufeinanderfolgenden Kantenpunkten in Distance zurückgeliefert. Dabei entspricht Distance[i] dem Abstand von Edge[i] und Edge[i+1]. D.h., das Tupel Distance enthält ein Element weniger als die Tupel RowEdge und ColumnEdge.

Achtung

measure_pos liefert nur dann brauchbare Ergebnisse, falls die Annahme, daß die Kanten gerade und senkrecht zur Hauptachse des Rechtecks sind, erfüllt ist. measure_pos sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, daß das Rechteck möglichst senkrecht zu den Bildkanten liegt.

Parameter

- **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte
  Eingabebild.
- **MeasureHandle** (input_control) ..... measure_id ⇝ *integer*
  Handle des Measure-Objekts.
- **Sigma** (input_control) ..... number ⇝ *real*
  Sigma der Gaußglättung.
  Defaultwert : 1.0
  Wertevorschläge : Sigma ∈ {0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich : 0.4 ≤ Sigma ≤ 100 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Sigma ≥ 0.4

▷ **Threshold** (input_control) ..... number ⇝ *real*
Minimale Amplitude einer Kante.
Defaultwert : 30.0
Wertevorschläge : Threshold ∈ {5.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 90.0, 110.0}
Typischer Wertebereich : 1 ≤ Threshold ≤ 255 (lin)
Minimale Schrittweite : 2
Empfohlene Schrittweite : 0.5

▷ **Transition** (input_control) ..... string ⇝ *string*
Hell/dunkel oder dunkel/hell Kante.
Defaultwert : 'all'
Werteliste : Transition ∈ {'all', 'positive', 'negative'}

▷ **Select** (input_control) ..... string ⇝ *string*
Auswahl der Endpunkte.
Defaultwert : 'all'
Werteliste : Select ∈ {'all', 'first', 'last'}

▷ **RowEdge** (output_control) ..... point.y-array ⇝ *real*
Zeilenkoordinate des Mittelpunktes der Kante.

▷ **ColumnEdge** (output_control) ..... point.x-array ⇝ *real*
Spaltenkoordinate des Mittelpunktes der Kante.

▷ **Amplitude** (output_control) ..... real-array ⇝ *real*
Kantenamplitude der Kanten (mit Vorzeichen).

▷ **Distance** (output_control) ..... real-array ⇝ *real*
Abstand zwischen aufeinanderfolgenden Kanten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert measure_pos den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

measure_pos ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_measure_rectangle2

Mögliche Nachfolgerfunktionen

close_measure

Alternativen

edges_sub_pix, measure_pairs

Modul

Sub-pixel operators

measure_projection (Image : : MeasureHandle : GrayValues)

Extraktion eines Grauwertprofils senkrecht zu einem Rechteck.

measure_projection extrahiert ein eindimensionales Grauwertprofil durch Mittelung der Grauwerte entlang von Geraden senkrecht zur Hauptachse eines Rechtecks. Dabei wird das Eingabebild Image an Sub-Pixel-Positionen abgetastet, die einen ganzzahligen Zeilen- und Spalten-Abstand (im Koordinatensystem des Rechtecks) zum Mittelpunkt des Rechtecks haben. Weil die Abtastung des Bildes einige Berechnungen erfordert, die in mehreren Projektionen verwendet werden können, wird der Operator gen_measure_rectangle2 verwendet, um diese Berechnungen nur einmal ausführen zu müssen, und somit die Geschwindigkeit von measure_projection signifikant zu erhöhen. Aufgrund der Tatsache, daß eine bessere Interpolation bei der Sub-Pixel-Abtastung der Grauwerte zu einer besseren Genauigkeit des extrahierten Grauwertprofils führt, aber auch die Laufzeit des Operators erhöht, können verschiedene Interpolationsverfahren in gen_measure_rectangle2 gewählt werden (die Interpolation beeinflußt nur Rechtecke, die nicht parallel zu den Koordinatenachsen des Bildes sind). Das Measure-Objekt, das mit gen_measure_rectangle2 erzeugt wurde, wird in MeasureHandle übergeben.

Parameter

- ▷ **Image** (input_object) ..... singlechannel-image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **MeasureHandle** (input_control) ..... measure_id ⇝ *integer*
  Handle des Measure-Objekts.
- ▷ **GrayValues** (output_control) ..... number-array ⇝ *real*
  Grauwertprofil.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `measure_projection` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`measure_projection` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_measure_rectangle2`

Mögliche Nachfolgerfunktionen

`close_measure`

Alternativen

`gray_projections`

Modul

Sub-pixel operators

measure_thresh (Image : : MeasureHandle, Sigma, Threshold, Select : RowThresh, ColumnThresh, Distance)

Extraktion von Punkten mit einem bestimmten Grauwert entlang eines Rechtecks.

`measure_thresh` bestimmt in einem eindimensionalen Grauwertprofil die Punkte mit dem Grauwert `Threshold`. Da das Grauwertprofil auf die Hauptachse des mit `MeasureHandle` übergebenen Measure-Rechtecks projiziert wird, entsprechen die im Grauwertprofil ermittelten Punkte bestimmten Bildkoordinaten; diese werden mit `RowThresh` und `ColumnThresh` zurückgeliefert.

Durchstößt das Grauwertprofil an mehreren Stellen den übergebenen Schwellwert, kann mit dem Parameter `Select` ausgewählt werden, ob nur der erste (**’first’**), nur der letzte (**’last’**), der erste und der letzte (**’first_last’**) oder alle (**’all’**) Punkte zurückgeliefert werden sollen. In den letzten beiden Fällen enthält `Distance` die Abstände zwischen benachbarten Punkten.

Das zugrundegelegte Grauwertprofil wird durch Mittelung der Grauwerte entlang all der Liniensegmente gebildet, die wie folgt durch das Measure-Rechteck definiert werden:

1. Die Liniensegmente stehen senkrecht zur Hauptachse des Rechtecks,
2. sie haben einen ganzzahligen Abstand zum Rechteckmittelpunkt,
3. sie werden durch das Rechteck begrenzt.

Für jedes Liniensegment werden die Grauwerte aller Punkte, die einen ganzzahligen Abstand zur Hauptachse des Rechtecks besitzen, gemittelt. Durch Verschiebung und Drehung des Measure-Rechtecks gegenüber den Bildkoordinaten wird das Eingabebild `Image` im Allgemeinen an Subpixel-Positionen abgetastet.

Da für die Bildabtastung einige Berechnungen erforderlich sind, die in verschiedenen Operatoren Verwendung finden können und darüber hinaus mit einem gewissen Rechenaufwand verbunden sind, erfolgen sie losgelöst von den übrigen Berechnungen mit Hilfe des Operators `gen_measure_rectangle2`. Dort kann auch eine Interpolationsmethode, die für die Subpixel-Abtastung verwendet werden soll, angegeben werden.

Achtung

`measure_thresh` liefert nur dann brauchbare Ergebnisse, falls die Annahme, daß die Kanten gerade und senkrecht zur Hauptachse des Rechtecks sind, erfüllt ist. `measure_thresh` sollte daher z.B. nicht dazu verwendet werden, Kanten von gekrümmten Objekten zu extrahieren. Weiterhin sollte der Benutzer darauf achten, daß das Rechteck möglichst senkrecht zu den Bildkanten liegt.

Parameter

- ▷ **Image** (input_object) . . . . . singlechannel-image ⇝ *Hobject* : byte
  Eingabebild.
- ▷ **MeasureHandle** (input_control) . . . . . measure_id ⇝ *integer*
  Handle des Measure-Objekts.
- ▷ **Sigma** (input_control) . . . . . number ⇝ *real*
  Sigma der Gaußglättung.
  Defaultwert : 1.0
  Wertevorschläge : `Sigma` ∈ {0.0, 0.4, 0.6, 0.8, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 7.0, 10.0}
  Typischer Wertebereich : 0.4 ≤ `Sigma` ≤ 100 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : `Sigma` ≥ `0.0`
- ▷ **Threshold** (input_control) . . . . . number ⇝ *real*
  Schwellwert.
  Defaultwert : 128.0
  Typischer Wertebereich : 0 ≤ `Threshold` ≤ 255 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 0.5
- ▷ **Select** (input_control) . . . . . string ⇝ *string*
  Auswahl der Ergebnispunkte.
  Defaultwert : 'all'
  Werteliste : `Select` ∈ {'all', 'first', 'last', 'first_last'}
- ▷ **RowThresh** (output_control) . . . . . point.y-array ⇝ *real*
  Zeilenkoordinaten der Punkte mit Schwellwert.
- ▷ **ColumnThresh** (output_control) . . . . . point.x-array ⇝ *real*
  Spaltenkoordinaten der Punkte mit Schwellwert.
- ▷ **Distance** (output_control) . . . . . real-array ⇝ *real*
  Abstand zwischen aufeinanderfolgenden Punkten.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `measure_thresh` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`measure_thresh` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_measure_rectangle2`

Mögliche Nachfolgerfunktionen

`close_measure`

Alternativen

`measure_pos`, `edges_sub_pix`, `measure_pairs`

Modul

Sub-pixel operators

12.12 OCR

append_ocr_trainf (Character, Image : : Class, FileName :)

Anhängen von Trainingszeichen an eine Datei.

`append_ocr_trainf` dient zum Vorbereiten des Trainings mit `trainf_ocr_class_box`. Hierzu werden Regionen, die Buchstaben darstellen, mit ihren Grauwerten (Region und Pixel) und dem zugehörigen Klassennamen auf Datei geschrieben. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in `Character` muß der zugehörige Name (Klasse) in `Class` übergeben werden. Die Grauwerte übergibt man in `Image`. Der Dateiname ist frei wählbar. Im Gegensatz zu `write_ocr_trainf` werden die Zeichen an die Datei angefügt. Falls die Datei zum Zeitpunkt des Aufrufs noch nicht existiert, wird die Datei neu angelegt.

Parameter

- ▷ **Character** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu trainierende Zeichen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Grauwerte für die Zeichen.
- ▷ **Class** (input_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der Trainingsdatei.
  Defaultwert : ”train_ocr”

Beispiel

```
char       name[128];
char       class[128];

read_image(&Image,"character.tiff");
bin_threshold(Image,&Dark);
connection(Dark,&Character);
count_obj(Character,&num);
create_tuple(&Class,num);
open_window(0,0,-1,-1,0,"","",&WindowHandle);
set_color(WindowHandle,"red");
for (i=0; i<num; i++) {
  select_obj(Character,&SingleCharacter,i);
  clear_window(WindowHandle);
  disp_region(SingleCharacter,WindowHandle);
  printf("class of character %d ?\n",i);
  scanf("%s\n",class);
  append_ocr_trainf(Character,Image,name,class);
}
```

Ergebnis

Sind die Parameter korrekt, dann liefert `append_ocr_trainf` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`append_ocr_trainf` wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`threshold`, `connection`, `create_ocr_class_box`, `read_ocr`

Mögliche Nachfolgerfunktionen

`trainf_ocr_class_box`, `info_ocr_class_box`, `write_ocr`, `do_ocr_multi`, `do_ocr_single`

Alternativen

`write_ocr_trainf`, `write_ocr_trainf_image`

Modul

Optical character recognition

close_all_ocrs (: : :)

Löschen aller OCR-Klassifikatoren.

`close_all_ocrs` löscht alle OCR-Klassifikatoren und stellt den Speicherplatz wieder zur Verfügung. Die gesamte gelernte Information geht verloren.

Achtung

Da alle Klassifikatoren geschlossen werden, sind alle vorhandenen Handle ungültig.

Ergebnis

Gelingt es, die OCR-Klassifikatoren zu schließen, liefert `close_all_ocrs` den Wert 2 (H_MSG_TRUE). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_all_ocrs` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Alternativen

`close_ocr`

Modul

Optical character recognition

close_ocr (: : OcrHandle :)

Freigabe des Speichers eines OCR-Klassifikators.

`close_ocr` gibt den Speicher des Klassifikators mit der Nummer `OcrHandle` frei. Damit werden alle zugehörigen Daten gelöscht. Diese können gegebenenfalls mit `write_ocr` vorher gesichert werden. Die Nummer `OcrHandle` ist nach dem Aufruf ungültig. Sie kann aber später vom System für neue Klassifikatoren wieder verwendet werden.

Achtung

Alle Daten des Klassifikators werden im Hauptspeicher (nicht auf der Platte) gelöscht.

Parameter

- **OcrHandle** (input_control) . . . . . ocr ⇝ *integer*
  ID des zu löschenden OCR-Klassifikators.

Ergebnis

Falls `OcrHandle` gültig ist, liefert `close_ocr` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`close_ocr` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`write_ocr_trainf`

Mögliche Nachfolgerfunktionen

`read_ocr`

Modul

Optical character recognition

concat_ocr_trainf (: : SingleFiles, ComposedFile :)

Zusammenfügen mehrere Trainings-Dateien.

`concat_ocr_trainf` speichert alle Zeichen, die in den Dateien von `SingleFiles` abgelegt sind, in einer neuen Datei mit Namen `ComposedFile` ab.

Parameter

- **SingleFiles** (input_control) . . . . . filename-array ⇝ *string*
  Name der einzelnen Trainingsdateien.
  Defaultwert : ”
- **ComposedFile** (input_control) . . . . . filename ⇝ *string*
  Name der zusammengesetzten Trainingsdatei.
  Defaultwert : ’all_characters’

Ergebnis

Sind die Parameter korrekt, dann liefert `concat_ocr_trainf` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`concat_ocr_trainf` wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`write_ocr_trainf`, `append_ocr_trainf`

Mögliche Nachfolgerfunktionen

`trainf_ocr_class_box`, `info_ocr_class_box`, `write_ocr`, `do_ocr_multi`, `do_ocr_single`

Modul

Optical character recognition

create_ocr_class_box (: : WidthPattern, HeightPattern, Interpolation, Features, Character : OcrHandle)

Erzeugen eines neues OCR-Klassifikators.

`create_ocr_class_box` erzeugt einen neuen OCR-Klassifikator. Zur Beschreibung des Klassifikators siehe Operator `learn_class_box`. Dieser Klassifikator muß dann mit `traind_ocr_class_box` oder `trainf_ocr_class_box` trainiert werden.

Die Parameter `WidthPattern` und `HeightPattern` geben die Größe der Eingabeschicht des Klassifikators an. Diese Größe wird bei den Merkmalen **’projection_horizontal’**, **’projection_vertical’**, **’pixel’** und **’pixel_invar’** verwendet, um das zu erkennende Zeichen auf eine Standardgröße zu transformieren. Um so größer die Standardgröße ist, um so mehr Zeichen können (mit Grauwertmerkmalen) unterschieden werden. Dabei steigt aber auch die Zeit für das Training (und die Anzahl der Lernstichproben) und die Zeit für die Erkennung an. Der Parameter `Interpolation` gibt den Interpolationsmodus für die Anpassung der Zeichen im Bild auf die Größe im Klassifikator an. Dieser Parameter ist bei der Prozedur `affine_trans_image` beschrieben. Der Wert **0** erzeugt dieselbe Interpolation wie **’none’** in `affine_trans_image`, d.h. es erfolgt keine Interpolation. Für **1** erhält man dasselbe Verhalten wie **’constant’** in `affine_trans_image`, d.h. es erfolgt eine ungewichtete Interpolation zwischen den Grauwerten. Der Wert **2** erzeugt dieselbe Interpolation wie **’weighted’**, d.h. es erfolgt eine gewichtete Interpolation zwischen den Grauwerten. Der Parameter `Interpolation` muß so gewählt werden, daß bei der Skalierung der Zeichen auf die Standardgröße keine Aliasing-Effekte auftreten. In der Praxis bedeutet dies, daß `Interpolation` = **1** gewählt werden sollte, außer die Zeichen werden stark verkleinert. In diesem Fall sollte `Interpolation` = **2** gewählt werden. `Interpolation` = **0** sollte nur gewählt werden, falls die Zeichen nicht skaliert werden.

Der Parameter `Character` legt alle zu erkennenden Zeichen fest. Im Normalfall bestehen die übergebenen Strings aus einem Zeichen (z.B. Alphabet). Es können aber auch Strings mit beliebiger Länge gelernt werden. Die Anzahl unterscheidbarer Zeichen (Anzahl Strings in `Character`) ist auf **2048** beschränkt.

Mit dem Parameter `Features` können neben den Grauwerten zusätzliche Merkmale zur Erkennung der Buchstaben ausgewählt werden. Durch Angabe von **’default’** werden die Merkmale **’ratio’** und **’pixel_invar’** festegelegt.

Es stehen folgende Merkmale zur Verfügung

’ratio’ Seitenverhältnis des Zeichens.

’width’ Breite des Zeichens (nicht skalierungsinvariant).

’height’ Höhe des Zeichens(nicht skalierungsinvariant).

’zoom_factor’ Größenunterschied zwischen aktuellem Zeichen und den Werten von `WidthPattern` und `HeightPattern` (nicht skalierungsinvariant).

’foreground’ Relativer Anteil der Vordergrund-Pixeln.

’foreground_grid_9’ Relative Anteil der Vordergrund-Pixel in einem 3×3 Raster innerhalb des umschließenden Rechtecks des Zeichens.

’foreground_grid_16’ Relative Anteil der Vordergrund-Pixel in einem 4×4 Raster innerhalb des umschließenden Rechtecks des Zeichens.

’anisometry’ Formmerkmal Anisometry.

’compactness’ Formmerkmal Compactness.

’convexity’ Formmerkmal Convexity.

'moments_region_2nd_invar' Normierte 2. geometrischen Momente der Region. Siehe auch `moments_region_2nd_invar`.

'moments_region_2nd_rel_invar' Normierte 2. relativen geometrischen Momente der Region. Siehe auch `moments_region_2nd_rel_invar`.

'moments_region_3rd_invar' Normierte 3. geometrischen Momente der Region. Siehe auch `moments_region_3rd_invar`.

'moments_central' Normierte zentrale geometrischen Momente der Region. Siehe auch `moments_region_central`.

'phi' Orientierung (Winkel) des Zeichens.

'num_connect' Anzahl der Zusammenhangskomponenten.

'num_holes' Anzahl der Hohlflächen.

'projection_horizontal' Horizontal Projektion der Grauwert.

'projection_horizontal_invar' Horizontal Projektion der Grauwerte die maximal skaliert sind.

'projection_vertical' Vertikale Projektion der Grauwert.

'projection_vertical_invar' Vertikale Projektion der Grauwerte die maximal skaliert sind.

'cooc' Werte der binÌren Cooccurrence Matrix.

'moments_gray_plane' Normierte Grauwertmomente und die Winkel der Grauwertebene.

'num_runs' Anzahl Sehnen der Region normiert auf die Fläche.

'chord_histo' Häufigkeit der Sehnen pro Zeile.

'pixel' Grauwert der Zeichens.

'pixel_invar' Grauwerte der Zeichens mit automatischer maximaler Grauwertspreizung.

Parameter

▷ **`WidthPattern`** (input_control) . . . . . integer ⇝ *integer*
Breite des Eingabenetzes.
Defaultwert : 8
Wertevorschläge : `WidthPattern` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich : 1 ≤ `WidthPattern` ≤ 100

▷ **`HeightPattern`** (input_control) . . . . . integer ⇝ *integer*
Höhe des Eingabenetzes.
Defaultwert : 10
Wertevorschläge : `HeightPattern` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 16, 20}
Typischer Wertebereich : 1 ≤ `HeightPattern` ≤ 100

▷ **`Interpolation`** (input_control) . . . . . integer ⇝ *integer*
Interpolationsmodus bei der Skalierung der Zeichen.
Defaultwert : 1
Werteliste : `Interpolation` ∈ {0, 1, 2}

▷ **`Features`** (input_control) . . . . . string(-array) ⇝ *string*
Zusätzliche Merkmale.
Defaultwert : 'default'
Werteliste : `Features` ∈ {'default', 'zoom_factor', 'ratio', 'width', 'height', 'foreground', 'foreground_grid_9', 'foreground_grid_16', 'anisometry', 'compactness', 'convexity', 'moments_region_2nd_invar', 'moments_region_2nd_rel_invar', 'moments_region_3rd_invar', 'moments_central', 'phi', 'num_connect', 'num_holes', 'projection_horizontal', 'projection_vertical', 'projection_horizontal_invar', 'projection_vertical_invar', 'chord_histo', 'num_runs', 'pixel', 'pixel_invar', 'cooc', 'moments_gray_plane'}

▷ **`Character`** (input_control) . . . . . string-array ⇝ *string*
Alle Zeichen des Zeichensatzes.
Defaultwert : '['a','b','c']'

▷ **`OcrHandle`** (output_control) . . . . . ocr ⇝ *integer*
ID des erzeugten OCR-Klassifikators.

Ergebnis

Sind die Parameter korrekt, dann liefert `create_ocr_class_box` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`create_ocr_class_box` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`reset_obj_db`

Mögliche Nachfolgerfunktionen

`traind_ocr_class_box`, `trainf_ocr_class_box`, `info_ocr_class_box`, `write_ocr`, `ocr_change_char`

Siehe auch

`affine_trans_image`, `ocr_change_char`, `moments_region_2nd_invar`, `moments_region_2nd_rel_invar`, `moments_region_3rd_invar`, `moments_region_central`

Modul

Optical character recognition

do_ocr_multi (Character, Image : : OcrHandle : Class, Confidence)

Klassifikation von Zeichen.

`do_ocr_multi` ordnet jedem `Character` (Buchstabe) eine Klasse zu. Dabei werden (bei Graustufenmerkmalen) alle Grauwerte aus den umschließenden Rechtecken der Regionen verwendet. Die Grauwerte werden aus dem Parameter `Image` entnommen. Für jeden Buchstaben wird die zugehörige Klasse in `Class` und ein Konfidenzwert in `Confidence` zurückgegeben. Der Konfidenzwert beschreibt die Ähnlichkeit zwischen dem übergebenen und dem zugeordneten Zeichen.

Parameter

- **Character** (input_object) ..... region(-array) ⇝ *Hobject*
  Zu erkennende Zeichen.
- **Image** (input_object) ..... image ⇝ *Hobject*
  Grauwerte für Zeichen.
- **OcrHandle** (input_control) ..... ocr ⇝ *integer*
  ID des OCR Klassifikators.
- **Class** (output_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
  Parameteranzahl : `Class` = `Character`
- **Confidence** (output_control) ..... real(-array) ⇝ *real*
  Konfidenzwerte der Zeichen.
  Parameteranzahl : `Confidence` = `Character`

Ergebnis

If the input parameters are set correctly, the operator `do_ocr_multi` returns the value 2 (H_MSG_TRUE). Otherwise an exception will be raised.

Parallelisierungsinformation

`do_ocr_multi` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`traind_ocr_class_box`, `trainf_ocr_class_box`, `read_ocr`, `connection`, `sort_region`

Alternativen

`do_ocr_single`

Siehe auch

`write_ocr`

Modul

Optical character recognition

do_ocr_single (Character, Image : : OcrHandle : Classes, Confidences)

Klassifikation von einem Zeichen.

do_ocr_single ordnet dem Character (Buchstabe) Klassen zu. Dabei werden (bei Graustufenmerkmalen) alle Grauwerte aus den umschließenden Rechtecken der Regionen verwendet. Diese werden aus dem Parameter Image entnommen. Für jeden Buchstaben werden die beiden Klassen mit den höchsten Konfidenzen in Classes zurückgegeben. Die zugehörigen Konfidenzen werden in Confidences übergeben. Der Konfidenzwert beschreibt die Ähnlichkeit zwischen dem übergebenen und dem zugeordneten Zeichen.

Parameter

- ▷ **Character** (input_object) ..... region ⇝ *Hobject*
  Zu erkennendes Zeichen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Grauwerte für Zeichen.
- ▷ **OcrHandle** (input_control) ..... ocr ⇝ *integer*
  ID des OCR Klassifikator.
- ▷ **Classes** (output_control) ..... string-array ⇝ *string*
  Klassen (Name) der Zeichen.
  Parameteranzahl : 2
- ▷ **Confidences** (output_control) ..... real-array ⇝ *real*
  Konfidenzwerte der Zeichen.
  Parameteranzahl : 2

Ergebnis

Sind die Eingabeparameter korrekt besetzt, dann liefert do_ocr_single den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

do_ocr_single ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

traind_ocr_class_box, trainf_ocr_class_box, read_ocr, connection, sort_region

Alternativen

do_ocr_multi

Siehe auch

write_ocr

Modul

Optical character recognition

info_ocr_class_box (: : OcrHandle : WidthPattern, HeightPattern, Interpolation, WidthMaxChar, HeightMaxChar, Features, Characters)

Informationen über einen OCR Klassifikator.

info_ocr_class_box gibt alle einige Informationen über einen OCR Klassifikator aus. Die Parameter entsprechen denen von create_ocr_class_box. Die Parameter WidthMaxChar und HeightMaxChar geben die Ausdehnung des größten trainierten Zeichens an. Diese Werte können zur Steuerung der Segmentation verwendet werden.

Parameter

- ▷ **OcrHandle** (input_control) ..... ocr ⇝ *integer*
  ID des OCR Klassifikators.
- ▷ **WidthPattern** (output_control) ..... integer ⇝ *integer*
  Breite der skalierten Zeichen.
- ▷ **HeightPattern** (output_control) ..... integer ⇝ *integer*
  Höhe der skalierten Zeichen.

- ▷ **Interpolation** (output_control) ..... integer ⇝ *integer*
  Interpolationsmodus beim Skalieren der Zeichen.
- ▷ **WidthMaxChar** (output_control) ..... integer ⇝ *integer*
  Breite des größten trainierten Zeichens.
- ▷ **HeightMaxChar** (output_control) ..... integer ⇝ *integer*
  Höhe des größten trainierten Zeichens.
- ▷ **Features** (output_control) ..... string-array ⇝ *string*
  Verwendete Merkmale.
- ▷ **Characters** (output_control) ..... string-array ⇝ *string*
  Alle Zeichen des Zeichensatzes.

Ergebnis

info_ocr_class_box liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

info_ocr_class_box ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

read_ocr, create_ocr_class_box

Mögliche Nachfolgerfunktionen

write_ocr

Modul

Optical character recognition

ocr_change_char (: : OcrHandle, Character :)

Neue Umsetzungstabelle für Zeichen festlegen.

ocr_change_char setzt bei dem Klassifikator eine neue Zuordnungstabelle für die Zeichen. Die Anzahl der Strings in Character muß genauso groß sein wie bei dem Netz OcrHandle. ocr_change_char kann folgendermaßen verwendet werden um einen Zeichensatz zu erweitern: Beim Erzeugen eines Netzes (create_ocr_class_box) werden mehr Zeichen angegeben als zunächst benötigt. Es bleiben die letzten n Zeichen zunächst unbenutzt. Wenn man mehr Zeichen benötigt, dann werden mit ocr_change_char diese unbenutzten Zeichen besetzt und danach trainiert.

Parameter

- ▷ **OcrHandle** (input_control) ..... ocr ⇝ *integer*
  ID des zu ändernden OCR-Netzes.
- ▷ **Character** (input_control) ..... string-array ⇝ *string*
  Neue Zeichenzuordnung.
  Defaultwert : ’[’a’,’b’,’c’]’

Ergebnis

Stimmt die Anzahl der Zeichen in Character mit denen des Netzes überein, dann liefert ocr_change_char den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

ocr_change_char wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

read_ocr

Mögliche Nachfolgerfunktionen

do_ocr_multi, do_ocr_single, write_ocr

Modul

Optical character recognition

ocr_get_features (Character : : OcrHandle : FeatureVector)

Merkmale eines Zeichens abfragen.

ocr_get_features bestimmt die Merkmale für das übergebene Zeichen. Die Art und Anzahl der Merkmale werden durch den Klassifikator OcrHandle festgelegt. Es sind die gleichen Werte, die bei Operatoren wie traind_ocr_class_box oder trainf_ocr_class_box intern verwendet werden.

Parameter

- ▷ **Character** (input_object) ..... image ⇝ *Hobject*
  Zu trainierende Zeichen.
- ▷ **OcrHandle** (input_control) ..... ocr ⇝ *integer*
  ID des gewünschten OCR-Klassifikators.
- ▷ **FeatureVector** (output_control) ..... real-array ⇝ *real*
  Merkmalswerte.

Ergebnis

Sind die Parameter korrekt, dann liefert ocr_get_features den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

ocr_get_features ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

create_ocr_class_box, read_ocr, reduce_domain, threshold, connection

Mögliche Nachfolgerfunktionen

learn_class_box

Siehe auch

trainf_ocr_class_box, traind_ocr_class_box

Modul

Optical character recognition

read_ocr (: : FileName : OcrHandle)

OCR Klassifikator aus Datei einlesen.

read_ocr liest einen OCR Klassifikator aus der Datei FileName. Die Datei wird dabei im Verzeichnis ($HALCONROOT/ocr/) und im aktuellen Verzeichnis gesucht. Sind bereits zuviele Klassifikatoren geladen, erfolgt eine Fehlermeldung.

Parameter

- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der Datei mit dem OCR Klassifikator.
  Defaultwert : ’testnet’
- ▷ **OcrHandle** (output_control) ..... ocr ⇝ *integer*
  ID des eingelesenen OCR Klassifikators.

Ergebnis

Ist die angegebene Datei vorhanden und der Format korrekt, dann liefert read_ocr den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_ocr wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

reset_obj_db

Mögliche Nachfolgerfunktionen

do_ocr_multi, do_ocr_single, traind_ocr_class_box, trainf_ocr_class_box

Siehe auch

write_ocr, do_ocr_multi, traind_ocr_class_box, trainf_ocr_class_box

Modul

Optical character recognition

read_ocr_trainf (: Characters : TrainFileNames : CharacterNames)

Einlesen von Trainingszeichen für OCR als Bilder.

read_ocr_trainf liest alle Zeichen aus den angegebenen Trainings-Dateien ein und wandelt sie in Bilder um. Die Definitionsbereiche werden auf die Vordergrundregion der Zeichen (gemäß write_ocr_trainf) beschränkt. Die Namen der Zeichen werden im Parameter CharacterNames zurückgegeben. Es kann mehr als eine Datei eingelesen werden, dabei werden die Dateien entsprechend der Ordnung der Dateinamen abgearbeitet.

Parameter

- ▷ **Characters** (output_object) ..... image-array ⇝ *Hobject* : byte
  Eingelesene Zeichen.
- ▷ **TrainFileNames** (input_control) ..... filename.named(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert : ”
- ▷ **CharacterNames** (output_control) ..... string-array ⇝ *string*
  Bezeichnung der eingelesenen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_trainf den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_ocr_trainf ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

write_ocr_trainf

Mögliche Nachfolgerfunktionen

disp_image, select_obj, zoom_image_size

Alternativen

read_ocr_trainf_select

Siehe auch

trainf_ocr_class_box

Modul

Optical character recognition

read_ocr_trainf_names (: : TrainFileNames : CharacterNames, CharacterCount)

Abfrage welche Zeichen in Trainings-Dateien abgelegt sind.

read_ocr_trainf_names bestimmt die Namen aller Zeichen und deren Anzahl in den angegebenen Trainings-Dateien.

Parameter

- ▷ **TrainFileNames** (input_control) ..... filename.named(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert : ”
- ▷ **CharacterNames** (output_control) ..... string(-array) ⇝ *string*
  Bezeichnung der eingelesenen Zeichen.
- ▷ **CharacterCount** (output_control) ..... integer(-array) ⇝ *integer*
  Anzahl der jeweiligen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_trainf_names den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_ocr_trainf_names ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

write_ocr_trainf

Siehe auch

trainf_ocr_class_box

Modul

Optical character recognition

read_ocr_trainf_select (: Characters : TrainFileNames, SearchNames : FoundNames)

Einlesen von ausgewählten Trainingszeichen für OCR als Bilder.

read_ocr_trainf_select liest die in SearchNames angegebenen Zeichen aus den Trainings-Dateien ein und wandelt sie in Bilder um. Es arbeitet analog zu read_ocr_trainf nur können die zu extrahierenden Zeichen vorgegeben werden.

Parameter

- ▷ **Characters** (output_object) ..... image-array ⇝ *Hobject* : byte
  Eingelesene Zeichen.
- ▷ **TrainFileNames** (input_control) ..... filename.named(-array) ⇝ *string*
  Namen der Trainingsdateien.
  Defaultwert : ”
- ▷ **SearchNames** (input_control) ..... string(-array) ⇝ *string*
  Namen der zu extrahierenden Zeichen.
  Defaultwert : ’0’
- ▷ **FoundNames** (output_control) ..... string(-array) ⇝ *string*
  Bezeichnung der eingelesenen Zeichen.

Ergebnis

Sind die Parameterwerte korrekt, dann liefert read_ocr_trainf_select den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_ocr_trainf_select ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

write_ocr_trainf

Mögliche Nachfolgerfunktionen

disp_image, select_obj, zoom_image_size

Alternativen

read_ocr_trainf

Siehe auch

trainf_ocr_class_box

Modul

Optical character recognition

testd_ocr_class_box (Character, Image : : OcrHandle, Class : Confidence)

Trainieren eines OCR-Klassifikators durch Eingabe von Regionen.

testd_ocr_class_box testet die Güte der Klassenzugehörigkeit von Zeichen. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muß der zugehörige Name (Klasse) Class übergeben werden. Die Grauwerte übergibt man in Image. Confidence gibt Auskunft über die Zugehörigkeit zu einer (beliebig wählbaren) Klasse.

Parameter

- ▷ **Character** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Zu trainierende Zeichen.
- ▷ **Image** (input_object) . . . . . image ⇝ *Hobject*
  Grauwerte für die Zeichen.
- ▷ **OcrHandle** (input_control) . . . . . ocr ⇝ *integer*
  ID des gewünschten OCR-Klassifikators.
- ▷ **Class** (input_control) . . . . . string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
  Defaultwert : ”a”
- ▷ **Confidence** (output_control) . . . . . real(-array) ⇝ *real*
  Konfidenz mit der das Zeichen der Klasse zugehört.

Ergebnis

Sind die Parameter korrekt, dann liefert `testd_ocr_class_box` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`testd_ocr_class_box` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`read_ocr`, `trainf_ocr_class_box`, `traind_ocr_class_box`

Modul

Optical character recognition

traind_ocr_class_box (Character, Image : : OcrHandle, Class : AvgConfidence)

Trainieren eines OCR-Klassifikators durch Eingabe von Regionen.

`traind_ocr_class_box` trainiert den Klassifikator direkt über die eingegebenen Regionen in einem Bild. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in `Character` muß der zugehörige Name (Klasse) `Class` übergeben werden. Die Grauwerte übergibt man in `Image`. `AvgConfidence` gibt Auskunft über den Erfolg des Trainings: Er enthält die mittlere Konfidenz der trainierten Zeichen bei einer Reklassifikation. Dabei werden falsch klassifizierte Zeichen mit Konfidenz 0 bewertet.

Parameter

- ▷ **Character** (input_object) . . . . . region(-array) ⇝ *Hobject*
  Zu trainierende Zeichen.
- ▷ **Image** (input_object) . . . . . image ⇝ *Hobject*
  Grauwerte für die Zeichen.
- ▷ **OcrHandle** (input_control) . . . . . ocr ⇝ *integer*
  ID des gewünschten OCR-Klassifikators.
- ▷ **Class** (input_control) . . . . . string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
  Defaultwert : ”a”
- ▷ **AvgConfidence** (output_control) . . . . . real ⇝ *real*
  Mittlere Konfidenz bei einer Reklassifikation der trainierten Zeichen.

Ergebnis

Sind die Parameter korrekt, dann liefert `traind_ocr_class_box` den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`traind_ocr_class_box` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`create_ocr_class_box`, `read_ocr`

Mögliche Nachfolgerfunktionen

`traind_ocr_class_box`, `write_ocr`, `do_ocr_multi`, `do_ocr_single`

Alternativen

trainf_ocr_class_box

Modul

Optical character recognition

trainf_ocr_class_box (: : OcrHandle, FileName : AvgConfidence)

Trainieren eines OCR-Klassifikators mit einer Trainingsdatei.

trainf_ocr_class_box trainiert den Klassifikator OcrHandle über die angegebenen Trainingsdateien. Es können beliebig viele Dateien angegeben werden. AvgConfidence gibt Auskunft über den Erfolg des Trainings: Er enthält die mittlere Konfidenz der trainierten Zeichen bei einer Kontroll-Klassifikation. Dabei werden falsch klassifizierte Zeichen mit Konfidenz 0 bewertet.

Achtung

Die Namen der Buchstaben in der Datei müssen zu dem Netz passen.

Parameter

- **OcrHandle** (input_control) . . . . . ocr ⇝ *integer*
  ID des gewünschten OCR-Netzes.
- **FileName** (input_control) . . . . . filename(-array) ⇝ *string*
  Name(n) der Trainingsdatei(en).
  Defaultwert : "train_ocr"
- **AvgConfidence** (output_control) . . . . . real ⇝ *real*
  Mittlere Konfidenz bei einer Reklassifikation der trainierten Zeichen.

Ergebnis

Ist der Dateiname korrekt und die Daten passen zu dem Netz, dann liefert trainf_ocr_class_box den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

trainf_ocr_class_box wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

create_ocr_class_box, read_ocr

Mögliche Nachfolgerfunktionen

traind_ocr_class_box, write_ocr, do_ocr_multi, do_ocr_single

Alternativen

traind_ocr_class_box

Modul

Optical character recognition

write_ocr (: : OcrHandle, FileName :)

OCR Klassifikator in Datei schreiben.

write_ocr schreibt den OCR Klassifikator OcrHandle in die Datei FileName. Da die Daten des Klassifikators nach Beendigung des Programms verloren gehen, muß man nach dem Training die Daten sichern, wenn man sie in einem späteren Programmlauf wieder verwenden will. Die Daten können später wieder mit read_ocr eingelesen werden. Die Extension wird automatisch an FileName angehängt.

Achtung

Die Ausgabedatei FileName muß ohne Extension angegeben werden.

Parameter

- **OcrHandle** (input_control) . . . . . ocr ⇝ *integer*
  ID des OCR Klassifikators.

▷ **FileName** (input_control) ..... filename ⇝ *string*
Name der Datei für den OCR Klassifikator (ohne Extension).
Defaultwert : 'my_ocr'

Ergebnis

Ist OcrHandle gültig und kann die angegebene Datei geschrieben werden, dann liefert write_ocr den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_ocr ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

traind_ocr_class_box, trainf_ocr_class_box

Mögliche Nachfolgerfunktionen

do_ocr_multi, do_ocr_single

Siehe auch

read_ocr, do_ocr_multi, traind_ocr_class_box, trainf_ocr_class_box

Modul

Optical character recognition

write_ocr_trainf (Character, Image : : Class, FileName :)

Abspeichern von Trainingszeichen in eine Datei.

write_ocr_trainf dient zum Vorbereiten des Trainings mit trainf_ocr_class_box. Hierzu werden Regionen, die Buchstaben darstellen, mit ihren Grauwerten (Region und Pixel) und dem zugehörigen Klassennamen auf Datei geschrieben. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muß der zugehörige Name (Klasse) in Class übergeben werden. Die Grauwerte übergibt man in Image. Der Dateiname ist frei wählbar.

Parameter

▷ **Character** (input_object) ..... region(-array) ⇝ *Hobject*
Zu trainierende Zeichen.

▷ **Image** (input_object) ..... image ⇝ *Hobject*
Grauwerte für die Zeichen.

▷ **Class** (input_control) ..... string(-array) ⇝ *string*
Klasse (Name) der Zeichen.

▷ **FileName** (input_control) ..... filename ⇝ *string*
Name der Trainingsdatei.
Defaultwert : "train_ocr"

Ergebnis

Sind die Parameter korrekt, dann liefert write_ocr_trainf den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_ocr_trainf ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, connection, create_ocr_class_box, read_ocr

Mögliche Nachfolgerfunktionen

trainf_ocr_class_box, info_ocr_class_box, write_ocr, do_ocr_multi, do_ocr_single

Modul

Optical character recognition

write_ocr_trainf_image (Character : : Class, FileName :)

Schreibe Zeichen in eine Trainings-Datei.

write_ocr_trainf_image dient zum Vorbereiten des Trainings mit trainf_ocr_class_box. Hierzu werden Regionen, die Buchstaben darstellen, mit ihren Grauwerten (Region und Pixel) und dem zugehörigen Klassennamen in eine Datei geschrieben. Es können beliebig viele Regionen aus einem Bild übergeben werden. Für jedes Zeichen (Region) in Character muß der zugehörige Name (Klasse) in Class übergeben werden. Der Dateiname ist frei wählbar. Im Gegensatz zu write_ocr_trainf wird pro Zeichen ein Bild übergeben, dessen Definitionsbereich den Vordergrund des Zeichens angibt.

Parameter

- ▷ **Character** (input_object) ..... image(-array) ⇝ *Hobject*
  Zu trainierende Zeichen.
- ▷ **Class** (input_control) ..... string(-array) ⇝ *string*
  Klasse (Name) der Zeichen.
- ▷ **FileName** (input_control) ..... filename ⇝ *string*
  Name der Trainingsdatei.
  Defaultwert : "train_ocr"

Ergebnis

Sind die Parameter korrekt, dann liefert write_ocr_trainf_image den Wert 2 (H_MSG_TRUE). Andernfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_ocr_trainf_image ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

threshold, connection, create_ocr_class_box, read_ocr

Mögliche Nachfolgerfunktionen

trainf_ocr_class_box, info_ocr_class_box, write_ocr, do_ocr_multi, do_ocr_single

Alternativen

write_ocr_trainf, append_ocr_trainf

Modul

Optical character recognition

12.13 OCV

close_all_ocvs (: : :)

Freigeben aller OCV-Tools.

close_all_ocvs gibt *alle* OCV-Tools frei, die mit create_ocv_proj oder read_ocv angelegt wurden. Alle Handle sind nach dem Aufruf ungültig.

Achtung

Dieser Operator ist nur für den internen Gebrauch vorgesehen (wie z.B. in HDevelop bei einem Programm-Reset).

Ergebnis

close_all_ocvs liefert immer den Wert 2 (H_MSG_TRUE)

Parallelisierungsinformation

close_all_ocvs wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

read_ocv, create_ocv_proj

Alternativen

close_ocv

Modul

Optical character verification

close_ocv (: : OCVHandle :)

Freigeben eines OCV-Tools.

close_ocv gibt ein OCV-Tool frei, das mit create_ocv_proj oder read_ocv angelegt wurde. Das Handle ist nach dem Aufruf ungültig.

Parameter

▷ **OCVHandle** (input_control) ..... ocv ⇝ *integer*
Handle des OCV-Tools, das freigegeben werden soll.

Beispiel

```
read_ocv("ocv_file",&ocv_handle);
for (i=0; i<1000; i++)
{
  grab_image_async(&Image,fg_handle,-1);
  reduce_domain(Image,ROI,&Pattern);
  do_ocv_simple(Pattern,ocv_handle,"A",
                "true","true","false","true",10,
                &Quality);
}
close_ocv(ocv_handle);
```

Ergebnis

close_ocv liefert den Wert 2 (H_MSG_TRUE), falls das Handle gültig ist. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

close_ocv wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

read_ocv, create_ocv_proj

Siehe auch

close_ocr

Modul

Optical character verification

create_ocv_proj (: : PatternNames : OCVHandle)

Erzeugen eines neuen OCV-Tools basierend auf Grauwertprojektionen.

create_ocv_proj erzeugt ein neues OCV-Tool. Dieses kann danach mit “Gut-Zeichen” trainiert werden. Hierzu steht der Operator traind_ocv_proj zur Verfügung. Dieser Operator wird daher normalerweise im Anschluß an create_ocv_proj aufgerufen.

Das Verfahren des Objektvergleiches basiert auf den Grauwertprojektionen. Hierzu werden beim Training und beim Vergleich die Grauwerte des Musters horizontal und vertikal aufsummiert. Die so erhaltenen Daten werden dann miteinander verglichen. Um so ähnlicher sich die Projektionen sind, um so höher ist die Güte des Objektes.

Übergeben werden die Namen der Zeichen (PatternNames), die trainiert werden sollen. Die Anzahl der Zeichen und auch die zu vergebenden Namen können frei gewählt werden. Im einfachsten Fall wird nur ein Name angegeben. Die Namen werden beim Training und in der Anwendung (Bewertung von Zeichen) zur Identifikation der “Gut-Zeichen” verwendet. Die Namen können nicht geändert werden. Es können mehr Namen vergegeben werden als später trainiert werden. Geschlossen wird das Tool mit dem Operator close_ocv. Hierdurch wird der Speicher wieder freigegeben.

Parameter

▷ **PatternNames** (input_control) ..... string(-array) ⇝ *string*
Liste von Namen der zu lernenden Zeichen.
Defaultwert : ”a”

▷ **OCVHandle** (output_control) ..... ocv ⇝ *integer*
Handle des erzeugten OCV-Tools.

Beispiel

```
create_ocv_proj("A",&ocv_handle);
draw_region(&ROI,window_handle);
reduce_domain(Image,ROI,&Sample);
traind_ocv_proj(Sample,ocv_handle,"A","single");
```

Ergebnis

`create_ocv_proj` liefert den Wert 2 (H_MSG_TRUE), falls die Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`create_ocv_proj` wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Nachfolgerfunktionen

`traind_ocv_proj`, `write_ocv`, `close_ocv`

Alternativen

`read_ocv`

Siehe auch

`create_ocr_class_box`

Modul

Optical character verification

do_ocv_simple (Pattern : : OCVHandle, PatternName, AdaptPos, AdaptSize, AdaptAngle, AdaptGray, Threshold : Quality)

Bewerten eines Zeichens mit einem OCV-Tool.

`do_ocv_simple` bewertet das übergebene Zeichen (`Pattern`). Das Zeichen muß vorher mit dem OCV-Tool trainiert worden sein. Die Region sollte dabei in etwa die gleiche (relative) Ausdehnung und Form haben wie beim Training. Zur Kennzeichnung des Zeichens wird der gleichen Name wie beim Training in `PatternName` übergeben. Über die nächsten vier Parameter kann dann Einfluß auf die automatische Anpassung genommen werden: `AdaptPos` und `AdaptSize` beziehen sich auf die Geometrie des Zeichens. Mit `AdaptPos` wird festgelegt, ob eine Verschiebung des Zeichens kompensiert werden soll. `AdaptSize` legt fest, ob eine Größenveränderung angepaßt werden soll. `AdaptAngle` ist bisher nicht implementiert. Der Parameter `AdaptGray` bezieht sich auf die Grauwertanpassung. Zum einen wird eine additive Grauwertänderung kompensiert, gleichzeitig wird eine multiplikative Grauwertänderung angepaßt.

Der Parameter `Threshold` gibt den Mindestunterschied der Grauwerte an, der als Fehler interpretiert wird. Als Resultat ergibt sich dann der Prozentsatz der fehlerhaften Pixel. Wird mit dem Parameter `Threshold` eine Schwelle kleiner als 0 übergeben, besteht das Resultat stattdessen in der Summe aller Abweichungen normiert auf die Flache der Region.

Das Ergebnis des Operators ist eine Bewertung des Zeichens. Dies ist der Unterschied der Grauwerte zwischen dem aktuellen Zeichen und dem Trainingszeichen nach den entsprechenden automatischen Anpassungen. Der Wert von `Quality` liegt zwischen 0 und 1. Der Wert 1 bedeutet, daß das Zeichen mit dem Muster identisch ist. Der Wert 0 steht für eine sehr große Abweichung.

Parameter

- ▷ **Pattern** (input_object) ..... image(-array) ⇝ *Hobject*
  Zu bewertendes Zeichen.
- ▷ **OCVHandle** (input_control) ..... ocv ⇝ *integer*
  Handle des OCV-Tools.
- ▷ **PatternName** (input_control) ..... string(-array) ⇝ *string*
  Name des Zeichens.
  Defaultwert : ”a”
- ▷ **AdaptPos** (input_control) ..... string ⇝ *string*
  Anpassung an horizontale und vertikale Verschiebung.
  Defaultwert : ”true”
  Werteliste : `AdaptPos` ∈ {”true”, ”false”}

▷ **AdaptSize** (input_control) ..... string ⇝ *string*
Anpassung an horizontale und vertikale Größenveränderung.
Defaultwert : "true"
Werteliste : AdaptSize ∈ {"true", "false"}

▷ **AdaptAngle** (input_control) ..... string ⇝ *string*
Anpassung der Orientierung (noch nicht implementiert).
Defaultwert : "false"
Werteliste : AdaptAngle ∈ {"false"}

▷ **AdaptGray** (input_control) ..... string ⇝ *string*
Anpassung an additive und multiplikative Grauwertänderungen.
Defaultwert : "true"
Werteliste : AdaptGray ∈ {"true", "false"}

▷ **Threshold** (input_control) ..... number ⇝ *real*
Mindestdifferenz der Muster.
Defaultwert : 10
Wertevorschläge : Threshold ∈ {-1, 0, 1, 5, 10, 15, 20, 30, 40, 50, 60, 80, 100, 150}

▷ **Quality** (output_control) ..... real(-array) ⇝ *real*
Bewertung des Zeichens.
Typischer Wertebereich : $0.0 \leq$ Quality ≤ 1.0

Ergebnis

do_ocv_simple liefert den Wert 2 (H_MSG_TRUE), falls das Handle und die Vergleichsdaten gültig sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

do_ocv_simple ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

traind_ocr_class_box, trainf_ocr_class_box, read_ocv, threshold, connection, select_shape

Mögliche Nachfolgerfunktionen

close_ocv

Siehe auch

create_ocv_proj

Modul

Optical character verification

read_ocv (: : FileName : OCVHandle)

Einlesen eines OCV-Tools von Datei.

read_ocv liest ein OCV-Tool von Datei. Der Zustand des Tools ist identisch mit dem Zustand beim Speichern (write_ocv). Nach dem Lesen kann entweder das Training fortgesetzt werden (traind_ocv_proj), sofern dies noch nicht abgeschlossen wurde, oder es kann direkt eine Bewertung von Zeichen ausgeführt werden (do_ocv_simple).

Als Extension der Datei wird '.ocv' verwendet. Falls diese bei dem Dateinamen nicht angegeben ist, wird sie automatisch angefügt.

Parameter

▷ **FileName** (input_control) ..... filename ⇝ *string*
Name der Datei die gelesen werden soll.
Defaultwert : 'test_ocv'

▷ **OCVHandle** (output_control) ..... ocv ⇝ *integer*
Handle des gelesenen OCV-Tools.

Beispiel

```
read_ocv("ocv_file",&ocv_handle);
```

```
for (i=0; i<1000; i++)
{
  grab_image_async(&Image,fg_handle,-1);
  reduce_domain(Image,ROI,&Pattern);
  do_ocv_simple(Pattern,ocv_handle,"A",
                "true","true","false","true",10,
                &Quality);
}
close_ocv(ocv_handle);
```

Ergebnis

read_ocv liefert den Wert 2 (H_MSG_TRUE), falls die Datei korrekt ist. Falls die Datei nicht geöffnet werden kann, liefert read_ocv 5 (H_MSG_FAIL). Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

read_ocv wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

write_ocv

Mögliche Nachfolgerfunktionen

do_ocv_simple, close_ocv

Siehe auch

read_ocr

Modul

Optical character verification

traind_ocv_proj (Pattern : : OCVHandle, Name, Mode :)

Training eines OCV-Tools mit Zeichen.

traind_ocv_proj trainiert ein OCV-Tool das mit create_ocv_proj erzeugt oder mit read_ocv eingelesen wurde. Zum Training werden dem System die Zeichen als Regionen mit dem zugehörigen Bild übergeben. Es ist zu beachten, daß die Region nicht nur den Vordergrund des Zeichens (z.B. die dunklen Pixel) beinhaltet, sondern auch Pixel von der Umgebung des Zeichens. Dies kann z.B. das umschließende Rechteck des Zeichens sein. Ohne diesen Kontext kann ein Zeichen nicht beurteilt werden.

Falls mehr als ein Muster gelernt werden soll, kann das Training durch einen Aufruf mit mehreren Zeichen zusammen mit einem Tuple der zugehörigen Namen oder durch mehrfache Aufrufe des Operators mit jeweils einem Muster realisiert werden. Das Verhalten, d.h. die Bewertung eines Zeichens, wird hierdurch nicht beeinflußt. Die Laufzeit des Trainings wird jedoch bei einer Aufteilung auf mehrers Aufrufe etwas höher sein.

Parameter

- ▷ **Pattern** (input_object) ..... image(-array) ⇝ *Hobject*
  Zu trainierende Zeichen.
- ▷ **OCVHandle** (input_control) ..... ocv ⇝ *integer*
  Handle des zu trainierenden OCV-Tools.
- ▷ **Name** (input_control) ..... string(-array) ⇝ *string*
  Name(n) der zu untersuchenden Objekte.
  Defaultwert : ”a”
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus für Training (nur ein Modus verfügbar).
  Defaultwert : ’single’
  Werteliste : Mode ∈ {’single’}

Beispiel

```
create_ocv_proj("A",&ocv_handle);
draw_region(&ROI,window_handle);
reduce_domain(Image,ROI,&Sample);
traind_ocv_proj(Sample,ocv_handle,"A","single");
```

Ergebnis

traind_ocv_proj liefert den Wert 2 (H_MSG_TRUE), falls das Handle und die Trainingsdaten gültig sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

traind_ocv_proj wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

write_ocr_trainf, create_ocv_proj, read_ocv, threshold, connection, select_shape

Mögliche Nachfolgerfunktionen

close_ocv

Siehe auch

traind_ocr_class_box

Modul

Optical character verification

write_ocv (: : OCVHandle, FileName :)

Sichern eines neuen OCV-Tools in eine Datei.

write_ocv schreibt ein OCV-Tool in eine Datei. Dies kann nach dem Operator traind_ocv_proj zum Sichern des Zustandes verwendet werden. Das OCV-Tool wird durch das Schreiben nicht verändert. Die Datei kann später mit read_ocv wieder eingelesen werden. Als Extension wird ’.ocv’ verwendet. Wenn diese Extension nicht mit dem Dateinamen angegeben ist, wird sie automatisch angefügt.

Parameter

- **OCVHandle** (input_control) ..... ocv ⇝ *integer*
  Handle des zu schreibenden OCV-Tools.
- **FileName** (input_control) ..... filename ⇝ *string*
  Name der Datei, in die das Tool gesichert werden soll.
  Defaultwert : ’test_ocv’

Ergebnis

write_ocv liefert den Wert 2 (H_MSG_TRUE), falls die Daten korrekt sind und die Datei geschrieben werden kann. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

write_ocv ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

traind_ocv_proj

Mögliche Nachfolgerfunktionen

close_ocv

Siehe auch

write_ocr

Modul

Optical character verification

12.14 Shape-from

depth_from_focus (MultiFocusImage : Depth, Confidence : Filter, Selection :)

Bestimmung der Tiefe aus mehreren Fokusebenen.

depth_from_focus bestimmt aus mehreren Bildern einer Fokusserie die Tiefe. Die Bilder der Fokusserie werden dabei als ein mehrkanaliges Bild in MultiFocusImage übergeben. Die Tiefe für jeden Bildpunkt wird als die Nummer des Kanals in Depth zurückgeben. Der Parameter Confidence gibt die Sicherheit an, mit der

die Tiefe für die Bildpunkte geschätzt werden konnte. Große Werte bedeuten dabei eine große Zuverlässigkeit der Schätzung.

Es wird dasjenige Pixel einer Fokusebene ausgewählt, das lokal am schärfsten ist. Das verwendete Verfahren wird über die Parameter `Filter` und `Selection` festgelegt.

'highpass' Die Schärfe wird über einen Hochpaßfilter bestimmt.

'bandpass' Die Schärfe wird über einen Bandpaßfilter bestimmt.

'next_maximum' Es wird das nächstliegende Schärfemaximum in der Umgebung jedes Pixels zur Bestimmung der optimalen Fokusebene verwendet.

'local' Es werden die für das Pixel ermittelten Schärfewerte aller Fokusebenen verwendet.

Parameter

- ▷ **`MultiFocusImage`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte
  Mehrkanaliges Graubild bestehend aus meheren Fokusebenen.
- ▷ **`Depth`** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject* : byte
  Tiefenbild.
- ▷ **`Confidence`** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject* : byte
  Sicherheit der Tiefenschätzung.
- ▷ **`Filter`** (input_control) ..... string(-array) ⇝ *string*
  Filter, um scharfe Pixel zu finden.
  Defaultwert : 'highpass'
  Werteliste : `Filter` ∈ {'highpass', 'bandpass'}
- ▷ **`Selection`** (input_control) ..... string(-array) ⇝ *string*
  Methode scharfe Pixel zu finden.
  Defaultwert : 'next_maximum'
  Werteliste : `Selection` ∈ {'next_maximum', 'local'}

Beispiel

```
compose3(Focus0,Focus1,Focus2,&MultiFocus);
depth_from_focus(MultiFocus,&Depth,&Confidence,'highpass','next_maximum');
mean_image(Depth,&Smooth,15,15);
select_grayvalues_from_channels(MultiChannel,Smooth,SharpImage);
threshold(Confidence,HighConfidence,10,255);
reduce_domain(SharpImage,HighConfidence,ConfidentSharp);
```

Parallelisierungsinformation

`depth_from_focus` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`compose2`, `compose3`, `compose4`, `add_channels`, `read_image`, `read_sequence`

Mögliche Nachfolgerfunktionen

`select_grayvalues_from_channels`, `mean_image`, `gauss_image`, `threshold`

Siehe auch

`count_channels`

Modul

Tools

`estimate_al_am ( Image : : : Albedo, Ambient )`

Berechnung des Reflexionskoeffizienten und der Stärke der indirekten Beleuchtung der Oberfläche.

Die Prozedur `estimate_al_am` berechnet aus dem Bild `Image` durch Maximum- und Minimumberechnung die Größe des Reflexionskoeffizienten, genannt `Albedo`, und die Stärke der indirekten Beleuchtung der Oberfläche, genannt `Ambient`.

Achtung

Es wird angenommen, daß im Bild mindestens ein Punkt enthalten ist, für die die Reflexionsfunktion ihr Minimum annimmt. Das sind z.B. Punkte, die in selbstbeschatteten Gebieten der Oberfläche liegen. Außerdem wird angenommen, daß ein Punkt im Bild vorkommt, für den die Reflexionsfunktion ihr Maximum annimmt. Ist dies nicht der Fall, wird ein falscher Wert geschätzt.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das Bild, für das der Reflexionskoeffizient und die Hintergrundbeleuchtung geschätzt werden soll.
- ▷ **Albedo** (output_control) ..... real(-array) ⇝ *real*
  Größe des Reflexionskoeffizienten der Oberfläche.
- ▷ **Ambient** (output_control) ..... real(-array) ⇝ *real*
  Stärke der indirekten Beleuchtung der Oberfläche.

Ergebnis

`estimate_al_am` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`estimate_al_am` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `phot_stereo`, `shade_height_field`

Modul

Tools

estimate_sl_al_lr (Image : : : Slant, Albedo)

Berechnung des Slants der Lichtquelle und des Reflexionskoeffizienten der Oberfläche.

Die Prozedur `estimate_sl_al_lr` berechnet nach dem Algorithmus von Lee und Rosenfeld aus dem Bild `Image` den `Slant` der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle mit der positiven z-Achse bildet, und den Reflexionskoeffizienten `Albedo` der im Bild dargestellten Oberfläche, d.h. das Verhältnis von einfallender Strahlung zu austretender Strahlung.

Achtung

Der Reflexionskoeffizient `Albedo` wird für die gesamte im Bild dargestellte Oberfläche als konstant angenommen.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das Bild, für das der Slant und der Reflexionskoeffizient geschätzt werden soll.
- ▷ **Slant** (output_control) ..... angle.deg(-array) ⇝ *real*
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
- ▷ **Albedo** (output_control) ..... real(-array) ⇝ *real*
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.

Ergebnis

`estimate_sl_al_lr` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`estimate_sl_al_lr` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `phot_stereo`, `shade_height_field`

Modul

Tools

estimate_sl_al_zc (Image : : : Slant, Albedo)

Berechnung des Slants der Lichtquelle und des Reflexionskoeffizienten der Oberfläche.

Die Prozedur estimate_sl_al_zc berechnet nach dem Algorithmus von Zheng und Chellappa aus dem Bild Image den Slant der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle mit der positiven z-Achse bildet, und den Reflexionskoeffizienten Albedo der im Bild dargestellten Oberfläche, d.h. das Verhältnis von einfallender Strahlung zu austretender Strahlung.

Achtung

Der Reflexionskoeffizient Albedo wird für die gesamte im Bild dargestellte Oberfläche als konstant angenommen.

Parameter

- **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das Bild, für das der Slant und der Reflexionskoeffizient geschätzt werden soll.
- **Slant** (output_control) ..... angle.deg(-array) ⇝ *real*
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
- **Albedo** (output_control) ..... real(-array) ⇝ *real*
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.

Ergebnis

estimate_sl_al_zc liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

estimate_sl_al_zc ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

sfs_mod_lr, sfs_orig_lr, sfs_pentland, phot_stereo, shade_height_field

Modul

Tools

estimate_tilt_lr (Image : : : Tilt)

Berechnung des Tilts der Lichtquelle.

Die Prozedur estimate_tilt_lr berechnet nach dem Algorithmus von Lee und Rosenfeld aus dem Bild Image den Tilt der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle nach Projektion in die xy-Ebene mit der x-Achse bildet.

Parameter

- **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das Bild, für das der Tilt geschätzt werden soll.
- **Tilt** (output_control) ..... angle.deg(-array) ⇝ *real*
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).

Ergebnis

estimate_tilt_lr liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

estimate_tilt_lr ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

sfs_mod_lr, sfs_orig_lr, sfs_pentland, phot_stereo, shade_height_field

Modul

Tools

estimate_tilt_zc (Image : : : Tilt)

Berechnung des Tilts der Lichtquelle.

Die Prozedur estimate_tilt_zc berechnet nach dem Algorithmus von Zheng und Chellappa aus dem Bild Image den Tilt der Lichtquelle, d.h. den Winkel, den die Richtung der Lichtquelle nach Projektion in die xy-Ebene mit der x-Achse bildet.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das Bild, für das der Tilt geschätzt werden soll.
- ▷ **Tilt** (output_control) ..... angle.deg(-array) ⇝ *real*
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).

Ergebnis

`estimate_tilt_zc` liefert immer den Wert 2 (H_MSG_TRUE).

Parallelisierungsinformation

`estimate_tilt_zc` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Nachfolgerfunktionen

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `phot_stereo`, `shade_height_field`

Modul

Tools

phot_stereo (Images : Height : Slants, Tilts :)

Rekonstruktion der relativen Höhe aus den Grauwerten.

Die Prozedur `phot_stereo` berechnet nach dem Algorithmus von Woodham aus mindestens drei Bildern `Images`, die als mehrkomponentiges Bild übergeben werden müssen, und den zu diesen Bildern gehörigen Lichtquellen, die durch die Parameter `Slants` und `Tilts` bestimmt werden, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquellen liegen dabei im Unendlichen in der Richtung, die durch den jeweiligen Slant und Tilt bestimmt werden.

Achtung

`phot_stereo` nimmt an, daß die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. Da die Funktion auf der schnellen Fouriertransformation basiert, werden nur quadratische Bilder mit einer Potenz von 2 als Seitenlänge verarbeitet. Alle Bilder müssen Byte-Bilder sein. Es müssen mindestens drei Bilder übergeben werden. Die Bilder müssen als ein mehrkomponentiges Bild übergeben werden. `Slants` und `Tilts` müssen genauso viele Elemente enthalten, wie Bilder in `Images` übergeben worden sind. Mindestens drei der Lichtquellen müssen linear unabhängig sein.

Parameter

- ▷ **Images** (input_object) ..... (multichannel-)image ⇝ *Hobject*
  Die schattierten Bilder. Es müssen mindestens drei Bilder als ein mehrkomponentiges Bild übergeben werden.
- ▷ **Height** (output_object) ..... image ⇝ *Hobject*
  Rekonstruiertes Höhenfeld.
- ▷ **Slants** (input_control) ..... angle.deg-array ⇝ *real* / integer
  Winkel zwischen der Richtung den Lichtquellen und der positiven z-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : `Slants` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ `Slants` ≤ 180.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10.0
- ▷ **Tilts** (input_control) ..... angle.deg-array ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : `Tilts` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ `Tilts` ≤ 360.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `phot_stereo` den Wert 2 (H_MSG_TRUE), sonst eine Fehlermeldung.

Parallelisierungsinformation

`phot_stereo` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`estimate_sl_al_lr`, `estimate_sl_al_zc`, `estimate_tilt_lr`, `estimate_tilt_zc`

Mögliche Nachfolgerfunktionen

`shade_height_field`

Modul

Tools

`select_grayvalues_from_channels ( MultichannelImage, IndexImage : Selected : : )`

Auswahl von Grauwerten eines mehrkanaligen Bildes mit Hilfe eines Indexbildes.

`select_grayvalues_from_channels` wählt aus einem mehrkanaligen Bild `MultichannelImage` pro Kanal einen Grauwert aus. Die Kanalnummer des Pixels wird dabei aus dem Indexbild `IndexImage` bestimmt: Der Grauwert ist die Nummer des Kanals.

Parameter

- ▷ **`MultichannelImage`** (input_object) ..... multichannel-image(-array) ⇝ *Hobject* : byte
  Mehrkanaliges Graubild.
- ▷ **`IndexImage`** (input_object) ..... singlechannel-image(-array) ⇝ *Hobject* : byte
  Bild, bei dem die Grauwerte als Index interpretiert werden.
- ▷ **`Selected`** (output_object) ..... singlechannel-image(-array) ⇝ *Hobject* : byte
  Tiefenbild.

Beispiel

```
compose3(Focus0,Focus1,Focus2,&MultiFocus);
depth_from_focus(MultiFocus,&Depth,&Confidence,'highpass','next_maximum');
mean_image(Depth,&Smooth,15,15);
select_grayvalues_from_channels(MultiChannel,Smooth,SharpImage);
```

Parallelisierungsinformation

`select_grayvalues_from_channels` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*, *Domänen-Ebene*).

Mögliche Vorgängerfunktionen

`depth_from_focus`, `mean_image`

Mögliche Nachfolgerfunktionen

`disp_image`

Siehe auch

`count_channels`

Modul

Tools

`sfs_mod_lr ( Image : Height : Slant, Tilt, Albedo, Ambient : )`

Rekonstruktion der relativen Höhe aus den Grauwerten.

Die Prozedur `sfs_mod_lr` berechnet nach dem modifizierten Algorithmus von Lee und Rosenfeld aus dem Bild `Image` und einer Lichtquelle, die durch die Parameter `Slant`, `Tilt`, `Albedo` und `Ambient` bestimmt wird, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquelle liegt dabei im

Unendlichen in der Richtung, die durch Slant und Tilt bestimmt werden. Der Parameter Albedo gibt den Reflexionskoeffizienten der Oberfläche an. Ambient ist ein Maß für die Stärke der Hintergrundbeleuchtung. Dieser Parameter kann auf einen Wert größer als 0 gesetzt werden, falls z.B. der Weißabgleich der Kamera nicht genau vorgenommen worden ist.

Achtung

sfs_mod_lr nimmt an, daß die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. Da die Funktion auf der schnellen Fouriertransformation basiert, werden nur quadratische Bilder mit einer Potenz von 2 als Seitenlänge verarbeitet. sfs_mod_lr ist nur für Byte-Bilder implementiert.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das schattierte Bild.
- ▷ **Height** (output_object) ..... image(-array) ⇝ *Hobject*
  Das rekonstruierte Höhenfeld.
- ▷ **Slant** (input_control) ..... angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : Slant ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ Slant ≤ 180.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10.0
- ▷ **Tilt** (input_control) ..... angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : Tilt ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ Tilt ≤ 360.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10.0
- ▷ **Albedo** (input_control) ..... number ⇝ *real* / integer
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.
  Defaultwert : 1.0
  Wertevorschläge : Albedo ∈ {0.1, 0.5, 1.0, 5.0}
  Typischer Wertebereich : 0.0 ≤ Albedo ≤ 5.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Albedo ≥ 0.0
- ▷ **Ambient** (input_control) ..... number ⇝ *real* / integer
  Stärke der indirekten Beleuchtung.
  Defaultwert : 0.0
  Wertevorschläge : Ambient ∈ {0.1, 0.5, 1.0}
  Typischer Wertebereich : 0.0 ≤ Ambient ≤ 1.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Ambient ≥ 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sfs_mod_lr den Wert 2 (H_MSG_TRUE), sonst eine Fehlermeldung.

Parallelisierungsinformation

sfs_mod_lr ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

estimate_al_am, estimate_sl_al_lr, estimate_sl_al_zc, estimate_tilt_lr, estimate_tilt_zc

Mögliche Nachfolgerfunktionen

shade_height_field

Modul

Tools

sfs_orig_lr (Image : Height : Slant, Tilt, Albedo, Ambient :)

Rekonstruktion der relativen Höhe aus den Grauwerten.

Die Prozedur sfs_orig_lr berechnet nach dem originalen Algorithmus von Lee und Rosenfeld aus dem Bild Image und einer Lichtquelle, die durch die Parameter Slant, Tilt, Albedo und Ambient bestimmt wird, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquelle liegt dabei im Unendlichen in der Richtung, die durch Slant und Tilt bestimmt werden. Der Parameter Albedo gibt den Reflexionskoeffizienten der Oberfläche an. Ambient ist ein Maß für die Stärke der Hintergrundbeleuchtung. Dieser Parameter kann auf einen Wert größer als 0 gesetzt werden, falls z.B. der Weißabgleich der Kamera nicht genau vorgenommen worden ist.

Achtung

sfs_orig_lr nimmt an, daß die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. Da die Funktion auf der schnellen Fouriertransformation basiert, werden nur quadratische Bilder mit einer Potenz von 2 als Seitenlänge verarbeitet. sfs_mod_lr ist nur für Byte-Bilder implementiert.

Parameter

- ▷ **Image** (input_object) ..... image(-array) ⇝ *Hobject*
  Das schattierte Bild.
- ▷ **Height** (output_object) ..... image(-array) ⇝ *Hobject*
  Das rekonstruierte Höhenfeld.
- ▷ **Slant** (input_control) ..... angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : Slant ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ Slant ≤ 180.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10.0
- ▷ **Tilt** (input_control) ..... angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : Tilt ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ Tilt ≤ 360.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 10.0
- ▷ **Albedo** (input_control) ..... number ⇝ *real* / integer
  Anteil der Strahlung, der von der Oberfläche reflektiert wird.
  Defaultwert : 1.0
  Wertevorschläge : Albedo ∈ {0.1, 0.5, 1.0, 5.0}
  Typischer Wertebereich : 0.0 ≤ Albedo ≤ 5.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Albedo ≥ 0.0
- ▷ **Ambient** (input_control) ..... number ⇝ *real* / integer
  Stärke der indirekten Beleuchtung.
  Defaultwert : 0.0
  Wertevorschläge : Ambient ∈ {0.1, 0.5, 1.0}
  Typischer Wertebereich : 0.0 ≤ Ambient ≤ 1.0 (lin)
  Minimale Schrittweite : 0.01
  Empfohlene Schrittweite : 0.1
  Restriktion : Ambient ≥ 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `sfs_orig_lr` den Wert 2 (H_MSG_TRUE), sonst eine Fehlermeldung.

Parallelisierungsinformation

`sfs_orig_lr` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`estimate_al_am`, `estimate_sl_al_lr`, `estimate_sl_al_zc`, `estimate_tilt_lr`, `estimate_tilt_zc`

Mögliche Nachfolgerfunktionen

`shade_height_field`

Modul

Tools

`sfs_pentland` (Image : Height : Slant, Tilt, Albedo, Ambient :)

Rekonstruktion der relativen Höhe aus den Grauwerten.

Die Prozedur `sfs_pentland` berechnet nach dem Algorithmus von Pentland aus dem Bild `Image` und einer Lichtquelle, die durch die Parameter `Slant`, `Tilt`, `Albedo` und `Ambient` bestimmt wird, die entsprechenden Höhen, die bei der Bilderzeugung vorgelegen haben. Die Lichtquelle liegt dabei im Unendlichen in der Richtung, die durch `Slant` und `Tilt` bestimmt werden. Der Parameter `Albedo` gibt den Reflexionskoeffizienten der Oberfläche an. `Ambient` ist ein Maß für die Stärke der Hintergrundbeleuchtung. Dieser Parameter kann auf einen Wert größer als 0 gesetzt werden, falls z.B. der Weißabgleich der Kamera nicht genau vorgenommen worden ist.

Achtung

`sfs_pentland` nimmt an, daß die Höhen auf einem Gitter der Schrittweite 1 vorliegen sollen. Ist das nicht der Fall, so müssen die berechneten Höhen nach Ausführung der Funktion noch mit der Gitterschrittweite multipliziert werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet. Da die Funktion auf der schnellen Fouriertransformation basiert, werden nur quadratische Bilder mit einer Potenz von 2 als Seitenlänge verarbeitet. `sfs_pentland` ist nur für Byte-Bilder implementiert.

Parameter

- ▷ **`Image`** (input_object) ..... image(-array) ⇝ *Hobject*
  Das schattierte Bild.
- ▷ **`Height`** (output_object) ..... image(-array) ⇝ *Hobject*
  Das rekonstruierte Höhenfeld.
- ▷ **`Slant`** (input_control) ..... angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : `Slant` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ `Slant` ≤ 180.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0
- ▷ **`Tilt`** (input_control) ..... angle.deg ⇝ *real* / integer
  Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
  Defaultwert : 45.0
  Wertevorschläge : `Tilt` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
  Typischer Wertebereich : 0.0 ≤ `Tilt` ≤ 360.0 (lin)
  Minimale Schrittweite : 1.0
  Empfohlene Schrittweite : 10.0

▷ **Albedo** (input_control) . . . . . number ⇝ *real* / integer
Anteil der Strahlung, der von der Oberfläche reflektiert wird.
Defaultwert : 1.0
Wertevorschläge : Albedo ∈ {0.1, 0.5, 1.0, 5.0}
Typischer Wertebereich : 0.0 ≤ Albedo ≤ 5.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1
Restriktion : Albedo ≥ 0.0

▷ **Ambient** (input_control) . . . . . number ⇝ *real* / integer
Stärke der indirekten Beleuchtung.
Defaultwert : 0.0
Wertevorschläge : Ambient ∈ {0.1, 0.5, 1.0}
Typischer Wertebereich : 0.0 ≤ Ambient ≤ 1.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1
Restriktion : Ambient ≥ 0.0

Ergebnis

Sind die Parameterwerte korrekt, dann liefert sfs_pentland den Wert 2 (H_MSG_TRUE), sonst eine Fehlermeldung.

Parallelisierungsinformation

sfs_pentland ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

estimate_al_am, estimate_sl_al_lr, estimate_sl_al_zc, estimate_tilt_lr, estimate_tilt_zc

Mögliche Nachfolgerfunktionen

shade_height_field

Modul

Tools

shade_height_field (ImageHeight : ImageShade : Slant, Tilt, Albedo, Ambient, Shadows :)

Schattieren eines Höhenfeldes.

shade_height_field berechnet aus einem Höhenfeld (einer Matrix, deren Einträge Höhen auf einem Gitter angeben) ein Bild, das das betreffende Höhenfeld bei Beleuchtung durch eine unendlich weit entfernte Punktlichtquelle darstellt. Dabei wird angenommen, daß die durch das Höhenfeld beschriebene Oberfläche Lambertsche Reflexionseigenschaften besitzt. Es kann zusätzlich bestimmt werden, ob Schatten berechnet werden sollen oder nicht.

Achtung

shade_height_field nimmt an, daß die Höhen auf einem Gitter der x- und y-Schrittweite 1 vorliegen. Ist dies nicht der Fall, so sollten die Höhen vorher entsprechend skaliert werden (z.B. durch Division durch die Gitter-Schrittweite), da die Ableitungen sonst als „zu flach“ oder „zu steil“ geschätzt werden. Beispiel: Das Höhenfeld ist an 100*100 Werten auf dem Quadrat [0,1]*[0,1] gegeben. Dann sollten die Höhen zunächst durch 1/100 geteilt werden. Es wird ein rechtshändiges kartesisches Koordinatensystem mit Nullpunkt in der linken unteren Ecke des Bildes verwendet.

Parameter

▷ **ImageHeight** (input_object) . . . . . image(-array) ⇝ *Hobject*
Zu schattierendes Höhenfeld.

▷ **ImageShade** (output_object) . . . . . image(-array) ⇝ *Hobject*
Schattiertes Bild.

▷ **`Slant`** (input_control) . . . . . angle.deg ⇝ *real* / integer
Winkel zwischen der Richtung der Lichtquelle und der positiven z-Achse (im Gradmaß).
Defaultwert : 0.0
Wertevorschläge : `Slant` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
Typischer Wertebereich : 0.0 ≤ `Slant` ≤ 180.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 10.0

▷ **`Tilt`** (input_control) . . . . . angle.deg ⇝ *real* / integer
Winkel zwischen der Richtung der Lichtquelle nach Projektion in die xy-Ebene und der x-Achse (im Gradmaß).
Defaultwert : 0.0
Wertevorschläge : `Tilt` ∈ {1.0, 5.0, 10.0, 20.0, 40.0, 60.0, 90.0}
Typischer Wertebereich : 0.0 ≤ `Tilt` ≤ 360.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 10.0

▷ **`Albedo`** (input_control) . . . . . number ⇝ *real* / integer
Anteil der Strahlung, der von der Oberfläche reflektiert wird.
Defaultwert : 1.0
Wertevorschläge : `Albedo` ∈ {0.1, 0.5, 1.0, 5.0}
Typischer Wertebereich : 0.0 ≤ `Albedo` ≤ 5.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1
Restriktion : `Albedo` ≥ `0.0`

▷ **`Ambient`** (input_control) . . . . . number ⇝ *real* / integer
Stärke der indirekten Beleuchtung.
Defaultwert : 0.0
Wertevorschläge : `Ambient` ∈ {0.1, 0.5, 1.0}
Typischer Wertebereich : 0.0 ≤ `Ambient` ≤ 1.0 (lin)
Minimale Schrittweite : 0.01
Empfohlene Schrittweite : 0.1
Restriktion : `Ambient` ≥ `0.0`

▷ **`Shadows`** (input_control) . . . . . string ⇝ *string*
Berechnung von Schatten ein- bzw. ausschalten.
Defaultwert : 'false'
Wertevorschläge : `Shadows` ∈ {'true', 'false'}

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `shade_height_field` den Wert 2 (H_MSG_TRUE), sonst eine Fehlermeldung.

Parallelisierungsinformation

`shade_height_field` ist *wiedereintrittsfähig* („reentrant") und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`sfs_mod_lr`, `sfs_orig_lr`, `sfs_pentland`, `phot_stereo`

Modul

Tools

Kapitel 13

Tupel

13.1 Arithmetik

tuple_abs (: : T : Abs)

Absolutbetrag eines Tupels.

tuple_abs berechnet den Absolutbetrag des Eingabetupels T. Der Absolutbetrag einer ganzen Zahl ist wieder eine ganze Zahl. Der Absolutbetrag einer Gleitpunktzahl ist eine Gleitpunktzahl. Der Absolutbetrag von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Abs** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Absolutbetrag des Eingabetupels.

Parallelisierungsinformation

tuple_abs ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_fabs

Modul

Operators not requiring licensing

tuple_acos (: : T : ACos)

Arcus-Kosinus eines Tupels.

tuple_acos berechnet den Arcus-Kosinus des Eingabetupels T. Der Arcus-Kosinus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Arcus-Kosinus von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
  Restriktion : $(-1 \leq T) \leq 1$
- ▷ **ACos** (output_control) . . . . . number(-array) ⇝ *real*
  Arcus-Kosinus des Eingabetupels.

Parallelisierungsinformation

tuple_acos ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_asin, tuple_atan, tuple_atan2

Siehe auch

tuple_cos

Modul

Operators not requiring licensing

tuple_add (: : S1, S2 : Sum)

Addition zweier Tupel.

tuple_add berechnet die Summe der Eingabetupel S1 und S2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel addiert. Ansonsten muß entweder S1 oder S2 die Länge 1 haben. In diesem Fall wird die Addition für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen addiert werden, ist das Ergebnis wieder eine ganze Zahl. Falls eine Gleitpunktzahl mit einer anderen Zahl addiert wird, ist das Ergebnis eine Gleitpunktzahl. Falls zwei Strings addiert werden, entspricht die Addition einer String-Konkatenation. Falls eine Zahl und ein String addiert werden, wird die Zahl in einen String umgewandelt. Die Addition entspricht so auch der String-Konkatenation.

Parameter

- **S1** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel 1.
- **S2** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel 2.
- **Sum** (output_control) . . . . . number(-array) ⇝ *real* / integer / string
  Summe der Eingabetupel.

Parallelisierungsinformation

tuple_add ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_sub

Modul

Operators not requiring licensing

tuple_asin (: : T : ASin)

Arcus-Sinus eines Tupels.

tuple_asin berechnet den Arcus-Sinus des Eingabetupels T. Der Arcus-Sinus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Arcus-Sinus von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
  Restriktion : $(-1 \leq T) \leq 1$
- **ASin** (output_control) . . . . . number(-array) ⇝ *real*
  Arcus-Sinus des Eingabetupels.

Parallelisierungsinformation

tuple_asin ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_acos, tuple_atan, tuple_atan2

Siehe auch

tuple_sin

Modul

Operators not requiring licensing

tuple_atan (: : T : ATan)

Arcus-Tangens eines Tupels.

tuple_atan berechnet den Arcus-Tangens des Eingabetupels T. Der Arcus-Tangens wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Arcus-Tangens von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **ATan** (output_control) ..... number(-array) ⇝ *real*
  Arcus-Tangens des Eingabetupels.

Parallelisierungsinformation

tuple_atan ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_atan2, tuple_asin, tuple_acos

Siehe auch

tuple_tan

Modul

Operators not requiring licensing

tuple_atan2 (: : Y, X : ATan)

Arcus-Tangens eines Tupels für alle vier Quadranten.

tuple_atan2 berechnet den Arcus-Tangens der Eingabetupel Y/X mit richtiger Behandlung aller vier Quadranten. Der Arcus-Tangens wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Arcus-Tangens von Strings ist nicht erlaubt.

Parameter

- ▷ **Y** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel der y-Werte.
- ▷ **X** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel der x-Werte.
- ▷ **ATan** (output_control) ..... number(-array) ⇝ *real*
  Arcus-Tangens des Eingabetupels.

Parallelisierungsinformation

tuple_atan2 ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_atan, tuple_asin, tuple_acos

Siehe auch

tuple_tan

Modul

Operators not requiring licensing

tuple_ceil (: : T : Ceil)

Ceiling-Funktion eines Tupels.

tuple_ceil berechnet die Ceiling-Funktion des Eingabetupels T, d.h. die kleinste ganze Zahl größer gleich T. Die Ceiling-Funktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Ceiling-Funktion von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- **Ceil** (output_control) . . . . . number(-array) ⇝ *real*
  Ceiling-Funktion des Eingabetupels.

Parallelisierungsinformation

`tuple_ceil` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_ceil`

Modul

Operators not requiring licensing

tuple_cos (: : T : Cos)

Kosinus eines Tupels.

`tuple_cos` berechnet den Kosinus des Eingabetupels `T`. Der Kosinus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Kosinus von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- **Cos** (output_control) . . . . . number(-array) ⇝ *real*
  Kosinus des Eingabetupels.

Parallelisierungsinformation

`tuple_cos` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sin`, `tuple_tan`

Siehe auch

`tuple_acos`

Modul

Operators not requiring licensing

tuple_cosh (: : T : Cosh)

Hyperbolischer Kosinus eines Tupels.

`tuple_cosh` berechnet den hyperbolischen Kosinus des Eingabetupels `T`. Der hyperbolische Kosinus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der hyperbolische Kosinus von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- **Cosh** (output_control) . . . . . number(-array) ⇝ *real*
  Hyperbolischer Kosinus des Eingabetupels.

Parallelisierungsinformation

`tuple_cosh` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sinh`, `tuple_tanh`

Modul

Operators not requiring licensing

tuple_div (: : Q1, Q2 : Quot)

Division zweier Tupel.

tuple_div berechnet den Quotienten der Eingabetupel Q1 und Q2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel dividiert. Ansonsten muß entweder Q1 oder Q2 die Länge 1 haben. In diesem Fall wird die Division für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen dividiert werden, ist das Ergebnis wieder eine ganze Zahl. Falls einer der Operanden eine Gleitpunktzahl ist, ist das Ergebnis eine Gleitpunktzahl. Die Division von Strings ist nicht erlaubt.

Parameter

- ▷ **Q1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **Q2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
  Restriktion : Q2 ≠ 0
- ▷ **Quot** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Quotient der Eingabetupel.

Parallelisierungsinformation

tuple_div ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_mult

Modul

Operators not requiring licensing

tuple_exp (: : T : Exp)

Exponentialfunktion eines Tupels.

tuple_exp berechnet die Exponentialfunktion des Eingabetupels T. Die Exponentialfunktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Exponentialfunktion von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Exp** (output_control) . . . . . number(-array) ⇝ *real*
  Exponentialfunktion des Eingabetupels.

Parallelisierungsinformation

tuple_exp ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_pow

Siehe auch

tuple_log, tuple_log10

Modul

Operators not requiring licensing

tuple_fabs (: : T : Abs)

Absolutbetrag eines Tupels (als Gleitkommazahlen).

tuple_fabs berechnet den Absolutbetrag des Eingabetupels T. Im Gegensatz zu tuple_abs wird der Absolutbetrag von tuple_fabs immer als eine Gleitpunktzahl zurückgelierfert. Der Absolutbetrag von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Abs** (output_control) . . . . . number(-array) ⇝ *real*
  Absolutbetrag des Eingabetupels.

Parallelisierungsinformation

`tuple_fabs` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_abs`

Modul

Operators not requiring licensing

tuple_floor (: : T : Floor)

Floor-Funktion eines Tupels.

`tuple_floor` berechnet die Floor-Funktion des Eingabetupels `T`, d.h. die größte ganze Zahl kleiner gleich `T`. Die Floor-Funktion wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Floor-Funktion von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Floor** (output_control) . . . . . number(-array) ⇝ *real*
  Floor-Funktion des Eingabetupels.

Parallelisierungsinformation

`tuple_floor` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_ceil`

Modul

Operators not requiring licensing

tuple_fmod (: : T1, T2 : Fmod)

Rest der Gleitpunkt-Division zweier Tupel.

`tuple_fmod` berechnet die Rest der Gleitpunkt-Division der Eingabetupel `T1`/`T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel dividiert. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Division für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Division von Strings ist nicht erlaubt.

Parameter

- ▷ **T1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
  Restriktion : `T2` $\neq$ `0.0`
- ▷ **Fmod** (output_control) . . . . . number(-array) ⇝ *real*
  Rest der Division der Eingabetupel.

Parallelisierungsinformation

`tuple_fmod` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

tuple_floor, tuple_ceil

Modul

Operators not requiring licensing

tuple_ldexp (: : T1, T2 : Ldexp)

Ldexp-Funktion zweier Tupel.

tuple_ldexp berechnet die Ldexp-Funktion der Eingabetupel, d.h. $T1 * 2^{T2}$. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel verwendet. Ansonsten muß entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Ldexp-Funktion von Strings ist nicht erlaubt.

Parameter

- **T1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- **T2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- **Ldexp** (output_control) . . . . . number(-array) ⇝ *real*
  Ldexp-Funktion der Eingabetupel.

Parallelisierungsinformation

tuple_ldexp ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

tuple_exp

Modul

Operators not requiring licensing

tuple_log (: : T : Log)

Natürlicher Logarithmus eines Tupels.

tuple_log berechnet den natürlichen Logarithmus des Eingabetupels T. Der natürliche Logarithmus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der natürliche Logarithmus von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
  Restriktion : $T > 0$
- **Log** (output_control) . . . . . number(-array) ⇝ *real*
  Natürlicher Logarithmus des Eingabetupels.

Parallelisierungsinformation

tuple_log ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_log10

Siehe auch

tuple_exp, tuple_pow

Modul

Operators not requiring licensing

tuple_log10 (: : T : Log)

Logarithmus zur Basis 10 eines Tupels.

tuple_log10 berechnet den Logarithmus zur Basis 10 des Eingabetupels T. Der Logarithmus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Logarithmus von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
  Restriktion : T > 0
- ▷ **Log** (output_control) . . . . . number(-array) ⇝ *real*
  Logarithmus zur Basis 10 des Eingabetupels.

Parallelisierungsinformation

tuple_log10 ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_log

Siehe auch

tuple_exp, tuple_pow

Modul

Operators not requiring licensing

tuple_mult (: : P1, P2 : Prod)

Multiplikation zweier Tupel.

tuple_mult berechnet das Produkt der Eingabetupel P1 und P2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel multipliziert. Ansonsten muß entweder P1 oder P2 die Länge 1 haben. In diesem Fall wird die Multiplikation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen multipliziert werden, ist das Ergebnis wieder eine ganze Zahl. Falls einer der Operanden eine Gleitpunktzahl ist, ist das Ergebnis eine Gleitpunktzahl. Die Multiplikation von Strings ist nicht erlaubt.

Parameter

- ▷ **P1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **P2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Prod** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Produkt der Eingabetupel.

Parallelisierungsinformation

tuple_mult ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_div

Modul

Operators not requiring licensing

tuple_neg (: : T : Neg)

Negierung eines Tupels.

tuple_neg berechnet die Negierung des Eingabetupels T, d.h. Neg $= -$T. Die Negierung einer ganzen Zahl ist wieder eine ganze Zahl. Die Negierung einer Gleitpunktzahl ist eine Gleitpunktzahl. Die Negierung von Strings ist nicht erlaubt. Die Negierung eines leeren Eingabtupels ergibt wiederum ein leeres Tupel als Ausgabe.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Neg** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Negation des Eingabetupels.

Parallelisierungsinformation

`tuple_neg` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Operators not requiring licensing

tuple_pow (: : T1, T2 : Pow)

Potenzierung zweier Tupel.

`tuple_pow` berechnet die Potenzierung der Eingabetupel $T1^{T2}$. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel potenziert. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Potenzierung für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Das Ergebnis ist in jedem Fall eine Gleitpunktzahl. Die Potenzierung von Strings ist nicht erlaubt.

Parameter

- ▷ **T1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Pow** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Potenzfunktion der Eingabetupel.

Parallelisierungsinformation

`tuple_pow` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_exp`

Siehe auch

`tuple_log`, `tuple_log10`

Modul

Operators not requiring licensing

tuple_rad (: : Deg : Rad)

Konversion eines Tupels von Grad nach Bogenmaß.

`tuple_rad` konvertiert das Eingabetupel `Deg` von Grad in Bogenmaß. Das Ergebnis wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Konversion von Strings ist nicht erlaubt.

Parameter

- ▷ **Deg** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Rad** (output_control) . . . . . number(-array) ⇝ *real*
  Eingabetupel im Bogenmaß.

Parallelisierungsinformation

`tuple_rad` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

`tuple_deg`

Modul

Operators not requiring licensing

tuple_sin (: : T : Sin)

Sinus eines Tupels.

tuple_sin berechnet den Sinus des Eingabetupels T. Der Sinus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Sinus von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Sin** (output_control) . . . . . number(-array) ⇝ *real*
  Sinus des Eingabetupels.

Parallelisierungsinformation

tuple_sin ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_cos, tuple_tan

Siehe auch

tuple_asin

Modul

Operators not requiring licensing

tuple_sinh (: : T : Sinh)

Hyperbolischer Sinus eines Tupels.

tuple_sinh berechnet den hyperbolischen Sinus des Eingabetupels T. Der hyperbolische Sinus wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der hyperbolische Sinus von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Sinh** (output_control) . . . . . number(-array) ⇝ *real*
  Hyperbolischer Sinus des Eingabetupels.

Parallelisierungsinformation

tuple_sinh ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_cosh, tuple_tanh

Modul

Operators not requiring licensing

tuple_sqrt (: : T : Sqrt)

Quadratwurzel eines Tupels.

tuple_sqrt berechnet die Quadratwurzel des Eingabetupels T. Die Quadratwurzel wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Quadratwurzel von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
  Restriktion : T ≥ 0
- ▷ **Sqrt** (output_control) . . . . . number(-array) ⇝ *real*
  Quadratwurzel des Eingabetupels.

Parallelisierungsinformation

`tuple_sqrt` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Operators not requiring licensing

tuple_sub (: : D1, D2 : Diff)

Subtraktion zweier Tupel.

`tuple_sub` berechnet die Differenz der Eingabetupel `D1` und `D2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel subtrahiert. Ansonsten muß entweder `D1` oder `D2` die Länge 1 haben. In diesem Fall wird die Subtraktion für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Falls zwei ganze Zahlen subtrahiert werden, ist das Ergebnis wieder eine ganze Zahl. Falls einer der Operanden eine Gleitpunktzahl ist, ist das Ergebnis eine Gleitpunktzahl. Die Subtraktion von Strings ist nicht erlaubt.

Parameter

- ▷ **D1** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 1.
- ▷ **D2** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel 2.
- ▷ **Diff** (output_control) . . . . . number(-array) ⇝ *real* / integer
  Differenz der Eingabetupel.

Parallelisierungsinformation

`tuple_sub` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_add`

Modul

Operators not requiring licensing

tuple_tan (: : T : Tan)

Tangens eines Tupels.

`tuple_tan` berechnet den Tangens des Eingabetupels `T`. Der Tangens wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der Tangens von Strings ist nicht erlaubt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Tan** (output_control) . . . . . number(-array) ⇝ *real*
  Tangens des Eingabetupels.

Parallelisierungsinformation

`tuple_tan` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sin`, `tuple_cos`

Siehe auch

`tuple_atan`, `tuple_atan2`

Modul

Operators not requiring licensing

tuple_tanh (: : T : Tanh)

Hyperbolischer Tangens eines Tupels.

tuple_tanh berechnet den hyperbolischen Tangens des Eingabetupels T. Der hyperbolische Tangens wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Der hyperbolische Tangens von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- **Tanh** (output_control) . . . . . number(-array) ⇝ *real*
  Hyperbolischer Tangens des Eingabetupels.

Parallelisierungsinformation

tuple_tanh ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Alternativen

tuple_sinh, tuple_cosh

Modul

Operators not requiring licensing

13.2 Bitoperationen

tuple_band (: : T1, T2 : BAnd)

Bitweises Und zweier Tupel.

tuple_band berechnet das bitweise Und der Eingabetupel T1 und T2. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder T1 oder T2 die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- **T1** (input_control) . . . . . integer(-array) ⇝ *integer*
  Eingabetupel 1.
- **T2** (input_control) . . . . . integer(-array) ⇝ *integer*
  Eingabetupel 2.
- **BAnd** (output_control) . . . . . integer(-array) ⇝ *integer*
  Binäres Und der Eingabetupel.

Parallelisierungsinformation

tuple_band ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_bor, tuple_bxor, tuple_bnot

Siehe auch

tuple_and, tuple_or, tuple_xor, tuple_not

Modul

Operators not requiring licensing

tuple_bnot (: : T : BNot)

Bitweises Nicht zweier Tupel.

tuple_bnot berechnet das bitweise Nicht des Eingabetupels T. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- ▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel.
- ▷ **BNot** (output_control) ..... integer(-array) ⇝ *integer*
  Binäres Nicht des Eingabetupels.

Parallelisierungsinformation

`tuple_bnot` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_band`, `tuple_bor`, `tuple_bxor`

Siehe auch

`tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Modul

Operators not requiring licensing

tuple_bor (: : T1, T2 : BOr)

Bitweises Oder zweier Tupel.

`tuple_bor` berechnet das bitweise Oder der Eingabetupel `T1` und `T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- ▷ **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **BOr** (output_control) ..... integer(-array) ⇝ *integer*
  Binäres Oder der Eingabetupel.

Parallelisierungsinformation

`tuple_bor` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_band`, `tuple_bxor`, `tuple_bnot`

Siehe auch

`tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Modul

Operators not requiring licensing

tuple_bxor (: : T1, T2 : BXor)

Bitweises Exklusiv-Oder zweier Tupel.

`tuple_bxor` berechnet das bitweise Exklusiv-Oder der Eingabetupel `T1` und `T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- ▷ **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.

▷ **BXor** (output_control) . . . . . integer(-array) ⇝ *integer*
Binäres Exklusiv-Oder der Eingabetupel.

Parallelisierungsinformation

`tuple_bxor` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_band`, `tuple_bor`, `tuple_bnot`

Siehe auch

`tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Modul

Operators not requiring licensing

tuple_lsh (: : T, Shift : Lsh)

Bitweises Verschieben eines Tupels nach links.

`tuple_lsh` verschiebt das Tupel `T` bitweise um `Shift` Stellen nach links. Falls kein Überlauf eintritt, ist diese Operation äquivalent zu einer Multiplikation mit $2^{\texttt{Shift}}$. Falls `T` negativ ist, hängt das Ergebnis von der Rechnerarchitektur ab. Das Ergebnis ist undefiniert, falls `Shift` negativ oder größer als 32 ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T` oder `Shift` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

▷ **T** (input_control) . . . . . integer(-array) ⇝ *integer*
Eingabetupel.
▷ **Shift** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl Stellen, um die verschoben werden soll.
▷ **Lsh** (output_control) . . . . . integer(-array) ⇝ *integer*
Verschobenes Eingabetupel.

Parallelisierungsinformation

`tuple_lsh` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_mult`

Siehe auch

`tuple_rsh`

Modul

Operators not requiring licensing

tuple_rsh (: : T, Shift : Rsh)

Bitweises Verschieben eines Tupels nach rechts.

`tuple_rsh` verschiebt das Tupel `T` bitweise um `Shift` Stellen nach rechts. Diese Operation ist äquivalent zu einer Division mit $2^{\texttt{Shift}}$. Falls `T` negativ ist, hängt das Ergebnis von der Rechnerarchitektur ab. Das Ergebnis ist undefiniert, falls `Shift` negativ oder größer als 32 ist. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T` oder `Shift` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

▷ **T** (input_control) . . . . . integer(-array) ⇝ *integer*
Eingabetupel.
▷ **Shift** (input_control) . . . . . integer(-array) ⇝ *integer*
Anzahl Stellen, um die verschoben werden soll.

▷ **Rsh** (output_control) . . . . . integer(-array) ⇝ *integer*
Verschobenes Eingabetupel.

Parallelisierungsinformation

`tuple_rsh` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_div`

Siehe auch

`tuple_lsh`

Modul

Operators not requiring licensing

13.3 Elementauswahl

tuple_first_n (: : Tuple, Index : Selected)

Auswählen der vorderen Elemente eines Tupels.

`tuple_select` wählt die vorderen Elemente des Tupels `Tuple` aus und liefert sie in dem Ausgabetupel `Selected` zurück. Das Ausgabetupel enthält also alle Elemente von `Tuple`, beginnend mit dem ersten Element von `Tuple` bis einschließlich des “n-ten” Elements von `Tuple`. Der Index “n” wird durch den Parameter `Index` festgelegt. Folgerichtig muß `Index` eine ganze Zahl beinhalten (falls `Index` eine Gleitkommazahl enthält, so muß diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Der Index eines Tupelelements wird beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt.

Parameter

▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Index** (input_control) . . . . . number ⇝ *integer* / real
Index des ersten auszuwählenden Elements.

▷ **Selected** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
Ausgewählte Tupelelemente.

Parallelisierungsinformation

`tuple_first_n` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_last_n`, `tuple_select`, `tuple_last_n`, `tuple_str_bit_select`, `tuple_concat`

Modul

Operators not requiring licensing

tuple_last_n (: : Tuple, Index : Selected)

Auswählen aller Elemente ab Index “n” bis um Ende eines Tupels.

`tuple_last_n` wählt die hinteren Elemente des Tupels `Tuple` aus und liefert sie in dem Ausgabetupel `Selected` zurück. `Selected` enthält dann alle Elemente des Tupels `Tuple` ab dem “n-ten” Element (inklusive des “n-ten” Elements). Der Index “n” wird durch den Parameter `Index` festgelegt. Folgerichtig muß `Index` eine ganze Zahl beinhalten (falls `Index` eine Gleitkommazahl enthält, so muß diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Der Index eines Tupelelements wird beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt.

Parameter

▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Index** (input_control) . . . . . number ⇝ *integer* / real
Index des ersten auszuwählenden Elements.

▷ **Selected** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
Ausgewählte Tupelelemente.

Parallelisierungsinformation

`tuple_last_n` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_first_n`, `tuple_select`, `tuple_str_bit_select`, `tuple_concat`

Modul

Operators not requiring licensing

tuple_select (: : Tuple, Index : Selected)

Auswählen einzelner Elemente aus einem Tupel.

`tuple_select` wählt einzelne Elemente aus einem Tupel `Tuple` aus und liefert sie über den Parameter `Selected` zurück. Der Parameter `Index` gibt hierbei einen oder mehrere Indizes an, über die festgelegt wird, welche Elemente ausgewählt werden. Folgerichtig darf `Index` ausschließlich ganze Zahlen beinhalten (falls `Index` eine Gleitkommazahl enthält, so muß diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Tupelindizes werden beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt.

Parameter

▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Index** (input_control) . . . . . number(-array) ⇝ *integer* / real
Indizes der auszuwählenden Elemente.

▷ **Selected** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
Ausgewähltes Tupelelement.

Parallelisierungsinformation

`tuple_select` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_first_n`, `tuple_last_n`, `tuple_str_bit_select`, `tuple_concat`

Modul

Operators not requiring licensing

tuple_select_range (: : Tuple, Leftindex, Rightindex : Selected)

Auswählen mehrerer Elemente eines Tupels.

`tuple_select_range` wählt mehrere aufeinanderfolgende Elemente eines Tupels `Tuple` aus und liefert sie über den Parameter `Selected` zurück. Der Parameter `Leftindex` gibt hierbei den Index des ersten auszuwählenden Tupelelements und der Parameter `Rightindex` den Index des letzten auszuwählenden Tupelelements an. Folgerichtig müssen beide Parameter `Leftindex` und `Rightindex` eine ganze Zahl beinhalten (falls sie eine Gleitkommazahl enthalten, so muß diese eine ganze Zahl repräsentieren; alle Nachkommastellen der Zahl müssen also 0 sein). Der Index eines Tupelelements wird beginnend mit 0 gezählt, das heißt, das erste Tupelelement wird über den Index 0 ausgewählt. Das Ergebnistupel `Selected` enthält dann alle Elemente von `Tuple`, die zwischen den Positionen `Leftindex` und `Rightindex` stehen (inklusive der Elemente mit dem Index `Leftindex` und `Rightindex`). Der Index in `Rightindex` muß größer oder gleich zu dem Index in `Leftindex` sein. Sind beide Indices gleich groß, so wird nur ein einzelnes Element ausgewählt.

Parameter

▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
Eingabetupel.

▷ **Leftindex** (input_control) . . . . . number(-array) ⇝ *integer* / real
Index des ersten auszuwählenden Elements.

- ▷ **Rightindex** (input_control) . . . . . number(-array) ⇝ *integer* / real
  Index des letzten auszuwählenden Elements.
- ▷ **Selected** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
  Ausgewählte Tupelelemente.

Parallelisierungsinformation

`tuple_select_range` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_select`, `tuple_first_n`, `tuple_last_n`, `tuple_str_bit_select`, `tuple_concat`

Modul

Operators not requiring licensing

tuple_str_bit_select (: : Tuple, Index : Selected)

Auswählen eines einzelnen Zeichens oder Bits aus einem Tupel.

`tuple_str_bit_select` wählt aus einem Tupel `Tuple` von Strings und/oder Zahlen ein einzelnes Zeichen beziehungsweise Bit aus und gibt es in dem Ausgabetupel `Selected` zurück. `Tuple` kann hierbei aus Strings und/oder ganzen Zahlen bestehen. Welches Zeichen beziehungsweise Bit ausgewählt wird, hängt von dem Eingabeparameter `Index` ab, der ausschließlich aus einer Zahl “n” bestehen darf. Ist dies eine Gleitkommazahl, so muß sie eine ganze Zahl repräsentieren (d.h., alle Nachkommastellen müssen 0 sein). Das Ausgabetupel `Selected` enthält nun für jedes Element aus `Tuple` ein neues Element, das aus dem “n-ten” Zeichen (bei Strings) beziehungsweise dem “n-ten” Bit (bei ganzen Zahlen) des korrespondierenden Elements von `Tuple` besteht.

Parameter

- ▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *string* / integer
  Eingabetupel.
- ▷ **Index** (input_control) . . . . . number ⇝ *integer* / real
  Position des Zeichens oder Bits.
- ▷ **Selected** (output_control) . . . . . number(-array) ⇝ *string* / integer
  Tupel mit den ausgewählten Zeichen und Bits.

Parallelisierungsinformation

`tuple_str_bit_select` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_str_bit_select`, `tuple_select`, `tuple_first_n`, `tuple_last_n`, `tuple_concat`, `tuple_strchr`, `tuple_strrchr`, `tuple_str_first_n`, `tuple_str_last_n`, `tuple_and`, `tuple_or`, `tuple_xor`, `tuple_not`

Modul

Operators not requiring licensing

13.4 Elementreihenfolge

tuple_inverse (: : Tuple : Inverted)

Invertiert ein Tupel.

`tuple_inverse` invertiert das Eingabetupel `Tuple`. Das Ergebnistupel `Inverted` enthält also dieselben Elemente wie `Tuple` jedoch in umgekehrter Reihenfolge.

Parameter

- ▷ **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **Inverted** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
  Inversion des Eingabetupels.

Parallelisierungsinformation

`tuple_inverse` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sort`, `tuple_sort_index`

Modul

Operators not requiring licensing

tuple_sort (: : Tuple : Sorted)

Sortiert die Elemente eines Tupels in aufsteigender Reihenfolge.

`tuple_sort` sortiert die Elemente des Eingabetupels `Tuple` in aufsteigender Reihenfolge und liefert das Resultat in dem Ergebnistupel `Sorted` zurück. Als Voraussetzung hierfür müssen alle Elemente von `Tuple` vergleichbar sein, das heißt, entweder `Tuple` besteht gänzlich aus Strings oder es enthält ausschließlich Zahlen, wobei in letzterem Fall auch ganze Zahlen und Gleitkommazahlen vermischt auftreten dürfen.

Parameter

- **Tuple** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- **Sorted** (output_control) ..... number(-array) ⇝ *integer* / real / string
  Sortiertes Tupel.

Parallelisierungsinformation

`tuple_sort` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sort_index`, `tuple_inverse`

Modul

Operators not requiring licensing

tuple_sort_index (: : Tuple : Indices)

Sortiert ein Tupel und gibt die Indizes des sortierten Tupels zurück.

`tuple_sort_index` sortiert die Elemente des Eingabetupels `Tuple` in aufsteigender Reihenfolge und liefert als Resultat die Indizes der Elemente des sortierten Tupels (bezogen auf das Eingabetupel) in dem Ergebnistupel `Indices` zurück. Als Voraussetzung hierfür müssen alle Elemente von `Tuple` vergleichbar sein, das heißt, entweder `Tuple` besteht gänzlich aus Strings oder es enthält ausschließlich Zahlen, wobei in letzterem Fall auch ganze Zahlen und Gleitkommazahlen auch vermischt auftreten dürfen.

Parameter

- **Tuple** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- **Indices** (output_control) ..... number(-array) ⇝ *integer*
  Sortiertes Tupel.

Parallelisierungsinformation

`tuple_sort_index` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sort`, `tuple_inverse`

Modul

Operators not requiring licensing

13.5 Generierung

tuple_concat (: : T1, T2 : Concat)

Verknüpfe zwei Tupel zu einem neuen Tupel.

tuple_concat verknüpft die zwei Eingabetupel T1 und T2 zu einem neuen Tupel Concat, dessen vordere Elemente aus den Elementen von T1 und dessen hintere Elemente aus denen von T2 bestehen

Parameter

- ▷ **T1** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Concat** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
  Ergebnis der Verknüpfung der Eingabetupel.

Parallelisierungsinformation

tuple_concat ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_gen_const, tuple_str_bit_select, tuple_select, tuple_str_first_n, tuple_str_last_n

Modul

Operators not requiring licensing

tuple_gen_const (: : Length, Const : Newtuple)

Erzeugen eines Tupels definierter Länge mit einer initialen Belegung.

tuple_gen_const erzeugt ein neues Tupel und gibt es in Newtuple zurück. Die Anzahl der Tupelelemente wird mit dem Parameter Length festgelegt. Length darf folglich nur aus einer einzigen Zahl bestehen. Enthält Length eine Gleitkommazahl, so darf diese nur eine ganze Zahl enthalten (alle Nachkommastellen müssen 0 sein). Der Datentyp der einzelnen Elemente des neu generierten Tupels und die initiale Belegung der Elemente wird durch den zweiten Eingabeparameter Const festgelegt. Const darf nur ein einziges Element enthalten. Alle Elemente von Newtuple entsprechen dann in ihrem Typ und ihrer Belegung exakt dem Datenelement von Const.

Parameter

- ▷ **Length** (input_control) . . . . . number ⇝ *integer* / real
  Länge des zu erzeugenden Tupels.
- ▷ **Const** (input_control) . . . . . number ⇝ *integer* / real / string
  Konstante für die Initialisierung der Tupelelemente.
- ▷ **Newtuple** (output_control) . . . . . number(-array) ⇝ *integer* / real / string
  Neues Tupel.

Parallelisierungsinformation

tuple_gen_const ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_str_bit_select, tuple_select, tuple_str_first_n, tuple_str_last_n, tuple_concat

Modul

Operators not requiring licensing

13.6 Konversion

tuple_chr (: : T : Chr)

Konversion eines Tupels in Strings mit entsprechendem ASCII-Code.

tuple_chr konvertiert das Eingabetupel T aus ganzen Zahlen in ein Tupel mit Strings der Länge 1, deren Zeichen den ASCII-Code der entsprechenden Eingabezahl haben.

Parameter

- ▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel.
  Restriktion : (0 ≤ T) ≤ 255
- ▷ **Chr** (output_control) ..... string(-array) ⇝ *string*
  Strings, die dem ASCII-Code des Eingabetupels entsprechen.

Parallelisierungsinformation

tuple_chr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_chrt

Siehe auch

tuple_ord, tuple_ords

Modul

Operators not requiring licensing

tuple_chrt (: : T : Chrt)

Konversion eines Tupels in Strings mit entsprechendem ASCII-Code.

tuple_chrt konvertiert das Eingabetupel T aus ganzen Zahlen in ein Tupel mit Strings und ganzen Zahlen (wobei nur die ganze Zahl 0 vorkommen kann), deren Zeichen den ASCII-Code der entsprechenden Eingabezahl haben. Dabei wird versucht, möglichst viele Zeichen des Eingabetupels in einen String zu packen. Nur, wenn der Wert 0 in T vorhanden ist, wird an dieser Stelle der aktuelle String abgebrochen, eine ganze Zahl 0 in die Ausgabe eingefügt, und ein neuer String mit den restlichen Eingabewerten angefangen. Dieser Operator ist nützlich, um Eingaben von Zeichen, die mit read_serial gelesen wurden, in Strings umzuwandeln. Mit diesem Mechanismus ist es möglich, auch Bytes mit dem Wert 0 zu lesen.

Parameter

- ▷ **T** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel.
  Restriktion : (0 ≤ T) ≤ 255
- ▷ **Chrt** (output_control) ..... string(-array) ⇝ *string* / integer
  Strings, die dem ASCII-Code des Eingabetupels entsprechen.

Beispiel

```
read_serial (SerialHandle, 100, Data)
tuple_chrt (Data, Strings)
```

Parallelisierungsinformation

tuple_chrt ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_chr

Siehe auch

tuple_ord, tuple_ords, read_serial

Modul

Operators not requiring licensing

tuple_deg (: : Rad : Deg)

Konversion eines Tupels von Bogenmaß nach Grad.

tuple_deg konvertiert das Eingabetupel Rad von Bogenmaß in Grad. Das Ergebnis wird in jedem Fall als Gleitpunktzahl zurückgeliefert. Die Konversion von Strings ist nicht erlaubt.

Parameter

- ▷ **Rad** (input_control) . . . . . number(-array) ⇝ *real* / integer
  Eingabetupel.
- ▷ **Deg** (output_control) . . . . . number(-array) ⇝ *real*
  Eingabetupel in Grad.

Parallelisierungsinformation

tuple_deg ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

tuple_rad

Modul

Operators not requiring licensing

tuple_is_number (: : T : IsNumber)

Test eines Tupels (von Strings), ob es Zahlen darstellt.

tuple_is_number testet das Eingabetupel T daraufhin, welches Element eine Zahl darstellt. Dabei wird für schon vorhandene Zahlen 1 zurückgeliefert. Für Strings wird getestet, ob der String eine Zahl darstellt oder nicht. Für Strings, die Zahlen darstellen, wird eine 1 zurückgeliefert, ansonsten eine 0

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel.
- ▷ **IsNumber** (output_control) . . . . . integer(-array) ⇝ *integer*
  Tupel mit Booleschen Werten.

Parallelisierungsinformation

tuple_is_number ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

tuple_number

Modul

Operators not requiring licensing

tuple_number (: : T : Number)

Konversion eines Tupels (von Strings) in ein Tupel von Zahlen.

tuple_number konvertiert das Eingabetupel T in ein Tupel von Zahlen. Dabei werden schon vorhandene Zahlen im Eingabetupel kopiert. Strings werden in den passenden Zahlentyp (Gleitpunktzahlen oder ganze Zahlen) konvertiert oder als Strings kopiert, falls sie keine Zahl darstellen. Strings, die mit 0x bzw. 0 beginnen, werden als Hexadezimalzahlen bzw. Oktalzahlen interpretiert. Der String ’20’ z.B. wird in die Zahl 20 konvertiert, ’020’ in 16, ’0x20’ in 32.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel.
- ▷ **Number** (output_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel als Zahlen.

Parallelisierungsinformation

`tuple_number` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Siehe auch

`tuple_is_number`, `tuple_string`

Modul

Operators not requiring licensing

tuple_ord (: : T : Ord)

ASCII-Code eines Tupels von Strings der Länge 1.

`tuple_ord` konvertiert das Eingabetupel `T`, das ausschließlich Strings der Länge 1 enthalten darf, in ein Tupel von ganzen Zahlen, die den ASCII-Code der Zeichen der Strings darstellen.

Parameter

- ▷ **T** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel.
- ▷ **Ord** (output_control) . . . . . integer(-array) ⇝ *integer*
  ASCII-Code der Eingabetupel.

Parallelisierungsinformation

`tuple_ord` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_ords`

Siehe auch

`tuple_chr`, `tuple_chrt`

Modul

Operators not requiring licensing

tuple_ords (: : T : Ords)

ASCII-Code eines Tupels von Strings.

`tuple_ords` konvertiert das Eingabetupel `T`, das ausschließlich Strings und ganze Zahlen enthalten darf, in ein Tupel von ganzen Zahlen, die den ASCII-Code der Zeichen der Strings darstellen. Dabei werden die Zeichen der einzelnen Strings entsprechend ihrer Reihenfolge im String und der Reihenfolge im Tupel ausgegeben. Ganze Zahlen werden in das Ausgabetupel übernommen. Dieser Operator ist nützlich, um Ausgaben von Strings mit `write_serial` vorzubereiten. Insbesondere kann ein Byte mit Wert 0 geschrieben werden, indem in `T` eine ganze Zahl mit Wert 0 angegeben wird.

Parameter

- ▷ **T** (input_control) . . . . . string(-array) ⇝ *string* / integer
  Eingabetupel.
- ▷ **Ords** (output_control) . . . . . integer(-array) ⇝ *integer*
  ASCII-Code der Eingabetupel.

Beispiel

```
tuple_ords (["String 1", 0, "String 2", 0], Data)
write_serial (SerialHandle, Data)
```

Parallelisierungsinformation

`tuple_ords` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_ord`

Siehe auch

tuple_chr, tuple_chrt, write_serial

Modul

Operators not requiring licensing

tuple_real (: : T : Real)

Konversion eines Tupels in ein Tupel von Gleitkommazahlen.

tuple_real konvertiert das Eingabetupel T in ein Tupel von Gleitkommazahlen. Die Konversion von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
- **Real** (output_control) ..... number(-array) ⇝ *real*
  Eingabetupel als Gleitkommazahlen.

Parallelisierungsinformation

tuple_real ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Operators not requiring licensing

tuple_round (: : T : Round)

Konversion eines Tupels in ein Tupel von ganzen Zahlen.

tuple_round konvertiert das Eingabetupel T in ein Tupel von ganzen Zahlen. Dabei wird T zur nächsten ganzen Zahl gerundet. Die Konversion von Strings ist nicht erlaubt.

Parameter

- **T** (input_control) ..... number(-array) ⇝ *real* / integer
  Eingabetupel.
- **Round** (output_control) ..... number(-array) ⇝ *integer*
  Eingabetupel als ganze Zahlen.

Parallelisierungsinformation

tuple_round ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Operators not requiring licensing

tuple_string (: : T, Format : String)

Konvertierung eines Tupels in ein Tupel von Strings.

tuple_string konvertiert Zahlen in Strings und modifiziert Strings. Der Operator hat zwei Parameter: T ist die Zahl oder der String, der konvertiert werden muß. Format spezifiziert die Konversion. Dieser Format-String besteht aus den folgenden vier Teilen:

```
<flags><field width><precision><conversion characters>
```

Eine Konversion könnte also folgendermaßen aussehen:

```
tuple\_string(1332.4554, '.6e', String)
```

flags Null oder mehr Flags, in beliebiger Reihenfolge, die die Bedeutung der Konversionsspezifikation modifizieren. Die Flags können aus den folgenden Zeichen bestehen:

- **-** Das Ergebnis der Konversion wird links im Ergebnisfeld ausgerichtet.
- **+** Das Ergebnis einer vorzeichenbehafteten Konversion fängt immer mit dem Vorzeichen as, also + oder -.
- <**space**> Falls das erste Zeichen einer vorzeichenbehafteten Konversion kein Vorzeichen ist, wird ein Leerzeichen vor das Ergebnis eingefügt. Falls das <space> Flag und das + Flag gleichzeitig angegeben werden, wird das <space> Flag ignoriert.
- **#** Der Wert soll in eine „alternativen Form" konvertiert werden. Für die d und s Konversionen hat dieses Flag keine Auswirkungen. Bei der o Konversion (siehe unten) erhöht es die Präzision, so daß das erste Zeichen immer als 0 ausgegeben wird. Bei den x oder X Konversionen (siehe unten) wird an Ergebnisse ungleich Null immer 0x oder 0X am Anfang eingefügt. Bei den e, E, f, g, und G Konversionen enthält das Ergebnis immer den Dezimalpunkt, selbst wenn auf den Dezimalpunkt keine Ziffern folgen. Bei den g, und G Konversionen werden außerdem, im Gegensatz zum normalen Verhalten, abschließende Nullen nicht entfernt.

field width Eine optionale Zeichenfolge von Dezimalziffern, die eine minimale Breite des Ergebnisfeldes spezifiziert. Falls der konvertierte Wert weniger Zeichen als die Feldbreite hat, wird er links (oder rechts, falls das Linksausrichtungsflag - spezifiziert worden ist) mit Leerzeichen bis zur Feldbreite aufgefüllt.

precision Die Präzision spezifiziert die minimale Anzahl von Ziffern, die für die d, o, x, oder X Konversionen ausgegeben werden soll (das Ergebnisfeld wird mit führenden Nullen aufgefüllt). Für die e und f Konversionen spezifiziert sie die Anzahl der Ziffern, die hinter dem Dezimalpunkt erscheinen sollen, für die g Konversion die maximale Anzahl von signifikanten Stellen, und für die s Konversion die maximale Anzahl von Zeichen, die von einem String ausgegeben werden sollen. Die Präzision hat die Form . gefolgt von einr Folge von Dezimalzahlen. Eine leere Folge wird als Null interpretiert.

conversion characters Ein Konversionszeichen spezifiziert die Art der Konversion, die auszuführen ist:

- **d,o,x,X** Das ganzzahlige Argument wird als vorzeichenbehaftete Dezimalzahl (d), vorzeichenlose Oktalzahl (o) oder vorzeichenlose Hezadezimalzahl (x und X) ausgegeben. Die x Konversion verwendet die Zeichen und Buchstaben 0123456789abcdef, während die X Konversion 0123456789ABCDEF verwendet. Die Präzision spezifiziert hier die minimale Anzahl von Ziffern, die ausgegeben werden sollen. Wenn die zu konvertierende Zahl mit weniger Ziffern repräsentiert werden kann als das angegebene Minimum, werden führende Nullen eingefügt. Wenn nichts anderes angegeben wird, ist die Präzision 1. Wenn die Zahl 0 mit einer Präzision von 0 ausgegeben werden soll, wird ein leerer String erzeugt.
- **f** Die Gleitkommazahl im Argument wird in dezimaler Notation in folgender Art ausgegeben: [-]ddd.ddd. Hierbei ist die Anzahl Ziffern hinter dem Dezimalpunkt gleich der Präzision. Wenn die Präzision nicht angegeben wird, werden sechs Ziffern ausgegeben; wenn die Präzision explizit auf 0 gesetzt wird, werden kein Dezimalpunkt und keine Nachkommastellen ausgegeben.
- **e,E** Die Gleitkommazahl im Argument wird in dezimaler Notation in folgender Art ausgegeben: [-]d.ddde±dd, wobei genau eine Zahl vor dem Dezimalpunkt steht und die Anzahl von Ziffern hinter dem Dezimalpunkt gleich der Präzision ist. Wenn die Präzision nicht angegeben wird, werden sechs Ziffern ausgegeben; wenn die Präzision explizit auf 0 gesetzt wird, werden kein Dezimalpunkt und keine Nachkommastellen ausgegeben. In der E Konversion wird der Exponent in der Ausgabe durch E anstelle von e gekennzeichnet. Der Exponent enthält immer mindestens zwei Ziffern. Falls die auszugebende Zahl mehr als zwei Ziffern im Exponenten benötigt, werden weitere Ziffern ausgegeben.
- **g,G** Die Gleitkommazahl im Argument wird in der Art der f oder e Konversionen (oder in der Art der E Konversion, falls die G Konversion spezifiziert worden ist) ausgegeben, wobei die Präzision die Anzahl der signifikanten Stellen angibt. Die Art der Konversion hängt von der zu konvertierenden Zahl ab; die e Konversion wird nur dann verwendet, wenn der Exponent des Ergebnisses der Konversion kleiner als -4 oder größer gleich der Präzision ist. Abschließende Nullen werden aus dem Ergebnis entfernt. Ein Dezimalpunkt wird nur ausgegeben, falls er von einer Ziffer gefolgt wird.
- **s** Von dem String-Argument werden solange Zeichen ausgegeben, bis das Ende des Strings oder die in der Präzision angegebene Anzahl von Zeichen erreicht ist. Wenn die Präzision nicht angegeben wird, wird sie als unendlich groß interpretiert und alle Zeichen bis zum Ende des Strings werden ausgegeben.
- **b** Ähnlich wie die s Konversion, nur daß der String sog. Backslash-Escape-Sequenzen (Zeichenfolgen, die mit
  beginnen) enthalten kann, die in die Zeichen, die sie repräsentieren umgewandelt werden.

In keinem Fall führt eine nicht existierende oder ungenügende Feldbreite zur Beschneidung des Feldes; wenn das Ergebnis der Konversion breiter als die Feldbreite ist, wird das Feld verbreitert, so daß das Ergebnis hineinpaßt.

Parameter

- ▷ **T** (input_control) . . . . . number(-array) ⇝ *real* / integer / string
  Eingabetupel.
- ▷ **Format** (input_control) . . . . . string ⇝ *string*
  Format-String.
- ▷ **String** (output_control) . . . . . string(-array) ⇝ *string*
  In Strings konvertiertes Eingabetupel.

Parallelisierungsinformation

`tuple_string` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_sub`

Modul

Operators not requiring licensing

13.7 Logische-Operationen

tuple_and (: : T1, T2 : And)

Logisches Und zweier Tupel.

`tuple_and` berechnet das logische Und der Eingabetupel `T1` und `T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- ▷ **T1** (input_control) . . . . . integer(-array) ⇝ *integer*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . integer(-array) ⇝ *integer*
  Eingabetupel 2.
- ▷ **And** (output_control) . . . . . integer(-array) ⇝ *integer*
  Logisches Und der Eingabetupel.

Parallelisierungsinformation

`tuple_and` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_or`, `tuple_xor`, `tuple_not`

Siehe auch

`tuple_band`, `tuple_bor`, `tuple_bxor`, `tuple_bnot`

Modul

Operators not requiring licensing

tuple_not (: : T : Not)

Logisches Nicht zweier Tupel.

`tuple_not` berechnet das logische Nicht des Eingabetupels `T`. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- ▷ **T** (input_control) . . . . . integer(-array) ⇝ *integer*
  Eingabetupel.
- ▷ **Not** (output_control) . . . . . integer(-array) ⇝ *integer*
  Binäres Nicht des Eingabetupels.

Parallelisierungsinformation

`tuple_not` ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Alternativen

`tuple_and`, `tuple_or`, `tuple_xor`

Siehe auch

`tuple_band`, `tuple_bor`, `tuple_bxor`, `tuple_bnot`

Modul

Operators not requiring licensing

tuple_or (: : T1, T2 : Or)

Logisches Oder zweier Tupel.

`tuple_or` berechnet das logische Oder der Eingabetupel `T1` und `T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.
- **Or** (output_control) ..... integer(-array) ⇝ *integer*
  Logisches Oder der Eingabetupel.

Parallelisierungsinformation

`tuple_or` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_and`, `tuple_xor`, `tuple_not`

Siehe auch

`tuple_band`, `tuple_bor`, `tuple_bxor`, `tuple_bnot`

Modul

Operators not requiring licensing

tuple_xor (: : T1, T2 : Xor)

Logisches Exklusiv-Oder zweier Tupel.

`tuple_xor` berechnet das logische Exklusiv-Oder der Eingabetupel `T1` und `T2`. Falls beide Tupel dieselbe Länge haben, werden die entsprechenden Elemente der beiden Tupel veknüpft. Ansonsten muß entweder `T1` oder `T2` die Länge 1 haben. In diesem Fall wird die Operation für jedes Element des längeren Tupels mit dem einzigen Element des anderen Tupels ausgeführt. Die Eingabezahlen müssen ganze Zahlen sein.

Parameter

- **T1** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 1.
- **T2** (input_control) ..... integer(-array) ⇝ *integer*
  Eingabetupel 2.
- **Xor** (output_control) ..... integer(-array) ⇝ *integer*
  Binäres Exklusiv-Oder der Eingabetupel.

Parallelisierungsinformation

`tuple_xor` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_and`, `tuple_or`, `tuple_not`

Siehe auch

tuple_band, tuple_bor, tuple_bxor, tuple_bnot

Modul

Operators not requiring licensing

13.8 Merkmale

tuple_deviation (: : Tuple : Deviation)

Berechnet die Standardabweichung aller Elemente eines Zahlentupels.

tuple_deviation berechnet die Standardabweichung aller Elemente des Eingabetupels Tuple und liefert sie in dem Ausgabeparameter Deviation als Gleitkommazahl zurück. Das Eingabetupel muß ausschließlich aus (ganzen oder Gleitkomma-)Zahlen bestehen.

Parameter

- **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- **Deviation** (output_control) . . . . . number(-array) ⇝ *real*
  Standardabweichung der Tupelelemente.

Parallelisierungsinformation

tuple_deviation ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_mean, tuple_sum, tuple_min, tuple_max, tuple_length

Modul

Operators not requiring licensing

tuple_length (: : Tuple : Length)

Gibt die Länge eines Tupels zurück.

tuple_length liefert die Anzahl der Elemente des Eingabetupels Tuple zurück.

Parameter

- **Tuple** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- **Length** (output_control) . . . . . number(-array) ⇝ *real*
  Anzahl der Elemente des Eingabetupels.

Parallelisierungsinformation

tuple_length ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_min, tuple_max, tuple_mean, tuple_deviation, tuple_sum

Modul

Operators not requiring licensing

tuple_max (: : Tuple : Max)

Gibt das maximale Element eines Tupels an.

tuple_max bestimmt das maximale Element von allen Elementen des Eingabetupels Tuple und liefert es in dem Ausgabeparameter Max zurück. Das Eingabetupel muß hierfür entweder durchwegs Strings enthalten oder ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Eine Mischung aus Strings und Zahlen ist nicht erlaubt.

Das Ergebnis besteht in einer Gleitkommazahl, sobald mindestens ein Element des Eingabetupels eine Gleitkommazahl ist. Falls alle Elemente von `Tuple` hingegen ganze Zahlen sind, so besteht auch `Max` aus einer ganzen Zahl.

Parameter

- ▷ **`Tuple`** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **`Max`** (output_control) . . . . . number(-array) ⇝ *real*
  Maximales Element aller Eingabetupelelemente.

Parallelisierungsinformation

`tuple_max` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_min`, `tuple_mean`, `tuple_deviation`, `tuple_sum`, `tuple_length`

Modul

Operators not requiring licensing

`tuple_mean ( : : Tuple : Mean )`

Berechnet den Durchschnittswert aller Elemente eines Zahlentupels.

`tuple_mean` berechnet den Durchschnittswert aller Elemente des Eingabetupels `Tuple` und liefert ihn in dem Ausgabeparameter `Mean` als Gleitkommazahl zurück. Das Eingabetupel muß ausschließlich aus (ganzen oder Gleitkomma-)Zahlen bestehen.

Parameter

- ▷ **`Tuple`** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **`Mean`** (output_control) . . . . . number(-array) ⇝ *real*
  Durchschnittswert der Tupelelemente.

Parallelisierungsinformation

`tuple_mean` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_deviation`, `tuple_sum`, `tuple_min`, `tuple_max`, `tuple_length`

Modul

Operators not requiring licensing

`tuple_min ( : : Tuple : Min )`

Gibt das minimale Element eines Tupels an.

`tuple_min` bestimmt das minimale Element von allen Elementen des Eingabetupels `Tuple` und liefert es in dem Ausgabeparameter `Min` zurück. Das Eingabetupel muß hierfür entweder durchwegs Strings enthalten oder ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Eine Mischung aus Strings und Zahlen ist nicht erlaubt. Das Ergebnis besteht in einer Gleitkommazahl, sobald mindestens ein Element des Eingabetupels eine Gleitkommazahl ist. Falls alle Elemente von `Tuple` hingegen ganze Zahlen sind, so besteht auch `Min` aus einer ganzen Zahl.

Parameter

- ▷ **`Tuple`** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- ▷ **`Min`** (output_control) . . . . . number(-array) ⇝ *real*
  Minimales Element aller Eingabetupelelemente.

Parallelisierungsinformation

`tuple_min` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_max, tuple_mean, tuple_deviation, tuple_sum, tuple_length

Modul

Operators not requiring licensing

tuple_sum (: : Tuple : Sum)

Berechnet die Summe aller Elemente eines Tupels.

tuple_sum berechnet die Summe aus allen Elementen des Eingabetupels Tuple und liefert sie in dem Ausgabeparameter Sum zurück. Das Eingabetupel muß hierfür entweder durchwegs Strings enthalten oder ausschließlich (ganze oder Gleitkomma-)Zahlen beinhalten. Eine Mischung aus Strings und Zahlen ist nicht erlaubt. Das Ergebnis besteht in einer Gleitkommazahl, sobald mindestens ein Element des Eingabetupels eine Gleitkommazahl ist. Falls alle Elemente von Tuple hingegen ganze Zahlen sind, so besteht auch Sum aus einer ganzen Zahl. Bei Strings wird zur Summenbildung statt der Addition die Konkatenation verwendet, so daß die einzelnen Elemente der Reihe nach verknüpft werden.

Parameter

- **Tuple** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel.
- **Sum** (output_control) ..... number(-array) ⇝ *real*
  Summe der Tupelelemente.

Parallelisierungsinformation

tuple_sum ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_mean, tuple_deviation, tuple_min, tuple_max, tuple_length

Modul

Operators not requiring licensing

13.9 String-Operationen

tuple_environment (: : Names : Values)

Einlesen einer oder mehrerer Umgebungsvariablen.

tuple_environment liest den Inhalt aller Umgebungsvariablen, deren Namen in dem Eingabetupel Names definiert sind und liefert ihn in den Elementen des Ausgabetupels Values zurück. Das Eingabetupel darf nur Strings enthalten. Enthält das Eingabetupel einen Namen, für den keine gültige Umgebungsvariable existiert, so wird für diesen Namen ein leerer String als Ergebnis zurückgeliefert.

Parameter

- **Names** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel mit den Namen der Umgebungsvariable(n).
- **Values** (output_control) ..... string(-array) ⇝ *string*
  Inhalt der Umgebungsvariable(n).

Parallelisierungsinformation

tuple_environment ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strstr, tuple_strrstr, tuple_strchr, tuple_strrchr, tuple_strlen, tuple_str_first_n, tuple_str_last_n, tuple_split

Modul

Operators not requiring licensing

tuple_split (: : T1, T2 : Splitted)

Teile Strings mittels Trennsymbolen in mehrere Strings auf.

tuple_split sucht in den einzelnen Strings des Eingabetupels T1 nach einem oder mehreren Trennsymbolen, die in dem Eingabetupel T2 definiert sind. Die untersuchten Strings werden dann in die Teil-Strings, die zwischen den Trennsymbolen stehen, aufgeteilt. Die beiden Eingabetupel T1 und T2 müssen ausschließlich aus Strings bestehen. Anderenfalls bricht tuple_split mit einem entsprechenden Fehler ab. Enthalten die Strings von T2 mehr als ein Zeichen, so werden alle in einem String enthaltenen Zeichen als Trennsymbole interpretiert. Beinhaltet T1 nur einen einzigen String, so wird dieser anhand der im Tupel T2 definierten Trennzeichen aufgeteilt. Besteht T1 zum Beispiel aus dem String “data1;data2:7;data3” und enthält T2 die beiden Strings “;” und “:;”, so beinhaltet das Ausgabetupel Splitted die Strings “data1”, “data2:7”, “data3” als Ergebnis der Trennung gemäß dem ersten Element von T2 und die Strings “data1”, “data2”, “7” und “data3” als Ergebnis der Trennung gemäß dem zweiten Element von T2. Besitzen beide Eingabetupel T1 und T2 gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, der erste String von T1 wird gemäß den Trennzeichen des ersten Strings von T2 aufgeteilt, der zweite String von T1 wird gemäß den Trennzeichen des zweiten Strings von T2 aufgeteilt und so weiter. Besteht T2 nur aus einem String, so dienen die dort definierten Trennzeichen zur Aufteilung sämtlicher Strings des Eingabetupels T1. Weisen die beiden Eingabetupel T1 und T2 eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_split mit einem entsprechenden Fehler ab.

Parameter

- ▷ **T1** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 2.
- ▷ **Splitted** (output_control) . . . . . string(-array) ⇝ *string*
  Zwischen den Trennzeichen aufgeteilte Strings.

Parallelisierungsinformation

tuple_split ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strstr, tuple_strrstr, tuple_strchr, tuple_strrchr, tuple_strlen, tuple_str_first_n, tuple_str_last_n, tuple_environment

Modul

Operators not requiring licensing

tuple_str_first_n (: : T1, T2 : Substring)

Ausschneiden aller Zeichen bis zur Position “n” aus einem Stringtupel.

tuple_str_first_n schneidet aus jedem String des Eingabetupels T1 die ersten Zeichen bis einschließlich zur Position “n” aus und liefert sie als einen neuen String in dem Ausgabetupel Substring zurück (Bemerkung: Die Position innerhalb eines Strings wird beginnend mit 0 gezählt). Die Zahl “n” wird hierbei durch das zweite Eingabetupel T2 definiert. Enthält T2 nur ein Element, so legt dieses Element die Zahl “n” für alle Strings von T1 fest. Beinhaltet T2 genauso viele Elemente wie T1, so legt das erste Element von T2 die Zahl “n” für den ersten String von T1 fest, das zweite Element von T2 legt die Zahl “n” für den zweiten String von T1 fest und so weiter. Enthält T2 mehr als ein Element und besteht T1 nur aus einem String, so werden aus diesem String mehrere Anfangssegmente ausgeschnitten, deren Längen durch die Elemente von T2 festgelegt sind. Beinhalten die beiden Eingabetupel mehr als ein Element und besitzen sie eine unterschiedliche Zahl an Elementen, so bricht tuple_str_first_n mit einer Fehlermeldung ab.

Parameter

- ▷ **T1** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *integer* / real
  Eingabetupel 2.
- ▷ **Substring** (output_control) . . . . . string(-array) ⇝ *string*
  Die ersten Zeichen bis zu Position “n” aller Eingabe-Strings.

Parallelisierungsinformation

`tuple_str_first_n` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_str_last_n`, `tuple_strstr`, `tuple_strrstr`, `tuple_strlen`, `tuple_strchr`, `tuple_strrchr`, `tuple_split`, `tuple_environment`

Modul

Operators not requiring licensing

tuple_str_last_n (: : T1, T2 : Substring)

Ausschneiden aller Zeichen ab der Position “n” aus einem Stringtupel.

`tuple_str_last_n` schneidet aus jedem String des Eingabetupels `T1` alle Zeichen bis zum String-Ende beginnend mit dem Zeichen an der Position “n” aus und liefert sie als einen neuen String in dem Ausgabetupel `Substring` zurück. Die Zahl “n” wird hierbei durch das zweite Eingabetupel `T1` definiert. Enthält `T2` nur ein Element, so legt dieses Element die Zahl “n” für alle Strings von `T1` fest. Beinhaltet `T2` genauso viele Elemente wie `T1`, so legt das erste Element von `T2` die Zahl “n” für den ersten String von `T1` fest, das zweite Element von `T2` legt die Zahl “n” für den zweiten String von `T1` fest und so weiter. Enthält `T2` mehr als ein Element und besteht `T1` nur aus einem String, so werden aus diesem String mehrere Endsegmente ausgeschnitten, deren Länge durch die Elemente von `T2` festgelegt ist. Beinhalten die beiden Eingabetupel mehr als ein Element und besitzen sie eine unterschiedliche Zahl an Elementen, so bricht `tuple_str_last_n` mit einer Fehlermeldung ab.

Parameter

- **T1** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel 1.
- **T2** (input_control) ..... number(-array) ⇝ *integer* / real
  Eingabetupel 2.
- **Substring** (output_control) ..... string(-array) ⇝ *string*
  Alle Zeichen von Position “n” bis zum String-Ende.

Parallelisierungsinformation

`tuple_str_last_n` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_str_last_n`, `tuple_strstr`, `tuple_strrstr`, `tuple_strlen`, `tuple_strchr`, `tuple_strrchr`, `tuple_split`, `tuple_environment`

Modul

Operators not requiring licensing

tuple_strchr (: : T1, T2 : Position)

Vorwärtssuche nach einem Zeichen in einem Tupel von Strings.

`tuple_strchr` sucht in dem Eingabetupel `T1` nach den in dem Eingabetupel `T2` definierten Zeichen. Beide Eingabetupel müssen durchgehend aus Strings bestehen. Anderenfalls bricht `tuple_strchr` mit einem entsprechenden Fehler ab. Enthalten die Strings von `T2` mehr als ein Zeichen, so wird ausschließlich das erste Zeichen in einem String berücksichtigt und alle folgenden Zeichen ignoriert. Besteht `T1` nur aus einem einzigen String, so werden in diesem alle in `T2` definierten Zeichen gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel `T2`. Ist die Suche nach einem Zeichen erfolgreich, so wird die Position (beginnend mit 0 für das erste Zeichen eines Strings), an der das Zeichen das erste Mal innerhalb des untersuchten Strings gefunden wurde, in dem Ausgabetupel `Position` zurückgegeben. Wurde ein Zeichen nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel `T1` und `T2` gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, das erste Zeichen im ersten Element von `T2` wird in dem ersten String von `T1`, das erste Zeichen des zweiten ELements von `T2` wird im zweiten String von `T1` gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht `T2` nur aus einem Element, so wird das dort definierte Zeichen in allen Strings des Eingabetupels

T1 gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel T1. Weisen die beiden Eingabetupel T1 und T2 eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strchr mit einem entsprechenden Fehler ab.

Parameter

- ▷ **T1** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel 1.
- ▷ **T2** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel 2.
- ▷ **Position** (output_control) ..... integer(-array) ⇝ *integer*
  Position des gesuchten Zeichens innerhalb des Strings.

Parallelisierungsinformation

tuple_strchr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strrchr, tuple_strstr, tuple_strrstr, tuple_strlen, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Operators not requiring licensing

tuple_strlen (: : T1 : Length)

Länge der einzelnen Strings eines String-Tupels.

tuple_strlen untersucht jeden String des Eingabetupels T1 auf dessen Länge und liefert die Länge der einzelnen Strings in dem Ausgabetupel Length zurück. Das Eingabetupel T1 darf nur Strings enthalten. Ansonsten bricht tuple_strlen mit einer entsprechenden Fehlermeldung ab.

Parameter

- ▷ **T1** (input_control) ..... string(-array) ⇝ *string*
  Eingabetupel.
- ▷ **Length** (output_control) ..... integer(-array) ⇝ *integer*
  Länge der Strings des Eingabetupels.

Parallelisierungsinformation

tuple_strlen ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strstr, tuple_strrstr, tuple_strchr, tuple_strrchr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Operators not requiring licensing

tuple_strrchr (: : T1, T2 : Position)

Rückwärtssuche nach einem Zeichen in einem Tupel von Strings.

tuple_strrchr sucht in dem Eingabetupel T1 nach den in dem Eingabetupel T2 definierten Zeichen. Beide Eingabetupel müssen durchwegs aus Strings bestehen. Anderenfalls bricht tuple_strrchr mit einem entsprechenden Fehler ab. Enthalten die Strings von T2 mehr als ein Zeichen, so wird ausschließlich das erste Zeichen in einem String berücksichtigt und alle folgenden Zeichen ignoriert. Die Suche nach einem Zeichen geschieht rückwärts also vom Ende zum Anfang des zu untersuchenden Strings. Besteht T1 nur aus einem einzigen String, so werden in diesem alle in T2 definierten Zeichen gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel T2. Ist die Suche nach einem Zeichen erfolgreich, so wird die Position (beginnend mit 0 für das erste Zeichen eines Strings), an der das Zeichen das erste Mal innerhalb des untersuchten Strings gefunden wurde, in dem Ausgabetupel Position zurückgegeben. Wurde ein Zeichen nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel T1 und T2 gleich

viele Elemente, so werden sie elementweise bearbeitet, das heißt, das erste Zeichen im ersten Element von T2 wird in dem ersten String von T1, das erste Zeichen des zweiten ELements von T2 wird im zweiten String von T1 gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht T2 nur aus einem Element, so wird das dort definierte Zeichen in allen Strings des Eingabetupels T1 gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel T1. Weisen die beiden Eingabetupel T1 und T2 eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strrchr mit einem entsprechenden Fehler ab.

Parameter

- ▷ **T1** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 2.
- ▷ **Position** (output_control) . . . . . integer(-array) ⇝ *integer*
  Position des gesuchten Zeichens im String.

Parallelisierungsinformation

tuple_strrchr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strchr, tuple_strstr, tuple_strrstr, tuple_strlen, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Operators not requiring licensing

tuple_strrstr (: : T1, T2 : Position)

Rückwärtssuche nach einem Teilstring in einem Tupel von Strings.

tuple_strrstr sucht in dem Eingabetupel T1 nach den in dem Eingabetupel T2 definierten Strings. Beide Eingabetupel müssen durchgehend aus Strings bestehen. Anderenfalls bricht tuple_strrstr mit einem entsprechenden Fehler ab. Besteht T1 nur aus einem einzigen String, so werden in diesem alle in T2 definierten Strings gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel T2. Die Suche geschieht rückwärts also vom Ende zum Anfang des zu untersuchenden Strings. Ist die Suche nach einem String erfolgreich, so wird die Position, an der der gesuchte String innerhalb des untersuchten Strings das erste Mal gefunden wurde, in dem Ausgabetupel Position zurückgegeben (die Position innerhalb eines Strings wird beginnend mit 0 gezählt). Wurde ein String nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel T1 und T2 gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, der erste String von T2 wird in dem ersten String von T1, der zweite von T2 im zweiten von T1 gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht T2 nur aus einem String, so wird dieser String in allen Strings des Eingabetupels T1 gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel T1. Weisen die beiden Eingabetupel T1 und T2 eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strrstr mit einem entsprechenden Fehler ab.

Parameter

- ▷ **T1** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 2.
- ▷ **Position** (output_control) . . . . . integer(-array) ⇝ *integer*
  Position des gesuchten Strings in den untersuchten Strings.

Parallelisierungsinformation

tuple_strrstr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strstr, tuple_strlen, tuple_strchr, tuple_strrchr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Operators not requiring licensing

tuple_strstr (: : T1, T2 : Position)

Vorwärtssuche nach einem Teilstring in einem Tupel von Strings.

tuple_strstr sucht in dem Eingabetupel T1 nach den in dem Eingabetupel T2 definierten Strings. Beide Eingabetupel müssen durchgehend aus Strings bestehen. Anderenfalls bricht tuple_strstr mit einem entsprechenden Fehler ab. Besteht T1 nur aus einem einzigen String, so werden in diesem alle in T2 definierten Strings gesucht. Das Ausgabetupel besteht also in diesem Fall aus genauso vielen Elementen wie das Eingabetupel T2. Ist die Suche nach einem String erfolgreich, so wird die Position, an der der gesuchte String innerhalb des untersuchten Strings das erste Mal gefunden wurde, in dem Ausgabetupel Position zurückgegeben (die Position wird beginnende mit 0 gezählt). Wurde ein String nicht gefunden, so wird für dessen Position der Wert -1 zurückgegeben. Enthalten beide Eingabetupel T1 und T2 gleich viele Elemente, so werden sie elementweise bearbeitet, das heißt, der erste String von T2 wird in dem ersten String von T1, der zweite von T2 im zweiten von T1 gesucht und so weiter. Die einzelnen Elemente des Ausgabetupels enthalten dann das Resultat der elementweisen Suche. Besteht T2 nur aus einem String, so wird dieser String in allen Strings des Eingabetupels T1 gesucht. Das Ausgabetupel Position besitzt dann also genauso viele Elemente wie das Eingabetupel T1. Weisen die beiden Eingabetupel T1 und T2 eine unterschiedliche Anzahl an Elementen auf, die von 1 verschieden ist, so bricht tuple_strstr mit einem entsprechenden Fehler ab.

Parameter

- **T1** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 1.
- **T2** (input_control) . . . . . string(-array) ⇝ *string*
  Eingabetupel 2.
- **Position** (output_control) . . . . . integer(-array) ⇝ *integer*
  Position des gesuchten Strings innerhalb des untersuchten Strings.

Parallelisierungsinformation

tuple_strstr ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_strrstr, tuple_strlen, tuple_strchr, tuple_strrchr, tuple_str_first_n, tuple_str_last_n, tuple_split, tuple_environment

Modul

Operators not requiring licensing

13.10 Vergleich

tuple_equal (: : T1, T2 : Equal)

Test, ob zwei Tupel gleich sind.

tuple_equal testet, ob die beiden Eingabetupel T1 und T2 gleich sind. Die Eingabetupel werden elementweise verglichen. Zwei Tupelelemente gelten als gleich, wenn sie beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sind und dieselbe Belegung aufweisen. Darüberhinaus müssen beide Eingabetupel über dieselbe Anzahl an Elementen verfügen, um als gleich zu gelten.

Parameter

- **T1** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- **T2** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- **Equal** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Parallelisierungsinformation

tuple_equal ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

tuple_not_equal, tuple_less, tuple_greater, tuple_less_equal, tuple_greater_equal

Modul

Operators not requiring licensing

tuple_greater (: : T1, T2 : Greater)

Test, ob ein Tupel größer als ein zweites Tupel ist.

`tuple_greater` gibt an, ob das Eingabetupel `T1` größer als das Eingabetupel `T2` ist. Ein Tupel `T1` gilt als größer gegenüber einem Tupel `T2`, falls es sich entweder in einem elementweisen Vergleich als größer erweist oder falls `T1` - im Falle, daß es sich für den elementweisen Vergleich zu `T2` nicht als größer erwiesen hat - mehr Elemente als `T2` besitzt. Beim elementweisen Vergleich werden die einzelnen Elemente von `T1` und `T2` der Reihe nach miteinander verglichen (das erste Element von `T1` mit dem ersten Element von `T2`, das zweite Element von `T1` mit dem zweiten Element von `T2` usw.). Ist hierbei ein Element von `T1` größer als das zugehörige Element von `T2`, so gilt `T1` als größer gegenüber `T2`. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_greater` mit einer entsprechenden Fehlermeldung ab.

Parameter

- **T1** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- **T2** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- **Greater** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Parallelisierungsinformation

`tuple_greater` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_greater_equal`, `tuple_less`, `tuple_less_equal`, `tuple_equal`, `tuple_not_equal`

Modul

Operators not requiring licensing

tuple_greater_equal (: : T1, T2 : Greatereq)

Test, ob ein Tupel größer oder gleich einem zweiten Tupel ist.

`tuple_greater_equal` gibt an, ob das Eingabetupel `T1` größer oder gleich dem Tupel `T2` ist. Ein Tupel `T1` gilt als größer oder gleich einem Tupel `T2`, falls es nicht kleiner als `T2` ist. Ein Tupel `T1` gilt als kleiner gegenüber einem Tupel `T2`, falls es sich entweder in einem elementweisen Vergleich als kleiner erweist oder falls `T1` - im Falle, daß es sich für den elementweisen Vergleich zu `T2` nicht als kleiner erwiesen hat - weniger Elemente als `T2` besitzt. Beim elementweisen Vergleich werden die einzelnen Elemente von `T1` und `T2` der Reihe nach miteinander verglichen (das erste Element von `T1` mit dem ersten Element von `T2`, das zweite Element von `T1` mit dem zweiten Element von `T2` usw.). Ist hierbei ein Element von `T1` kleiner als das zugehörige Element von `T2`, so gilt `T1` als kleiner gegenüber `T2`. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_greater_equal` mit einer entsprechenden Fehlermeldung ab.

Parameter

- **T1** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- **T2** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- **Greatereq** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Parallelisierungsinformation

`tuple_greater_equal` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_greater`, `tuple_less`, `tuple_less_equal`, `tuple_equal`, `tuple_not_equal`

Modul

Operators not requiring licensing

tuple_less (: : T1, T2 : Less)

Test, ob ein Tupel kleiner als ein zweites Tupel ist.

`tuple_less` gibt an, ob das Eingabetupel `T1` kleiner als das Eingabetupel `T2` ist. Ein Tupel `T1` gilt als kleiner gegenüber einem Tupel `T2`, falls es sich entweder in einem elementweisen Vergleich als kleiner erweist oder falls `T1` - im Falle, daß es sich für den elementweisen Vergleich zu `T2` nicht als kleiner erwiesen hat - weniger Elemente als `T2` besitzt. Beim elementweisen Vergleich werden die einzelnen Elemente von `T1` und `T2` der Reihe nach miteinander verglichen (das erste Element von `T1` mit dem ersten Element von `T2`, das zweite Element von `T1` mit dem zweiten Element von `T2` usw.). Ist hierbei ein Element von `T1` kleiner als das zugehörige Element von `T2`, so gilt `T1` als kleiner gegenüber `T2`. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_less` mit einer entsprechenden Fehlermeldung ab.

Parameter

- **T1** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- **T2** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- **Less** (output_control) ..... integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Parallelisierungsinformation

`tuple_less` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_less_equal`, `tuple_greater`, `tuple_greater_equal`, `tuple_equal`, `tuple_not_equal`

Modul

Operators not requiring licensing

tuple_less_equal (: : T1, T2 : Lesseq)

Test, ob ein Tupel kleiner oder gleich einem zweiten Tupel ist.

`tuple_less_equal` gibt an, ob das Eingabetupel `T1` kleiner oder gleich dem Tupel `T2` ist. Ein Tupel `T1` gilt als kleiner oder gleich einem Tupel `T2`, falls es nicht größer als `T2` ist. Ein Tupel `T1` gilt als größer gegenüber einem Tupel `T2`, falls es sich entweder in einem elementweisen Vergleich als größer erweist oder falls `T1` - im Falle, daß es sich für den elementweisen Vergleich zu `T2` nicht als größer erwiesen hat - mehr Elemente als `T2` besitzt. Beim elementweisen Vergleich werden die einzelnen Elemente von `T1` und `T2` der Reihe nach miteinander verglichen (das erste Element von `T1` mit dem ersten Element von `T2`, das zweite Element von `T1` mit dem zweiten Element von `T2` usw.). Ist hierbei ein Element von `T1` größer als das zugehörige Element von `T2`, so gilt `T1` als größer gegenüber `T2`. Als Voraussetzung für den elementweisen Vergleich müssen die miteinander verglichenen Elemente der Eingabetupel entweder beide (ganze oder Gleitkomma-)Zahlen oder beide Strings sein. Falls dies nicht der Fall ist, bricht `tuple_less_equal` mit einer entsprechenden Fehlermeldung ab.

Parameter

- **T1** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- **T2** (input_control) ..... number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- **Lesseq** (output_control) ..... integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Parallelisierungsinformation

`tuple_less_equal` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_less`, `tuple_greater`, `tuple_greater_equal`, `tuple_equal`, `tuple_not_equal`

Modul

Operators not requiring licensing

tuple_not_equal (: : T1, T2 : Nequal)

Test, ob sich zwei Tupel unterscheiden.

`tuple_not_equal` testet, ob die beiden Eingabetupel `T1` und `T2` verschieden sind. Zwei Tupel gelten als verschieden, sobald sie unterschiedlich viele Elemente beinhalten oder sich in mindestens einer Position beim elementweisen Vergleich unterscheiden. Zwei miteinander verglichene Tupelelemente gelten als verschieden, falls sie entweder aus nicht vergleichbaren Typen bestehen (zum Beispiel eine Gleitkommazahl und ein String), oder falls sie zwar vergleichbare Typen besitzen, jedoch eine unterschiedliche Belegung (Werte) aufweisen.

Parameter

- ▷ **T1** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 1.
- ▷ **T2** (input_control) . . . . . number(-array) ⇝ *integer* / real / string
  Eingabetupel 2.
- ▷ **Nequal** (output_control) . . . . . integer ⇝ *integer*
  Ergebnis des Vergleichs der Eingabetupel.

Parallelisierungsinformation

`tuple_not_equal` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Alternativen

`tuple_equal`, `tuple_less`, `tuple_greater`, `tuple_less_equal`, `tuple_greater_equal`

Modul

Operators not requiring licensing

Kapitel 14

XLD

14.1 Generierung

gen_contour_polygon_xld (: Contour : Row, Col :)

Erzeugen einer XLD-Kontur aus einem Polygon (als Tupel).

gen_contour_polygon_xld erzeugt aus einem Polygon, das in den Tupeln Row und Col übergeben wird, eine XLD-Kontur Contour. Dieser Operator ist dazu gedacht, Segmentationsergebnisse, die nicht in der Kern-Bibliothek erzeugt worden sind, einzuspeichern, um auf ihnen Funktionen wie Polygonapproximation oder Erkennung von Parallelen auszuführen.

Parameter

- ▷ **Contour** (output_object) . . . . . xld_cont ⇝ *Hobject*
  Ausgabe-Kontur.
- ▷ **Row** (input_control) . . . . . number-array ⇝ *real* / integer
  Zeilen-Koordinaten des Polygons.
  Defaultwert : '[0,1,2,2,2]'
  Wertevorschläge : Row ∈ {0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500}
- ▷ **Col** (input_control) . . . . . number-array ⇝ *real* / integer
  Spalten-Koordinaten des Polygons.
  Defaultwert : '[0,0,0,1,2]'
  Wertevorschläge : Col ∈ {0, 1, 2, 3, 4, 5, 10, 20, 50, 100, 200, 500}

Parallelisierungsinformation

gen_contour_polygon_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

get_region_contour

Mögliche Nachfolgerfunktionen

smooth_contours_xld, gen_polygons_xld

Siehe auch

gen_contours_skeleton_xld

Modul

Sub-pixel operators

gen_contour_region_xld (Regions : Contours : Mode :)

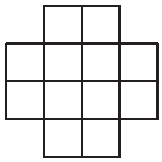
Erzeugen von XLD-Konturen aus Regionen.

gen_contour_region_xld erzeugt aus einer oder mehreren Konturen XLD-Konturen Contours. Dieser Operator ist dazu gedacht, Segmentationsergebnisse, die als Regionen erzeugt worden sind, umzuwandeln, um auf

ihnen Funktionen wie Polygonapproximation oder Erkennung von Parallelen auszuführen. Für jede Zusammenhangskomponente der Eingaberegionen wird eine geschlossene Kontur der Randes erzeugt. Der Parameter `Mode` kann folgende Werte annehmen:

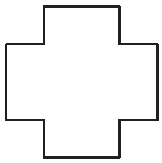
- **'center'**: Die Mittelpunkte der Randpixel werden als Konturpunkte verwendet.
- **'border'**: Der äußere Rand der Randpixel wird als Konturpunkte verwendet.

Der Unterschied zwischen den zwei Modi kann am Beispiel der folgenden Region leicht gesehen werden:

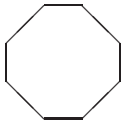


wobei □ ein einzelnes Pixel symbolisiert.

Die mit **'border'** und **'center'** berechneten Konturen sehen wie folgt aus:



'border'



'center'

Das bedeutet zum Beispiel, daß Konturen, die mit **'border'** erzeugt werden, im allgemeinen eine viel größere Euklidsche Länge (siehe `length_xld`) haben als Konturen, die mit **'center'** erzeugt werden. Dies liegt daran, daß für diagonale Randelemente bei **'border'** zwei Kontursegmente der Länge 1 erzeugt werden, während bei **'center'** ein einzelnes Element der Länge $\sqrt{2}$ verwendet wird. Offensichtlich haben auch andere Merkmale, z.B. die Fläche (siehe `area_center_xld`), verschiedene Werte.

Parameter

▷ **`Regions`** (input_object) . . . . . region(-array) ⇝ *Hobject*
Eingaberegionen.

▷ **`Contours`** (output_object) . . . . . xld_cont(-array) ⇝ *Hobject*
Ausgabe-Konturen.

▷ **`Mode`** (input_control) . . . . . string ⇝ *string*
Art des Konturerzeugung.
Defaultwert : 'border'
Werteliste : `Mode` ∈ {'border', 'center'}

Parallelisierungsinformation

`gen_contour_region_xld` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Nachfolgerfunktionen

`smooth_contours_xld`, `gen_polygons_xld`

Alternativen

`gen_contour_polygon_xld`, `get_region_contour`

Siehe auch

`gen_contours_skeleton_xld`

Modul

Sub-pixel operators

`gen_contours_skeleton_xld` (Skeleton : Contours : Length, Mode :)

Umwandlung eines Skeletts in XLD-Konturen.

Mit gen_contours_skeleton_xld wird das in Skeleton als Region übergebene Skelett (z.B. Bildkanten) in die Konturdarstellung umgerechnet. Es wird angenommen, daß der Großteil der Region nur ein Pixel breit ist (siehe skeleton). Das Bild wird zunächst so transformiert, daß es nur noch Linienzüge in 8-Nachbarschaft enthält. Dabei werden jedoch Sonderfälle ausgenommen: Punkte, deren 8-Nachbarschaft auf einen gemeinsamen Endpunkt (Kreuzungspunkt) von drei oder vier Konturen schließen läßt, bleiben (in allen vier Rotationen) erhalten:

$$\begin{matrix}0&0&1\\1&1&0\\0&1&0\end{matrix}\qquad\begin{matrix}0&1&0\\0&1&1\\0&1&0\end{matrix}\qquad\begin{matrix}0&1&0\\1&1&1\\0&1&0\end{matrix}$$

In einem zweiten Schritt werden alle Kreuzungspunkte markiert, wobei sechs verschiedene charakteristische 8-Nachbarschaften in allen vier Rotationen unterschieden werden:

$$\begin{matrix}1&0&1\\0&2&0\\0&0&1\end{matrix}\quad\begin{matrix}1&0&1\\0&2&0\\1&0&1\end{matrix}\quad\begin{matrix}1&0&0\\0&2&1\\0&1&0\end{matrix}\quad\begin{matrix}1&0&0\\0&2&1\\1&0&0\end{matrix}\quad\begin{matrix}0&1&0\\0&2&1\\0&1&0\end{matrix}\quad\begin{matrix}0&1&0\\1&2&1\\0&1&0\end{matrix}$$

wobei 0 = Hintergrund, 1 = Vordergrund und 2 = Kreuzungspunkt.

Anschließend werden die Konturen erfaßt, die einschließlich End- und Kreuzungspunkten mindestens Length Punkte lang sind. Die mit gen_contours_skeleton_xld erzeugten Konturen enden immer an Kreuzungs- oder Endpunkten. Bei geschlossenen Konturen liegt der erste Punkt in der 8-Nachbarschaft des letzten erfaßten Punktes der Kontur. Um die Nachbarschaft von Konturen festzustellen, genügt die Untersuchung der jeweils ersten und letzten Punkte der Konturen.

Die Unterbrechung der Konturen an Kreuzungspunkten hat zur Folge, daß auch längere Linienzüge durch Kreuzen von kurzen Linien (auch wenn diese weniger als Length Punkte lang sind) in mehrere Konturen zerfallen (Mode 'filter'). Dies wird mit dem Filtermodus (Mode) 'generalize1' vermieden. In diesem Fall werden die Konturen so erzeugt, wie wenn die Konturen kürzer als Length Punkte nicht vorhanden wären. Damit auch Linienzüge, die durch Kreuzen von kurzen Linien in sehr kurze Konturen zerfallen, erhalten bleiben, kann der Filtermodus (Mode) 'generalize2' eingestellt werden. In diesem Fall bleiben Linienstücke erhalten, deren beide Endpunkte Kreuzungspunkte mit anderen Linien sind, auch wenn sie kürzer als Length Punkte sind.

Parameter

- ▷ **Skeleton** (input_object) ..... region ⇝ *Hobject*
  Skelett für die Konturberechnung.
- ▷ **Contours** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Ausgabe-Konturen.
- ▷ **Length** (input_control) ..... integer ⇝ *integer*
  Mindestlänge zu erfassender Konturen.
  Defaultwert : 1
  Wertevorschläge : Length ∈ {1, 2, 3, 5, 10, 20}
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Kontur-Filtermodus.
  Defaultwert : 'filter'
  Werteliste : Mode ∈ {'filter', 'generalize1', 'generalize2'}

Parallelisierungsinformation

gen_contours_skeleton_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

skeleton

Mögliche Nachfolgerfunktionen

smooth_contours_xld, get_contour_xld, gen_polygons_xld

Siehe auch

edges_image, threshold, get_region_contour

Modul

Sub-pixel operators

gen_ellipse_contour_xld (: ContEllipse : Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder, Resolution :)

Erzeugung einer XLD-Kontur aus einem Ellipsenbogen.

gen_ellipse_contour_xld generiert einen oder mehrere Ellipsenbögen bzw. geschlossene Ellipsen. Die Ellipsen werden dabei durch ihren Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die Länge der großen Radius1 bzw. kleinen Halbachse Radius2 beschrieben. Ellipsenbögen werden zusätzlich durch den Anfangs- und Endwinkel StartPhi und EndPhi des Start- bzw. Endpunktes, sowie den zugehörigen Umlaufsinn PointOrder charakterisiert. Die Winkel werden dabei im Ellipsenkoordinatensystem bezogen auf die Hauptachse der Ellipse mathematisch positiv im Intervall $[0, 2\pi]$ angegeben. Die beiden Hauptpole der Ellipse haben demnach die Winkel 0 bzw. π, die Nebenpole die Winkel $\pi/2$ bzw. $3\pi/2$. Um eine geschlossene Ellipse zu erhalten, müssen die Werte 0 und 2π (bei positivem Umlaufsinn) übergeben werden. Die Auflösung der resultierenden XLD Konturen ContEllipse wird über den Parameter Resolution festgelegt: Er bestimmt den maximalen euklidischen Abstand zwischen benachbarten Konturpunkten.

Parameter

▷ **ContEllipse** (output_object) . . . . . xld_cont(-array) ⇝ *Hobject*
Ausgabe-Kontur.

▷ **Row** (input_control) . . . . . ellipse.center.y(-array) ⇝ *real*
Zeilenkoordinate des Mittelpunktes der Ellipse.
Defaultwert : 200.0
Wertevorschläge : Row ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0}

▷ **Column** (input_control) . . . . . ellipse.center.x(-array) ⇝ *real*
Spaltenkoordinate des Mittelpunktes der Ellipse.
Defaultwert : 200.0
Wertevorschläge : Column ∈ {0.0, 10.0, 20.0, 50.0, 100.0, 200.0, 300.0, 400.0}

▷ **Phi** (input_control) . . . . . ellipse.angle.rad(-array) ⇝ *real*
Orientierung der Hauptachse [rad].
Defaultwert : 0.0
Wertevorschläge : Phi ∈ {-1.178097, -0.785398, -0.392699, 0.0, 0.392699, 0.785398, 1.178097}
Restriktion : (Phi ≥ 0) ∧ (Phi ≤ 6.283185307)

▷ **Radius1** (input_control) . . . . . ellipse.radius1(-array) ⇝ *real*
Länge der großen Halbachse.
Defaultwert : 100.0
Wertevorschläge : Radius1 ∈ {2.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Restriktion : Radius1 > 0

▷ **Radius2** (input_control) . . . . . ellipse.radius2(-array) ⇝ *real*
Länge der kleinen Halbachse.
Defaultwert : 50.0
Wertevorschläge : Radius2 ∈ {1.0, 2.0, 4.0, 5.0, 10.0, 20.0, 50.0, 100.0, 256.0, 300.0, 400.0}
Restriktion : Radius2 ≥ 0

▷ **StartPhi** (input_control) . . . . . real(-array) ⇝ *real*
Winkel des Startpunktes [rad].
Defaultwert : 0.0
Wertevorschläge : StartPhi ∈ {0.0, 0.78539, 1.57079, 2.35619, 3.14159, 3.92699, 4.71238, 5.49778, 6.28318}
Restriktion : (StartPhi ≥ 0) ∧ (StartPhi ≤ 6.283185307)

▷ **EndPhi** (input_control) . . . . . real(-array) ⇝ *real*
Winkel des Endpunktes [rad].
Defaultwert : 6.28318
Wertevorschläge : EndPhi ∈ {0.0, 0.78539, 1.57079, 2.35619, 3.14159, 3.92699, 4.71238, 5.49778, 6.28318}
Restriktion : (EndPhi ≥ 0) ∧ (EndPhi ≤ 6.283185307)

▷ **PointOrder** (input_control) . . . . . string(-array) ⇝ *string*
mathematischer Umlaufsinn.
Defaultwert : 'positive'
Werteliste : PointOrder ∈ {'positive', 'negative'}

▷ **Resolution** (input_control) . . . . . real ⇝ *real*
Auflösung: Maximaler Abstand zwischen benachbarten Konturpunkten.
Defaultwert : 1.5
Wertevorschläge : Resolution ∈ {0.5, 1.0, 1.5, 2.0, 2.5, 3.0}
Restriktion : Resolution > 0

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert gen_ellipse_contour_xld den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

gen_ellipse_contour_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

draw_ellipse

Mögliche Nachfolgerfunktionen

disp_xld, get_points_ellipse

Modul

Sub-pixel operators

gen_parallels_xld (Polygons : Parallels : Len, Dist, Alpha, Merge :)

Suche nach parallelen XLD-Polygonen.

Mit gen_parallels_xld werden die in Polygons übergebenen Polygone auf Parallelität hin untersucht. Die resultierenden parallelen Teilstücke der Polygone werden unter Parallels ausgegeben. Falls der Parameter Merge auf **'true'** gestetzt wird, werden nebeneinanderliegende parallele Teilstcke in einer einzigen Relation gespeichert. Ansonsten wird für jedes Paar von parallelen Liniensegmenten eine separate Parallelenrelation ausgegeben. Als Kriterien für Parallelität gehen der maximale Abstand Dist der Polygone, eine maximale Winkeltoleranz Alpha (Bogenmaß), und eine minimale Länge der einzelnen berücksichtigten Polygonlinien ein. Zudem muß noch ein errechneter gemeinsamer Überlappungsbereich vorhanden sein. Als Nebeneffekt wird zu jedem parallelen Linienpaar ein sog. Qualitätsfaktor berechnet. Dieser ergibt sich aus der normierten Winkeldifferenz und des normierten Quotienten aus der gemeinsamen Überlappungsstrecke und den Längen der beteiligten Linien:

$$quality = \frac{\pi - \delta\alpha}{\pi/2} \frac{2overlap}{len_1 + len_2} \qquad \text{mit} \quad 0 \leq q \leq 1$$

Dabei ist @delta_alphaa$\delta\alpha$ die Winkeldifferenz der Polygonsegmente, $overlap$ der Überlappungsbereich, len_1 und l_2 die Länge der Polygonsegmente und $quality$ der errechnete Qualitätsfaktor.

Dieser Wert stellt ein Maß fuer die Parallelität dar (je größer dieser Wert, desto „paralleler“ die Linien). Anschließend werden die Qualitäts-Faktoren aller parallelen Linien in gemeinsamen Polygonen, gewichtet nach ihrem Anteil an der gemeinsamen Überlappung, aufsummiert.

Parameter

▷ **Polygons** (input_object) . . . . . xld_poly-array ⇝ *Hobject*
Eingabepolygone.
▷ **Parallels** (output_object) . . . . . xld_para-array ⇝ *Hobject*
Parallelenrelationen.
▷ **Len** (input_control) . . . . . number ⇝ *real* / integer
Mindestlänge der einzelnen Polygonlinien.
Defaultwert : 10.0
Wertevorschläge : Len ∈ {5.0, 10.0, 15.0, 20.0}
Restriktion : Len > 0.0

▷ **Dist** (input_control) ..... number ⇝ *real* / integer
Maximalabstand der einzelnen Polygonlinien.
Defaultwert : 30.0
Wertevorschläge : Dist ∈ {20.0, 25.0, 30.0, 40.0, 50.0, 75.0}
Restriktion : Dist > 0.0

▷ **Alpha** (input_control) ..... number ⇝ *real* / integer
Maximale Winkeltoleranz der einzelnen Polygonlinien.
Defaultwert : 0.15
Wertevorschläge : Alpha ∈ {0.05, 0.10, 0.15, 0.20, 0.30}
Restriktion : (0 ≤ Alpha) ∧ (Alpha ≤ (pi/2))

▷ **Merge** (input_control) ..... string ⇝ *string*
Sollen nebeneinanderliegende Parallelenrelationen zusammengefügt werden?
Defaultwert : 'true'
Werteliste : Merge ∈ {'true', 'false'}

Parallelisierungsinformation

gen_parallels_xld ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_polygons_xld

Mögliche Nachfolgerfunktionen

mod_parallels_xld, get_parallels_xld

Modul

Sub-pixel operators

gen_polygons_xld (Contours : Polygons : Type, Alpha :)

Polygonapproximation von XLD-Konturen.

Mit gen_polygons_xld werden, unter Angabe eines Approximatinsverfahrens Type und eines Schwellenwertes Alpha, XLD-Konturen, die in Contours abgespeichert sind, approximiert. Es werden Polygone berechnet, mit deren Stützpunkten die Konturen angenähert werden können. Es können dabei sowohl offene als auch geschlossene Konturen approximiert werden. Die gewonnenen Stützpunkte werden in dem Parameter Polygons zurückgegeben.

Es stehen die Verfahren von Ramer, Ray und Sato zur Verfügung. Beim Verfahren von Ramer werden die Konturen so approximiert, daß der euklidische Abstand eines jeden Konturpunktes von der Polygonlinie höchstens Alpha Pixeleinheiten beträgt. Das Verfahren von Ray kommt ohne Approximationsschwelle aus. Der Wert Alpha wird ignoriert. Hier wird eine Punktsequenz durch eine möglichst lange Linie approximiert, wobei die Summe der Abstände aller Punkte zum angenäherten Segment minimal sein soll. Beim Verfahren von Sato ergibt sich ein Stützpunkt an der Stelle innerhalb einer Punktsequenz, an der der Abstand zu den Endpunkten dieser Punktsequenz maximal ist. Der Gesamtapproximationsfehler ergibt sich dabei in jeder Iteration zu $(L - L')/L$, wobei L die euklidische Konturlänge ist und L' die approximierte Länge wiedergibt.

Parameter

▷ **Contours** (input_object) ..... xld_cont-array ⇝ *Hobject*
Konturen, die approximiert werden sollen.

▷ **Polygons** (output_object) ..... xld_poly-array ⇝ *Hobject*
Approximierende Polygone.

▷ **Type** (input_control) ..... string ⇝ *string*
Name des Polygonverfahrens.
Defaultwert : 'ramer'
Werteliste : Type ∈ {'ramer', 'ray', 'sato'}

▷ **Alpha** (input_control) ..... number ⇝ *real* / integer
Schwellenwert der Polygonapproximation.
Defaultwert : 2.0
Wertevorschläge : Alpha ∈ {1.0, 1.5, 2.0, 3.0, 4.0}
Restriktion : Alpha > 0.0

Parallelisierungsinformation

`gen_polygons_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Mögliche Nachfolgerfunktionen

`gen_parallels_xld`

Siehe auch

`get_region_polygon`

Modul

Sub-pixel operators

`mod_parallels_xld` (`Parallels, Image : ModParallels, ExtParallels : Quality, MinGray, MaxGray, MaxStandard` :)

Suche nach parallelen Polygonen mit einschließender homogener Fläche.

`mod_parallels_xld` überprüft die Grauwerte im Bild `Image` zwischen zwei parallelen Linien (Überlappungsbereich), die unter `Parallels` abgespeichert sind, auf Homogenität.

Es werden nur parallele Polygone betrachtet, deren Quality-Faktor größer gleich `Quality` ist. Zwischen je zwei parallelen Linien wird deren gemeinsame zentrale Überlappungsstrecke entlanggelaufen und jeweils der dazu senkrechte Grauwertschnitt errechnet (cross section).

In einem ersten Durchlauf wird der mittlere Grauwert für jede Schnittgerade ermittelt. In einem zweiten Durchlauf werden nun die Standardabweichungen jeder Schnittgeraden bezüglich ihres mittleren, empirisch ermittelten Grauwertes berechnet.

Falls der mittlere Grauwert einer betrachteten Fläche im Intervall [`MinGray`,`MaxGray`] liegt und das Mittel aus allen Standardabweichungen kleiner als die obere Schranke `MaxStandard` ist, so werden die parallelen Linien in `ModParallels` als modifizierte parallele Polygonteilstücke zurückgegeben.

Eine zweite Routine ist in diese Funktion eingebettet. Diese überprüft auf die gleiche Art wie oben erwähnt alle Anschlußlinien, also alle Polygonlinien, die Anschlußstücke der gefundenen Parallelrelationen sind, auf Homogenität. Dabei wird entlang dieser Anschlußlinien gelaufen und ein Bereich untersucht, der genauso breit ist, wie die Breite des letzten homogenen Parallelenstücks, untersucht, also die Seite der Linie, auf der auch das in der Relation zugehörige Polygon liegt. Ist auch hier der Grauwertverlauf im Sinne von oben homogen, so wird der Bereich der nächsten Anschlußlinie untersucht. Die so gefundenen Endpunkte jedes Polygons aus der modifizierten Parallelenrelation werden in `ExtParallels` als erweiterte Parallelen zurückgegeben.

Parameter

- ▷ **`Parallels`** (input_object) ..... xld_para-array ⇝ *Hobject*
  Parallele Polygonteilstücke.
- ▷ **`Image`** (input_object) ..... image ⇝ *Hobject*
  Zu untersuchendes Eingabebild.
- ▷ **`ModParallels`** (output_object) ..... xld_mod_para-array ⇝ *Hobject*
  Modifizierte parallele Polygonteilstücke.
- ▷ **`ExtParallels`** (output_object) ..... xld_ext_para-array ⇝ *Hobject*
  Erweiterte parallele Polygonteilstücke.
- ▷ **`Quality`** (input_control) ..... number ⇝ *real* / integer
  Minimale Schwelle des Quality-Faktors (Maß für Paralelität).
  Defaultwert : 0.4
  Wertevorschläge : `Quality` ∈ {0.1, 0.2, 0.3, 0.4, 0.5, 0.6}
  Restriktion : (`0.0` ≤ `Quality`) ∧ (`Quality` ≤ `1.0`)
- ▷ **`MinGray`** (input_control) ..... integer ⇝ *integer*
  Minimale Schwelle des mittleren Grauwertes.
  Defaultwert : 160
  Wertevorschläge : `MinGray` ∈ {80, 100, 120, 140, 160, 180}
  Restriktion : (`0` ≤ `MinGray`) ∧ (`MinGray` ≤ `255`)

▷ **MaxGray** (input_control) ..... integer ⇝ *integer*
Maximale Schwelle des mittleren Grauwertes.
Defaultwert : 220
Wertevorschläge : MaxGray ∈ {140, 160, 180, 200, 220, 240}
Restriktion : ((0 ≤ MaxGray) ∧ (MaxGray ≤ 255)) ∧ (MaxGray ≥ MinGray)

▷ **MaxStandard** (input_control) ..... number ⇝ *real* / integer
Maximal zugelassene gemittelte Standardabweichung bzgl. der mittleren Grauwerte.
Defaultwert : 10.0
Wertevorschläge : MaxStandard ∈ {5.0, 10.0, 15.0, 20.0}
Restriktion : MaxStandard ≥ 0.0

Parallelisierungsinformation

mod_parallels_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_parallels_xld

Mögliche Nachfolgerfunktionen

max_parallels_xld

Siehe auch

info_parallels_xld

Modul

Sub-pixel operators

14.2 Merkmale

area_center_xld (XLD : : : Area, Row, Column, PointOrder)

Fläche und Schwerpunkt von Konturen oder Polygonen.

area_center_xld berechnet die Fläche und den Schwerpunkt der von den Eingabekonturen oder -polygonen XLD eingeschlossenen Fläche sowie den mathematischen Umlaufsinn der entsprechenden Stützpunkte. Die Fläche und der Schwerpunkt wird durch Anwendung des Gaußchen Integralsatzes nur unter Verwendung der Kontur- oder Polygonpunkte berechnet, d.h. es wird keine explizite Fläche zur Berechnung erzeugt. Sind die Punkte der Kontur entgegen des Uhrzeigersinnes angeordnet (d.h. im mathematisch positiven Sinn), ist PointOrder **positive**. Es wird vorausgesetzt, daß die Kontur geschlossen ist. Ist das nicht der Fall, wird sie künstlich geschlossen. Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Kontur bzw. des Polygons in der Eingabe entspricht.

Parameter

▷ **XLD** (input_object) ..... xld(-array) ⇝ *Hobject*
Zu untersuchende Konturen bzw. Polygone.

▷ **Area** (output_control) ..... real(-array) ⇝ *real*
Fläche, die von der Kontur eingeschlossen wird.

▷ **Row** (output_control) ..... point.y(-array) ⇝ *real*
Zeilenindex des Schwerpunktes.

▷ **Column** (output_control) ..... point.x(-array) ⇝ *real*
Spaltenindex des Schwerpunktes.

▷ **PointOrder** (output_control) ..... string(-array) ⇝ *string*
Mathematischer Umlaufsinn (positive”/negative”) der Stützpunkte.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

area_center_xld liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe läßt sich mittels set_system(::’no_object_result’,<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

area_center_xld ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Siehe auch

moments_xld, moments_any_xld, area_center, moments_region_2nd

Modul

Sub-pixel operators

contour_point_num_xld (Contour : : : Length)

Ausgabe der Anzahl von Stützpunkten einer XLD-Kontur.

Mit contour_point_num_xld wird in Length die Länge der Kontur Contour, d.h. die Anzahl der Konturpunkte, ausgegeben.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-XLD-Kontur.
- ▷ **Length** (output_control) ..... integer ⇝ *integer*
  Anzahl der Stützpunkte.

Parallelisierungsinformation

contour_point_num_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Mögliche Nachfolgerfunktionen

get_contour_xld, get_contour_attrib_xld

Alternativen

get_regress_params_xld

Siehe auch

query_contour_attribs_xld

Modul

Sub-pixel operators

dist_ellipse_contour_xld (Contours : : Mode, MaxNumPoints, ClippingEndPoints, Row, Column, Phi, Radius1, Radius2 : MinDist, MaxDist, AvgDist, SigmaDist)

Abstand von Konturen zu einer Ellipse.

dist_ellipse_contour_xld berechnet den Abstand der Konturen in Contours zu einer Ellipse, die durch ihren Mittelpunkt (Row, Column), die Orientierung ihrer Hauptachse Phi und die Länge der großen Radius1 bzw. kleinen Halbachse Radius2 beschrieben wird. Zur Bestimmung des Abstandes eines Konturpunktes $X = (x_i, y_i)$ zur Ellipse stehen dabei folgende Verfahren, selektierbar über Mode zur Verfügung:

'algebraic' Als Abstandsmaß wird die algebraische Distanz $ax_i^2+bx_iy_i+cy_i^2+dx_i+ey_i+f$ verwendet, wobei die Ellipsenparameter a - f so normiert werden, daß der Mittelpunkt der Ellipse als Abstandswert die Länge der kleineren Halbachse Radius2 erhält. Das Maß ist mit einem *high curvature bias* behaftet: In der Nähe von Ellipsenpunkten mit hoher Krümmung (insbesondere die Pole auf der Hauptachse) ist der Abstand kleiner als in der Nähe von Ellipsenpunkten mit geringer Krümmung (etwa die Pole auf der Nebenachse).

'geometric' Als Abstandsmaß wird die Abweichung $XF_1 + XF_2 - 2a$ verwendet, wobei F_1,F_2 die beiden Brennpunkte der Ellipse bezeichnen und a für die Länge der großen Halbachse Radius1 steht. Das Maß ist mit einem *low curvature bias* behaftet: In der Nähe von Ellipsenpunkten mit hoher Krümmung (insbesondere die Pole auf der Hauptachse) ist der Abstand größer als in der Nähe von Ellipsenpunkten mit geringer Krümmung (etwa die Pole auf der Nebenachse).

'bisec' Als Abstandsmaß wird die Entfernung des Konturpunkts vom Schnittpunkt der Winkelhalbierenden der beiden Geraden durch X und die beiden Brennpunkte mit der Ellipse verwendet. Dies ist eine sehr gute Näherung an den Orthogonalabstand des Konturpunktes von der Ellipse.

Zurückgegeben werden dann folgende statistische Werte über den Absolutbeträgen der einzelnen Abstände: Der minimale `MinDist` und maximale Abstand `MaxDist`, der mittlere Abstand `AvgDist` und die Standardabweichung des Abstandes `SigmaDist`.

Zur Reduktion des Aufwandes läßt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich -1 übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Abstandsberechnung auszuschließen.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
  Eingabekonturen.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Methode zur Abstandsbestimmung.
  Defaultwert : 'bisec'
  Werteliste : `Mode` ∈ {'geometric', 'algebraic', 'bisec'}
- ▷ **`MaxNumPoints`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl Konturpunkte für Berechnung (-1 für alle Punkte).
  Defaultwert : -1
  Restriktion : `MaxNumPoints` ≥ `3`
- ▷ **`ClippingEndPoints`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Abstandsberechnung ignoriert werden sollen.
  Defaultwert : 0
  Restriktion : `ClippingEndPoints` ≥ `0`
- ▷ **`Row`** (input_control) ..... ellipse.center.y ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **`Column`** (input_control) ..... ellipse.center.x ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **`Phi`** (input_control) ..... ellipse.angle.rad ⇝ *real*
  Orientierung der Hauptachse in Bogenmaß.
  Restriktion : (`Phi` ≥ `0`) ∧ (`Phi` ≤ `6.283185307`)
- ▷ **`Radius1`** (input_control) ..... ellipse.radius1 ⇝ *real*
  Länge der großen Halbachse.
  Restriktion : `Radius1` > `0`
- ▷ **`Radius2`** (input_control) ..... ellipse.radius2 ⇝ *real*
  Länge der kleinen Halbachse.
  Restriktion : `Radius2` ≥ `0`
- ▷ **`MinDist`** (output_control) ..... real(-array) ⇝ *real*
  Minimaler Abstand.
- ▷ **`MaxDist`** (output_control) ..... real(-array) ⇝ *real*
  Maximaler Abstand.
- ▷ **`AvgDist`** (output_control) ..... real(-array) ⇝ *real*
  Mittlerer Abstand.
- ▷ **`SigmaDist`** (output_control) ..... real(-array) ⇝ *real*
  Standardabweichung des Abstands.

Beispiel

```
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 |Row|-1 by 1
    Object_Selected := EdgesMarks[i+1]
    dist_ellipse_contour_xld (ObjectSelected, 'bisec', -1, 0, Row[i],
                              Column[i], Phi[i], Radius1[i], Radius2[i],
                              MinDist, MaxDist, AvgDist, SigmaDist)
endfor.
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `dist_ellipse_contour_xld` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`dist_ellipse_contour_xld` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`fit_ellipse_contour_xld`

Modul

Sub-pixel operators

`fit_circle_contour_xld` (Contours : : Algorithm, MaxNumPoints, MaxClosureDist, ClippingEndPoints, Iterations, ClippingFactor : Row, Column, Radius, StartPhi, EndPhi, PointOrder)

Approximation von bogenförmigen XLD-Konturen durch Kreise.

`fit_circle_contour_xld` approximiert die XLD-Konturen `Contours` durch Kreise. Eine Segmentation der Eingabekonturen erfolgt dabei nicht. Es muß also sichergestellt sein, daß jede Kontur in `Contours` genau einem Kreis entspricht. Zurückgegeben wird der Mittelpunkt (`Row`, `Column`) und der Radius `Radius`.

Das gewünschte Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

'algebraic' Minimiert wird der algebraische Abstand zwischen den und dem gesuchten Kreis.

'ahuber' Wie 'algebraic'. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

'atukey' Wie 'algebraic'. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Tukey gedämpft werden.

In den Modi 'ahuber' und 'atukey' wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von dem approximierenden Kreis zu ermitteln. Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wie auch die Kreisanpassung wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen.

Zur Reduktion des Aufwandes läßt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich -1 übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet.

Für Kreisbögen, also offene Konturen, werden die Anfangs- und Endpunkte bestimmt, indem zu den Anfangs- und Endpunkten der Eingabekonturen die nächstgelegenen Kreispunkte ermittelt werden. Zurückgegeben werden dann in `StartPhi` und `EndPhi` die korrespondierenden Winkel dieser Punkte bezogen auf die Horizontale, vgl. `gen_ellipse_contour_xld`. Konturen, deren Anfangs- und Endpunkte weniger als `MaxClosureDist` voneinander entfernt sind, werden als geschlossen betrachtet und entsprechend durch einen Kreis (anstelle eines Kreisbogens) approximiert. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Kreisanpassung auszuschließen. Sie werden jedoch weiterhin für die Bestimmung der Anfangs- und Endwinkel verwendet.

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *Hobject*
  Eingabekonturen.
- ▷ **Algorithm** (input_control) . . . . . string ⇝ *string*
  Algorithmus zur Kreisanpassung.
  Defaultwert : 'algebraic'
  Werteliste : `Algorithm` ∈ {'algebraic', 'ahuber', 'atukey'}
- ▷ **MaxNumPoints** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl Konturpunkte zur Kreisanpassung (-1 für alle Punkte).
  Defaultwert : -1
  Restriktion : `MaxNumPoints` ≥ `3`
- ▷ **MaxClosureDist** (input_control) . . . . . real ⇝ *real*
  Maximaler Abstand zweier Konturendpunkte, so daß die Kontur noch als geschlossene Kontur akzeptiert wird.
  Defaultwert : 0.0
  Restriktion : `MaxClosureDist` ≥ `0.0`
- ▷ **ClippingEndPoints** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Kreisanpassung ignoriert werden sollen.
  Defaultwert : 0
  Restriktion : `ClippingEndPoints` ≥ `0`
- ▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Iterationen.
  Defaultwert : 3
  Restriktion : `Iterations` ≥ `0`
- ▷ **ClippingFactor** (input_control) . . . . . real ⇝ *real*
  Clipping Faktor für die Ausreißerdämpfung (typisch: 1.0 bei Huber und 2.0 bei Tukey).
  Defaultwert : 2.0
  Werteliste : `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
  Restriktion : `ClippingFactor` > `0`
- ▷ **Row** (output_control) . . . . . circle.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes des Kreises.
- ▷ **Column** (output_control) . . . . . circle.center.x(-array) ⇝ *real*
  Spaltenkoordinate des Mittelpunktes des Kreises.
- ▷ **Radius** (output_control) . . . . . circle.radius(-array) ⇝ *real*
  Radius des Kreises.
- ▷ **StartPhi** (output_control) . . . . . real(-array) ⇝ *real*
  Winkel des Startpunktes [rad].
- ▷ **EndPhi** (output_control) . . . . . real(-array) ⇝ *real*
  Winkel des Endpunktes [rad].
- ▷ **PointOrder** (output_control) . . . . . string(-array) ⇝ *string*
  Mathematischer Umlaufsinn.
  Werteliste : `PointOrder` ∈ {'positive', 'negative'}

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Kreisanpassung durchgeführt werden, liefert `fit_circle_contour_xld` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fit_circle_contour_xld` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `smooth_contours_xld`

Mögliche Nachfolgerfunktionen

`gen_ellipse_contour_xld`, `disp_circle`, `get_points_ellipse`

Siehe auch

`fit_ellipse_contour_xld`, `fit_line_contour_xld`

Modul

Sub-pixel operators

fit_ellipse_contour_xld (Contours : : Algorithm, MaxNumPoints, MaxClosureDist, ClippingEndPoints, VossTabSize, Iterations, ClippingFactor : Row, Column, Phi, Radius1, Radius2, StartPhi, EndPhi, PointOrder)

Approximation von bogenförmigen XLD-Konturen durch Ellipsen.

fit_ellipse_contour_xld approximiert die XLD-Konturen Contours durch Ellipsenbögen bzw. geschlossene Ellipsen. Eine Segmentation der Eingabekonturen erfolgt dabei nicht. Es muß also sichergestellt sein, daß jede Kontur in Contours genau einem Ellipsenbogen entspricht. Zurückgegeben wird der Mittelpunkt (Row, Column), die Orientierung der Hauptachse Phi und die Länge der großen Radius1 bzw. der kleinen Halbachse Radius2 jeder Ellipse. Ellipsenbögen werden zusätzlich durch den Anfangs- und Endwinkel StartPhi und EndPhi des Start- bzw. Endpunktes, sowie den zugehörigen Umlaufsinn PointOrder charakterisiert. Bei geschlossenen Ellipsen werden die entsprechenden Ausgabeparameter mit 0, 2π und 'positive' besetzt.

Das gewünschte Approximationsverfahren wird über den Parameter Algorithm ausgewählt:

'fitzgibbon' Minimiert wird der algebraische Abstand $ax_i^2 + bx_iy_i + cy_i^2 + dx_i + ey_i + f$ zwischen den Konturpunkten (x_i, y_i) und der gesuchten Ellipse. Durch die Nebenbedingung $4ac - b^2 = 1$ wird dabei sichergestellt, daß eine Ellipse (und nicht etwa eine Hyperbel oder Parabel) angepaßt wird.

'fhuber' Wie 'fitzgibbon'. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

'ftukey' Wie 'fitzgibbon'. Es wird jedoch eine Gewichtung der Konturpunkte vorgenommen, bei der Ausreißer nach dem Ansatz von Tukey gedämpft werden.

'voss' Die Eingabekontur wird in eine affine Standardlage transformiert. Dann wird vermöge der Momente der transformierten Kontur (bzw. der davon eingeschlossenen Bildregion) dasjenige Einheitskreissegment bestimmt, dessen Standardlage am besten mit der der transformierten Kontur übereinstimmt. Aus der zugehörigen Ellipse in Standardlage und der affinen Abbildung, die die Kontur in Standardlage überführt hat, wird dann die entsprechende Ellipse ermittelt. Es werden VossTabSize Einheitskreissegment unterschieden (und deren Momente etc. beim ersten Aufruf des Operators tabelliert).

'focpoints' Für jeden Punkt P auf einer Ellipse gilt, daß die Summe seiner Abstände zu den Brennpunkten F_1,F_2 der Ellipse gleich dem doppelten Hauptradius a ist. Bei diesem Ansatz wird die Abweichung $PF_1 + PF_2 - 2a$ für alle Konturpunkte durch eine *least squares* Ausgleichsrechnung minimiert.

'fphuber' Wie 'focpoints'. Es wird jedoch eine *gewichtete* least squares Ausgleichsrechnung durchgeführt, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

'fptukey' Wie 'focpoints'. Es wird jedoch eine *gewichtete* least squares Ausgleichsrechnung durchgeführt, bei der Ausreißer nach dem Ansatz von Tukey gedämpft werden.

In den Modi '*Huber' und '*Tukey' wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von der approximierenden Ellipse zu ermitteln. Der Parameter ClippingFactor (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wie auch die Anpassung im Modus 'focpoints' wird iteriert. Der Parameter Iterations enthält die Anzahl durchzuführender Iterationen.

Zur Reduktion des Aufwandes läßt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für MaxNumPoints eine Zahl ungleich -1 übergeben, werden nur maximal MaxNumPoints gleichmäßig über die Kontur verteilte Punkte verwendet.

Für Ellipsenbögen, also offene Konturen, werden die Anfangs- und Endpunkte bestimmt, indem zu den Anfangs- und Endpunkten der Eingabekonturen die nächstgelegenen EllipsEndpunkte ermittelt werden. Zurückgegeben werden dann in StartPhi und EndPhi die korrespondierenden Winkel dieser Punkte bezogen auf die Hauptachse der Ellipse, vgl. gen_ellipse_contour_xld. Konturen, deren Anfangs- und Endpunkte weniger als MaxClosureDist voneinander entfernt sind, werden als geschlossen betrachtet und entsprechend durch eine Ellipse approximiert. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, ClippingEndPoints Punkte am Anfang und Ende der Kontur von der Ellipsenanpassung auszuschließen. Sie werden jedoch weiterhin für die Bestimmung der Anfangs- und Endwinkel verwendet.

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *Hobject*
  Eingabekonturen.
- ▷ **Algorithm** (input_control) . . . . . string ⇝ *string*
  Algorithmus zur Ellipsenanpassung.
  Defaultwert : 'fitzgibbon'
  Werteliste : `Algorithm` ∈ {'fitzgibbon', 'fhuber', 'ftukey', 'voss', 'focpoints', 'fphuber', 'fptukey'}
- ▷ **MaxNumPoints** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl Konturpunkte zur Ellipsenanpassung (-1 für alle Punkte).
  Defaultwert : -1
  Restriktion : `MaxNumPoints` ≥ `3`
- ▷ **MaxClosureDist** (input_control) . . . . . real ⇝ *real*
  Maximaler Abstand zweier Konturendpunkte, so daß die Kontur noch als geschlossene Kontur akzeptiert wird.
  Defaultwert : 0.0
  Restriktion : `MaxClosureDist` ≥ `0.0`
- ▷ **ClippingEndPoints** (input_control) . . . . . integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Ellipsenanpassung ignoriert werden sollen.
  Defaultwert : 0
  Restriktion : `ClippingEndPoints` ≥ `0`
- ▷ **VossTabSize** (input_control) . . . . . integer ⇝ *integer*
  Anzahl Einheitskreissegmente für das Verfahren nach Voss.
  Defaultwert : 200
  Restriktion : (`VossTabSize` ≥ `25`) ∧ (`VossTabSize` ≤ `5000`)
- ▷ **Iterations** (input_control) . . . . . integer ⇝ *integer*
  Maximale Anzahl von Iterationen (unbenutzt bei 'fitzgibbon' und 'voss').
  Defaultwert : 3
  Restriktion : `Iterations` ≥ `0`
- ▷ **ClippingFactor** (input_control) . . . . . real ⇝ *real*
  Clipping Faktor für die Ausreißerdämpfung (typisch: 1.0 bei '*Huber' und 2.0 bei '*Tukey').
  Defaultwert : 2.0
  Werteliste : `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
  Restriktion : `ClippingFactor` > `0`
- ▷ **Row** (output_control) . . . . . ellipse.center.y(-array) ⇝ *real*
  Zeilenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Column** (output_control) . . . . . ellipse.center.x(-array) ⇝ *real*
  Spaltenkoordinate des Mittelpunktes der Ellipse.
- ▷ **Phi** (output_control) . . . . . ellipse.angle.rad(-array) ⇝ *real*
  Orientierung der Hauptachse in Bogenmaß.
- ▷ **Radius1** (output_control) . . . . . ellipse.radius1(-array) ⇝ *real*
  Länge der großen Halbachse.
- ▷ **Radius2** (output_control) . . . . . ellipse.radius2(-array) ⇝ *real*
  Länge der kleinen Halbachse.
- ▷ **StartPhi** (output_control) . . . . . real(-array) ⇝ *real*
  Winkel des Startpunktes [rad].
- ▷ **EndPhi** (output_control) . . . . . real(-array) ⇝ *real*
  Winkel des Endpunktes [rad].
- ▷ **PointOrder** (output_control) . . . . . string(-array) ⇝ *string*
  mathematischer Umlaufsinn.
  Werteliste : `PointOrder` ∈ {'positive', 'negative'}

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Ellipsenanpassung durchgeführt werden, liefert `fit_ellipse_contour_xld` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fit_ellipse_contour_xld` ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `smooth_contours_xld`

Mögliche Nachfolgerfunktionen

`gen_ellipse_contour_xld`, `disp_ellipse`, `get_points_ellipse`

Siehe auch

`fit_line_contour_xld`

Modul

Sub-pixel operators

fit_line_contour_xld

```
fit_line_contour_xld ( Contours : : Algorithm, MaxNumPoints,
ClippingEndPoints, Iterations, ClippingFactor : RowBegin, ColBegin,
RowEnd, ColEnd, Nr, Nc, Dist )
```

Approximation von XLD-Konturen durch Liniensegmente.

`fit_line_contour_xld` approximiert die XLD-Konturen `Contours` durch Liniensegmente. Eine Segmentation der Eingabekonturen erfolgt dabei nicht. Es muß also sichergestellt sein, daß jede Kontur in `Contours` genau einem Liniensegment entspricht. Zurückgegeben wird der Anfangs- (`RowBegin`, `ColBegin`) bzw. Endpunkt (`RowEnd`, `ColEnd`), sowie die Geradengleichung jedes Liniensegments, codiert durch den Normalenvektor (`Nr`, `Nc`) der Gerade und dem Abstand `Dist` der Gerade vom Ursprung, d.h. die Geradengleichung ist gegeben durch $rn\_r + cn\_c - d = 0$.

Das gewünschte Approximationsverfahren wird über den Parameter `Algorithm` ausgewählt:

'regression' Standard 'least squares' Geradenanpassung.

'huber' Gewichtete *least squares* Geradenanpassung, bei der Ausreißer nach dem Ansatz von Huber gedämpft werden.

'tukey' Gewichtete *least squares* Geradenanpassung, bei der Ausreißer nach dem Ansatz von Tukey gedämpft werden.

'drop' Gewichtete *least squares* Geradenanpassung, bei der Ausreißer vollständig eliminiert werden.

'gauss' Gewichtete *least squares* Geradenanpassung, bei der Ausreißer gestützt auf Mittelwert und Standardabweichung der Abweichungen aller Konturpunkte von der Geraden gedämpft werden.

In den Modi 'huber', 'tukey' und 'drop' wird eine robuste Fehlerstatistik verwendet, um die Standardabweichung der Abstände der Konturpunkte (ohne Ausreißer) von der approximierenden Geraden zu ermitteln. Der Parameter `ClippingFactor` (ein Skalierungsfaktor für diese Standardabweichung) steuert in diesen Modi den Grad der Ausreißerdämpfung: Je kleiner der Wert gewählt wird, desto stärker ist die Dämpfung. Die Ausreißerdetektion wird iteriert. Der Parameter `Iterations` enthält die Anzahl durchzuführender Iterationen. Er wird im Modus 'regression' ignoriert.

Zur Reduktion des Aufwandes läßt sich die Berechnung auf eine Teilmenge der Konturpunkte einschränken: Wird für `MaxNumPoints` eine Zahl ungleich -1 übergeben, werden nur maximal `MaxNumPoints` gleichmäßig über die Kontur verteilte Punkte verwendet.

Die Anfangs- und Endpunkte werden bestimmt, indem zu den Anfangs- und Endpunkten der Eingabekonturen die nächstgelegenen Punkte auf den zugrundeliegenden Regressionsgeraden ermittel werden. Da die Start- und Endpunkte einer Kontur je nach Vorverarbeitung nicht exakt bekannt sein können, besteht die Möglichkeit, `ClippingEndPoints` Punkte am Anfang und Ende der Kontur von der Geradenanpassung auszuschließen. Sie werden jedoch weiterhin für die Bestimmung der Anfangs- und Endpunkte verwendet.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
  Eingabekonturen.
- ▷ **`Algorithm`** (input_control) ..... string ⇝ *string*
  Algorithmus zur Geradenanpassung.
  Defaultwert : 'tukey'
  Werteliste : `Algorithm` ∈ {'regression', 'huber', 'tukey', 'gauss', 'drop'}
- ▷ **`MaxNumPoints`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl Konturpunkte zur Geradenanpassung (-1 für alle Punkte).
  Defaultwert : -1
  Restriktion : `MaxNumPoints` ≥ `3`
- ▷ **`ClippingEndPoints`** (input_control) ..... integer ⇝ *integer*
  Anzahl der Konturpunkte am Anfang und Ende der Kontur, die für die Geradenanpassung ignoriert werden sollen.
  Defaultwert : 0
  Restriktion : `ClippingEndPoints` ≥ `0`
- ▷ **`Iterations`** (input_control) ..... integer ⇝ *integer*
  Maximale Anzahl von Iterationen (unbenutzt bei 'regression').
  Defaultwert : 5
  Restriktion : `Iterations` ≥ `0`
- ▷ **`ClippingFactor`** (input_control) ..... real ⇝ *real*
  Clipping Faktor für die Ausreißerdämpfung (typisch: 1.0 bei 'huber' und 'drop' sowie 2.0 bei 'tukey').
  Defaultwert : 2.0
  Werteliste : `ClippingFactor` ∈ {1.0, 1.5, 2.0, 2.5, 3.0}
  Restriktion : `ClippingFactor` > `0`
- ▷ **`RowBegin`** (output_control) ..... line.begin.y(-array) ⇝ *real*
  Zeilenkoordinate der Anfangspunkte der Liniensegmente.
- ▷ **`ColBegin`** (output_control) ..... line.begin.x(-array) ⇝ *real*
  Spaltenkoordinate der Anfangspunkte der Liniensegmente.
- ▷ **`RowEnd`** (output_control) ..... line.end.y(-array) ⇝ *real*
  Zeilenkoordinate der Endpunkte der Liniensegmente.
- ▷ **`ColEnd`** (output_control) ..... line.end.x(-array) ⇝ *real*
  Spaltenkoordinate der Endpunkte der Liniensegmente.
- ▷ **`Nr`** (output_control) ..... number(-array) ⇝ *real*
  Geradenparameter: Zeilenkoordinate von Normalenvektor
- ▷ **`Nc`** (output_control) ..... number(-array) ⇝ *real*
  Geradenparameter: Spaltenkoordinate von Normalenvektor
- ▷ **`Dist`** (output_control) ..... number(-array) ⇝ *real*
  Geradenparameter: Abstand der Geraden vom Ursprung

Beispiel

```
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).
```

Ergebnis

Sind die Parameterwerte korrekt und konnte eine Geradenanpassung durchgeführt werden, liefert `fit_line_contour_xld` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`fit_line_contour_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `smooth_contours_xld`

Mögliche Nachfolgerfunktionen

`disp_line`, `select_lines`, `line_orientation`

Siehe auch

`regress_contours_xld`, `get_regress_params_xld`

Modul

Sub-pixel operators

`get_contour_angle_xld` (Contour : : AngleMode, CalcMode, Lookaround : Angles)

Punktweise Berechnung der Tangentenrichtung einer Kontur.

Die Prozedur `get_contour_angle_xld` berechnet für jeden Punkt der Kontur `Contour` die Richtung der Tangente. Dabei kann zwischen zwei Winkelausgaben `AngleMode` gewählt werden: Mit **’abs’** werden Winkel bezüglich der Horizontalen zwischen 0 und 2π (entgegen dem Uhrzeigersinn) ausgegeben, mit **’rel’** wird der Betrag der Richtungsänderung gegenüber dem vorhergehenden Konturpunkt ausgegeben. Die Werte liegen in diesem Fall zwischen 0 und π.

Drei mögliche Berechnugsarten `CalcMode` berechnen die Tangentenrichtung im Konturpunkt i mit Hilfe der Konturpunkte im Bereich $i-$`Lookaround` bis $i+$`Lookaround`. Mit **’range’** wird der Winkel der Geraden durch die beiden Randpunkte dieses Bereichs ermittelt. Bei der Berechnungsart **’mean’** werden hingegen die Winkel aller Einzelschritte zwischen den Konturpunkten des Bereichs arithmetisch gemittelt. Mit **’regress’** wird die Steigung der Regressionsgeraden zwischen allen (gleichberechtigten) Punkten dieses Bereichs ermittelt. `Lookaround` ist ein Maß für die Glättung der Kontur. Die Winkel werden im Bogenmaß in dem Tupel `Angles` ausgegeben.

Parameter

- ▷ **`Contour`** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-Kontur.
- ▷ **`AngleMode`** (input_control) ..... string ⇝ *string*
  Ausgabeart der Winkel.
  Defaultwert : ’abs’
  Werteliste : `AngleMode` ∈ {’abs’, ’rel’}
- ▷ **`CalcMode`** (input_control) ..... string ⇝ *string*
  Berechnugsart der Winkel.
  Defaultwert : ’range’
  Werteliste : `CalcMode` ∈ {’range’, ’mean’, ’regress’}
- ▷ **`Lookaround`** (input_control) ..... integer ⇝ *integer*
  Größe der zu betrachtenden Nachbarschaft.
  Defaultwert : 3
  Restriktion : `Lookaround` > `0`
- ▷ **`Angles`** (output_control) ..... real-array ⇝ *real*
  Richtung der Konturpunkte.

Parallelisierungsinformation

`get_contour_angle_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Siehe auch

get_contour_xld, get_contour_attrib_xld

Modul

Sub-pixel operators

get_contour_attrib_xld (Contour : : Name : Attrib)

Auslesen von Punkt-Attributen einer XLD-Kontur.

get_contour_attrib_xld liefert für die XLD-Kontur Contour den Wert des für jeden Punkt der Kontur definierten Attributs mit Namen Name im Parameter Attrib zurück. Attribute sind für jeden Konturpunkt definierte, zusätzliche Werte, wie z.B. die Richtung senkrecht zur Kontur (**'angle'**). Operatoren, die solche Attribute definieren, enthalten eine Beschreibung der Namen und der Semantik der von ihnen definierten Werte. Welche Attribute für eine Kontur definiert sind, kann mit query_contour_attribs_xld abgefragt werden.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-Kontur.
- ▷ **Name** (input_control) ..... string ⇝ *string*
  Name des Attributs.
  Defaultwert : 'angle'
  Wertevorschläge : Name ∈ {'angle', 'edge_direction', 'width_right', 'width_left', 'response', 'contrast', 'asymmetry'}
- ▷ **Attrib** (output_control) ..... real-array ⇝ *real*
  Zurückgeliefertes Attribut.

Parallelisierungsinformation

get_contour_attrib_xld ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

lines_gauss, lines_facet, edges_sub_pix

Siehe auch

query_contour_attribs_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Modul

Sub-pixel operators

get_contour_global_attrib_xld (Contour : : Name : Attrib)

Auslesen von globalen Attributen einer XLD-Kontur.

get_contour_global_attrib_xld liefert für die XLD-Kontur Contour den Wert des global definierten Attributs mit Namen Name im Parameter Attrib zurück. Globale Attribute sind für eine gesamte Kontur definierte, zusätzliche Werte, wie z.B. der Normalenvektor der Regressionsgerade einer Kontur (**'regr_norm_row'** und **'regr_norm_col'**). Operatoren, die solche Attribute definieren, enthalten eine Beschreibung der Namen und der Semantik der von ihnen definierten Werte. Welche Attribute für eine Kontur definiert sind, kann mit query_contour_global_attribs_xld abgefragt werden.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-Kontur.
- ▷ **Name** (input_control) ..... string(-array) ⇝ *string*
  Name des Attributs.
  Defaultwert : 'regr_norm_row'
  Wertevorschläge : Name ∈ {'regr_norm_row', 'regr_norm_col', 'regr_mean_dist', 'regr_dev_dist', 'cont_approx'}

▷ **Attrib** (output_control) . . . . . real-array ⇝ *real*
Zurückgeliefertes Attribut.

Parallelisierungsinformation

`get_contour_global_attrib_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`lines_gauss`, `lines_facet`, `edges_sub_pix`

Siehe auch

`query_contour_global_attribs_xld`, `get_contour_attrib_xld`, `query_contour_attribs_xld`

Modul

Sub-pixel operators

get_regress_params_xld (Contours : : : Length, Nx, Ny, Dist, Fpx, Fpy, Lpx, Lpy, Mean, Deviation)

Ausgabe von Konturparametern.

Diese Routine gibt folgende Konturparameter für alle Konturen des Tupels `Contours` als Tupel aus:

- die Anzahl der Konturpunkte `Length`,
- die Koordinaten `Nx` und `Ny` des Normalenvektors der Regressionsgeraden
- den Abstand `Dist` der Regressionsgeraden vom Ursprung
- die subpixelgenauen Koordinaten `Fpx` und `Fpy` des Fußpunktes des Lotes das ersten Endpunktes auf die optimale Regressionsgerade,
- die subpixelgenauen Koordinaten `Lpx` und `Lpy` des Fußpunktes des Lotes das zweiten Endpunktes auf die optimale Regressionsgerade,
- den Mittelwert des absoluten euklidischen Abstandes der Konturpunkte zur optimalen Regressionsgeraden,
- die Standardabweichung dieser Abstände.

Achtung

Bevor Konturinformationen mit `get_regress_params_xld` ausgegeben werden können, müssen die Regressionsgeradenparameter mit `regress_contours_xld` berechnet werden.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *Hobject*
Eingabe-Konturen.

▷ **Length** (output_control) . . . . . integer-array ⇝ *integer*
Länge der Kontur.

▷ **Nx** (output_control) . . . . . point.x-array ⇝ *real*
X-Koordinate des Normalenvektors.

▷ **Ny** (output_control) . . . . . point.y-array ⇝ *real*
Y-Koordinate des Normalenvektors.

▷ **Dist** (output_control) . . . . . number-array ⇝ *real*
Abstand der Regressionsgerade vom Ursprung.

▷ **Fpx** (output_control) . . . . . point.x-array ⇝ *real*
X-Koordinate des ersten Endpunkts.

▷ **Fpy** (output_control) . . . . . point.y-array ⇝ *real*
Y-Koordinate des ersten Endpunkts.

▷ **Lpx** (output_control) . . . . . point.x-array ⇝ *real*
X-Koordinate des zweiten Endpunkts.

▷ **Lpy** (output_control) . . . . . point.y-array ⇝ *real*
Y-Koordinate des zweiten Endpunkts.

▷ **Mean** (output_control) . . . . . real-array ⇝ *real*
Mittlerer Abstand der Konturpunkte von der Regressionsgerade.

▷ **Deviation** (output_control) . . . . . real-array ⇝ *real*
Standardabweichung des Abstands der Punkte von der Geraden.

Parallelisierungsinformation

`get_regress_params_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`regress_contours_xld`

Mögliche Nachfolgerfunktionen

`disp_line`, `select_lines`, `line_orientation`

Siehe auch

`fit_line_contour_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`, `get_contour_xld`, `get_contour_attrib_xld`, `gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Modul

Sub-pixel operators

info_parallels_xld (Parallels, Image : : : QualityMin, QualityMax, GrayMin, GrayMax, StandardMin, StandardMax)

Informationen zur Fläche, die je zwei parallele Linien einschließen.

`info_parallels_xld` charakterisiert die von je zwei parallelen Linien, die unter `Parallels` abgespeichert sind, eingeschlossene Fläche. Dabei wird das Bild `Image` herangezogen, um die Grauwertbeschaffenheit dieser Fläche zu untersuchen. Der Algorithmus dieser Routine entspricht im wesentlichen dem von `mod_parallels_xld`. Es werden die Werte Qualitäts-Faktor (`QualityMin` und `QualityMax`), mittlerer Grauwert (`GrayMin` und `GrayMax`) und mittlere Standardabweichung (`StandardMin` und `StandardMax`) bzgl. des zugehörigen mittleren Grauwerts zurückgeliefert.

Dieser Operator dient zur Bestimmung von geeigneten Schwellenwerten für `mod_parallels_xld`.

Parameter

▷ **Parallels** (input_object) . . . . . xld_para-array ⇝ *Hobject*
Parallele Polygonteilstücke.

▷ **Image** (input_object) . . . . . image ⇝ *Hobject*
Zu untersuchendes Eingabebild.

▷ **QualityMin** (output_control) . . . . . real ⇝ *real*
Minimaler Qualitäts-Faktor.

▷ **QualityMax** (output_control) . . . . . real ⇝ *real*
Maximaler Qualitäts-Faktor.

▷ **GrayMin** (output_control) . . . . . integer ⇝ *integer*
Minimaler mittlerer Grauwert.

▷ **GrayMax** (output_control) . . . . . integer ⇝ *integer*
Maximaler mittlerer Grauwert.

▷ **StandardMin** (output_control) . . . . . real ⇝ *real*
Minimale Standardabweichung.

▷ **StandardMax** (output_control) . . . . . real ⇝ *real*
Maximale Standardabweichung.

Parallelisierungsinformation

`info_parallels_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_parallels_xld`

Mögliche Nachfolgerfunktionen

`mod_parallels_xld`

Siehe auch

`intensity`, `min_max_gray`

Modul

Sub-pixel operators

length_xld (XLD : : : Length)

Länge von Konturen oder Polygonen.

length_xld berechnet die Gesamtlänge der Kontour oder des Polygons, das in XLD übergeben wird. Die Länge ergibt sich als Summe der Euklidschen Abstände der einzelnen Kurvenpunkte. Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Kontur bzw. des Polygons in der Eingabe entspricht.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *Hobject*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Length** (output_control) ..... real(-array) ⇝ *real*
  Länge der Kontur bzw. des Polygons.
  Zusicherung : Length ≥ 0

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

length_xld liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

length_xld ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Siehe auch

area_center_xld, moments_any_xld, moments_xld, contlength

Modul

Sub-pixel operators

local_max_contours_xld (Contours,
Image : LocalMaxContours : MinPercent, MinDiff, Distance :)

Selektion von Konturen, deren Grauwert ein lokales Maximum ist.

Die Funktion local_max_contours_xld wählt aus den Konturen, die in Contours übergeben werden, diejenigen Konturen aus, deren Grauwerte ein lokales Maximum im Eingabebild Image darstellen. Damit eine Kontur als lokales Maximum gilt, müssen mindestens MinPercent der Konturpunkte ein lokales Maximum senkrecht zur Kontur-Richtung sein. Die Kontur-Richtung wird berechnet, indem eine Regressionsgerade durch fünf umgebende Punkte gelegt wird. Zur Entscheidung, ob ein lokales Maximum vorliegt, wird ein Grauwertprofil in beide Normalenrichtungen bis zum Abstand Distance vom aktuellen Konturpunkt betrachtet. Der Grauwert an der Konturposition in beiden Richtungen um mindestens MinDiff größer sein, als der Grauwert an einer der Positionen des Profils, um als Maximum zu gelten. Die selektierten Konturen werden in LocalMaxContours zurückgegeben.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *Hobject*
  Zu untersuchende Konturen.
- ▷ **Image** (input_object) ..... image ⇝ *Hobject*
  Zu untersuchendes Eingabebild.
- ▷ **LocalMaxContours** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Selektierte Konturen.
- ▷ **MinPercent** (input_control) ..... number ⇝ *integer* / real
  Mindestanteil von Maximumspunkten.
  Defaultwert : 70
  Wertevorschläge : MinPercent ∈ {60, 70, 75, 80, 85, 90, 95}
  Restriktion : (0.0 ≤ MinPercent) ∧ (MinPercent ≤ 100.0)

▷ **MinDiff** (input_control) ..... integer ⇝ *integer*
Minimale Differenz, um die das Maximum größer sein muß als die Werte am Rand.
Defaultwert : 15
Wertevorschläge : MinDiff ∈ {5, 8, 10, 12, 15, 20}
Restriktion : (0 ≤ MinDiff) ∧ (MinDiff ≤ 255)

▷ **Distance** (input_control) ..... integer ⇝ *integer*
Maximalabstand der Punkte, die zur Bestimmung, ob ein Punkt ein lokales Maximum ist, verwendet werden, zum Punkt auf der Kontur.
Defaultwert : 4
Wertevorschläge : Distance ∈ {2, 3, 4, 5, 6}
Restriktion : Distance ≥ 1

Parallelisierungsinformation

local_max_contours_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Mögliche Nachfolgerfunktionen

gen_polygons_xld

Siehe auch

smooth_contours_xld

Modul

Sub-pixel operators

max_parallels_xld (ExtParallels : MaxPolygons : :)

Parallele Polygone mit maximaler Ausdehnung.

Mit max_parallels_xld werden alle modifizierten parallelen Polygonteilstücke, die unter ExtParallels abgespeichert sind, zu einem Polygonstück zusammengefaßt. Die gefundenen Polygone werden mit ihrem kleinsten Anfangsindex und größtem Endindex abgespeichert. Es werden somit Polygone, die an mehreren Stellen Parallelität aufweisen, als ein langes Polygonstück (von der ersten Linie bis zur letzten parallelen Linie) in MaxPolygons ausgegeben.

Parameter

▷ **ExtParallels** (input_object) ..... xld_ext_para-array ⇝ *Hobject*
Erweiterte Parallelen.

▷ **MaxPolygons** (output_object) ..... xld_poly-array ⇝ *Hobject*
Maximal erweiterte Polygonstücke.

Parallelisierungsinformation

max_parallels_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

mod_parallels_xld

Mögliche Nachfolgerfunktionen

get_polygon_xld, get_lines_xld

Modul

Sub-pixel operators

moments_any_xld (XLD : : Mode, PointOrder, Area, CenterRow, CenterCol, P, Q : M)

Geometrische Momente beliebiger Ordnung von Konturen oder Polygonen.

moments_any_xld berechnet Momente beliebiger Ordnung der von Konturen oder Polygonen XLD umschlossenen Bildregionen. Die Momente werden durch Anwendung des Gaußchen Integralsatzes nur unter Verwendung der Kontur- oder Polygonpunkte berechnet, d.h. es wird keine explizite Fläche zur Berechnung erzeugt. Es wird

vorausgesetzt, daß die Kontur geschlossen ist. Ist das nicht der Fall, wird sie künstlich geschlossen. Abhängig vom Parameter `Mode` werden die berechneten Momente wie folgt nomiert:

'unnormalized': Keine Normalisierung. Sei R die eingeschlossene Bildregion. So entspricht das berechnete Moment $M_{p,q}$

$$M_{p,q} = \int_R r^p c^q \, dr \, dc$$

'unnormalized_central': Verschiebung der Bildregion um ihren Schwerpunkt $[\bar{r}, \bar{c}]$ = [`CenterRow`, `CenterCol`]:

$$M_{p,q} = \int_R (r - \bar{r})^p (c - \bar{c})^q \, dr \, dc$$

'normalized': Normalisierung durch die Fläche $A =$ `Area` der eingeschlossenen Bildregion:

$$M_{p,q} = \frac{1}{A} \int_R r^p c^q \, dr \, dc$$

'central': Normalisierung durch die Fläche $A =$ `Area` der eingeschlossenen Bildregion und zusätzlich Verschiebung der Bildregion um ihren Schwerpunkt $[\bar{r}, \bar{c}] =$ [`CenterRow`, `CenterCol`]:

$$M_{p,q} = \frac{1}{A} \int_R (r - \bar{r})^p (c - \bar{c})^q \, dr \, dc$$

Für die Normierung der Momente werden ggf. drei spezielle Momente benötigt: Die Fläche `Area` der eingeschlossenen Bildregion sowie die Koordinaten `CenterRow`,`CenterRow` des Schwerpunktes, vgl. `area_center_xld`. Außerdem erwartet `moments_any_xld` den Umlaufsinn der Eingabekonturen bzw. -polygone in `PointOrder`, vgl. neuerlich `area_center_xld`. Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Momente für die einzelnen Konturen bzw. Polygone hintereinander in `M` abgelegt.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *Hobject*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Modus.
  Defaultwert : 'unnormalized'
  Wertevorschläge : `Mode` ∈ {'unnormalized', 'unnormalized_central', 'normalized', 'central'}
- ▷ **PointOrder** (input_control) ..... string(-array) ⇝ *string*
  Mathematischer Umlaufsinn der Stützpunkte.
  Defaultwert : 'positive'
  Wertevorschläge : `PointOrder` ∈ {'positive', 'negative'}
- ▷ **Area** (input_control) ..... real(-array) ⇝ *real*
  Fläche, die von der Kontur eingeschlossen wird.
- ▷ **CenterRow** (input_control) ..... point.y(-array) ⇝ *real*
  Zeilenindex des Schwerpunktes.
- ▷ **CenterCol** (input_control) ..... point.x(-array) ⇝ *real*
  Spaltenindex des Schwerpunktes.
- ▷ **P** (input_control) ..... point.x(-array) ⇝ *integer*
  Erster Index der zu berechnenden Momente M[P,Q].
  Defaultwert : 1
- ▷ **Q** (input_control) ..... point.x(-array) ⇝ *integer*
  Zweiter Index der zu berechnenden Momente M[P,Q].
  Defaultwert : 1
- ▷ **M** (output_control) ..... real(-array) ⇝ *real*
  Die berechneten Momente.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

moments_any_xld liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

moments_any_xld ist *lokal* auszuführen („local“) und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

area_center_xld, gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

moments_xld

Siehe auch

moments_xld, area_center_xld, moments_region_2nd, area_center

Modul

Sub-pixel operators

moments_xld (XLD : : : M11, M20, M02)

Geometrische Momente M20, M02 und M11 von Konturen oder Polygonen.

moments_xld berechnet die Momente (M20, M02) und das Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen (M11). Zur Definition dieser Größen siehe moments_region_2nd. Die Momente werden durch Anwendung des Gaußchen Integralsatzes nur unter Verwendung der Kontur- oder Polygonpunkte berechnet, d.h. es wird keine explizite Fläche zur Berechnung erzeugt. Es wird vorausgesetzt, daß die Kontur geschlossen ist. Ist das nicht der Fall, wird sie künstlich geschlossen. Wird mehr als eine Kontur oder ein Polygon übergeben, dann werden die Ergebnisse in Tupeln abgespeichert, wobei der Index eines Wertes in dem Tupel dem Index der Kontur bzw. des Polygons in der Eingabe entspricht.

Parameter

- ▷ **XLD** (input_object) ..... xld(-array) ⇝ *Hobject*
  Zu untersuchende Konturen bzw. Polygone.
- ▷ **M11** (output_control) ..... real(-array) ⇝ *real*
  Trägheitsprodukt der Achsen durch den Schwerpunkt parallel zu den Koordinatenachsen.
- ▷ **M20** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung in Zeilenrichtung.
- ▷ **M02** (output_control) ..... real(-array) ⇝ *real*
  Moment 2. Ordnung in Spaltenrichtung.

Komplexität

Sei n die Anzahl der Punkte in der Kontur bzw. dem Polygon. Dann ist die Laufzeit $O(n)$.

Ergebnis

moments_xld liefert den Wert 2 (H_MSG_TRUE), falls die Eingabe nicht leer ist. Das Verhalten bei leerer Eingabe läßt sich mittels set_system(::'no_object_result',<Result>:) festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

moments_xld ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, smooth_contours_xld, gen_polygons_xld

Alternativen

moments_any_xld

Siehe auch

moments_any_xld, area_center_xld, moments_region_2nd, area_center

Modul

Sub-pixel operators

query_contour_attribs_xld (Contour : : : Attribs)

Auslesen der Namen von definierten Attributen einer XLD-Kontur.

query_contour_attribs_xld liefert für die XLD-Kontur Contour die Namen aller für jeden Punkt der Kontur definierten Attribute in Attribs zurück. Attribute sind für jeden Konturpunkt definierte, zusätzliche Werte, wie z.B. die Richtung senkrecht zur Kontur (**'angle'**). Operatoren, die solche Attribute definieren, enthalten eine Beschreibung der Namen und der Semantik der von ihnen definierten Werte. Die Werte des Attributs können mit get_contour_attrib_xld ausgelesen werden.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-Kontur.
- ▷ **Attribs** (output_control) ..... string-array ⇝ *string*
  Liste der definierten Kontur-Attribute.

Parallelisierungsinformation

query_contour_attribs_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

lines_gauss, lines_facet, edges_sub_pix

Siehe auch

get_contour_attrib_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Modul

Sub-pixel operators

query_contour_global_attribs_xld (Contour : : : Attribs)

Auslesen der Namen von definierten globalen Attributen einer XLD-Kontur.

query_contour_global_attribs_xld liefert für die XLD-Kontur Contour die Namen aller global definierten Attribute in Attribs zurück. Globale Attribute sind für eine gesamte Kontur definierte, zusätzliche Werte, wie z.B. der Normalenvektor der Regressionsgerade einer Kontur (**'regr_norm_row'** und **'regr_norm_col'**). Operatoren, die solche Attribute definieren, enthalten eine Beschreibung der Namen und der Semantik der von ihnen definierten Werte. Der Wert des Attributs kann mit get_contour_global_attrib_xld ausgelesen werden.

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-Kontur.
- ▷ **Attribs** (output_control) ..... string-array ⇝ *string*
  Liste der definierten globalen Kontur-Attribute.

Parallelisierungsinformation

query_contour_global_attribs_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

lines_gauss, lines_facet, edges_sub_pix

Siehe auch

get_contour_global_attrib_xld, get_contour_attrib_xld, query_contour_attribs_xld

Modul

Sub-pixel operators

select_contours_xld (Contours : SelectedContours : Feature, Min1, Max1, Min2, Max2 :)

Filter für XLD-Konturen.

select_contours_xld filtert die Konturen Contours. Dabei kann zwischen folgenden Merkmalen Feature ausgewählt werden:

'contour_length' Alle Konturen, deren Länge kleiner als Min1 oder größer als Max1 ist, werden gelöscht, (Min2 und Max2 sind hier ohne Bedeutung).

'maximum_extent' Alle Konturen, deren maximale Ausdehnung (gemessen zwischen den acht Extrempunkten in Zeilen- und Spaltenrichtung, wie sie durch Haralick und Shapiro Computer and Robot Vision, Addison-Wesley 1992, Kapitel 3.2 beschrieben werden) kleiner als Min1 oder größer als Max1 ist, werden gelöscht, (Min2 und Max2 sind hier ohne Bedeutung).

'direction' Nur alle Konturen, deren Richtungen der optimalen Regressionsgeraden zwischen Min1 und Max1 (im Bogenmaß, mathematischer Drehsinn) liegen, werden behalten. Die Parameter Min1 und Max1 werden auf den Bereich [0,2*PI[abgebildet. (Min2 und Max2 sind hier ohne Bedeutung).

'curvature' Nur alle Konturen, deren mittlerer Abstand der Konturpunkte von der optimalen Regressionsgeraden zwischen Min1 und Max1 liegt und der Standardabweichung dieses Abstands zwischen Min2 und Max2 liegt, werden behalten.

'closed' Nur Konturen, deren Anfangs- und Endpunkte einen Abstand von maximal Max1 Pixeln voneinander haben, werden behalten.

'open' Nur Konturen, deren Anfangs- und Endpunkte einen Abstand von mindestens Min1 Pixeln voneinander haben, werden behalten.

Wenn beim Filtern nach der Krümmung Min1 = Max1 = 0 oder Min2 = Max2 = 0 gewählt wird, spielt das zugeordnete Merkmal keine einschränkende Rolle für die Selektion.

Achtung

Bevor Konturen mit select_contours_xld gemäß Merkmal 'direction' oder 'curvature' gefiltert werden können, müssen die Regressionsgeradenparameter mit regress_contours_xld berechnet werden.

Parameter

▷ **Contours** (input_object) ..... xld_cont-array ⇝ *Hobject*
Eingabe-Konturen.

▷ **SelectedContours** (output_object) ..... xld_cont-array ⇝ *Hobject*
Ausgabe-Konturen.

▷ **Feature** (input_control) ..... string ⇝ *string*
Selektionsmerkmal.
Defaultwert : 'contour_length'
Werteliste : Feature ∈ {'contour_length', 'maximum_extent', 'direction', 'curvature', 'closed', 'open'}

▷ **Min1** (input_control) ..... real ⇝ *real*
Untere Schranke.
Defaultwert : 0.5

▷ **Max1** (input_control) ..... real ⇝ *real*
Obere Schranke.
Defaultwert : 200.0

▷ **Min2** (input_control) ..... real ⇝ *real*
Untere Schranke.
Defaultwert : -0.5

▷ **Max2** (input_control) ..... real ⇝ *real*
Obere Schranke.
Defaultwert : 0.5

Parallelisierungsinformation

select_contours_xld ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

regress_contours_xld

Siehe auch

get_contour_xld, get_contour_attrib_xld, gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, get_regress_params_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Literatur

R. Haralick, L. Shapiro: „Computer and Robot Vision" Vol. 1; Kapitel 3.2, Addison-Wesley 1992

Modul

Sub-pixel operators

14.3 Transformation

```
add_noise_white_contour_xld ( Contours : NoisyContours : NumRegrPoints, Amp : )
```

Verrauschen von XLD-Konturen.

`add_noise_white_contour_xld` erzeugt aus den Konturen, die in `Contours` übergeben werden, neue Konturen `NoisyContours` mit verrauschten Konturpunkten. Dazu wird für jeden Konturpunkt der Eingabekonturen die lokale Regressionsgeraden aus den `NumRegrPoints` benachbarten Punkten bestimmt. Dann wird der Punkt senkrecht zur Regressionsgeraden verschoben. Der Betrag dieser Verschiebung entspricht gleichverteiltem, mittelwertfreiem, weißem Rauschen mit maximaler Amplitude `Amp`, das mittels der C-Funktion „drand48“ mit zeitabhängigem Seed generiert wird.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
  Zu verrauschende Konturen.
- ▷ **`NoisyContours`** (output_object) ..... xld_cont(-array) ⇝ *Hobject*
  Verrauschte Konturen.
- ▷ **`NumRegrPoints`** (input_control) ..... integer ⇝ *integer*
  Anzahl der für die Regressionsgerade verwendeten Punkte.
  Defaultwert : 5
  Wertevorschläge : `NumRegrPoints` ∈ {3, 5, 7, 9}
  Restriktion : (`NumRegrPoints` ≥ 3) ∧ odd(`NumRegrPoints`)
- ▷ **`Amp`** (input_control) ..... real ⇝ *real*
  Maximale Amplitude des additiven Rauschens (in [-Amp,Amp] gleichverteilt).
  Defaultwert : 1.0
  Wertevorschläge : `Amp` ∈ {0.25, 0.5, 1.0, 1.5, 2.0, 3.0, 4.0, 5.0, 10.0}
  Restriktion : `Amp` > `0`

Beispiel

```
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, ERadius1, ERadius2, EStartPhi,
                         EEndPhi, 'EPointOrder).
```

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `add_noise_white_contour_xld` den Wert 2 (H_MSG_TRUE). Das Verhalten bei leerer Eingabe (keine Eingabebilder vorhanden) läßt sich mittels `set_system ('no_object_result',<Result>)` festlegen. Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`add_noise_white_contour_xld` ist *wiedereintrittsfähig* („reentrant“), *lokal* auszuführen („local“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `gen_ellipse_contour_xld`

Mögliche Nachfolgerfunktionen

`smooth_contours_xld`

Siehe auch

`smooth_contours_xld`, `add_noise_white`

Modul

Sub-pixel operators

affine_trans_contour_xld (
Contours : ContoursAffinTrans : HomMat2D :)

Anwendung einer affinen Abbildung auf XLD-Konturen.

affine_trans_contour_xld dient zur Vergrößerung, Verkleinerung, Drehung oder Verschiebung einer XLD-Kontur. Dazu verwendet die Prozedur eine Transformationsmatrix, die in HomMat2D übergeben wird. Diese kann mit den Routinen hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate und hom_mat2d_translate aufgebaut werden. Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Achtung

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont(-array) ⇝ *Hobject*
  Eingabe-XLD-Konturen.
- ▷ **ContoursAffinTrans** (output_object) . . . . . xld_cont(-array) ⇝ *Hobject*
  Transformierte XLD-Konturen.
- ▷ **HomMat2D** (input_control) . . . . . affine2d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 6

Ergebnis

Sind die Parameterwerte korrekt, dann liefert affine_trans_contour_xld den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

affine_trans_contour_xld ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

hom_mat2d_identity, hom_mat2d_translate, hom_mat2d_rotate, hom_mat2d_scale

Siehe auch

affine_trans_image, affine_trans_region

Modul

Sub-pixel operators

affine_trans_polygon_xld (
Polygons : PolygonsAffinTrans : HomMat2D :)

Anwendung einer affinen Abbildung auf XLD-Polygone

affine_trans_polygon_xld dient zur Vergrößerung, Verkleinerung, Drehung oder Verschiebung eines XLD-Polygons. Dazu verwendet die Prozedur eine Transformationsmatrix, die in HomMat2D übergeben wird. Diese kann mit den Routinen hom_mat2d_identity, hom_mat2d_scale, hom_mat2d_rotate und hom_mat2d_translate aufgebaut werden.

Achtung

Die Einträge der homogenen Transformationsmatrix werden so interpretiert, daß die *Zeilen*-Koordinate des Bildes der x-Koordinate der Matrix und die *Spalten*-Koordinate des Bildes der y-Koordinate der Matrix entspricht. Dies ist notwendig, um ein rechtshändiges Koordinatensystem, das für die homogenen Transformationsmatrizen angenommen wird, für das Bild zu erhalten. Insbesondere werden dadurch Rotationen im korrekten Drehsinn möglich. Beachten Sie, daß die Koordinatenreihenfolge (x,y) der Matrizen der üblichen Koordinatenreihenfolge (Zeile,Spalte) der Bilder entspricht.

Die XLD-Konturen, die möglicherweise von `Polygons` referenziert werden, werden weder transformiert noch mit den Ergebnispolygonen abgespeichert, da dies im allgemeinen nicht möglich ist, ohne Inkonsistenzen bei den Attributen der XLD-Konturen zu erzeugen. Daher kann es beim Aufruf von Operatoren, die auf die zu einem Polygon gehörigen Konturen zugreifen, z.B. `split_contours_xld`, zu Fehlermeldungen kommen.

Parameter

- ▷ **`Polygons`** (input_object) ..... xld_poly(-array) ⇝ *Hobject*
  Eingabe-XLD-Polygone.
- ▷ **`PolygonsAffinTrans`** (output_object) ..... xld_poly(-array) ⇝ *Hobject*
  Transformierte XLD-Polygone.
- ▷ **`HomMat2D`** (input_control) ..... affine2d-array ⇝ *real*
  Eingabe-Transformations-Matrix.
  Parameteranzahl : 6

Ergebnis

Sind die Parameterwerte korrekt, dann liefert `affine_trans_polygon_xld` den Wert 2 (H_MSG_TRUE). Gegebenenfalls wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

`affine_trans_polygon_xld` ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

`hom_mat2d_identity`, `hom_mat2d_translate`, `hom_mat2d_rotate`, `hom_mat2d_scale`

Siehe auch

`affine_trans_image`, `affine_trans_region`, `affine_trans_contour_xld`

Modul

Sub-pixel operators

`clip_contours_xld` (`Contours : ClippedContours : Row1, Column1, Row2, Column2 : )`

Clipping von XLD-Konturen.

`clip_contours_xld` beschneidet alle Konturen, die in `Contours` übergeben werden, d.h., es werden nur solche Konturpunkte eingetragen, die innerhalb des Rechtecks, das durch `Row1`, `Column1`, `Row2` und `Column2` gegeben ist, liegen. Gegebenenfalls werden Konturen aufgebrochen und als mehrere Konturen eingetragen. Die resultierenden Konturen werden in `ClippedContours` ausgegeben.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont-array ⇝ *Hobject*
  Zu beschneidende Konturen.
- ▷ **`ClippedContours`** (output_object) ..... xld_cont(-array) ⇝ *Hobject*
  Beschnittene Konturen.
- ▷ **`Row1`** (input_control) ..... rectangle.origin.y ⇝ *integer*
  Zeile der linken oberen Ecke des Clip-Rechtecks.
  Defaultwert : 0
  Wertevorschläge : `Row1` ∈ {0, 500, 1000, 1500, 2000}
- ▷ **`Column1`** (input_control) ..... rectangle.origin.x ⇝ *integer*
  Spalte der linken oberen Ecke des Clip-Rechtecks.
  Defaultwert : 0
  Wertevorschläge : `Column1` ∈ {0, 500, 1000, 1500, 2000}

▷ **Row2** (input_control) . . . . . rectangle.corner.y ⇝ *integer*
Zeile der rechten unteren Ecke des Clip-Rechtecks.
Defaultwert : 512
Wertevorschläge : Row2 ∈ {512, 1024, 1536, 2048}

▷ **Column2** (input_control) . . . . . rectangle.corner.x ⇝ *integer*
Spalte der rechten unteren Ecke des Clip-Rechtecks.
Defaultwert : 512
Wertevorschläge : Column2 ∈ {512, 1024, 1536, 2048}

Parallelisierungsinformation

clip_contours_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Mögliche Nachfolgerfunktionen

gen_polygons_xld

Siehe auch

clip_region, crop_part

Modul

Sub-pixel operators

combine_roads_xld (EdgePolygons, ModParallels, ExtParallels, CenterLines : RoadSides : MaxAngleParallel, MaxAngleColinear, MaxDistanceParallel, MaxDistanceColinear :)

Kombination von Straßenhypothesen aus zwei Auflösungsstufen.

Mit combine_roads_xld werden die Straßenhypothesen aus zwei Auflösungsstufen kombiniert. Es werden nur Straßenhypothesen selektiert, die in beiden Auflösungsstufen unterstützende Hypothesen besitzen. Dabei stellen die Eingabeobjekte EdgePolygons, ModParallels und ExtParallels die Ergebnisse einer Straßenextraktion auf der höchsten Auflösungsstufe dar. Der Parameter CenterLines ist das Ergebnis der Straßenextraktion auf einer niedrigen Auflösungsstufe. Es werden von den Polygonen EdgePolygons diejenigen in RoadSides zurückgegeben, für die Evidenz in beiden Stufen gefunden wurde. Die Parameter MaxAngleParallel und MaxAngleColinear geben die Winkel an, die zwei parallele bzw. kollineare Linien einschließen dürfen. Die Parameter MaxDistanceParallel und MaxDistanceColinear geben den Maximalabstand von parallelen bzw. kollinearen Linien an. Die Kombination erfolgt intern durch eine Anzahl von Regeln.

Parameter

▷ **EdgePolygons** (input_object) . . . . . xld_poly-array ⇝ *Hobject*
Polygone, die untersucht werden sollen.

▷ **ModParallels** (input_object) . . . . . xld_mod_para-array ⇝ *Hobject*
Modifizierte Parallelen für EdgePolygons.

▷ **ExtParallels** (input_object) . . . . . xld_ext_para-array ⇝ *Hobject*
Erweiterte Parallelen für EdgePolygons.

▷ **CenterLines** (input_object) . . . . . xld_poly-array ⇝ *Hobject*
Mittelachsen-Polygone, die untersucht werden sollen.

▷ **RoadSides** (output_object) . . . . . xld_poly-array ⇝ *Hobject*
Gefundene Straßenränder (als Polygone).

▷ **MaxAngleParallel** (input_control) . . . . . angle.rad ⇝ *real* / integer
Maximaler Winkel zwischen zwei parallelen Linien.
Defaultwert : 0.523598775598
Wertevorschläge : MaxAngleParallel ∈ {0.349065850399, 0.523598775598, 0.6981317008}
Restriktion : (0 ≤ MaxAngleParallel) ≤ (pi/2)

▷ **MaxAngleColinear** (input_control) . . . . . angle.rad ⇝ *real* / integer
Maximaler Winkel zwischen zwei kollinearen Linien.
Defaultwert : 0.261799387799
Wertevorschläge : MaxAngleColinear ∈ {0.174532925199, 0.261799387799, 0.349065850399}
Restriktion : (0 ≤ MaxAngleColinear) ≤ (pi/2)

▷ **MaxDistanceParallel** (input_control) . . . . . real ⇝ *real* / integer
Maximaler Abstand zwischen zwei parallelen Linien.
Defaultwert : 40
Wertevorschläge : MaxDistanceParallel ∈ {20, 30, 40, 50, 60}
Restriktion : MaxDistanceParallel > 0

▷ **MaxDistanceColinear** (input_control) . . . . . real ⇝ *real* / integer
Maximaler Abstand zwischen zwei kollinearen Linien.
Defaultwert : 40
Wertevorschläge : MaxDistanceColinear ∈ {20, 30, 40, 50, 60}
Restriktion : MaxDistanceColinear > 0

Parallelisierungsinformation

combine_roads_xld wird ohne Parallelisierung *exklusiv* gegenüber sich selbst („mutual exclusive“) ausgeführt.

Mögliche Vorgängerfunktionen

mod_parallels_xld, gen_polygons_xld, affine_trans_contour_xld

Mögliche Nachfolgerfunktionen

get_polygon_xld, get_lines_xld

Siehe auch

lines_gauss, lines_facet, get_channel_info, edges_sub_pix

Literatur

C. Steger, C. Glock, W. Eckstein, H. Mayer, B. Radig; „Model-based Road Extraction from Images“; in „Automatic Extraction of Man-Made Objects from Aerial and Space Images“; A. Gruen, O. Kuebler, P. Agouris (Editors); Birkhäuser Verlag (1995), pp. 275-284.

C. Heipke, C. Steger, R. Multhammer; „A Hierarchical Approach to Automatic Road Extraction from Aerial Imagery“; in „Integrating Photogrammetric Techniques with Scene Analysis and Machine Vision II“; D. M. McKeown, Jr., I. J. Dowman (Editors); Proc. SPIE 2486 (1995), pp. 222-231.

Modul

Sub-pixel operators

gen_parallel_contour_xld (Contours : ParallelContours : Mode, Distance :)

Berechnung der Parallelkontur einer XLD-Kontur

gen_parallel_contour_xld berechnet zu jeder der Eingabekonturen Contours eine parallele Kontur mit dem Abstand Distance. Die berechneten Konturen werden in ParallelContours zurückgegeben. Zur Berechnung der Parallelkontur wird der Normalenvektor der Eingabekontur in jedem Konturpunkt benötigt. Der Parameter Mode gibt an, wie die Normalenvektoren bestimmt werden. Falls Mode = **’gradient’**, wird angenommen, daß die Eigabekonturen Kanten sind, und die Normaleninformation aus der Gradientenrichtung der Kante gewonnen (siehe edges_sub_pix). Dazu muß bei der Eingabekontur das Attribut **’edge_direction’** definiert sein. Falls Mode = **’contour_normal’**, wird eine eventuell schon vorhandene Normaleninformation zur Berechnung der Normalen verwendet. Dazu muß das Konturattribut **’angle’** definiert sein (siehe lines_gauss oder edges_sub_pix). Falls schließlich Mode = **’regression_normal’**, wird die Normalenrichtung aus einer lokalen Regressionsgeraden in jedem Konturpunkt bestimmt. Dabei werden die Normalenvektoren so ausgerichtet, daß sie auf die rechte Seite der Kontur zeigen. Dieser Modus kann, im Gegensatz zu den ersten zwei Modi, für alle XLD-Konturen angewendet werden, ganz gleich wie sie erzeugt wurden.

Parameter

▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *Hobject*
Zu transformierende Konturen.

▷ **ParallelContours** (output_object) . . . . . xld_cont-array ⇝ *Hobject*
Parallelkonturen.

▷ **Mode** (input_control) . . . . . string ⇝ *string*
Modus, mit dem die Richtungsinformation gewonnen wird.
Defaultwert : ’regression_normal’
Wertevorschläge : Mode ∈ {’gradient’, ’contour_normal’, ’regression_normal’}

▷ **Distance** (input_control) . . . . . number ⇝ *real* / integer
Abstand der Parallelkontur.
Defaultwert : 1
Werteliste : Distance ∈ {0.2, 0.4, 0.6, 0.8, 1, 2, 3, 4, 5, 7, 10, 15, 20, 30, 40, 50}

Parallelisierungsinformation

gen_parallel_contour_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, threshold_sub_pix

Mögliche Nachfolgerfunktionen

gen_polygons_xld

Siehe auch

get_contour_xld

Modul

Sub-pixel operators

merge_cont_line_scan_xld (CurrConts, PrevConts : CurrMergedConts, PrevMergedConts : ImageHeight, Margin, MergeBorder, MaxImagesCont :)

Zusammenfügen von Konturen aus aufeinanderfolgenden Zeilenkamerabildern.

Der Operator merge_cont_line_scan_xld verbindet Konturen, die aus räumlich aneinandergrenzenden Bildern der Höhe ImageHeight extrahiert wurden und über die Bildränder hinaus ineinander überführt werden können. Die Bilder können beispielsweise von einer Zeilenkamera nacheinander aufgenommen worden sein. Dabei wird davon ausgegangen, dass CurrConts die Konturen des aktuellen Bildes enthält und PrevConts die des vorherigen, welches die räumliche Forfsetzung des aktuellen Bildes darstellt.

Mit Hilfe des Paramaters MergeBorder kann angegeben werden, ob die Oberkante des aktuellen Bildes an die Unterkante des vorherigen Bildes stößt (**'top'**) oder die Unterkannte des aktuellen Bildes an die Oberkante des vorherigen Bildes (**'bottom'**). MergeBorder definiert den Abstand vom Rand, in dem die Endpunkte der Kontur liegen müssen, um für das Merging berücksichtigt zu werden.

Der Parameter MaxImagesCont bestimmt bei der rekursiven Anwendung des Operators merge_cont_line_scan_xld, wieviel Bilder eine aktuelle Ausgangskontur maximal zurückreichen kann. Alle Konturpunkte aus früheren Bildern werden aus der Ausgabekontur entfernt.

Der Operator merge_cont_line_scan_xld liefert zwei Konturenarrays zurück. PrevMergedConts enthält alle Konturen, die ausschließlich in der alten Konturenliste enthalten sind und nicht mit den aktuellen Konturen verbunden werden konnten. In CurrMergedConts werden dagegen alle aktuellen und die mit diesen verbundenen Konturen eingetragen. Die miteinander verbundenen Konturen erscheinen als eine einzige neue Kontur in CurrMergedConts, wobei diese verbundenen Konturen über den Bildrand hinausgehen. Das bedeutet, dass der an eine aktuelle Kontur angebundene alte Teil um die Bildhöhe nach oben (MergeBorder=**'top'**) oder nach unten (MergeBorder=**'bottom'**) verschoben wird.

Parameter

▷ **CurrConts** (input_object) . . . . . xld_cont(-array) ⇝ *Hobject*
Aktuelle Eingabekonturen.

▷ **PrevConts** (input_object) . . . . . xld_cont(-array) ⇝ *Hobject*
Im vorhergehenden Zyklus zusammengefügte Konturen.

▷ **CurrMergedConts** (output_object) . . . . . xld_cont(-array) ⇝ *Hobject*
Aktuelle Konturen, die ggf. mit den alten Konturen verbunden wurden.

▷ **PrevMergedConts** (output_object) . . . . . xld_cont(-array) ⇝ *Hobject*
Alte Konturen, die nicht mit den aktuellen verbunden werden konnten.

▷ **ImageHeight** (input_control) . . . . . integer ⇝ *integer*
Höhe der Ausgangsbilder.
Defaultwert : 512
Werteliste : ImageHeight ∈ {240, 480, 512}

▷ **Margin** (input_control) . . . . . real ⇝ *real*
Maximaler Abstand der Konturen vom Rand.
Defaultwert : 0
Werteliste : Margin ∈ {0, 1, 2, 3, 4, 5}

▷ **MergeBorder** (input_control) . . . . . string ⇝ *string*
Im aktuellen Bild die Zeile, die mit dem vorhergehenden Bild zusammenstößt.
Defaultwert : ”top”
Werteliste : MergeBorder ∈ {”top”, ”bottom”}

▷ **MaxImagesCont** (input_control) . . . . . integer ⇝ *integer*
Maximale Anzahl der Bilder, über die sich eine Kontur erstrecken darf.
Defaultwert : 3
Wertevorschläge : MaxImagesCont ∈ {1, 2, 3, 4, 5}

Ergebnis

merge_cont_line_scan_xld liefert den Wert 2 (H_MSG_TRUE), falls die übergebenen Parameter korrekt sind. Ansonsten wird eine Exception-Behandlung durchgeführt.

Parallelisierungsinformation

merge_cont_line_scan_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Modul

Sub-pixel operators

regress_contours_xld (Contours : RegressContours : Mode, Iterations :)

Berechnung von Regressionsgeradenparametern für Konturen

Mit regress_contours_xld werden für die Konturen Contours folgende Parameter berechnet und in der Kontur als globale Attribute abgelegt:

- die Koordinaten des Normalenvektors der optimalen Regressionsgeraden durch alle Konturpunkte einer Kontur; dieser Normalenvektor verläuft vom Ursprung zur Regressionsgeraden (Attribute: **’regr_norm_row’**, **’regr_norm_col’**)
- der mittlere Abstand der Konturpunkte zu dieser Regressionsgeraden (Attribut: **’regr_mean_dist’**)
- die Standardabweichung des Abstandes der Konturpunkte zu dieser Regressionsgeraden (Attribut: **’regr_dev_dist’**).

Für Mode = **’no’** werden die Parameter fuer die optimale Regressionsgerade durch alle Konturpunkte berechnet. Bei der Berechnung der Regressionsgeraden können aber auch drei verschiedene Ausreißerbehandlungen vorgenommen werden. Ausreißer sind Konturpunkte, die „offensichtlich“ nicht in der globalen Konturrichtung liegen und deswegen die Regressionsgeradenberechnung „verfälschen“.

Mode =

- ’drop’: Alle Konturpunkte, die weiter als der durchschnittliche Abstand der Konturpunkte zur optimalen Regressionsgerade von dieser entfernt sind, werden bei der Berechnung der bereinigten Regressionsgerade unterschlagen.
- ’gauss’: Die Konturpunktabstände werden bei der Regressionsgeradenberechnung mit ihrer Eintrittswahrscheinlichkeit bei Normalverteilung um die optimale Regressionsgerade gewichtet.
- ’median’: Es wird ebenso eine Normalverteilung der Konturpunktabstaende zur optimalen Regressionsgeraden zugrunde gelegt, allerdings mit der ausreisserunabhängigen Standardabweichung $\frac{median(alleAbst.)}{0.6745}$. Die Abstände werden wieder gewichtet und Punkte ab einem gewissen Abstand unterschlagen.

Die Berechnung der bereinigten Regressionsgeraden kann mehrfach iteriert werden (Iterations).

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *Hobject*
  Eingabe-Konturen.
- ▷ **RegressContours** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Ausgabe-Konturen.
- ▷ **Mode** (input_control) ..... string ⇝ *string*
  Ausreißerbehandlung.
  Defaultwert : 'no'
  Werteliste : Mode ∈ {'no', 'drop', 'gauss', 'median'}
- ▷ **Iterations** (input_control) ..... integer ⇝ *integer*
  Anzahl Iterationen der Ausreißerbehandlung.
  Defaultwert : 1
  Wertevorschläge : Iterations ∈ {1, 2, 3, 5, 10, 20}

Parallelisierungsinformation

regress_contours_xld ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Mögliche Nachfolgerfunktionen

get_regress_params_xld

Siehe auch

smooth_contours_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Literatur

H. Suesse, K. Voss: „Adaptive Ausgleichsrechnung und Ausreißerproblematik für die digitale Bildverarbeitung"; Proc. 15. DAGM Symposium, Springer Verlag, Lübeck 1993

R. Haralick, L. Shapiro: „Computer and Robot Vision" Vol. 2; Kapitel 14.9, Addison-Wesley 1992

Modul

Sub-pixel operators

segment_contours_xld (Contours : ContoursSplit : Mode, SmoothCont, MaxLineDist1, MaxLineDist2 :)

Segmentation von Konturen in Liniensegmente und Kreis- oder Ellipsenbögen.

segment_contours_xld segmentiert die Eingabekonturen Contours in Linien, falls Mode=**'lines'**, in Linien und Kreisbögen, falls Mode=**'lines_circles'**, oder in Linien und Ellipsenbögen, falls Mode=**'lines_ellipses'**. Die segmentierten Konturen werden in ContoursSplit zurückgegeben. Die Unterscheidung, ob eine Ausgabekontur ein Liniensegment oder einen Kreis- oder Ellipsenbogen darstellt, erfolgt über das globale Konturattribut **'cont_approx'** (siehe get_contour_global_attrib_xld). Falls **'cont_approx'**=-1, liegt ein Liniensegment vor, falls **'cont_approx'**=0 ein Ellipsenbogen und falls **'cont_approx'**=1 ein Kreisbogen.

Intern führt segment_contours_xld zunächst eine Polygonapproximation der Eingabekonturen durch. Dadurch werden die Konturen in gekrümmten Bereichen übersegmentiert. Hierauf werden iterativ benachbarte Liniensegmente zu Kreis- bzw. Ellipsenbögen verschmolzen, falls dadurch die Kontur besser approximiert werden kann. Wenn in SmoothCont ein Wert > 0 angegeben wird, werden die Konturen zunächst geglättet (siehe smooth_contours_xld). Dies kann erforderlich sein, um sehr kurze Liniensegmente bei der Polygonapproximation zu verhindern und eine stabilere Kreis- bzw. Ellipsenanpassung zu erreichen, da durch die Glättung Ausreißer auf den Konturen unterdrückt werden.

Die initiale Polygonapproximation wird mit dem Algorithmus von Ramer (siehe gen_polygons_xld) mit einem Maximalabstand von MaxLineDist1 durchgeführt. Hierauf werden in benachbarte Liniensegmente Kreis- bzw. Ellipsenbögen angepaßt. Wenn der Maximalabstand des so entstehenden Bogens zur Kontur kleiner ist als der Maximalabstand der zwei betrachteten Liniensegmente, werden die zwei Liniensegmente durch diesen Bogen ersetzt. Dieser Vorgang wird solange wiederholt, bis keine Veränderungen mehr auftreten.

Hierauf werden die Teile der Kontur, die noch als Liniensegmente approximiert werden, einer Polygonsegmentation mit dem Maximalabstand MaxLineDist2 unterzogen, und die neu entstehenden Liniensegmente wiederum zu Kreis- oder Ellipsenbögen zusammengefaßt. Dies ändert die Ausgabe natürlich nur, falls

`MaxLineDist2` < `MaxLineDist1`. Dieses zweistufige Verfahren ist effizienter, als ein einstufiges Verfahren mit `MaxLineDist2`, da im ersten Schritt weniger Liniensegmente entstehen, und dadurch die Kreis- bzw. Ellipsenanpassung weniger oft durchgeführt werden muß. Daher könnnen Teile der Eingabekonturen, die sich durch lange Bögen approximieren lassen, effizienter gefunden werden. Im zweiten Schritt werden dann Teile der Kontur gefunden, die sich durch kurze Bögen approximieren lassen und die Endstücke der im ersten Schritt gefundenen Konturen werden verfeinert.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont(-array) ⇝ *Hobject*
  Konturen, die segmentiert werden sollen.
- ▷ **`ContoursSplit`** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Zerlegte Konturen.
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Modus für die Segmentation der Konturen.
  Defaultwert : 'lines_circles'
  Werteliste : `Mode` ∈ {'lines', 'lines_circles', 'lines_ellipses'}
- ▷ **`SmoothCont`** (input_control) ..... integer ⇝ *integer*
  Einzugsbereich für die Glättung der Konturen.
  Defaultwert : 5
  Wertevorschläge : `SmoothCont` ∈ {0, 3, 5, 7, 9}
  Restriktion : (`SmoothCont` = `0`) ∨ ((`SmoothCont` ≥ `3`) ∧ odd(`SmoothCont`))
- ▷ **`MaxLineDist1`** (input_control) ..... real ⇝ *real*
  Maximaler Abstand zwischen einer Kontur und der approximierenden Gerade (erster Durchlauf).
  Defaultwert : 4.0
  Wertevorschläge : `MaxLineDist1` ∈ {1.0, 1.5, 2.0, 2.5, 3.0, 3.5}
  Restriktion : `MaxLineDist1` ≥ `0.0`
- ▷ **`MaxLineDist2`** (input_control) ..... real ⇝ *real*
  Maximaler Abstand zwischen einer Kontur und der approximierenden Gerade (zweiter Durchlauf).
  Defaultwert : 2.0
  Wertevorschläge : `MaxLineDist2` ∈ {1.0, 1.5, 2.0, 2.5, 3.0, 3.5}
  Restriktion : `MaxLineDist2` ≥ `0.0`

Beispiel

```
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)
```

Parallelisierungsinformation

`segment_contours_xld` ist *lokal* auszuführen („local") und wird ohne Parallelisierung *vollständig exklusiv* („completely exclusive") ausgeführt.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `edges_sub_pix`

Mögliche Nachfolgerfunktionen

`fit_line_contour_xld`, `fit_ellipse_contour_xld`, `fit_circle_contour_xld`

Siehe auch

`split_contours_xld`, `get_contour_global_attrib_xld`, `smooth_contours_xld`, `gen_polygons_xld`

Modul

Sub-pixel operators

smooth_contours_xld (Contours : SmoothedContours : NumRegrPoints :)

Glättung von XLD-Konturen.

`smooth_contours_xld` erzeugt aus den Konturen, die in `Contours` übergeben werden, neue Konturen `SmoothedContours`, indem die Konturpunkte der Ausgabekontur durch Projektion des entsprechenden Punktes der Eingabekontur auf eine lokale Regressionsgerade berechnet werden. Diese Gerade wird aus den `NumRegrPoints` benachbarten Punkten des jeweils aktuellen Punktes berechnet. Diese Funktion sollte z.B. aufgerufen werden, bevor Konturen skaliert werden.

Parameter

- ▷ **Contours** (input_object) ..... xld_cont-array ⇝ *Hobject*
  Zu glättende Konturen.
- ▷ **SmoothedContours** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Geglättete Konturen.
- ▷ **NumRegrPoints** (input_control) ..... integer ⇝ *integer*
  Anzahl der für die Regressionsgerade verwendeten Punkte.
  Defaultwert : 5
  Wertevorschläge : `NumRegrPoints` ∈ {3, 5, 7, 9}
  Restriktion : (`NumRegrPoints` ≥ 3) ∧ odd(`NumRegrPoints`)

Parallelisierungsinformation

`smooth_contours_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Mögliche Nachfolgerfunktionen

`affine_trans_contour_xld`, `gen_polygons_xld`, `local_max_contours_xld`

Siehe auch

`get_contour_xld`

Modul

Sub-pixel operators

split_contours_xld (Polygons : Contours : Mode, Weight, Smooth :)

Aufspalten von Konturen an regional markanten Punkten.

Mit `split_contours_xld` werden die den Polygonen `Polygons` zugrundeliegenden Konturen an regional markanten Punkten aufgespalten. Im Modus **’polygon’** erfolgt die Aufspaltung gemäß den Polygonen selbst. Im Modus **’dominant’** werden Konturpunkte zur Aufspaltung verwendet, für die die in ihnen ermittelte Konturrichtungsänderung den (empirischen) Grenzwert 2π`Weight`/*Konturlänge* überschreitet und in deren (empirischen) Umgebung von $\sqrt{\texttt{Smooth}} * \log(\textit{Konturlänge})$ Punkten keine größere Richtungsänderung auftritt. Die Konturrichtung wird mit einer optimalen Regressionsgeraden durch alle Punkte der Nachbarschaft der Breite `Smooth` ermittelt. Die so ermittelte Richtungsinformation für jeden Konturpunkt wird vor der Suche nach markanten Punkten mit einer Gaussmaske der Breite `Smooth` geglättet. `Weight` ist damit ein Gewichtungsfaktor für die Empfindlichkeit des Operators. Je größer `Weight` gewählt wird, umso weniger markante Punkte werden gefunden.

Parameter

- ▷ **Polygons** (input_object) . . . . . xld_poly(-array) ⇝ *Hobject*
  Polygone, deren zugrundeliegende Konturen aufgespalten werden sollen.
- ▷ **Contours** (output_object) . . . . . xld_cont(-array) ⇝ *Hobject*
  Aufgespaltene Konturen.
- ▷ **Mode** (input_control) . . . . . string ⇝ *string*
  Modus fuer das Aufspalten der Konturen.
  Defaultwert : 'polygon'
  Werteliste : `Mode` ∈ {'polygon', 'dominant'}
- ▷ **Weight** (input_control) . . . . . integer ⇝ *integer*
  Gewicht für die Empfindlichkeit.
  Defaultwert : 1
- ▷ **Smooth** (input_control) . . . . . integer ⇝ *integer*
  Breite der Glättungsmaske.
  Defaultwert : 5

Parallelisierungsinformation

`split_contours_xld` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_polygons_xld`

Mögliche Nachfolgerfunktionen

`regress_contours_xld`

Siehe auch

`gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`

Modul

Sub-pixel operators

union_straight_contours_histo_xld (Contours : UnionContours, SelectedContours : RefLineStartRow, RefLineStartColumn, RefLineEndRow, RefLineEndColumn, Width, MaxWidth, FilterSize : HistoValues)

Vereinigen von benachbarten geraden Konturen anhand des Abstandes von einer gegebenen Linie.

Mit `union_straight_contours_histo_xld` werden die Konturen `Contours` verglichen und unter gewissen Voraussetzungen vereinigt.

Vereinigung von Konturen, die nahe beieinander liegen. Es wir die minmale und maximale Distanz der Kontur zu einer gegebenen Referenzlinie berechnet. Mit den Distanzen wird ein Histogramm erzeugt. Soll das Histogramm geglättet werden, so muß `FilterSize` > 1 sein. Das resultierende Histogramm wird anschließend in Bereiche unterteilt (von Minima zu Minima). Zum Abschluss werden die Konturen, die in einem Bereich liegen, zu einer neuen Kontur zusammengefaßt. Ist die Breite des Bereichs gröeßer als `MaxWidth`, so werden alle Konturen des Bereichs ignoriert (verworfen) Liegt eine Kontur in mehreren Bereichen, so wird diese ebenfalls ignoriert. Bei parallel verlaufenden Konturen besteht die Gefahr, daß nebeneinander liegende Konturen vereinigt werden.

Für jede durch Vereinigung neu entstandene Kontur werden die Regressionsgeradenparameter neu berechnet.

Die resultierenden Konturen können nicht dargestellt werden.

Achtung

Bevor Konturen mit `union_straight_contours_histo_xld` vereinigt werden können, müssen die Regressionsgeradenparameter mit `regress_contours_xld` berechnet werden.

Parameter

- ▷ **Contours** (input_object) . . . . . xld_cont-array ⇝ *Hobject*
  Eingabe-Konturen.
- ▷ **UnionContours** (output_object) . . . . . xld_cont-array ⇝ *Hobject*
  Ausgabe-Konturen.
- ▷ **SelectedContours** (output_object) . . . . . xld_cont-array ⇝ *Hobject*
  Ausgabe-Konturen.

- ▷ **RefLineStartRow** (input_control) ..... line.begin.y ⇝ *integer*
  y-Koordinate des Startpunktes der Referenzlinie.
  Defaultwert : 0
- ▷ **RefLineStartColumn** (input_control) ..... line.begin.x ⇝ *integer*
  x-Koordinate des Startpunktes der Referenzlinie.
  Defaultwert : 0
- ▷ **RefLineEndRow** (input_control) ..... line.end.y ⇝ *integer*
  y-Koordinate des Endpunktes der Referenzlinie.
  Defaultwert : 0
- ▷ **RefLineEndColumn** (input_control) ..... line.end.x ⇝ *integer*
  x-Koordinate des Endpunktes der Referenzlinie.
  Defaultwert : 0
- ▷ **Width** (input_control) ..... integer ⇝ *integer*
  Maximale Distanz.
  Defaultwert : 1
- ▷ **MaxWidth** (input_control) ..... integer ⇝ *integer*
  Maximale Breite zwischen zwei Minimas.
  Defaultwert : 1
- ▷ **FilterSize** (input_control) ..... integer ⇝ *integer*
  Grße des Glättungsfilters.
  Defaultwert : 1
  Typischer Wertebereich : $1 \leq$ FilterSize ≤ 63
- ▷ **HistoValues** (output_control) ..... integer-array ⇝ *integer*
  Ausgabe der Werte des Histogramms.

Parallelisierungsinformation

union_straight_contours_histo_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

regress_contours_xld

Siehe auch

fit_line_contour_xld, get_contour_xld, get_contour_attrib_xld, gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix, get_regress_params_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Modul

Sub-pixel operators

union_straight_contours_xld (Contours : UnionContours : MaxDist, MaxDiff, Percent, Mode, Iterations :)

Vereinigen von benachbarten geraden Konturen mit ähnlicher Richtung.

Mit union_straight_contours_xld werden die Konturen Contours verglichen und unter gewissen Voraussetzungen vereinigt. Diese Konkatenation erfolgt nicht rekursiv und jeweils nur zwischen zwei Konturen. Mit dem Parameter Iterations kann gesteuert werden, wie oft dieser Vereinigungsschritt wiederholt wird.

Zwei Konturen werden vereinigt, wenn deren kürzester Endpunktabstand (die Endpunkte sind die Fußpunkte der Lote von den Konturendpunkten auf die optimale Regressionsgerade) kleiner als MaxDist ist und wenn ihre Richtungsdifferenz (bzgl. der zugehörigen optimalen Regressionsgeraden) kleiner als MaxDiff (Bogenmaß) ist.

Wenn nur eine der beiden Bedingungen zutrifft, kann die Entscheidung zugunsten einer Vereinigung durch die Gewichtung Percent beeinflußt werden, indem die prozentuale Überschreitung des angegebenen Grenzwertes gegen die prozentuale Unterschreitung des anderen Grenzwertes angerechnet wird. Dabei wird der Endpunktabstand mit Percent, die Richtungsdifferenz mit $100 -$ Percent bewertet.

Wenn zwei Konturen beispielsweise eine minimalen Endpunktabstand von 5.0 und eine Richtungsdifferenz von 0.5 haben (bei Grenzwerten MaxDist = 4.0 und MaxDiff = 0.625), so weichen beide Werte von den Grenzwerten um 25% ab. Durch die Wahl von Percent = 60% fällt nun der große Endpunktabstand stärker ins Gewicht, als

die geringe Richtungsdifferenz honoriert wird, die Konturen werden nicht vereinigt. Vice versa, wenn in diesem Fall `Percent` = 40% gewählt wird: die Konturen werden vereinigt.

Bei `Percent` = 100% geht nur der Endpunktabstand in die Entscheidung ein, bei `Percent` = 0% geht nur die Richtungsdifferenz in die Entscheidung ein, bei `Percent` = 50% sind beide Merkmale gleichberechtigt.

Bei parallel verlaufenden Konturen besteht die Gefahr, daß nebeneinander liegende Konturen vereinigt werden. Soll dieser Effekt verhindert werden, so muß für `Mode` 'noparallel', ansonsten 'paralleltoo' angegeben werden. Für 'every' werden die Konturen bedingungslos vereinigt. Alle anderen Parameter haben in diesem Fall keinen Einfluß.

Für jede durch Vereinigung neu entstandene Kontur werden die Regressionsgeradenparameter neu berechnet.

Achtung

Bevor Konturen mit `union_straight_contours_xld` vereinigt werden können, müssen die Regressionsgeradenparameter mit `regress_contours_xld` berechnet werden.

Parameter

- ▷ **`Contours`** (input_object) ..... xld_cont-array ⇝ *Hobject*
  Eingabe-Konturen.
- ▷ **`UnionContours`** (output_object) ..... xld_cont-array ⇝ *Hobject*
  Ausgabe-Konturen.
- ▷ **`MaxDist`** (input_control) ..... real ⇝ *real*
  Maximaler Abstand der Endpunkte.
  Defaultwert : 5.0
- ▷ **`MaxDiff`** (input_control) ..... angle.rad ⇝ *real*
  Maximale Richtungsdifferenz.
  Defaultwert : 0.5
- ▷ **`Percent`** (input_control) ..... real ⇝ *real*
  Gewichtungsfaktor für die zwei Selektionskriterien.
  Defaultwert : 50.0
- ▷ **`Mode`** (input_control) ..... string ⇝ *string*
  Berücksichtigung paralleler Konturen.
  Defaultwert : 'noparallel'
  Werteliste : `Mode` ∈ {'noparallel', 'paralleltoo', 'every'}
- ▷ **`Iterations`** (input_control) ..... string ⇝ *string* / integer
  Anzahl Iterationen oder 'maximum'.
  Defaultwert : 'maximum'
  Wertevorschläge : `Iterations` ∈ {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'maximum'}
  Typischer Wertebereich : 1 ≤ `Iterations` ≤ 500 (lin)
  Minimale Schrittweite : 1
  Empfohlene Schrittweite : 1

Parallelisierungsinformation

`union_straight_contours_xld` ist *wiedereintrittsfähig* („reentrant") und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`regress_contours_xld`

Siehe auch

`fit_line_contour_xld`, `get_contour_xld`, `get_contour_attrib_xld`, `gen_contours_skeleton_xld`, `lines_gauss`, `lines_facet`, `edges_sub_pix`, `get_regress_params_xld`, `get_contour_global_attrib_xld`, `query_contour_global_attribs_xld`

Modul

Sub-pixel operators

14.4 Zugriff

`get_contour_xld` (`Contour` : : : `Row, Col`)

Auslesen der Koordinaten einer XLD-Kontur.

get_contour_xld liefert für die XLD-Kontur Contour folgende Werte:

Row Zeilen-Koordinate des Konturpunktes
Col Spalten-Koordinate des Konturpunktes

Parameter

- ▷ **Contour** (input_object) ..... xld_cont ⇝ *Hobject*
  Eingabe-Kontur.
- ▷ **Row** (output_control) ..... point.y-array ⇝ *real*
  Zeilen-Koordinate des Konturpunktes.
- ▷ **Col** (output_control) ..... point.x-array ⇝ *real*
  Spalten-Koordinate des Konturpunktes.

Parallelisierungsinformation

get_contour_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_contours_skeleton_xld, lines_gauss, lines_facet, edges_sub_pix

Siehe auch

get_contour_attrib_xld, query_contour_attribs_xld, get_contour_global_attrib_xld, query_contour_global_attribs_xld

Modul

Sub-pixel operators

get_lines_xld (Polygon : : : BeginRow, BeginCol, EndRow, EndCol, Length, Phi)

Ausgabe aller Polygonseiten als Linien inkl. Länge und Orientierung.

get_lines_xld gibt alle Eingabepolygone Polygon als Linien aus. Im einzelnen werden zurückgebeben:

BeginRow: Zeilen-Koordinate des Anfangspunktes
BeginCol: Spalten-Koordinate des Anfangspunktes
EndRow: Zeilen-Koordinate des Endpunktes
EndCol: Spalten-Koordinate des Endpunktes
Length: Seitenlänge
Phi: Seitenorientierung

Parameter

- ▷ **Polygon** (input_object) ..... xld_poly(-array) ⇝ *Hobject*
  Polygone.
- ▷ **BeginRow** (output_control) ..... line.begin.y-array ⇝ *real*
  Zeilen-Koordinate des Anfangspunktes.
- ▷ **BeginCol** (output_control) ..... line.begin.x-array ⇝ *real*
  Spalten-Koordinate des Anfangspunktes.
- ▷ **EndRow** (output_control) ..... line.end.y-array ⇝ *real*
  Zeilen-Koordinate des Endpunktes.
- ▷ **EndCol** (output_control) ..... line.end.x-array ⇝ *real*
  Spalten-Koordinate des Endpunktes.
- ▷ **Length** (output_control) ..... real-array ⇝ *real*
  Seitenlänge.
- ▷ **Phi** (output_control) ..... angle.rad-array ⇝ *real*
  Seitenorientierung.

Parallelisierungsinformation

get_lines_xld ist *wiedereintrittsfähig* („reentrant“) und wird automatisch *parallelisiert* (auf *Tupel-Ebene*).

Mögliche Vorgängerfunktionen

gen_polygons_xld

Alternativen

get_polygon_xld

Modul

Sub-pixel operators

get_parallels_xld (Parallels : : : Row1, Col1, Length1, Phi1, Row2, Col2, Length2, Phi2)

Ausgabe der parallelen Polygonseiten inkl. Länge und Orientierung.

get_parallels_xld liefert für die parallelen Polygone Parallels folgende Werte:

Row1: Zeilen-Koordinate des Polygonpunktes auf Polygon P1
Col1: Spalten-Koordinate des Polygonpunktes auf Polygon P1
Length1: Seitenlänge auf Polygon P1
Phi2: Seitenorientierung auf Polygon P1
Row2: Zeilen-Koordinate des Polygonpunktes auf Polygon P2
Col2: Spalten-Koordinate des Polygonpunktes auf Polygon P2
Length2: Seitenlänge auf Polygon P2
Phi2: Seitenorientierung auf Polygon P2

Parameter

▷ **Parallels** (input_object) ..... xld ⇝ *Hobject*
Parallelen.
▷ **Row1** (output_control) ..... polygon.y-array ⇝ *integer*
Zeilen-Koordinate des Polygonpunktes auf Polygon P1.
▷ **Col1** (output_control) ..... polygon.x-array ⇝ *integer*
Spalten-Koordinate des Polygonpunktes auf Polygon P1.
▷ **Length1** (output_control) ..... real-array ⇝ *real*
Seitenlänge auf Polygon P1.
▷ **Phi1** (output_control) ..... angle.rad-array ⇝ *real*
Seitenorientierung auf Polygon P1.
▷ **Row2** (output_control) ..... polygon.y-array ⇝ *integer*
Zeilen-Koordinate des Polygonpunktes auf Polygon P2.
▷ **Col2** (output_control) ..... polygon.x-array ⇝ *integer*
Spalten-Koordinate des Polygonpunktes auf Polygon P2.
▷ **Length2** (output_control) ..... real-array ⇝ *real*
Seitenlänge auf Polygon P2.
▷ **Phi2** (output_control) ..... angle.rad-array ⇝ *real*
Seitenorientierung auf Polygon P2.

Parallelisierungsinformation

get_parallels_xld ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

gen_parallels_xld

Siehe auch

get_polygon_xld, get_lines_xld

Modul

Sub-pixel operators

get_polygon_xld (Polygon : : : Row, Col, Length, Phi)

Auslesen der Daten eines XLD-Polygons.

get_polygon_xld liefert für das XLD-Polygon Polygon folgende Werte:

Row: Zeilen-Koordinate des Polygonpunktes
Col: Spalten-Koordinate des Polygonpunktes
Length: Länge der Polygonseite zwischen Punkt i und $i + 1$
Phi: Winkel der Polygonseite zwischen Punkt i und $i + 1$

Parameter

- ▷ **Polygon** (input_object) . . . . . xld_poly ⇝ *Hobject*
  Eingabe-Polygon.
- ▷ **Row** (output_control) . . . . . point.y-array ⇝ *real*
  Zeilen-Koordinate des Polygonpunktes.
- ▷ **Col** (output_control) . . . . . point.x-array ⇝ *real*
  Spalten-Koordinate des Polygonpunktes.
- ▷ **Length** (output_control) . . . . . real-array ⇝ *real*
  Länge der Polygonseite.
- ▷ **Phi** (output_control) . . . . . angle.rad-array ⇝ *real*
  Winkel der Polygonseite.

Parallelisierungsinformation

`get_polygon_xld` ist *wiedereintrittsfähig* („reentrant“) und wird *nicht* parallelisiert.

Mögliche Vorgängerfunktionen

`gen_polygons_xld`

Alternativen

`get_lines_xld`

Modul

Sub-pixel operators

Index