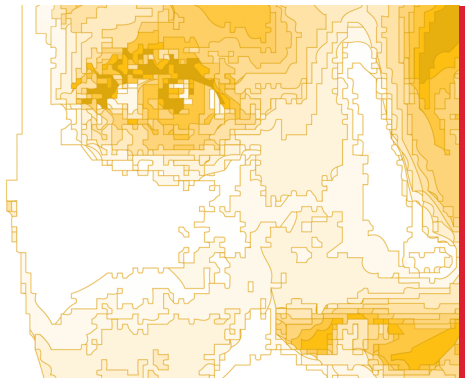
HALCON

the Power of Machine Vision

Solution Guide I

Basics

8.0



MVTec Software GmbH

Building Vision for Business

How to use HALCON's machine vision methods, Version 8.0.4

Edition 1	June 2007	(HALCON 8.0)
Edition 1a	October 2007	(HALCON 8.0.1)
Edition 1b	April 2008	(HALCON 8.0.2)

More information about HALCON can be found at:

`http://www.halcon.com/`

Contents

Chapter 1

Guide to HALCON Methods

This manual introduces you to important machine vision methods. To guide you from your specific application to the sections of the documentation to read, this section lists common application areas and the methods used for them.

Generally, a lot of applications use the following methods:

- Image Acquisition on page 13 for accessing images via an image acquisition device or via file.
- Visualization on page 269 for the visualization of, e.g., artificially created images or results of an image processing task.
- Region of interest on page 19 for reducing the search space for a following image processing task.
- Morphology (Reference Manual, chapter "Morphology"), e.g., for the elimination of small gaps or protrusions from regions or from structures in gray value images.

Other methods are more specific and thus are suited for specific application areas. Additionally, some application areas are part of another application area. To make the relations more obvious, for the following application areas the corresponding methods and related application areas are listed:

- Color Inspection (page 6)
- Completeness Check (page 6)
- Identification (page 6)
- Measuring and Comparison 2D (page 7)
- Measuring and Comparison 3D (page 7)
- Object Recognition 2D (page 8)
- Object Recognition 3D (page 8)

1.1 Color Inspection

For color inspection, see the descriptions for Color Processing on page 191.

1.2 Completeness Check

Completeness checks can be realized by different means. Common approaches are:

- Object and position recognition 2D/3D (see page 8 ff), which is suitable, e.g., when inspecting objects on an assembly line.
- Variation Model on page 159, which compares images containing similar objects and returns the difference between them considering a certain tolerance at the object's border.

1.3 Identification

Dependent on the symbols or objects you have to identify, the following methods are suitable:

- Identify symbols or characters
  - 1D Bar Code on page 209
  - 2D Data Code on page 219
  - OCR on page 233
- Identify general objects
  - Object and position recognition 2D/3D (see page 8 ff)

1.4 Measuring and Comparison 2D

For measuring 2D features in images, several approaches are available. In the Solution Guide II-E, section 4.1 on page 23, a graph leads you from the specific features you want to measure and the appearance of the objects in the image to the suitable measuring approach. Generally, the following approaches are common:

- Blob Analysis on page 33 for objects that consist of regions of similar gray value, color, or texture.
- Contour Processing on page 97 for objects that are represented by clear-cut edges. The contours can be obtained by different means:
  - Edge Extraction (Pixel-Precise) on page 73 if pixel precision is sufficient.
  - Edge Extraction (Subpixel-Precise) on page 85 if subpixel precision is needed.
- Template Matching on page 115 for objects that can be represented by a template. Template matching comprises different approaches, amongst others shape-based matching. For detailed information about shape-based matching see the Solution Guide II-B.
- 1D Measuring on page 57 if you want to obtain the positions, distances, or angles of edges that are measured along a line or an arc. More detailed information can be found in the Solution Guide II-D.

1.5 Measuring and Comparison 3D

For measuring in 3D, the following approaches are available:

- The approaches used for measuring and comparison 2D (see page 7) in combination with a camera calibration (see Solution Guide II-F, section 3.1 on page 29) for measuring objects that are viewed by a single camera and that lie in a single plane.
- Pose estimation (Solution Guide II-F, section 6 on page 86) for the estimation of the poses of 3D objects that are viewed by a single camera and for which knowledge about their 3D model (known points, known circular or rectangular shape) is available.
- Stereo on page 259 for measuring in images obtained by a binocular stereo system. Further information can be found in the Solution Guide II-F, section 7 on page 91.
- Depth from focus (Reference Manual, chapter "Tools ▷ Shape-from") for getting depth information from a sequence of images of the same object but with different focus.
- Shape from shading (Reference Manual, chapter "Tools ▷ Shape-from") for getting information about an object's shape because of its shading behavior (e.g., by the operator `phot_stereo`).

1.6 Object Recognition 2D

For finding specific objects in images, various methods are available. Common approaches comprise:

- Blob Analysis on page 33 for objects that are represented by regions of similar gray value, color, or texture.
- Contour Processing on page 97 for objects that are represented by clear-cut edges. The contours can be obtained by different means:
  - Edge Extraction (Pixel-Precise) on page 73 if pixel precision is sufficient.
  - Edge Extraction (Subpixel-Precise) on page 85 if subpixel precision is needed.
- Template Matching on page 115 for objects that can be represented by a template. Template matching comprises different approaches, amongst others shape-based matching. For detailed information about shape-based matching see the Solution Guide II-B.
- Classification on page 171 for the recognition of objects by a classification using, e.g., Gaussian mixture models, neural nets, or support vector machines.
- Color Processing on page 191 for the recognition of objects that can be separated from the background by their color.
- Texture analysis (Reference Manual, chapter "Filter ▷ Texture") for the recognition of objects that can be separated from the background by their specific texture.
- Movement detection (see section 1.13 on page 10) for the recognition of moving objects.

1.7 Object Recognition 3D

For the recognition of 3D objects that are described by a 3D Computer Aided Design (CAD) model, see the descriptions for 3D Matching on page 143.

1.8 Position Recognition 2D

In parts, the approaches for measuring 2D features are suitable to get the position of objects. In the Solution Guide II-E, section 4.1 on page 23, a graph leads you from several specific features, amongst others the object position, and the appearance of the objects in the image to the suitable approach. For position recognition, in particular the following approaches are common:

- Blob Analysis on page 33 for objects that consist of regions of similar gray value, color, or texture.
- Contour Processing on page 97 for objects that are represented by clear-cut edges. The contours can be obtained by different means:
  - Edge Extraction (Pixel-Precise) on page 73 if pixel precision is sufficient.

  - Edge Extraction (Subpixel-Precise) on page 85 if subpixel precision is needed.

- Template Matching on page 115 for objects that can be represented by a template. Template matching comprises different approaches, amongst others shape-based matching. For detailed information about shape-based matching see the Solution Guide II-B.
- 1D Measuring on page 57 if you want to obtain the positions of edges that are measured along a line or an arc. More detailed information can be found in the Solution Guide II-D.

1.9 Position Recognition 3D

For the position recognition of 3D objects, the following approaches are available:

- The approaches used for measuring and comparison 2D (see page 7) in combination with a camera calibration (see Solution Guide II-F, section 3.1 on page 29) for measuring objects that are viewed by a single camera, and which lie in a single plane.
- Pose estimation (Solution Guide II-F, section 6 on page 86) for the estimation of the poses of 3D objects that are viewed by a single camera and for which knowledge about their 3D model (known points, known circular or rectangular shape) is available.
- Stereo on page 259 for measuring positions in images obtained by a binocular stereo system. Further information can be found in the Solution Guide II-F, section 7 on page 91.
- 3D Matching on page 143 for objects that are searched for based on a 3D Computer Aided Design (CAD) model.

1.10 Print Inspection

For print inspection, the suitable method depends on the type of print you need to inspect:

- Optical character verification (Reference Manual, chapter “Tools ▷ OCV”) for the verification of characters.
- 2D Data Code on page 219 for inspecting the quality of a printed 2D data code symbol. Further information can be found in the Solution Guide II-C, section 6 on page 43.
- Variation Model on page 159, which compares images containing similar objects and returns the difference between them considering a certain tolerance at the object’s border.
- Component-based matching (part of Template Matching on page 115), if the print is built by several components which may vary in their relation to each other (orientation and distance) within a specified tolerance.

1.11 Quality Inspection

How to inspect the quality of an object depends on the features describing the quality. The following application areas and the methods used by them can be applied for quality inspection:

- Surface inspection (see page 11) if, e.g., scratches in an object's surface have to be detected.
- Completeness check (see page 6) if you need to check an object for missing parts.
- Measuring and comparison 2D/3D (see page 7 ff) if an object has to fulfill certain requirements related to its area, position, orientation, dimension, or number of parts.

Additionally, Classification on page 171 can be used to check the color or texture of objects.

1.12 Robot Vision

For robot vision, you can combine the approaches for object and position recognition 2D/3D (see page 8 ff) with the hand-eye calibration described in the Solution Guide II-F, section 8 on page 112.

1.13 Security System

For a sequence analysis or movement detection you can use, e.g., one of the following approaches:

- Blob Analysis on page 33, e.g., by using the operator `dyn_threshold` to obtain the difference between two images and thus detect moving objects. The approach is fast, but detects objects only as long as they are moving.
- Background estimator (Reference Manual, chapter "Tools ▷ Background-Estimator") for the recognition of moving objects even if they stop temporarily. It adapts to global changes concerning, e.g., illumination.
- Optical flow (Reference Manual, chapter "Filter ▷ Match") for the recognition of moving objects even if they stop temporarily. Because of complex calculations, it is slower than the other approaches, but additionally returns the velocity for each object.
- Kalman filter (Reference Manual, chapter "Tools ▷ Kalman-Filter") can be applied after the recognition of moving objects to predict the future positions of objects.

For the recognition of, e.g., irises or faces, Classification on page 171 may be suitable.

Examples solving different tasks relevant for security systems can be found via the Open Example dialog inside HDevelop for the category `Industry/Surveillance and Security`.

1.14 Surface Inspection

For surface inspection, several approaches are available:

- Via comparison with a reference image
  - Variation Model on page 159, which compares images containing similar objects and returns the difference between them. This is suited especially if you need to detect irregularities that are placed inside the object area, whereas small irregularities at the object's border can be tolerated.
- Via comparison with a reference pattern or color
  - Texture analysis (see page 11)
  - Color Processing on page 191
  - Classification on page 171
- Via defect description for uniformly structured surfaces
  - Blob Analysis on page 33 for objects that consist of regions of similar gray value, color, or texture.
  - Contour Processing on page 97 for objects that are represented by clear-cut edges. The contours can be obtained by an Edge Extraction (Pixel-Precise) on page 73 if pixel precision is sufficient, or by an Edge Extraction (Subpixel-Precise) on page 85 if subpixel precision is needed.

If a single image is not suited to cover the object to inspect, several approaches exist to combine images after their acquisition:

- Calibrated mosaicking (Solution Guide II-F, section 4 on page 60) for a high-precision mosaicking of a discrete number of overlapping images obtained by two ore more cameras.
- Uncalibrated mosaicking (Solution Guide II-F, section 5 on page 72) for a less precise mosaicking of a discrete number of overlapping images obtained as an image sequence.
- Combination of lines obtained by a line scan camera to so-called pages (Solution Guide II-A, section 6.3 on page 37) for inspecting continuous material on an assembly line.

1.15 Texture Inspection

Common approaches for texture inspection are:

- Fast Fourier Transformation (Reference Manual, chapter "Filter ▷ FFT")
- Classification on page 171
- Texture analysis (Reference Manual, chapter "Filter ▷ Texture")

Chapter 2

Image Acquisition

Obviously, the acquisition of images is a task that must be solved in all machine vision applications. Unfortunately, this task mainly consists of interacting with special, non-standardized hardware in the form of the image acquisition device, e.g., a frame grabber board or an IEEE 1394 camera. To let you concentrate on the actual machine vision problem, HALCON provides you with interfaces performing this interaction for a large number of image acquisition devices (see `http://www.mvtec.com/halcon/framegrabber` for the latest information).

Within your HALCON application, the task of image acquisition is thus reduced to a few lines of code, i.e., a few operator calls. What's more, this simplicity is not achieved at the cost of limiting the available functionality: Using HALCON, you can acquire images from various configurations of acquisition devices and cameras in different timing modes.

Besides acquiring images from cameras, HALCON also allows you to input images that were stored in files (supported formats: BMP, TIFF, GIF, JPEG, PNG, PNM, PCX, XWD). Of course, you can also store acquired images in files.

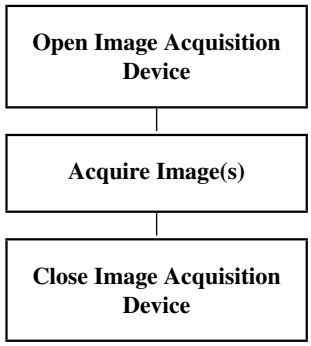
2.1 Basic Concept

Acquiring images with HALCON basically consists of three steps. Reading images from files is even simpler: It consists of a single call to the operator `read_image`.

2.1.1 Open Image Acquisition Device

If you want to acquire images from a frame grabber board or an image acquisition device like an IEEE 1394 camera, the first step is to connect to this device. HALCON relieves you of all device-specific details; all you need to do is to call the operator `open_framegrabber`, specifying the name of the corresponding image acquisition interface.

There is also a "virtual" image acquisition interface called `File`. As its name suggests, this "frame grabber" reads images from files, and also from so-called image sequence files. The latter are HALCON-specific files, typically with the extension `.seq`; they contain a list of image file names, separated by new



lines (you can create it easily using a text editor). If you connect to such a sequence, subsequent calls to grab_image return the images in the sequence specified in the file. Alternatively, you can also read all images from a specific directory. Then, you do not have to create a sequence file, but simply specify the directory name instead of the sequence file as value for the parameter 'CameraType'. Now, subsequent calls to grab_image return the images found in the specified image directory. Both approaches are useful if you want to test your application with a sequence of image files and later switch to a real image acquisition device.

2.1.2 Acquire Image(s)

Having connected to the device, you acquire images by simply calling grab_image.

To load an image from disk, you use read_image. Images are searched for in the current directory and in the directories specified in the environment variable HALCONIMAGES.

2.1.3 Close Image Acquisition Device

At the end of the application, you close the connection to the image acquisition device to free its resources with the operator close_framegrabber.

2.1.4 A First Example

As already remarked, acquiring images from file corresponds to a single operator call:

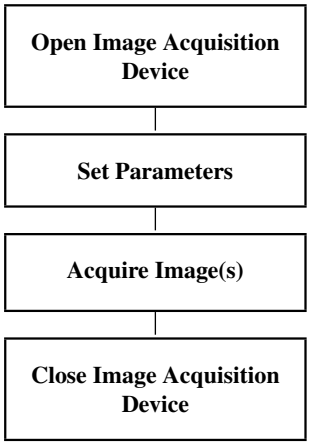
```
read_image (Image, 'particle')
```

The following code processes images read from an image sequence file:

```
SequenceName := 'datacode/ecc200/ecc200_cpu_light.seq'
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1,
                   'default', SequenceName, 'default', -1, -1, FGHandle)
while (1)
grab_image (Image, FGHandle)
   ... process image ...
endwhile
```


2.2 Extended Concept

In real applications, it is typically not enough to tell the camera to acquire an image; instead, it may be important that images are acquired at the correct moment or rate, and that the camera and the image acquisition interface are configured suitably. Therefore, HALCON allows to further parameterize the acquisition process. In HDevelop, an assistant is provided via the menu item `Assistants > Image Acquisition` that assists you when selecting your image source, adjusting the parameters, and generating suitable code.



2.2.1 Open Image Acquisition Device

When connecting to your image acquisition device with `open_framegrabber`, the main parameter is the name of the corresponding HALCON image acquisition interface. As a result, you obtain a so-called handle, with which you can access the device later, e.g., to acquire images with `grab_image` or `grab_image_async`.

With other parameters of `open_framegrabber` you can describe the configuration of image acquisition device(s) and camera(s), which is necessary when using more complex configurations, e.g., multiple cameras connected to different ports on different frame grabber boards. Further parameters allow you to specify the desired image format (size, resolution, pixel type, color space). For most of these parameters

there are default values that are used if you specify the values 'default' (string parameters) or -1 (numeric parameters).

With the operator info_framegrabber you can query information like the version number of the interface or the available boards, port numbers, and camera types.

Detailed information about the parameters of open_framegrabber can be found in the Solution Guide II-A (configuring the connection: section 3 on page 8; configuring the acquired image: section 4 on page 14).

2.2.2 Set Parameters

As described above, you already set parameters when connecting to the image acquisition device with open_framegrabber. These parameters (configuration of image_acquisition device(s) / camera(s) and image size etc.) are the so-called *general parameters*, because they are common to almost all image acquisition interfaces. However, image acquisition devices differ widely regarding the provided functionality, leading to many more *special parameters*. These parameters can be customized with the operator set_framegrabber_param.

With the operator get_framegrabber_param you can query the current values of the common and special parameters.

Detailed information about setting parameters can be found in the Solution Guide II-A in section 4 on page 14.

2.2.3 Acquire Image(s)

Actually, in a typical machine vision application you will not use the operator grab_image to acquire images, but grab_image_async. The difference between these two operators is the following: If you acquire and process images in a loop, grab_image always requests the acquisition of a new image and then blocks the program until the acquisition has finished. Then, the image is processed, and afterwards, the program waits for the next image. When using grab_image_async, in contrast, images are acquired and processed in parallel: While an image is processed, the next image is already being acquired. This, of course, leads to a significant speedup of the applications.

HALCON offers many more modes of acquiring images, e.g., triggering the acquisition by external signals or acquiring images simultaneously from multiple cameras. Detailed information about the various modes of acquiring images can be found in the Solution Guide II-A in section 5 on page 18.

2.3 Programming Examples

Example programs for all provided image acquisition interfaces can be found in the subdirectory examples\hdevelop\Image\Acquisition. Further examples are described in the Solution Guide II-A.

2.4 Selecting Operators

2.4.1 Open Image Acquisition Device

Standard:

open_framegrabber

Advanced:

info_framegrabber

2.4.2 Set Parameters

Standard:

set_framegrabber_param, get_framegrabber_param

2.4.3 Acquire Image(s)

Standard:

read_image, grab_image, grab_image_async

2.4.4 Close Image Acquisition Device

Standard:

close_framegrabber

2.5 Tips & Tricks

2.5.1 Direct Access to External Images in Memory

You can also pass externally created images, i.e., the raw image matrix in the computer's memory, to HALCON using the operators gen_image1, gen_image3, or gen_image1_extern. For an example see the Solution Guide II-A on page 31.

2.5.2 Unsupported Image Acquisition Devices

If you want to use an image acquisition device that is currently not supported by HALCON, i.e., for which no HALCON image acquisition interface exists, you can create your own interface; how to do this is described in detail in the Image Acquisition Interface Programmer's Manual.

Chapter 3

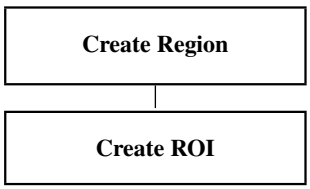
Region Of Interest

The concept of regions of interest (ROIs) is essential for machine vision in general and for HALCON in particular. The aim is to focus the processing on a specific part of the image. This approach combines region information with the image matrix: Only the image part corresponding to the region remains relevant, which reduces the number of pixels to be processed.

The advantages of using ROIs are manifold. First of all, it is a very good method to speed up a process because fewer pixels need to be processed. Furthermore, it focuses processing, e.g., a gray value feature is usually calculated only for a part of the image. Finally, ROIs are used to define templates, e.g., for matching. HALCON allows to make full use of the concept of ROIs because it enables using arbitrary shapes for the regions. This means that you are not limited to standard shapes like rectangles or polygons, but can really use *any* form - the best one to solve a given problem.

3.1 Basic Concept

Making use of ROIs is split into two simple parts: creating regions and combining them with the image.



3.1.1 Create Region

HALCON provides many ways to create regions, which can then be used as ROIs. The traditional way is to generate standard shapes like circles, ellipses, rectangles, or polygons. In addition, regions can be derived by converting them from other data types like XLD, by segmenting an image, or by user interaction.

3.1.2 Create ROI

By combining a region with an image, the region assumes the role of an ROI, i.e., it defines which part of the image must be processed. In HALCON, the ROI is also called the domain of the image. This term comes from mathematics where an image can be treated as a function that maps coordinates to gray values. An ROI reduces the domain of this function from the complete image to the relevant part. Therefore, the operator to combine regions and images is called `reduce_domain`. This simple operator fulfills the desired task in almost all applications.

3.1.3 A First Example

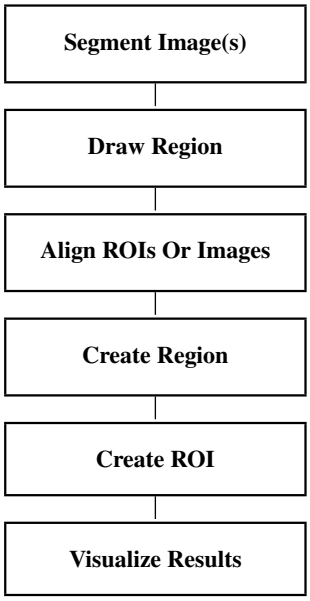
As an example for the basic concept, the following program shows all important steps to make use of an ROI. The image is acquired from file. Inside the image, only a circular part around the center should be processed. To achieve this, a circular region is generated with `gen_circle`. This region is combined with the image using `reduce_domain`. This has the effect that only the pixels of the ROI are processed when calling an operator. If, e.g., the operator `edges_sub_pix` is applied to this image, the subpixel accurate contours are extracted only inside the circle. To make this visible, some visualization operators are added to the end of the example program.

```
read_image (Image, 'mreut')
gen_circle (ROI, 256, 256, 200)
reduce_domain (Image, ROI, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
dev_display (Image)
dev_display (ROI)
dev_display (Edges)
```

Figure 3.1: Processing the image only within the circular ROI.

3.2 Extended Concept

When we take a closer look at ROIs, extra steps become important if an application needs to be more flexible.



3.2.1 Segment Image(s)

Very typical for HALCON is the creation of ROIs by a segmentation step. Instead of having a predefined ROI, the parts of the image that are relevant for further processing are extracted from the image using image processing methods. This approach is possible because ROIs are nothing else but normal HALCON regions, and therefore share all their advantages like efficient processing and arbitrary shapes (see Quick Guide in section 2.1.2.2 on page 17 for more information on HALCON regions). The segmentation of regions used for ROIs follows the same approach as standard blob analysis. For more details, please refer to the description of this step on page 34.

3.2.2 Draw Region

The standard way to specify ROIs is to draw the shape interactively using the mouse. To make this easy, HALCON provides special operators for standard shapes and free-form shapes. All operators for this kind of action start with the prefix `draw_`. The drawing is performed by making use of the left mouse button (drawing, picking, and dragging) and finished by clicking the right mouse button. For each such draw-operator HALCON provides operators to generate regions by using the returned parameters (see the description of the step Create Region on page 19). Operators for mouse interaction can be found in the reference manual in the chapter "Graphics ▷ Drawing". More information on user interaction can be also found in the chapter Visualization on page 269.

3.2.3 Align ROIs Or Images

Sometimes the coordinates of an ROI depend on the position of another object in the image. If the object moves, the ROI must be moved (aligned) accordingly. This is achieved by first locating the object using template matching. Based on the determined position and the orientation, the coordinates of the ROIs are then transformed.

How to perform alignment using shape-based matching is described in the Solution Guide II-B in section 4.3.4 on page 37.

3.2.4 Create Region

The standard way is to generate regions based on the coordinates and dimensions returned by a user interaction or by coordinate values stored in a file. In this case, operators like `gen_circle`, `gen_rectangle2`, or `gen_region_polygon_filled` are used. More advanced are special shapes used to guide a preprocessing step to save execution time. Typical examples for this are grids of lines or dots or checker boards. With these shapes, the images can be covered in a systematic way and checked for specific object occurrences. If you want to segment, e.g., blobs of a given minimum size it is sufficient to use in a first step a search grid that is finer than the minimum object size to locate fragments. In a second step these fragments are dilated (`dilation_rectangle1`) and the segmentation method is called once again, now within this enlarged area. If the objects cover only a relatively small area of the image this approach can speed up the process significantly.

3.2.5 Create ROI

This step combines the region and the image to make use of the region as the domain of the image. The standard method that is recommended to be used is `reduce_domain`. It has the advantage of being safe and having a simple semantics. `rectangle1_domain` is a shortcut for generating rectangular ROIs (instead of calling `gen_rectangle1` and `reduce_domain` in sequence). For advanced applications `change_domain` can be used as a slightly faster version than `reduce_domain`. This operator does not perform an intersection with the existing domain and does not check if the region is outside the image - which will cause a system crash when applying an operator to the data afterwards if the region lies partly outside the image. If the programmer ensures that the input region is well defined, this is a way to save (a little) execution time.

3.2.6 Visualize Results

Finally, you might want to display the ROIs or the reduced images. With the operator `get_domain`, the region currently used by the image can be accessed and displayed (and processed) like any other region. When displaying an image, e.g., with `disp_image`, only the defined pixels are displayed. Pixels in the graphics window outside the domain of the image will not be modified.

For detailed information see the description of this method on page 269.

3.3 Programming Examples

This section gives a brief introduction to programming ROIs in HALCON. Two examples show the principles of region generation, combining these with images, and then processing the data.

3.3.1 Processing inside a User Defined Region

Example: examples\solution_guide\basics\critical_points.dev

Figure 3.2 shows an image with marks that are used for a camera calibration in a 3D application. Here, we assume that the marks must be extracted in a given part of the image only.

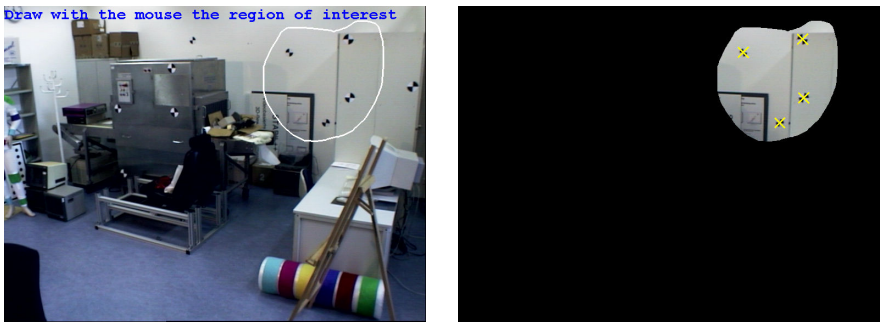


Figure 3.2: (a) Original image with drawn ROI; (b) reduced image with extracted points.

To achieve this, the user draws a region of interest with the mouse. The corresponding operator is draw_region. It has the window handle returned by dev_open_window as input and returns a region when the right mouse button is pressed. The operator reduce_domain combines this region with the image.

```
draw_region (Region, WindowHandle)
reduce_domain (Image, Region, ImageReduced)
```

When calling the point extraction operator critical_points_sub_pix on this reduced image, only points inside the ROI are found. The final part of the program shows how to display these points overlaid on the image.

```
critical_points_sub_pix (ImageReduced, 'facet', 1.5, 8, _, _, _, _,
                         RowSaddle, ColSaddle)
dev_clear_window ()
dev_display (ImageReduced)
dev_set_color ('yellow')
for i := 0 to |RowSaddle|-1 by 1
  gen_cross_contour_xld (Cross, RowSaddle[i], ColSaddle[i], 25, 0.785398)
  dev_display (Cross)
endfor
```

3.3.2 Interactive Partial Filtering of an Image

Example: examples\solution_guide\basics\median_interactive.dev

The task is to filter an image with a median filter only at the points where the user clicks with the mouse into the image, i.e., in the graphics window displaying the image.

Figure 3.3: Partially filtered image.

To do this, a loop is used inside which the mouse position is continuously requested with get_mposition. Because this operator throws an exception if the mouse is outside the graphics window the call is protected with dev_set_check.

```
Button := 0
while (Button # 4)
  Row := -1
  Column := -1
  dev_set_check ('~give_error')
  get_mposition (WindowHandle, Row, Column, Button)
  dev_set_check ('give_error')
```

If the mouse is over the window, a circular region is displayed, which shows where the filter would be applied.

```
  if (Row >= 0 and Column >= 0)
    gen_circle (Circle, Row, Column, 20)
    boundary (Circle, RegionBorder, 'inner')
    dev_display (RegionBorder)
```

If the left mouse button is pressed, median_image must be applied in the local neighborhood of the current mouse position. This is done by generating a circle with gen_circle and then calling reduce_domain.

```
    if (Button = 1)
      reduce_domain (Image, Circle, ImageReduced)
```

Now, the filter is called with this reduced image and the result is painted back into the input image for possible repetitive filtering. The loop will be terminated when the right mouse button is clicked.

```
      median_image (ImageReduced, ImageMedian, 'circle', 5,
                    'mirrored')
      overpaint_gray (Image, ImageMedian)
    endif
```

3.3.3 Inspecting the Contours of a Tool

Example: examples\hdevelop\Applications\FA\circles.dev

The task of this example is to inspect the contours of the tool depicted in figure 3.4.

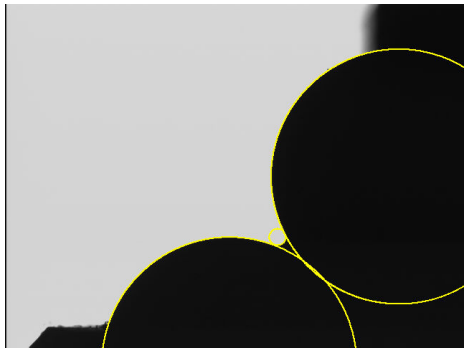


Figure 3.4: Fitting circles to the contours of the tool.

Because the subpixel-precise contour extraction is time-consuming, in a first step an ROI is created via a standard blob analysis (see Blob Analysis on page 33): With a threshold operator the object to be measured is extracted. This region is converted to its boundary, omitting the pixels at the image border.

```
fast_threshold (Image, Region, 0, 120, 7)
boundary (Region, RegionBorder, 'inner')
clip_region_rel (RegionBorder, RegionClipped, 5, 5, 5, 5)
```

The result is a small region close to the edge of the object. The boundary of the region, i.e., the edge, is dilated to serve as the ROI for the edge extraction. Now, the subpixel-precise edge extractor is called and the contour is segmented into straight lines and circular arcs.

```
dilation_circle (RegionClipped, RegionDilation, 2.5)
reduce_domain (Image, RegionDilation, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 40, 60)
segment_contours_xld (Edges, ContoursSplit, 'lines_circles', 5, 4, 3)
```

For the segments that represent arcs, the corresponding circle parameters are determined. For inspection purposes, circles with the same parameters are generated and overlaid on the image (see also Contour Processing on page 97).

```
get_contour_global_attrib_xld (ObjectSelected, 'cont_approx', Attrib)
if (Attrib > 0)
  fit_circle_contour_xld (ObjectSelected, 'ahuber', -1, 2, 0, 3, 2,
                          Row, Column, Radius, StartPhi, EndPhi,
                          PointOrder)
  gen_ellipse_contour_xld (ContEllipse, Row, Column, 0, Radius,
                           Radius, 0, rad(360), 'positive', 1.0)
  dev_display (ContEllipse)
endif
```

3.3.4 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\forest.dev
  Extract trees and meadows from forest
  → description here in the Solution Guide Basics on page 40
- examples\hdevelop\Applications\Aerial\roads.dev
  Extract roads from aerial image
  → description here in the Solution Guide Basics on page 106
- examples\hdevelop\Applications\Barcode\get_rectangle_pose_barcode.dev
  Estimate 3D pose of bar codes
- examples\hdevelop\Applications\Calibration\3d-coordinates.dev
  Measure slanted object in world coordinates
- examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev
  Determine the pose of circles in 3D from their perspective 2D projections
- examples\hdevelop\Applications\Calibration\3d_position_of_rectangle.dev
  Estimate 3D pose of rectangular objects
- examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev
  Measure distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- examples\hdevelop\Applications\FA\ball.dev
  Inspect ball bondings
  → description here in the Solution Guide Basics on page 43

- examples\hdevelop\Applications\FA\ball_seq.dev
  Inspect ball bondings (multiple images)
- examples\hdevelop\Applications\FA\find_pads.dev
  Find pads on a die.
- examples\hdevelop\Applications\FA\holes.dev
  Extract positions and radii of holes
- examples\hdevelop\Applications\FA\ic.dev
  Extract resistors, capacitors and ICs from a board using color information
  → description here in the Solution Guide Basics on page 202
- examples\hdevelop\Applications\FA\matching_multi_channel_yogurt.dev
  Find yogurts of different flavors using color shape-based matching
- examples\hdevelop\Applications\FA\print_check.dev
  Perform a typical print quality inspection using a variation model
  → description here in the Solution Guide Basics on page 161
- examples\hdevelop\Applications\FA\print_check_single_chars.dev
  Perform a typical print quality inspection using variation models for each character
- examples\hdevelop\Applications\FA\rim.dev
  Inspect holes and characters on a rim
- examples\hdevelop\Applications\Medicine\particle.dev
  Extract particles of varying sizes
  → description here in the Solution Guide Basics on page 39
- examples\hdevelop\Applications\OCR\ocrcolor.dev
  Read numbers using color information
  → description here in the Solution Guide Basics on page 245
- examples\hdevelop\Applications\OCR\ocrcolort.dev
  Segment numbers using color information and train the OCR
- examples\hdevelop\Applications\OCV\adaption_ocv.dev
  Analyze the impact of illumination changes on reported character quality
- examples\hdevelop\Applications\OCV\print_quality.dev
  Inspect the quality of the letter A in different images
- examples\hdevelop\Applications\Sequences\autobahn.dev
  Detect lane markers fast
- examples\hdevelop\Filter\Geometric-Transformations\
  projective_trans_image_reduced.dev
  Apply projective transformations to an image and its domain
- examples\hdevelop\OCR\Neural-Nets\label_word_process_mlp.dev
  Read a best-before label using a lexicon to improve results
- examples\hdevelop\Regions\Creation\gen_grid_region.dev
  Create a grid region consisting of lines or points

- examples\hdevelop\Segmentation\Classification\novelty_detection_gmm.dev
  Inspect a web using texture classification with gaussian mixture models
- examples\hdevelop\Segmentation\Classification\novelty_detection_svm.dev
  Inspect a web using texture classification with support vector machine classifiers
- examples\hdevelop\Segmentation\Threshold\dual_threshold.dev
  Segment signed images into positive and negative regions
- examples\hdevelop\Tools\2D-Transformations\vector_angle_to_rigid.dev
  Match a pattern and display the normalized image
- examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev
  Rectify image of stadium to simulate overhead view
- examples\hdevelop\Tools\Grid-Rectification\grid_rectification.dev
  Rectify an arbitrarily distorted image using a regular grid
- examples\hdevelop\Tools\Hough\hough_lines.dev
  Detect lines in an image using the Hough transform
- examples\hdevelop\Tools\Hough\hough_lines_dir.dev
  Detect lines in an image using the Hough transform and local gradient directions
- examples\hdevelop\Tools\Stereo\binocular_disparity_segmentation.dev
  Demonstrate stereo results using an artificial image pair
- examples\hdevelop\XLD\Features\fit_ellipse_contour_xld.dev
  Approximate XLD contours with ellipses or elliptic arcs
- examples\hdevelop\XLD\Features\fit_rectangle2_contour_xld.dev
  Detect errors in punched rectangular holes.
- examples\hdevelop\XLD\Features\statistics_points_xld.dev
  Analyze 2D statistical data using *_points_xld operators
- examples\solution_guide\2d_measuring\measure_chip.dev
  Get extents and positions of rectangular chip components
  → description in the Solution Guide II-E on page 49
- examples\solution_guide\2d_measuring\measure_circles.dev
  Get radii and positions of circular shapes
  → description in the Solution Guide II-E on page 52
- examples\solution_guide\2d_measuring\measure_grid.dev
  Get junctions of a grid that separates keys
  → description in the Solution Guide II-E on page 42
- examples\solution_guide\2d_measuring\measure_metal_part_extended.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 39
- examples\solution_guide\2d_measuring\measure_metal_part_first_example.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 6

- examples\solution_guide\2d_measuring\measure_metal_part_id.dev
  Inspect metal part for missing or deviating circular shapes
  → description in the Solution Guide II-E on page 64
- examples\solution_guide\2d_measuring\measure_perspective_scratch.dev
  Measure the length of a scratch in world coordinates
  → description in the Solution Guide II-E on page 71
- examples\solution_guide\2d_measuring\measure_pump.dev
  Measure positions and radii of circular shapes
  → description in the Solution Guide II-E on page 54
- examples\solution_guide\2d_measuring\measure_screw.dev
  Measure several features of a screw
  → description in the Solution Guide II-E on page 31
- examples\solution_guide\3d_machine_vision\3d_information_for_selected_points.dev
  Calculate world coordinates for a point in a stereo image pair
  → description in the Solution Guide II-F on page 109
- examples\solution_guide\3d_machine_vision\camera_calibration_exterior.dev
  Measure positions on a caliper rule using camera calibration
  → description in the Solution Guide II-F on page 41
- examples\solution_guide\3d_machine_vision\grid_rectification_ruled_surface.dev
  Rectify an arbitrarily distorted image using a regular grid
  → description in the Solution Guide II-F on page 130
- examples\solution_guide\shape_matching\align_measurements.dev
  Inspect individual razor blades using shape-based matching to align ROIs for the measure tool
  → description in the Solution Guide II-B on page 37
- examples\solution_guide\shape_matching\first_example_shape_matching.dev
  Introduce HALCON's shape-based matching
  → description in the Solution Guide II-B on page 5
- examples\solution_guide\shape_matching\multiple_models.dev
  Search for two types of objects simultaneously
  → description in the Solution Guide II-B on page 26
- examples\solution_guide\shape_matching\multiple_objects.dev
  Search for multiple instances of a security ring
  → description in the Solution Guide II-B on page 35
- examples\solution_guide\shape_matching\multiple_scales.dev
  Search for nuts of different sizes
  → description in the Solution Guide II-B on page 44
- examples\solution_guide\shape_matching\process_shape_model.dev
  Create a model ROI by modifying the result of inspect_shape_model
  → description in the Solution Guide II-B on page 11
- examples\solution_guide\shape_matching\reuse_model.dev

Store and reuse a shape model
→ description in the Solution Guide II-B on page 46

C++

- `examples\cpp\source\example2.cpp`
  Accessing image data and applying various tasks
- `examples\cpp\source\example3.cpp`
  Iterating over a set of image pixels
- `examples\cpp\source\pen.cpp`
  Inspect the quality of print on a pen using the variation model of HALCON

3.4 Selecting Operators

3.4.1 Segment Image(s)

Please refer to the detailed operator list for the step Segment Image(s) on page 52.

3.4.2 Draw Region

Standard:

`draw_circle`, `draw_rectangle1`, `draw_rectangle2`, `draw_region`

Advanced:

`draw_circle_mod`, `draw_rectangle1_mod`, `draw_rectangle2_mod`, `draw_xld`, `draw_xld_mod`

3.4.3 Align ROIs Or Images

Operators for aligning ROIs or images are described in the Solution Guide II-B.

3.4.4 Create Region

Standard:

`gen_circle`, `gen_ellipse`, `gen_rectangle2`, `gen_region_polygon_filled`

Advanced:

`gen_checker_region`, `gen_grid_region`

More operators to generate regions can be found in the reference manual in chapter "Regions ▷ Creation".

3.4.5 Create ROI

Standard:

reduce_domain, rectangle1_domain

Advanced:

change_domain, full_domain, add_channels

3.4.6 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

3.5 Relation to Other Methods

One class of operators does not follow the standard rules for ROI handling: the operators of the measure tool (see the description of this method on page 57). Here, the ROI is defined during the creation of a tool (gen_measure_arc and gen_measure_rectangle2) by specifying the coordinates as numeric values. The domain defined for the image will be ignored in this case.

3.6 Tips & Tricks

3.6.1 Modifying ROI Shapes

Sometimes the shape of a given ROI, either generated from the program or defined by the user, does not fulfill the requirements. Here, HALCON provides many operators to modify the shape to adapt it accordingly. Often used operators are, e.g., fill_up to fill holes inside the region, shape_trans to apply a general transformation like the convex hull or the smallest rectangle, or morphological operators like erosion_circle to make the region smaller or closing_circle to fill gaps. More operators like these can be found in the Reference Manual in the chapters "Morphology ▷ Region" and "Regions ▷ Transformation".

3.6.2 Storing ROI shapes

If an ROI is used multiple times it is useful to save the region to file and load it at the beginning of the application. Storing to file is done using write_region, loading with read_region.

3.6.3 Speed Up

ROIs are a perfect way to save execution time: The smaller the ROI, the faster the application. This can be used as a general rule. If we consider this in more detail, we also need to think about the shape of ROIs. Because ROIs are based on the HALCON regions they use runlength encoding (see Quick Guide in section 2.1.2.2 on page 17. for more information on HALCON regions). This type of encoding is perfect if the runs are long. Therefore, a horizontal line can be both stored and processed more efficiently than a vertical line. This holds as well for the processing time of ROIs. Obviously this type of overhead is very small and can only be of importance with very fast operators like `threshold`.

3.7 Advanced Topics

3.7.1 Filter masks and ROIs

If a filter is applied with a reduced domain, the result along the ROI boundary might be surprising because gray values lying outside the boundary are used as input for the filter process. To understand this, you must consider the definition of domains in this context: A domain defines for a filter for which input pixels output pixels must be calculated. But pixels outside the domain (which lie within the image matrix) can be used for processing. If this behavior is not desired, the operator `expand_domain_gray` can be used to propagate the border gray values outward. The result looks as if the boundary is copied multiple times with larger and larger distances.

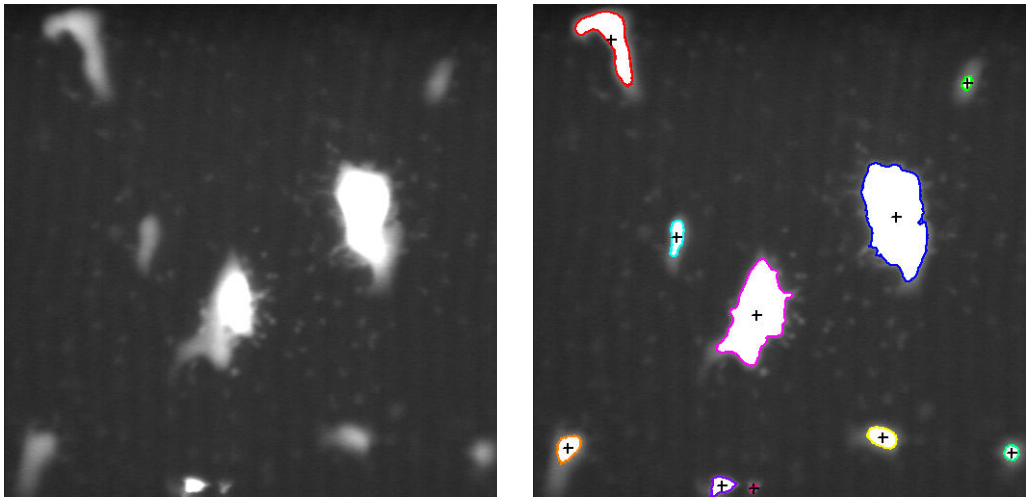
3.7.2 Binary Images

In some applications, it might be necessary to use ROIs that are available as binary images. To convert these to HALCON regions, you must use `gen_image1` to convert them into a HALCON image, followed by `threshold` to generate the region. The conversion back can easily be achieved using `region_to_bin` followed by `get_image_pointer1`. It is also possible to import binary image files using `read_region`.

Chapter 4

Blob Analysis

The idea of blob analysis is quite easy: In an image the pixels of the relevant objects (also called foreground) can be identified by their gray value. For example, figure 4.1 shows tissue particles in a liquid. These particles are bright and the liquid (background) is dark. By selecting bright pixels (thresholding) the particles can be detected easily. In many applications this simple condition of dark and bright pixels no longer holds, but the same results can be achieved with extra pre-processing or alternative methods for pixel selection / grouping.



a) b)

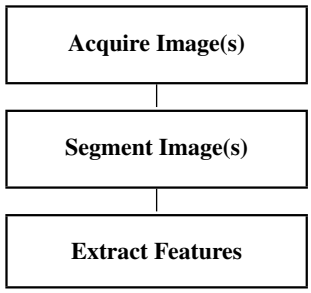
Figure 4.1: Basic idea of blob analysis: (a) original image, (b) extracted blobs with calculated center points.

The advantage of blob analysis is the extreme flexibility that comes from the huge number of operators that HALCON offers in this context. Furthermore, these methods typically have a very high performance.

Methods known from blob analysis can also be combined with many other vision tasks, e.g., as a pre-processing step for a flexible generation of regions of interest.

4.1 Basic Concept

Blob analysis mainly consists of three parts:



4.1.1 Acquire Image(s)

First, an image is acquired.

For detailed information see the description of this method on page 13.

4.1.2 Segment Image(s)

Having acquired the image, the task is to select the foreground pixels. This is also called segmentation. The result of this process typically is referred to as blobs (binary large objects). In HALCON, the data type is called a region.

4.1.3 Extract Features

In the final step, features like the area (i.e., the number of pixels), the center of gravity, or the orientation are calculated.

4.1.4 A First Example

An example for this basic concept is the following program, which belongs to the example explained above. Here, the image is acquired from file. All pixels that are brighter than 120 are selected using `threshold`. Then, an extra step is introduced which is not so obvious: The operator `connection` separates the set of all bright pixels into so called connected components. The effect of this step is that we now have multiple regions instead of the single region that is returned by `threshold`. The last step

of this program is the calculation of some features. Here, the operator area_center determines the size (number of pixels) and the center of gravity. Please note that area_center returns multiple values for all three feature parameters (one value for each connected component).

```
read_image (Image, 'particle')
threshold (Image, BrightPixels, 120, 255)
connection (BrightPixels, Particles)
area_center (Particles, Area, Row, Column)
```

4.2 Extended Concept

In many cases the segmentation of blobs will be more advanced than in the above example. Reasons for this are, e.g., clutter or inhomogeneous illumination. Furthermore, postprocessing like transforming the features to real world units or visualization of results are often required.

4.2.1 Align ROIs Or Images

In some applications, the regions of interest must be aligned relative to another object. Alternatively, the image itself can be aligned, e.g., by rotating or cropping it.

How to perform alignment using shape-based matching is described in the Solution Guide II-B in section 4.3.4 on page 37.

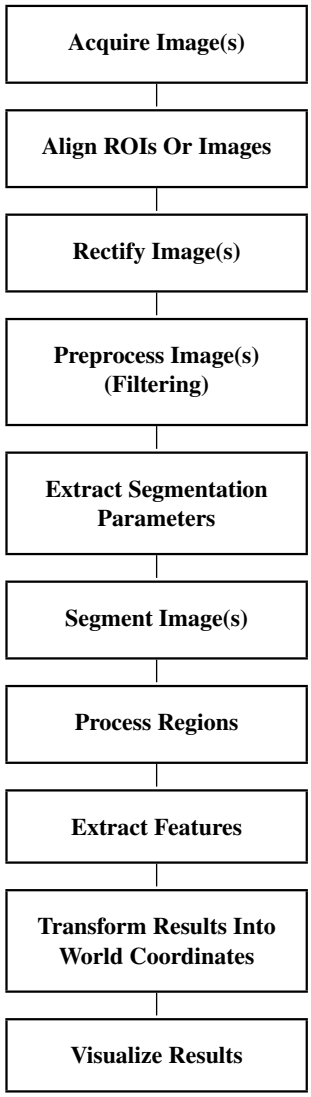
4.2.2 Rectify Image(s)

Similarly to alignment, it may be necessary to rectify the image, e.g., to remove lens distortions or to transform the image into a reference point of view.

Detailed information about rectifying images can be found in the Solution Guide II-F in section 3.3 on page 49.

4.2.3 Preprocess Image(s) (Filtering)

The next important part is the pre-processing of the image. Here, operators like mean_image or gauss_image can be used to eliminate noise. A fast but less perfect alternative to gauss_image is binomial_filter. The operator median_image is useful for suppressing small spots or thin lines. The operator anisotrope_diff is useful for edge-preserving smoothing, and finally fill_interlace is used to eliminate defects caused by interlaced cameras.



4.2.4 Extract Segmentation Parameters

Instead of using fixed threshold values, they can be extracted dynamically for each image. One example for this is a gray value histogram that has multiple peaks, one for each object class. Here, you can use the operators `gray_histo_abs` and `histo_to_thresh`.

As an advanced alternative, you can use the operator `intensity` in combination with a reference image that contains only background: During setup, you determine the mean gray value of a background region. During the inspection, you again determine this mean gray value. If it has changed, you adapt the threshold accordingly.

4.2.5 Segment Image(s)

For the segmentation various methods can be used. The most simple method is `threshold`, where one or more gray value ranges that belong to the foreground objects are specified. Another very common method is `dyn_threshold`. Here, a second image is passed as a reference. With this approach, a local threshold instead of a global threshold is used. These local threshold values are stored in the reference image. The reference image can either be static by taking a picture of the empty background or can be determined dynamically with smoothing filters like `mean_image`.

4.2.6 Process Regions

Once blob regions are segmented, it is often necessary to modify them, e.g., by suppressing small areas, regions of a given orientation, or regions that are close to other regions. In this context, the morphological operators `opening_circle` and `opening_rectangle1` are often used to suppress noise and `closing_circle` and `closing_rectangle1` to fill gaps.

Blobs with a specific feature can be selected with `select_shape`, `select_shape_std`, and `select_shape_proto`.

4.2.7 Extract Features

To finalize the image processing, features of the blobs are extracted. The type of features needed depends on the application. A full list can be found in the Reference Manual in the chapters "Regions ▷ Features" and "Image ▷ Features".

4.2.8 Transform Results Into World Coordinates

Features like the area or the center of gravity often must be converted to world coordinates. This can be achieved with the HALCON camera calibration.

How to transform results into world coordinates is described in detail in the Solution Guide II-F in section 3.2 on page 44.

4.2.9 Visualize Results

Finally, you might want to display the images, the blob (regions), and the features.

For detailed information see the description of this method on page 269.

4.3 Programming Examples

This section gives a brief introduction to using HALCON for blob analysis.

4.3.1 Crystals

Example: examples\solution_guide\basics\crystal.dev

Figure 4.2a shows an image taken in the upper atmosphere with collected samples of crystals. The task is to analyze the objects to determine the frequency of specific shapes. One of the important objects are the hexagonally shaped crystals.

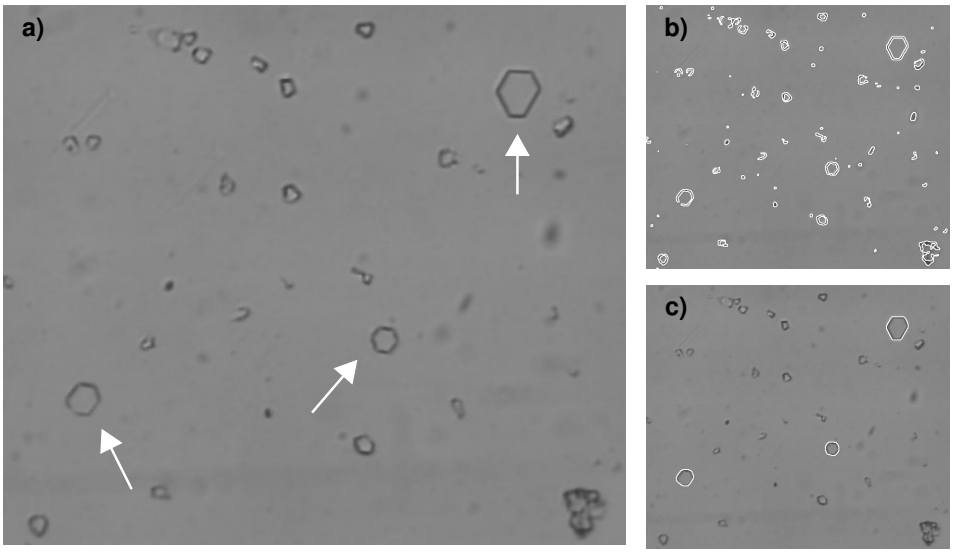


Figure 4.2: Extracting hexagonal crystals: (a) original image with arrows marking the crystals to be extracted, (b) result of the initial segmentation, (c) finally selected blobs.

First, the image is read from file with read_image. The segmentation of objects is performed with a local threshold because of the relatively low contrast of the crystals combined with a non-homogeneous background. The background is determined with the average filter mean_image. The filter mask size is selected such that it has about three times the width of the dark areas. dyn_threshold now compares the smoothed with the original gray values, selecting those pixels that are darker by a contrast of 8 gray values. connection separates the objects into connected components. Figure 4.2b shows the result of this initial segmentation.

```
read_image (Image, 'crystal')
mean_image (Image, ImageMean, 21, 21)
dyn_threshold (Image, ImageMean, RegionDynThresh, 8, 'dark')
connection (RegionDynThresh, ConnectedRegions)
```

In the following processing step, the task now is to select only the hexagonally shaped crystals. For this, they are first transformed into their convex hull. This is like putting a rubber band around each region. From these regions, those that are big (select_shape) and have a given gray value distribution (select_gray) are selected. The parameters for the selection are determined so that only the relevant crystals remain (see figure 4.2c).

```
shape_trans (ConnectedRegions, ConvexRegions, 'convex')
select_shape (ConvexRegions, LargeRegions, 'area', 'and', 600, 2000)
select_gray (LargeRegions, Image, Crystals, 'entropy', 'and', 1, 5.6)
```

4.3.2 Atoms

Example: examples\solution_guide\basics\atoms.dev

Specialized microscopes are able to determine the rough location of single atoms. This is useful, e.g., to analyse the grid change of crystals at a p-n-junction. A segmentation that works perfectly well on images like these is the watershed method. Here, each dark basin is returned as a single region. Because at the outer part of the image atoms are only partially visible, the first task is to extract only those that are not close to the image border. Finally, the irregularity is extracted. This is done by looking for those atoms that have an abnormal (squeezed) shape (see figure 4.3).

```
gauss_image (Image, ImageGauss, 5)
watersheds (ImageGauss, Basins, Watersheds)
select_shape (Basins, SelectedRegions1, 'column1', 'and', 2, Width-1)
select_shape (SelectedRegions1, SelectedRegions2, 'row1', 'and', 2,
              Height-1)
select_shape (SelectedRegions2, SelectedRegions3, 'column2', 'and', 1,
              Width-3)
select_shape (SelectedRegions3, Inner, 'row2', 'and', 1, Height-3)
select_shape (Inner, Irregular, 'compactness', 'and', 1.45, 3)
```

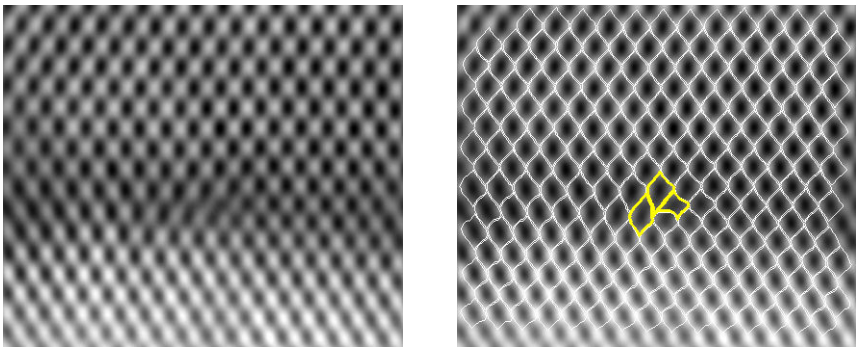
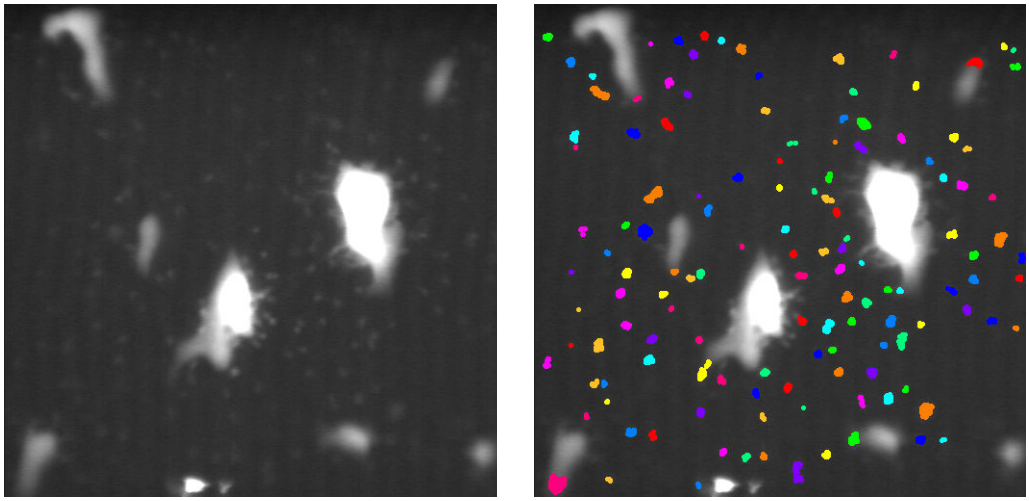


Figure 4.3: Inspecting atom structure.

4.3.3 Analyzing Particles

Example: examples\hdevelop\Applications\Medicine\particle.dev



a) b)

Figure 4.4: Extracting the small particles: (a) original image, (b) result.

The task of this example is to analyze particles in a liquid. The main difficulty in this application is the presence of two types of objects: big bright objects and small objects with low contrast. In addition, the presence of noise complicates the segmentation.

The program segments the two classes of objects separately using two different methods: global and local thresholding. With additional post-processing, the small particles can be extracted in a robust manner.

```
threshold (Image, Large, 110, 255)
dilation_circle (Large, LargeDilation, 7.5)
complement (LargeDilation, NotLarge)
reduce_domain (Image, NotLarge, ParticlesRed)

mean_image (ParticlesRed, Mean, 31, 31)
dyn_threshold (ParticlesRed, Mean, SmallRaw, 3, 'light')
opening_circle (SmallRaw, Small, 2.5)
connection (Small, SmallConnection)
```

4.3.4 Extracting Forest Features from Color Infrared Image

Example: examples\hdevelop\Applications\Aerial\forest.dev

The task of this example is to detect different object classes in the color infrared image depicted in figure 4.5: trees (coniferous and deciduous), meadows, and roads.

The image data is a color infrared image, which allows to extract roads very easily because of their specific color. For that, the multi-channel image is split into its three channels and each channel is

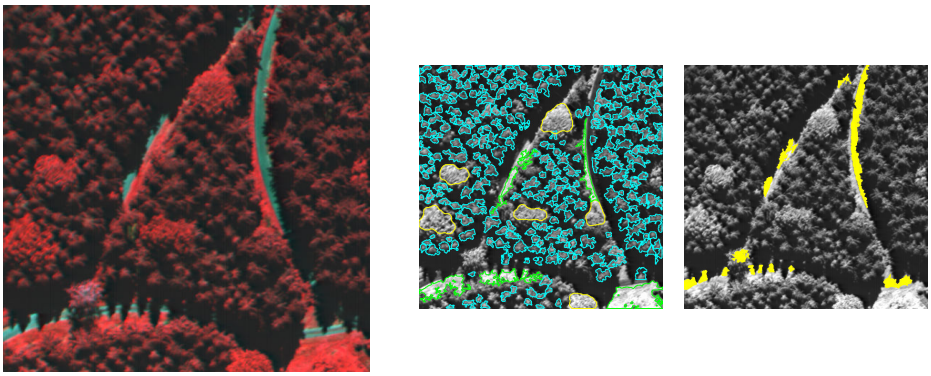


Figure 4.5: (a) Original image; (b) extracted trees and meadows; (c) extracted roads.

investigated individually. Further information about decomposing multi-channel images can be found in Color Processing on page 191.

```
read_image (Forest, 'forest_air1')
decompose3 (Forest, Red, Green, Blue)
threshold (Blue, BlueBright, 80, 255)
connection (BlueBright, BlueBrightConnection)
select_shape (BlueBrightConnection, Path, 'area', 'and', 100, 100000000)
```

Beech trees are segmented in the red channel based on their intensity and minimum size.

```
threshold (Red, RedBright, 120, 255)
connection (RedBright, RedBrightConnection)
select_shape (RedBrightConnection, RedBrightBig, 'area', 'and', 1500,
              10000000)
closing_circle (RedBrightBig, RedBrightClosing, 7.5)
opening_circle (RedBrightClosing, RedBrightOpening, 9.5)
connection (RedBrightOpening, RedBrightOpeningConnection)
select_shape (RedBrightOpeningConnection, BeechBig, 'area', 'and', 1000,
              100000000)
select_gray (BeechBig, Blue, Beech, 'mean', 'and', 0, 59)
```

Meadows have similar spectral properties, but are slightly brighter.

```
union1 (Beech, BeechUnion)
complement (BeechUnion, NotBeech)
difference (NotBeech, Path, NotBeechNotPath)
reduce_domain (Red, NotBeechNotPath, NotBeechNotPathRed)
threshold (NotBeechNotPathRed, BrightRest, 150, 255)
connection (BrightRest, BrightRestConnection)
select_shape (BrightRestConnection, Meadow, 'area', 'and', 500, 1000000)
```

The coniferous trees are extracted using the watershed approach with an additional thresholding inside the basins to get rid of the shadow areas.

```
union2 (Path, RedBrightClosing, BeechPath)
smooth_image (Red, RedGauss, 'gauss', 4.0)
invert_image (RedGauss, Invert)
watersheds (Invert, SpruceRed, Watersheds)
select_shape (SpruceRed, SpruceRedLarge, 'area', 'and', 100, 5000)
select_gray (SpruceRedLarge, Red, SpruceRedInitial, 'max', 'and', 100, 200)
gen_empty_obj (LocalThresh)
count_obj (SpruceRedInitial, NumSpruce)
for i := 1 to NumSpruce by 1
  select_obj (SpruceRedInitial, SingleSpruce, i)
  min_max_gray (SingleSpruce, Red, 50, Min, Max, Range)
  reduce_domain (Red, SingleSpruce, SingleSpruceRed)
  threshold (SingleSpruceRed, SingleSpruceBright, Min, 255)
  connection (SingleSpruceBright, SingleSpruceBrightCon)
  select_shape_std (SingleSpruceBrightCon, MaxAreaSpruce, 'max_area', 70)
  concat_obj (MaxAreaSpruce, LocalThresh, LocalThresh)
endfor
opening_circle (LocalThresh, FinalSpruce, 1.5)
```

4.3.5 Checking a Boundary for Fins

Example: examples\hdevelop\Applications\FA\fin.dev

The task of this example is to check the outer boundary of a plastic part. In this case, some objects show fins that are not allowed for faultless pieces (see figure 4.6).

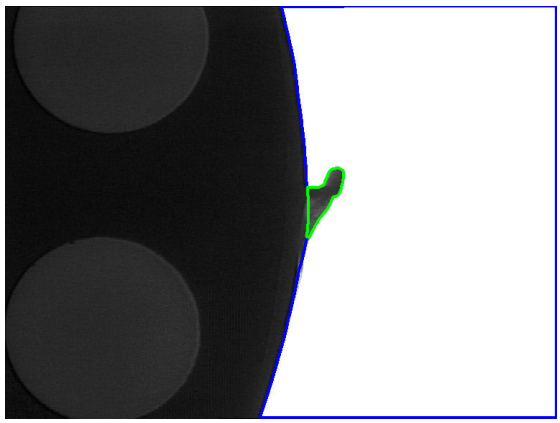


Figure 4.6: Boundary with extracted fin.

The program first extracts the plastic part and then forms the complement to extract the background region (in which the fin appears as an indentation).

```
bin_threshold (Fin, Dark)
difference (Fin, Dark, Background)
```

This indentation in the background region is then closed using a morphological operator.

```
closing_circle (Background, ClosedBackground, 250)
```

Significant differences between the closed region and the original region are production errors.

```
difference (ClosedBackground, Background, RegionDifference)
opening_rectangle1 (RegionDifference, FinRegion, 5, 5)
```

4.3.6 Bonding Balls

Example: examples\hdevelop\Applications\FA\ball.dev

The task of this example is to inspect the diameter of the ball bonds depicted in figure 4.7.

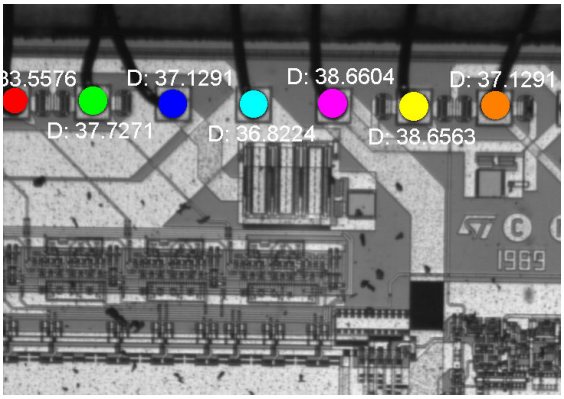


Figure 4.7: Measuring the diameter of ball bonds.

The extraction of the ball bonds is a two step approach: First, the die is located by segmenting bright areas and transforming them into their smallest surrounding rectangle.

```
threshold (Bond, Bright, 100, 255)
shape_trans (Bright, Die, 'rectangle2')
```

Now, the processing is focused to the region inside the die using reduce_domain. In this ROI, the program checks for dark areas that correspond to wire material.

```
reduce_domain (Bond, Die, DieGrey)
threshold (DieGrey, Wires, 0, 50)
fill_up_shape (Wires, WiresFilled, 'area', 1, 100)
```

After removing irrelevant structures and arranging the bonds in a predefined order, the desired features are extracted.

```
opening_circle (WiresFilled, Balls, 15.5)
connection (Balls, SingleBalls)
select_shape (SingleBalls, IntermediateBalls, 'circularity', 'and', 0.85,
              1.0)
sort_region (IntermediateBalls, FinalBalls, 'first_point', 'true',
             'column')
smallest_circle (FinalBalls, Row, Column, Radius)
```

4.3.7 Surface Scratches

Example: examples\solution_guide\basics\surface_scratch.dev

This example detects scratches on a metal surface (see figure 4.8).

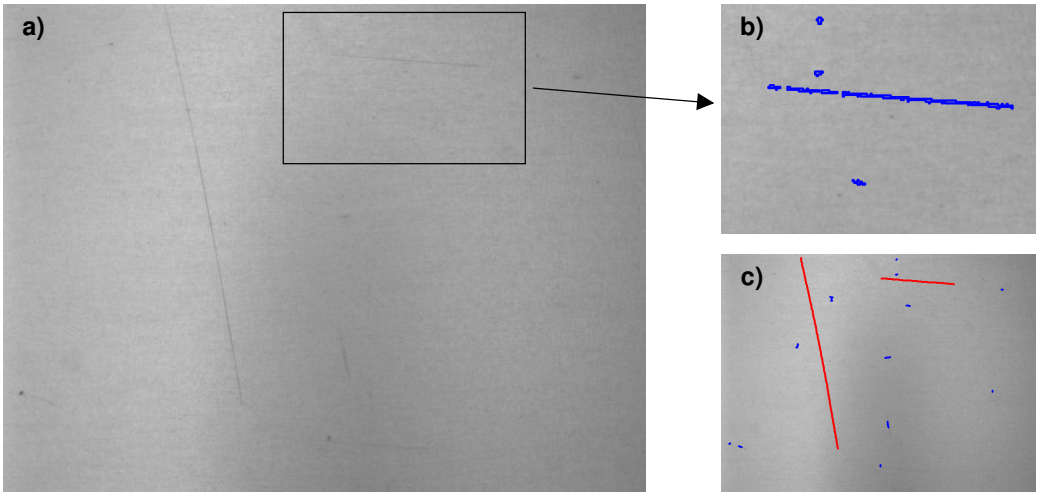


Figure 4.8: Detecting scratches on a metal surface: (a) original image, (b) extracted scratches still partly fractioned, (c) final result with merged scratches.

The main difficulties for the segmentation are the inhomogenous background and the fact that the scratches are thin structures. Both problems can be solved using a local threshold, i.e., the operators mean_image and dyn_threshold. After connection, the small objects that are mainly noise are removed (see figure 4.8b).

```
mean_image (Image, ImageMean, 7, 7)
dyn_threshold (Image, ImageMean, DarkPixels, 5, 'dark')
connection (DarkPixels, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 10, 1000)
```

The scratches are part of the selection, but if we look closely we see that they are partially fractioned. To solve this problem we combine all fractions again into one big region. By applying dilation_circle, neighboring parts with a given maximum distance are now combined. To finally get the correct shape of the scratches - which are now too wide because of the dilation - skeleton is used to thin the shape to a width of one pixel.

```
union1 (SelectedRegions, RegionUnion)
dilation_circle (RegionUnion, RegionDilation, 3.5)
skeleton (RegionDilation, Skeleton)
connection (Skeleton, Errors)
```

The last step is to distinguish between small dots and scratches on the surface. This is achieved with select_shape, using the size as feature. Figure 4.8c depicts the result.

```
select_shape (Errors, Scratches, 'area', 'and', 50, 10000)
select_shape (Errors, Dots, 'area', 'and', 1, 50)
```

4.3.8 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\dem_trees.dev
  Extract trees using texture and a digital elevation model
- examples\hdevelop\Applications\Aerial\high.dev
  Extract high objects
- examples\hdevelop\Applications\Barcode\get_rectangle_pose_barcode.dev
  Estimate 3D pose of bar codes
- examples\hdevelop\Applications\Calibration\3d-coordinates.dev
  Measure slanted object in world coordinates
- examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev
  Determine the pose of circles in 3D from their perspective 2D projections
- examples\hdevelop\Applications\Calibration\3d_position_of_rectangle.dev
  Estimate 3D pose of rectangular objects
- examples\hdevelop\Applications\Color\color_segmentation_pizza.dev
  Find salami pieces on pizza based on color image processing
- examples\hdevelop\Applications\DFF\phot_stereo_board.dev
  Reconstruct 3D SMD components using photometric stereo
- examples\hdevelop\Applications\DFF\phot_stereo_braille.dev
  Get 3D model of braille using photometric stereo
- examples\hdevelop\Applications\DFF\resistor.dev
  Extract solder by depth using multiple focus levels

- examples\hdevelop\Applications\FA\ball_seq.dev
  Inspect ball bondings (multiple images)
- examples\hdevelop\Applications\FA\board.dev
  Detect missing solder in images taken with different illumination direction
- examples\hdevelop\Applications\FA\circles.dev
  Fit circles into curved contour segments
  → description here in the Solution Guide Basics on page 25
- examples\hdevelop\Applications\FA\clip.dev
  Determine the position and orientation of clips
- examples\hdevelop\Applications\FA\find_pads.dev
  Find pads on a die.
- examples\hdevelop\Applications\FA\holes.dev
  Extract positions and radii of holes
- examples\hdevelop\Applications\FA\hull.dev
  Inspect an injection molded nozzle
- examples\hdevelop\Applications\FA\ic.dev
  Extract resistors, capacitors and ICs from a board using color information
  → description here in the Solution Guide Basics on page 202
- examples\hdevelop\Applications\FA\matching_multi_channel_yogurt.dev
  Find yogurts of different flavors using color shape-based matching
- examples\hdevelop\Applications\FA\print_check.dev
  Perform a typical print quality inspection using a variation model
  → description here in the Solution Guide Basics on page 161
- examples\hdevelop\Applications\FA\print_check_single_chars.dev
  Perform a typical print quality inspection using variation models for each character
- examples\hdevelop\Applications\FA\rim.dev
  Inspect holes and characters on a rim
- examples\hdevelop\Applications\Medicine\vessel.dev
  Extract and measure a blood vessel
- examples\hdevelop\Applications\Monitoring\movement_col.dev
  Extract moving objects and scan them for specific color
- examples\hdevelop\Applications\Monitoring\optical_flow.dev
  Monitor traffic using optical flow
- examples\hdevelop\Applications\Monitoring\optical_flow_bicycle.dev
  Monitor moving bicycle using optical flow
- examples\hdevelop\Applications\Monitoring\xing.dev
  Monitor traffic using background estimation with postprocessing

- examples\hdevelop\Applications\Monitoring\xing_simple.dev
  Monitor traffic using background estimation
- examples\hdevelop\Applications\OCR\bottle.dev
  Read numbers on a beer bottle
  → description here in the Solution Guide Basics on page 243
- examples\hdevelop\Applications\OCR\bottlet.dev
  Train numbers on a beer bottle
- examples\hdevelop\Applications\OCR\dotprt.dev
  Read a dot print
- examples\hdevelop\Applications\OCR\engraved.dev
  Read characters on a metal surface
  → description here in the Solution Guide Basics on page 244
- examples\hdevelop\Applications\OCR\engravedt.dev
  Train characters on a metal surface
- examples\hdevelop\Applications\OCR\font.dev
  Read printed characters with interactive selection
- examples\hdevelop\Applications\OCR\fontt.dev
  Train printed characters and reclassify them
- examples\hdevelop\Applications\OCR\letter.dev
  Read printed characters with interactive selection
- examples\hdevelop\Applications\OCR\lettert.dev
  Train printed characters and reclassify them
- examples\hdevelop\Applications\OCR\ocr_dongle.dev
  Read dongle ID using OCR and regular expressions
- examples\hdevelop\Applications\OCR\ocr_gradient_feature.dev
  Compare OCR features
- examples\hdevelop\Applications\OCR\ocr_lot_number.dev
  Correct OCR result using regular expressions
- examples\hdevelop\Applications\OCR\ocrcolor.dev
  Read numbers using color information
  → description here in the Solution Guide Basics on page 245
- examples\hdevelop\Applications\OCR\ocrcolort.dev
  Segment numbers using color information and train the OCR
- examples\hdevelop\Applications\OCR\stamp_catalogue.dev
  Segment and group characters on a cluttered page
- examples\hdevelop\Applications\OCV\print_quality.dev
  Inspect the quality of the letter A in different images

- examples\hdevelop\Applications\Sequences\autobahn.dev
  Detect lane markers fast
- examples\hdevelop\Applications\Sequences\movement.dev
  Detect movements online using image differences
- examples\hdevelop\Applications\Sequences\movement_bg.dev
  Detect movements online using background estimator
- examples\hdevelop\Applications\Sequences\optical_flow_hydraulic_engineering.dev
  Get position, speed, and movement direction of particles using optical flow
- examples\hdevelop\Applications\Sequences\sequence_diff.dev
  Monitor traffic by comparing two successive images
- examples\hdevelop\Applications\Stereo\board_components.dev
  Extract board components by height using binocular stereo
  → description here in the Solution Guide Basics on page 262
- examples\hdevelop\Applications\Stereo\board_segmentation_uncalib.dev
  Extract board components by height using uncalibrated binocular stereo
- examples\hdevelop\Filter\Texture\entropy_image.dev
  Segment an image based on the entropy of gray values
- examples\hdevelop\Image\Features\gray_features.dev
  Calculate standard gray value features
- examples\hdevelop\Morphology\Gray-Values\pcb_inspection.dev
  Find defects on a PCB using gray value morphology
- examples\hdevelop\OCR\Neural-Nets\label_word_process_mlp.dev
  Read a best-before label using a lexicon to improve results
- examples\hdevelop\OCR\Support-Vector-Machines\letters_svm.dev
  Train an SVM OCR classifier and reclassify the training samples
- examples\hdevelop\Regions\Features\inner_rectangle1.dev
  Find knot-free rectangles in a piece of wood
- examples\hdevelop\Regions\Features\orientation_region.dev
  Calculate the orientation of regions
- examples\hdevelop\Regions\Transformations\connection.dev
  Calculate connected components of regions
- examples\hdevelop\Regions\Transformations\fill_up_shape.dev
  Fill up holes in regions that have a given shape
- examples\hdevelop\Regions\Transformations\max_connection.dev
  Limit the number of regions returned by the connection operator
- examples\hdevelop\Segmentation\Classification\class_2dim_sup.dev
  Segment a color image using two-dimensional pixel classification

- examples\hdevelop\Segmentation\Classification\class_2dim_unsup.dev
  Segment a color image by clustering
- examples\hdevelop\Segmentation\Classification\class_ndim_box.dev
  Classify pixels using hyper-cuboids
- examples\hdevelop\Segmentation\Classification\class_ndim_box.dev
  Classify pixels using hyper-cuboids
- examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev
  Classifying pixels using hyper-spheres
- examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev
  Classify pixels using hyper-spheres
- examples\hdevelop\Segmentation\Classification\classify_image_class_gmm.dev
  Segment an RGB image with a GMM classifier
- examples\hdevelop\Segmentation\Classification\classify_image_class_mlp.dev
  Segment an RGB image with an MLP classifier
- examples\hdevelop\Segmentation\Classification\classify_image_class_svm.dev
  Segment an RGB image with an SVM classifier
- examples\hdevelop\Segmentation\Classification\novelty_detection_gmm.dev
  Inspect a web using texture classification with gaussian mixture models
- examples\hdevelop\Segmentation\Classification\novelty_detection_svm.dev
  Inspect a web using texture classification with support vector machine classifiers
- examples\hdevelop\Segmentation\Threshold\threshold.dev
  Select gray values lying within an interval
- examples\solution_guide\1d_measuring\measure_ring.dev
  Determine the width of cogs with a circular measure object
  → description in the Solution Guide II-D on page 23
- examples\solution_guide\2d_measuring\inspect_bga.dev
  Inspect BGA for missing or deviating balls
  → description in the Solution Guide II-E on page 57
- examples\solution_guide\2d_measuring\measure_chip.dev
  Get extents and positions of rectangular chip components
  → description in the Solution Guide II-E on page 49
- examples\solution_guide\2d_measuring\measure_circles.dev
  Get radii and positions of circular shapes
  → description in the Solution Guide II-E on page 52
- examples\solution_guide\2d_measuring\measure_grid.dev
  Get junctions of a grid that separates keys
  → description in the Solution Guide II-E on page 42
- examples\solution_guide\2d_measuring\measure_metal_part_extended.dev

Measure several features of a metal part
→ description in the Solution Guide II-E on page 39

- examples\solution_guide\2d_measuring\measure_metal_part_first_example.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 6
- examples\solution_guide\2d_measuring\measure_metal_part_id.dev
  Inspect metal part for missing or deviating circular shapes
  → description in the Solution Guide II-E on page 64
- examples\solution_guide\2d_measuring\measure_perspective_scratch.dev
  Measure the length of a scratch in world coordinates
  → description in the Solution Guide II-E on page 71
- examples\solution_guide\2d_measuring\measure_pump.dev
  Measure positions and radii of circular shapes
  → description in the Solution Guide II-E on page 54
- examples\solution_guide\2d_measuring\measure_screw.dev
  Measure several features of a screw
  → description in the Solution Guide II-E on page 31
- examples\solution_guide\3d_machine_vision\grid_rectification_arbitrary_distortion.dev
  Determine differences between two printed pages, even if there are distortions in the vertical direction
  → description in the Solution Guide II-F on page 133
- examples\solution_guide\3d_machine_vision\handeye_stationarycam_grasp_nut.dev
  Calculate pose for grasping a nut based on results of hand-eye calibration for a stationary camera
  → description in the Solution Guide II-F on page 120
- examples\solution_guide\basics\color_fuses.dev
  Sort fuses by color
  → description here in the Solution Guide Basics on page 196
- examples\solution_guide\basics\color_pieces.dev
  Check for completeness of colored game pieces using MLP classification
  → description here in the Solution Guide Basics on page 198
- examples\solution_guide\basics\color_pieces_euclid.dev
  Check for completeness of game color pieces using Euclidean classification
- examples\solution_guide\basics\color_simple.dev
  Segment color image in HSV color space
  → description here in the Solution Guide Basics on page 195
- examples\solution_guide\basics\edge_segments.dev
  Extract connected edges segments
  → description here in the Solution Guide Basics on page 77
- examples\solution_guide\basics\gen_training_file.dev
  Generate a training file for the OCR
  → description here in the Solution Guide Basics on page 241

- examples\solution_guide\basics\simple_reading.dev
  Read characters using a trained font
  → description here in the Solution Guide Basics on page 243
- examples\solution_guide\image_acquisition\line_scan.dev
  Simulate grabbing from a line scan camera and merge images and extracted regions
  → description in the Solution Guide II-A on page 37

C++

- examples\cpp\source\example1.cpp
  Reading an image and performing a simple segmentation
- examples\cpp\source\xing.cpp
  Monitors traffic using background estimation with postprocessing

Visual Basic

- examples\vb\Applications\Monitoring\xing.vbp
  Monitor traffic using background estimation
- examples\vb\Manual\eyes.vbp
  Locate the eyes of a monkey
- examples\vb\Segmentation\monkey.vbp
  Blob analysis with interactive control of some parameters

C

- examples\c\source\example2.c
  Performing some basic segmentation
- examples\c\source\example_multithreaded1.c
  Using multiple threads with Parallel HALCON

4.4 Selecting Operators

4.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

4.4.2 Align ROIs Or Images

Operators for rectifying images are described in the Solution Guide II-B.

4.4.3 Rectify Image(s)

Operators for rectifying images are described in the Solution Guide II-F.

4.4.4 Preprocess Image(s) (Filtering)

Standard:

mean_image, gauss_image, binomial_filter, median_image

Advanced:

smooth_image, anisotrope_diff, fill_interlace, rank_image

4.4.5 Extract Segmentation Parameters

Standard:

gray_histo_abs, histo_to_thresh

Advanced:

intensity

4.4.6 Segment Image(s)

Standard:

threshold, fast_threshold, bin_threshold, dyn_threshold, histo_to_thresh, gray_histo

Advanced:

watersheds, watersheds_threshold, regiongrowing, regiongrowing_mean, var_threshold

4.4.7 Process Regions

Standard:

connection, select_shape, opening_circle, closing_circle, opening_rectangle1, closing_rectangle1, difference, intersection, union1, shape_trans, fill_up

Advanced:

select_shape_proto, select_gray, clip_region, sort_region, skeleton, partition_dynamic, rank_region

Morphological operators can be found in the Reference Manual in the chapter "Morphology".

4.4.8 Extract Features

Standard:

area_center, smallest_rectangle1, smallest_rectangle2, compactness, eccentricity, elliptic_axis, area_center_gray, intensity, min_max_gray

Advanced:

diameter_region, inner_rectangle1, inner_circle, gray_histo_abs, entropy_gray

4.4.9 Transform Results Into World Coordinates

Standard:

image_points_to_world_plane

Advanced:

gen_contour_region_xld, contour_to_world_plane_xld

More operators for transforming results into world coordinates are described in the Solution Guide II-F.

4.4.10 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

4.5 Relation to Other Methods

4.5.1 Methods that are Useful for Blob Analysis

Color Processing (see description on page 191)
Color processing can be considered as an advanced way of blob analysis that uses three color channels instead of one gray value channel. HALCON provides operators for color space transformation, feature extraction, and pixel classification, which can be used in combination with blob analysis.

4.5.2 Methods that are Using Blob Analysis

OCR (see description on page 233)
Blob analysis is typically used as a preprocessing step for OCR to segment the characters.

4.5.3 Alternatives to Blob Analysis

Edge Extraction (Subpixel-Precise) (see description on page 85)
In blob analysis a region is described by the gray values of its pixels. As an alternative, a region could be described by the change of the gray values at their borders. This approach is called edge detection.

Classification (see description on page 171)
To select specific gray values, thresholds must be determined. In most cases, fixed values are used or the current value is determined by a feature operator. In some cases it is useful if the system determines the ranges automatically. This can be achieved by using a classifier. In addition, a classifier can also be used to automatically distinguish between good and bad objects based on the extracted features and samples for both classes.

4.6 Tips & Tricks

4.6.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for blob analysis. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the segmentation is performed can be restricted, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

4.6.2 Connected Components

By default, most HALCON segmentation operators like `threshold` return one region even if you see multiple not connected areas on the screen. To transform this region into separated objects (i.e., connected components in the HALCON nomenclature) one has to call `connection`.

4.6.3 Speed Up

Many online applications require maximum speed. Because of its flexibility, HALCON offers many ways to achieve this goal. Here the most common methods are listed.

- Regions of interest are the standard method to increase the speed by processing only those areas where objects need to be inspected. This can be done using pre-defined regions but also by an online generation of the regions of interest that depend on other objects found in the image.
- If an object has a specific minimum size, the operator `fast_threshold` is a fast alternative to `threshold`. This kind of fast operator can also directly be generated by using operators like `gen_grid_region` and `reduce_domain` before calling the thresholding operator.
- By default, HALCON performs some data consistency checks. These can be switched off using `set_check`.

- By default, HALCON initializes new images. Using `set_system` with the parameter `"init_new_image"`, this behavior can be changed.

4.7 Advanced Topics

4.7.1 Line Scan Cameras

In general, line scan cameras are treated like normal area sensors. In some cases, however, not single images but an "infinite" sequence of images showing objects, e.g., on a conveyor belt, must be processed. In this case the end of one image is the beginning of the next one. This means that objects that partially lie in both images must be combined into one object. For this purpose HALCON provides the operator `merge_regions_line_scan`. This operator is called after the segmentation of one image, and combines the current objects with those of previous images. For more information see the Solution Guide II-A.

4.7.2 High Accuracy

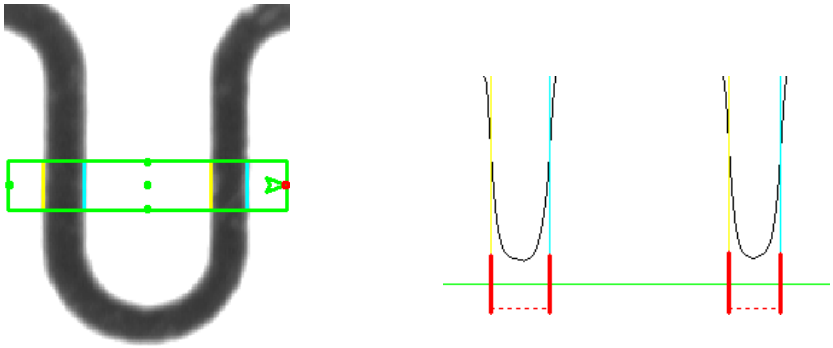
Sometimes high accuracy is required. This is difficult with blob analysis because objects are only extracted with integer pixel coordinates. Note, however, that many features that can be calculated for regions, e.g., the center of gravity, will be subpixel-precise. One way to get higher accuracy is to use a higher resolution. This has the effect that the higher number of pixels for each region results in better statistics to estimate features like the center of gravity (`area_center`). As an alternative, gray value features (like `area_center_gray`) can be used if the object fulfills specific gray value requirements. Here, the higher accuracy comes from the fact that for each pixel 255 values instead of one value (foreground or background) is used. If a very high accuracy is required, you should use the subpixel-precise edge and line extraction on page 85.

Chapter 5

1D Measuring

The idea of 1D measuring (also called 1D metrology or caliper) is very intuitive: Along a predefined region of interest, edges are located that are mainly perpendicular to the orientation of the region of interest. Here, edges are defined as transitions from dark to bright or from bright to dark.

Based on the extracted edges, you can measure the dimensions of parts. For example, you can measure the width of a part by placing a region of interest over it and locating the edges on its left and the right side. The effect of this can be seen in figure 5.1a, whereas figure 5.1b shows the corresponding gray value profile.



a) b)

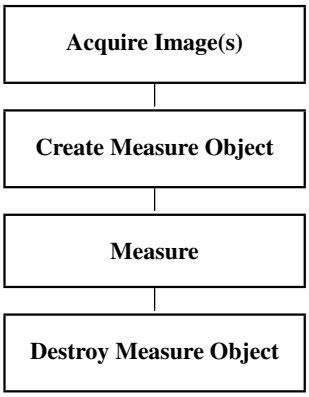
Figure 5.1: (a) Measuring a fuse wire; (b) gray value profile along the region of measurement with extracted edges.

As an alternative to these simple rectangular regions of interest, circular arcs can be used to measure, e.g., the widths of the cogs on a cog wheel.

The advantage of the measure approach is its ease of use combined with a short execution time and a very high accuracy. With only a few operators, high-performing applications can be realized.

5.1 Basic Concept

Measuring consists of four main steps:



5.1.1 Acquire Image(s)

First, an image is acquired.

For detailed information see the description of this method on page 13.

5.1.2 Create Measure Object

Having acquired the image, you specify where to measure, i.e., you describe the position, orientation, etc. of the line or arc along which you want to measure. Together with some other parameters, this information is stored in the so-called measure object.

You access the measure object by using a so-called handle. Similarly to a file handle, this handle is needed when working with the tool. Each time the measure tool is executed, this handle is passed as a parameter.

In object-oriented languages like C++ it is possible to use the measure class instead of the low-level approach with handles. Here, creation and destruction are realized with the standard object-oriented methods.

5.1.3 Measure

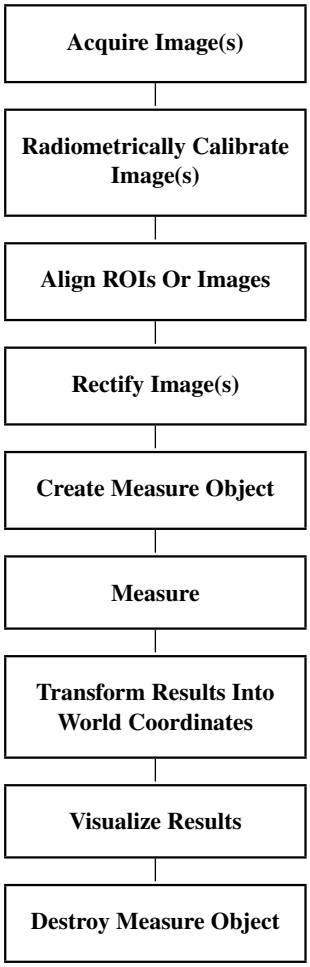
Then, you can apply the measuring by specifying the measure object and some other vision parameters like, e.g., the minimum contrast. You can find detailed information about this step in the Solution Guide II-D in section 3 on page 12.

5.1.4 Destroy Measure Object

When you no longer need the measure object, you destroy it by passing the handle to `close_measure`.

5.2 Extended Concept

In many cases, a measuring application will be more complex than described above. Reasons for this are, e.g., clutter or inhomogeneous illumination. Furthermore, post-processing like transforming the features to real-world units, or visualization of results may be required.



5.2.1 Radiometrically Calibrate Image(s)

To allow high-accuracy measurements, the camera should have a linear response function, i.e., the gray values in the images should depend linearly on the incoming energy. Since some cameras do not have a linear response function, HALCON provides the so-called radiometric calibration (gray value calibration): With the operator `radiometric_self_calibration` you can determine the inverse response function of the camera (offline) and then apply this function to the images using `lut_trans` before performing the measuring.

5.2.2 Align ROIs Or Images

In some applications, the line or arc along which you want to measure, must be aligned relative to another object.

How to perform alignment using shape-based matching is described in the Solution Guide II-B in section 4.3.4 on page 37.

5.2.3 Rectify Image(s)

Similarly to alignment, it may be necessary to rectify the image, e.g., to remove lens distortion.

Detailed information about rectifying images can be found in the Solution Guide II-F in section 3.3 on page 49.

5.2.4 Create Measure Object

You can teach the measurement line or arc interactively with operators like `draw_rectangle2` or read its parameters from file (`read_string`). As an alternative, its coordinates can be generated based on the results of other vision tools like Blob Analysis (see the description of this method on page 33). In particular, the measurement line or arc may need to be aligned to a certain object as described above.

If the measurement is always performed along the same line or arc, you can create the measure object offline and then use it multiple times before destroying it. However, if you want to align the measurement, the position and orientation of the line or arc will differ for each image. In this case, you must create a new measure object for each image. An exception to this rule is if only the position changes but not the orientation. Then, you can keep the measure object and adapt its position via `translate_measure`.

Please refer to the Solution Guide II-D, section 2 on page 6, for more information.

5.2.5 Transform Results Into World Coordinates

If you have calibrated your vision system, you can easily transform the results of measuring into world coordinates with `image_points_to_world_plane`. How to do this is described in the Solution Guide II-D in section 3.5 on page 21.

This is described in detail in the Solution Guide II-F in section 3.2 on page 44.

5.2.6 Visualize Results

The best way to visualize edge positions is to create (short) XLD line segments with operators like gen_contour_polygon_xld.

For detailed information see the description of this method on page 269.

5.3 Programming Examples

The following examples gives a brief introduction to using the 1D measuring tool of HALCON. The longest parts are pre- and postprocessing; the measurement itself consists only of two operator calls. Further examples are described in the Solution Guide II-D.

5.3.1 Inspecting a Fuse

Example: examples\solution_guide\basics\fuse.dev

Preprocessing consists of the generation of the measurement line. In the example program, this step is accomplished by assigning the measure object's parameters to variables.

```
read_image (Fuse, 'fuse')
Row := 297
Column := 545
Length1 := 80
Length2 := 10
Angle := rad(90)
gen_measure_rectangle2 (Row, Column, Angle, Length1, Length2, Width,
                        Height, 'bilinear', MeasureHandle)
```

Now the actual measurement is performed by applying the measure object to the image. The parameters are chosen such that edges around dark areas are grouped to so called pairs, returning the position of the edges together with the width and the distance of the pairs.

```
measure_pairs (Fuse, MeasureHandle, 1, 1, 'negative', 'all', RowEdgeFirst,
               ColumnEdgeFirst, AmplitudeFirst, RowEdgeSecond,
               ColumnEdgeSecond, AmplitudeSecond, IntraDistance,
               InterDistance)
```

The last part of the program displays the results by generating a region with the parameters of the measurement line and converting the edge positions to short XLD contours (see figure 5.2).

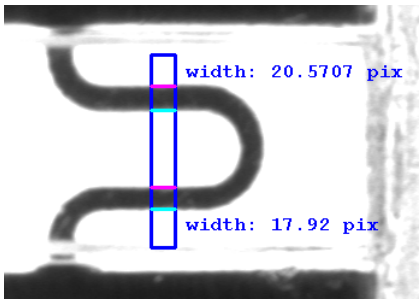


Figure 5.2: Measuring the width of the fuse wire.

```
for i := 0 to |RowEdgeFirst|-1 by 1
  gen_contour_polygon_xld (EdgeFirst,
                           [-sin(Angle+rad(90))*Length2+RowEdgeFirst[i],
                           -sin(Angle-rad(90))*Length2+RowEdgeFirst[i]],
                           [cos(Angle+rad(90))*Length2+ColumnEdgeFirst[i],
                           cos(Angle-rad(90))*Length2+ColumnEdgeFirst[i]])
  gen_contour_polygon_xld (EdgeSecond,
                           [-sin(Angle+rad(90))*Length2+RowEdgeSecond[i],
                           -sin(Angle-rad(90))*Length2+RowEdgeSecond[i]],
                           [cos(Angle+rad(90))*Length2+ColumnEdgeSecond[i],
                           cos(Angle-rad(90))*Length2+ColumnEdgeSecond[i]])
  write_string (WindowID, 'width: '+IntraDistance[i]+' pix')
endfor
```

5.3.2 Inspect Cast Part

Example: examples\hdevelop\Applications\Measure\measure_arc.dev

The task of this example is to inspect the distance between elongated holes of a cast part after chamfering (see figure 5.3). Note that to achieve best accuracy it would be recommended to use backlight combined with a telecentric lens instead of the depicted setup.

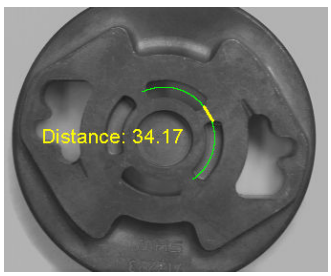


Figure 5.3: Measuring the distance between the holes.

This task can be solved easily by using the measure tool with a circular measurement ROI. The center of the ROI is placed into the center of the cast part; its radius is set to the distance of the elongated holes from the center.

```
Row := 275
Column := 335
Radius := 107
AngleStart := -rad(55)
AngleExtent := rad(170)
gen_measure_arc (Row, Column, Radius, AngleStart, AngleExtent, 10, Width,
                 Height, 'nearest_neighbor', MeasureHandle)
```

Now, the distance between the holes can be measured with a single operator call:

```
measure_pos (Zeiss1, MeasureHandle, 1, 10, 'all', 'all', RowEdge,
             ColumnEdge, Amplitude, Distance)
```

5.3.3 Inspecting an IC Using Fuzzy Measuring

Example: examples\hdevelop\Applications\Measure\fuzzy_measure_pin.dev

The task of this example is to inspect the lead width and the lead distance of the IC depicted in figure 5.4.

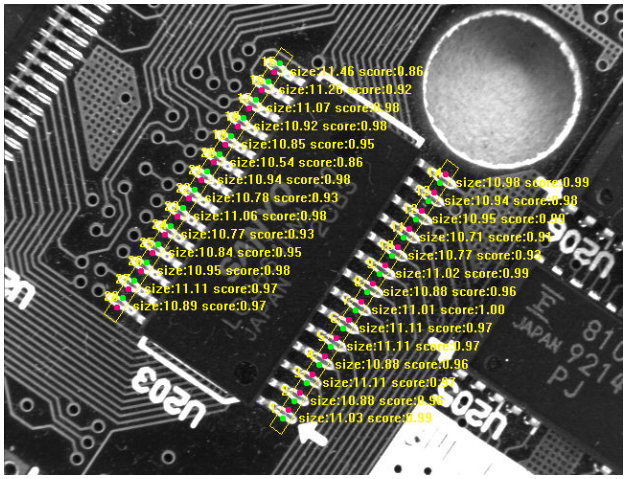


Figure 5.4: Measuring the width and distance of the leads.

The illumination conditions in this example are quite difficult. This has the effect that four edges are visible for each lead. Fuzzy rules are used to restrict the measurement to the correct (outer) leads.

```
gen_measure_rectangle2 (305.5, 375.5, 0.982, 167, 7.5, Width, Height,
                        'nearest_neighbor', MeasureHandle1)
create_funct_1d_pairs ([0.0, 0.3], [1.0,0.0], FuzzyAbsSizeDiffFunction)
set_fuzzy_measure_norm_pair (MeasureHandle1, 11.0, 'size_abs_diff',
                             FuzzyAbsSizeDiffFunction)
fuzzy_measure_pairs (Image, MeasureHandle1, 1, 30, 0.5, 'positive',
                     RowEdgeFirst1, ColumnEdgeFirst1, AmplitudeFirst1,
                     RowEdgeSecond1, ColumnEdgeSecond1, AmplitudeSecond1,
                     RowEdgeMiddle1, ColumnEdgeMiddle1, FuzzyScore1,
                     IntraDistance1, InterDistance1)
```

5.3.4 Measuring Leads of a Moving IC

Example: examples\hdevelop\Applications\FA\pm_measure_board.dev

The task of this example is to measure the positions of the leads of a chip (see figure 5.5). Because the chip can appear at varying positions and angles, the regions of interest used for the measurement must be aligned.

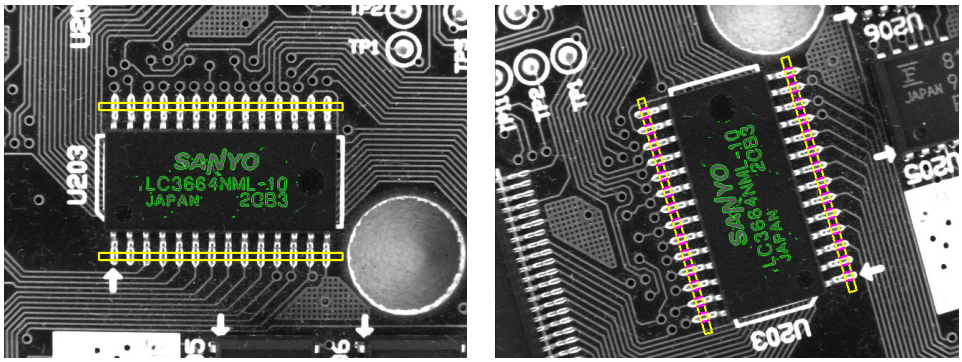


Figure 5.5: (a) Model image with measurement ROIs; (b) measuring the leads in the aligned ROIs.

In this case, the alignment is achieved by searching for the print on the chip using shape-based matching (see Template Matching on page 115).

```
gen_rectangle1 (Rectangle, Row1, Column1, Row2, Column2)
reduce_domain (Image, Rectangle, ImageReduced)
create_shape_model (ImageReduced, 4, 0, rad(360), rad(1), 'none',
                    'use_polarity', 30, 10, ModelID)
```

After the print has been found, the positions of the measurement ROIs are transformed relative to the position of the print.

```
find_shape_model (ImageCheck, ModelID, 0, rad(360), 0.7, 1, 0.5,
                  'least_squares', 4, 0.7, RowCheck, ColumnCheck,
                  AngleCheck, Score)

hom_mat2d_identity (HomMat2DIdentity)
hom_mat2d_translate (HomMat2DIdentity, RowCheck, ColumnCheck,
                     HomMat2DTranslate)
hom_mat2d_rotate (HomMat2DTranslate, AngleCheck, RowCheck,
                  ColumnCheck, HomMat2DRotate)
affine_trans_pixel (HomMat2DRotate, Rect1Row, Rect1Col,
                    Rect1RowCheck, Rect1ColCheck)
```

Then, the measure tools are created and the measurement is applied.

```
gen_measure_rectangle2 (Rect1RowCheck, Rect1ColCheck, AngleCheck,
                        RectLength1, RectLength2, Width, Height,
                        'bilinear', MeasureHandle1)
measure_pairs (ImageCheck, MeasureHandle1, 2, 90, 'positive',
               'all', RowEdgeFirst1, ColumnEdgeFirst1,
               AmplitudeFirst1, RowEdgeSecond1, ColumnEdgeSecond1,
               AmplitudeSecond1, IntraDistance1, InterDistance1)
```

5.3.5 Inspect IC

Example: examples\hdevelop\Applications\Measure\measure_pin.dev

The task of this example is to inspect major dimensions of an IC (see figure 5.6).

In the first step the extent of each lead and the distance between the leads is measured. For this, a rectangle that contains the leads is defined (see figure 5.6a), which is used to generate the measure object. This is used to extract pairs of straight edges that lie perpendicular to the major axis of the rectangle.

```
gen_measure_rectangle2 (Row, Column, Phi, Length1, Length2, Width, Height,
                        'nearest_neighbor', MeasureHandle)
measure_pairs (Image, MeasureHandle, 1.5, 30, 'negative', 'all',
               RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst,
               RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond,
               PinWidth, PinDistance)
```

From the extracted pairs of straight edges, the number of leads, their average width, and the average distance between them is derived.

```
numPins := |PinWidth|
avgPinWidth := sum(PinWidth)/|PinWidth|
avgPinDistance := sum(PinDistance)/|PinDistance|
```

The second part shows the power of the measure tool: The length of the leads are determined. This is possible although each lead has a width of only a few pixels. For this, a new measure object is generated

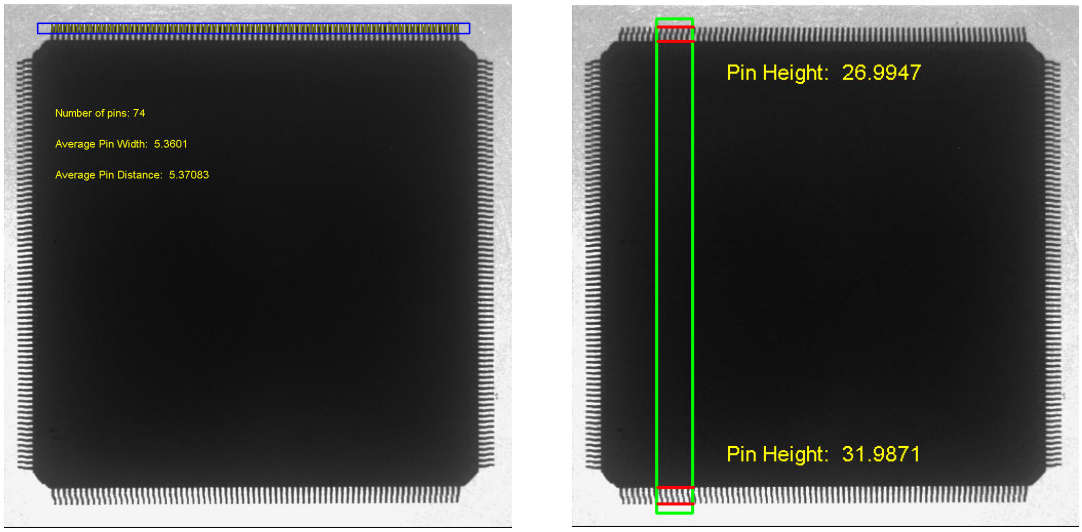


Figure 5.6: Measuring the dimensions of leads: (a) width of the leads and distance between them; (b) length of the leads.

based on a rectangle that contains the leads on two opposite sides of the IC (see figure 5.6b). The distance between the first and the second found edge is the length of the upper leads, and the distance between the third and the fourth edge is the length of the lower leads.

```
gen_measure_rectangle2 (Row, Column, Phi, Length1, Length2, Width, Height,
                        'nearest_neighbor', MeasureHandle)
measure_pos (Image, MeasureHandle, 1.5, 30, 'all', 'all', RowEdge,
             ColumnEdge, Amplitude, Distance)
```

5.3.6 Other Examples

HDevelop

- examples\hdevelop\Applications\Calibration\3d-coordinates.dev
  Measure slanted object in world coordinates
- examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev
  Measure distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- examples\hdevelop\Applications\Measure\measure_online.dev
  Measure your object in a live image
- examples\hdevelop\Tools\Measure\gen_measure_arc.dev
  Measure edges perpendicular to a given arc

- examples\hdevelop\Tools\Measure\gen_measure_rectangle2.dev
  Measure edges perpendicular to a given line
- examples\solution_guide\1d_measuring\fuzzy_measure_switch.dev
  Determine the width of and the distance between the pins of a switch with a fuzzy measure object
  → description in the Solution Guide II-D on page 27
- examples\solution_guide\1d_measuring\measure_caliper.dev
  Measure the distance between the pitch lines of a caliper
  → description in the Solution Guide II-D on page 17
- examples\solution_guide\1d_measuring\measure_ic_leads.dev
  Measure leads of an IC
  → description in the Solution Guide II-D on page 12
- examples\solution_guide\1d_measuring\measure_ring.dev
  Determine the width of cogs with a circular measure object
  → description in the Solution Guide II-D on page 23
- examples\solution_guide\1d_measuring\measure_switch.dev
  Determine the width of and the distance between the pins of a switch
  → description in the Solution Guide II-D on page 5
- examples\solution_guide\3d_machine_vision\camera_calibration_exterior.dev
  Measure positions on a caliper rule using camera calibration
  → description in the Solution Guide II-F on page 41
- examples\solution_guide\3d_machine_vision\camera_calibration_multi_image.dev
  Calibrate the camera and measures positions on a caliper rule
  → description in the Solution Guide II-F on page 39
- examples\solution_guide\basics\close_contour_gaps.dev
  Close gaps in extracted straight contours
  → description here in the Solution Guide Basics on page 105
- examples\solution_guide\shape_matching\align_measurements.dev
  Inspect individual razor blades using shape-based matching to align ROIs for the measure tool
  → description in the Solution Guide II-B on page 37

C++

- examples\cpp\source\fuzzy_measure_pin.cpp
  Measures pins of an IC using fuzzy measure
- examples\mfc\Matching\Matching.cpp
  Locate an IC using HALCON/C++ and MFC, creating a HALCON window
- examples\mfc\MatchingCOM\Matching.cpp
  Locate an IC using HALCON/COM and MFC
- examples\mfc\MatchingExtWin\Matching.cpp
  Locate an IC using HALCON/C++ and MFC, paint into an existing window

- examples\motif\Matching\source\matching.cpp
  Locate an IC using HALCON/C++ and Motif
- examples\qt\Matching\matching.cpp
  Locate an IC using HALCON/C++ and Qt

C#

- examples\c#\Matching\vs.net\Matching.csproj
  Locate an IC on a board and measure pin distances

Visual Basic .NET

- examples\vb.net\Matching\vs.net\Matching.vbproj
  Locate an IC on a board and measures pin distances

Visual Basic

- examples\vb\Online\Measure\measure.vbp
  Measuring edge positions in a live image
- examples\vb\Tools\Matching\matching.vbp
  Locate an IC on a board and measure pin distances
- examples\vb\Tools\Measure\measure.vbp
  Measuring pins with interactive control of parameters

Delphi

- examples\delphi\Matching\matching.dpr
  Locate an IC on a board and measure pin distances

5.4 Selecting Operators

5.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

5.4.2 Radiometrically Calibrate Image(s)

Standard:

radiometric_self_calibration, lut_trans

5.4.3 Align ROIs Or Images

Operators for aligning ROIs or images are described in the Solution Guide II-B.

5.4.4 Rectify Image(s)

Operators for rectifying images are described in the Solution Guide II-F.

5.4.5 Create Measure Object

Standard:

gen_measure_rectangle2, gen_measure_arc, translate_measure

5.4.6 Measure

Standard:

measure_pos, measure_pairs

Advanced:

set_fuzzy_measure, fuzzy_measure_pos, fuzzy_measure_pairs, fuzzy_measure_pairing

5.4.7 Transform Results Into World Coordinates

Standard:

image_points_to_world_plane

Advanced:

gen_contour_region_xld, contour_to_world_plane_xld

More operators for transforming results into world coordinates are described in the Solution Guide II-F.

5.4.8 Visualize Results

Advanced:

gen_contour_polygon_xld

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

5.4.9 Destroy Measure Object

Standard:

close_measure

5.5 Relation to Other Methods

5.5.1 Alternatives to 1D Measuring

Edge Extraction (Subpixel-Precise) (see description on page 85)
A very flexible way to measure parameters of edges is to extract the edge contour with edges_sub_pix. The advantage of this approach is that it can handle free-form shapes. Furthermore, it allows to determine attributes like the edge direction for each edge point.

5.6 Tips & Tricks

5.6.1 Suppress Clutter or Noise

In many applications there is clutter or noise that must be suppressed. The measure operators offer multiple approaches to achieve this. The best one is to increase the threshold for the edge extraction to eliminate faint edges. In addition, the value for the smoothing parameter can be increased to smooth irrelevant edges away.

When grouping edges to pairs, noise edges can lead to an incorrect grouping if they are in the vicinity of the "real" edge and have the same polarity. In such a case you can suppress the noise edges by selecting only the strongest edges of a sequence of consecutive rising and falling edges.

5.6.2 Reuse of Measure Objects

Because the creation of a measure object needs some time, we recommend to reuse them if possible. If no alignment is needed, the measure object can, for example, be created offline and reused for each image. If the alignment involves only a translation, translate_measure can be used to correct the position.

5.6.3 Use an Absolute Gray Value Threshold

As an alternative to edge extraction, the measurements can be performed based on an absolute gray value threshold by using the operator measure_thresh. Here, all positions where the gray value crosses the given threshold are selected.

5.7 Advanced Topics

5.7.1 Fuzzy Measuring

In case there are extra edges that do not belong to the measurement, HALCON offers an extended version of measuring: fuzzy measuring. This tool allows to define so-called fuzzy rules, which describe the features of good edges. Possible features are, e.g., the position, the distance, the gray values, or the amplitude of edges. These functions are created with `create_funct_1d_pairs` and passed to the tool with `set_fuzzy_measure`. Based on these rules, the tool will select the most appropriate edges.

The advantage of this approach is the flexibility to deal with extra edges even if a very low minimum threshold or smoothing is used. An example for this approach is the example program `fuzzy_measure_pin.dev` on page 63.

Please refer to the Solution Guide II-D, section 4 on page 26, for more information.

5.7.2 Evaluation of Gray Values

To have full control over the evaluation of the gray values along the measurement line or arc, you can use `measure_projection`. The operator returns the projected gray values as an array of numbers, which can then be further processed with HALCON operators for tuple or function processing (see the chapters "Tuple" and "Tools ▷ Function" in the Reference Manual). Please refer to the Solution Guide II-D, section 3.4 on page 17, for more information.

Chapter 6

Edge Extraction (Pixel-Precise)

The traditional way of finding edges, i.e., dark / light transitions in an image, is to apply an edge filter. These filters have the effect to find pixels at the border between light and dark areas. In mathematical terms this means that these filters determine the image gradient. This image gradient is typically returned as the edge amplitude and/or the edge direction. By selecting all pixels with a high edge amplitude, contours between areas can be extracted.

HALCON offers all standard edge filters like the Sobel, Roberts, Robinson, or Frei filters. Besides these, post-processing operators like hysteresis thresholding or non-maximum suppression are provided. In addition, state-of-the-art filters that determine the edge amplitude and edge direction accurately are provided. This enables you to apply the filters in a flexible manner.

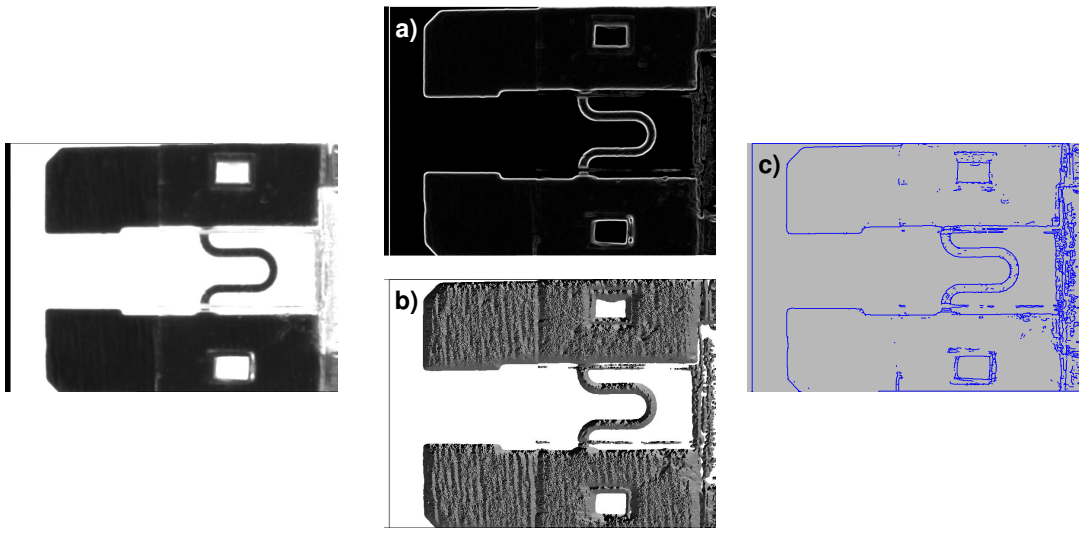


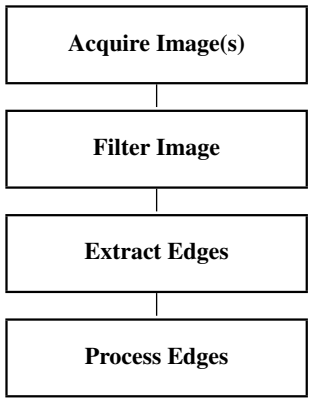
Figure 6.1: Result of applying an edge filter: (a) amplitude, (b) direction, (c) extracted edges.

Please note that in addition to this classical approach, HALCON provides advanced operators for

subpixel-precise edge and line extraction (see the description of this method on page 85) and for successive post-processing and feature extraction.

6.1 Basic Concept

Using edge filters typically consists of three basic steps:



6.1.1 Acquire Image(s)

First, an image is acquired.

For detailed information see the description of this method on page 13.

6.1.2 Filter Image

On the input image, an edge filter is applied. This operation results in one or two images. The basic result is the edge amplitude, which is typically stored as a byte image, with the gray value of each pixel representing the local edge amplitude. Optionally, the direction of the edges is returned. These values are stored in a so-called direction image, with the values 0...179 representing the angle in degrees divided by two.

6.1.3 Extract Edges

The result of applying the edge filter is an image containing the edge amplitudes. From this image, the edges are extracted by selecting the pixels with a given minimum edge amplitude using a threshold operator. The resulting edges are typically broader than one pixel and therefore have to be thinned. For this step, various methods are available.

6.1.4 Process Edges

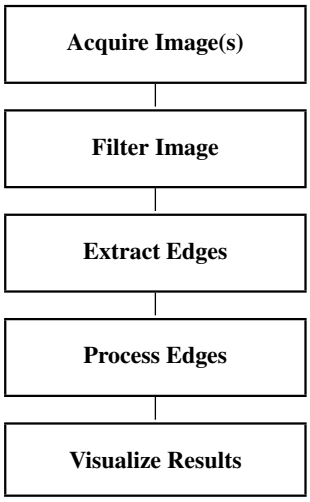
Having extracted the edges, two further processing steps can be applied: The first step is to convert the edge regions into another data structure for a potential further processing and for extracting features. The second step is to extract the regions enclosed by the edges.

6.1.5 A First Example

The following program shows an example for the basic concept of edge filters. As an edge filter, sobel_amp is applied with the mode 'thin_sum_abs' to get thin edges together with a 3x3 filter mask. Then, the operator threshold is used to extract all pixels with an edge amplitude higher than 20. The resulting region contains some areas where the edge is wider than one pixel. Therefore, the operator skeleton is applied to thin all edges completely. The result is depicted in figure 6.1c on page 73.

```
read_image (Image, 'fuse')
sobel_amp (Image, EdgeAmplitude, 'thin_sum_abs', 3)
threshold (EdgeAmplitude, Region, 20, 255)
skeleton (Region, Skeleton)
```

6.2 Extended Concept



6.2.1 Filter Image

HALCON offers a wide range of edge filters. One of the most popular filters is the Sobel filter. This is the best of the old-fashioned filters. It combines speed with a reasonable quality. The corresponding operators are called sobel_amp and sobel_dir.

In contrast, edges_image provides the state of the art of edge filters. This operator is actually more than just a filter. It includes a thinning of the edges using a non-maximum suppression and a hysteresis threshold for the selection of significant edge points. It also returns the edge direction and the edge amplitude very accurately, which is not the case with the Sobel filter. This operator is recommended if higher quality is more important than a longer execution time. If the images are not noisy or blurred, you can even combine accuracy and speed by using the mode 'sobel_fast' inside edges_image. The corresponding operator to find edges in multi-channel images, e.g., a color image, is edges_color.

6.2.2 Extract Edges

The easiest way to extract the edges from the edge amplitude image is to apply threshold to select pixels with a high edge amplitude. The result of this step is a region that contains all edge points. With skeleton, these edges can be thinned to a width of one pixel. As an advanced version for threshold, hysteresis_threshold can be used to eliminate insignificant edges. A further advanced option is to call the operator nonmax_suppression_dir before skeleton, which in difficult cases may result in more accurate edges. Note that in order to use this operator you must have computed the edge direction image.

In contrast, the advanced filter edges_image already includes the non-maximum suppression and the hysteresis threshold. Therefore, in this case a simple threshold suffices to extract edges that are one pixel wide.

If only the edge points as a region are needed, the operator inspect_shape_model can be used. Here, all steps including edge filtering, non-maximum suppression, and hysteresis thresholding are performed in one step with high efficiency.

6.2.3 Process Edges

If you want to extract the coordinates of edge segments, split_skeleton_lines is the right choice. This operator must be called for each connected component (result of connection) and returns all the control points of the line segments. As an alternative, a Hough transform can be used to obtain the line segments. Here, the operators hough_lines_dir and hough_lines are available.

You can extract the regions enclosed by the edges easily using background_seg. If regions merge because of gaps in the edges, the operators close_edges or close_edges_length can be used in advance to close the gaps before regions are extracted. As an alternative, morphological operators like opening_circle can be applied to the output regions of background_seg. In general, all operators described for the method Process Regions on page 37 can be applied here as well.

6.2.4 Visualize Results

Finally, you might want to display the images, the edges (regions), and the line segments.

For detailed information see the description of this method on page 269.

6.3 Programming Examples

This section gives a brief introduction to using HALCON for edge filtering and edge extraction.

6.3.1 Aerial Image Interpretation

Example: examples\solution_guide\basics\edge_segments.dev

Figure 6.2 shows an image taken from an aeroplane. The task is to extract the edges of roads and buildings as a basis for the image interpretation.

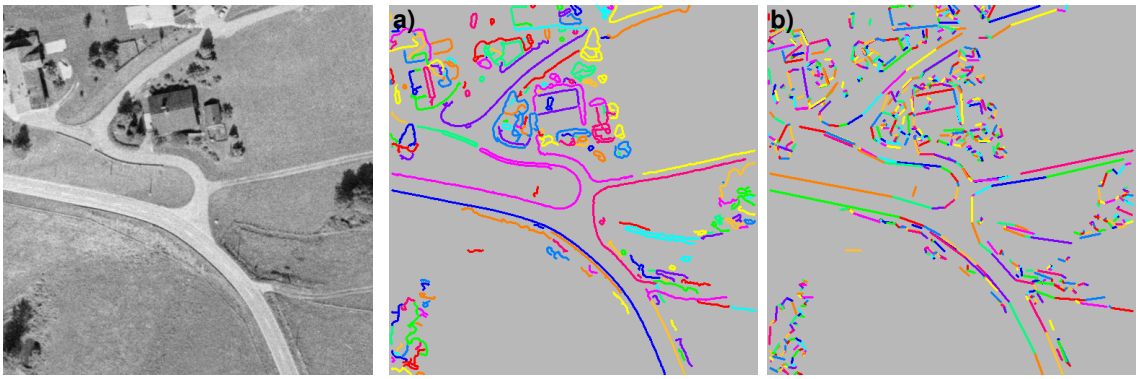


Figure 6.2: (a) Extracting edges and (b) approximating them by segments.

The extraction of edges is very simple and reliable when using the operator edges_image. This operator returns both the edge amplitude and the edge direction. Here, the parameters are selected such that a non-maximum suppression (parameter value 'nms') and a hysteresis threshold (threshold values 20 and 40) are performed. The non-maximum suppression has the effect that only pixels in the center of the edge are returned, together with the corresponding values for the amplitude and the direction. All other pixels are set to zero. Therefore, a threshold with the minimum amplitude of 1 is sufficient here. As a preparation for the next step, the edge contour regions are split up into their connected components.

```
read_image (Image, 'mreut')
edges_image (Image, ImaAmp, ImaDir, 'lanser2', 0.5, 'nms', 20, 40)
threshold (ImaAmp, Region, 1, 255)
connection (Region, ConnectedRegions)
```

The rest of the example program converts the region data into numeric values. To be more precise: the edges are approximated by individual line segments. This is performed by calling split_skeleton_lines for each connected component. The result of this call are four tuples that contain the start and the end coordinates of the line segments. For display purposes, each of these line segments is converted into an XLD contour.

```
count_obj (ConnectedRegions, Number)
gen_empty_obj (XLDContours)
for i := 1 to Number by 1
  select_obj (ConnectedRegions, SingleEdgeObject, i)
  split_skeleton_lines (SingleEdgeObject, 2, BeginRow, BeginCol, EndRow,
                        EndCol)
  for k := 0 to |BeginRow|-1 by 1
    gen_contour_polygon_xld (Contour, [BeginRow[k],EndRow[k]],
                             [BeginCol[k],EndCol[k]])
    concat_obj (XLDContours, Contour, XLDContours)
  endfor
endfor
dev_display (XLDContours)
```

6.3.2 Segmenting a Color Image

Example: examples\hdevelop\Filter\Edges\edges_color.dev

The task of this example is to segment the color image depicted in figure 6.3.

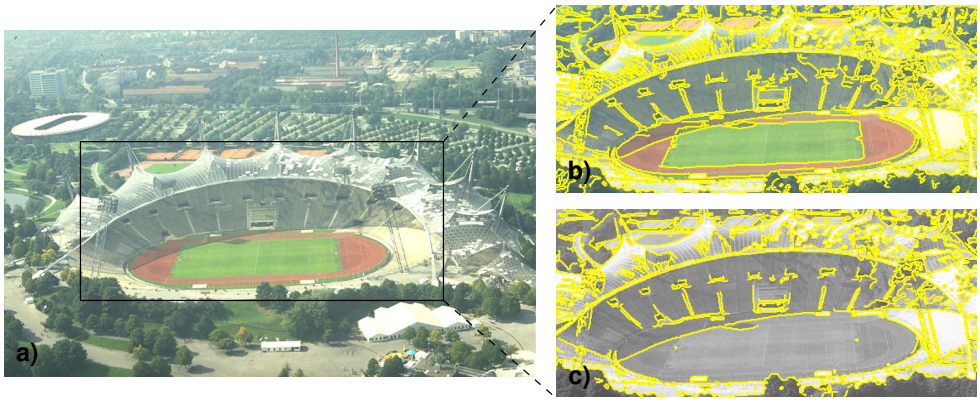


Figure 6.3: (a) Original image; (b) extracted color edges, overlaid on the color image; (c) extracted gray value edges, overlaid on the gray value image.

The example demonstrates the possibilities of a multi-channel edge filter. First, the gray value image is derived from the color information to show that some object borders can no longer be seen. For example, the (green) soccer field cannot be distinguished from the surrounding (red) track.

```
read_image (Image, 'olympic_stadium')
rgb1_to_gray (Image, GrayImage)
```

The color edge filter is applied and the edge amplitude is displayed. If you compare this to the filter result of the gray image the difference can be easily seen.

```
edges_color (Image, ImaAmp, ImaDir, 'canny', 1, 'none', -1, -1)
edges_image (GrayImage, ImaAmpGray, ImaDirGray, 'canny', 1, 'none', -1, -1)
```

Finally, the edge segments are extracted for both the color and the gray image and overlaid on the original image.

```
edges_color (Image, ImaAmpHyst, ImaDirHyst, 'canny', 1, 'nms', 20, 40)
threshold (ImaAmpHyst, RegionColor, 1, 255)
skeleton (RegionColor, EdgesColor)
dev_display (Image)
dev_display (EdgesColor)
stop ()
edges_image (GrayImage, ImaAmpGrayHyst, ImaDirGrayHyst, 'canny', 1, 'nms',
             20, 40)
threshold (ImaAmpGrayHyst, RegionGray, 1, 255)
skeleton (RegionGray, EdgesGray)
dev_display (GrayImage)
dev_display (EdgesGray)
```

6.3.3 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\roads.dev
  Extract roads from aerial image
  → description here in the Solution Guide Basics on page 106
- examples\hdevelop\Applications\FA\sharpness.dev
  Determine the sharpness of an image using different approaches
- examples\hdevelop\Applications\Sequences\autobahn.dev
  Detect lane markers fast
- examples\hdevelop\Filter\Edges\close_edges.dev
  Close edge gaps using the edge amplitude image
- examples\hdevelop\Filter\Edges\close_edgeslength.dev
  Close edge gaps using the edge amplitude image
- examples\hdevelop\Filter\Edges\derivate_gauss.dev
  Convolve an image with derivatives of the Gaussian (various usages)
- examples\hdevelop\Filter\Edges\diff_of_gauss.dev
  Approximate the LoG operator (Laplace of Gaussian)
- examples\hdevelop\Filter\Edges\edges_image.dev
  Extract edges (amplitude and direction) using Deriche, Lanser, Shen, or Canny filters
- examples\hdevelop\Filter\Edges\frei_amp.dev
  Extract edges (amplitude) using the Frei-Chen operator

- examples\hdevelop\Filter\Edges\frei_dir.dev
  Extract edges (amplitude and direction) using the Frei-Chen operator
- examples\hdevelop\Filter\Edges\highpass_image.dev
  Extract high frequency components from an image
- examples\hdevelop\Filter\Edges\info_edges.dev
  Estimate the width of a filter in edges_image
- examples\hdevelop\Filter\Edges\kirsch_amp.dev
  Extract edges (amplitude) using the Kirsch operator
- examples\hdevelop\Filter\Edges\kirsch_dir.dev
  Extract edges (amplitude and direction) using the Kirsch operator
- examples\hdevelop\Filter\Edges\laplace.dev
  Extract edges using the Laplace Operator
- examples\hdevelop\Filter\Edges\laplace_of_gauss.dev
  Extract edges using the LoG operator (Laplace of Gaussian)
- examples\hdevelop\Filter\Edges\prewitt_amp.dev
  Extract edges (amplitude) using the Prewitt operator
- examples\hdevelop\Filter\Edges\prewitt_dir.dev
  Extract edges (amplitude and direction) using the Prewitt operator
- examples\hdevelop\Filter\Edges\roberts.dev
  Extract edges using the Roberts filter
- examples\hdevelop\Filter\Edges\robinson_amp.dev
  Extract edges (amplitude) using the Robinson operator
- examples\hdevelop\Filter\Edges\robinson_dir.dev
  Extract edges (amplitude and direction) using the Robinson operator
- examples\hdevelop\Filter\Edges\sobel_amp.dev
  Extract edges (amplitude) using the Sobel operator
- examples\hdevelop\Filter\Edges\sobel_dir.dev
  Extract edges (amplitude and direction) using the Sobel operator
- examples\hdevelop\Filter\Lines\bandpass_image.dev
  Extract lines using bandpass filter
- examples\hdevelop\Graphics\Output\disp_xld.dev
  Display an XLD object
- examples\hdevelop\Regions\Transformations\background_seg.dev
  Calculate connected background components for given foreground regions
- examples\hdevelop\Segmentation\Edges\hysteresis_threshold.dev
  Perform a hysteresis threshold operation on an edge image
- examples\hdevelop\Segmentation\Edges\nonmax_suppression_amp.dev
  Suppress non-maximum points

- examples\hdevelop\Segmentation\Edges\nonmax_suppression_dir.dev
  Suppress non-maximum points on edges
- examples\hdevelop\Tools\Hough\hough_lines.dev
  Detect lines in an image using the Hough transform
- examples\hdevelop\Tools\Hough\hough_lines_dir.dev
  Detect lines in an image using the Hough transform and local gradient directions
- examples\solution_guide\basics\surface_scratch.dev
  Detect scratches on a surface via local thresholding and morphology
  → description here in the Solution Guide Basics on page 44

C++

- examples\cpp\source\example10.cpp
  Working with contour data and zooming results
- examples\cpp\source\example3.cpp
  Iterating over a set of image pixels
- examples\cpp\source\example4.cpp
  Extracting edges using a Sobel filter
- examples\cpp\source\example9.cpp
  Extract roads from aerial images

Visual Basic

- examples\vb\Online\Movement\movement.vbp
  Detect moving objects

C

- examples\c\source\example2.c
  Performing some basic segmentation
- examples\c\source\example4.c
  Applying an LUT for visualization of relevant details

6.4 Selecting Operators

6.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

6.4.2 Filter Image

Standard:

sobel_amp, sobel_dir, edges_image

Advanced:

derivate_gauss, edges_color

6.4.3 Extract Edges

Standard:

threshold, skeleton, inspect_shape_model

Advanced:

hysteresis_threshold, nonmax_suppression_dir

6.4.4 Process Edges

Standard:

background_seg, close_edges, close_edges_length, opening_circle, split_skeleton_lines, hough_lines_dir, hough_lines

6.4.5 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

6.5 Relation to Other Methods

6.5.1 Alternatives to Edge Extraction (Pixel-Precise)

Blob Analysis (see description on page 33)
As an alternative to edge extraction, blob analysis can be used. This approach provides many methods from simple thresholding to region growing and watershed methods.

6.6 Tips & Tricks

6.6.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for edge extraction. With domains, the processing can be focused to a certain area in the image and can thus be sped up. The more the region in which the edge filtering is performed can be restricted, the faster and more robust the extraction will be. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

6.6.2 Speed Up

Many online applications require maximum speed. Because of its flexibility, HALCON offers many ways to achieve this goal. Here the most common ones are listed.

- Regions of interest are the standard way to reduce the processing to only those areas where objects must be inspected. This can be achieved using pre-defined regions but also by an online generation of the regions of interest that depends on other objects found in the image.
- If high speed is important, the operators `sobel_amp` and `inspect_shape_model` are the preferred choice.
- By default, HALCON initializes new images. Using `set_system('init_new_image', 'false')`, this behavior can be changed to save execution time.

6.7 Advanced Topics

6.7.1 Subpixel Edge and Line Extraction

The state-of-the-art alternative to edge filters are the subpixel-accurate edge and line extractors of HALCON. Here, the result is not an edge image, but XLD contours that correspond to the locations of the edges and lines without any intermediate step. For more information see Edge Extraction (Subpixel-Precise) on page 85.

6.7.2 Contour Processing

As a post-processing step, you can convert the edge region into XLD contours by using, e.g., the operator `gen_contours_skeleton_xld`. The advantage of this approach is the extended set of operators offered for contour processing on page 97.

Chapter 7

Edge Extraction (Subpixel-Precise)

In addition to the traditional way of applying an edge filter to get the edge amplitude and thus the edges (see the method Edge Extraction (Pixel-Precise) on page 73), HALCON provides one-step operators that return subpixel-precise XLD contours. Besides this, not only edges but also lines can be extracted. This approach can also be applied to color images.

The advantage of this approach is its ease of use, because only a single operator call is needed. Furthermore, the accuracy and stability of the found contours is extremely high. Finally, HALCON offers a wide set of operators for the post-processing of the extracted contours, which includes, e.g., contour segmentation and fitting of circles, ellipses, and lines.

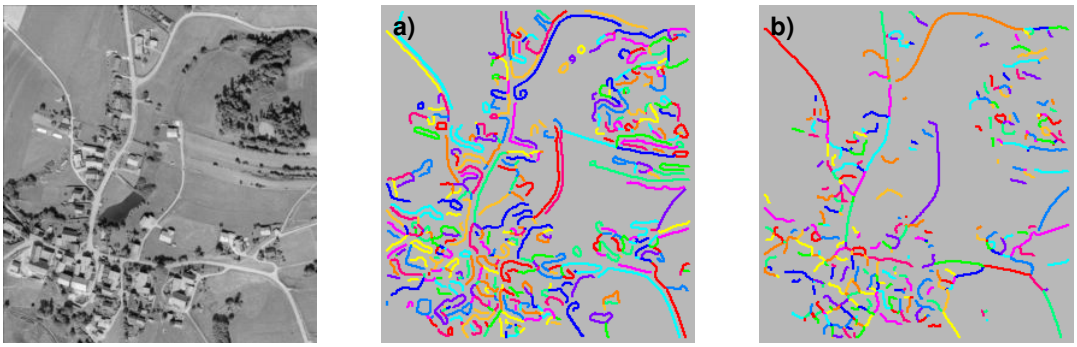
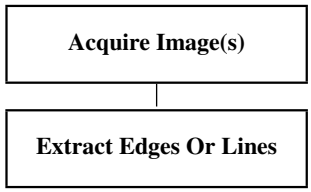


Figure 7.1: Result of contour extraction: (a) edge contours, (b) line contours.

This chapter covers only the extraction of contours. For information about processing them see the method Contour Processing on page 97.

7.1 Basic Concept

Extracting contours can easily be performed in a single step. Normally, no other operation is required.



7.1.1 Acquire Image(s)

First, an image is acquired as input for the process.

For detailed information see the description of this method on page 13.

7.1.2 Extract Edges Or Lines

HALCON offers various operators for the subpixel-accurate extraction of contours. The standard operator is based on the first derivative. It takes the image as input and returns the XLD contours. When using the second derivatives, first a Laplace operator must be executed before the contours along the zero crossings can be extracted. Besides the gray-value-based methods, HALCON provides the latest technology for the extraction of color edges.

Besides the extraction of edges, HALCON provides operators for the extraction of lines. In other systems lines are also called ridges. In contrast to edges, a line consists of two gray value transitions. Thus, a line can be considered as two parallel edges.

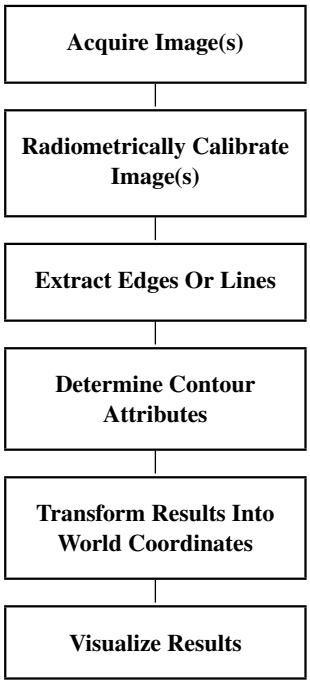
7.1.3 A First Example

The following program explains the basic concept of edge extraction. The only operator needed to extract edge contours is `edges_sub_pix`. It has the image as input and returns the XLD contours. Here, the filter `'lanser2'` is selected with a medium-sized smoothing mask. The low value for the parameter `Low` ensures that contours are tracked even along low-contrast parts. To show that the result consists of multiple contours, the 12-color mode for visualization is selected. The result is depicted in figure 7.1b on page 85.

```
read_image (Image, 'mreut4_3')
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 8, 50)
dev_set_colored (12)
dev_clear_window ()
dev_display (Edges)
```

7.2 Extended Concept

In addition to the extraction, optional steps can be performed.



7.2.1 Radiometrically Calibrate Image(s)

To extract edges or lines with high accuracy, the camera should have a linear response function, i.e., the gray values in the images should depend linearly on the incoming energy. Since some cameras do not have a linear response function, HALCON provides the so-called radiometric calibration (gray value calibration): With the operator `radiometric_self_calibration` you can determine the inverse response function of the camera (offline) and then apply this function to the images using `lut_trans` before performing the edge and line extraction.

7.2.2 Extract Edges Or Lines

The most often used operator for edge contour extraction is `edges_sub_pix`. You can select various filter methods by specifying the corresponding name with the parameter `Filter`. For standard applications, common values are, e.g., `'canny'` (based on a Gaussian convolution) or `'lanser2'`. The advantage of `'lanser2'` is the recursive implementation which has no increase in execution time when using a large smoothing. As a fast version the parameter value `'sobel_fast'` can be used, which is recommended as long as the image is not noisy or blurred.

The operator `zero_crossing_sub_pix` can be used in combination with a filter like `derivate_gauss` with parameter value `'laplace'`. The Laplace operator is mainly applied in the medical area.

To extract edges in multi-channel images, e.g., in a color image, HALCON provides the operator `edges_color_sub_pix` . Similar to `edges_sub_pix`, the parameter value `'sobel_fast'` is recommended for a fast edge extraction as long as the image is not noisy or blurred.

The most commonly used operator for line extraction is lines_gauss. Compared to lines_facet it is more robust and provides more flexibility. The width of lines that should be extracted is specified by the parameter Sigma: The wider the line, the larger the value must be chosen. For very wide lines we recommend to zoom down the image (zoom_image_factor) in order to reduce the overall execution time.

Like for edges, HALCON provides line extraction also for multi-channel images. The corresponding operator is lines_color.

7.2.3 Determine Contour Attributes

The edge and line extraction operators not only provide the XLD contours but also so-called attributes. Attributes are numerical values; they are associated either with each control point of the contour (called contour attribute) or with each contour as a whole (global contour attribute). The operators get_contour_attrib_xld and get_contour_global_attrib_xld enable you to access these values by specifying the attribute name.

The attribute values are returned as tuples of numbers. Typical attributes for edges are, e.g., the edge amplitude and direction. For lines a typical attribute is the line width. The available attributes can be queried for a given contour with query_contour_attribs_xld and query_contour_global_attribs_xld.

7.2.4 Transform Results Into World Coordinates

In many applications the coordinates of contours should be transformed into another coordinate system, e.g., into 3D world coordinates. After you have calibrated your vision system, you can easily perform the transformation with the operator contour_to_world_plane_xld. With this approach you can also eliminate lens distortions and perspective distortions.

This is described in detail in the Solution Guide II-F in section 3.2 on page 44.

7.2.5 Visualize Results

Finally, you might want to display the images and the contours.

For detailed information see the description of this method on page 269.

7.3 Programming Examples

This section gives a brief introduction to using HALCON for edge extraction.

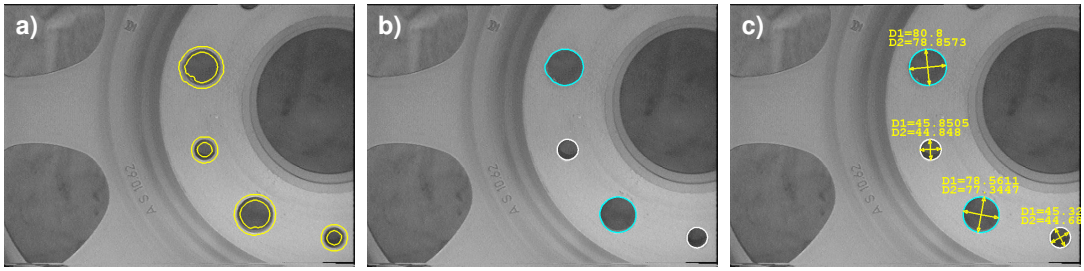


Figure 7.2: (a) automatically determined ROIs; (b) extracted edges; (c) computed ellipses and diameters.

7.3.1 Measuring the diameter of drilled holes

Example: `examples\solution_guide\basics\rim_simple.dev`

Figure 7.2 shows an image of a car rim. The task is to measure the diameters of the drilled holes.

First, a segmentation step is performed to roughly find the borders of the holes. The actual edge extraction is then performed only in these regions of interest (ROIs). This has two advantages: First, there are many edges in the image that are of no interest for the measurement. By restricting the processing to ROIs you can easily select the relevant objects. Secondly, the contour extraction is time-consuming. Thus, a reduced domain is an efficient way to speed-up the process.

Locating the holes is quite easy: First, all dark pixels are selected. After selecting all those connected components that are circular and have a certain size, only the holes remain. Finally, the regions of interest are obtained by accessing the borders of the holes and dilating them. The resulting ROIs are depicted in figure 7.2a.

```
threshold (Image, Dark, 0, 128)
connection (Dark, DarkRegions)
select_shape (DarkRegions, Circles, ['circularity','area'], 'and', [0.85,
            50], [1.0,99999])
boundary (Circles, RegionBorder, 'inner')
dilation_circle (RegionBorder, RegionDilation, 6.5)
union1 (RegionDilation, ROIEdges)
```

Calling `reduce_domain` changes the domain of the image to the prepared region of interest. Now, the edge extractor can be applied (see figure 7.2b).

```
reduce_domain (Image, ROIEdges, ImageROI)
edges_sub_pix (ImageROI, Edges, 'lanser2', 0.3, 10, 30)
```

The extracted contours are further processed to determine their diameter: With `fit_ellipse_contour_xld`, ellipses are fitted to the contours. In other words, those ellipses are determined that fit the extracted contours as closely as possible. The operator returns the parameters of the ellipses. With the operator `gen_ellipse_contour_xld`, the corresponding ellipses are created and displayed (compare figure 7.2b and figure 7.2c).

```
fit_ellipse_contour_xld (Edges, 'ftukey', -1, 2, 0, 200, 3, 2, Row, Column,
                         Phi, Ra, Rb, StartPhi, EndPhi, PointOrder)
NumHoles := |Ra|
gen_ellipse_contour_xld (ContEllipse, Row, Column, Phi, Ra, Rb,
                         gen_tuple_const(NumHoles,0),
                         gen_tuple_const(NumHoles,rad(360)),
                         gen_tuple_const(NumHoles,'positive'), 1)
```

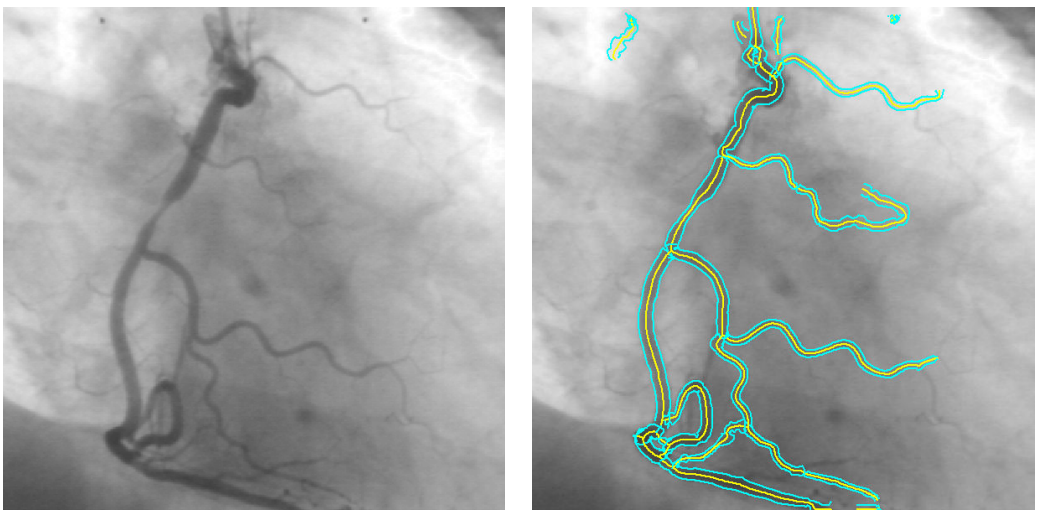
The diameters can easily be computed from the ellipse parameters and then be displayed in the image using `write_string` (see figure 7.2c).

```
for i := 0 to NumHoles-1 by 1
  write_string (WindowID, 'D1=' + 2*Ra[i])
  write_string (WindowID, 'D2=' + 2*Rb[i])
endfor
```

7.3.2 Angiography

Example: `examples\hdevelop\Filter\Lines\lines_gauss.dev`

The task of this example is to extract the blood vessels in the X-ray image of the heart depicted in figure 7.3. The vessels are emphasized by using a contrast medium. For the diagnosis it is important to extract the width of the vessels to determine locally narrowed parts (stenoses).



a) **b)**

Figure 7.3: (a) X-ray image of the heart; (b) extracted blood vessels.

The vessels are extracted using `lines_gauss`. The result of this operator are the centers of the vessels in

the form of XLD contours. Besides this, attributes are associated with the contour points, one of which is the local line width. This width is requested and displayed as contours.

```
lines_gauss (Angio, Lines, 2.2, 0, 0.8, 'dark', 'true', 'true', 'true')
count_obj (Lines, Number)
for I := 1 to Number by 1
  select_obj (Lines, Line, I)
  get_contour_xld (Line, Row, Col)
  get_contour_attrib_xld (Line, 'angle', Angle)
  get_contour_attrib_xld (Line, 'width_left', WidthL)
  get_contour_attrib_xld (Line, 'width_right', WidthR)
  RowR := Row+cos(Angle)*WidthR
  ColR := Col+sin(Angle)*WidthR
  RowL := Row-cos(Angle)*WidthL
  ColL := Col-sin(Angle)*WidthL
  disp_polygon (WindowID, RowL, ColL)
  disp_polygon (WindowID, RowR, ColR)
endfor
```

7.3.3 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\roads.dev
  Extract roads from aerial image
  → description here in the Solution Guide Basics on page 106
- examples\hdevelop\Applications\Barcode\get_rectangle_pose_barcode.dev
  Estimate 3D pose of bar codes
- examples\hdevelop\Applications\Calibration\3d-coordinates.dev
  Measure slanted object in world coordinates
- examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev
  Determine the pose of circles in 3D from their perspective 2D projections
- examples\hdevelop\Applications\Calibration\3d_position_of_rectangle.dev
  Estimate 3D pose of rectangular objects
- examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev
  Measure distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- examples\hdevelop\Applications\FA\circles.dev
  Fit circles into curved contour segments
  → description here in the Solution Guide Basics on page 25
- examples\hdevelop\Applications\FA\find_pads.dev
  Find pads on a die.
- examples\hdevelop\Applications\FA\rim.dev
  Inspect holes and characters on a rim

- examples\hdevelop\Applications\Medicine\angio.dev
  Extract blood vessels and their diameters from an angiogram
- examples\hdevelop\File\XLD\dxf_input_output.dev
  Write and read DXF files
- examples\hdevelop\Filter\Edges\edges_color_sub_pix.dev
  Extract edges with sub-pixel precision using color information
- examples\hdevelop\Filter\Edges\edges_sub_pix.dev
  Extract edges with sub-pixel precision
- examples\hdevelop\Filter\Lines\lines_color.dev
  Extract lines using color information
  → description here in the Solution Guide Basics on page 201
- examples\hdevelop\Filter\Lines\lines_facet.dev
  Extract lines using the facet model
- examples\hdevelop\Graphics\Output\disp_xld.dev
  Display an XLD object
- examples\hdevelop\Graphics\Output\dump_window_data.dev
  Dump the content of the graphics window to an image object
- examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev
  Rectify image of stadium to simulate overhead view
- examples\hdevelop\Tools\Calibration\change_radial_distortion_contours_xld.dev
  Eliminate radial distortions from extracted contours
- examples\hdevelop\Tools\Geometry\distance_pc.dev
  Determine the minimum and maximum radius of drill holes by calculating the distance between a point and a contour
- examples\hdevelop\XLD\Features\fit_ellipse_contour_xld.dev
  Approximate XLD contours with ellipses or elliptic arcs
- examples\hdevelop\XLD\Features\fit_ellipse_tooth_rim_xld.dev
  Approximate the contour of a tooth rim with an ellipse to find its center.
- examples\hdevelop\XLD\Features\fit_rectangle2_contour_xld.dev
  Detect errors in punched rectangular holes.
- examples\hdevelop\XLD\Transformation\clip_contours_xld.dev
  Clip an XLD contour
- examples\hdevelop\XLD\Transformation\crop_contours_xld.dev
  Crop an XLD contour
- examples\hdevelop\XLD\Transformation\gen_parallel_contour_xld.dev
  Compute the parallel contour of an XLD contour
- examples\hdevelop\XLD\Transformation\sort_contours_xld.dev
  Sort XLD contours by spatial position

- examples\hdevelop\XLD\Transformation\union_cocircular_contours_xld.dev
  Merge contours belonging to the same circle
- examples\hdevelop\XLD\Transformation\union_contours_xld.dev
  Find pads on a die by connecting collinear line segments
- examples\solution_guide\2d_measuring\measure_chip.dev
  Get extents and positions of rectangular chip components
  → description in the Solution Guide II-E on page 49
- examples\solution_guide\2d_measuring\measure_circles.dev
  Get radii and positions of circular shapes
  → description in the Solution Guide II-E on page 52
- examples\solution_guide\2d_measuring\measure_grid.dev
  Get junctions of a grid that separates keys
  → description in the Solution Guide II-E on page 42
- examples\solution_guide\2d_measuring\measure_metal_part_extended.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 39
- examples\solution_guide\2d_measuring\measure_metal_part_first_example.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 6
- examples\solution_guide\2d_measuring\measure_metal_part_id.dev
  Inspect metal part for missing or deviating circular shapes
  → description in the Solution Guide II-E on page 64
- examples\solution_guide\2d_measuring\measure_perspective_scratch.dev
  Measure the length of a scratch in world coordinates
  → description in the Solution Guide II-E on page 71
- examples\solution_guide\2d_measuring\measure_pump.dev
  Measure positions and radii of circular shapes
  → description in the Solution Guide II-E on page 54
- examples\solution_guide\2d_measuring\measure_screw.dev
  Measure several features of a screw
  → description in the Solution Guide II-E on page 31
- examples\solution_guide\3d_machine_vision\camera_calibration_exterior.dev
  Measure positions on a caliper rule using camera calibration
  → description in the Solution Guide II-F on page 41
- examples\solution_guide\3d_machine_vision\radial_distortion.dev
  Eliminate radial distortions
  → description in the Solution Guide II-F on page 48
- examples\solution_guide\basics\close_contour_gaps.dev
  Close gaps in extracted straight contours
  → description here in the Solution Guide Basics on page 105

- examples\solution_guide\basics\measure_metal_part.dev
  Inspect metal part by fitting lines and circles
  → description here in the Solution Guide Basics on page 103

C++

- examples\cpp\source\example9.cpp
  Extract roads from aerial images
- examples\mfc\FGMultiThreading\FGMultiThreading.cpp
  Using multiple threads to grab and process images in parallel

7.4 Selecting Operators

7.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

7.4.2 Radiometrically Calibrate Image(s)

Standard:

radiometric_self_calibration, lut_trans

7.4.3 Extract Edges Or Lines

Standard:

edges_sub_pix, derivate_gauss, lines_gauss, lines_facet

Advanced:

zero_crossing_sub_pix, edges_color_sub_pix, lines_color

7.4.4 Determine Contour Attributes

Standard:

get_contour_attrib_xld, get_contour_global_attrib_xld,
query_contour_attribs_xld, query_contour_global_attribs_xld

7.4.5 Transform Results Into World Coordinates

Standard:

`contour_to_world_plane_xld`

More operators for transforming results into world coordinates are described in the Solution Guide II-F.

7.4.6 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

7.5 Relation to Other Methods

7.5.1 Alternatives to Edge Extraction (Subpixel-Precise)

Subpixel Thresholding
Besides the subpixel-accurate edge and line extractors, HALCON provides a subpixel-accurate threshold operator called `threshold_sub_pix`. If the illumination conditions are stable, this can be a fast alternative.

Subpixel Point Extraction
In addition to the contour-based subpixel-accurate data, HALCON offers subpixel-accurate point operators for various applications. In the reference manual, these operators can be found in the chapter "Filter ▷ Points".

7.6 Tips & Tricks

7.6.1 Use of Domains (Regions of Interest)

The concept of domains (HALCON term for region of interest) is very important for contour extraction. With domains, the processing can be focused to a certain area in the image and can thus be sped up. The more the region in which edges or lines are extracted can be restricted, the faster and more robust the extraction will be. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

7.7 Advanced Topics

7.7.1 Contour Processing

Typically, the task is not finished by just extracting the contours and accessing the attributes. HALCON provides further processing like contour segmentation, feature extraction, or approximation. For more information see the method Contour Processing on page 97.

Chapter 8

Contour Processing

One of HALCON's powerful tool sets are the subpixel-accurate contours. Contours belong to the data type XLD (see Quick Guide in section 2.1.2.3 on page 18 for more information). These contours are typically the result of some kind of image processing and represent, e.g., the borders of objects. Figure 8.1a shows such edges overlaid on the original image; Figure 8.1b zooms into the rectangular area marked in Figure 8.1a and furthermore highlights the so-called control points of the contours with crosses. Here, you can clearly see the highly accurate positioning of the control points.

HALCON provides operators to perform advanced types of measurements with these contours. For example, the contours can be segmented into lines and circular or elliptic arcs (see Figure 8.1c). The parameters of these segments, e.g., their angle, center, or radius, can then be determined and used, e.g., in the context of a measuring task.

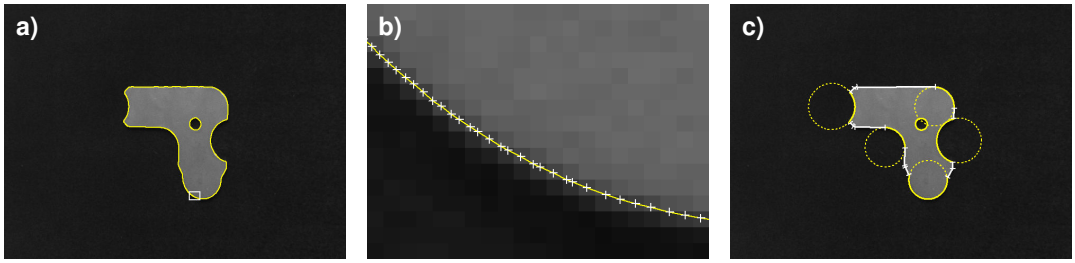
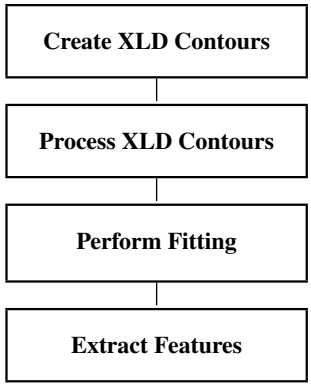


Figure 8.1: XLD contours: (a) edge contours, (b) zoom into rectangular area, (c) segmented lines and elliptic arcs.

The advantage of contour processing is twofold: First, its high accuracy enables reliable measurements. Secondly, the extensive and flexible set of operators provided for this data type enables you to solve problems that cannot be solved with classical methods like 1D measuring. More detailed information about contour processing can be found in the Solution Guide II-E.

8.1 Basic Concept

The processing of contours consists of multiple steps that can be combined in a flexible way.



8.1.1 Create XLD Contours

The most common way to create XLD contours is to apply one of the subpixel-accurate extraction operators described for the method Extract Edges Or Lines on page 87. As an alternative, an edge filter with some post-processing can be used. The resulting regions are then converted to XLD contours. Please note that this approach is only pixel-accurate. For more information about this approach see the method Edge Extraction (Pixel-Precise) on page 73.

8.1.2 Process XLD Contours

Typically, only certain contours of an object are used for an inspection task. One possibility to restrict the extraction of contours to the desired ones is to use a well-fitting region of interest as, e.g., depicted in figure 8.2a: The rectangular ROI just covers the upper part of the blades. When applying an edge extractor, exactly one contour on each side of the objects is found.

In many cases, however, not only the desired contours are extracted. An example is depicted in figure 8.2b, where the ROI was chosen too large. Thus, the contours must be processed to obtain the desired parts of the contours. In the example, the contours are segmented into parts and only parallel segments with a certain length are selected (see the result in figure 8.2c).

Another reason for processing contours occurs if the extraction returns unwanted contours caused by noise or texture or if there are gaps between contours because of a low contrast or contour intersections.

8.1.3 Perform Fitting

Having obtained contour segments that represent a line, a rectangle, or a circular or elliptic arc, you can determine the corresponding parameters, e.g., the coordinates of the end points of a line or the center and

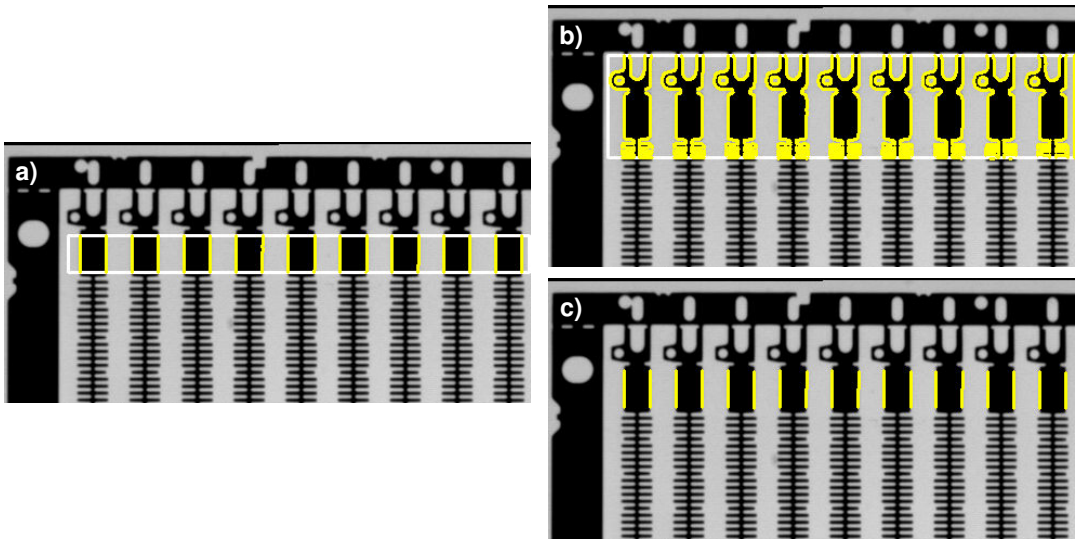


Figure 8.2: Selecting the desired contours: (a) exactly fitting ROI, (b) too many contours because of too large ROI, (c) result of post-processing the contours from (b).

radius of a circle, by calling one of the fitting operators. Their goal is to approximate the input contour as closely as possible to a line, rectangle, or a circular or elliptic arc. Because the used minimization algorithms are very advanced and all contour points are used for the process, the parameters can be calculated very reliably.

8.1.4 Extract Features

From both raw contours and processed contour parts features can be determined. Some of these consider the contour as a linear object. Others treat a contour as the outer boundary of an object. Obviously, the center of gravity makes sense only for a closed object, whereas the curvature is a feature of a linear object.

8.1.5 A First Example

The following program is an example for the basic concept of contour processing. It shows how short segments returned by the line extractor can be grouped to longer ones.

First, an image is acquired from file using `read_image`. The task is to extract the roads, which show up as thin bright lines in the image. For this the operator `lines_gauss` is used. When we look at the result of the contour extraction in figure 8.3a, we see that a lot of unwanted small segments are extracted. They can be suppressed easily by calling `select_contours_xld` with a minimum contour length. A further problem is that some roads are split into more than one segment. They can be combined with the operator `union_collinear_contours_xld`. Looking at the result in figure 8.3b, we see that many fragments

have been combined along straight road parts. In curves this method fails because the orientation of the segments differs too much.

```
read_image (Image, 'mreut4_3')
lines_gauss (Image, Lines, 1.5, 2, 8, 'light', 'true', 'true', 'true')
select_contours_xld (Lines, LongContours, 'contour_length', 15, 1000, 0, 0)
union_collinear_contours_xld (LongContours, UnionContours, 30, 2, 9, 0.7,
                              'attr_keep')
```

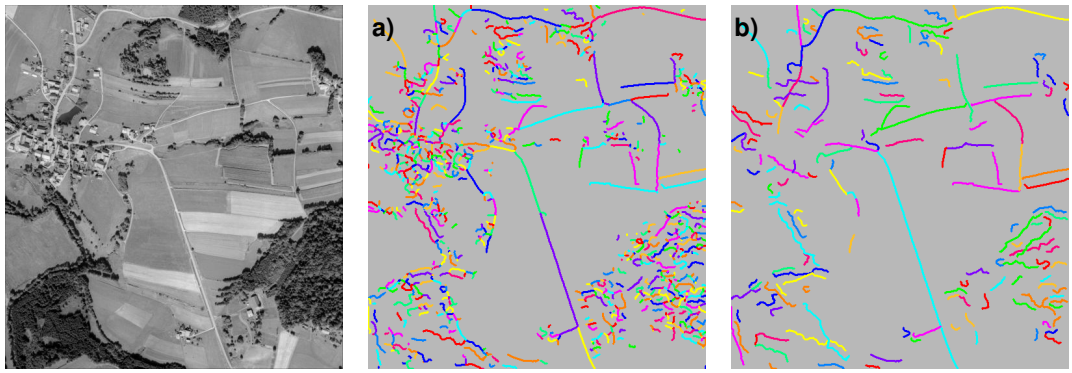


Figure 8.3: Processing XLD contours, (a) extracted contours, (b) processed contours.

8.2 Extended Concept

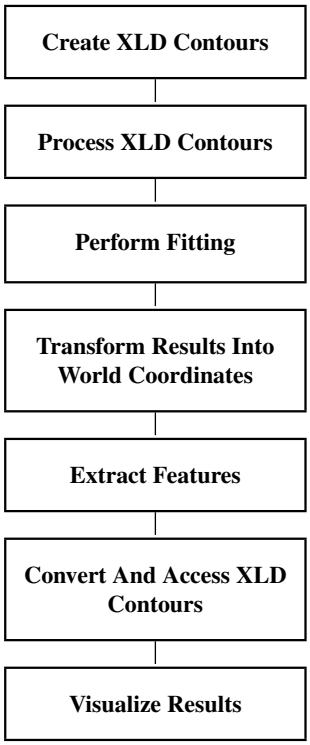
In addition to the standard contour processing, HALCON provides other tools that can be added. Typical examples for these are camera calibration, geometric transformations, or type conversions. With these, the contour methods can be integrated into the overall vision task.

8.2.1 Create XLD Contours

The standard method to create contours is to call a contour extraction operator. Contour extraction for edges is performed with `edges_sub_pix`, `edges_color_sub_pix`, or `zero_crossing_sub_pix`. Lines are extracted using `lines_gauss`, `lines_facet`, or `lines_color`. For subpixel blob analysis the operator `threshold_sub_pix` can be used. These operators are described in more detail with the method Edge Extraction (Subpixel-Precise) on page 85.

If pixel-accuracy is sufficient, you can use an edge filter (like `sobel_amp` or `edges_image`) or a line filter (like `bandpass_image`) followed by thresholding and thinning. The resulting elongated regions are then converted into XLD contours with the operator `gen_contours_skeleton_xld`. For more information on this approach see the method Edge Extraction (Pixel-Precise) on page 73.

Contours can also be synthesized from different sources, e.g., CAD data, user interaction, or measuring. Having obtained the coordinates of the control points from such a source, the operators



gen_contour_polygon_xld and gen_contour_polygon_rounded_xld convert them to XLD contours. You can also draw XLD contours interactively with the operators draw_xld and draw_xld_mod.

Finally, the border of regions can be converted into XLD contours. The corresponding operator is called gen_contour_region_xld.

8.2.2 Process XLD Contours

The first method to segment contours is to call segment_contours_xld. This operator offers various modes: Splitting into line segments, linear and circular segments, or linear and elliptic segments. The individual contour segments can then be selected with select_obj and passed to one of the fitting operators described with the step Perform Fitting on page 98. Whether a contour segment represents a line, a circular, or an elliptic arc can be queried via the global contour attribute 'cont_approx' using the operator get_contour_global_attrib_xld.

If only line segments are needed, you can use the combination of gen_polygons_xld followed by split_contours_xld. The behavior is similar to using segment_contours_xld. The main difference is the possible postprocessing: When applying gen_polygons_xld, a so-called XLD polygon is generated. This is a different data type, which represents the initial step for grouping of segments to parallel lines.

An important step during contour processing is the suppression of irrelevant contours. This can be accomplished with the operator select_shape_xld, which provides almost 30 different shape features.

By specifying the desired minimum and maximum value and possibly combining multiple features, contours can be selected very flexibly. As an alternative, you can use the operator select_contours_xld, which offers typical features of linear structures. Finally, the operator select_xld_point can be used in combination with mouse functions to interactively select contours.

If there are gaps within a contour, the pieces are treated as separate objects, which makes further processing and feature extraction difficult. You can merge linear segments with the operators union_collinear_contours_xld or union_straight_contours_xld. Additionally, you can also merge adjacent contours (union_adjacent_contours_xld) or contours that lie on the same circle (union_cocircular_contours_xld). To handle contours with a complex shape you must first segment them into linear, circular, or elliptic segments (see above).

HALCON also provides an operator for general shape modifications: shape_trans_xld. With this operator you can, e.g., transform the contour into its surrounding circle, convex hull, or surrounding rectangle. Further, for closed contours or polygons, set theoretical operations can be applied to combine contours. For example, with intersection_closed_contours_xld you can intersect the regions that are enclosed by the closed contours, with difference_closed_contours_xld you can calculate the difference between the enclosed regions, or with union2_closed_contours_xld you can merge the enclosed regions.

8.2.3 Perform Fitting

With the operator fit_line_contour_xld you can determine the parameters of a line segment. The operator provides different optimization methods, most of which are suppressing outliers. It returns the coordinates of the start and the end point of the fitted line segment and the normal form of the line. To visualize the results, you can use the operator gen_contour_polygon_xld.

To fit a rectangle into a contour, the operator fit_rectangle2_contour_xld can be used. It provides various optimization methods as well. The returned parameters comprise mainly the center position, the extent, and the orientation of the rectangle. To generate the obtained rectangle for a visualization, you can use the operator gen_rectangle2_contour_xld.

For the fitting of circular and elliptic segments the operators fit_circle_contour_xld and fit_ellipse_contour_xld are available. They also provide various optimization methods. For a circular segment the center and the radius are returned together with the angle range of the visible part. In addition, a second radius and the orientation of the main axis are returned for elliptic segments. To visualize the results of both operators, you can use the operator gen_ellipse_contour_xld.

8.2.4 Transform Results Into World Coordinates

As a post-processing step, it may be necessary to correct the contours, e.g., to remove lens distortions, or to transform the contours into a 3D world coordinate system in order to extract dimensional features in world units. Such a transformation is based on calibrating the camera. After the calibration, you simply call the operator contour_to_world_plane_xld to transform the contours.

How to transform contours into world coordinates is described in detail in the Solution Guide II-F in section 3.2 on page 44.

8.2.5 Extract Features

HALCON offers various operators to access the feature values. Commonly used shape features are calculated by area_center_xld, compactness_xld, convexity_xld, eccentricity_xld, diameter_xld, and orientation_xld. The hulls of the contours can be determined with smallest_circle_xld or smallest_rectangle2_xld. Features based on geometric moments are calculated, e.g., by moments_xld.

8.2.6 Convert And Access XLD Contours

Finally, it might be necessary to access the raw data of the contours or to convert contours into another data type, e.g., into a region.

You can access the coordinates of the control points with the operator get_contour_xld. It returns the row and column coordinates of all control points of a contour in two tuples of floating-point values. In case of a contour array (tuple), you must loop over all the contours and select each one using select_obj.

To convert contours to regions, simply call the operator gen_region_contour_xld. The operator paint_xld paints the contour with anti-aliasing into an image.

The operators for edge and line extraction not only return the XLD contours but also so-called attributes. Attributes are numerical values; they are associated either with each control point (called contour attribute) or with each contour as a whole (global contour attribute). The operators get_contour_attrib_xld and get_contour_global_attrib_xld enable you to access these values by specifying the attribute name. More information on this topic can be found in the description of the step Determine Contour Attributes on page 88.

8.2.7 Visualize Results

Finally, you might want to display the images and the contours.

For detailed information see the description of this method on page 269.

8.3 Programming Examples

This section gives a brief introduction to using HALCON for contour processing.

8.3.1 Measuring Lines and Arcs

Example: examples\solution_guide\basics\measure_metal_part.dev

The first example shows how to segment a contour into lines and (circular) arcs and how to determine the corresponding parameters. Figure 8.4 shows the final result of the fitted primitives overlaid on the input image.

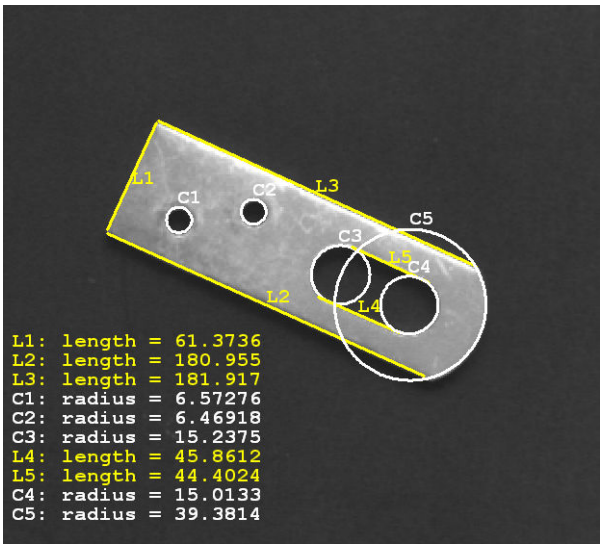


Figure 8.4: Fitted lines and circles.

As the initial step, the contours of the metal part are extracted using the operator edges_sub_pix. The resulting contours are segmented into lines and circular arcs and sorted according to the position of their upper left corner.

```
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 40, 90)
segment_contours_xld (Edges, ContoursSplit, 'lines_circles', 6, 4, 4)
sort_contours_xld (ContoursSplit, SortedContours, 'upper_left', 'true',
                   'column')
```

Then, lines and circles are fitted to the extracted segments. As already noted, the individual segments must be accessed inside a loop. For this, first their total number is determined with count_obj. Inside the loop, the individual segments are selected with the operator select_obj. Then, their type (line or circular arc) is determined by accessing a global attribute with get_contour_global_attrib_xld. Depending on the result, either a circle or a line is fitted. For display purposes, circles and lines are created using the determined parameters. Furthermore, the length of the lines is computed with the operator distance_pp.

```
count_obj (SortedContours, NumSegments)
for i := 1 to NumSegments by 1
  select_obj (SortedContours, SingleSegment, i)
  get_contour_global_attrib_xld (SingleSegment, 'cont_approx', Attrib)
  if (Attrib = 1)
    fit_circle_contour_xld (SingleSegment, 'atukey', -1, 2, 0, 5, 2,
                            Row, Column, Radius, StartPhi, EndPhi,
                            PointOrder)
    gen_ellipse_contour_xld (ContEllipse, Row, Column, 0, Radius,
                             Radius, 0, rad(360), 'positive', 1.0)
  else
    fit_line_contour_xld (SingleSegment, 'tukey', -1, 0, 5, 2,
                          RowBegin, ColBegin, RowEnd, ColEnd, Nr, Nc,
                          Dist)
    gen_contour_polygon_xld (Line, [RowBegin,RowEnd], [ColBegin,
                             ColEnd])
    distance_pp (RowBegin, ColBegin, RowEnd, ColEnd, Length)
                 (ColBegin+ColEnd)/2)
  endif
endfor
```


8.3.2 Close gaps in a contour

Example: examples\solution_guide\basics\close_contour_gaps.dev

The second example demonstrates how to close gaps in an object contour (see figure 8.5). The example is based on synthetic data. Instead of using a real image, a light gray square on a dark gray background is generated and a part of its boundary is blurred.

```
gen_rectangle1 (Rectangle, 30, 20, 100, 100)
region_to_bin (Rectangle, BinImage, 130, 100, 120, 130)
rectangle1_domain (BinImage, ImageReduced, 20, 48, 40, 52)
mean_image (ImageReduced, SmoothedImage, 15, 15)
paint_gray (SmoothedImage, BinImage, Image)
```

The extraction of contours with edges_sub_pix thus results in an interrupted boundary (see figure 8.5a). Note that the edge extraction is restricted to the inner part of the image, otherwise edges would be extracted at the boundary of the image.

```
rectangle1_domain (BinImage, ImageReduced, 20, 48, 40, 52)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 1.1, 22, 30)
```

A suitable operator for closing gaps in linear segments is union_collinear_contours_xld. Before we can apply this operator, some pre-processing is necessary: First, the contours are split into linear segments using segment_contours_xld. Then, regress_contours_xld is called to determine the regression parameters for each segment. These parameters are stored with each contour and could be accessed with get_regress_params_xld. Finally, union_collinear_contours_xld is called. Its result is depicted in figure 8.5b.

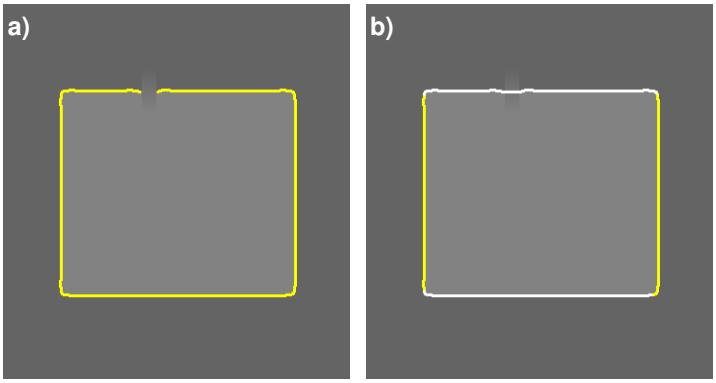


Figure 8.5: Original edges and result of grouping process.

```
segment_contours_xld (Edges, LineSegments, 'lines', 5, 4, 2)
regress_contours_xld (LineSegments, RegressContours, 'no', 1)
union_collinear_contours_xld (RegressContours, UnionContours, 10, 1, 2,
                              0.1, 'attr_keep')
```

8.3.3 Extract Roads

Example: `examples\hdevelop\Applications\Aerial\roads.dev`

The task of this example is to extract the roads in the aerial image depicted in figure 8.6

Figure 8.6: (a) Original image; (b) zoomed image part; (c) extracted roads.

The programs starts by first extracting lines on a reduced scale. These lines correspond very well to roads.

```
threshold (Mreut43, Bright, 160, 255)
reduce_domain (Mreut43, Bright, Mreut43Bright)
lines_gauss (Mreut43Bright, RoadCenters, 1.2, 5, 14, 'light', 'true',
             'true', 'true')
```

To eliminate wrong candidates, edges are extracted on a higher scale. For the road extraction it is assumed that a road consists of two parallel edges with homogeneous gray values and a line segment in between. Using some contour processing operators, this model is refined step by step.

```
edges_image (Part, PartAmp, PartDir, 'mderiche2', 0.3, 'nms', 20, 40)
threshold (PartAmp, EdgeRegion, 1, 255)
clip_region (EdgeRegion, ClippedEdges, 2, 2, PartWidth - 3, PartHeight - 3)
skeleton (ClippedEdges, EdgeSkeleton)
gen_contours_skeleton_xld (EdgeSkeleton, RoadEdges, 1, 'filter')
gen_polygons_xld (RoadEdges, RoadEdgePolygons, 'ramer', 2)
gen_parallels_xld (RoadEdgePolygons, ParallelRoadEdges, 10, 30, 0.15,
                   'true')
mod_parallels_xld (ParallelRoadEdges, Part, ModParallelRoadEdges,
                   ExtParallelRoadEdges, 0.3, 160, 220, 10)
combine_roads_xld (RoadEdgePolygons, ModParallelRoadEdges,
                   ExtParallelRoadEdges, RoadCenterPolygons, RoadSides,
                   rad(40), rad(20), 40, 40)
```

8.3.4 Other Examples

HDevelop

- examples\hdevelop\Applications\Barcode\get_rectangle_pose_barcode.dev
  Estimate 3D pose of bar codes
- examples\hdevelop\Applications\Calibration\3d-coordinates.dev
  Measure slanted object in world coordinates
- examples\hdevelop\Applications\Calibration\3d_position_of_rectangle.dev
  Estimate 3D pose of rectangular objects
- examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev
  Measure distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- examples\hdevelop\Applications\FA\circles.dev
  Fit circles into curved contour segments
  → description here in the Solution Guide Basics on page 25
- examples\hdevelop\Applications\FA\find_pads.dev
  Find pads on a die.

- examples\hdevelop\Applications\FA\matching_multi_channel_yogurt.dev
  Find yogurts of different flavors using color shape-based matching
- examples\hdevelop\Applications\FA\pm_world_plane.dev
  Recognize planar objects using shape-based matching in perspectively distorted images
- examples\hdevelop\Applications\FA\rim.dev
  Inspect holes and characters on a rim
- examples\hdevelop\Applications\Medicine\angio.dev
  Extract blood vessels and their diameters from an angiogram
- examples\hdevelop\Applications\Sequences\optical_flow_hydraulic_engineering.dev
  Get position, speed, and movement direction of particles using optical flow
- examples\hdevelop\File\XLD\dxf_input_output.dev
  Write and read DXF files
- examples\hdevelop\Filter\Lines\lines_color.dev
  Extract lines using color information
  → description here in the Solution Guide Basics on page 201
- examples\hdevelop\Filter\Lines\lines_gauss.dev
  Extract lines and their widths
  → description here in the Solution Guide Basics on page 90
- examples\hdevelop\Graphics\Output\disp_xld.dev
  Display an XLD object
- examples\hdevelop\Graphics\Output\dump_window_data.dev
  Dump the content of the graphics window to an image object
- examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev
  Rectify image of stadium to simulate overhead view
- examples\hdevelop\Tools\Calibration\change_radial_distortion_contours_xld.dev
  Eliminate radial distortions from extracted contours
- examples\hdevelop\Tools\Geometry\distance_cc_min.dev
  Calculate the distance between two contours
- examples\hdevelop\XLD\Creation\gen_contour_region_xld.dev
  Extract the contours of regions as XLD objects
- examples\hdevelop\XLD\Features\features_points_xld.dev
  Compare *_xld with corresponding *_points_xld operators
- examples\hdevelop\XLD\Features\fit_circle_contour_xld.dev
  Approximate an XLD contour with a circle
- examples\hdevelop\XLD\Features\fit_ellipse_contour_xld.dev
  Approximate XLD contours with ellipses or elliptic arcs
- examples\hdevelop\XLD\Features\fit_ellipse_tooth_rim_xld.dev
  Approximate the contour of a tooth rim with an ellipse to find its center.

- examples\hdevelop\XLD\Features\fit_line_contour_xld.dev
  Approximate XLD contours with line segments
- examples\hdevelop\XLD\Features\fit_rectangle2_contour_xld.dev
  Detect errors in punched rectangular holes.
- examples\hdevelop\XLD\Features\select_xld_point.dev
  Select XLD contours containing a specified point
- examples\hdevelop\XLD\Features\shape_trans_xld.dev
  Transform contours into various standard shapes
- examples\hdevelop\XLD\Features\statistics_points_xld.dev
  Analyze 2D statistical data using *_points_xld operators
- examples\hdevelop\XLD\Features\test_self_intersection_xld.dev
  Test if an XLD contour intersects itself
- examples\hdevelop\XLD\Features\test_xld_point.dev
  Test if an XLD contour contains a specific point
- examples\hdevelop\XLD\Sets\difference_closed_contours_xld.dev
  Demonstrate the use of difference_closed_contours_xld
- examples\hdevelop\XLD\Sets\difference_closed_polygons_xld.dev
  Demonstrate the use of difference_closed_polygons_xld
- examples\hdevelop\XLD\Sets\intersection_closed_contours_xld.dev
  Demonstrate the use of intersection_closed_contours_xld
- examples\hdevelop\XLD\Sets\intersection_closed_polygons_xld.dev
  Demonstrate the use of intersection_closed_polygons_xld
- examples\hdevelop\XLD\Sets\symm_difference_closed_contours_xld.dev
  Demonstrate the use of symm_difference_closed_contours_xld
- examples\hdevelop\XLD\Sets\symm_difference_closed_polygons_xld.dev
  Demonstrate the use of symm_difference_closed_polygons_xld
- examples\hdevelop\XLD\Sets\union2_closed_contours_xld.dev
  Demonstrate the use of union2_closed_contours_xld
- examples\hdevelop\XLD\Sets\union2_closed_polygons_xld.dev
  Demonstrate the use of union2_closed_contours_xld
- examples\hdevelop\XLD\Transformation\clip_contours_xld.dev
  Clip an XLD contour
- examples\hdevelop\XLD\Transformation\close_contours_xld.dev
  Close XLD contours
- examples\hdevelop\XLD\Transformation\crop_contours_xld.dev
  Crop an XLD contour
- examples\hdevelop\XLD\Transformation\gen_parallel_contour_xld.dev
  Compute the parallel contour of an XLD contour

- examples\hdevelop\XLD\Transformation\projective_trans_contour_xld.dev
  Rotate an XLD contour in 3D using hom_mat3d_project and projective_trans_region
- examples\hdevelop\XLD\Transformation\sort_contours_xld.dev
  Sort XLD contours by spatial position
- examples\hdevelop\XLD\Transformation\union_cocircular_contours_xld.dev
  Merge contours belonging to the same circle
- examples\hdevelop\XLD\Transformation\union_contours_xld.dev
  Find pads on a die by connecting collinear line segments
- examples\solution_guide\2d_measuring\measure_chip.dev
  Get extents and positions of rectangular chip components
  → description in the Solution Guide II-E on page 49
- examples\solution_guide\2d_measuring\measure_circles.dev
  Get radii and positions of circular shapes
  → description in the Solution Guide II-E on page 52
- examples\solution_guide\2d_measuring\measure_grid.dev
  Get junctions of a grid that separates keys
  → description in the Solution Guide II-E on page 42
- examples\solution_guide\2d_measuring\measure_metal_part_extended.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 39
- examples\solution_guide\2d_measuring\measure_metal_part_first_example.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 6
- examples\solution_guide\2d_measuring\measure_metal_part_id.dev
  Inspect metal part for missing or deviating circular shapes
  → description in the Solution Guide II-E on page 64
- examples\solution_guide\2d_measuring\measure_perspective_scratch.dev
  Measure the length of a scratch in world coordinates
  → description in the Solution Guide II-E on page 71
- examples\solution_guide\2d_measuring\measure_pump.dev
  Measure positions and radii of circular shapes
  → description in the Solution Guide II-E on page 54
- examples\solution_guide\2d_measuring\measure_screw.dev
  Measure several features of a screw
  → description in the Solution Guide II-E on page 31
- examples\solution_guide\3d_machine_vision\camera_calibration_exterior.dev
  Measure positions on a caliper rule using camera calibration
  → description in the Solution Guide II-F on page 41
- examples\solution_guide\3d_machine_vision\handeye_stationarycam_grasp_nut.dev
  Calculate pose for grasping a nut based on results of hand-eye calibration for a stationary camera

→ description in the Solution Guide II-F on page 120

- examples\solution_guide\3d_machine_vision\radial_distortion.dev
  Eliminate radial distortions
  → description in the Solution Guide II-F on page 48
- examples\solution_guide\basics\critical_points.dev
  Locate saddle point markers in an image
  → description here in the Solution Guide Basics on page 23

C++

- examples\cpp\source\example10.cpp
  Working with contour data and zooming results
- examples\cpp\source\example9.cpp
  Extract roads from aerial images

8.4 Selecting Operators

8.4.1 Create XLD Contours

Standard:

gen_contour_polygon_xld, gen_contour_region_xld, gen_ellipse_contour_xld, gen_rectangle2_contour_xld, draw_xld, draw_xld_mod

Advanced:

gen_contour_polygon_rounded_xld

Further operators can be found in the following places: detailed operator list for the step Extract Edges Or Lines on page 94, operator list for the method Edge Extraction (Pixel-Precise) (see section 6.4 on page 81).

8.4.2 Process XLD Contours

Standard:

segment_contours_xld, gen_polygons_xld, split_contours_xld, select_shape_xld, select_contours_xld, select_xld_point, union_collinear_contours_xld, union_straight_contours_xld, union_adjacent_contours_xld, union_cocircular_contours_xld, shape_trans_xld, intersection_closed_contours_xld, intersection_closed_polygons_xld, difference_closed_contours_xld, difference_closed_polygons_xld, union2_closed_contours_xld, union2_closed_polygons_xld

Advanced:

union_collinear_contours_ext_xld, symm_difference_closed_contours_xld, symm_difference_closed_polygons_xld

8.4.3 Perform Fitting

Standard:

fit_line_contour_xld, fit_circle_contour_xld, fit_ellipse_contour_xld, fit_rectangle2_contour_xld

8.4.4 Transform Results Into World Coordinates

Standard:

contour_to_world_plane_xld, change_radial_distortion_contours_xld

More operators for transforming results into world coordinates are described in the Solution Guide II-F.

8.4.5 Extract Features

Standard:

area_center_xld, orientation_xld, smallest_circle_xld, smallest_rectangle1_xld, smallest_rectangle2_xld, compactness_xld, convexity_xld, diameter_xld, eccentricity_xld

A full list of operators can be found in the Reference Manual in the chapter "XLD ▷ Features".

8.4.6 Convert And Access XLD Contours

Standard:

get_contour_xld, gen_region_contour_xld

Advanced:

paint_xld

More information on operators for accessing XLD data can be found in the Reference Manual in the chapter "XLD ▷ Access". Operators for determining contour attributes can be found in the detailed operator list for the step Determine Contour Attributes on page 94.

8.4.7 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

8.5 Relation to Other Methods

8.5.1 Alternatives to Contour Processing

Line Processing

A very basic alternative to contour processing are the operators for line processing. In this context, lines are treated as tuples of start and end points. The extraction can, e.g., be performed with `detect_edge_segments`. Of course, XLD polygons can also be converted into this type of lines. Operators for processing this type of lines can be found in the Reference Manual in the chapter "Lines".

8.6 Advanced Topics

8.6.1 Line Scan Cameras

In general, line scan cameras are treated like normal area sensors. But in some cases, not single images but an infinite sequence of images showing objects, e.g., on a conveyor belt, have to be processed. In this case the end of one image is the beginning of the next one. This means that contours that partially lie in both images must be combined into one contour. For this purpose HALCON provides the operator `merge_cont_line_scan_xld`. This operator is called after the processing of one image and combines the current contours with those of previous images. For more information see Solution Guide II-A.

Chapter 9

Template Matching

The idea of template matching is quite simple: In a training image a so-called template is presented. The system derives a model from this template. This model is then used to locate objects that look "similar" to the template in search images. Depending on the selected method, this approach is able to handle changes in illumination, clutter, varying size, position, and rotation, or even relative movement of parts of the template.

The advantage of template matching is its ease of use combined with great robustness and flexibility. Template matching does not require any kind of segmentation of the desired objects. Objects can be located even if they are overlapped by other objects. Furthermore, template matching has only a few parameters. Even these can be determined automatically in most cases. This makes this method especially attractive for applications where the end user only has little skills in machine vision.

HALCON offers different methods for template matching. The selection depends on the image data and the task to be solved.

- The *gray-value-based matching* is the classical method. It can be used if the gray values inside the object do not vary and if there are no missing parts and no clutter. The method can handle single instances of objects, which can appear rotated in the search image.
- The *shape-based matching* represents the state of the art in machine vision. Instead of using the gray values, features along contours are extracted and used both for the model generation and the matching. This has the effect that this method is invariant to changes in illumination and variations of the objects gray values. It can handle missing object parts, clutter, and noise. Furthermore, multiple instances can be found and multiple models can be used at the same time. The method allows the objects to be rotated and scaled and can be applied also for multi-channel images.

  How to use shape-based matching is described in detail in the Solution Guide II-B.
- The *component-based matching* can be considered as a high-level shape-based matching: The enhancement is that an object can consist of multiple parts that can move (rotate and translate) relative to each other. A simple example of this is a pair of pliers. Logically this is considered as one object, but physically it consists of two parts. The component-based matching allows handling such a compound object in one search step. The advantage is an improved execution time and increased robustness compared to handling the parts as distinct models.

- The *correlation-based matching* is based on gray values and a normalized cross correlation. Its operators are used similar to the operators of shape-based matching. The advantage over shape-based matching is that also defocus, texture, or slightly deformed shapes can be handled. On the other hand, the object may only be rotated but not scaled, and the approach is limited to gray-value images. Additionally, the approach is sensitive to occlusion, clutter, and non-linear illumination changes. But in contrast to the classical gray-value-based matching, linear illumination changes can be coped with.
- The *point-based matching* has the intention to combine two overlapping images. This is done by first extracting significant points in both images. These points are the input for the actual matching process. The result of the matching is a mapping from one image to the other, allowing translation, rotation, scaling, and perspective distortions. This mapping is typically used to combine the two images into a single, larger one. Of course, one image can also be treated as a template and the other image as showing an instance of the object that should be found. The advantage is the ability to handle perspective distortions without calibration. The disadvantage is the increased execution time, which comes mainly from the extraction of the significant points.

Figure 9.1 summarizes the characteristics of the different template matching approaches and helps you to select the appropriate approach for your task.

9.1 Basic Concept

Template matching is divided into the following parts:

9.1.1 Acquire Image(s)

Both for training and matching, first an image is acquired.

For detailed information see the description of this method on page 13.

9.1.2 Train Model

To create a matching model, first a region of interest that covers the template in the training image must be specified. Only those parts of the image that are really significant and stable should be used for training. The input for the training operator is the reduced image together with control parameters. The handle of the model is the output of the training. The model will then be used for immediate search or stored to file.

9.1.3 Find Model

Having created (or loaded) a model, it can now be used for locating objects in the image. Each method offers specific methods to perform this task. If one or multiple objects are found, their poses (position, rotation, and scaling) together with a score are returned. These values can already be the desired result or serve as input for the next step of the vision process, e.g., for aligning regions of interest.

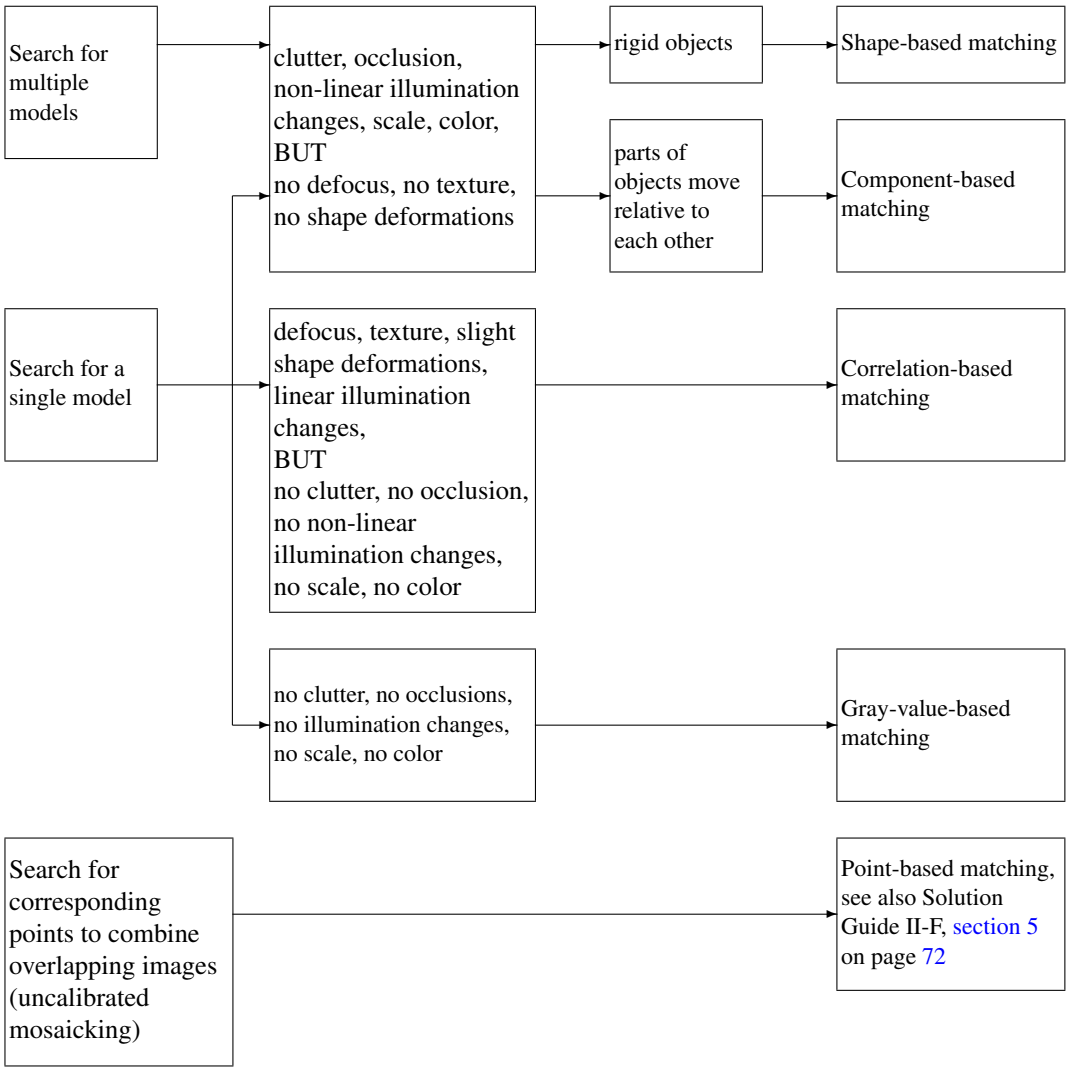


Figure 9.1: Guide to the different matching approaches.

9.1.4 Destroy Model

When you no longer need the matching model you should destroy it. For example, for shape-based matching you destroy it using `clear_shape_model`.

9.1.5 A First Example

An example for this basic concept is the following program, which shows all necessary steps from model generation to object finding using shape-based matching.

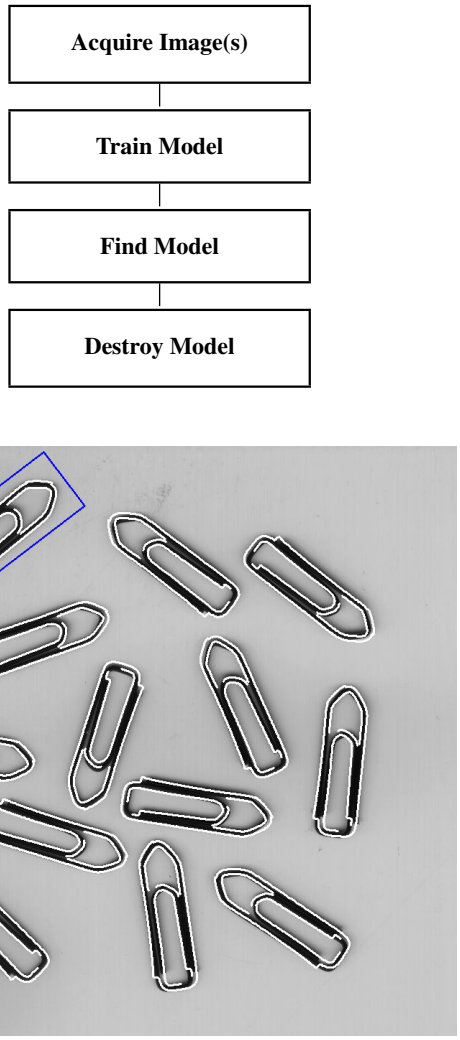


Figure 9.2: Finding all clips in the image.

A training image is acquired from file. A region is generated as region of interest, covering one of the clips in the image. After combining the region with the image, it is used as input for the training operator `create_shape_model`. To keep the example simple, the same image is used to test the matching by searching for all clips with `find_shape_model`. For visualization purposes the model contours are accessed, moved to the corresponding positions, and overlaid on the image. Finally, the model is cleared to release the memory.

```
read_image (Image, 'clip')
gen_rectangle2 (ROI, 124, 181, 0.653, 129, 47)
reduce_domain (Image, ROI, ImageReduced)
create_shape_model (ImageReduced, 0, 0, rad(360), 0, 'no_pregeneration',
                    'use_polarity', 40, 10, ModelID)
find_shape_model (Image, ModelID, 0, rad(360), 0.7, 13, 0.5,
                  'interpolation', 0, 0.9, Row, Column, Angle, Score)
get_shape_model_contours (ModelContours, ModelID, 1)
for i := 0 to |Row|-1 by 1
  vector_angle_to_rigid (0, 0, 0, Row[i], Column[i], Angle[i], HomMat2D)
  affine_trans_contour_xld (ModelContours, ContoursAffinTrans, HomMat2D)
endfor
clear_shape_model (ModelID)
```

9.2 Extended Concept

In many cases, template matching will be more complex than in the example above. Reasons for this are, e.g., advanced training using a synthetic template, searching with multiple models at the same time, or using the matching results as input for further processing like alignment.

Figure 9.3 shows the major steps. The first part is offline and consists of the training of the model, using different input sources that depend on the application. The model can then be stored to be loaded again for later use. For the matching itself, one or more models are used to search for objects in images. The results are the poses, i.e., the position, rotation, and scale of the found objects.

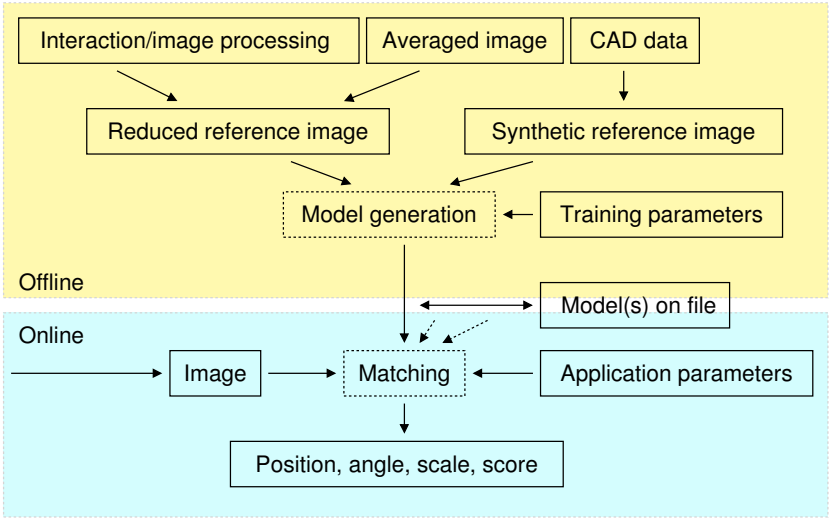
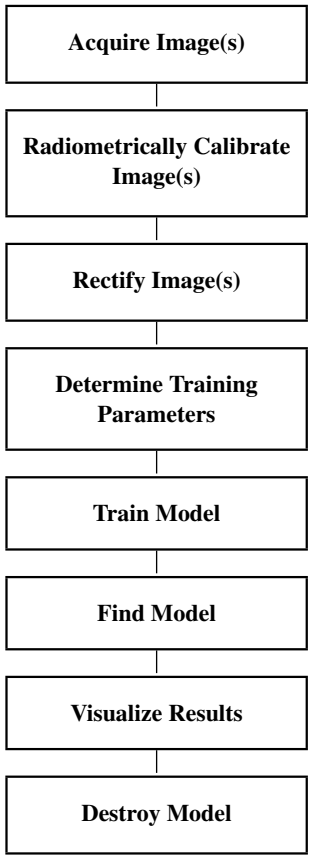


Figure 9.3: Overview of the matching process.



9.2.1 Radiometrically Calibrate Image(s)

To allow high-accuracy matching, the camera should have a linear response function, i.e., the gray values in the images should depend linearly on the incoming energy. Since some cameras do not have a linear response function, HALCON provides the so-called radiometric calibration (gray value calibration): With the operator `radiometric_self_calibration` you can determine the inverse response function of the camera (offline) and then apply this function to the images using `lut_trans` before performing the matching.

9.2.2 Rectify Image(s)

As a preprocessing step, it may be necessary to rectify the image, e.g., to remove lens distortions or to transform the image into a reference point of view. This will allow using template matching even if the camera is looking from the side onto the object plane or the surface is a cylinder.

Detailed information about rectifying images can be found in the Solution Guide II-F in section 3.3 on page 49.

9.2.3 Determine Training Parameters

To make the use of the matching as easy as possible it is necessary to determine the training parameters automatically. Because the shape-based matching is used most frequently, this section focuses on this method. In addition, the shape-based matching is the basis for the component-based matching, and therefore has very similar parameters.

The operators create_scaled_shape_model, create_shape_model, and create_aniso_shape_model allow to use the value 0 for many parameters. This has the effect that the values are determined automatically. The allowed ranges of orientations and scales are typically known from the setup of the system. They can also be determined easily in HDevelop using the assistant provided via the menu item Assistants > Matching. If 'no_pregeneration' is used for the parameter Optimization wide ranges can be used and restricted during the finding process. The other training parameters (Contrast, MinContrast) can be determined also by using the assistant. If an automatic method is needed, please contact your local distributor. Their support can provide application-specific procedures to determine the parameters.

9.2.4 Train Model

Creating a model differs from method to method, but several rules hold for all methods.

- You can use both real and synthetic images for the training. Using real images is easier. However, if no such image is available or if the real images are too noisy you can create a synthetic image, e.g., with the operators gen_contour_polygon_rounded_xld, gen_image_const, and paint_xld.

  If the training image is too noisy, you can enhance it, e.g., by smoothing noisy areas or by averaging multiple templates into a single one.

- To speed up the matching process, a so-called image pyramid is created, consisting of the original, fullsized image and a set of downsampled images. The model is then created and searched on the different pyramid levels. You can view the image pyramid and the corresponding models for shape-based matching with the matching assistant in HDevelop.

- Each model (except for the point-based matching) is manipulated via a handle. This handle points to the real data, which can consume a large amount of memory. Therefore, it is important to clear the model when it is no longer needed.

Having the general rules in mind, we now show how the creation is performed for the different methods.

- To create a model for *gray-value-based matching*, the operator create_template_rot is used. The model can then be stored to file with write_template.

- To create a model for the *shape-based matching*, the operator create_scaled_shape_model (or create_aniso_shape_model for anisotropic scaling is used. If no scaling is needed, you can use create_shape_model. The model can then be stored to file with write_shape_model

- For the *component-based matching*, the model can be created in different ways: If the components and their relations are known, you can use the operator create_component_model. If the components are known, but not their relations, you can train the relations using the operator train_model_components with a set of training images, and then create the model using create_trained_component_model. Finally, if the components themselves are not known, you can

determine them with the operator gen_initial_components, which then serve as the input for train_model_components.

The model can then be stored to file with write_component_model. The training components can be saved with write_training_components.

- To create a model for the *correlation-based matching*, the operator create_ncc_model is used. The model can then be stored to file with write_ncc_model
- For the training of the *point-based matching*, no real model is generated. Here, only coordinates must be determined, which will later be matched: For both the template and the search image, subpixel locations of significant points are extracted. This is done using an operator like points_foerstner. The returned positions, together with the image data, will then be input for the matching.

9.2.5 Find Model

Finding an object with matching is an online process that has the image and one or multiple models as input. Depending on the selected method, the search operators offer different ways of controlling the search.

- You can search objects with *gray-value-based matching* using the operator create_template_rot. This operator finds single instances of rotated objects. If the objects are not rotated, the operator fast_match_mg allows to find all points with a high score. These locations are returned as a region and can be further processed, e.g., with blob analysis (see the description of this method on page 33).
- The standard operator for *shape-based matching* is find_scaled_shape_model. It allows locating multiple instances of the same template with rotations and scaling. If multiple models are needed, the operator find_scaled_shape_models is used. For both operators, there are variants for anisotropic scaling and without scaling.
- For the *component-based matching*, find_component_model is used to locate multiple instances of objects, which may be rotated and whose components may move with respect to each other.
- To find objects using *correlation-based matching*, the operator find_ncc_model is applied. It returns the position, rotation and score of the found instances of a model.
- The matching operator for the *point-based matching* is proj_match_points_ransac. The input are the two images containing the template *and* the object to be located, together with the significant points. The result of the matching process is a 2D mapping from the template image to the object image. This mapping is the input for operators like gen_projective_mosaic, which can be used to combine the two images.

9.2.6 Visualize Results

A typical visualization task is to display the model contours overlaid on the image at the found position. An easy way to do this is to access the model contours by calling get_shape_model_contours (for the shape-based matching). With vector_angle_to_rigid and disp_xld, the contours can be displayed.

Displaying the results of the component-based matching is more complex. The easiest way is to call `get_found_component_model`, which returns regions, one for each component. These can directly be displayed with `disp_region`.

For detailed information see the description of this method on page 269.

9.3 Programming Examples

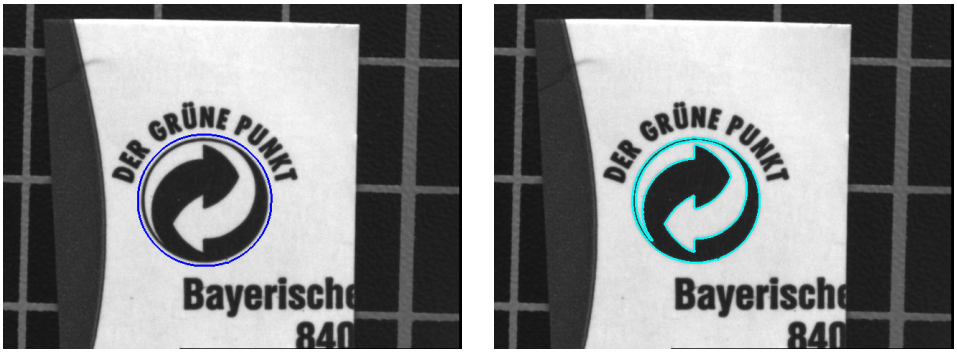
This section gives a brief introduction to using HALCON for template matching.

9.3.1 Creating a Model for the "Green Dot"

Example: `examples\solution_guide\basics\create_model_green_dot.dev`

This example shows how to use the shape-based matching with objects of varying size. Figure 9.4 depicts the training image, which contains the so-called "green dot", a symbol used in Germany for recycling packages. The template is not defined by a user interaction but by a segmentation step: Using `threshold` all dark pixels are selected and the connected component with the appropriate size is chosen (`select_shape`). This region is then filled and slightly dilated. Figure 9.4a depicts the result of the segmentation.

```
threshold (Image, Region, 0, 128)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 10000,
              20000)
fill_up (SelectedRegions, RegionFillUp)
dilation_circle (RegionFillUp, RegionDilation, 5.5)
```



a) b)

Figure 9.4: Creating a model for template matching: (a) ROI for the template region; (b) model contours.

The extracted region is then combined with the image (`reduce_domain`) to be used as the domain, i.e., as the region of interest. To check whether the value for the parameter `Contrast` has been chosen correctly,

inspect_shape_model is called. The training is finally applied using create_scaled_shape_model. The resulting model contours are depicted in figure 9.4b.

```
reduce_domain (Image, RegionDilation, ImageReduced)
inspect_shape_model (ImageReduced, ModelImages, ModelRegions, 1, 40)
create_scaled_shape_model (ImageReduced, 5, rad(-45), rad(90), 0, 0.8, 1.0,
                           0, ['none','no_pregeneration'],
                           'ignore_global_polarity', 40, 10, ModelID)
```

To complete the program, the model is written to file (write_shape_model); then, the memory of the model is released with clear_shape_model.

```
write_shape_model (ModelID, 'green-dot.shm')
clear_shape_model (ModelID)
```

9.3.2 Locating "Green Dots"

Example: examples\solution_guide\basics\matching_green_dot.dev

In this example, we use the model created in the previous example to locate the so-called "green dots" in a search image. As you can see in figure 9.5a, the search image contains three "green dots" in different orientations and scales. Furthermore, some of them are partly occluded.

a) b)

Figure 9.5: Using template matching to locate rotated and scaled objects: (a) search image; (b) matches.

First, the shape model is read from file.

```
read_shape_model ('green-dot.shm', ModelID)
```

Then, the search operator find_scaled_shape_model is executed. The result of the operator are the positions, the orientations, and the scales of the "green dots". To display the model contours overlaid on the image, the contours are accessed with get_shape_model_contours. The for-loop is used to

handle the contours for each found location. This is done by generating an appropriate transformation and then moving the contours to the correct position. Figure 9.5b depicts the result.

```
find_scaled_shape_model (ImageSearch, ModelID, rad(-45), rad(90), 0.8, 1.0,
                         0.5, 0, 0.5, 'least_squares', 5, 0.8, Row, Column,
                         Angle, Scale, Score)
get_shape_model_contours (ModelContours, ModelID, 1)
for I := 0 to |Score|-1 by 1
  vector_angle_to_rigid (0, 0, 0, Row[I], Column[I], Angle[I],
                         HomMat2DRotate)
  hom_mat2d_scale (HomMat2DRotate, Scale[I], Scale[I], Row[I], Column[I],
                   HomMat2DScale)
  affine_trans_contour_xld (ModelContours, ModelTrans, HomMat2DScale)
  dev_display (ModelTrans)
endfor
```

At the end of the program, the memory of the model is released with `clear_shape_model`.

```
clear_shape_model (ModelID)
```

9.3.3 Distinguishing coins

Example: `examples\solution_guide\basics\matching_coins.dev`

The task of this example is to distinguish different types of Euro coins, depending on the country of origin. The coins differ on one side, having a specific symbol for each country (see figure 9.6). Like the examples described before, the task is solved by shape-based matching. For each country symbol, one model is used.

The program consists of multiple procedures for the different tasks. The main program simply defines the working environment and calls these procedures. The first part is the creation of four models for the four different types of 20 cent coins. Inside a `for`-loop, the training procedure `train_model` is called with the corresponding training images, and the model IDs are collected in a tuple.

```
Names := ['german','italian','greek','spanish']
Models := []
for i := 0 to 3 by 1
  read_image (Image, 'coins/20cent_'+Names[i])
  dev_display (Image)
  train_model (Image, ModelID)
  Models := [Models,ModelID]
endfor
```

In the second part of the main program, the created models are used to recognize the origin of a coin. After applying the matching with the procedure `find_coin`, the result is visualized with the procedure `display_model`.

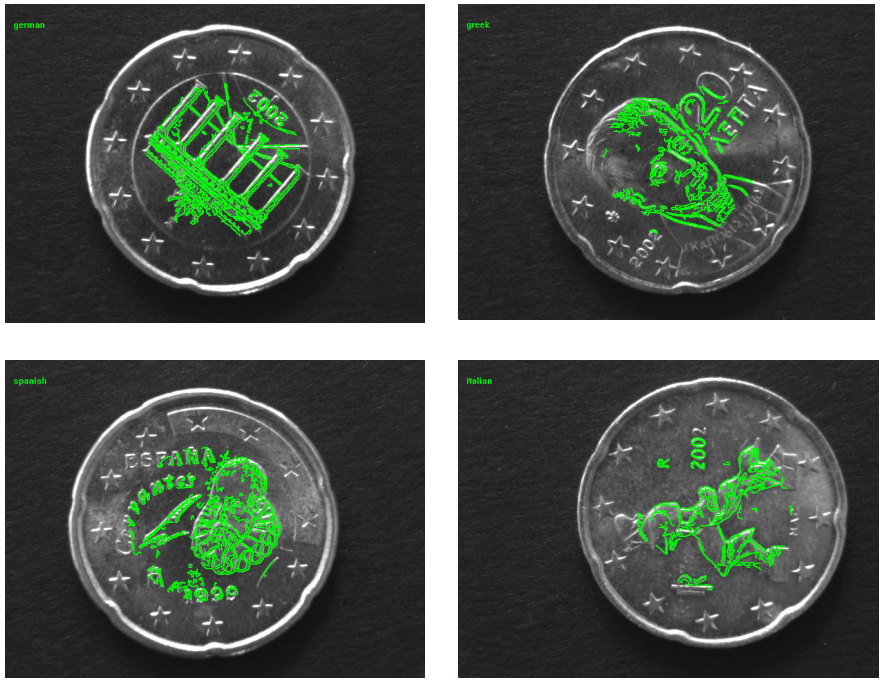


Figure 9.6: Multiple coins that are distinguished.

```
for i := 1 to 13 by 1
  read_image (Image, 'coins/20cent_'+i$'.2'+'.png')
  find_coin (Image, Models, Row, Column, Angle, Score, Model)
  display_model (Image, Model, Row, Column, Angle, Names, WindowHandle)
endfor
```

For the training, ROIs are generated automatically using the result of the procedure `locate_coin`. This procedure applies `threshold` to extract all bright pixels in the image. From the connected components, the largest one is selected with `select_shape_std`. The selected region contains several holes, which are filled with `shape_trans` using the parameter value `'convex'`, thereby transforming the region to its convex hull. Because only the inner part of the coins will be used as a template, a circle located in the center of gravity with a fixed radius is generated.

```
  threshold (Image, Region, 70, 255)
  connection (Region, ConnectedRegions)
  select_shape_std (ConnectedRegions, SelectedRegions, 'max_area', 0)
  shape_trans (SelectedRegions, RegionTrans, 'convex')
  area_center (RegionTrans, _, Row, Column)
  gen_circle (Coin, Row, Column, 120)
```

What remains to do in `train_model` is to determine the contrast parameter and then to create the model using `create_shape_model`. Note that the value `'ignore_local_polarity'` is used for the param-

eter Metric because the dark/light transitions can change locally due to the varying illumination.

```
Contrast := 20
HysteresisContrast := [Contrast/2,Contrast+6,10]
reduce_domain (Image, Coin, ImageReduced)
create_shape_model (ImageReduced, 'auto', 0, rad(360), 'auto', 'none',
                    'ignore_local_polarity', HysteresisContrast, 5,
                    ModelID)
```

The procedure find_coin also first calls locate_coin to derive a region of interest that is as small as possible. This will speedup the matching process significantly. A circle of radius 30 in the center of the coin is used as the search ROI. Inside this region, find_shape_models is called to determine which coin can be seen and to precisely determine its position and orientation.

```
locate_coin (Image, Coin)
area_center (Coin, _, Row, Column)
gen_circle (Circle, Row, Column, 35)
reduce_domain (Image, Circle, ImageReduced)
find_shape_models (ImageReduced, Models, 0, rad(360), 0.6, 1, 0,
                   'least_squares', 0, 1, Row, Column, Angle, Score,
                   Model)
```

The procedure display_model accesses the model edges and transforms them according to the found position and orientation and displays them overlaid on the image. As additional feedback, the type of the coin is displayed in the graphics window.

```
get_shape_model_contours (ModelContours, Model, 1)
vector_angle_to_rigid (0, 0, 0, Row, Column, Angle, HomMat2D)
affine_trans_contour_xld (ModelContours, ContoursAffinTrans, HomMat2D)
dev_display (Image)
dev_set_color ('green')
dev_set_line_width (2)
dev_display (ContoursAffinTrans)
set_tposition (WindowHandle, 24, 12)
write_string (WindowHandle, Names[Model])
```

Finally, the model is destroyed.

```
for i := 0 to 3 by 1
  clear_shape_model (Models[i])
endfor
```

9.3.4 Locate Components on a PCB

Example: examples\hdevelop\Matching\Component-Based\cbm_modules_simple.dev

The task of this example is to locate multiple components on a printed circuit board in one step (see figure 9.7). On a printed circuit board, typically multiple different objects are mounted, whose positions

can vary because of the tolerances in the mounting process. To locate all objects quickly and in a robust manner, the component-based matching is used.

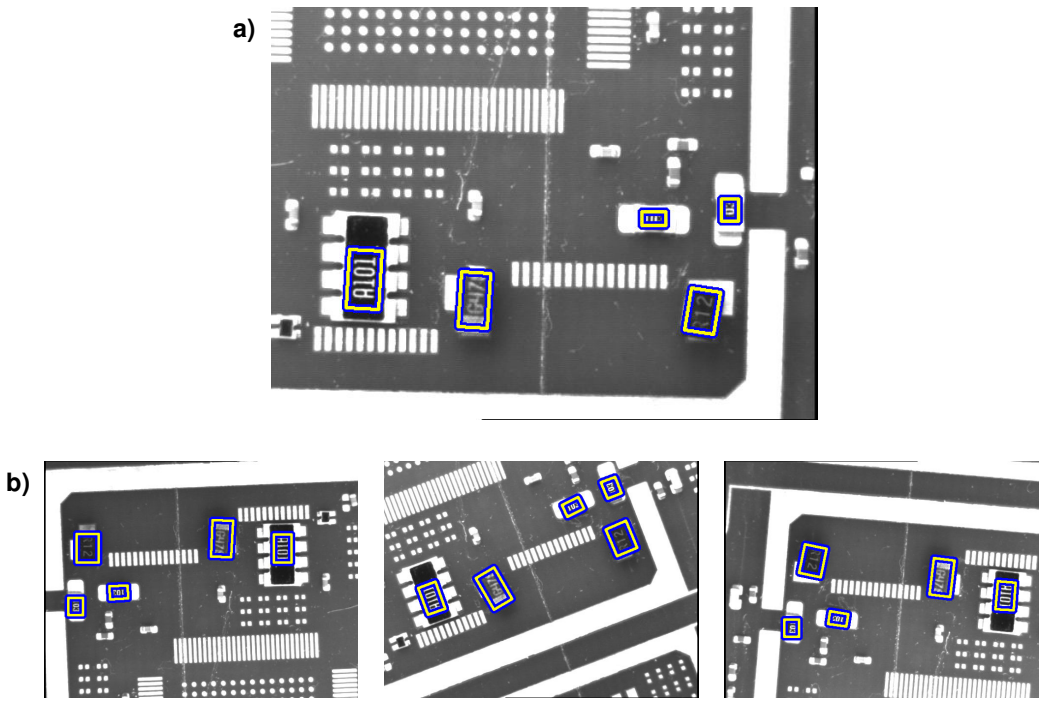


Figure 9.7: Component-based matching: (a) training objects; (b) objects, located in different images where the relation of the objects with respect to each other varies.

In the training step, each object is marked with a region of interest (figure 9.7a). The possible movement and rotation of the objects is known from the manufacturing process and is passed as a parameter to the training.

```
gen_rectangle2 (ComponentRegions, 318, 109, -1.62, 34, 19)
gen_rectangle2 (Rectangle2, 342, 238, -1.63, 32, 17)
gen_rectangle2 (Rectangle3, 355, 505, 1.41, 25, 17)
gen_rectangle2 (Rectangle4, 247, 448, 0, 14, 8)
gen_rectangle2 (Rectangle5, 237, 537, -1.57, 13, 10)
concat_obj (ComponentRegions, Rectangle2, ComponentRegions)
concat_obj (ComponentRegions, Rectangle3, ComponentRegions)
concat_obj (ComponentRegions, Rectangle4, ComponentRegions)
concat_obj (ComponentRegions, Rectangle5, ComponentRegions)
create_component_model (ModelImage, ComponentRegions, 20, 20, rad(25), 0,
                        rad(360), 15, 40, 15, 10, 0.8, [4,3,3,3,3], 0,
                        'none', 'use_polarity', 'true', ComponentModelID,
                        RootRanking)
```

Now, the component-based matching is able to find all objects in one step (figure 9.7a), returning the

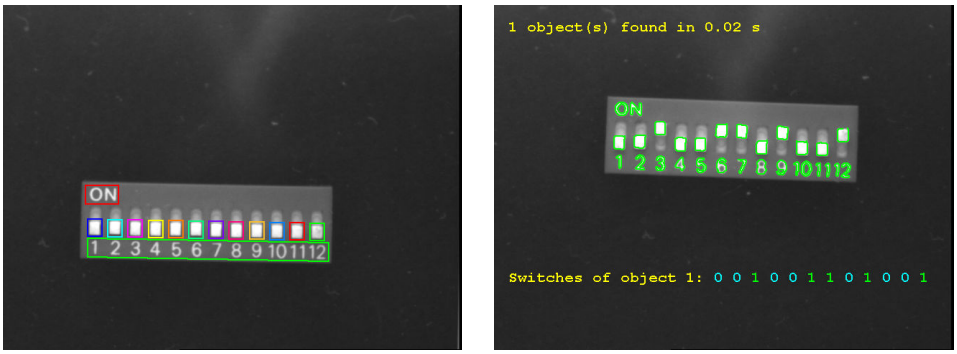
relative positions of each object.

```
find_component_model (SearchImage, ComponentModelID, RootRanking, 0,
                      rad(360), 0.5, 0, 0.5, 'stop_search',
                      'search_from_best', 'none', 0.8, 'interpolation',
                      0, 0.8, ModelStart, ModelEnd, Score, RowComp,
                      ColumnComp, AngleComp, ScoreComp, ModelComp)
```

9.3.5 Check the State of a Dip Switch

Example: examples\hdevelop\Applications\FA\cbm_dip_switch.dev

The task of this example is to check a dip switch, i.e., to determine the positions of the single switches relative to the casing (see figure 9.8).



a) b)

Figure 9.8: (a) Model image of the dip switch with ROIs for the components; (b) located dip switch with state of the switches.

The task is solved using component-based matching. The dip switch consists of 14 components: 12 for the switches and two for the printed text on the casing (see figure 9.8a). The training is performed using training images that show all possible positions of the switches. From this, the system learns the possible movements of the switches relative to the casing.

```
train_model_components (ModelImage, InitialComponents, TrainingImages,
                        ModelComponents, 45, 45, 30, 0.95, -1, -1,
                        rad(20), 'speed', 'rigidity', 0.2, 0.5,
                        ComponentTrainingID)
```

To make the recognition more robust, small tolerances are added to the trained movements.

```
modify_component_relations (ComponentTrainingID, 'all', 'all', 0, rad(4))
```

Now, the model can be created.

```
create_trained_component_model (ComponentTrainingID, 0, rad(360), 10,
                                MinScoreComp, NumLevelsComp, 'auto',
                                'none', 'use_polarity', 'false',
                                ComponentModelID, RootRanking)
```

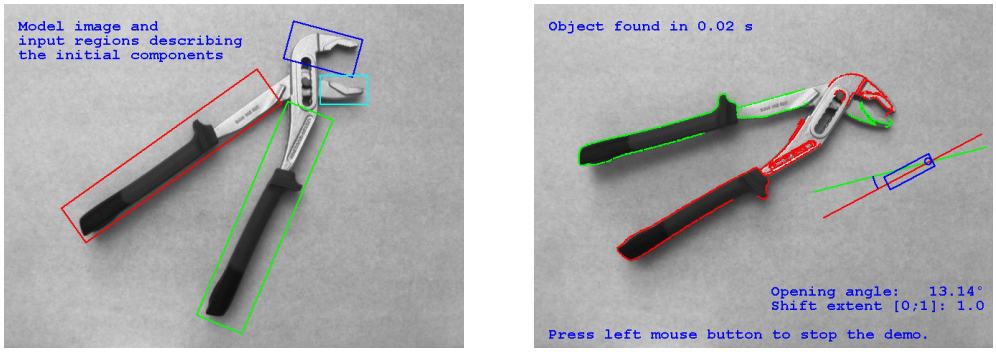
When searching for the dip switch, not only the casing but each single dip together with its relative position is returned. Based on this information, the global status of the switch can easily be derived.

```
find_component_model (SearchImage, ComponentModelID, RootRanking, 0,
                      rad(360), 0, 0, 0.5, 'stop_search',
                      'prune_branch', 'none', MinScoreComp,
                      'least_squares', 0, 0.9, ModelStart, ModelEnd,
                      Score, RowComp, ColumnComp, AngleComp, ScoreComp,
                      ModelComp)
```

9.3.6 Locating a Pipe Wrench in Different States

Example: examples\hdevelop\Applications\FA\cbm_pipe_wrench.dev

The task of this example is to locate a pipe wrench based on four predefined ROIs, two for each rigid part (see figure 9.9). Again, the task is solved using component-based matching.



a) b)

Figure 9.9: (a) Model image with specified ROIs for the components; (b) located pipe wrench in another state.

Showing the system multiple example images trains the relative movements of the parts.

```
read_image (ModelImage, 'pipe_wrench/pipe_wrench_model')
gen_rectangle2 (InitialComponentRegions, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle2, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
concat_obj (InitialComponentRegions, Rectangle2, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle3, InitialComponentRegions)
concat_obj (InitialComponentRegions, Rectangle4, InitialComponentRegions)
gen_empty_obj (TrainingImages)
for i := 1 to 4 by 1
  read_image (TrainingImage, 'pipe_wrench/pipe_wrench_training_'+i)
  concat_obj (TrainingImages, TrainingImage, TrainingImages)
  set_tposition (WindowHandle, 20, 20)
  write_string (WindowHandle, 'Training image '+i+' of 4')
  print_mouse_message (WindowHandle, 'Press mouse button to continue.')
endfor
train_model_components (ModelImage, InitialComponentRegions,
                        TrainingImages, ModelComponents, 22, 60, 30, 0.65,
                        -1, -1, rad(60), 'speed', 'rigidity', 0.2, 0.4,
                        ComponentTrainingID)
```

The components and their relations are displayed as follows:

```
get_training_components (ModelComponents, ComponentTrainingID,
                         'model_components', 'model_image', 'false',
                         RowRef, ColumnRef, AngleRef, ScoreRef)
dev_display (ModelComponents)

get_component_relations (Relations, ComponentTrainingID, i,
                         'model_image', Row, Column, Phi, Length1,
                         Length2, AngleStart, AngleExtent)
dev_display (Relations)
```

Based on the training, the actual component model is created. Component relations are represented in a tree structure.

```
create_trained_component_model (ComponentTrainingID, rad(-90), rad(180),
                                10, 0.6, 4, 'auto', 'none', 'use_polarity',
                                'false', ComponentModelID, RootRanking)
get_component_model_tree (Tree, Relations, ComponentModelID, RootRanking,
                          'model_image', StartNode, EndNode, Row, Column,
                          Phi, Length1, Length2, AngleStart, AngleExtent)
dev_display (ModelImage)
dev_display (Tree)
dev_display (Relations)
```

Finally, test images are used to locate the pipe wrench. For each image, the model contours are overlaid and the shift and opening angle of the pipe wrench is visualized.

```
find_component_model (SearchImage, ComponentModelID, RootRanking,
                      rad(-90), rad(180), 0, 0, 1, 'stop_search',
                      'prune_branch', 'none', 0.6, 'least_squares', 0,
                      0.7, ModelStart, ModelEnd, Score, RowComp,
                      ColumnComp, AngleComp, ScoreComp, ModelComp)
dev_display (SearchImage)
NumFound := |ModelStart|
if (NumFound)
  get_found_component_model (FoundComponents, ComponentModelID,
                             ModelStart, ModelEnd, RowComp,
                             ColumnComp, AngleComp, ScoreComp,
                             ModelComp, 0, 'false', RowCompInst,
                             ColumnCompInst, AngleCompInst,
                             ScoreCompInst)
  dev_display (FoundComponents)
  visualize_pipe_wrench_match (AngleCompInst, WindowHandle,
                               RowCompInst, ColumnCompInst, RowRef,
                               ColumnRef)
endif
```

9.3.7 Creating a Mosaic Image

Example: examples\hdevelop\Tools\2D-Transformations\gen_projective_mosaic.dev

The task of this example is to create an image of the elongated printed circuit board depicted in figure 9.10. Using standard image sizes, most of the image would be empty. The solution is to acquire images from multiple viewpoints and then to create a mosaic image using a point-based matching.

Multiple overlapping images of a printed circuit board are the input for the program. In a first step, significant points are extracted in each of these images. These points are the input for the point-based matching. In each step, two successive images are matched. The result of this process is a mapping from one image to the next.

```
points_foerstner (ImageF, 1, 2, 3, 200, 0.3, 'gauss', 'false',
                  RowJunctionsF, ColJunctionsF, CoRRJunctionsF,
                  CoRCJunctionsF, CoCCJunctionsF, RowAreaF, ColAreaF,
                  CoRRAreaF, CoRCAreaF, CoCCAreaF)
points_foerstner (ImageT, 1, 2, 3, 200, 0.3, 'gauss', 'false',
                  RowJunctionsT, ColJunctionsT, CoRRJunctionsT,
                  CoRCJunctionsT, CoCCJunctionsT, RowAreaT, ColAreaT,
                  CoRRAreaT, CoRCAreaT, CoCCAreaT)
proj_match_points_ransac (ImageF, ImageT, RowJunctionsF, ColJunctionsF,
                          RowJunctionsT, ColJunctionsT, 'ncc', 21, 0,
                          0, 480, 640, 0, 0.5, 'gold_standard', 1,
                          4364537, ProjMatrix, Points1, Points2)
ProjMatrices := [ProjMatrices,ProjMatrix]
```

These mappings are collected and finally used to construct a single high-resolution image of the complete PCB.

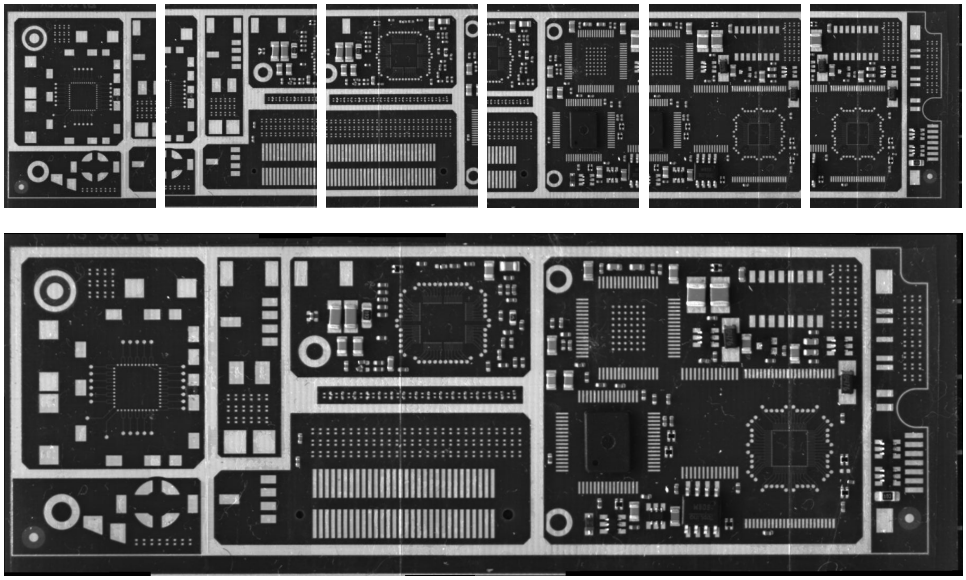


Figure 9.10: Creating a mosaic image from multiple overlapping images.

```
gen_projective_mosaic (Images, MosaicImage, 2, From, To, ProjMatrices,
                       'default', 'false', MosaicMatrices2D)
```


9.3.8 Other Examples

9.3.8.1 Template Matching (Component-Based)

HDevelop

- examples\hdevelop\Applications\FA\cbm_bin_switch.dev
  Locate a switch and test its state using component-based matching
- examples\hdevelop\Applications\FA\cbm_caliper.dev
  Measure the setting of a caliper using component-based matching in a perspectively distorted image
- examples\hdevelop\Applications\FA\cbm_dip_switch.dev
  Locate dip switches and test their state using component-based matching
  → description here in the Solution Guide Basics on page 129
- examples\hdevelop\Applications\FA\cbm_pipe_wrench.dev
  Locate a pipe wrench that consists of two components
  → description here in the Solution Guide Basics on page 130
- examples\hdevelop\Matching\Component-Based\cbm_label_simple.dev
  Locate a label using component-based matching

- examples\hdevelop\Matching\Component-Based\cbm_modules_simple.dev
  Locate modules on a board using component-based matching
  → description here in the Solution Guide Basics on page 127
- examples\hdevelop\Matching\Component-Based\cbm_param_visual.dev
  Analyze the parameters of component-based matching
- examples\hdevelop\Matching\Component-Based\cbm_pipe_wrench_simple.dev
  Locate compound objects using component-based matching
- examples\hdevelop\Matching\Component-Based\cbm_sbm.dev
  Compare component-based matching to shape-based matching
- examples\solution_guide\3d_machine_vision\grid_rectification_arbitrary_distortion.dev
  Determine differences between two printed pages, even if there are distortions in the vertical direction
  → description in the Solution Guide II-F on page 133

9.3.8.2 Template Matching (Correlation-Based)

HDevelop

- examples\hdevelop\Matching\Correlation-Based\find_ncc_model_defocused.dev
  Find an object despite changes in texture and focus using correlation-based matching
- examples\hdevelop\Matching\Correlation-Based\find_ncc_model_exposure.dev
  Find an object despite linear illumination changes using correlation-based matching

9.3.8.3 Template Matching (Shape-Based)

HDevelop

- examples\hdevelop\Applications\FA\matching_image_border.dev
  Locate multiple dies even if they extend beyond the image border
- examples\hdevelop\Applications\FA\matching_multi_channel_clamp.dev
  Locate upper clamp from a pile using color shape-based matching
- examples\hdevelop\Applications\FA\matching_multi_channel_yogurt.dev
  Find yogurts of different flavors using color shape-based matching
- examples\hdevelop\Applications\FA\pm_measure_board.dev
  Locate IC on a board and measure pin distances
  → description here in the Solution Guide Basics on page 64
- examples\hdevelop\Applications\FA\pm_world_plane.dev
  Recognize planar objects using shape-based matching in perspectively distorted images
- examples\hdevelop\Applications\FA\print_check.dev
  Perform a typical print quality inspection using a variation model
  → description here in the Solution Guide Basics on page 161

- examples\hdevelop\Applications\FA\print_check_single_chars.dev
  Perform a typical print quality inspection using variation models for each character
- examples\hdevelop\Matching\Shape-Based\find_aniso_shape_model.dev
  Find SMD capacitors using shape-based matching (with anisotropic scaling and rotation)
- examples\solution_guide\basics\variation_model_illumination.dev
  Inspect print using variation model with varying illumination
  → description here in the Solution Guide Basics on page 167
- examples\solution_guide\basics\variation_model_single.dev
  Inspect print using variation model with single reference image
  → description here in the Solution Guide Basics on page 165
- examples\solution_guide\shape_matching\align_measurements.dev
  Inspect individual razor blades using shape-based matching to align ROIs for the measure tool
  → description in the Solution Guide II-B on page 37
- examples\solution_guide\shape_matching\first_example_shape_matching.dev
  Introduce HALCON's shape-based matching
  → description in the Solution Guide II-B on page 5
- examples\solution_guide\shape_matching\multiple_models.dev
  Search for two types of objects simultaneously
  → description in the Solution Guide II-B on page 26
- examples\solution_guide\shape_matching\multiple_objects.dev
  Search for multiple instances of a security ring
  → description in the Solution Guide II-B on page 35
- examples\solution_guide\shape_matching\multiple_scales.dev
  Search for nuts of different sizes
  → description in the Solution Guide II-B on page 44
- examples\solution_guide\shape_matching\process_shape_model.dev
  Create a model ROI by modifying the result of inspect_shape_model
  → description in the Solution Guide II-B on page 11
- examples\solution_guide\shape_matching\reuse_model.dev
  Store and reuse a shape model
  → description in the Solution Guide II-B on page 46
- examples\solution_guide\shape_matching\synthetic_circle.dev
  Use a synthetic model (circle) to search for capacitors on a board
  → description in the Solution Guide II-B on page 20

C++

- examples\cpp\source\pen.cpp
  Inspect the quality of print on a pen using the variation model of HALCON
- examples\mfc\Matching\Matching.cpp
  Locate an IC using HALCON/C++ and MFC, creating a HALCON window

- examples\mfc\MatchingCOM\Matching.cpp
  Locate an IC using HALCON/COM and MFC
- examples\mfc\MatchingExtWin\Matching.cpp
  Locate an IC using HALCON/C++ and MFC, paint into an existing window
- examples\motif\Matching\source\matching.cpp
  Locate an IC using HALCON/C++ and Motif
- examples\qt\Matching\matching.cpp
  Locate an IC using HALCON/C++ and Qt

C#

- examples\c#\Matching\vs.net\Matching.csproj
  Locate an IC on a board and measure pin distances
- examples\hdevengine\c#\MultiThreading\vs.net\MultiThreading.csproj
  Execute an HDevelop procedure in parallel by two threads using Parallel HDevEngine
  → description in the Programmer's Guide on page 194
- examples\hdevengine\c#\MultiThreadingTwoWindows\vs.net\
  MultiThreadingTwoWindows.csproj
  Execute different HDevelop procedures in parallel by two threads using Parallel HDevEngine
  → description in the Programmer's Guide on page 202

Visual Basic .NET

- examples\vb.net\Matching\vs.net\Matching.vbproj
  Locate an IC on a board and measures pin distances

Visual Basic

- examples\vb\Tools\Matching\matching.vbp
  Locate an IC on a board and measure pin distances

Delphi

- examples\delphi\Matching\matching.dpr
  Locate an IC on a board and measure pin distances

9.3.8.4 Template Matching (Gray-Value-Based)

HDevelop

- examples\hdevelop\Applications\FA\adapt_pattern.dev
  Apply online pattern matching using an adapting gray value template
- examples\hdevelop\Applications\FA\pattern.dev
  Apply online pattern matching using a gray value template

- examples\hdevelop\Applications\FA\pm_illu.dev
  Apply online pattern matching using a gray value template that adapts to illumination changes
- examples\hdevelop\Applications\FA\pm_illu_rectify.dev
  Find a rotated pattern using a gray value template and align the image to the pattern
- examples\hdevelop\Applications\FA\pm_illu_rot.dev
  Find a rotated pattern using a gray value template
- examples\hdevelop\Filter\Match\exhaustive_match.dev
  Match template and image (gray-value-based matching)
- examples\hdevelop\Matching\Gray-Value-Based\best_match.dev
  Find the best match of a gray value template
- examples\hdevelop\Matching\Gray-Value-Based\best_match_mg.dev
  Find the best match of a gray value template in a pyramid
- examples\hdevelop\Matching\Gray-Value-Based\best_match_rot_mg.dev
  Find the best match (with rotation) of a gray value template in a pyramid
- examples\hdevelop\Matching\Gray-Value-Based\create_template_rot.dev
  Find the best match (with rotation) of a gray value template
- examples\hdevelop\Matching\Gray-Value-Based\fast_match_mg.dev
  Find matches of a gray value template in a pyramid
- examples\hdevelop\Matching\Gray-Value-Based\set_offset_template.dev
  Adapt to illumination changes by adding a gray value offset to a template
- examples\hdevelop\Matching\Gray-Value-Based\set_reference_template.dev
  Define the reference position for a gray value template
- examples\hdevelop\Matching\Gray-Value-Based\write_template.dev
  Read and write gray value template files
- examples\hdevelop\Tools\2D-Transformations\vector_angle_to_rigid.dev
  Match a pattern and display the normalized image

Visual Basic

- examples\vb\Applications\FA\pm_illu_rot.vbp
  Find a rotated pattern using a gray value template

9.3.8.5 Template Matching (Point-Based)

HDevelop

- examples\hdevelop\Tools\2D-Transformations\gen_projective_mosaic.dev
  Combine several images of a PCB into a large mosaic image
  → description here in the Solution Guide Basics on page 132

- examples\solution_guide\3d_machine_vision\bundle_adjusted_mosaicking.dev
  Use bundle-adjusted mosaicking to merge partial images of a BGA into one large image
  → description in the Solution Guide II-F on page 84
- examples\solution_guide\3d_machine_vision\mosaicking.dev
  Use mosaicking to merge partial images of a BGA into one large image
  → description in the Solution Guide II-F on page 72

9.4 Selecting Operators

9.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

9.4.2 Radiometrically Calibrate Image(s)

Standard:

radiometric_self_calibration, lut_trans

9.4.3 Rectify Image(s)

Operators for rectifying images are described in the Solution Guide II-F.

9.4.4 Determine Training Parameters

Standard:

determine_shape_model_params, get_shape_model_params, inspect_shape_model

9.4.5 Train Model

Standard:

create_template_rot, create_shape_model, create_scaled_shape_model,
create_aniso_shape_model, create_component_model, create_ncc_model,
gen_initial_components, train_model_components, create_trained_component_model,
points_foerstner, points_harris

9.4.6 Find Model

Standard:

best_match_mg, find_shape_model, find_scaled_shape_model, find_aniso_shape_model, find_component_model, find_ncc_model, proj_match_points_ransac, gen_projective_mosaic

Advanced:

best_match_pre_mg, fast_match_mg, find_shape_models, find_scaled_shape_models, find_aniso_shape_models, projective_trans_image_size

9.4.7 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

9.4.8 Destroy Model

Standard:

clear_template, clear_shape_model, clear_component_model, clear_training_components, clear_ncc_model

9.5 Relation to Other Methods

9.5.1 Methods that are Using Template Matching

1D Measuring (see description on page 57)
OCR (see description on page 233)
Variation Model (see description on page 159)
1D Bar Code (see description on page 209)

The pose returned by the matching operators can be used as the input for a so-called alignment. This means that either the position of an ROI is transformed relative to the movement of a specified object in the image, or the image itself is transformed so that the pixels are moved to the desired position.

A detailed description of alignment can be found in the Solution Guide II-B in section 4.3.4 on page 37. For an example of using alignment as a preprocessing for 1D Measuring on page 57 see the description of pm_measure_board.dev on page 64. Further examples can be found in the description of the method Variation Model on page 159.

9.5.2 Alternatives to Template Matching

Blob Analysis (see description on page 33)
In some applications, the object to find can be extracted with classical segmentation methods. With the operators `area_center` and `orientation_region` you can then determine its pose and use this information, e.g., to align an ROI. With this approach, the execution time can be reduced significantly.
If the objects to be found appear only translated but not rotated or scaled, the morphological operators `erosion1` and `opening` can be used as binary template matching methods. Unlike in other vision systems, in HALCON these operators are extremely fast.

9.6 Tips & Tricks

9.6.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for template matching, both for the creation and the search: Especially during the creation it is very effective to make use of arbitrarily shaped ROIs and thus mark out parts that do not belong to the template. During the search, ROIs can be used to focus the process only on relevant parts, which reduces the execution time. Two important notes on ROIs, which often cause confusion:

- The reference point of the template is defined by the center of gravity of the *ROI* used during creation, not by the center of gravity of the contours returned by `inspect_shape_model`.
- During the search process, only the reference point of the model must fit into the search ROI, not the complete shape. Therefore, sometimes very small ROIs consisting only of a few pixels can be used.

For an overview on how to construct regions of interest and how to combine them with the image see Region Of Interest on page 19.

9.6.2 Speed Up

Many online applications require maximum speed. Because of its flexibility, HALCON offers many ways to achieve this goal. Below, the most common ones are listed:

- Regions of interest are the standard way to increase the speed by processing only those areas where objects need to be inspected. This can be achieved using pre-defined regions, but also by an online generation of the regions of interest that depends on other objects in the image.
- If multiple objects are searched for, it is more efficient to use `find_scaled_shape_models` (or `find_aniso_shape_models`) instead of using `find_scaled_shape_model` (or `find_aniso_shape_model`) multiple times.
- Increasing the values for the parameters `MinContrast` and `Greediness` will decrease the execution time. However, you must make sure that the relevant objects will still be found.

- Using a lower value for the parameter `Contrast` during the training typically results in a better performance because more pyramid levels can be used. If the contrast is too low, irrelevant contours will also be included into the model, which typically causes a lower recognition rate and a decreased accuracy.
- Very small templates cannot be found as quickly as larger ones. The reason for this is the reduced number of significant contours within an image, which makes it harder to distinguish the template from other structures. Furthermore, with smaller templates fewer pyramid levels can be used.

9.7 Advanced Topics

9.7.1 High Accuracy

Sometimes very high accuracy is required. Here the matching methods offer various interpolation methods to achieve this goal. However, with a poor image quality even a better interpolation method will not improve the accuracy any more - it will just become slower. Here, it is important to consider both the image quality and a reasonable interpolation method.

Chapter 10

3D Matching

The aim of 3D matching is to find a specific 3D object in an image and determine its pose, i.e., the position and orientation of the object in space. For the shape-based 3D matching, a 3D shape model is generated from a 3D computer aided design (CAD) model. The 3D shape model consists of 2D projections of the 3D object seen from different views. To restrain the needed memory and run time for the shape-based 3D matching, you should restrict the allowed pose range of the shape model and thus minimize the number of 2D projections that have to be computed and stored in the 3D shape model. Analogously to the shape-based matching of 2D structures described in the chapter Template Matching on page 115, the 3D shape model is used to recognize instances of the object in the image. But here, instead of a 2D position, orientation, and scaling, the 3D pose of each instance is returned. Additionally, the 3D shape model is always derived directly from a 3D model and not from images. Further, for the 3D shape matching a camera calibration is necessary.

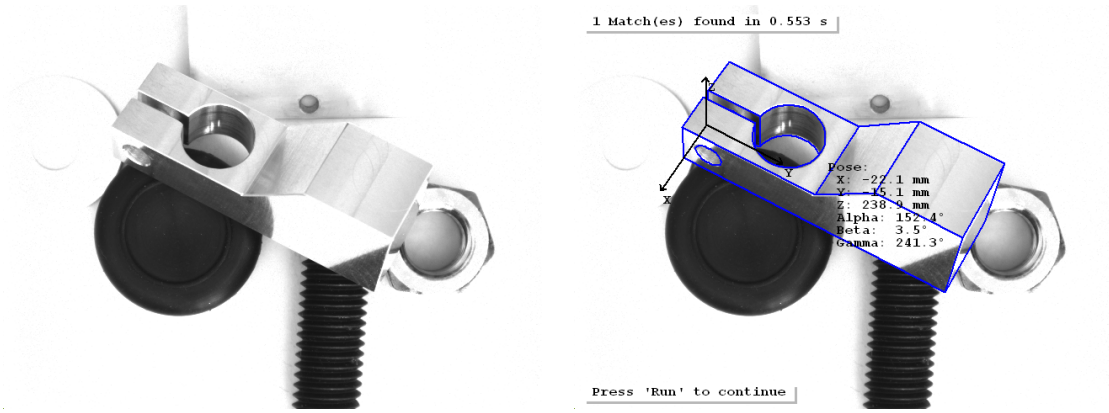
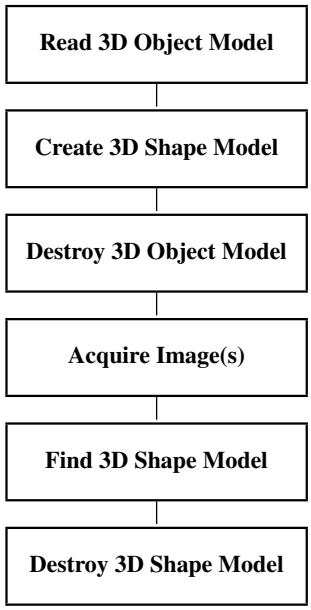


Figure 10.1: 3D matching: (left) original image containing a clamp, (right) 3D model of the found clamp projected in the image and display of its pose.

10.1 Basic Concept

Shape-based 3D matching consists of the following basic steps:



10.1.1 Read 3D Object Model

The 3D object model describing the object of interest must be available in DXF format (DXF version AC1009, AutoCad release 12) and may consist of the DXF entities POLYLINE (Polyface meshes), 3DFACE, LINE, CIRCLE, ARC, ELLIPSE, SOLID, BLOCK, and INSERT. The model is accessed in HALCON with the operator `read_object_model_3d_dxf`. The description of the operator in the Reference Manual provides additional tips how to obtain a suitable model.

10.1.2 Create 3D Shape Model

Having access to the 3D object model, the operator `create_shape_model_3d` creates a 3D shape model and thus prepares the 3D object model for the 3D matching. The shape model is generated by computing different views of the 3D object model within a user-specified pose range. The views are obtained by placing virtual cameras around the object model and projecting the 3D object model into the image plane of each camera position. The resulting 2D shape representations of all views are stored in the 3D shape model.

An important task is to specify the pose range. To ease this task, imagine a sphere that surrounds the object. On the surface of the sphere, a camera is placed that looks at the object. Now, the pose range can be defined by restricting the position of the camera to a part of the sphere's surface. Additionally, the

minimum and maximum distance of the camera to the object, i.e., the radii of different spheres, must be specified.

In the following, the position of the sphere relative to the object and the definition of the surface part are described. The position of the sphere is defined by placing its center at the center of the object's bounding box. The radius of the sphere corresponds to the distance of the camera to the center of the object. To define a specific part of the sphere's surface, the geographical coordinates longitude (λ) and latitude (φ) are used (see figure 10.2a). For these, minimum and maximum values are specified so that a quadrilateral on the sphere is obtained (see figure 10.2b).

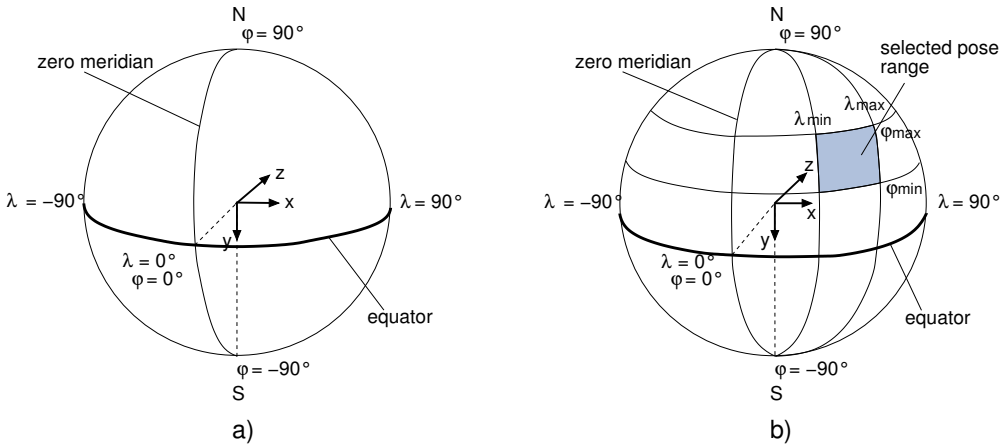


Figure 10.2: Geographic coordinate system: (a) the geographical coordinates longitude (λ) and latitude (φ) describe positions on the sphere's surface, (b) the minimum and maximum values for λ and φ describe a quadrilateral on the sphere, which defines the pose range.

To describe the orientation of the geographical coordinate system we introduce an object-centered coordinate system. This is obtained by moving the origin of the object coordinate system (DXF model) to the center of the sphere. The xz-plane of the object-centered coordinate system defines the equatorial plane of the geographical coordinate system. The north pole lies on the negative y-axis. The origin of the geographical coordinate system ($\lambda = \varphi = 0°$), i.e., the intersection of the equator with the zero meridian, lies in the negative z-axis.

To illustrate the above description, let us assume that we have the object model shown in figure 10.3a. For illustrative purpose additionally the object coordinate system is visualized. In figure 10.3b, the corresponding geographical coordinate system is shown together with a camera placed at its origin ($\lambda = \varphi = 0°$). Consequently, this camera view corresponds to a bottom view of the object.

Note that the coordinate system introduced here is only used to specify the pose range. The pose resulting from the 3D matching always refers to the original object coordinate system used in the DXF file and not to the center of the object's bounding box.

In most cases, the specification of the pose range can be simplified by changing the origin of the geographical coordinate system (i.e., the orientation of the sphere) such that it coincides with a mean viewing direction of the real camera to the object. This can be achieved by rotating the object-centered coordinate system, which defines the geographical coordinate system as described above. The rotation can be specified by passing the rotation angles to the parameters `RefRotX`, `RefRotY`, and `RefRotZ` of the operator

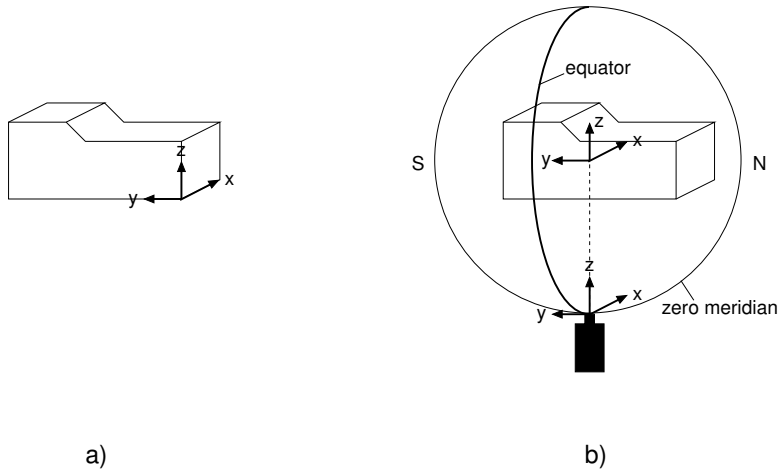


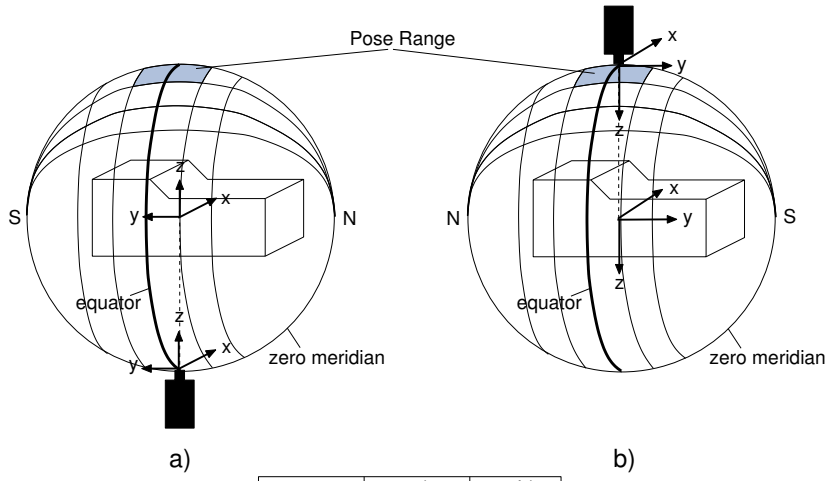
Figure 10.3: Object coordinate systems: (a) original object coordinate system of the DXF model, (b) object-centered geographical coordinate system obtained by moving the object to the center of the sphere: the camera is placed on the sphere's surface at the position $\lambda = \varphi = 0°$ and the object is placed at the center of the sphere. Note that in contrast to the previous image, the sphere is tilted, i.e., it is visualized in a way that the equator is vertical and the zero meridian is completely visible.

`create_shape_model_3d`. That is, you can specify the pose range either by adjusting the longitude and latitude, leaving the origin of the sphere at its initial position, or by rotating the object-centered coordinate system to get a mean reference view and then specify the pose range in a more intuitive way (see figure 10.4 on page 147), which is recommended in most cases.

Figure 10.5 shows a rotation-symmetric object, for which a reference view is specified. Rotation-symmetric objects have the specific advantage that they have the same appearance from all directions that are perpendicular to their rotation axis. Thus, if the mean reference view is selected such that the camera view is perpendicular to the rotation axis (and the rotation axis crosses the poles of the sphere), the longitude range can collapse to a single value so that the number of calculated 2D projections is significantly reduced. That is, less memory is needed and the matching becomes faster. Here, the initial z-axis of the object-centered coordinate system corresponds to the rotation axis of the object. To obtain the intended reference view, the orientation of the sphere is changed by rotating the object-centered coordinate system by -90° around its x-axis. Now, the pose range for φ is specified like described before and the pose range for λ is restricted to 0°.

Besides the definition of the pose range, also the camera roll angle, i.e., the allowed range for the rotation of the virtual camera around its z-axis, must be set. In most cases, it is recommended to allow a full circle for the camera roll angle. For details, we recommend to read the description of the operator `create_shape_model_3d` in the Reference Manual.

To create the model views correctly, the camera parameters are needed. The camera parameters can be obtained by a camera calibration. How to apply a camera calibration is described in detail in the Solution Guide II-F, section 3.1 on page 29.



	a)	b)
λ min/max	180° +/−α	0° +/−α
φ min/max	0° +/−α	0° +/−α
RefRotX	0°	180°
RefRotY	0°	0°
RefRotZ	0°	0°

Figure 10.4: Specify pose range: (a) only by longitude and latitude, or (b) by additionally rotating the object-centered coordinate system to a reference view. Note that because of the rotation around the x-axis, the positions of the poles have changed..

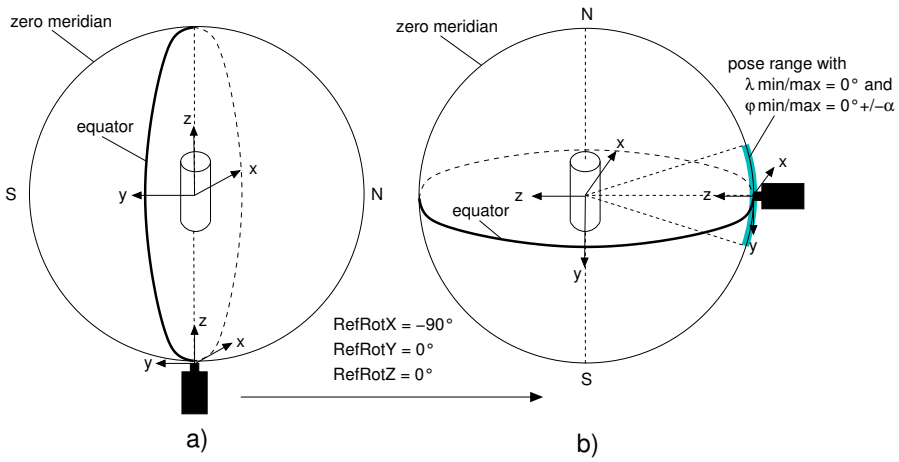


Figure 10.5: Change the origin of the geographical coordinate system for (a) a rotation-symmetric object with the z-axis of the object-centered coordinate system corresponding to the rotation axis such that (b) the z-axis becomes perpendicular to the rotation axis.

10.1.3 Destroy 3D Object Model

After creating the 3D shape model, the object model is not needed anymore and can be destroyed for memory reasons using the operator `clear_object_model_3d`.

10.1.4 Acquire Image(s)

Then, images are acquired.

For detailed information see the description of this method on page 13.

10.1.5 Find 3D Shape Model

With the 3D shape model that was created by `create_shape_model_3d` or read from file by `read_shape_model_3d` (see Re-use 3D Shape Model on page 150), the object can be searched for in images. For the search, the operator `find_shape_model_3d` is applied. Several parameters can be set to control the search process. For detailed information, we recommend to read the description of the operator in the Reference Manual. The operator returns the pose of the matching model, the standard deviation of the pose, and the score of the found instances of the 3D shape model that describes how much of the model is visible in the image.

10.1.6 Destroy 3D Shape Model

When the 3D shape model is not needed anymore, it is destroyed with the operator `clear_shape_model_3d`.

10.1.7 A First Example

An example for this basic concept is the following program.

The DXF file containing the 3D model of the clamp shown in Figure 10.1 on page 143 is read with `read_object_model_3d_dxf`.

```
read_object_model_3d_dxf ('clamp_sloped', 'mm', [], [], ObjectModel3DID,
                          DxfStatus)
```

For the creation of a 3D shape model, the camera parameters are needed. They can be determined by a camera calibration as described in the Solution Guide II-F, section 3.1 on page 29. Here, they are known and just assigned to the variable `CamParam`.

```
CamParam := [0.01221,-2791,7.3958e-6,7.4e-6,308.21,245.92,640,480]
```

Further parameters needed for the creation of the 3D shape model comprise the reference orientation and the pose range in which the object is expected, i.e., the minimum and maximum longitude and latitude, as

well as the minimum and maximum distance between the camera and the center of the object's bounding box (which correspond to the radii of the spheres that are built to compute the virtual camera positions). The range for the camera roll angle is set to a full circle. With these parameters, the 3D shape model is created using the operator create_shape_model_3d.

```
create_shape_model_3d (ObjectModel3DID, CamParam, RefRotX, RefRotY,
                       RefRotZ, 'gba', LongitudeMin, LongitudeMax,
                       LatitudeMin, LatitudeMax, 0, rad(360), DistMin,
                       DistMax, 10, 'min_face_angle', MinFaceAngle,
                       ShapeModel3DID)
```

After creating the 3D shape model, the object model is not needed anymore and is destroyed by the operator clear_object_model_3d.

```
clear_object_model_3d (ObjectModel3DID)
```

The image is acquired and the actual matching is applied in the image using the operator find_shape_model_3d.

```
read_image (Image, 'clamp_sloped/clamp_sloped_'+ImageNo$'02')
find_shape_model_3d (Image, ShapeModel3DID, 0.7, 0.9, 0, 'num_matches',
                     2, Pose, CovPose, Score)
```

At the end of the program, the 3D shape model is destroyed by the operator clear_shape_model_3d.

```
clear_shape_model_3d (ShapeModel3DID)
```

10.2 Extended Concept

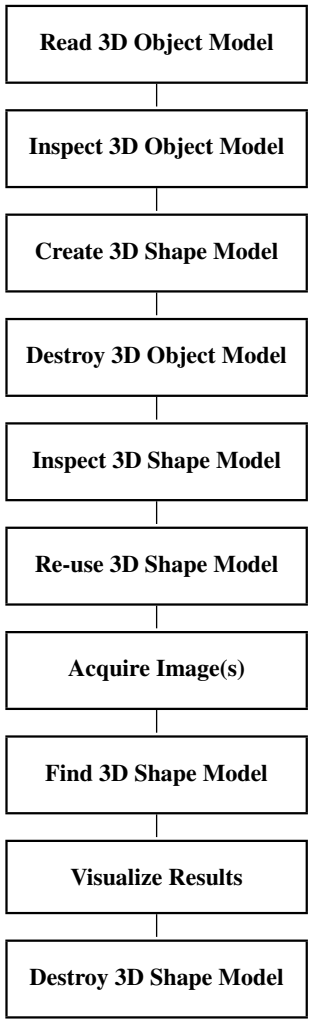
Often, more than the essential steps are necessary. You can, e.g., store the shape model into a file and read it from another application to separate the creation of the shape model from the actual search process. These and further advanced steps are described in the following sections.

10.2.1 Inspect 3D Object Model

The 3D object model that was accessed by read_object_model_3d_dxf can be inspected using get_object_model_3d_params. Depending on the parameter specified in GenParamNames, you can query either the bounding box of the object, i.e., the smallest enclosing cuboid that is parallel to the coordinate axes, or the coordinates of the center point of the bounding box.

10.2.2 Inspect 3D Shape Model

The 3D shape model that was created by create_shape_model_3d can be inspected using get_shape_model_3d_params. Several parameters like the reference orientation of the 3D object



model, the camera parameters, or the pose range used for the creation of the 3D shape model can be queried.

The contours of a specific view of the 3D shape model can be accessed with `get_shape_model_3d_contours`. The contours can be used to visualize and rate the 3D shape model and thus decide if certain parameters have to be adjusted within the search process using `find_shape_model_3d` or if the creation of the 3D shape model has to be repeated with adjusted parameters.

10.2.3 Re-use 3D Shape Model

Because of the complex calculations, the creation of a 3D shape model can be rather time consuming. Thus, if you need a 3D shape model more than once, it is recommended to store it into a file rather than to

repeat the model creation. To store the 3D shape model into file, the operator write_shape_model_3d is used. With the operator read_shape_model_3d, you read the 3D shape model from file again to re-use the model in another application.

10.2.4 Visualize Results

A typical visualization task is to use the operator project_object_model_3d or project_shape_model_3d to project the outline of the 3D object model into the image. For both operators, an object-camera pose is needed, which can be determined, e.g., by the operator create_cam_pose_look_at_point. Another common visualization task is to display the pose that is obtained by find_shape_model_3d.

For general information about visualizing results see the description of the method Visualization on page 269.

10.3 Programming Examples

This section shows how to use HALCON for 3D matching.

10.3.1 Recognize 3D Clamps and Their Poses in Images

Example: examples\hdevelop\Applications\FA\3d_matching_clamps.dev

This example shows how to find a 3D object, in particular the clamp shown in Figure 10.1 on page 143, in images using a shape-based 3D matching. Besides the basic steps introduced in the first example, advanced steps are applied and a procedure is provided that helps to interactively determine an appropriate pose range for the creation of the 3D shape model.

At the beginning of the program, you can select by commenting or uncommenting the corresponding code lines whether you want to use an already created 3D shape model or whether you want to create a new one. Further, the camera parameters are assigned to a variable and the 3D object model of the clamps is loaded.

```
ReCreateShapeModel3D := false
* ReCreateShapeModel3D := true
CamParam := [0.01221,-2791,7.3958e-6,7.4e-6,308.21,245.92,640,480]
read_object_model_3d_dxf ('clamp_sloped', 'mm', [], [], ObjectModel3DID,
                          DxfStatus)
```

The procedure inspect_object_model_3d can now be used to interactively visualize the model, set the reference pose, and determine further parameters for the creation of the 3D shape model. For example, selected images that were acquired before calling the procedure can be succesively added to the visualization by selecting the buttons Next Image or Previous Image. With the mouse, you can rotate the 3D object model and, after selecting the mouse mode Move Image, move the underlying image so that the object model matches the image (see Figure 10.6). The returned pose for each image can be

added to the pose range by the button `Add to Pose Range`. When leaving the procedure by selecting `Exit`, the procedure returns values for a pose range that can be used to create a 3D shape model.

```
gen_empty_obj (Images)
for Index := 1 to 3 by 1
  read_image (Image, 'clamp_sloped/clamp_sloped_'+Index$'02')
  concat_obj (Images, Image, Images)
endfor
inspect_object_model_3d (Images, ObjectModel3DID, CamParam, RefRotX,
                         RefRotY, RefRotZ, LongitudeMin, LongitudeMax,
                         LatitudeMin, LatitudeMax, CamRollMin, CamRollMax,
                         DistMin, DistMax, MinFaceAngle)
```

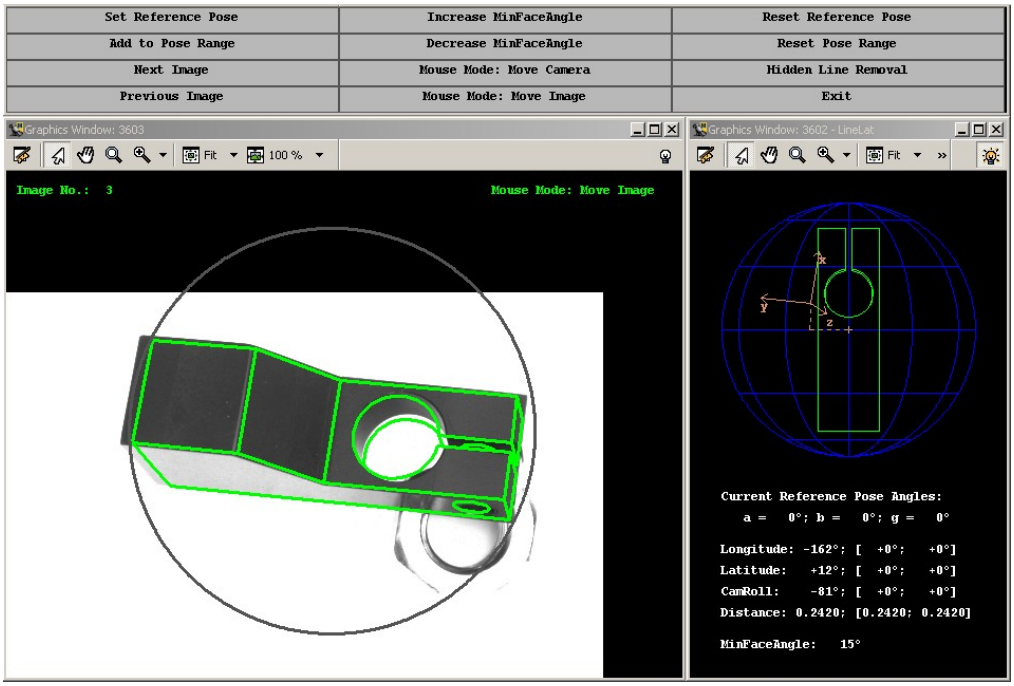


Figure 10.6: Interactive determination of the reference pose and the pose range for the 3D matching of clamps.

Additionally, within the procedure a `Hidden Line Removal` mode can be selected and the minimum face angle (`Increase MinFaceAngle`, `Decrease MinFaceAngle`) can be set. The latter is used, e.g., to suppress edges that are needed to approximate curved structures in the 3D object model but do not appear in the image and thus cannot be matched in the image (see Figure 10.7).

Within the main procedure, the operator `create_shape_model_3d` creates a new 3D shape model if the variable `ReCreateShapeModel3D` was set to `true`. For this, the parameters determined by the procedure described above are used. The resulting 3D shape model is written into file using `write_shape_model_3d`. If an already existing 3D shape model is used (`ReCreateShapeModel3D` set

to `false`) and the procedure was used mainly for visualization purposes, the existing 3D shape model is read from file using `read_shape_model_3d`. Since the 3D shape model is available now, we do not need the 3D object model anymore. It is destroyed by `clear_object_model_3d`.

```
if (ReCreateShapeModel3D)
  create_shape_model_3d (ObjectModel3DID, CamParam, RefRotX, RefRotY,
                         RefRotZ, 'gba', LongitudeMin, LongitudeMax,
                         LatitudeMin, LatitudeMax, 0, rad(360), DistMin,
                         DistMax, 10, 'min_face_angle', MinFaceAngle,
                         ShapeModel3DID)
  write_shape_model_3d (ShapeModel3DID, 'clamp_sloped_user.sm3')
else
  read_shape_model_3d ('clamp_sloped_35.sm3', ShapeModel3DID)
  endif
clear_object_model_3d (ObjectModel3DID)
```

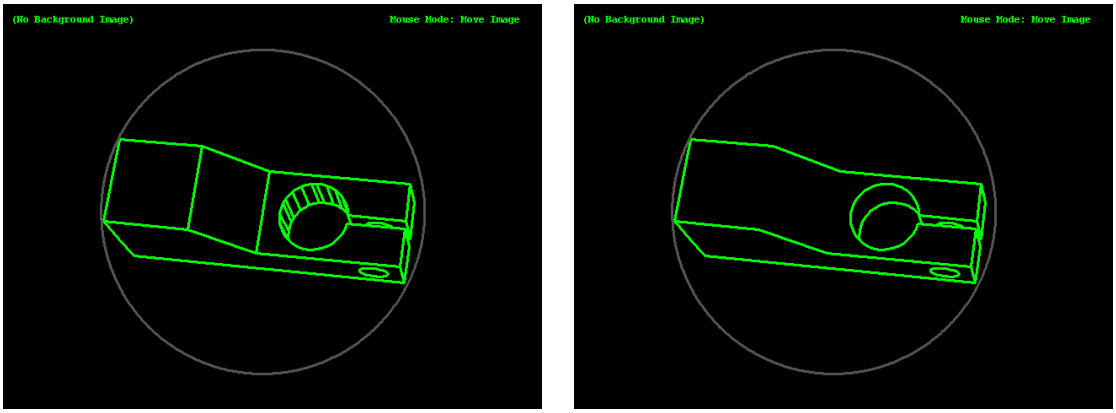


Figure 10.7: Different minimum face angles: (left) 8 degrees, (right) 45 degrees.

For the actual 3D matching, all images that contain the clamps are read and for each image the operator `find_shape_model_3d` searches for instances of the 3D shape model in the image. With the resulting values, the edges of the 3D shape model are projected into the image using `project_shape_model_3d`. The numerical values of the pose are displayed using the procedure `display_match_pose`.

```
for ImageNo := 1 to 35 by 1
  read_image (Image, 'clamp_sloped/clamp_sloped_'+ImageNo$'02')
  find_shape_model_3d (Image, ShapeModel3DID, 0.7, 0.9, 0, 'num_matches',
                       2, Pose, CovPose, Score)
  for I := 0 to |Score|-1 by 1
    PoseI := Pose[I*7:I*7+6]
    CovPoseI := CovPose[I*6:I*6+5]
    ScoreI := Score[I]
    project_shape_model_3d (ModelContours, ShapeModel3DID, CamParam,
                            PoseI, 'true', 0.261799)
    display_match_pose (ShapeModel3DID, PoseI, WindowHandle)
    endfor
  endfor
```

Inside the procedure `display_match_pose` the reference point and the camera parameters are queried by `get_shape_model_3d_params` to align the displayed strings in the image.

```
  get_shape_model_3d_params (ShapeModel3DID, 'reference_point',
                             ReferencePoint)
  get_shape_model_3d_params (ShapeModel3DID, 'cam_param', CamParam)
  pose_to_hom_mat3d (Pose, HomMat3D)
  affine_trans_point_3d (HomMat3D, ReferencePoint[0], ReferencePoint[1],
                         ReferencePoint[2], X, Y, Z)
  project_3d_point (X, Y, Z, CamParam, Row, Column)
  set_tposition (WindowHandle, Row, Column - 10)
  write_string (WindowHandle, 'Pose:')
  set_tposition (WindowHandle, Row+15, Column)
  write_string (WindowHandle, 'X: '+(1000*Pose[0])$'4.1f'+' mm')
  ... etc. ...
```

At the end of the program, the shape model is destroyed by the operator clear_shape_model_3d.

```
clear_shape_model_3d (ShapeModel3DID)
```

10.4 Selecting Operators

10.4.1 Read 3D Object Model

Standard:

read_object_model_3d_dxf

10.4.2 Inspect 3D Object Model

Standard:

get_object_model_3d_params

10.4.3 Create 3D Shape Model

Standard:

create_shape_model_3d

Advanced:

convert_point_3d_cart_to_spher, convert_point_3d_spher_to_cart

10.4.4 Destroy 3D Object Model

Standard:

clear_object_model_3d

10.4.5 Inspect 3D Shape Model

Standard:

get_shape_model_3d_contours, get_shape_model_3d_params

10.4.6 Re-use 3D Shape Model

Standard:

write_shape_model_3d, read_shape_model_3d

10.4.7 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

10.4.8 Find 3D Shape Model

Standard:

find_shape_model_3d

10.4.9 Visualize Results

Standard:

project_object_model_3d, project_shape_model_3d

Further operators can be found in the operator list for the method Visualization (see section 18.4 on page 283).

10.4.10 Destroy 3D Shape Model

Standard:

clear_shape_model_3d

10.5 Tips & Tricks

10.5.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is useful for 3D matching. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the 3D object is searched for can be restricted, the faster and more robust is the search. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

10.5.2 Speed up

Besides using Regions of Interest, you can speed up the 3D matching by adjusting several parameters that are set for the operators introduced above:

- The more the pose range is restricted while creating a 3D shape model with `create_shape_model_3d`, the faster is the search process. But note that only those object instances are found that correspond to the selected pose range.
- If edges are contained in the 3D shape model that are not visible in the image, the performance, i.e., the robustness and speed, of the 3D matching decreases. Especially for objects that contain curved surfaces, which are approximated by multiple planar faces in the 3D object model (see Figure 10.7 on page 153), the generic parameter `min_face_angle` should be adjusted within `create_shape_model_3d` to eliminate unnecessary edges for the 3D shape model. To check which edges are visible with a specific minimum face angle, you can display the corresponding contours with the operator `project_object_model_3d`.
- In `create_shape_model_3d` as well as in `find_shape_model_3d`, the number of used pyramid levels can be set by the parameter `num_levels`. To speed up the search process, the value should be as high as possible, but the object should be still recognizable in the model. To check a view on the object in a specific pyramid level, you can query the corresponding contours by the operator `get_shape_model_3d_contours`.
- In `find_shape_model_3d` you can adjust the parameter `MinScore`. A large `MinScore` speeds up the search process, but allows less occlusions for the object, so that possibly some objects are not recognized.
- In `find_shape_model_3d` you can adjust the parameter `Greediness`. A large `Greediness` speeds up the search process. But because the search becomes less robust, possibly some objects are not recognized.
- In `find_shape_model_3d` you can adjust the generic parameter `pose_refinement`. If it is set to `none`, the search is fast but the pose is determined with limited accuracy. A tradeoff between run time and accuracy is to set the pose refinement to `least_squares_high`. This is recommended for most applications.
- In `find_shape_model_3d` you can adjust the generic parameter `border_model`. If it is set to `true`, also objects that extend beyond the image borders can be found. Because this is rather time-consuming, if you only search for objects that are completely contained in the image, it is recommended to leave the default value `false` unchanged.
- When visualizing the 3D object model or the 3D shape model by projecting them into the image using `project_object_model_3d` or `project_shape_model_3d`, respectively, you can speed up the visualization by setting the parameter `HiddenLineRemoval` to `false`. Then, also those edges of the model are visualized that would be hidden by faces.

Additionally, a reduced resolution of the image can speed up the online as well as the offline phase of the 3D matching, because less 2D projections have to be generated. Note that if you reduce the resolution of the images (e.g., from 1 Megapixel to 640x480 pixels), you must also change the camera parameters used for the creation of the 3D shape model by adapting `Sx`, `Sy`, `Cx`, `Cy`, `ImageWidth`, and `ImageHeight` (see description of `camera_calibration` in the Reference Manual). For example, if the

image is scaled down by a factor of 0.5, Sx and Sy must be multiplied by 2, whereas Cx, Cy, ImageWidth, and ImageHeight must be multiplied by 0.5.

Finally, you can speed up the 3D matching by using a homogeneous background for the object during the image acquisition.

10.5.3 Robust 3D matching

For a robust 3D matching it is important that the edges of the object are cleary visible in the image. Thus, the following general tips may help you to enhance your application already when taking the images of the object:

- To get clearly visible edges of the object in your images, take special care of the lighting conditions during image acquisition. In general, the edges that are included in the 3D shape model should also be visible in the image. The edges that are included in the model can be adjusted with the generic parameter min_face_angle of create_shape_model_3d. A more convenient way to visualize the effect of this parameter is to use the procedure inspect_object_model_3d, which can be found in the example program examples\hdevelop\Applications\FA\3d_matching_clamps.dev.
- If possible, use a background with a good contrast to the object, so that the background can be cleary separated from the object.
- Multi-channel images contain more information and thus may lead to a more robust edge extraction.

Chapter 11

Variation Model

The main principle of a variation model is to compare one or more images to an ideal image to find significant differences. With this, you can, e.g., identify incorrectly manufactured objects by comparing them to correctly manufactured objects. The ideal image is often obtained by a training using several reference images. Besides the ideal image, the training derives information about the allowed gray value variation for each point of the image. This information is stored in a so-called variation image. Both images are used to create a variation model, to which other images can be compared.

The advantage of the variation model is that images can be directly compared by their gray values and the comparison is spatially weighted by the variation image.

11.1 Basic Concept

Variation Model mainly consists of the following parts:

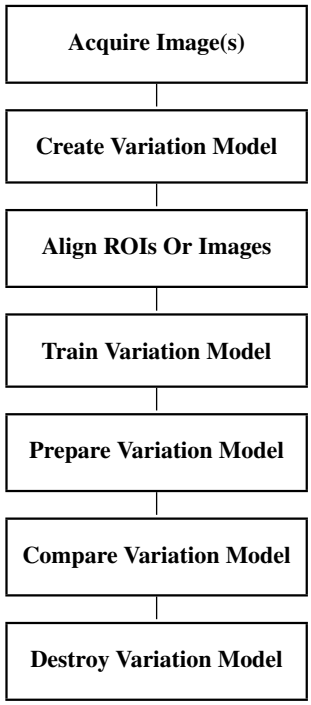
11.1.1 Acquire Image(s)

Both for the training and for the comparison, images are acquired.

For detailed information see the description of this method on page 13.

11.1.2 Create Variation Model

First, you have to create the variation model used for the image comparison by the operator `create_variation_model`. It stores information that is successively added during the following steps.



11.1.3 Align ROIs Or Images

For training the variation model, all training images must be placed in the same position and with the same orientation. Thus, before training the model the objects must be aligned. Similarly, the images that are to be compared to the variation model must be aligned.

How to perform alignment using shape-based matching is described in the Solution Guide II-B in section 4.3.4 on page 37.

11.1.4 Train Variation Model

The variation model can be trained by providing a set of images containing good samples. The operator `train_variation_model` uses the training images to calculate an ideal image and a corresponding variation image. The variation image represents the amount of gray value variation, i.e., the tolerance, at every point of the image.

If you do not want to train the model with multiple images, e.g., for memory reasons, you can also use a single image as ideal image, but then you need knowledge about the spatial distribution of the variations. If you know, e.g., that the acceptable tolerances are near the edges of your object, you can create the variation image by applying an edge filter to the ideal image. For this proceeding, the training step is skipped.

11.1.5 Prepare Variation Model

To prepare the variation model for the image comparison, the ideal image and the variation image are converted into two threshold images. If you have trained the model with several training images using `train_variation_model`, you prepare the model with the operator `prepare_variation_model`. If you use a single image as ideal image and create the corresponding variation image manually by filtering the ideal image, e.g., using `sobel_amp`, `edges_image`, or `gray_range_rect`, you have to use `prepare_direct_variation_model` for the preparation because the ideal image and the variation image are not yet connected to the variation model.

The obtained threshold images can be directly read out using the operator `get_thresh_images_variation_model`.

11.1.6 Compare Variation Model

You compare an image to the prepared variation model with the operator `compare_variation_model`. There, the two thresholds obtained during the preparation step (and stored in the variation model) are used to determine a region containing all points of the image that significantly differ from the model. Extended parameter settings for the comparison are available when using `compare_ext_variation_model` instead.

11.1.7 Destroy Variation Model

When you no longer need the variation model, you destroy it with the operator `clear_variation_model`.

11.1.8 A First Example

Example: `examples\hdevelop\Applications\FA\print_check.dev`

An example for this basic concept is the following program. Here, the logos on the pen clips depicted in figure 11.1 are inspected using a variation model. The model is trained successively by a set of images containing correct prints. As the pen clips move from image to image, in each image the pen clip is located and aligned by shape-based matching. After the training, the variation model is prepared for the comparison with other images.

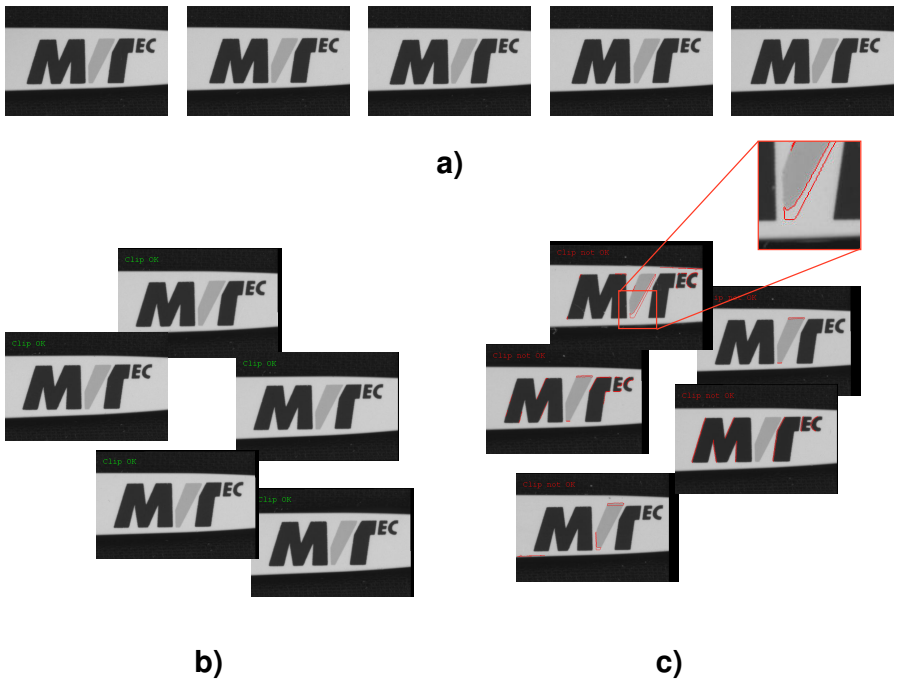


Figure 11.1: Inspection of the logo that is printed on the pen clip: (a) training images; (b) accepted images; (c) rejected images.

```
create_shape_model (ImageReduced, 5, rad(-10), rad(20), 'auto', 'none',
                    'use_polarity', 20, 10, ShapeModelID)
create_variation_model (Width, Height, 'byte', 'standard',
                        VariationModelID)
for I := 1 to 15 by 1
  read_image (Image, 'pen/pen-'+I$'02d')
  find_shape_model (Image, ShapeModelID, rad(-30), rad(60), 0.5, 1, 0.5,
                    'least_squares', 0, 0.9, Row, Column, Angle, Score)
  if (|Score| = 1)
    vector_angle_to_rigid (Row, Column, Angle, RowRef, ColumnRef, 0,
                           HomMat2D)
    affine_trans_image (Image, ImageTrans, HomMat2D, 'constant',
                        'false')
    train_variation_model (ImageTrans, VariationModelID)
  endif
endfor
get_variation_model (MeanImage, VarImage, VariationModelID)
prepare_variation_model (VariationModelID, 20, 3)
```

During the inspection phase, also the images that are to be inspected are aligned by shape-based matching. Then, the images are compared to the variation model to check the print for errors. The erroneous regions are extracted and displayed. At the end of the program, the handles for the shape model and the

variation model are destroyed.

```
find_shape_model (Image, ShapeModelID, rad(-10), rad(20), 0.5, 1, 0.5,
                  'least_squares', 0, 0.9, Row, Column, Angle, Score)
if (|Score| = 1)
  vector_angle_to_rigid (Row, Column, Angle, RowRef, ColumnRef, 0,
                         HomMat2D)
  affine_trans_image (Image, ImageTrans, HomMat2D, 'constant',
                      'false')
  reduce_domain (ImageTrans, RegionROI, ImageReduced)
  compare_variation_model (ImageReduced, RegionDiff,
                           VariationModelID)
  connection (RegionDiff, ConnectedRegions)
  select_shape (ConnectedRegions, RegionsError, 'area', 'and', 20,
                1000000)
  dev_display (ImageTrans)
  endif
clear_shape_model (ShapeModelID)
clear_variation_model (VariationModelID)
```

11.2 Extended Concept

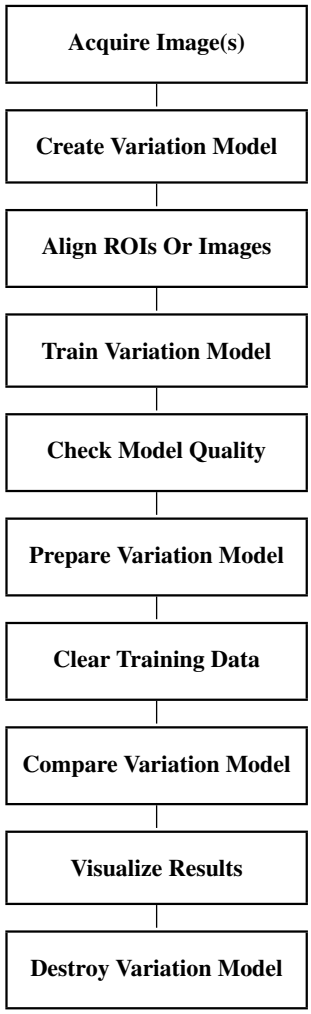
Often more than the essential steps are necessary. In many cases, after the comparison, a visualization of the result is required. Additionally, you might want to check the quality of the training or you need to save memory. The advanced steps are described in the following sections.

11.2.1 Check Model Quality

After training a variation model with several images, the image of the ideal object and the corresponding variation image can be queried by the operator `get_variation_model` to check if the images used for the training all contained similar objects. If the variation image contains large variations in areas that should exhibit no variations, this leads to the conclusion that at least one of the training images contained a bad object.

11.2.2 Clear Training Data

After the preparation of the variation model, you can reduce the amount of memory needed for the variation model by applying the operator `clear_train_data_variation_model`. But this is only recommended if you do not need to use the variation model for anything else than for the actual comparison anymore. A further training or the application of `get_variation_model` is not possible after deleting the training data.



11.2.3 Visualize Results

A typical visualization task is to display the image and mark the parts of the image that do not correspond to the model. These parts can be explicitly extracted by applying `connection` to separate the connected components of the region obtained by the image comparison and afterwards selecting the regions that are within a specific area range by `select_shape`.

For detailed information see the description of this method on page 269.

11.3 Programming Examples

This section gives a brief introduction to using HALCON for variation model.

11.3.1 Inspect a Printed Logo Using a Single Reference Image

Example: examples\solution_guide\basics\variation_model_single.dev

This example inspects the same logo as the first example, but instead of training the variation model with multiple images, a single image is used as ideal image.

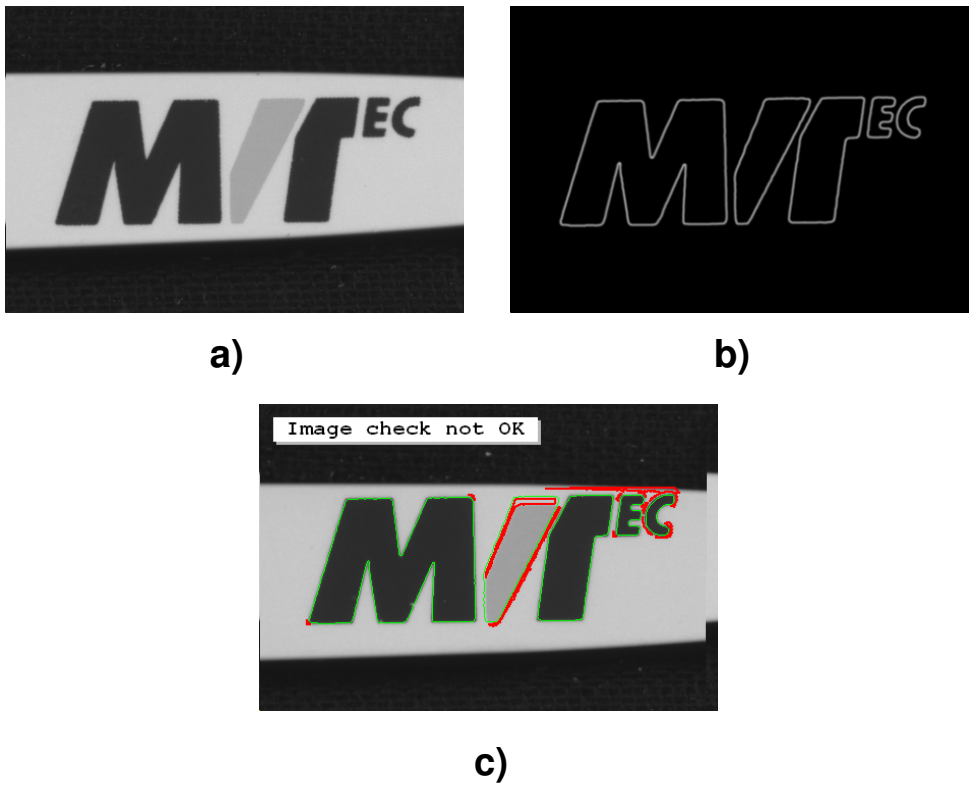


Figure 11.2: Model for the inspection of the pen clips: a) ideal image; b) variation image; c) inspected image with displayed errors.

From the ideal image a shape model is derived for the later alignment and the variation image is determined manually. To obtain a variation image with tolerances at the edges of the object, the edges in the domain of the corresponding region of interest (ImageReduced) are extracted by the operator edges_sub_pix. These edges are converted into regions and the regions are enlarged by a dilation to create a generic image that contains the slightly enlarged boundaries of the object. As the conversion of subpixel-precise edges into pixel-precise regions leads to aliasing, the edges are scaled before applying the conversion. After the region processing, the generic image is zoomed back to the original size using a weighting function, which additionally smoothes the borders of the regions. A further smoothing is realized by the operator binomial_filter. The image is used now as variation image (see figure 11.2) and the variation model is prepared for the image comparison.

```
create_shape_model (ImageReduced, 'auto', -rad(10), rad(20), 'auto',
                    'auto', 'use_polarity', [40,50], 40, ShapeModelID)
edges_sub_pix (ImageReduced, Edges, 'sobel_fast', 0.5, 10, 20)
hom_mat2d_identity (HomMat2DIdentity)
hom_mat2d_scale (HomMat2DIdentity, 4, 4, 0, 0, HomMat2DScale)
affine_trans_contour_xld (Edges, ZoomedEdges, HomMat2DScale)
gen_image_const (VarImageBig, 'byte', 4*Width, 4*Height)
count_obj (ZoomedEdges, NEdges)
for i := 1 to NEdges by 1
  select_obj (ZoomedEdges, ObjectSelected, i)
  get_contour_xld (ObjectSelected, RowEdge, ColEdge)
  gen_region_polygon (Region1, RowEdge, ColEdge)
  dilation_circle (Region1, RegionDilation, 2.5)
  paint_region (RegionDilation, VarImageBig, VarImageBig, 255, 'fill')
endfor
zoom_image_size (VarImageBig, VarImageSmall, Width, Height, 'weighted')
binomial_filter (VarImageSmall, VarImage, 3, 3)
create_variation_model (Width, Height, 'byte', 'direct', VarModelID)
prepare_direct_variation_model (Image, VarImage, VarModelID, 15, 4)
```

During the inspection, in each image to be checked the boundary of the object is searched for to align the image to the ideal image. The aligned image is then compared to the variation model using compare_ext_variation_model. As the mode absolut is set, you can alternatively use the operator compare_variation_model. Differing regions of a certain size are obtained and stored in NDefects.

```
for i := 1 to 30 by 1
  read_image (Image, 'pen/pen-'+i$'02d')
  find_shape_model (Image, ShapeModelID, -rad(10), rad(20), 0.5, 1, 0.5,
                    'least_squares', 0, 0.9, Row, Column, Angle, Score)
  if (|Score| # 0)
    vector_angle_to_rigid (Row, Column, Angle, ModelRow, ModelColumn,
                           0, HomMat2D)
    affine_trans_image (Image, ImageAffinTrans, HomMat2D, 'constant',
                        'false')
    reduce_domain (ImageAffinTrans, LogoArea, ImageReduced1)
    compare_ext_variation_model (ImageReduced1, RegionDiff, VarModelID,
                                 'absolute')
    connection (RegionDiff, ConnectedRegions)
    select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 10,
                  99999)
    count_obj (SelectedRegions, NDefects)
    endif
  endfor
```

Finally, the handle for the variation model is deleted.

```
clear_shape_model (ShapeModelID)
clear_variation_model (VarModelID)
```

11.3.2 Inspect a Printed Logo under Varying Illumination

Example: `examples\solution_guide\basics\variation_model_illumination.dev`

This example inspects caps of bottles using a variation model (see figure 11.3). The difficulty here is the changing illumination during the inspection. Like in the example described before, a single image is used as ideal image.

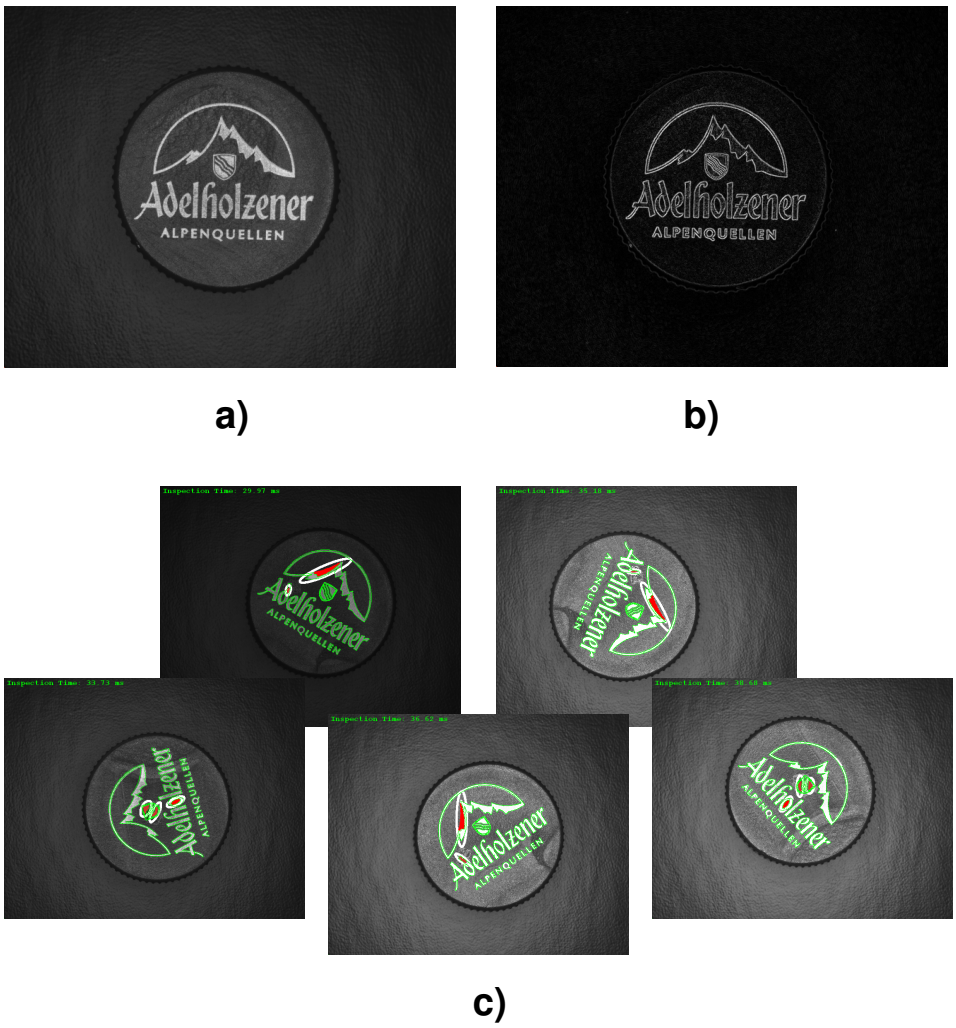


Figure 11.3: Model for the inspection of the caps: a) ideal image; b) variation image; c) results of image comparison for images taken under different illumination conditions.

The variation model here is obtained by filtering the ideal image with the operator `sobel_amp`.

```
create_variation_model (Width, Height, 'byte', 'direct', VariationID)
sobel_amp (ModelImage, VarImage, 'sum_abs', 5)
prepare_direct_variation_model (ModelImage, VarImage, VariationID, [20,25],
                                [1.6,1.6])
```

To compensate for the changing illumination, inside the procedure `get_grayval_range` the gray value range of the variation model is determined.

```
get_grayval_range (ModelImage, RegionROI, RegionForeground,
                   RegionBackground, BackgroundGVModel, ForegroundGVModel)
```

Inside the procedure the ideal image is reduced to the domain of the region of interest that encloses the print on the cap. Then, the reduced image is split into foreground and background using `bin_threshold` and `difference`. Finally, for both regions the mean and standard deviation of the gray values is queried by the operator `intensity`.

```
reduce_domain (Image, RegionROI, ImageReduced)
bin_threshold (ImageReduced, RegionBackground)
difference (RegionROI, RegionBackground, RegionForeground)
intensity (RegionForeground, Image, ForegroundGVal, DeviationFG)
intensity (RegionBackground, Image, BackgroundGVal, DeviationBG)
```

The inspection of the caps is realized inside the procedure `inspect_cap`.

```
inspect_cap (rImage, RegionROI, WindowHandle, ModelID, VariationID,
             RowModel, ColumnModel, BackgroundGVModel,
             ForegroundGVModel)
```

There, in each image the object is searched for and aligned as described in the previous example. Then, the procedure `get_grayval_range` derives also the gray value range for the ROI of the object to be inspected, so that the image can be adapted to the gray value range of the variation model before comparing it to the variation model. Thus, also images that are taken under varying illumination can be compared. The mode for the image comparison using the oparator `compare_ext_variation_model` is set to `light_dark`, so that separate regions for too bright and too dark image points are returned in `RegionDiff`.

```
get_grayval_range (ImageAffinTrans, RegionROI,
                   RegionForegroundImage, RegionBackgroundImage,
                   BackgroundImage, ForegroundImage)
Mult := (ForegroundGVModel - BackgroundGVModel)/ (ForegroundImage
        - BackgroundImage)
Add := ForegroundGVModel - Mult*ForegroundImage
scale_image (ImageReduced, ImageScaled, Mult, Add)
compare_ext_variation_model (ImageScaled, RegionDiff, VariationID,
                             'light_dark')
```

At the end of the program, again the handle for the variation model is deleted.

```
clear_variation_model (VariationID)
```

11.3.3 Other Examples

HDevelop

- examples\hdevelop\Applications\FA\print_check.dev
  Perform a typical print quality inspection using a variation model
  → description here in the Solution Guide Basics on page 161
- examples\hdevelop\Applications\FA\print_check_single_chars.dev
  Perform a typical print quality inspection using variation models for each character

C++

- examples\cpp\source\pen.cpp
  Inspect the quality of print on a pen using the variation model of HALCON

11.4 Selecting Operators

11.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

11.4.2 Create Variation Model

Standard:

create_variation_model

11.4.3 Align ROIs Or Images

Operators for aligning ROIs or images are described in the Solution Guide II-B.

11.4.4 Train Variation Model

Standard:

train_variation_model

11.4.5 Check Model Quality

Standard:

get_variation_model

11.4.6 Prepare Variation Model

Standard:

prepare_variation_model

Advanced:

prepare_direct_variation_model, get_thresh_images_variation_model, sobel_amp, edges_image, gray_range_rect

11.4.7 Clear Training Data

Standard:

clear_train_data_variation_model

11.4.8 Compare Variation Model

Standard:

compare_variation_model

Advanced:

compare_ext_variation_model

11.4.9 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

11.4.10 Destroy Variation Model

Standard:

clear_variation_model

Chapter 12

Classification

Classification is the technical term for the assignment of objects to individual instances of a set of classes. The objects as well as the available classes are described by specific features, e.g., the color of a pixel or the shape of a region. To define the classes, the features have to be specified, e.g., by a training that is based on known objects. After the training, the classifier compares the features of the object with the features associated to the available classes and returns the class with the largest correspondence. Depending on the selected classifier, possibly additional information about the probabilities of the classes or the confidence of the classification is given.

Generally, two approaches for a classification of image data can be distinguished. One approach is based on a pure pixel classification and segments images based on color or texture. The other approach is more general and classifies arbitrary features, i.e., you can additionally classify regions based on region features like, e.g., shape, size, or color. For the programming examples introduced here, the focus is on the first approach. Note that actually the optical character recognition (OCR) provided by HALCON is a further classification approach, for which specific operators are provided. But these are described in more detail in the chapter OCR on page 233.

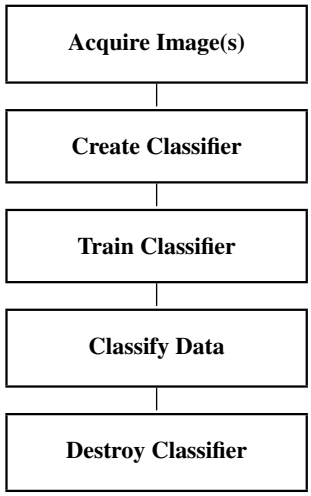
HALCON provides different classifiers. The most important classifiers are the neural network (multi-layer perceptron or MLP) classifier, a classifier that is based on support vector machines (SVM), and a classifier that is based on Gaussian mixture models (GMM). Additionally, a box classifier is available, but as the GMM classifier leads to comparable results and is more robust, here only the GMM classifier is described. Further, more 'simple' classifiers can be used for image segmentation. These comprise the classifiers for two-dimensional pixel classification with `class_2dim_sup` or `class_2dim_unsup`, and the n-dimensional pixel classification approaches based on hyper-cuboids (`class_ndim_box`), which is related to the box classifier, or hyper-spheres (`class_ndim_norm`), which can be used, e.g., for an euclidean classification. These approaches are less flexible, so the focus of this chapter is on the MLP, SVM, and GMM classifiers. If you want to use one of the 'simple' classifiers, have a look at the corresponding examples.

To decide which classifier to use for a specific task and to choose suitable parameters for the selected classification is rather challenging, as each classification task is different and thus needs different handling. HALCON provides the operators to apply a classification, and this chapter provides some general hints about their usage. Further, the most salient advantages and disadvantages of the different classifiers

are summarized for the step Create Classifier and minor differences between the classifiers, which are related to individual operators, are stated with the corresponding step descriptions or in the corresponding sections of the Reference Manual. Nevertheless, in contrast to other methods described in this manual, for classification no set of fixed rules for the selection of the suitable classifier and the selection of the parameters can be given. In many cases, you have to try out different classifiers for your specific task and with your specific training data. Independent on the selected approach, in almost any case you have to play with the parameters to get a satisfying result. Allow extra time for tests and do not despair if your first tests do not immediately lead to a satisfying result. Classification is complex!

12.1 Basic Concept

Classification consists of the following basic steps:



12.1.1 Acquire Image(s)

When classifying images, both for the generation of the training data and for the classification images must be acquired.

For detailed information see the description of this method on page 13.

12.1.2 Create Classifier

The first step of a classification is the creation of a new classifier. Here, you have to decide which classifier to apply for your specific classification task. The main advantages and disadvantages of the three classifiers (assuming an optimal tuning of the parameters) are as follows:

- Gaussian Mixture Models: The advantage of the GMM classification is that, controlled by the parameter settings, a feature vector that does not match to one of the trained classes can be assigned to a rejection class. Additionally, you can apply a second training that is appended to the first training, e.g., in order to add new training samples. The disadvantage of the GMM classifier is that the recognition rates are not as good as the recognition rates obtained by the MLP or SVM approaches. Further, for the feature vector a length of up to 15 features is recommended, whereas for the MLP and SVM classifiers a feature vector with 500 features is still realistic.
- Multi-Layer Perceptron: The MLP classifier leads to good recognition rates and is rather fast at classification. In exchange, the training is not as fast as for the SVM classification, especially for huge training sets. If the classification is time critical but the training can be applied offline, the MLP approach is a good choice. A rejection class is returned, but in comparison to the GMM classifier, it may be infuenced by outliers, so that an additional, explicit training of a rejection class is recommended. Additionally, if you want to add additional training samples, you should not append a second training, but repeat the training with both the old and the new training samples.
- Support Vector Machines: Compared to the MLP classifier, the SVM classifier leads to slightly better recognition rates and is faster at training, especially for huge training sets. Additionally, a training of new training samples can be simply appended to a previous training. In exchange, the classification is not as fast as for the MLP approach. A rejection class is not obtained automatically.

Dependent on the selected classification approach, you create the classifier using `create_class_gmm`, `create_class_mlp`, or `create_class_svm`.

When creating a classifier, the correct parameter settings are essential for the success of the later classification. These are very specific for the different approaches. To name just a few, the parameter `OutputFunction` should be set to `'softmax'` for an MLP classifier, as the classifier can also be used for regression and not only for classification. Additionally, the value for the parameter `NumHidden` has to be adjusted very carefully. In contrast to that, for the SVM approach, the parameter `KernelType` in most cases should be set to `'rbf'` and the values of the parameters `Nu` and `KernelParam` strongly influence the classification result and thus should be carefully adjusted. Further, different modes and specific algorithms for the preprocessing of the feature vector can be selected. For details, see the corresponding operator descriptions in the Reference Manual.

12.1.3 Train Classifier

The training consists of two important steps: First, representative training samples, i.e., a set of feature vectors for each class, are added to the classifier using the operator `add_sample_class_gmm`, `add_sample_class_mlp`, or `add_sample_class_svm`. If you want to apply a pixel-based classification, i.e., you want to segment images into regions of different color or texture, you can alternatively add the samples using the operators `add_samples_image_class_gmm`, `add_samples_image_class_mlp`, or `add_samples_image_class_svm`. Then, you can immediately insert a single multi-channel image instead of many feature vectors (one per pixel).

In a second step, the classifier is trained by `train_class_gmm`, `train_class_mlp`, or `train_class_svm`. Note that complex calculations are performed during the training. Thus, depending on the number of training samples and some approach-specific influences, e.g., the size of the MLP used for an MLP classification, the training can take from a few seconds to several hours.

12.1.4 Classify Data

For the general classification you use the operators classify_class_gmm, classify_class_mlp, or classify_class_svm. These use the trained classifier to return the classes that are most probable for the feature vector of the data to classify. In most cases, you need only the best class. Then, you set the parameter Num to '1'. In a few cases, e.g., when working with overlapping classes, also the second best class may be of interest (Num='2'). Like for the step of adding samples to the training, for a pixel-based classification, i.e., if an image is to be segmented into regions of different color or texture classes, special operators are provided. For image segmentation, you can use classify_image_class_gmm, classify_image_class_mlp, or classify_image_class_svm. Again, these take images as input, instead of individual feature vectors for each pixel, and return regions as output.

Besides the most probable classes, for the GMM classification the probabilities of the classes, and for the MLP classification the confidence of the classification are returned. Note that the probabilities for the GMM classification are relatively precise, whereas for the confidence of the MLP classification outliers are possible because of the way an MLP is calculated. For example, when classifying positions of objects (rows and columns build the feature vectors), the confidence may be high for positions that are far from the training samples and low in case of overlapping classes.

For the SVM classification, no information about the confidence of the classification or the probabilities of the returned classes are given.

12.1.5 Destroy Classifier

When you no longer need the classifier, you destroy it with the operator clear_class_gmm, clear_class_mlp, or clear_class_svm, dependent on the used classifier. When working with several classifiers of the same type, you can destroy them all in one step using clear_all_class_gmm, clear_all_class_mlp, or clear_all_class_mlp.

12.1.6 A First Example

An example for this basic concept is the following program, which segments the image depicted in figure 12.1 into regions of the three classes Sea, Deck, and Walls using a pixel-based GMM classification. Parts of the image that can not be clearly assigned to one of the specified classes are returned as a rejection class region.

First, the image is read and the sample regions for the classes Sea, Deck, and Walls are defined and listed in the tuple Classes.

```
read_image (Image, 'patras')
gen_rectangle1 (Sea, 10, 10, 120, 270)
gen_rectangle2 (Deck, [170,400], [350,375], [-0.56192,-0.75139], [64,104],
                [26,11])
union1 (Deck, Deck)
gen_rectangle1 (Walls, 355, 623, 420, 702)
concat_obj (Sea, Deck, Classes)
concat_obj (Classes, Walls, Classes)
```

Figure 12.1: Segmentation of a color image: (a) original image, (b) sample regions for the classes Sea (white), Deck (light gray), and Walls (gray), (c) regions returned by the classification (black marks the rejected regions).

Then, a new classifier of type Gaussian Mixture Model is created with `create_class_gmm`. The classifier works for a feature vector with three dimensions (the three channels of the color image) and for three classes (`Sea`, `Deck`, and `Walls`). The tuple containing the regions that specify each class to be trained is assigned as a set of samples to the classifier using the operator `add_samples_image_class_gmm`. The samples are used now by the operator `train_class_gmm` to train the classifier.

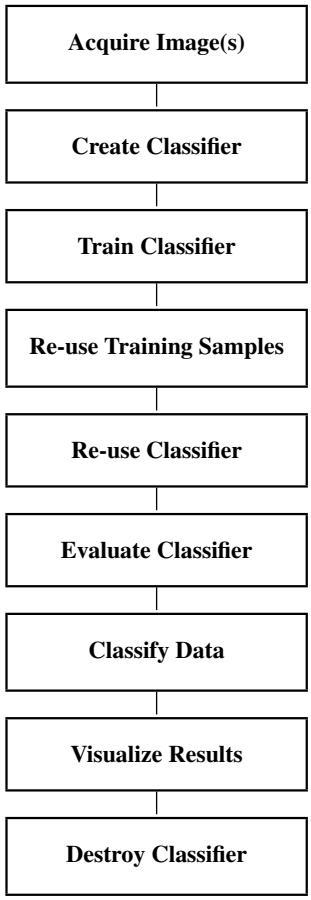
```
create_class_gmm (3, 3, [1,10], 'full', 'none', 2, 42, GMMHandle)
add_samples_image_class_gmm (Image, Classes, GMMHandle, 2.0)
train_class_gmm (GMMHandle, 500, 1e-4, 'uniform', 1e-4, Centers, Iter)
```

Then, the whole image is segmented by `classify_image_class_gmm` into regions of the three trained classes and a rejection class, which consists of the parts that can not be assigned clearly to one of the trained classes. A rejection class is returned automatically for GMM and MLP classification. For the MLP approach the rejection class can be influenced by outliers, so it should be created explicitly like shown in the example `examples\hdevelop\Segmentation\Classification\classify_image_class_mlp.dev`. At the end of the program, the classifier is destroyed with `clear_class_gmm`.

```
classify_image_class_gmm (Image, ClassRegions, GMMHandle, 0.0001)
clear_class_gmm (GMMHandle)
```

12.2 Extended Concept

When we look more closely at Classification, further operations can be applied. For example, the training samples or the trained classifier can be stored to file to be re-used in a later step, or the trained classifier can be evaluated further.



12.2.1 Train Classifier

If a classification does not lead to a satisfying result, one way to enhance the recognition rate is to add further training samples to the classifier. For the GMM and the SVM approach, you can add new samples to the classifier and train the classifier again, so that only those classes are trained again for which new training samples were added (for details, see the corresponding sections in the Reference Manual). But if the run time of your application is not critical, we recommend to train the classifier anew, using both the old and the new training samples. For the MLP approach a new training should be applied in any case, as the result of appending a second training most likely is not satisfying.

You may want to access the individual training samples if you have the suspicion that one of them is classified incorrectly. To do so, you apply the operators `get_sample_class_gmm`, `get_sample_class_mlp`, or `get_sample_class_svm`. To get the number of available training samples, the operators `get_sample_num_class_gmm`, `get_sample_num_class_mlp`, and `get_sample_num_class_svm` can be applied.

If the training and the classification is applied in the same process, you can save memory by applying the operators `clear_samples_class_gmm`, `clear_samples_class_mlp`, or

clear_samples_class_svm after the training. These clear the training samples from memory. This step is only recommended if you do not want to access the individual samples in a later step. If the training process and the actual classification is separated (see the step Re-use Classifier), the clearing of the samples is not necessary as the training samples are not stored anyway (unless you store them explicitly, see the step Re-use Training Samples).

For the SVM approach, you can use reduce_class_svm to reduce the number of support vectors after the training. The returned classifier is faster than the originally used classifier.

12.2.2 Re-use Training Samples

In many cases it is necessary to access the samples used for the training in a later step, e.g., if you want to repeat the training with some additional samples or if you want to access individual samples to check their correctness. In most applications, you will separate the offline training and the online classification. Thus, in the classification phase the information about the individual samples is not implicitly available anymore. To nevertheless access the information, we recommend to write the used samples to file during the training phase with the operators write_samples_class_gmm, write_samples_class_mlp, or write_samples_class_svm. To read them from file again, you use the operators read_samples_class_gmm, read_samples_class_mlp, and read_samples_class_svm.

12.2.3 Re-use Classifier

In most applications, you will separate the offline training and the online classification. At the end of the training phase, you typically save the classifier to disk by write_class_gmm, write_class_mlp, or write_class_svm. To re-use it for the classification phase, you read the stored classifier by read_class_gmm, read_class_mlp, or read_class_svm, respectively.

If the training and the actual classification are separated, in the classification phase no information about the creation and training of the classifier is present anymore. To nevertheless get information about the parameters used for the creation of the classifier, the operators get_params_class_gmm, get_params_class_mlp, or get_params_class_svm can be applied. If a preprocessing was used for the feature vectors, which was specified with the creation of the classifier, you can also query the information content of the preprocessed feature vectors of the training samples. For this, you use get_prep_info_class_gmm, get_prep_info_class_mlp, or get_prep_info_class_svm.

12.2.4 Evaluate Classifier

After training a classifier, a feature vector can be evaluated. If you need only its most probable class, or perhaps also the second best class, and the related probabilities or confidences, respectively, these are returned also when applying the actual classification (see Classify Data). But if all probabilities are needed, e.g., to get the distribution of the probabilities for a specific class (see the example description for examples\hdevelop\Classification\Neural-Nets\class_overlap.dev in the Programming Examples section), an evaluation is necessary. For the GMM approach, the evaluation with evaluate_class_gmm returns three different probability values: the a-posteriori probability (stored in the variable ClassProb), the probability density, and the k-sigma error ellipsoid. For the MLP approach,

the result of the evaluation of a feature vector with `evaluate_class_mlp` is stored in the variable `Result`. The values of `ClassProb` and `Result` both can be interpreted as probabilities of the classes. The position of the maximum value corresponds to the class of the feature vector, and the corresponding value is the probability of the class. For the SVM classification, no information about the confidence of the classification or the probabilities of the returned classes are given.

12.2.5 Visualize Results

Finally, you might want to display the result of the classification.

For detailed information see the description of this method on page 269.

12.3 Programming Examples

This section gives a brief introduction to using HALCON for classification.

12.3.1 Inspection of Plastic Meshes via Texture Classification

Example: `examples\hdevelop\Segmentation\Classification\novelty_detection_svm.dev`

This example shows how to apply an SVM texture classification in order to detect deviations from the training data (novelties), in this case defects, in a textured material. The image depicted in figure 12.2 shows one of five images used to train the texture of a plastic mesh and the result of the classification for a mesh with defects.

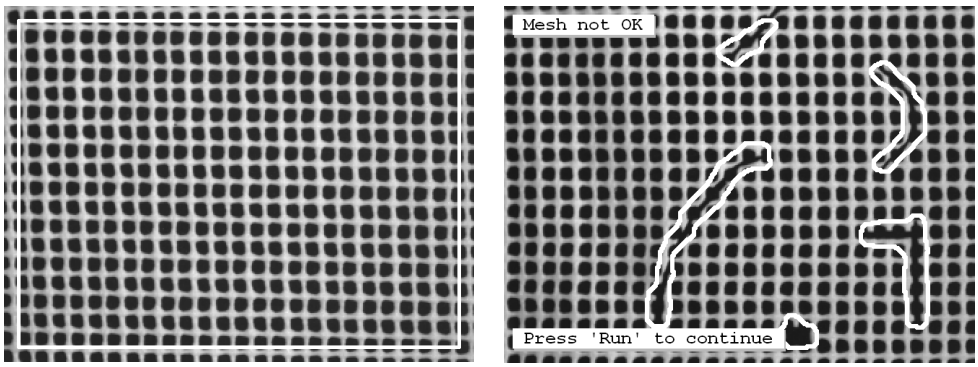


Figure 12.2: Inspection of a plastic mesh: (a) (zoomed) training image with correct texture (region of interest marked in white), (b) mesh with defects (texture novelties marked in white).

The program reads the first training image, defines a region of interest to avoid problems at the image border that would occur because of the later used texture filters and the specific structure of the mesh,

and then creates the SVM classifier SVMHandle using create_class_svm. There, the mode 'novelty-detection' is selected. With this mode, feature vectors that are not contained in the training samples are classified as a new class. Thus, novelties in the texture of the plastic mesh can be detected. For novelty detection, always the kernel type 'rbf' has to be selected, but also for other modes it is recommended in most cases.

```
read_image (Image, 'plastic_mesh/plastic_mesh_01')
gen_rectangle1 (Rectangle, 10, 10, Height/2-11, Width/2-11)
create_class_svm (5, 'rbf', 0.01, 0.0005, 1, 'novelty-detection',
                  'normalization', 5, SVMHandle)
```

Then, all five training images are read in a loop, zoomed, and transformed into texture images by the procedure gen_texture_image.

```
for J := 1 to 5 by 1
  read_image (Image, 'plastic_mesh/plastic_mesh_'+J$'02')
  zoom_image_factor (Image, ImageZoomed, 0.5, 0.5, 'constant')
  gen_texture_image (ImageZoomed, ImageTexture)
```

The procedure creates a texture image as follows: Five different texture filters (texture_laws with different filter types) are applied to the zoomed image and the resulting five images are used to build a five-channel image (compose5). Additionally, the five-channel image is smoothed.

```
  texture_laws (Image, ImageEL, 'el', 5, 5)
  texture_laws (Image, ImageLE, 'le', 5, 5)
  texture_laws (Image, ImageES, 'es', 1, 5)
  texture_laws (Image, ImageSE, 'se', 1, 5)
  texture_laws (Image, ImageEE, 'ee', 2, 5)
  compose5 (ImageEL, ImageLE, ImageES, ImageSE, ImageEE, ImageLaws)
  smooth_image (ImageLaws, ImageTexture, 'gauss', 5)
```

After the creation of each texture image, it is added as training sample to the classifier by add_samples_image_class_svm.

```
  add_samples_image_class_svm (ImageTexture, Rectangle, SVMHandle)
  endfor
```

After all training samples were added to the classifier, the training is applied by train_class_svm. To enhance the speed of the classification, the number of support vectors is reduced by reduce_class_svm, which leads to a new classifier named SVMHandleReduced.

```
train_class_svm (SVMHandle, 0.001, 'default')
reduce_class_svm (SVMHandle, 'bottom_up', 2, 0.001, SVMHandleReduced)
```

Now, the images that have to be inspected are read in a loop. Each image is again zoomed and transformed into a texture image. The image is reduced to the part described by the region of interest and the reduced image is classified using the operator classify_image_class_svm with the reduced classifier SVMHandleReduced.

```
for J := 1 to 14 by 1
  read_image (Image, 'plastic_mesh/plastic_mesh_'+J$'02')
  zoom_image_factor (Image, ImageZoomed, 0.5, 0.5, 'constant')
  gen_texture_image (ImageZoomed, ImageTexture)
  reduce_domain (ImageTexture, Rectangle, ImageTextureReduced)
  classify_image_class_svm (ImageTextureReduced, Errors,
                            SVMHandleReduced)
```

The regions `Errors` that are returned by the classification and which describe the classes with the novelties, can now be visualized in the image. For that, morphological operators are applied to smooth the borders of the error regions, and connected components are extracted. Regions of a certain size are selected and the result of the texture inspection is visualized as text. In particular, the texture of the mesh is classified as being good, if the number of selected regions (`NumErrors`) is `0`. Otherwise, the mesh is classified as being erroneous.

```
  opening_circle (Errors, ErrorsOpening, 3.5)
  closing_circle (ErrorsOpening, ErrorsClosing, 10.5)
  connection (ErrorsClosing, ErrorsConnected)
  select_shape (ErrorsConnected, FinalErrors, 'area', 'and', 300,
                1000000)
  count_obj (FinalErrors, NumErrors)
  if (NumErrors > 0)
    write_message (WindowHandle, 10, 10, 'Mesh not OK', true)
  else
    write_message (WindowHandle, 10, 10, 'Mesh OK', true)
  endif
  if (J < 14)
    write_message (WindowHandle, Height-50, 10,
                   'Press \'Run\' to continue', true)
  else
    write_message (WindowHandle, Height-50, 10,
                   'Press \'Run\' to clear SVM', true)
  endif
endfor
```

At the end of the program, both classifiers are destroyed.

```
clear_class_svm (SVMHandle)
clear_class_svm (SVMHandleReduced)
```

12.3.2 Classification with Overlapping Classes

Example: `examples\hdevelop\Classification\Neural-Nets\class_overlap.dev`

This artificial example shows how an MLP classification behaves for classes with overlapping feature vectors. Three overlapping classes are defined by 2D feature vectors that consist of the rows and columns of region points. The classes are depicted in figure 12.3.

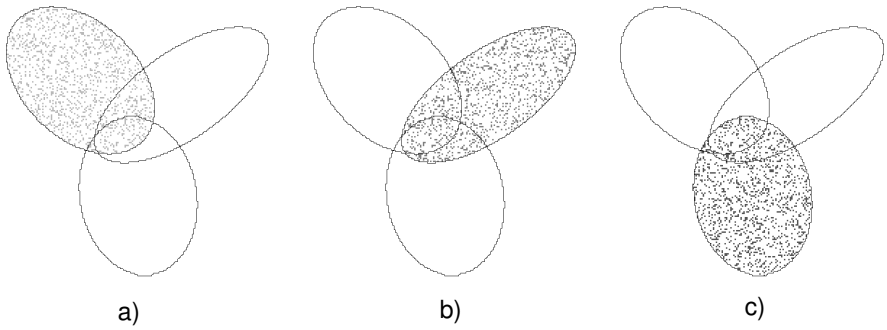


Figure 12.3: Three overlapping classes (feature vectors consist of rows and columns): (a) class1, (b) class 2, (c) class 3.

The classes are created as follows: For each class, an elliptic region is generated. Then, noise is added to a generic image, to which a threshold is applied to get three randomly distributed regions consisting of small dots. These regions are intersected with the elliptic regions of the individual classes. The result is an elliptic cluster of small dots for each class.

```
gen_ellipse (RegionClass1, 60, 60, rad(-45), 60, 40)
gen_ellipse (RegionClass2, 70, 130, rad(-145), 70, 30)
gen_ellipse (RegionClass3, 140, 100, rad(100), 55, 40)
gen_image_const (Image, 'byte', 200, 200)
add_noise_white (Image, ImageNoise1, 60)
add_noise_white (Image, ImageNoise2, 60)
add_noise_white (Image, ImageNoise3, 60)
threshold (ImageNoise1, RegionNoise1, 40, 255)
threshold (ImageNoise2, RegionNoise2, 40, 255)
threshold (ImageNoise3, RegionNoise3, 40, 255)
intersection (RegionClass1, RegionNoise1, SamplesClass1)
intersection (RegionClass2, RegionNoise2, SamplesClass2)
intersection (RegionClass3, RegionNoise3, SamplesClass3)
```

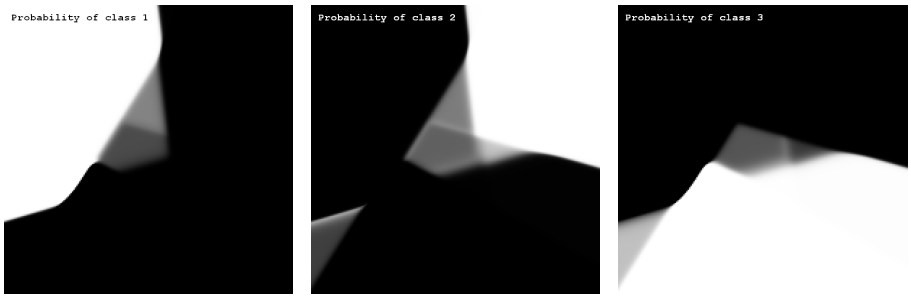
For each elliptic cluster, the positions of the region points are obtained by `get_region_points`. Each position builds a feature vector (`Rows[j]`, `Cols[j]`) that is added to the training by `add_sample_class_mlp`. Then, the classifier is trained with `train_class_mlp`.

```
concat_obj (SamplesClass1, SamplesClass2, Samples)
concat_obj (Samples, SamplesClass3, Samples)
create_class_mlp (2, 5, 3, 'softmax', 'normalization', 1, 42, MLPHandle)
for Class := 0 to 2 by 1
  select_obj (Samples, SamplesClass, Class+1)
  get_region_points (SamplesClass, Rows, Cols)
  for J := 0 to |Rows|-1 by 1
    add_sample_class_mlp (MLPHandle, real([Rows[J],Cols[J]]), Class)
  endfor
endfor
train_class_mlp (MLPHandle, 300, 0.01, 0.01, Error, ErrorLog)
```

After the training, the classifier is evaluated. To visualize the probabilities of the classes, we create a probability image for each class. In a loop, the operator `evaluate_class_mlp` returns the probabilities for each pixel position. The first value of the returned tuple shows the probability of the pixel to belong to class 1, the second value shows the probability of the pixel to belong to class 2, etc. The probability value of each class is then used as gray value for the corresponding probability image. Additionally, the result of the classification that is returned by `classify_class_mlp`, i.e., the most probable class for each position, is visualized in a probability image. The probability images showing the probabilities of the individual classes are depicted in figure 12.4. In figure 12.5a the probabilities of the three classes are united into a three-channel (color) image with `compose3`. Figure 12.5b shows the result of the classification, i.e., the most probable class for each position in the image.

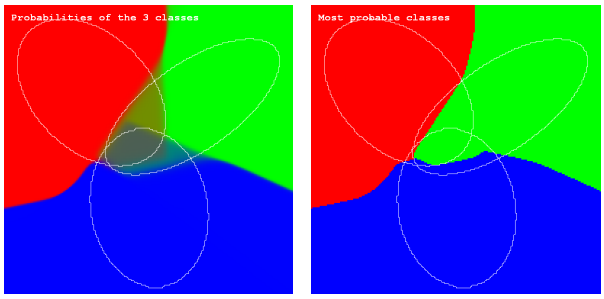
```
gen_image_const (ProbClass1, 'real', 200, 200)
gen_image_const (ProbClass2, 'real', 200, 200)
gen_image_const (ProbClass3, 'real', 200, 200)
gen_image_const (LabelClass, 'byte', 200, 200)
for R := 0 to 199 by 1
  for C := 0 to 199 by 1
    Features := real([R,C])
    evaluate_class_mlp (MLPHandle, Features, Prob)
    classify_class_mlp (MLPHandle, Features, 1, Class, Confidence)
    set_grayval (ProbClass1, R, C, Prob[0])
    set_grayval (ProbClass2, R, C, Prob[1])
    set_grayval (ProbClass3, R, C, Prob[2])
    set_grayval (LabelClass, R, C, Class)
  endfor
endfor
label_to_region (LabelClass, Classes)
compose3 (ProbClass1, ProbClass2, ProbClass3, Probs)
clear_class_mlp (MLPHandle)
```

Similar examples, but with GMM and SVM classifiers instead of an MLP classifier are `examples\hdevelop\Classification\Gaussian-Mixture-Models\class_overlap_gmm.dev` and `examples\hdevelop\Classification\Support-Vector-Machines\class_overlap_svm.dev`. For the GMM example, additionally the k-sigma probability is calculated for each feature vector. This can be used to reject all feature vectors that do not belong to one of the three classes. For the SVM example, the importance of selecting suitable parameter values is shown. For SVM classification, especially the parameter value `Nu`, which specifies the ratio of outliers in the training data set, has to be selected



a) b) c)

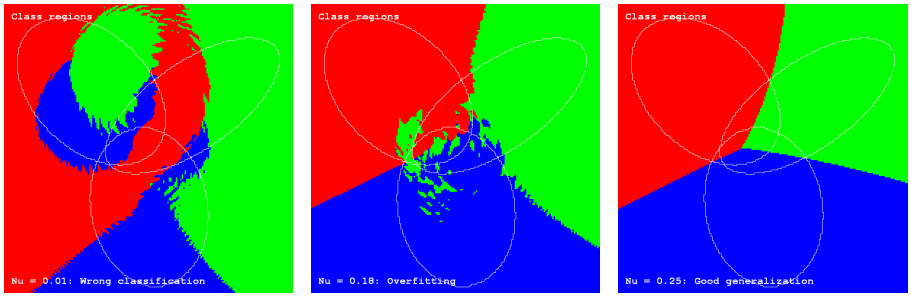
Figure 12.4: Probabilities for the overlapping classes ("white"=most probable): (a) probability of class 1 (b) probability of class 2, (c) probability of class 3.



a) b)

Figure 12.5: Probabilities for each pixel: (a) probabilities of the individual classes composed to a three-channel image, (b) confidences returned by the classification.

sufficiently (see figure 12.6).



a) b) c)

Figure 12.6: For SVM classification, the value for the parameter Nu has to be selected carefully: (a) Nu =0.01 (too small, wrong classification), (b) Nu = 0.18 (still too small, overfitting), (c) Nu = 25 (good classification).

12.3.3 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\gen_principal_comp_trans.dev
  Extract forest, water, and trees from a multi-channel image using principal component analysis
- examples\hdevelop\Applications\Monitoring\movement_col.dev
  Extract moving objects and scan them for specific color
- examples\hdevelop\Classification\Gaussian-Mixture-Models\class_overlap_gmm.dev
  Use a GMM for classifying two-dimensional data
- examples\hdevelop\Classification\Neural-Nets\regression.dev
  Approximate a function using an MLP
- examples\hdevelop\Classification\Support-Vector-Machines\class_overlap_svm.dev
  Use an SVM for classifying two-dimensional data
- examples\hdevelop\Classification\Support-Vector-Machines\get_support_vector_class_svm.dev
  Read out the support vectors from an SVM and visualize them along with the class boundaries
- examples\hdevelop\Classification\Support-Vector-Machines\novelty_detection.dev
  Show how to use an SVM classifier in the novelty-detection mode
- examples\hdevelop\Filter\Texture\texture_laws.dev
  Filter an image using a Laws texture filter
- examples\hdevelop\Filter\Texture\texture_laws_mlp.dev
  Segment an image based on texture using an MLP pixel classifier
- examples\hdevelop\OCR\Neural-Nets\letters_mlp.dev
  Train an MLP OCR classifier and reclassifiy the training samples
- examples\hdevelop\OCR\Support-Vector-Machines\compare_ocr_svm_mlp.dev
  Compare the recognition rates and the training and classification speeds of the SVM and MLP OCR classifiers
- examples\hdevelop\OCR\Support-Vector-Machines\letters_svm.dev
  Train an SVM OCR classifier and reclassify the training samples
- examples\hdevelop\Segmentation\Classification\class_2dim_sup.dev
  Segment a color image using two-dimensional pixel classification
- examples\hdevelop\Segmentation\Classification\class_2dim_unsup.dev
  Segment a color image by clustering
- examples\hdevelop\Segmentation\Classification\class_ndim_box.dev
  Classify pixels using hyper-cuboids

- examples\hdevelop\Segmentation\Classification\class_ndim_box.dev
  Classify pixels using hyper-cuboids
- examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev
  Classifying pixels using hyper-spheres
- examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev
  Classify pixels using hyper-spheres
- examples\hdevelop\Segmentation\Classification\classify_image_class_gmm.dev
  Segment an RGB image with a GMM classifier
- examples\hdevelop\Segmentation\Classification\classify_image_class_mlp.dev
  Segment an RGB image with an MLP classifier
- examples\hdevelop\Segmentation\Classification\classify_image_class_svm.dev
  Segment an RGB image with an SVM classifier
- examples\hdevelop\Segmentation\Classification\novelty_detection_gmm.dev
  Inspect a web using texture classification with gaussian mixture models
- examples\hdevelop\Segmentation\Topography\pouring.dev
  Segment an image by "pouring water" over it
- examples\solution_guide\basics\color_pieces.dev
  Check for completeness of colored game pieces using MLP classification
  → description here in the Solution Guide Basics on page 198
- examples\solution_guide\basics\color_pieces_euclid.dev
  Check for completeness of game color pieces using Euclidean classification
- examples\solution_guide\basics\simple_training.dev
  Train an OCR font from a training file
  → description here in the Solution Guide Basics on page 242

C++

- examples\cpp\source\bottle.cpp
  Reads numbers on a beer bottle
- examples\cpp\source\engraved.cpp
  Segmenting and reading characters on a metal surface

12.4 Selecting Operators

12.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

12.4.2 Create Classifier

Standard:

create_class_gmm, create_class_mlp, create_class_svm

12.4.3 Train Classifier

Standard:

add_sample_class_gmm, add_sample_class_mlp, add_sample_class_svm,
add_samples_image_class_gmm, add_samples_image_class_mlp,
add_samples_image_class_svm, train_class_gmm, train_class_mlp, train_class_svm

Advanced:

clear_samples_class_gmm, clear_samples_class_mlp, clear_samples_class_svm,
get_sample_num_class_gmm, get_sample_num_class_mlp, get_sample_num_class_svm,
get_sample_class_gmm, get_sample_class_mlp, get_sample_class_svm,
reduce_class_svm

12.4.4 Re-use Training Samples

Standard:

write_samples_class_gmm, write_samples_class_mlp, write_samples_class_svm,
read_samples_class_gmm, read_samples_class_mlp, read_samples_class_svm

12.4.5 Re-use Classifier

Standard:

write_class_gmm, write_class_mlp, write_class_svm, read_class_gmm, read_class_mlp,
read_class_svm

Advanced:

get_params_class_gmm, get_params_class_mlp, get_params_class_svm,
get_prep_info_class_gmm, get_prep_info_class_mlp, get_prep_info_class_svm

12.4.6 Evaluate Classifier

Standard:

evaluate_class_gmm, evaluate_class_mlp

12.4.7 Classify Data

Standard:

classify_class_gmm, classify_class_mlp, classify_class_svm,
classify_image_class_gmm, classify_image_class_mlp, classify_image_class_svm

12.4.8 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

12.4.9 Destroy Classifier

Standard:

clear_class_gmm, clear_class_mlp, clear_class_svm, clear_all_class_gmm,
clear_all_class_mlp, clear_all_class_svm

12.5 Relation to Other Methods

12.5.1 Methods that are Useful for Classification

Filtering
When applying a pixel-based classification, a preprocessing, in particular a filtering of the image may be convenient to minimize problems like noise, texture, or overlaid structures. Operators like mean_image or gauss_image can be used to eliminate noise. A fast but slightly less perfect alternative to gauss_image is binomial_filter. The operator median_image is helpful for suppressing small spots or thin lines and gray value morphology can be used to eliminate noise structures.

Region Of Interest (see description on page 19)
The concept of domains (the HALCON term for a region of interest) is useful for classification. Domains are used, e.g., to restrict the classification to the image part that has to be classified. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

12.5.2 Methods that are Using Classification

Color Processing (see description on page 191)
Classification is used for color processing. There, the feature vector for a pixel consists of the gray values connected to each channel of the image. The classification of the pixels due to their color can then be used, e.g., to segment an image into regions of different color classes. With the description of the color processing method you find the example examples\solution_guide\basics\color_pieces.dev which uses an MLP classification to check the completeness of colored game pieces.

OCR (see description on page 233)
Classification is used for optical character recognition (OCR). HALCON provides OCR-specific operators that classify previously segmented regions of images into character classes. The OCR-specific operators are available for the MLP, SVM, and box classifier. The latter is not recommended anymore as MLP and SVM are more powerful.

12.5.3 Alternatives to Classification

Blob Analysis (see description on page 33)
Classification is time consuming and rather complex to handle. Thus, it should be applied mainly if other methods fail. One method that may also solve a classification problem is blob analysis. For example, scratches can be separated from a surface by a classification, but in most cases a comparable result can be obtained also by a segmentation of the image via blob analysis, which needs significantly less time and is easy to apply.

Template Matching (see description on page 115)
If 2D shapes with fixed outlines are to be classified, in many cases it is more convenient to apply a template matching instead of a classification. A classification may be used if not only the shape but also the color or texture of the object is needed to distinguish the objects of different classes.

Variation Model (see description on page 159)
If differences between images and a reference image are searched for, the variation model may be used. In contrast to, e.g., template matching, images can be directly compared by their gray values and the comparison is spatially weighted by the variation image.

12.6 Tips & Tricks

12.6.1 OCR for General Classification

The OCR-specific classification operators are developed for optical character recognition but are not limited to characters. Actually, assuming a suitable training, you can classify any shape of a region with OCR. In some cases, it may be more convenient to classify the shapes of regions using the OCR-specific operators instead of the classification operators described here.

12.7 Advanced Topics

12.7.1 Selection of Training Samples

The quality of a classification depends not only on the selected parameters, but also on the set of training samples used to train the classifier. Thus, for a good classification result, you should carefully select your training samples:

- Use as many training samples as possible.

- Use approximately the same number of samples for each class.
- Take care that the samples of a class show some variations. If you do not have enough samples with variations, you can artificially create more of them by slightly changing your available samples.

Chapter 13

Color Processing

The idea of color processing is to take advantage of the additional information encoded in color or multi-spectral images. Processing color images can simplify many machine vision tasks and provide solutions to certain problems that are simply not possible in gray value images. In HALCON the following approaches in color processing can be distinguished: First, the individual channels of a color image can be processed using standard methods like blob analysis. In this approach the channels of the original image have to be decomposed first. An optional color space transformation is often helpful in order to access specific properties of a color image. Secondly, HALCON can process the color image as a whole by calling specialized operators, e.g., for pixel classification. Advanced applications of color processing include lines and edges extraction.

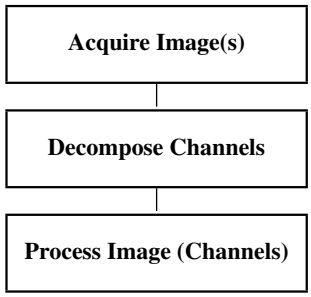
a) b)

Figure 13.1: Simple color segmentation.

The example illustrated in figure 13.1 shows how to segment blue pieces of plasticine in a color image.

13.1 Basic Concept

Simple color processing, which is using the methods of blob analysis, mainly consists of three parts:



13.1.1 Acquire Image(s)

First, an image is acquired.

For detailed information see the description of this method on page 13.

13.1.2 Decompose Channels

In order to be able to process the individual channels, RGB color images have to be split up into a red, green, and blue channel by using the operator `decompose3`.

13.1.3 Process Image (Channels)

Depending on the application, the individual channels can be processed using standard methods described in this chapter. One of the most frequently used methods is blob analysis (see Blob Analysis on page 33).

13.1.4 A First Example

An example for this basic concept is the following program, which belongs to the example explained above.

Here, an RGB image is acquired from file. The image is split into its channels using `decompose3`. The red and green channels are subtracted from the blue channel using `sub_image`. The purpose of this process is to fade out pixels with high values in the other channels, leaving pure blue pixels only. Using `threshold`, the blue pixels with a certain intensity are selected.

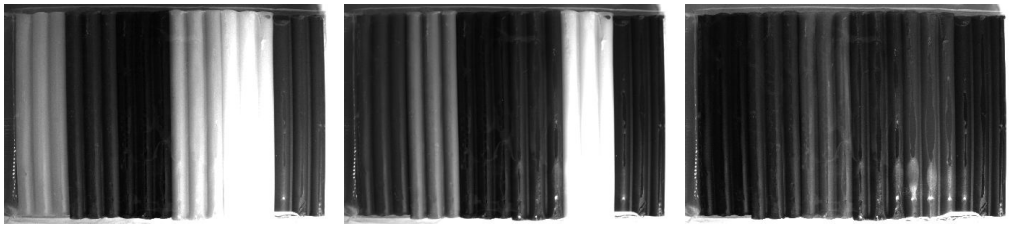


Figure 13.2: Color image decomposed to its red, green, and blue channels.

```
read_image (Image, 'plasticine')
decompose3 (Image, Red, Green, Blue)
sub_image (Blue, Red, RedRemoved, 1, 0)
sub_image (RedRemoved, Green, RedGreenRemoved, 1, 0)
threshold (RedGreenRemoved, BluePixels, 10, 255)
```

13.2 Extended Concept

In many cases the processing of color images will be more advanced than in the above example. Depending on the actual application, the order of the following steps may vary, or some steps may be omitted altogether.

13.2.1 Demosaick Bayer Pattern

If the acquired image is a Bayer image, it can be converted to RGB using the operator `cfa_to_rgb`. The encoding type of the Bayer pattern (the color of the first two pixels of the first row) must be known (see figure 13.3).

13.2.2 Transform Color Space

The RGB color space is not always the most appropriate starting point to process color images. If this is the case, a transformation to a different color space might be useful. HALCON supports many important color spaces. Namely, the HSV and HSI color spaces are favorable to select distinct colors independent of their intensity. Therefore, color segmentations in these color spaces are very robust under varying illumination. The i1i2i3 color space qualifies for color classification, whereas the cielab color space is a close match to human perception.

13.2.3 Train Colors

In order to do color classification the colors that need to be distinguished have to be trained. There are different approaches for full color classification including Euclidean, box, gaussian mixture models

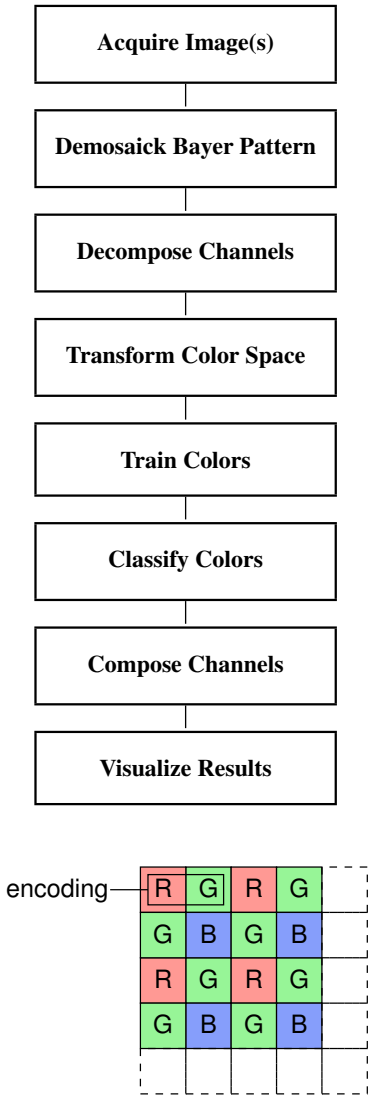


Figure 13.3: Sample Bayer pattern and corresponding encoding.

(GMM), multilayer perceptron (MLP), and support vector machine (SVM) classification. For further information about classification, see Classification on page 171.

13.2.4 Classify Colors

The colors trained in the previous step are used in subsequent images to do the actual classification.

13.2.5 Compose Channels

Any number of channels can be joined to a multi-channel image using the operators `compose2` through `compose7`, or `append_channel`. This way, channels that were processed separately can be composed back to color images for visualization purposes.

13.2.6 Visualize Results

Finally, you might want to display the images, the regions, and the features.

For detailed information see the description of this method on page 269.

13.3 Programming Examples

This section gives a brief introduction to using HALCON for color processing.

13.3.1 Robust Color Extraction

Example: `examples\solution_guide\basics\color_simple.dev`

The object of this example is to segment the yellow cable in a color image in a robust manner.

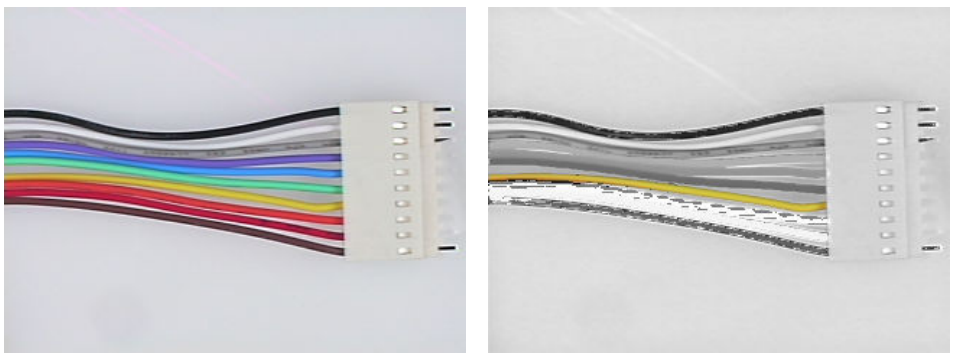


Figure 13.4: Segmentation of a specific color.

Here, an RGB image is acquired from file. The image is split into its channels using `decompose3`. Afterwards, a color space transformation from RGB to HSV is performed using `trans_from_rgb`. This transformation converts the image channels into the separate components hue, saturation and intensity. In the next steps the operator `threshold` selects all pixels with a high saturation value, followed by `reduce_domain` in the hue channel which effectively filters out pale colors and grays. A histogram of the remaining saturated (vivid) colors is displayed in figure 13.5. Each peak in this histogram corresponds to a distinct color. The corresponding color band is shown below the histogram. Finally, the last `threshold` selects the yellowish pixels.

```
read_image (Image, 'cable' + i)
decompose3 (Image, Red, Green, Blue)
trans_from_rgb (Red, Green, Blue, Hue, Saturation, Intensity, 'hsv')
threshold (Saturation, HighSaturation, 100, 255)
reduce_domain (Hue, HighSaturation, HueHighSaturation)
threshold (HueHighSaturation, Yellow, 20, 50)
```

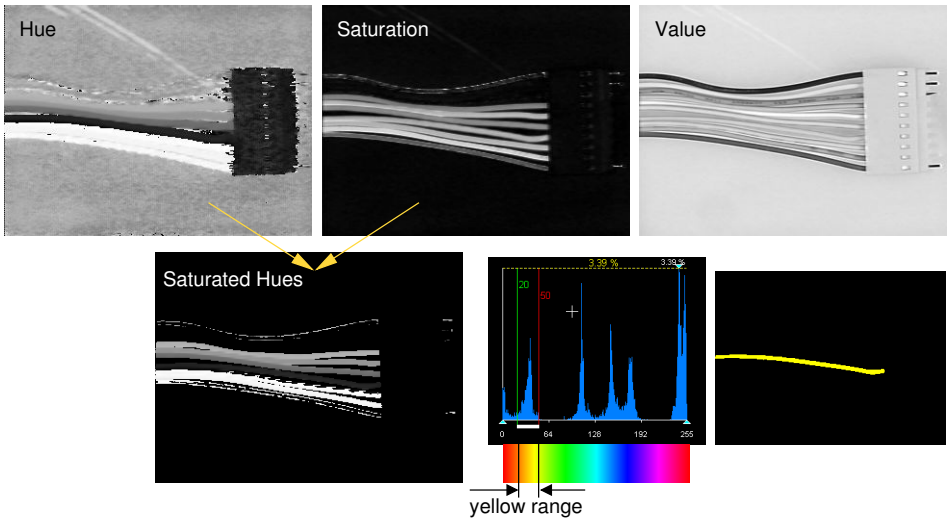


Figure 13.5: Segmentation in HSV color space.

Finding the proper threshold margins is crucial in applications like this. In HDevelop the Gray Histogram tool can be used to determine the values interactively. See the HDevelop User's Guide for more information. To generate the color band shown in figure 13.5, use the following code snippet:

gen_image_gray_ramp (Hue, 0, 1, 128, 32, 128, 256, 64) gen_image_proto (Hue, White, 255) trans_to_rgb (Hue, White, White, Red, Green, Blue, 'hsv') compose3 (Red, Green, Blue, MultiChannelImage)

13.3.2 Sorting Fuses

Example: examples\solution_guide\basics\color_fuses.dev

In this example different types of fuses are classified using color images. The applied method is similar to the previous example. A training image has been used to specify ranges of hue for the fuse types that need to be distinguished. The determined ranges are hard-coded in the program.

```
FuseColors := ['Orange','Red','Blue','Yellow','Green']
FuseTypes := [5,10,15,20,30]
* HueRanges: Orange 10-30, Red 0-10...
HueRanges := [10,30,0,10,125,162,30,64,96,128]
```

Figure 13.6: Simple classification of fuses by color with varying illumination.

A sequence of images is acquired from file, converted to the HSV color space, and reduced to contain only saturated colors just like in the previous example. As already mentioned, color selection in this color space is pretty stable under changing illumination. That is why the hard-coded color ranges are sufficient for a reliable classification. However, it has to be kept in mind that a certain degree of color saturation must be guaranteed for the illustrated method to work.

```
decompose3 (Image, Red, Green, Blue)
trans_from_rgb (Red, Green, Blue, Hue, Saturation, Intensity, 'hsv')
threshold (Saturation, Saturated, 60, 255)
reduce_domain (Hue, Saturated, HueSaturated)
```

The classification iterates over the fuse types and checks for sufficiently large areas in the given hue range. This is done using blob analysis. Afterwards, an additional inner loop labels the detected fuses.

```
for Fuse := 0 to |FuseTypes|-1 by 1
  threshold (HueSaturated, CurrentFuse, HueRanges[Fuse*2],
             HueRanges[Fuse*2+1])
  connection (CurrentFuse, CurrentFuseConn)
  fill_up (CurrentFuseConn, CurrentFuseFill)
  select_shape (CurrentFuseFill, CurrentFuseSel, 'area', 'and', 6000,
                20000)
  area_center (CurrentFuseSel, FuseArea, Row1, Column1)
  dev_set_color ('magenta')
  for i := 0 to |FuseArea|-1 by 1
    set_tposition (WH, Row1[i], Column1[i])
    write_string (WH,
                  FuseColors[Fuse] + ' ' +FuseTypes[Fuse]+' Ampere')
  endfor
  set_tposition (WH, 24*(Fuse+1), 12)
  dev_set_color ('slate blue')
  write_string (WH, FuseColors[Fuse] + ' Fuses: ' + |FuseArea|)
endfor
stop ()
```

13.3.3 Completeness Check of Colored Game Pieces

Example: examples\solution_guide\basics\color_pieces.dev

Completeness checks are very common in machine vision. Usually, packages assembled on a production line have to be inspected for missing items. Before this inspection can be done, the items have to be trained. In the example presented here, a package of game pieces has to be inspected. The game pieces come in three different colors, and the package should contain four of each type. The pieces themselves can be of slightly different shape, so shape-based matching is not an option. The solution to this problem is to classify the game pieces by color. The method applied here is a classification using neural nets (MLP classification).

In the training phase an image is acquired, which contains the different types of game pieces. The task is to specify sample regions for the game pieces and the background using the mouse (see Figure 13.7a). This is accomplished by looping over the draw_rectangle1 and gen_rectangle1 operators to draw and create the corresponding regions. The tuple Classes, which will be used for the actual training, is extended each time.

```
read_image (Image, ImageRootName+'0')
for i := 1 to 4 by 1
  dev_display (Image)
  dev_display (Classes)
  set_tposition (WindowHandle, 24, 12)
  write_string (WindowHandle,
                'Drag rectangle inside ' + Regions[i-1] + ' color. Click right mouse button to co
  draw_rectangle1 (WindowHandle, Row1, Column1, Row2, Column2)
  gen_rectangle1 (Rectangle, Row1, Column1, Row2, Column2)
  concat_obj (Classes, Rectangle, Classes)
endfor
```

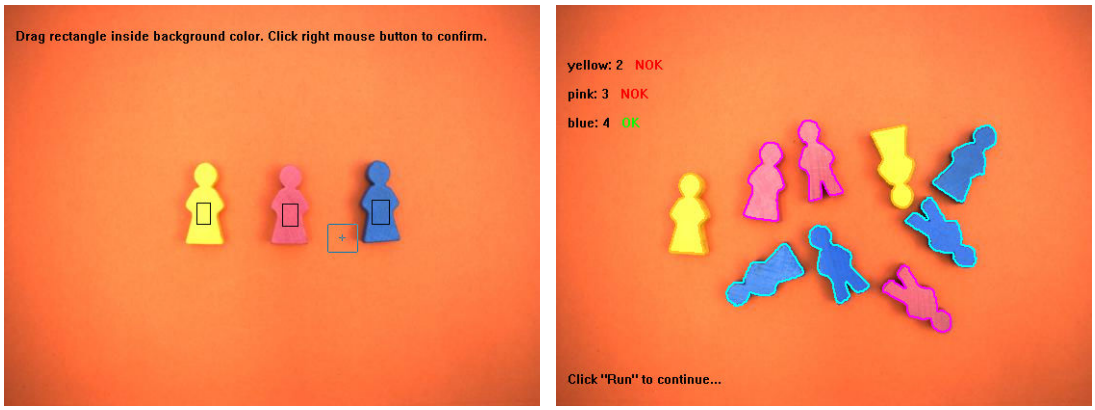


Figure 13.7: Example MLP classification: (a) Training, (b) Result.

Once the classes are specified, a multilayer perceptron is created using `create_class_mlp`. With the operator `add_samples_image_class_mlp` the training samples from the image are added to the training data of the multilayer perceptron. The actual training is started with `train_class_mlp`. The duration of the training depends on the complexity and sizes of the training regions.

```
create_class_mlp (3, 4, 4, 'softmax', 'normalization', 3, 42,
                  MLPHandle)
add_samples_image_class_mlp (Image, Classes, MLPHandle)
set_tposition (WindowHandle, 100, 12)
write_string (WindowHandle, 'Training...')
train_class_mlp (MLPHandle, 200, 1, 0.01, Error, ErrorLog)
```

After the training has finished, subsequent images are acquired and classified using `classify_image_class_mlp`. The operator returns a classified region.

```
for img := 0 to 3 by 1
  read_image (Image, ImageRootName + img)
  classify_image_class_mlp (Image, ClassRegions, MLPHandle, 0.5)
stop ()
endfor
```

The returned result is processed further using blob analysis. Each class of the classified region (with the exception of the background class) is accessed using `copy_obj`. The regions of each class are split up using `connection` and reduced to regions of a relevant size (`select_shape`). The remaining few lines of code calculate the number of game pieces found for each class and make a decision whether the result was OK or not.

```
for figure := 1 to 3 by 1
  copy_obj (ClassRegions, ObjectsSelected, figure, 1)
  connection (ObjectsSelected, ConnectedRegions)
  select_shape (ConnectedRegions, SelectedRegions, 'area', 'and',
                400, 99999)
  count_obj (SelectedRegions, Number)
  dev_set_color (Highlight[figure-1])
  dev_display (SelectedRegions)
  dev_set_color ('black')
  set_tposition (WindowHandle, 24+30*figure, 12)
  write_string (WindowHandle, Regions[figure-1] + ': ' +Number)
  write_string (WindowHandle, '   ')
  dev_set_color ('green')
  if (Number#4)
    dev_set_color ('red')
    write_string (WindowHandle, 'N')
  endif
  write_string (WindowHandle, 'OK')
  dev_set_color ('black')
endfor
```

To illustrate the advantage of using color information, and to compare the classification results, the example program runs an additional training and classification on a converted gray image. As can be seen in figure 13.8 only the yellow region is detected faithfully.

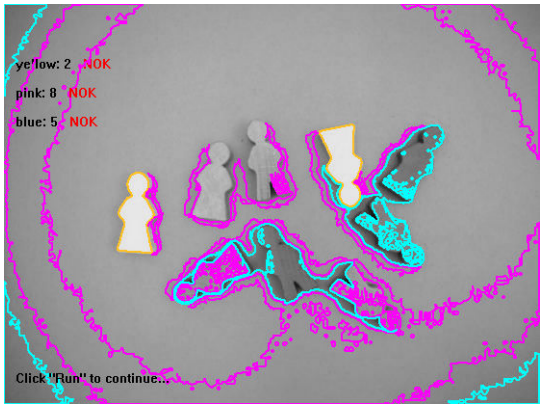


Figure 13.8: Poor classification result when only using the gray scale image.

```
rgb1_to_gray (Image, GrayImage)
compose3 (GrayImage, GrayImage, GrayImage, Image)
```

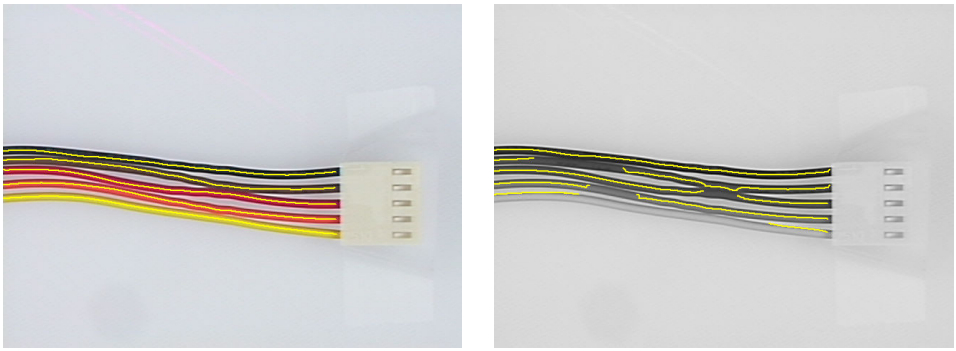
For more complex (and time-consuming) classifications, it is recommended to save the training data to the file system using `write_class_mlp`. Later, the saved data can be restored using `read_class_mlp`. As an alternative to the MLP classification you can also apply a classification based on support vector machines (SVM). The corresponding op-

erators are create_class_svm, add_samples_image_class_svm, train_class_svm, classify_image_class_svm, write_class_svm, and read_class_svm. In order to speed up the classification itself, you can use a different classification method like Euclidean classification (see examples\solution_guide\basics\color_pieces_euclid.dev).

13.3.4 Inspect Power Supply Cables

Example: examples\hdevelop\Filter\Lines\lines_color.dev

The task of this example is to locate and inspect the power supply cables depicted in figure 13.9.



a) b)

Figure 13.9: (a) Original color image with cable centers extracted using the color line extractor; (b) corresponding results when using the gray value image.

The input for the program are sample images of colored power supply cables. The task is to extract the centers of each cable together with the width. This is performed using the subpixel-precise color line extractor. To remove irrelevant structures, contours that are too short are removed.

```
lines_color (Image, Lines, 3.5, 0, 12, 'true', 'false')
select_contours_xld (Lines, LongLines, 'contour_length', 450, 100000,
                     0, 0)
```

The cable width is determined by accessing the line width attribute. For display purposes, a contour is generated for each side.

```
count_obj (LongLines, Number)
gen_empty_obj (EdgesL)
gen_empty_obj (EdgesR)
for K := 1 to Number by 1
  select_obj (LongLines, Line, K)
  get_contour_xld (Line, Row, Col)
  get_contour_attrib_xld (Line, 'angle', Angle)
  get_contour_attrib_xld (Line, 'width_right', WidthR)
  get_contour_attrib_xld (Line, 'width_left', WidthL)
  EdgeRR := Row+cos(Angle)*WidthR
  EdgeRC := Col+sin(Angle)*WidthR
  EdgeLR := Row-cos(Angle)*WidthL
  EdgeLC := Col-sin(Angle)*WidthL
  gen_contour_polygon_xld (EdgeR, EdgeRR, EdgeRC)
  gen_contour_polygon_xld (EdgeL, EdgeLR, EdgeLC)
  concat_obj (EdgesL, EdgeL, EdgesL)
  concat_obj (EdgesR, EdgeR, EdgesR)
endfor
```

To compare this result with the classical line extraction approach (see Edge Extraction (Subpixel-Precise) on page 85), a line extractor is also applied to the gray value image. The result is depicted in figure 13.9b. Here, it becomes obvious how hard it is to extract the cable using the luminance only.

```
rgb1_to_gray (Image, GrayImage)
lines_gauss (GrayImage, LinesGray, 3.5, 0, 0.7, 'dark', 'true', 'true',
```

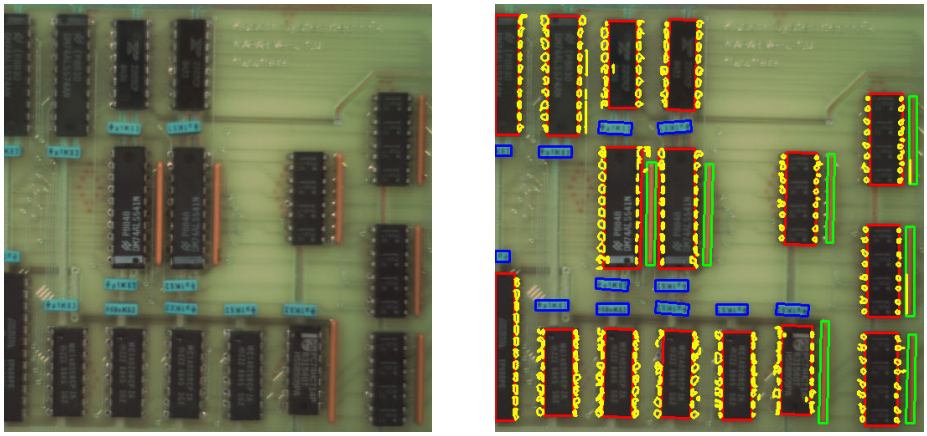
13.3.5 Locating Board Components by Color

Example: `examples\hdevelop\Applications\FA\ic.dev`

The task of this example is to locate all components on the printed circuit board depicted in figure 13.10.

The input data is a color image, which allows locating components like capacitors and resistors very easily by their significant color: Using a color space transformation, the hue values allow the selection of the corresponding components. The following code extracts the resistors; the extraction of the capacitors is performed along the same lines.

```
decompose3 (Image, Red, Green, Blue)
trans_from_rgb (Red, Green, Blue, Hue, Saturation, Intensity, 'hsv')
threshold (Saturation, Colored, 100, 255)
reduce_domain (Hue, Colored, HueColored)
threshold (HueColored, Red, 10, 19)
connection (Red, RedConnect)
select_shape (RedConnect, RedLarge, 'area', 'and', 150.000000,
              99999.000000)
shape_trans (RedLarge, Resistors, 'rectangle2')
```



a) b)

Figure 13.10: (a) Original image; (b) extracted ICs, resistors, and capacitors.

The extraction of the ICs is more difficult because of the bright imprints, which do not allow a simple thresholding in one step. Instead of this, dark areas are selected first, which are then combined using a dilation.

```
threshold (Intensity, Dark, 0, 50)
dilation_rectangle1 (Dark, DarkDilation, 14, 14)
connection (DarkDilation, ICLarge)
```

After this, the segmentation is repeated inside the thus extracted connected components.

```
add_channels (ICLarge, Intensity, ICLargeGray)
threshold (ICLargeGray, ICDark, 0, 50)
shape_trans (ICDark, IC, 'rectangle2')
```

To locate the contact points, small ROIs are generated on the left and right side of each IC.

```
dilation_rectangle1 (IC, ICDilation, 5, 1)
difference (ICDilation, IC, SearchSpace)
dilation_rectangle1 (SearchSpace, SearchSpaceDilation, 14, 1)
union1 (SearchSpaceDilation, SearchSpaceUnion)
```

Inside these areas, locally bright spots are detected.

```
reduce_domain (Intensity, SearchSpaceUnion, SearchGray)
mean_image (SearchGray, SearchMean, 15, 15)
dyn_threshold (SearchGray, SearchMean, PinsRaw, 5.000000, 'light')
connection (PinsRaw, PinsConnect)
fill_up (PinsConnect, PinsFilled)
select_shape (PinsFilled, Pins, 'area', 'and', 10, 100)
```

13.3.6 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\forest.dev
  Extract trees and meadows from forest
  → description here in the Solution Guide Basics on page 40
- examples\hdevelop\Applications\Color\color_segmentation_pizza.dev
  Find salami pieces on pizza based on color image processing
- examples\hdevelop\Applications\FA\matching_multi_channel_clamp.dev
  Locate upper clamp from a pile using color shape-based matching
- examples\hdevelop\Applications\FA\matching_multi_channel_yogurt.dev
  Find yogurts of different flavors using color shape-based matching
- examples\hdevelop\Applications\Monitoring\movement_col.dev
  Extract moving objects and scan them for specific color
- examples\hdevelop\Applications\OCR\ocrcolor.dev
  Read numbers using color information
  → description here in the Solution Guide Basics on page 245
- examples\hdevelop\Applications\OCR\ocrcolort.dev
  Segment numbers using color information and train the OCR
- examples\hdevelop\Filter\Color\cfa_to_rgb.dev
  Convert a Bayer image (color filter array) into an RGB image
- examples\hdevelop\Filter\Edges\edges_color.dev
  Extract edges using color information
  → description here in the Solution Guide Basics on page 78
- examples\hdevelop\Filter\Edges\edges_color_sub_pix.dev
  Extract edges with sub-pixel precision using color information
- examples\hdevelop\Filter\Misc\symmetry.dev
  Analyze symmetry in horizontal direction
- examples\hdevelop\Segmentation\Classification\class_2dim_sup.dev
  Segment a color image using two-dimensional pixel classification
- examples\hdevelop\Segmentation\Classification\class_2dim_unsup.dev
  Segment a color image by clustering
- examples\hdevelop\Segmentation\Classification\class_ndim_box.dev
  Classify pixels using hyper-cuboids
- examples\hdevelop\Segmentation\Classification\class_ndim_box.dev
  Classify pixels using hyper-cuboids

- examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev
  Classifying pixels using hyper-spheres
- examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev
  Classify pixels using hyper-spheres
- examples\hdevelop\Segmentation\Classification\classify_image_class_gmm.dev
  Segment an RGB image with a GMM classifier
- examples\hdevelop\Segmentation\Classification\classify_image_class_mlp.dev
  Segment an RGB image with an MLP classifier
- examples\hdevelop\Segmentation\Classification\classify_image_class_svm.dev
  Segment an RGB image with an SVM classifier
- examples\hdevelop\Segmentation\Regiongrowing\regiongrowing_n.dev
  Apply a regiongrowing for multi-channel images
- examples\hdevelop\Segmentation\Topography\pouring.dev
  Segment an image by "pouring water" over it
- examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev
  Rectify image of stadium to simulate overhead view
- examples\solution_guide\basics\color_pieces_euclid.dev
  Check for completeness of game color pieces using Euclidean classification

C

- examples\c\source\example_multithreaded1.c
  Using multiple threads with Parallel HALCON

13.4 Selecting Operators

13.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

13.4.2 Demosaick Bayer Pattern

Standard:

cfa_to_rgb

13.4.3 Decompose Channels

Standard:

decompose3, access_channel

13.4.4 Transform Color Space

Standard:

trans_from_rgb, trans_to_rgb

The following color spaces are supported:

- argyb
- cielab
- ciexyz
- hls
- hsi
- hsv
- i1i2i3
- ihs
- rgb
- yiq
- yuv

13.4.5 Train Colors

Standard:

histo_2dim, learn_ndim_norm, learn_ndim_box

Advanced:

train_class_mlp, train_class_svm

13.4.6 Process Image (Channels)

Standard:

smooth_image, mean_image, median_image

Advanced:

lines_color, edges_color, edges_color_sub_pix

Further operators can be found in the operator list for the method Blob Analysis (see section 4.4 on page 51).

13.4.7 Classify Colors

Standard:

class_2dim_sup, class_2dim_unsup, class_ndim_norm, class_ndim_box

Advanced:

classify_image_class_mlp, classify_image_class_svm

13.4.8 Compose Channels

Standard:

compose3, append_channel

13.4.9 Visualize Results

Standard:

disp_color, disp_obj

Further operators can be found in the operator list for the method Visualization (see section 18.4 on page 283).

13.5 Tips & Tricks

13.5.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is also useful for color processing. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the segmentation is performed can be restricted, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

13.5.2 Speed Up

Many online applications require maximum speed. Because of its flexibility, HALCON offers many ways to achieve this goal. Here, the most common methods are listed.

- Of the color processing approaches discussed in this section, the simplest is also the fastest (decompose color image and apply blob analysis). If you want to do color classification, you should consider using `class_ndim_norm` or `class_2dim_sup` for maximum performance.
- Regions of interest are the standard method to increase the speed by processing only those areas where objects need to be inspected. This can be done using pre-defined regions but also by an online generation of the regions of interest that depend on other objects found in the image.
- By default, HALCON performs some data consistency checks. These can be switched off using `set_check`.
- By default, HALCON initializes new images. Using `set_system` with the parameter `"init_new_image"`, this behavior can be changed.

13.6 Advanced Topics

13.6.1 Color Edge Extraction

Similar to the operator `edges_image` for gray value images (see Edge Extraction (Pixel-Precise) on page 73), the operator `edges_color` can be applied to find pixel-precise edges in color images. For a subpixel-precise edge extraction in color images, the operator `edges_color_sub_pix` is provided. It corresponds to the gray value based operator `edges_sub_pix` (see Edge Extraction (Subpixel-Precise) on page 85). Processing the detected color edges is the same as for gray value images (see Contour Processing on page 97).

13.6.2 Color Lines Extraction

Similar to color edge extraction, HALCON supports the detection of lines in color images using `lines_color`. For processing the detected lines, please refer to Edge Extraction (Subpixel-Precise) on page 85 and Contour Processing on page 97.

Chapter 14

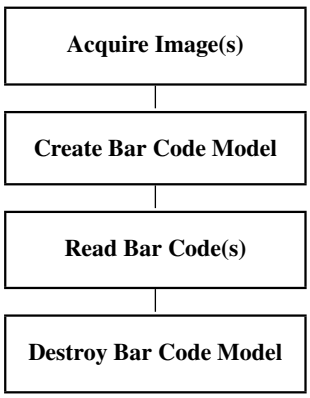
1D Bar Code

The idea of bar code reading is quite easy. You initialize a bar code model and then execute the operator for reading bar codes. Within this operator, you specify the desired bar code type. That way, you can read different bar code types by successively applying the operator with different parameters for the type, but without the need to create a separate model for each bar code type. The result of the reading is a region that contains the bar code and the decoded string.

The advantage of the HALCON bar code reader is its ease of use. No advanced experience in programming or image processing is required. Only a few operators in a clear and simple order are applied. Furthermore, the bar code reader is very powerful and flexible. An example for this is its ability to read an arbitrary number of bar codes of the same code type in any orientation even if parts are missing.

14.1 Basic Concept

Bar code reading consists mainly of four steps:



14.1.1 Acquire Image(s)

For the online part, i.e., during reading only, images must be acquired.

For detailed information see the description of this method on page 13.

14.1.2 Create Bar Code Model

You create a bar code model with the operator `create_bar_code_model`. The returned handle provides all necessary information about the structure of the bar code. In most cases, you do not have to adjust any parameters.

14.1.3 Read Bar Code(s)

Bar codes are read using the operator `find_bar_code`. Within this operator, you specify the model and the bar code type that you search for. The operator returns the regions and the decoded strings of all bar codes of the specified type that are found in the image or in the specified region of interest.

14.1.4 Destroy Bar Code Model

When you no longer need the bar code model, you destroy it with the operator `clear_bar_code_model`.

14.1.5 A First Example

As an example for this basic concept, here is a very simple program, that reads the EAN 13 bar code depicted in figure 14.1.

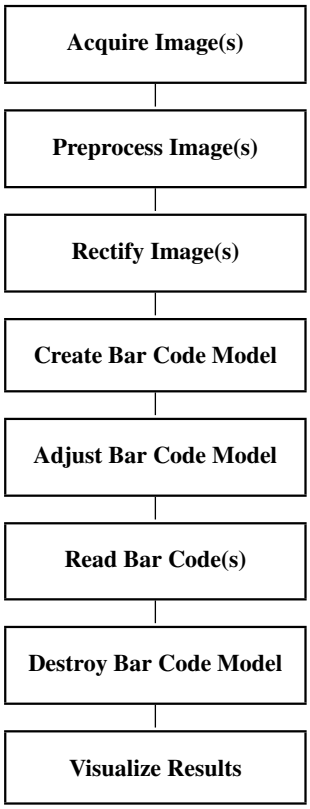
A test image is acquired from file and the bar code model is created with `create_bar_code_model`. Then, the operator `find_bar_code` (with `CodeType` set to `'EAN-13'`) returns the region and the decoded string for the found bar code. At the end of the program, the bar code model is destroyed using `clear_bar_code_model`.

```
read_image (image, 'barcode/ean13/ean1301')
create_bar_code_model ([], [], BarCodeHandle)
find_bar_code (image, SymbolRegions, BarCodeHandle, 'EAN-13',
               DecodedDataStrings)
clear_bar_code_model (BarCodeHandle)
```

14.2 Extended Concept

In some cases, bar code reading can be more advanced than in the example above. For example, a preprocessing or rectification of the image as well as the visualization of results can be necessary.

Figure 14.1: Reading a bar code.



14.2.1 Preprocess Image(s)

HALCON expects bar codes to be printed dark on a light background. To read light bar codes on a dark background you must first invert the image using the operator `invert_image`.

14.2.2 Rectify Image(s)

The bar code reader can handle perspective distortions up to a certain extent. For very strong distortions, it might be necessary to rectify the image before applying the bar code reader.

How to perform the rectification for a bar code that is printed radially on a CD is shown in the description of the example program `circular_barcode.dev`. Detailed information about compensating distortions caused, e.g., by non-perpendicularly mounted cameras, can be found in the Solution Guide II-F in section 3.3 on page 49.

14.2.3 Create Bar Code Model

Sometimes, a better result can be obtained when adjusting parameters like the range for the element size (`'element_size_min'`, `'element_size_max'`). All available parameters can be changed either here, or in a later step using `set_bar_code_param`.

14.2.4 Adjust Bar Code Model

Although in most cases no adaptation of parameters is necessary for reading bar codes with HALCON, a few parameters can be adjusted with the operator `set_bar_code_param` before applying `find_bar_code`. In particular, you can adjust the range of the width of the bar code elements (`'element_size_min'`, `'element_size_max'`). For code types having a check character, you can determine if the check character is used to check the result or if it is returned as part of the decoded string (`'check_char'`). Finally, you can adjust the threshold that is used internally for the extraction of the element edges (`'meas_thresh'`).

To query the values that are currently used for the individual parameters, you can apply the operator `get_bar_code_param`.

14.2.5 Read Bar Code(s)

After reading the bar codes with `find_bar_code`, you can explicitly query the regions of the decoded bar codes or their candidates with the operator `get_bar_code_object`. The decoded strings or the underlying reference data (including, e.g., the start/stop characters) can be queried with the operator `get_bar_code_result`.

14.2.6 Visualize Results

Finally, you might want to display the images, the bar code regions, and the decoded content.

For detailed information see the description of this method on page 269.

14.3 Programming Examples

This section shows how to use the bar code reader.

14.3.1 Reading a Bar Code on a CD

Example: examples\hdevelop\Applications\Barcode\circular_barcode.dev

Figure 14.2 shows an image of a CD, on which a bar code is printed radially. The task is to read this circular bar code. Because the bar code reader cannot read this kind of print directly, first the image must be transformed such that the elements of the bar code are parallel.

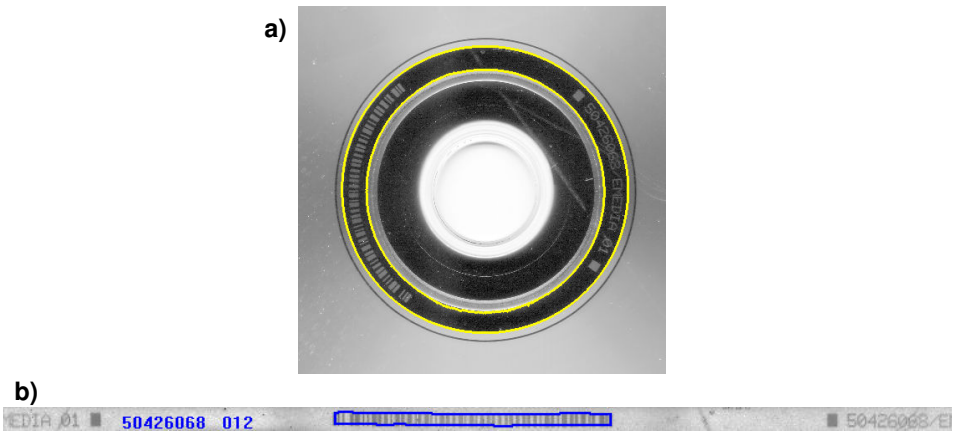


Figure 14.2: (a) Original image with segmented ring; (b) rectified ring with decoded bar code.

The first step is to segment the dark ring on which the bar code is printed. This is performed using bin_threshold followed by connection. This segmentation returns all dark areas in the image, including the dark ring. To select the ring, select_shape is used with corresponding values for the extent.

```
bin_threshold (Image, Region)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, Ring, ['width','height'], 'and', [550,550],
              [750,750])
```

After that, the parameters of the outer and the inner circle are determined. The outer circle can be determined directly using shape_trans. The inner circle is more complicated to extract. Here, it is calculated by creating the complement region of the ring with appropriate dimensions and then selecting its inner part. With smallest_circle, the parameters of the inner and outer circle are determined.

```
shape_trans (Ring, OuterCircle, 'outer_circle')
complement (Ring, RegionComplement)
connection (RegionComplement, ConnectedRegions)
select_shape (ConnectedRegions, InnerCircle, ['width','height'], 'and',
              [450,450], [650,650])
smallest_circle (Ring, Row, Column, OuterRadius)
smallest_circle (InnerCircle, InnerRow, InnerColumn, InnerRadius)
```

The parameters for inner and outer circle are the input for the polar transform (polar_trans_image_ext), which transforms the image inside the ring into a rectangular area. Then, the image is inverted to obtain dark bar code elements on a light background (see figure 14.2b).

```
WidthPolar := 1440
HeightPolar := round(OuterRadius-InnerRadius-10)
polar_trans_image_ext (Image, PolarTransImage, Row, Column, rad(360), 0,
                       OuterRadius-5, InnerRadius+5, WidthPolar,
                       HeightPolar, 'bilinear')
invert_image (PolarTransImage, ImageInvert)
```

To read the bar code, first a model is created with create_bar_code_model. As the bar code elements are very thin, parameters are adjusted with the operator set_bar_code_param. In particular, the parameter 'element_size_min' is set to 1.5 and the parameter 'meas_thresh' is set to 0.1. Then, find_bar_code finds and decodes the bar code, which is of the specified code type 'Code 128'. The bar code model is destroyed with clear_bar_code_model.

```
create_bar_code_model ([], [], BarCodeHandle)
set_bar_code_param (BarCodeHandle, 'element_size_min', 1.5)
set_bar_code_param (BarCodeHandle, 'meas_thresh', 0.1)
find_bar_code (ImageZoomed, SymbolRegions, BarCodeHandle, 'Code 128',
               DecodedDataStrings)
clear_bar_code_model (BarCodeHandle)
```

Finally, the region of the bar code that was returned by find_bar_code is transformed back to the original shape of the bar code by polar_trans_region_inv and is displayed in the image.

```
polar_trans_region_inv (SymbolRegions, CodeRegionCircular, Row, Column,
                        rad(360), 0, OuterRadius-5, InnerRadius+5,
                        WidthPolar, HeightPolar, Width, Height,
                        'nearest_neighbor')
dev_display (CodeRegionCircular)
```

14.3.2 Other Examples

HDevelop

- examples\hdevelop\Applications\Barcode\25Industry.dev
  Read a bar code of type 2/5 industrial

- examples\hdevelop\Applications\Barcode\25Interleaved.dev
  Read a bar code of type 2/5 interleaved
- examples\hdevelop\Applications\Barcode\barcode_orientation.dev
  Use the bar code parameter 'orientation'
- examples\hdevelop\Applications\Barcode\barcode_param_element_height_min.dev
  Use the bar code parameter 'element_height_min'
- examples\hdevelop\Applications\Barcode\barcode_param_max_diff_orient.dev
  Use the bar code parameter 'max_diff_orient'
- examples\hdevelop\Applications\Barcode\barcode_param_orientation.dev
  Use the bar code parameters 'orientation' and 'orientation_tol'
- examples\hdevelop\Applications\Barcode\Codabar.dev
  Read a bar code of type Codabar
- examples\hdevelop\Applications\Barcode\Code128.dev
  Read a bar code of type Code 128
- examples\hdevelop\Applications\Barcode\Code39.dev
  Read a bar code of type Code 39
- examples\hdevelop\Applications\Barcode\defect_barcode.dev
  Show the bar code reader's ability to read defect bar codes
- examples\hdevelop\Applications\Barcode\EAN13.dev
  Read a bar code of type EAN13
- examples\hdevelop\Applications\Barcode\EAN13AddOn5.dev
  Read a bar code of type EAN13 Add-On 5
- examples\hdevelop\Applications\Barcode\EAN8.dev
  Read a bar code of type EAN 8
- examples\hdevelop\Applications\Barcode\inspect_scanlines.dev
  Inspect scanlines of bar codes
- examples\hdevelop\Applications\Barcode\RSS14.dev
  Read bar codes of type RSS-14
- examples\hdevelop\Applications\Barcode\RSS14Stacked.dev
  Read bar codes of type RSS-14 Stacked
- examples\hdevelop\Applications\Barcode\RSS14StackedOmnidir.dev
  Read bar codes of type RSS-14 Stacked Omnidirectional
- examples\hdevelop\Applications\Barcode\RSS14Truncated.dev
  Read bar codes of type RSS-14 Truncated and show the influence of false candidates
- examples\hdevelop\Applications\Barcode\RSSComposite.dev
  Read bar codes of type RSS Composite
- examples\hdevelop\Applications\Barcode\RSSExpanded.dev
  Read a bar code of type RSS Expanded

- examples\hdevelop\Applications\Barcode\RSSExpandedStacked.dev
  Read bar codes of type RSS Expanded Stacked
- examples\hdevelop\Applications\Barcode\RSSLimited.dev
  Read a bar code of type RSS Limited
- examples\hdevelop\Tools\Grid-Rectification\grid_rectification.dev
  Rectify an arbitrarily distorted image using a regular grid
- examples\solution_guide\3d_machine_vision\grid_rectification_ruled_surface.dev
  Rectify an arbitrarily distorted image using a regular grid
  → description in the Solution Guide II-F on page 130

C++

- examples\cpp\source\ean13.cpp
  Reading a 1d bar code

Visual Basic

- examples\vb\Online\Barcode\barcode.vbp
  Reading bar codes in a live image

14.4 Selecting Operators

14.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

14.4.2 Preprocess Image(s)

Standard:

invert_image

14.4.3 Rectify Image(s)

Operators for rectifying images are described in the Solution Guide II-F.

14.4.4 Create Bar Code Model

Standard:

create_bar_code_model

14.4.5 Adjust Bar Code Model

Standard:

get_bar_code_param, set_bar_code_param

14.4.6 Read Bar Code(s)

Standard:

find_bar_code

Advanced:

get_bar_code_object, get_bar_code_result

14.4.7 Destroy Bar Code Model

Standard:

clear_bar_code_model

14.4.8 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

14.5 Relation to Other Methods

14.5.1 Alternatives to 1D Bar Code

OCR (see description on page 233)
With some bar codes, e.g., the EAN 13, the content of the bar code is printed in plain text below the elements. Here, OCR can be used, e.g., to check the consistency of the reading process.

14.6 Tips & Tricks

14.6.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is useful for bar code reading. With domains, the processing can be focused to a certain area in the image and thus sped up. The smaller

the region in which the bar code is searched for, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

Chapter 15

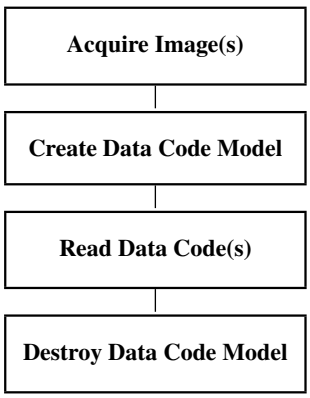
2D Data Code

Data codes are a special kind of two-dimensional patterns that encode text and numbers. HALCON is able to read the three most popular data codes: Data Matrix ECC 200, QR Code, and PDF417. These codes consist of a so-called finder pattern, which is used to locate the pattern and get basic information about the geometric properties. The code itself contains multiple dots or small squares. Because of the special design of the codes, they can be decoded even if some parts are disturbed.

The advantage of the HALCON data code reader is its ease of use. No advanced experience in programming or image processing is required. Only a few operators in a clear and simple order need to be applied. Furthermore, the data code reader is very powerful and flexible. Examples for this are its ability to read codes in many print styles and the possibility to automatically learn optimal parameters.

15.1 Basic Concept

Data code reading consists mainly of four steps:



15.1.1 Acquire Image(s)

For the online part, i.e., during reading, images are acquired.

For detailed information see the description of this method on page 13.

15.1.2 Create Data Code Model

First, you create a data code model with the operator `create_data_code_2d_model`. This model provides the reader with all necessary information about the structure of the code. For normal printed codes only the name needs to be provided and HALCON will select suitable default parameters. For special cases, you can modify the model by passing specific parameters.

15.1.3 Read Data Code(s)

To read a data code, just one operator is needed: `find_data_code_2d`. It will locate one or more data codes and decode the content.

15.1.4 Destroy Data Code Model

When you no longer need the data code model, you destroy it with the operator `clear_data_code_2d_model`.

15.1.5 A First Example

As an example for this basic concept, here a very simple program, which reads the data code on the chip depicted in figure 15.1, is discussed.

Figure 15.1: Reading a data code.

After reading an image from file, the data code model is generated by calling create_data_code_2d_model. As the only required parameter value, the code name 'ECC200' is specified.

```
read_image (Image, 'datacode/ecc200/ecc200_cpu_003')
create_data_code_2d_model ('ECC200', [], [], DataCodeHandle)
```

Then, the data code is read with the operator find_data_code_2d.

```
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [],
                   ResultHandles, DecodedDataStrings)
```

At the end of the program, the data code model is destroyed using clear_data_code_2d_model.

```
clear_data_code_2d_model (DataCodeHandle)
```

15.2 Extended Concept

In some cases, data code reading can be more advanced than in the example above. Reasons for this are, e.g., parameter optimization for improved execution time. Furthermore, preprocessing like rectification or the visualization of results might be required. The following sections give a brief overview. More detailed information can be found in the Solution Guide II-C.

15.2.1 Acquire Image(s)

Optionally, additional images can be acquired for parameter optimization (see the description of the step Optimize Model on page 223).

For detailed information see the description of this method on page 13.

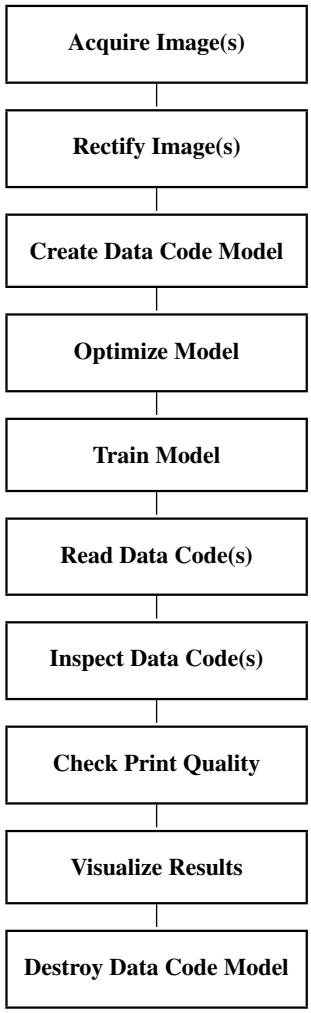
15.2.2 Rectify Image(s)

HALCON's data code reader is robust against image distortions up to a certain limit. But if the data code is printed on a cylindrical surface or if the camera is tilted relative to the surface, it might be necessary to rectify the image before applying the data code reader.

Detailed information about rectifying images can be found in the Solution Guide II-F in section 3.3 on page 49.

15.2.3 Create Data Code Model

The operator create_data_code_2d_model expects the name of the desired code and optionally parameters to specify the geometry and radiometry as input. By default, a parameter set is used that is suitable for data codes that fulfill the following requirements:



- The code must be printed dark on light,
- the contrast must be bigger than 30,
- the size of symbol is smaller or equal to 48x48 modules,
- the width and the height of the modules are between 6 and 20 pixels, and
- there is no or only a small gap in between neighboring modules.

This parameter set is also used if you specify the value `'standard_recognition'` for the parameter `GenParamValues`. In contrast, if you specify the value `'enhanced_recognition'`, a parameter set is used that detects codes that do not follow the rules given above. However, using this parameter set possibly results in a longer processing time.

15.2.4 Optimize Model

Using the default parameters, the data code reader is able to read a wide range of codes. For non-standard codes the parameters can be adapted accordingly. For this, the operator set_data_code_2d_param is used.

The easiest way is to use the parameter value 'enhanced_recognition', which uses a model that is able to find a very wide range of print styles. An alternative is to specify parameter values separately to adapt the model to the conditions of the used print style.

If a data code symbol is not detected although it is well visible in the image, check whether the symbol's appearance complies with the model. In particular, have a look at the polarity ('polarity': dark-on-light or light-on-dark), the module size ('module_size' and 'module_shape') and the minimum contrast ('contrast_min'). In addition, the parameters 'module_gap' (allowed gap between modules), 'symbol_size', and 'slant_max' (angle variance of the legs of the finder pattern) should be checked. The current settings of these values can be queried by the operator get_data_code_2d_param.

All possible parameter values can be checked in the Reference Manual. Besides this, they can also be queried with the operator query_data_code_2d_params.

As an alternative, you can train the model (see below).

15.2.5 Train Model

Instead of modifying the model parameters manually as described above, you can also let HALCON train the model automatically using the operator find_data_code_2d. All you need to do is to call this operator with the parameter values 'train' and 'all'. Then, HALCON will search for the best parameters needed to extract the given code. It is recommended to apply this to multiple example images to ensure that all variations are covered.

As an alternative, you can execute the finder with normal parameters and request the features of the found symbols with get_data_code_2d_results. These values can then be used to change the model with set_data_code_2d_param.

15.2.6 Read Data Code(s)

The operator find_data_code_2d returns for every successfully decoded symbol the surrounding XLD contour in SymbolXLDs, a handle to a result structure, which contains additional information about the symbol as well as about the search and decoding process (ResultHandles), and the string that is encoded in the symbol (DecodedDataStrings). With the result handles and the operators get_data_code_2d_results and get_data_code_2d_objects, additional data about the extraction process can be accessed.

15.2.7 Inspect Data Code(s)

Using the handles of the successfully decoded symbols returned by find_data_code_2d, you can request additional information about the symbol and the finding process using the operators

get_data_code_2d_results and get_data_code_2d_objects. This is useful both for process analysis and for displaying.

In addition, information about rejected candidates can also be queried by requesting the corresponding handles with get_data_code_2d_results using, e.g., the parameter values 'all_undecoded' and 'handle'.

The operator get_data_code_2d_results gives access to several alphanumerical results that were calculated while searching and reading the data code symbols. Besides basic information like the dimensions of the code, its polarity, or the found contrast, also the raw data can be accessed.

The operator get_data_code_2d_objects gives access to iconic objects that were created while searching and reading the data code symbols. Possible return values are the surrounding contours or the regions representing the foreground or background modules.

15.2.8 Check Print Quality

If your first aim is not to quickly read the 2D data code symbols but to check how good they were printed, with get_data_code_2d_results you can query the print quality of a symbol in accordance to the standard ISO/IEC 15415. For details, see the Solution Guide II-C in section 6 on page 43).

15.2.9 Visualize Results

Finally, you might want to display the images, the data code regions, and the decoded content.

For detailed information see the description of this method on page 269.

15.3 Programming Examples

This section gives a brief introduction to the programming of the data code reader.

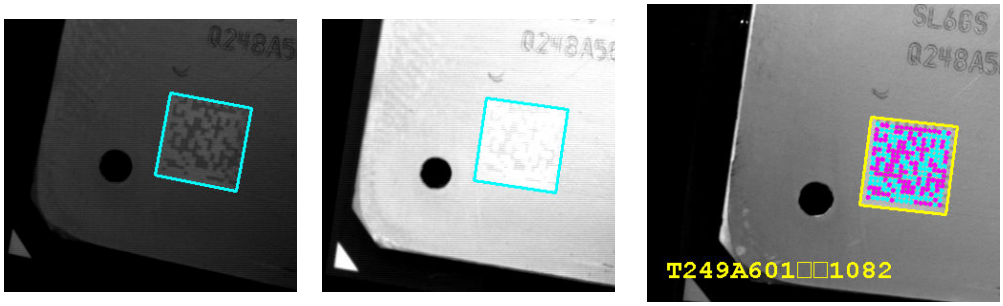
15.3.1 Training a Data Code Model

Example: examples\solution_guide\basics\ecc200_training_simple.dev

In this example we show how easy it is to train a data code model, here to allow changes in the illumination of the images. To prepare the reading of the data codes, three major steps are performed: First, the connection to an image sequence is established by calling open_framegrabber.

```
SequenceName := 'datacode/ecc200/ecc200_cpu_light.seq'
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1,
                   'default', SequenceName, 'default', -1, -1, FGHandle)
```

Then, a model for an ECC 200 is created with create_data_code_2d_model.



a) b) c)

Figure 15.2: (a) Dark training image; (b) bright training image; (c) read code with extracted modules.

```
create_data_code_2d_model ('ECC200', [], [], DataCodeHandle)
```

To get the optimal parameters for finding the data code, two sample images are loaded and passed to `find_data_code_2d` with the parameter value `'train'`.

```
grab_image (Image, FGHandle)
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all',
                   ResultHandles, DecodedDataStrings)
grab_image (Image, FGHandle)
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all',
                   ResultHandles, DecodedDataStrings)
```

As a preparation for the detection loop, the settings for the find operator are changed with `set_data_code_2d_param` to enable the saving of visual data.

```
set_data_code_2d_param (DataCodeHandle, 'persistence', 1)
```

Inside the `while`-loop, images are grabbed and `find_data_code_2d` is applied to read the code from the image. After that, iconic results and the read code are visualized. The regions with the foreground and background modules are requested with `get_data_code_2d_objects`.

```
while (1)
  grab_image (Image, FGHandle)
  dev_display (Image)
  find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [],
                     ResultHandles, DecodedDataStrings)
  dev_set_color ('yellow')
  dev_display (SymbolXLDs)
  set_tposition (WindowHandle, 430, 80)
  write_string (WindowHandle, DecodedDataStrings)
  get_data_code_2d_objects (Foreground, DataCodeHandle, ResultHandles[0],
                            'module_1_rois')
  get_data_code_2d_objects (Background, DataCodeHandle, ResultHandles[0],
                            'module_0_rois')
  dev_set_color ('cyan')
  dev_display (Foreground)
  dev_set_color ('magenta')
  dev_display (Background)
endwhile
```

At the end of the program the model for the data code reader is destroyed and the connection to the image sequence is closed.

```
clear_data_code_2d_model (DataCodeHandle)
close_framegrabber (FGHandle)
```

15.3.2 Reading 2D Data Codes on Chips

Example: `examples\hdevelop\Applications\Datacode\ecc200_optimized.dev`

This example program reads 2D data codes of type ECC200, which like the example described before are engraved in chips (see figure 15.3).

Figure 15.3: Decoded data code.

The example shows how to set optimized parameters for efficient data code reading. The code printed on a chip is always light on dark in this application and has a given size and number of modules. Also, the contrast is within a predefined range. By specifying these values for the model, the execution time can be sped up significantly.

```
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
set_data_code_2d_param (DataCodeHandle, ['module_size_min',
                        'module_size_max'], [4,7])
set_data_code_2d_param (DataCodeHandle, 'module_gap', 'no')
set_data_code_2d_param (DataCodeHandle, 'polarity', 'light_on_dark')
set_data_code_2d_param (DataCodeHandle, 'mirrored', 'no')
set_data_code_2d_param (DataCodeHandle, 'contrast_min', 10)
set_data_code_2d_param (DataCodeHandle, 'symbol_size', 18)

find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [],
                   ResultHandles, DecodedDataStrings)
```

15.3.3 Other Examples

HDevelop

- examples\hdevelop\Applications\Datacode\ecc200_simple.dev
  Read 2D data codes of type ECC200
- examples\hdevelop\Applications\Datacode\pdf417_optimized.dev
  Optimize parameters to read 2D data codes of type PDF417
- examples\hdevelop\Applications\Datacode\pdf417_simple.dev
  Read 2D data codes of type PDF417
- examples\hdevelop\Applications\Datacode\qrcode_optimized.dev
  Optimize parameters to read 2D data codes of type QR Code
- examples\hdevelop\Applications\Datacode\qrcode_simple.dev
  Read 2D data codes of type QR Code
- examples\hdevelop\Tools\Datacode\ecc200_default_settings.dev
  Read a 2d data code of type ECC200
- examples\hdevelop\Tools\Datacode\ecc200_optimized_settings.dev
  Optimize parameters for reading a 2d data code of type ECC200
- examples\hdevelop\Tools\Datacode\ecc200_print_quality.dev
  Inspect the print quality of an ECC200 data code
- examples\hdevelop\Tools\Datacode\ecc200_read_model.dev
  Load previously stored parameters for 2d data codes of type ECC200
- examples\hdevelop\Tools\Datacode\ecc200_training.dev
  Train parameters for reading a 2d data code of type ECC200

- examples\hdevelop\Tools\Datacode\ecc200_write_model.dev
  Store parameters for 2d data codes of type ECC200
- examples\hdevelop\Tools\Datacode\pdf417_default_settings.dev
  Read various 2d data codes of type PDF417
- examples\hdevelop\Tools\Datacode\pdf417_optimized_settings.dev
  Optimize parameters for reading various 2d data codes of type PDF417
- examples\hdevelop\Tools\Datacode\qrcode_default_settings.dev
  Read a 2d data code of type QR Code
- examples\hdevelop\Tools\Datacode\qrcode_optimized_settings.dev
  Optimize parameters for reading a 2d data code of type QR Code
- examples\solution_guide\2d_data_codes\2d_data_codes_arbitrary_distortions.dev
  Demonstrate distorted symbols of type ECC200
  → description in the Solution Guide II-C on page 39
- examples\solution_guide\2d_data_codes\2d_data_codes_data_access.dev
  Access intermediate results when trying to read symbols of type ECC200
  → description in the Solution Guide II-C on page 27
- examples\solution_guide\2d_data_codes\2d_data_codes_enlarge_modules.dev
  Increase the resolution of a symbol of type ECC200
  → description in the Solution Guide II-C on page 23
- examples\solution_guide\2d_data_codes\2d_data_codes_first_example.dev
  Read symbols of type ECC200
  → description in the Solution Guide II-C on page 6
- examples\solution_guide\2d_data_codes\2d_data_codes_global_settings.dev
  Switch between standard and enhanced mode for reading symbols of type ECC200
  → description in the Solution Guide II-C on page 9
- examples\solution_guide\2d_data_codes\2d_data_codes_manual_settings.dev
  Set parameters manually for reading symbols of type ECC200
  → description in the Solution Guide II-C on page 13
- examples\solution_guide\2d_data_codes\2d_data_codes_minimize_module_gaps.dev
  Minimize large gaps in a symbol of type ECC200
  → description in the Solution Guide II-C on page 24
- examples\solution_guide\2d_data_codes\2d_data_codes_minimize_noise.dev
  Minimize noise in a symbol of type ECC200
  → description in the Solution Guide II-C on page 25
- examples\solution_guide\2d_data_codes\2d_data_codes_rectify_symbol.dev
  Rectify a slanted symbol of type ECC200
  → description in the Solution Guide II-C on page 22
- examples\solution_guide\2d_data_codes\2d_data_codes_training.dev
  Train a 2D data code model for symbols of type ECC200
  → description in the Solution Guide II-C on page 11

- examples\solution_guide\2d_data_codes\read_2d_data_code_model.dev
  Read a 2D data code model from file
  → description in the Solution Guide II-C on page 21
- examples\solution_guide\2d_data_codes\write_2d_data_code_model.dev
  Train a 2D data code model and write it into a file
  → description in the Solution Guide II-C on page 21

C++

- examples\cpp\source\ecc200.cpp
  Reading 2d data codes of type ECC200
- examples\mfc\MultiThreading\MultiThreading.cpp
  Use multiple threads for image acquisition, processing, and display

C#

-
  Use multiple threads for image acquisition, processing, and display
- examples\hdevengine\c#\MultiThreadingTwoWindows\vs.net\MultiThreadingTwoWindows.csproj
  Execute different HDevelop procedures in parallel by two threads using Parallel HDevEngine
  → description in the Programmer's Guide on page 202

15.4 Selecting Operators

15.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

15.4.2 Rectify Image(s)

Operators for rectifying images are described in the Solution Guide II-F.

15.4.3 Create Data Code Model

Standard:

create_data_code_2d_model

15.4.4 Optimize Model

Standard:

set_data_code_2d_param, get_data_code_2d_param

Advanced:

query_data_code_2d_params

15.4.5 Train Model

Standard:

find_data_code_2d

Advanced:

get_data_code_2d_results, set_data_code_2d_param

15.4.6 Read Data Code(s)

Standard:

find_data_code_2d

15.4.7 Inspect Data Code(s)

Standard:

get_data_code_2d_results, get_data_code_2d_objects

15.4.8 Check Print Quality

Standard:

get_data_code_2d_results

15.4.9 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

15.4.10 Destroy Data Code Model

Standard:

`clear_data_code_2d_model`

15.5 Tips & Tricks

15.5.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is useful for data code reading. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the data code region is searched for can be restricted, the faster and more robust the search. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

Chapter 16

OCR

Optical Character Recognition (OCR) is the technical term for reading, i.e., identifying symbols. In HALCON, OCR is defined as the task to assign an interpretation to regions of an image. These regions typically represent single characters and therefore we consider this as reading single symbols.

In an offline phase, the characters are trained by presenting several samples for each character. In the online phase, the image is segmented to extract the regions representing the characters and then the OCR reader is applied to get the interpretation for each character.

Figure 16.1 shows the principal steps. The first part is offline and consists of collecting training samples and, after that, applying the training. The online part consists of extracting the characters and then reading them.

The advantage of OCR is the flexibility of the training, which allows to select features optimized for an application. Furthermore, the used classifiers are based on the latest neural network technology or on support vector machines, providing both best possible performance.

As a further advantage, HALCON provides you with a set of pretrained fonts, which are based on a large amount of training data from various application areas. These fonts allow you to read text in documents, on pharmaceutical or industrial products, dot prints, and even handwritten numbers. Furthermore, HALCON includes pretrained fonts for OCR-A and OCR-B.

16.1 Basic Concept

The OCR is split into two major parts: training and reading. Each of these major parts requires additional preparation steps:

16.1.1 Acquire Image(s)

Both for the generation of training data and for the OCR itself images must be acquired.

For detailed information see the description of this method on page 13.

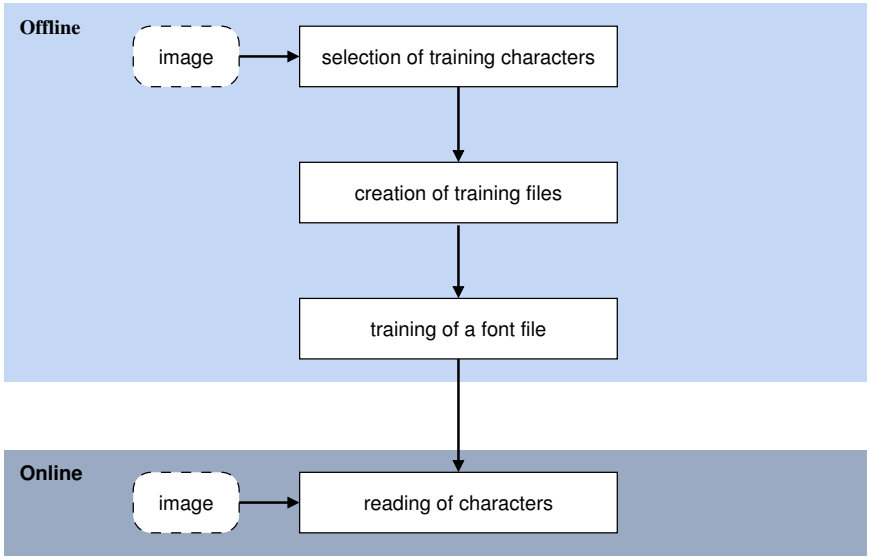
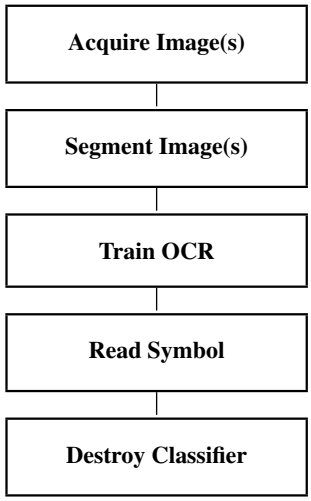


Figure 16.1: Main steps of OCR.



16.1.2 Segment Image(s)

Both for the training samples and for the online reading process, characters must be extracted from the image. This step is called segmentation. This means that the OCR operators like `do_ocr_single_class_svm` do not search for the characters within a given region of interest, but expect a segmented region, which then will be classified.

If the samples for training are taken from real application images, the same segmentation method will be applied for both training and reading. If the training images are more "artificial", a simpler method

might be used to segment the training images.

16.1.3 Train OCR

The training consists of two important steps: First, a number of samples for each character is selected and stored in so-called training files. In the second step, these files are input for a newly created OCR classifier.

As already noted, HALCON provides pretrained fonts, i.e., ready-to-use classifiers, which already solve many OCR applications. These fonts can be found in the subdirectory `ocr` of the folder where you installed HALCON.

16.1.4 Read Symbol

For reading, you only have to read the trained classifier from disk, segment the characters from the image, and use the segmented characters as input for one of the reading operators that will be discussed later.

16.1.5 Destroy Classifier

When you no longer need the classifier, you destroy it with the operator `clear_ocr_class_mlp` or `clear_ocr_class_svm`, dependent on the used classifier.

16.1.6 A First Example

An example for this basic concept is the following program, which uses one of the pretrained fonts provided by HALCON to read the numbers in the image depicted in figure 16.2.

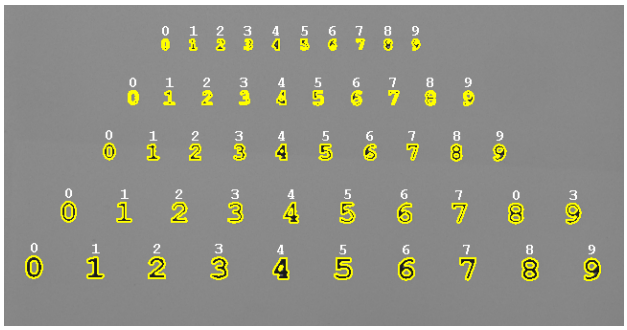


Figure 16.2: Applying a pretrained classifier.

First, the pretrained font `Document_0-9` is read using `read_ocr_class_mlp`. As no file extension is specified, it is searched for a file with the MLP specific extension ".omc" (since HALCON 8.0) or with the extension ".fnt", which was common for box and MLP classifiers in earlier HALCON versions.

```
FontFile := 'Document_0-9'
read_ocr_class_mlp (FontFile, OCRHandle)
```

Then, the numbers are segmented using `threshold` and `connection`. Because the order is irrelevant here, no further processing is applied.

```
threshold (Image, Region, 0, 125)
connection (Region, Characters)
```

Finally, the numbers are read in a for-loop. The operator `do_ocr_single_class_mlp` takes the single region, the image, and the OCR handle as input. As a result the best and the second best interpretation together with the confidences are returned.

```
count_obj (Characters, Number)
  do_ocr_single_class_mlp (SingleChar, Image, OCRHandle, 2, Class,
                           Confidence)
endfor
```

16.2 Extended Concept

When we look more closely at the OCR, many possibilities for adaptation to specific applications become apparent. For example, we have to consider an efficient way of collecting samples as well as correct parameters for the training. In the online phase, an optimal segmentation method is required to extract all characters in a robust manner.

16.2.1 Align ROIs Or Images

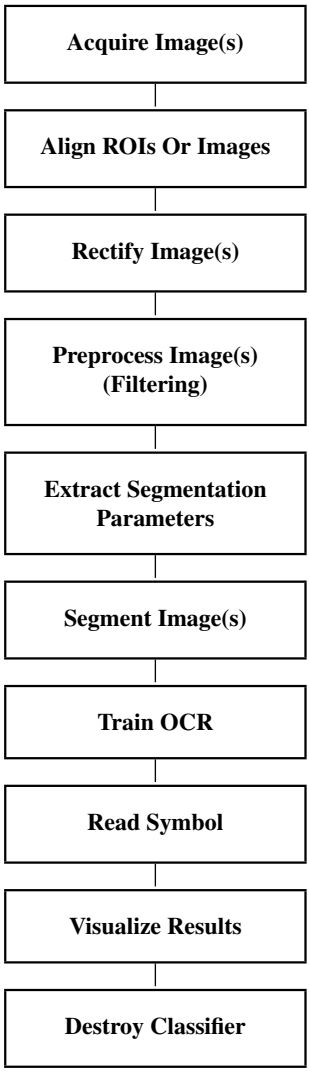
Because the reading of characters is not invariant to rotation, it may be necessary to correct the orientation of the image. This can be achieved either by directly determining the orientation of the text using the operator `text_line_orientation`, or by locating another object. Then, the part of the image containing the characters is cropped and aligned using the orientation of the text or the found object.

How to perform alignment using shape-based matching is described in the Solution Guide II-B in section 4.3.4 on page 37.

16.2.2 Rectify Image(s)

Similarly to alignment, it may be necessary to rectify the image, e.g., to remove perspective distortions. For slanted characters, the slant angle can be determined using the operator `text_line_slant`.

Detailed information about rectifying images can be found in the Solution Guide II-F in section 3.3 on page 49.



16.2.3 Preprocess Image(s) (Filtering)

Sometimes, the characters may be difficult to extract because of noise, texture, or overlaid structures. Here, operators like `mean_image` or `gauss_image` can be used to eliminate noise. A fast but slightly less perfect alternative to `gauss_image` is `binomial_filter`. The operator `median_image` is helpful for suppressing small spots or thin lines. The operator `dots_image` is optimized to emphasize a dot-print while suppressing other structures in the image. Gray value morphology can be used to eliminate noise structures and to adapt the stroke width of characters.

16.2.4 Extract Segmentation Parameters

When segmenting the characters with blob analysis, instead of using fixed threshold values, the values can be extracted dynamically for each image. For more details, please refer to the description of this step on page 36.

16.2.5 Segment Image(s)

For the segmentation various methods can be used. You can either use the operator `segment_characters` to get a region containing all character candidates and then apply `select_characters` to select those parts of the region that are candidates for the individual characters, or you use blob analysis. There, the most simple method is the operator `threshold`, with one or more gray value ranges specifying the regions that belong to the foreground objects. Another very common method is `dyn_threshold`. Here, a second image is passed as a reference. With this approach a local instead of a global threshold is used for each position.

Further information can be found in the description of this step for blob analysis on page 34.

16.2.6 Train OCR

Figure 16.3 shows an overview on the generation of the training files: First, the characters from sample images must be extracted using a segmentation method (see above). To each of the single characters a name must be assigned. This can be done either by typing it in, by a programmed input in the case of a well-structured image (having, e.g., multiple samples of each character in different lines), or by reading the character names from file. Then, the regions together with their names are written into training files. The most convenient operator to do this is `append_ocr_trainf`. Before applying the training, we recommend to check the correctness of the training files. This can, e.g., be achieved by using the operator `read_ocr_trainf` combined with visualization operators.

The actual training is depicted in figure 16.4. First, a new classifier is created. There are three different OCR classifiers available, the box classifier, a neural network (multi-layer perceptron or MLP) classifier, and a classifier that is based on support vector machines (SVM). Note that the box classifier is still available, but not recommended anymore as MLP and SVM are more powerful. Thus, in this manual only MLP and SVM are described further. Dependent on the chosen classifier, you create the classifier using `create_ocr_class_mlp` or `create_ocr_class_svm`. Then, the training is applied using `trainf_ocr_class_mlp` or `trainf_ocr_class_svm`, respectively. After the training, you typically save the classifier to disk for later use by `write_ocr_class_mlp` or `write_ocr_class_svm`.

The difference between the two recommended classifiers is very simple: The MLP classifier is fast at classification, but for a large training set slow at training (compared to the classifier based on SVM). If the training can be applied offline and thus is not time critical, MLP is a good choice. The classifier based on SVM leads to slightly better recognition rates than the MLP classifier and is faster at training (especially for large training sets). But compared to the MLP classifier the classification need more time.

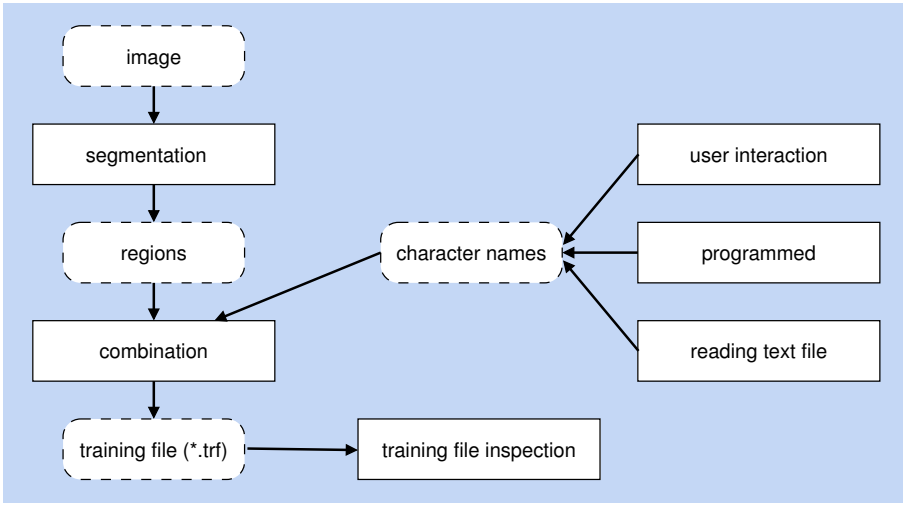


Figure 16.3: Creating training files.

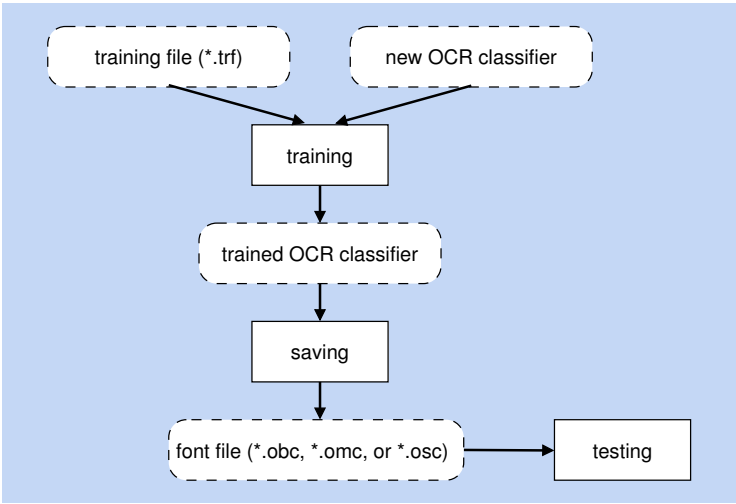


Figure 16.4: Training an OCR classifier.

16.2.7 Read Symbol

Figure 16.5 shows an overview on the reading process. First, the characters must be extracted using an appropriate segmentation method. Here, you must use a method that returns the characters in a form similar to the ones used for training. After reading the classifier (font file) from file (`read_ocr_class_mlp` or `read_ocr_class_svm`), the classifier can be used for reading.

For reading multiple operators are provided: In the easiest case, multiple characters are passed to the

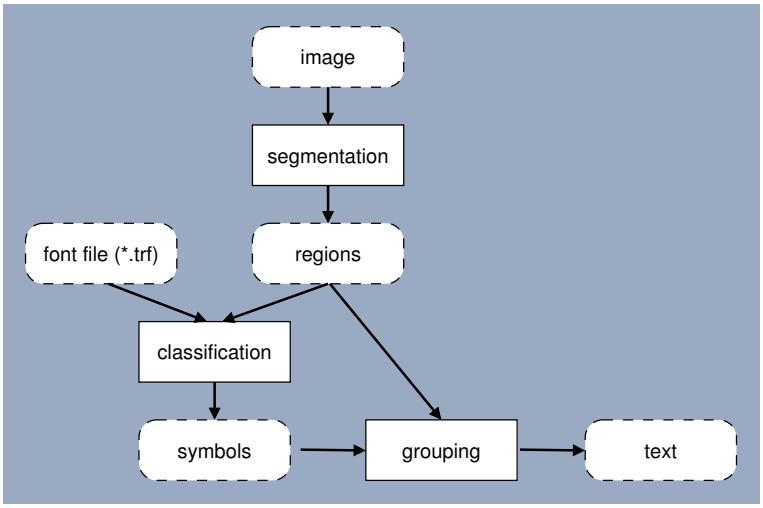


Figure 16.5: Reading characters.

reading operators (do_ocr_multi_class_mlp or do_ocr_multi_class_svm). Here, for each region the corresponding name and the confidence are returned. Sometimes, it can be necessary not only to obtain the characters with the highest confidence but also others with lower confidences. A zero, e.g., might easily be mistaken for the character "O". This information is returned by the operators do_ocr_single_class_mlp and do_ocr_single_class_svm.

As a final step it might be necessary to group digits to numbers or characters to words. This can be realized with the region processing operators like those described for the method blob analysis on page 37.

Additionally, HALCON provides operators for a syntactic and lexicon-based auto-correction. For example, you can use the operator do_ocr_word_mlp instead of do_ocr_multi_class_mlp to find sets of characters, i.e., words, that match a regular expression or that are stored in a lexicon, which was created by create_lexicon or imported by import_lexicon.

16.2.8 Visualize Results

Finally, you might want to display the images, the blob (regions), and the result of the reading process.

For detailed information see the description of this method on page 269.

16.3 Programming Examples

This section gives a brief introduction to using HALCON for OCR. All important steps from training file generation over training to reading are presented.

16.3.1 Generating a Training File

Example: examples\solution_guide\basics\gen_training_file.dev

Figure 16.6 shows a training image from which the characters in the fourth line are used as training samples. For this example image, the segmentation is very simple because the characters are significantly darker than the background. Therefore, threshold can be used.

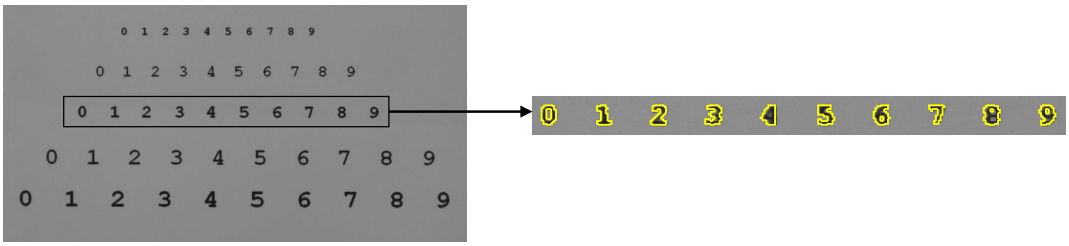


Figure 16.6: Collecting characters for a training file.

The number of the line of characters that is used for training is specified by the variable TrainingLine. To select this line, first the operator closing_rectangle1 is used to combine characters horizontally into lines. These lines are then converted to their connected components with connection. Out of all lines the relevant one is selected using select_obj. By using intersection with the original segmentation and the selected line as input, the characters for training are returned. These are sorted from left to right, using sort_region.

```
TrainingLine := 3
threshold (Image, Region, 0, 125)
closing_rectangle1 (Region, RegionClosing, 70, 10)
connection (RegionClosing, Lines)
select_obj (Lines, Training, TrainingLine)
intersection (Training, Region, TrainingChars)
connection (TrainingChars, ConnectedRegions)
sort_region (ConnectedRegions, SortedRegions, 'first_point', 'true',
             'column')
```

Now, the characters can be stored in the training file. As a preparation step a possibly existing older training file is deleted. Within a loop over all characters the single characters are selected. The variable Chars contains the names of the characters as a tuple of strings. With the operator append_ocr_trainf the selected regions, together with the gray values and the corresponding name, are added to the training file.

```
Chars := ['0','1','2','3','4','5','6','7','8','9']
TrainFile := 'numbers.trf'
dev_set_check ('~give_error')
delete_file (TrainFile)
dev_set_check ('give_error')
for i := 1 to 10 by 1
  select_obj (SortedRegions, TrainSingle, i)
  append_ocr_trainf (TrainSingle, Image, Chars[i-1], TrainFile)
endfor
```

16.3.2 Creating and Training an OCR Classifier

Example: examples\solution_guide\basics\simple_training.dev

Having prepared the training file, the creation and training of an OCR classifier is very simple. First, the names of the training file and the final font file are determined. Typically, the same base with different extensions is used. We recommend to use ".trf" for training files. For font files i.e., for OCR classifiers, we recommend to use ".obc" for the box classifier (which is not recommended anymore), ".omc" for the neural network classifier , and ".osc" for the classifier based on support vector machines. If no extension is specified during the reading process, for the box or neural network classification it is also searched for files with the extension ".fnt", which was common for both classifiers in earlier HALCON versions.

To create an OCR classifier, some parameters need to be determined. The most important one is the list of all possible character names. This list can easily be extracted from the training file by using the operator read_ocr_trainf_names.

```
TrainFile := 'numbers.trf'
read_ocr_trainf_names (TrainFile, CharacterNames, CharacterCount)
```

Another important parameter is the number of nodes in the hidden layer of the neural network. In this case, it is set to 20. As a rule of thumb, this number should be in the same order as the number of different symbols. Besides these two parameters, here only default values are used for create_ocr_class_mlp. The training itself is applied using trainf_ocr_class_mlp. We recommend to simply use the default values here as well.

```
NumHidden := 20
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames,
                      NumHidden, 'none', 1, 42, OCRHandle)
trainf_ocr_class_mlp (OCRHandle, TrainFile, 200, 1, 0.01, Error, ErrorLog)
```

Finally, the classifier is stored to disk and the memory is freed.

```
FontFile := 'numbers.omc'
write_ocr_class_mlp (OCRHandle, FontFile)
clear_ocr_class_mlp (OCRHandle)
```

16.3.3 Reading Numbers

Example: examples\solution_guide\basics\simple_reading.dev

This example works similar to the first example described for the basic concept , i.e., it uses a font file to read the numbers in the image depicted in figure 16.2. Instead of a pretrained font that is provided by HALCON, here the font trained in the previous example is used. That means, this time the file `numbers.omc` instead of the file `Document_0-9.omc` is used. The rest of the program corresponds to the first example.

```
FontFile := 'numbers.omc'
read_ocr_class_mlp (FontFile, OCRHandle)
```

16.3.4 "Best Before" Date

Example: examples\hdevelop\Applications\OCR\bottle.dev

The task of this example is to inspect the "best before" date on the bottle depicted in figure 16.7.

a) b)

Figure 16.7: (a) Original image; (b) read date.

The task is solved in multiple steps. First, dark areas are extracted and post-processed to eliminate structures that are too thin.

```
threshold (Bottle, RawSegmentation, 0, 95)
fill_up_shape (RawSegmentation, RemovedNoise, 'area', 1, 5)
opening_circle (RemovedNoise, ThickStructures, 2.5)
fill_up (ThickStructures, Solid)
```

Next, the region is split into the individual characters; even the characters that are so close that they touch each other can be separated.

```
opening_rectangle1 (Solid, Cut, 1, 7)
connection (Cut, ConnectedPatterns)
intersection (ConnectedPatterns, ThickStructures, NumberCandidates)
select_shape (NumberCandidates, Numbers, 'area', 'and', 300, 9999)
sort_region (Numbers, FinalNumbers, 'first_point', 'true', 'column')
```

Finally, the actual reading is performed.

```
read_ocr_class_mlp (FontName, OCRHandle)
do_ocr_multi_class_mlp (FinalNumbers, Bottle, OCRHandle, RecNum,
                        Confidence)
```

16.3.5 Reading Engraved Text

Example: examples\hdevelop\Applications\OCR\engraved.dev

The task of this example is to read the engraved text on the metal surface depicted in figure 16.8.

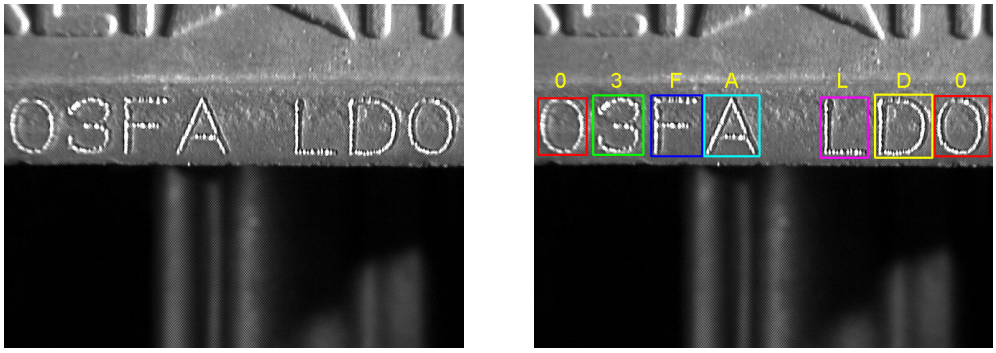


Figure 16.8: (a) Original image; (b) read characters.

The segmentation is solved by using advanced blob analysis: The characters cannot simply be extracted by selecting dark or light pixels. Instead, a simple segmentation would yield only fractions of the characters together with noise objects. Preprocessing the image using gray value morphology allows to segment the real characters.

```
gray_range_rect (Image, ImageResult, 7, 7)
invert_image (ImageResult, ImageInvert)
threshold (ImageResult, Region, 128, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 1000,
              99999)
sort_region (SelectedRegions, SortedRegions, 'first_point', 'true',
             'column')
```

Finally, the actual reading is performed.

```
read_ocr_class_mlp (FontName, OCRHandle)
for i := 1 to Number by 1
  select_obj (SortedRegions, ObjectSelected, i)
  do_ocr_single_class_mlp (ObjectSelected, ImageInvert, OCRHandle, 1,
                           Class, Confidence)
endfor
clear_ocr_class_mlp (OCRHandle)
```

16.3.6 Reading Forms

Example: examples\hdevelop\Applications\OCR\ocrcolor.dev

The task of this example is to extract the symbols in the form. A typical problem is that the symbols are not printed in the correct place, as depicted in figure 16.9.

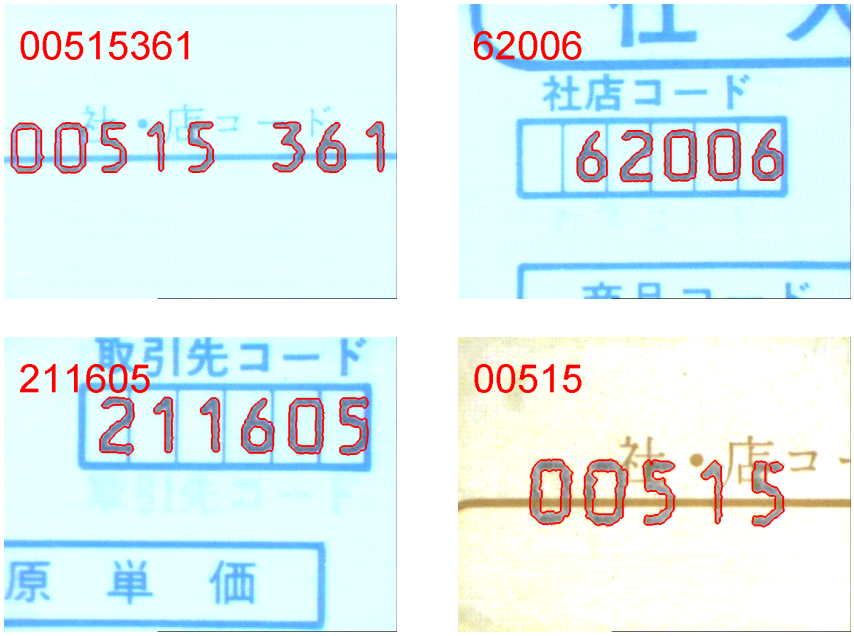


Figure 16.9: Example images for OCR.

To solve the problem of numbers printed on lines, color is used here: The hue value of the characters differs from the hue of the form. The color classification method is a very simple way to save execution time: In contrast to more difficult color processing problems, here it is sufficient to consider the difference of the red and green channel combined with the intensity.

```
threshold (Green, ForegroundRaw, 0, 220)
sub_image (RedReduced, GreenReduced, ImageSub, 2, 128)
mean_image (ImageSub, ImageMean, 3, 3)
bin_threshold (ImageMean, Cluster1)
difference (Foreground, Cluster1, Cluster2)
concat_obj (Cluster1, Cluster2, Cluster)
opening_circle (Cluster, Opening, 2.5)
```

The selected pixels are grouped and post-processed with morphological operators.

```
closing_rectangle1 (NumberRegion, NumberCand, 1, 20)
difference (Image, NumberCand, NoNumbers)
connection (NumberRegion, NumberParts)
intensity (NumberParts, Green, MeanIntensity, _)
expand_gray_ref (NumberParts, Green, NoNumbers, Numbers, 20, 'image',
                 MeanIntensity, 48)
union1 (Numbers, NumberRegion)
connection (NumberRegion, Numbers)
```

For reading, it is important not to use the gray values of the background because of the changes in color. To solve this, only region features are used for the font, i.e., the regions are used to create an artificial image that is printed dark on light.

```
paint_region (NoNumbers, Green, ImageOCRRaw, 255, 'fill')
paint_region (NumberRegion, ImageOCRRaw, ImageOCR, 0, 'fill')
```

The actual character classification is performed in the artificial image.

```
read_ocr_class_mlp ('Industrial_0-9', OCRHandle)
  do_ocr_multi_class_mlp (FinalNumbers, ImageOCR, OCRHandle, RecChar,
                          Confidence)
clear_ocr_class_mlp (OCRHandle)
```

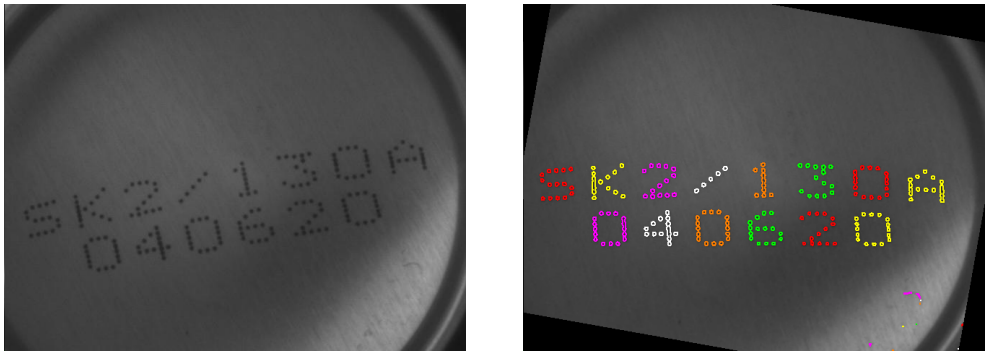
16.3.7 Segment and Select Characters

Example: examples\hdevelop\OCR\Tools\select_characters.dev

This example shows how to easily segment the characters of a rotated dot print using the segmentation operators that are provided especially for OCR (see figure 16.10).

First, the image is read from file. As the print is rotated, the orientation of the text line is determined via text_line_orientation. The obtained angle is used to rotate the image so that the print becomes horizontal.

```
read_image (Image, 'dot_print_rotated/dot_print_rotated_'+J$'02d')
text_line_orientation (Image, Image, 50, rad(-30), rad(30),
                       OrientationAngle)
rotate_image (Image, ImageRotate, -OrientationAngle/rad(180)*180,
              'constant')
```



a) b)

Figure 16.10: (a) Original image; b) selected characters.

Then, the operators segment_characters and select_characters are applied to first segment the region of the complete print and then select those parts of the region that are candidates for individual characters. In contrast to a classical segmentation using blob analysis, here the regions of the individual characters are found although they still consist of components that are not connected.

```
segment_characters (ImageRotate, ImageRotate, ImageForeground,
                    RegionForeground, 'local_auto_shape', 'false',
                    'true', 'medium', 25, 25, 0, 10, UsedThreshold)
select_characters (RegionForeground, RegionCharacters, 'true',
                   'ultra_light', 60, 60, 'false', 'false', 'none',
                   'true', 'wide', 'true', 0, 'completion')
```

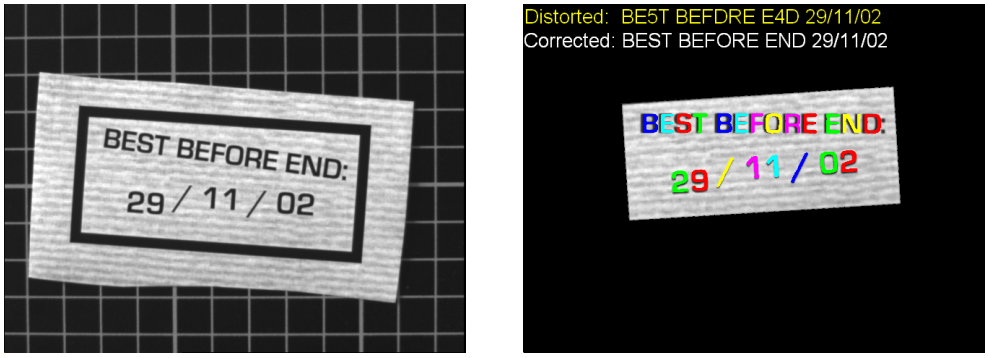
The extracted regions can now be used for an OCR application like those described above.

16.3.8 Syntactic and Lexicon-Based Auto-Correction of OCR Results

Example: examples\hdevelop\OCR\Neural-Nets\label_word_process_mlp.dev

This example reads the "best before" date depicted in figure 16.11. To correct an incorrect OCR result for the upper text line, a lexicon-based auto-correction is used. Errors can occur, e.g., because of the similarity of characters, e.g., between the character O and the number 0. For the second text line, regular expressions are used to ensure that the result has the correct format.

First, the pretrained font Industrial is read as preparation for the actual OCR reading. Then, the images are read, an ROI for the print is generated and aligned, and the regions for the characters are extracted and stored in the variable SortedWords using blob analysis.



a) b)

Figure 16.11: (a) Original image; (b) text is corrected using syntactic and lexicon-based auto-correction.

```
read_ocr_class_mlp ('Industrial', OCRHandle)
for i := 1 to 9 by 1
  read_image (Image, 'label/label_0' + i + '.png')
  ... get ROI, align print, and extract regions ...
```

For the upper line of the text, the three expected words are stored in a lexicon that is created with `create_lexicon`. Then, the text line is read twice with the operator `do_ocr_word_mlp`. The first time it is read without syntactic or lexicon-based auto-correction, and the second time the result is corrected by matching it to the lexicon entries.

```
create_lexicon ('label', ['BEST','BEFORE','END'], LexiconHandle)
sort_region (CharactersWords, SortedWords, 'character', 'true', 'row')
gen_empty_obj (Word)
Text := ''
OriginalText := ''
for j := 1 to |Column| - 1 by 1
  select_obj (SortedWords, Character, j)
  concat_obj (Word, Character, Word)
  if (j = |Column| or (Column[j] - Column[j-1]) > 30)
    do_ocr_word_mlp (Word, ImageOCR, OCRHandle, '.*', 1, 5, Class,
                     Confidence, WordText, WordScore)
    OriginalText := OriginalText + ' ' + WordText
    do_ocr_word_mlp (Word, ImageOCR, OCRHandle, '<label>', 1, 5,
                     Class, Confidence, WordText, WordScore)
    Text := Text + ' ' + WordText
    gen_empty_obj (Word)
  endif
endfor
```

The second text line, i.e., the actual date, is read using a regular expression that ensures the correct format for the date. This is done to suppress structures that may be extracted as candidates for characters but syntactically do not fit into the searched string.

```
sort_region (CharactersDate, SortedDate, 'character', 'true', 'row')
do_ocr_word_mlp (SortedDate, ImageOCR, OCRHandle, '.*', 5, 5, Class,
                 Confidence, OriginalDateText, DateScore)
do_ocr_word_mlp (SortedDate, ImageOCR, OCRHandle,
                 '^([0-2][0-9]|30|31)/(0[0-9]|11|12)/0[0-5]$', 5, 5,
                 Class, Confidence, DateText, DateScore)
```

16.3.9 Other Examples

HDevelop

- examples\hdevelop\Applications\OCR\bottlet.dev
  Train numbers on a beer bottle
- examples\hdevelop\Applications\OCR\dotprt.dev
  Read a dot print
- examples\hdevelop\Applications\OCR\engravedt.dev
  Train characters on a metal surface
- examples\hdevelop\Applications\OCR\font.dev
  Read printed characters with interactive selection
- examples\hdevelop\Applications\OCR\fontt.dev
  Train printed characters and reclassify them
- examples\hdevelop\Applications\OCR\letter.dev
  Read printed characters with interactive selection
- examples\hdevelop\Applications\OCR\lettert.dev
  Train printed characters and reclassify them
- examples\hdevelop\Applications\OCR\ocr_dongle.dev
  Read dongle ID using OCR and regular expressions
- examples\hdevelop\Applications\OCR\ocr_gradient_feature.dev
  Compare OCR features
- examples\hdevelop\Applications\OCR\ocr_lot_number.dev
  Correct OCR result using regular expressions
- examples\hdevelop\Applications\OCR\ocrcolort.dev
  Segment numbers using color information and train the OCR
- examples\hdevelop\Applications\OCR\rotchar.dev
  Estimate small inclinations of text lines
- examples\hdevelop\Filter\Points\dots_image.dev
  Segment a dot print using dots_image
- examples\hdevelop\OCR\Lexica\lexicon.dev
  Demonstrate the use of lexica

- examples\hdevelop\OCR\Neural-Nets\letters_mlp.dev
  Train an MLP OCR classifier and reclassifiy the training samples
- examples\hdevelop\OCR\Support-Vector-Machines\compare_ocr_svm_mlp.dev
  Compare the recognition rates and the training and classification speeds of the SVM and MLP OCR classifiers
- examples\hdevelop\OCR\Support-Vector-Machines\letters_svm.dev
  Train an SVM OCR classifier and reclassify the training samples
- examples\hdevelop\OCR\Tools\text_line_orientation.dev
  Detect and correct the orientation of text_lines.
- examples\hdevelop\OCR\Tools\text_line_slant.dev
  Show how to correct slanted characters for an OCR segmentation
- examples\hdevelop\XLD\Features\statistics_points_xld.dev
  Analyze 2D statistical data using *_points_xld operators

C++

- examples\cpp\source\bottle.cpp
  Reads numbers on a beer bottle
- examples\cpp\source\engraved.cpp
  Segmenting and reading characters on a metal surface

16.4 Selecting Operators

16.4.1 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

16.4.2 Align ROIs Or Images

Standard:

text_line_orientation

Further operators for aligning ROIs or images are described in the Solution Guide II-B.

16.4.3 Rectify Image(s)

Standard:

text_line_slant

Further operators for rectifying images are described in the Solution Guide II-F.

16.4.4 Preprocess Image(s) (Filtering)

Standard:

mean_image, gauss_image, binomial_filter, median_image, gray_opening_shape, gray_closing_shape, dots_image, gray_range_rect

Advanced:

gray_dilation_shape, gray_erosion_shape, anisotrope_diff

16.4.5 Extract Segmentation Parameters

Please refer to the detailed operator list for the step Extract Segmentation Parameters on page 52.

16.4.6 Segment Image(s)

Standard:

segment_characters, select_characters

Further operators can be found in the detailed operator list for the step Segment Image(s) on page 52.

16.4.7 Train OCR

Standard:

read_string, append_ocr_trainf, delete_file, read_ocr_trainf, create_ocr_class_mlp, trainf_ocr_class_mlp, write_ocr_class_mlp, create_ocr_class_svm, trainf_ocr_class_svm, write_ocr_class_svm

Advanced:

get_prep_info_ocr_class_mlp, get_prep_info_ocr_class_svm

16.4.8 Read Symbol

Standard:

read_ocr_class_mlp, do_ocr_multi_class_mlp, do_ocr_word_mlp, do_ocr_single_class_mlp, read_ocr_class_svm, do_ocr_multi_class_svm, do_ocr_word_svm, do_ocr_single_class_svm

Advanced:

clear_all_lexica, clear_lexicon, create_lexicon, import_lexicon, inspect_lexicon, lookup_lexicon, suggest_lexicon, dilation_rectangle1, closing_circle, partition_rectangle, partition_dynamic

16.4.9 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

16.4.10 Destroy Classifier

Standard:

`clear_ocr_class_mlp`, `clear_all_ocr_class_mlp`, `clear_ocr_class_svm`, `clear_all_ocr_class_svm`

16.5 Relation to Other Methods

16.5.1 Alternatives to OCR

Template Matching (see description on page 115)
As an alternative to classical OCR, matching can be used to read single characters or more complex symbols. In this case, one model for each character must be generated. The advantage of matching is the invariance to rotation. Furthermore, it is not necessary to segment the characters prior to the classification. Therefore, the matching approach should be considered when there is no robust way to separate the characters from the background.

Classification
You can consider the OCR tool as a convenient way of using a classifier. The OCR automatically derives invariant features and passes them to the underlying classifier. If the features offered by the OCR do not fulfill the needs of your application, you can create an "extended" OCR classifier by calculating the features using normal HALCON feature extraction operators and then using them with one of the classifiers that HALCON offers (see the chapters "Regions ▷ Features" and "Classification" in the Reference Manual).

16.6 Tips & Tricks

16.6.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is also useful for OCR. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the characters are searched can be restricted, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method Region Of Interest on page 19.

16.6.2 Composed Symbols

Some characters and symbols are composed of multiple sub-symbols, like an “i”, “%”, or “!”. For the OCR, these sub-symbols must be combined into a single region. If you use the operators segment_characters and select_characters for the segmentation of the characters, sub-symbols are automatically combined. Otherwise, you can combine them by calling closing_rectangle1 after thresholding, typically using a small width but a larger height. After calling connection to separate the characters, you use the operator intersection to get the original segmentation (input parameter 2), while preserving the correct connected components from connection (input parameter 1).

16.7 Advanced Topics

16.7.1 Line Scan Cameras

In general, line scan cameras are treated like normal area sensors. But in some cases, not single images but an “infinite” sequence of images showing objects, e.g., on a conveyor belt, must be processed. In this case, the end of one image is the beginning of the next one. This means that text or numbers, which partially lie in both images, must be combined into one object. For this purpose, HALCON provides the operator merge_regions_line_scan. This operator is called after the segmentation of one image, and combines the current objects with those of previous images. For more information see the Solution Guide II-A.

16.7.2 Circular Prints

In some cases the symbols are not printed as straight lines but along arcs, e.g., on a CD. To read these, the (virtual) center and radius of the corresponding circle are extracted. Using the operator polar_trans_image_ext, the image is then unwrapped. To project a region obtained in the unwrapped image back into the original image, you can use the operator polar_trans_region_inv.

16.7.3 OCR Features

HALCON offers many different features for the OCR. Most of these are for advanced use only. In most cases it is recommended to use the feature combination 'default'. This combination is based on the gray values within the surrounding rectangle of the character. In case that the background of the characters cannot be used, e.g., if it varies because of texture, the features 'pixel_binary', 'ratio', and 'anisometry' are good combinations. Here, only the region is used, the underlying gray values are ignored.

16.8 Pretrained OCR Fonts

The following sections shortly introduce you to the pretrained OCR fonts provided by HALCON. You can access them in the subdirectory ocr of the folder where you installed HALCON. Note that the

pretrained fonts were trained with symbols that are printed dark on light. If you want to read light on dark symbols with one of the provided fonts, you can either invert the image with the operator invert_image, or, if this does not lead to a satisfying result, preprocess the image by applying first the operator gen_image_proto with a light gray value and then overpaint_region with the gray value set to 0.

Note that pretrained fonts are not available for the classifier based on SVM.

16.8.1 Nomenclature for the Ready-to-Use OCR Fonts

There are several groups of OCR fonts available. The members of each group differ as they contain different symbol sets. The content of an OCR font is described by its name. For the names of the pretrained OCR fonts the following nomenclature is applied:

The name starts with the group name, e.g., Document or DotPrint, followed by indicators for the set of symbols contained in the OCR font. The meaning of the indicators is the following:

- 0-9: The OCR font contains the digits 0 to 9.
- A-Z: The OCR font contains the uppercase characters A to Z.
- \+ : The OCR font contains special characters. The list of special characters varies slightly over the individual OCR fonts. It is given below for each OCR font separately.

If the name of the OCR font does not contain any of the above indicators, typically, the OCR font contains the digits 0 to 9, the uppercase characters A to Z, the lowercase characters a to z, and special characters. Some of the OCR fonts do not contain lowercase characters or the full set of uppercase characters (e.g., MICR). For these OCR fonts, the contained symbols are named explicitly.

16.8.2 Ready-to-Use OCR Font 'Document'

The OCR font Document can be used to read characters printed in fonts like Arial, Courier, or Times New Roman. These are typical fonts for printing documents or letters.

Available special characters: - = + < > . # $ % & ( ) @ * , § €

The following OCR fonts with different symbol sets are available:

- Document
- Document_0-9
- Document_0-9A-Z

16.8.3 Ready-to-Use OCR Font 'DotPrint'

The OCR font DotPrint can be used to read characters printed with dot printers (see figure 16.12).

It contains no lowercase characters.

Available special characters: - / . * , :

The following OCR fonts with different symbol sets are available:

- DotPrint
- DotPrint_0-9
- DotPrint_0-9+
- DotPrint_0-9A-Z

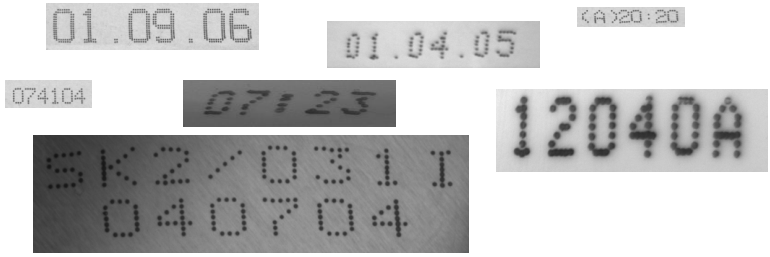


Figure 16.12: Examples for dot prints.

16.8.4 Ready-to-Use OCR Font 'HandWritten_0-9'

The OCR font HandWritten_0-9 can be used to read handwritten numbers (see figure 16.13).

It contains the digits 0-9.

Available special characters: none

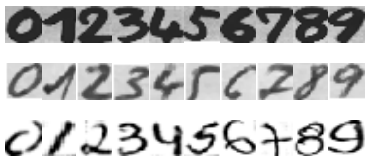


Figure 16.13: Examples for handwritten numbers.

16.8.5 Ready-to-Use OCR Font 'Industrial'

The OCR font Industrial can be used to read characters printed in fonts like Arial, OCR-B, or other sans-serif fonts (see figure 16.14). These fonts are typically used to print, e.g., labels.

Available special characters: - / + . $ % * , €

The following OCR fonts with different symbol sets are available:

- Industrial
- Industrial_0-9

- Industrial_0-9+
- Industrial_0-9A-Z

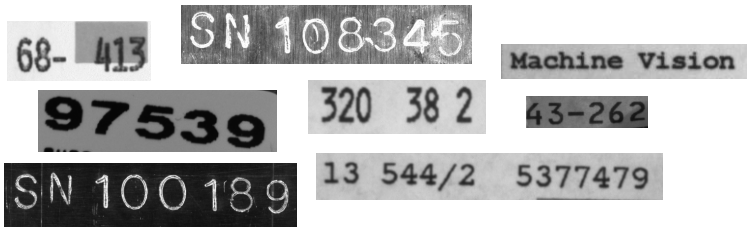


Figure 16.14: Examples for industrial prints.

16.8.6 Ready-to-Use OCR Font 'MICR'

The OCR font MICR can be used to read characters printed in the font MICR (see figure 16.15).

It contains the digits 0-9 and the characters A-D.

Available special characters: none

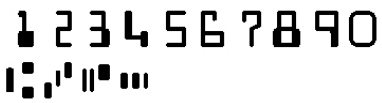


Figure 16.15: The MICR font.

16.8.7 Ready-to-Use OCR Font 'OCR-A'

The OCR font OCR-A can be used to read characters printed in the font OCR-A (see figure 16.16).

Available special characters: - ? ! / \{ } = + < > . # $ % & ( ) @ *

The following OCR fonts with different symbol sets are available:

- OCRA
- OCRA_0-9
- OCRA_0-9A-Z

Figure 16.16: The OCR-A font.

16.8.8 Ready-to-Use OCR Font 'OCR-B'

The OCR font `OCR-B` can be used to read characters printed in the font OCR-B (see figure 16.17).

Available special characters: - ? ! / \{ } = + < > . # $ % & ( ) @ *

The following OCR fonts with different symbol sets are available:

- `OCRB`
- `OCRB_0-9`
- `OCRB_0-9A-Z`

Figure 16.17: The OCR-B font.

16.8.9 Ready-to-Use OCR Font 'Pharma'

The OCR font `Pharma` can be used to read characters printed in fonts like Arial, OCR-B, and other fonts that are typically used in the pharmaceutical industry (see figure 16.18).

This OCR font contains no lowercase characters.

Available special characters: - / . ( ) :

The following OCR fonts with different symbol sets are available:

- `Pharma`
- `Pharma_0-9`
- `Pharma_0-9A-Z`

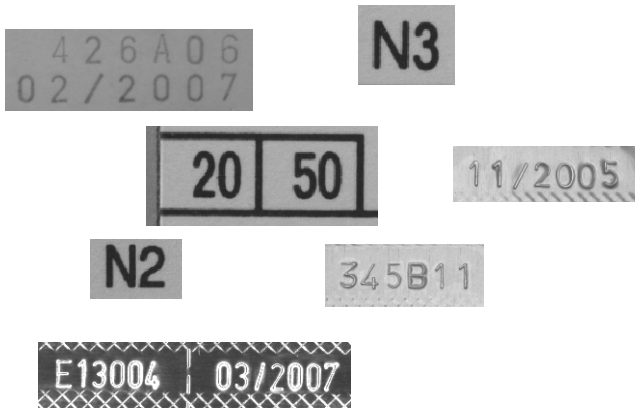


Figure 16.18: Examples for pharmaceutic labels.

Chapter 17

Stereo

The basic principle of binocular stereo is that 3D coordinates of object points are determined from two images that are acquired simultaneously from different points of view. For example, the top row of figure 17.1 shows the stereo image pair of a board. With binocular stereo, the heights of the individual components of the board can be determined. The bottom row of figure 17.1 contains the height map for the entire overlapping area of the stereo image pair. In addition, a part of the height map is shown as a 3D plot.

The advantage of binocular stereo is that 3D information of the surface of arbitrarily shaped objects can be determined from images. Binocular stereo can also be combined with other vision methods, e.g., as a preprocessing step for blob analysis, which can then be used to extract objects from the depth image.

Note that this section describes the fully calibrated version of stereo vision. HALCON also provides an uncalibrated version. Please refer to the Solution Guide II-F, section 7.5 on page 111, for more information.

17.1 Basic Concept

The derivation of 3D information with a binocular stereo system consists of four main steps:

17.1.1 Calibrate Stereo Camera System

First, the stereo camera system is calibrated. For this, a number of stereo calibration images must be acquired, each showing the HALCON calibration plate in a different position and orientation. The object coordinates of the calibration marks can be read from the calibration plate description file. The image coordinates of the calibration marks must be extracted from the calibration images. Then, the parameters of the stereo setup are determined. With this, the rectification maps for the rectification of the stereo images can be determined.

The calibration process is described in detail in the Solution Guide II-F in section 7.3 on page 97.

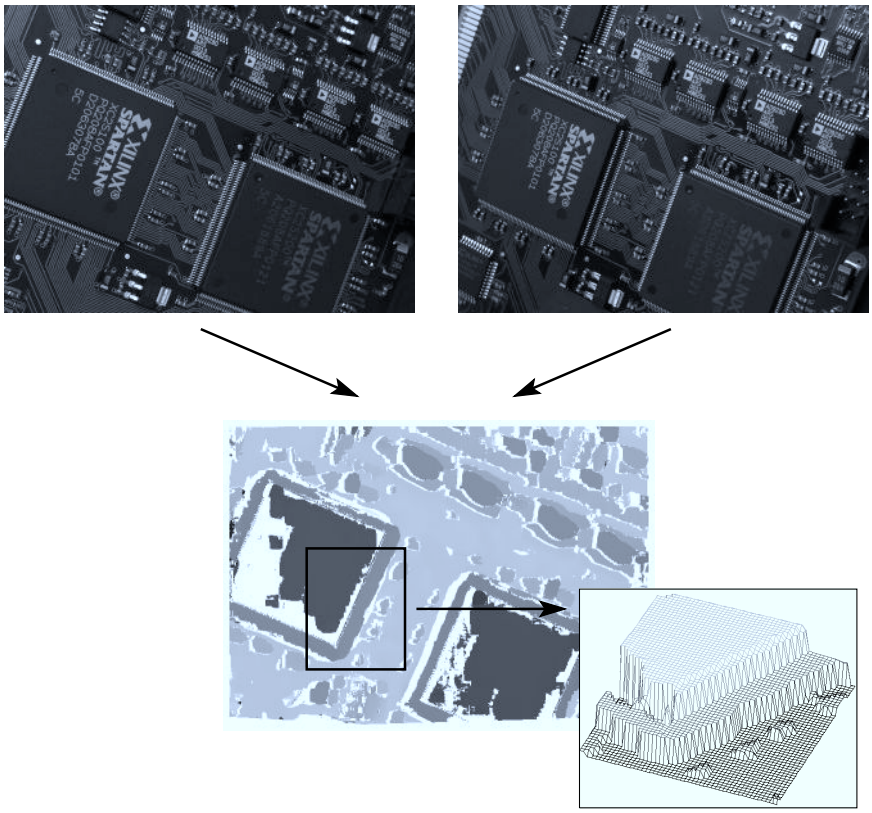
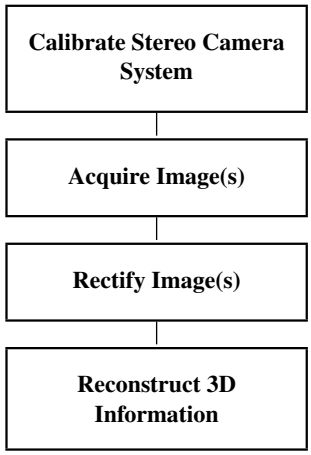


Figure 17.1: Basic principle of binocular stereo. Top: stereo image pair; bottom: height map.



17.1.2 Acquire Image(s)

Stereo images are acquired with the calibrated stereo camera system.

17.1.3 Rectify Image(s)

Having obtained a pair of stereo images, they must be rectified such that corresponding points (conjugate points) lie on the same row in both rectified images. For this, the rectification map that has been determined above must be used.

17.1.4 Reconstruct 3D Information

In the final step, for each point of the first rectified image the conjugate point in the second rectified image will be determined (stereo matching). For these points, either the disparity or the distance to the stereo camera system can be calculated and returned as a gray value image. The reference plane to which these distances are related can be changed (see the Solution Guide II-F, section 7.4.3 on page 106). It is also possible to derive the distance or the 3D coordinates for selected points for which the disparity is known. What is more, 3D coordinates can be determined from the image coordinates of each pair of conjugate points directly.

17.2 Extended Concept

In many cases the derivation of 3D information with a binocular stereo system will involve more steps than described above. Reasons for this are, e.g., the need to restrict the stereo reconstruction to an ROI. Furthermore, postprocessing, like transforming the 3D coordinates into another coordinate system or visualization of results, is often required.

17.2.1 Create ROI

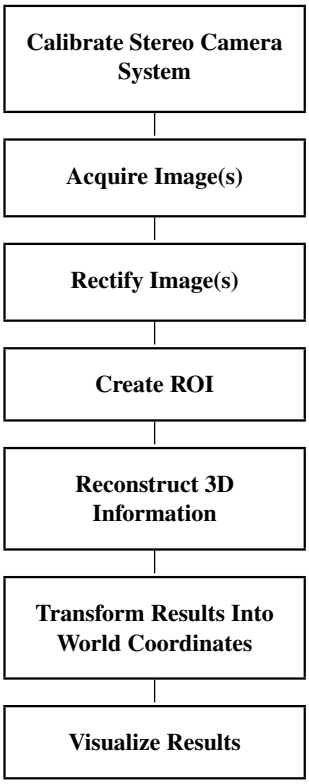
A region of interest can be created to reduce the image domain for which the stereo matching will be performed. This will reduce the processing time.

For detailed information see the description of this method on page 19.

17.2.2 Transform Results Into World Coordinates

In some applications, the 3D coordinates must be transformed into a given world coordinate system. For this, the relation between the given world coordinate system and the stereo camera system must be determined. Then, the 3D coordinates can be transformed as requested.

How to transform results into world coordinates is described in detail in the Solution Guide II-F in section 3.2 on page 44.



17.2.3 Visualize Results

Finally, you might want to visualize the disparity or distance images, e.g., as a 3D plot or as contour lines, or you may need to visualize the 3D coordinates.

For detailed information see the description of this method on page 269.

17.3 Programming Examples

This section gives a brief introduction to using HALCON for binocular stereo.

17.3.1 Segment the Components of a Board

Example: `examples\hdevelop\Applications\Stereo\board_components.dev`

Figure 17.2 shows a stereo image pair of a board together with the result of a segmentation of raised objects. The segmentation has been carried out based on the distance image that was derived with binocular stereo.

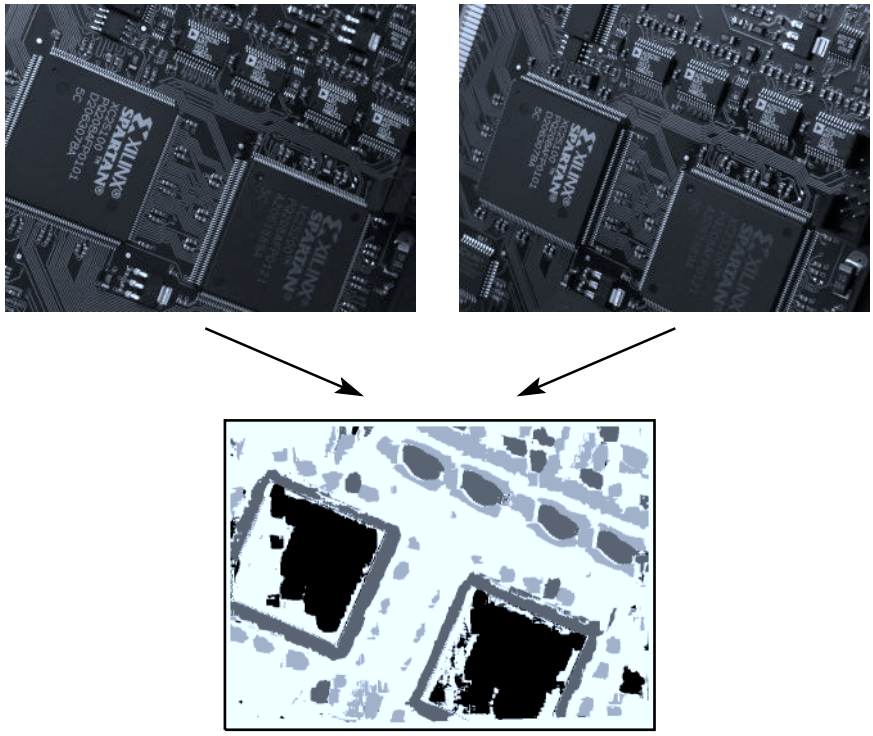


Figure 17.2: Segment board components based on their height.

First, the stereo camera system must be calibrated. For this, a number of calibration image pairs must be acquired. The calibration plate must be completely visible in each calibration image. A subset of these calibration images is shown in figure 17.3.

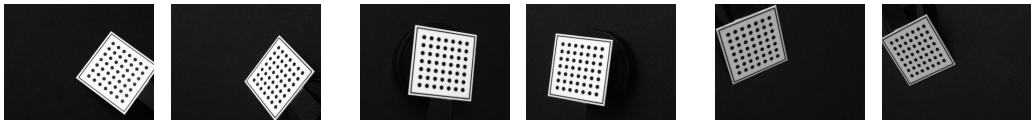


Figure 17.3: A subset of the calibration images that are used for the calibration of the stereo camera system.

The approximate position of the calibration plate is determined and the image coordinates of the calibration marks are extracted from the individual calibration images. The coordinates and the initial values for the poses of the calibration plates must be accumulated.

```
for imgnr := 1 to Number by 1
  find_caltab (ImageL, CaltabL, CalDescrFile, SizeGauss, MarkThresh,
               MinDiamMarks)
  find_caltab (ImageR, CaltabR, CalDescrFile, SizeGauss, MarkThresh,
               MinDiamMarks)
  find_marks_and_pose (ImageL, CaltabL, CalDescrFile, StartCamParL,
                       StartThresh, DeltaThresh, MinThresh, Alpha,
                       MinContLength, MaxDiamMarks, RCoordL, CCoordL,
                       StartPoseL)
  find_marks_and_pose (ImageR, CaltabR, CalDescrFile, StartCamParR,
                       StartThresh, DeltaThresh, MinThresh, 0.7,
                       MinContLength, MaxDiamMarks, RCoordR, CCoordR,
                       StartPoseR)
  RowsL := [RowsL,RCoordL]
  ColsL := [ColsL,CCoordL]
  StartPosesL := [StartPosesL,StartPoseL]
  RowsR := [RowsR,RCoordR]
  ColsR := [ColsR,CCoordR]
  StartPosesR := [StartPosesR,StartPoseR]
endfor
```

The object coordinates of the calibration marks can be read from the calibration plate description file.

```
caltab_points (CalDescrFile, X, Y, Z)
```

With this, the actual calibration of the stereo camera system can be performed.

```
binocular_calibration (X, Y, Z, RowsL, ColsL, RowsR, ColsR,
                       StartCamParL, StartCamParR, StartPosesL,
                       StartPosesR, 'all', CamParamL, CamParamR,
                       NFinalPoseL, NFinalPoseR, cLPcR, Errors)
```

Now, the rectification maps for the rectification of the stereo image pair can be generated.

```
gen_binocular_rectification_map (MapL, MapR, CamParamL, CamParamR, cLPcR,
                                 1, 'geometric', 'bilinear', RectCamParL,
                                 RectCamParR, CamPoseRectL, CamPoseRectR,
                                 RectLPosRectR)
```

Then, each stereo image pair acquired with the calibrated stereo camera system can be rectified. This has the effect that conjugate points have the same row coordinate in both images. The rectified images are displayed in figure 17.4

```
map_image (ImageL, MapL, ImageRectifiedL)
map_image (ImageR, MapR, ImageRectifiedR)
```

From the rectified images, a distance image can be derived in which the gray values represent the distance of the respective object point to the stereo camera system. This step is the core of the stereo approach. Here, the stereo matching, i.e., the determination of the conjugate points takes place.

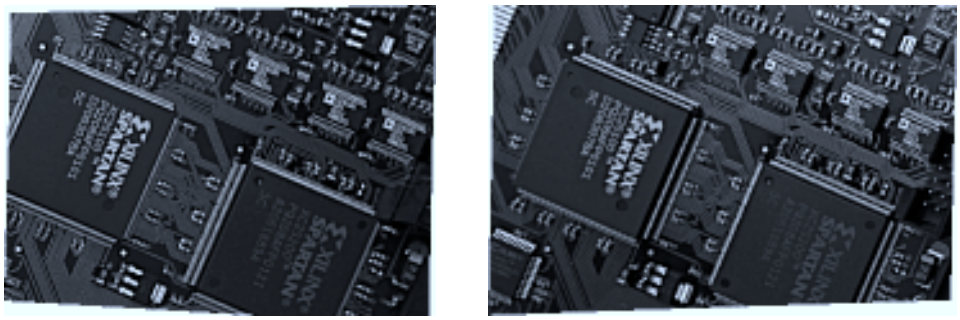


Figure 17.4: Rectified images.

```
binocular_distance (ImageRectifiedL, ImageRectifiedR, DistanceImage,
                    ScoreImageDistance, RectCamParL, RectCamParR,
                    RectLPosRectR, 'ncc', MaskWidth, MaskHeight,
                    TextureThresh, MinDisparity, MaxDisparity, NumLevels,
                    ScoreThresh, 'left_right_check', 'interpolation')
```

Finally, the distance image can, e.g., be corrected such that a given object plane receives a specified distance value, e.g., zero. Objects that deviate from the given object plane can thus be segmented very easily with a threshold operation.

```
  threshold (HeightAboveReferencePlaneReduced, Range1, Height1_Min,
             Height1_Max)
  threshold (HeightAboveReferencePlaneReduced, Range2, Height2_Min,
             Height2_Max)
  threshold (HeightAboveReferencePlaneReduced, Range3, Height3_Min,
             Height3_Max)
```

17.3.2 Other Examples

HDevelop

- examples\hdevelop\Applications\Stereo\board_segmentation_uncalib.dev
  Extract board components by height using uncalibrated binocular stereo
- examples\hdevelop\Applications\Stereo\disparity.dev
  Display disparity image for different stereo image pairs
- examples\hdevelop\Filter\Inpainting\harmonic_interpolation.dev
  Fill up unreconstructed areas in a distance image created by stereo reconstruction
- examples\hdevelop\Filter\Inpainting\inpainting_aniso.dev
  Fill up unreconstructed areas in a distance image created by stereo reconstruction

- examples\hdevelop\Filter\Inpainting\inpainting_ced.dev
  Fill up unreconstructed areas in a distance image created by stereo reconstruction
- examples\hdevelop\Filter\Inpainting\inpainting_mcf.dev
  Fill up unreconstructed areas in a distance image created by stereo reconstruction
- examples\hdevelop\Tools\Stereo\binocular_calibration.dev
  Calibrate stereo systems and rectify image pairs
- examples\hdevelop\Tools\Stereo\binocular_disparity.dev
  Calculate disparities from epipolar image pairs
- examples\hdevelop\Tools\Stereo\binocular_disparity_segmentation.dev
  Demonstrate stereo results using an artificial image pair
- examples\hdevelop\Tools\Stereo\disparity_to_point_3d.dev
  Reconstruct 3D information from disparity
- examples\hdevelop\Tools\Stereo\intersect_lines_of_sight.dev
  Reconstruct 3D information by intersecting lines of sight
- examples\hdevelop\Tools\Stereo\uncalib_stereo_boxes.dev
  Determine the surfaces of boxes using uncalibrated binocular stereo.
- examples\solution_guide\3d_machine_vision\3d_information_for_selected_points.dev
  Calculate world coordinates for a point in a stereo image pair
  → description in the Solution Guide II-F on page 109
- examples\solution_guide\3d_machine_vision\height_above_reference_plane_from_stereo.dev
  Extract chips using height information from binocular stereo
  → description in the Solution Guide II-F on page 103
- examples\solution_guide\3d_machine_vision\stereo_calibration.dev
  Calibrate a stereo system
  → description in the Solution Guide II-F on page 98

17.4 Selecting Operators

17.4.1 Calibrate Stereo Camera System

Standard:

caltab_points, find_caltab, find_marks_and_pose, binocular_calibration, gen_binocular_rectification_map

17.4.2 Acquire Image(s)

Please refer to the operator list for the method Image Acquisition (see section 2.4 on page 17).

17.4.3 Rectify Image(s)

Standard:

map_image

17.4.4 Create ROI

Standard:

reduce_domain

Further operators can be found in the operator list for the method Region Of Interest (see section 3.4 on page 30).

17.4.5 Reconstruct 3D Information

Standard:

binocular_disparity, binocular_distance, disparity_to_distance, disparity_to_point_3d, distance_to_disparity, intersect_lines_of_sight

17.4.6 Transform Results Into World Coordinates

Operators for transforming results into world coordinates are described in the Solution Guide II-F.

17.4.7 Visualize Results

Please refer to the operator list for the method Visualization (see section 18.4 on page 283).

17.5 Relation to Other Methods

17.5.1 Methods that are Using Stereo

Blob Analysis (see description on page 33)
The results of stereo can be used as the input for blob analysis. This may be useful if locally high structures must be extracted that cannot be reliably detected from the original image. By applying blob analysis to the distance image, such structures may be extracted very easily.

17.6 Tips & Tricks

17.6.1 Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for stereo. With domains, the processing can be focused to a certain area in the image and thus sped up. For example, you can restrict the computation of 3D information to a certain region. For an overview on how to construct regions of interest and how to combine them with the image, see the method Region Of Interest on page 19.

17.6.2 Speed Up

Many online applications require maximum speed. Although the stereo matching is a very complex task, HALCON offers ways to speed up this process.

- Regions of interest are the standard method to increase the speed by processing only those areas where objects must be inspected. This can be done by using pre-defined regions but also by an online generation of the region of interest that depends on other objects found in the image.
- The use of image pyramids for the stereo matching reduces the processing time. The matching is started on the specified level of the image pyramid. The respective results are used as initial values for the matching on the next lower level of the image pyramid.

17.7 Advanced Topics

17.7.1 High Accuracy

Sometimes very high accuracy is required. To achieve a high distance resolution, i.e., a high accuracy with which the distance of the object surface from the stereo camera system can be determined, special care should be taken of the configuration of the stereo camera setup. The setup should be chosen such that the distance between the cameras as well as the focal length are large, and that the stereo camera system is placed as close as possible to the object. For more information, please refer to the Solution Guide II-F, section 7.2 on page 95.

Chapter 18

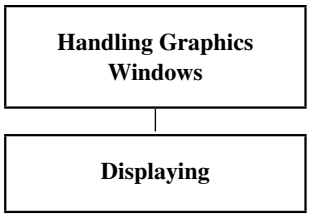
Visualization

Displaying data in HALCON is quite easy: In the graphics windows provided by HALCON, all supported data types can be visualized directly using specific display operators. Both the creation of these windows and the displaying requires only little programming effort because the functionality is optimized for the use in machine vision.

Making use of the HALCON visualization provides several advantages. First of all, it saves a lot of time during the development because all important visualization methods are already predefined. Furthermore, the functionality is independent of the operating system: Writing a program under Windows using the visualization of HALCON can be ported easily to Linux because the visualization operators behave identically and therefore only user code making use of operating system functions needs to be rewritten.

18.1 Basic Concept

For visualization there are two important aspects to consider: The graphics windows and the data that must be visualized.



18.1.1 Handling Graphics Windows

HALCON provides an easy-to-use operator to create a graphics window for visualization: `open_window`. This operator takes all necessary parameters to specify the dimensions, the mode, and the

relation to a potential parent window. As a result, a `WindowHandle` is returned with which you refer to the window when displaying into it or when visualization parameters are changed. Note that the dimension of the window is not limited by the size of the virtual display. Thus, you can work also on systems with multiple screens. The most important operators for controlling a window are `clear_window` to reset it to its background color, `set_part` to specify the display coordinate system and `close_window` when the window is no longer needed.

18.1.2 Displaying

For each HALCON data type, specific operators for displaying are provided. The most convenient operator for iconic data (images, regions, and XLD) is `disp_obj`. This operator automatically handles gray or color images, regions, and XLDs. To control the way how data is presented in the window, operators with the prefix `set_` (or `dev_set_` for the visualization in HDevelop) are used. They allow to control the color, the draw mode, the line width, and many other parameters.

18.1.3 A First Example

An example for this basic concept is the following program, which shows how to visualize an image overlaid with a segmentation result. Here, the visualization operators provided in HDevelop are used.

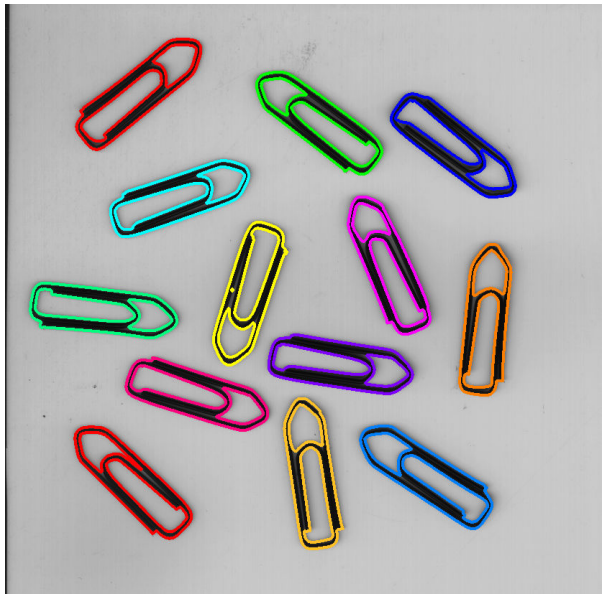


Figure 18.1: Visualizing the segmented clips.

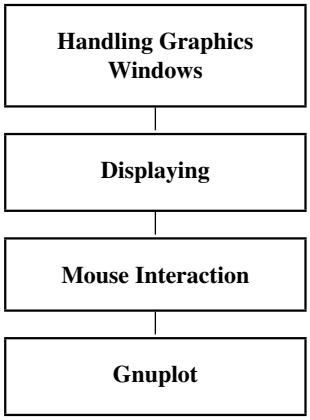
After reading the image from file the dark clips are selected with `bin_threshold`, which automatically selects the threshold value. After determining the connected components and selecting regions with the appropriate size, the result is visualized. First, the image is displayed. After this, the parameters

for regions are set to multi-colors (12) and margin mode. Finally, the regions are displayed with these settings.

```
read_image (Image, 'clip')
bin_threshold (Image, Dark)
connection (Dark, Single)
select_shape (Single, Selected, 'area', 'and', 5000, 10000)
dev_display (Image)
dev_set_colored (12)
dev_set_draw ('margin')
dev_display (Selected)
```

18.2 Extended Concept

In advanced applications it is required to gain complete control over the visualization process. This can include using graphics windows in programs developed with Microsoft Visual Basic or Microsoft C++, changing the behavior of the graphics windows, making use of buffered output, or even using external programs for the visualization. HALCON provides full control over all these topics to provide advanced flexibility, in addition to the ease of use.



18.2.1 Handling Graphics Windows

In this section we consider the concept of graphics windows in more detail.

- The graphics windows are designed such that each one stores all the corresponding parameters in a kind of graphics context. Whenever an operator like `set_draw` is called, this context is modified to be used for this window until the value is overwritten. All operators that modify the graphics context start with the prefix `set_`. The current value of the context can be requested using the corresponding operator with the prefix `get_`, e.g., `get_draw`.

- Besides display parameters, each graphics window has a coordinate system that can be defined with set_part. The upper left corner of an image is (0,0), the lower right corner is (height-1,width-1). The size of an image can be requested using get_image_pointer1. Please note that unlike in HDevelop this coordinate system must be specified by calling set_part. Otherwise, the part of the image that is visualized is undefined.
- The operator open_window has a parameter called Father. By default, the value 0 is used, which means that the window has no parent and floats as an independent instance. You can construct hierarchies of windows by passing the handle of one window as parent to the other. Besides this, it is also possible to use the parent mechanism to embed the graphics windows into other forms (see section 18.6 on page 285).

18.2.2 Displaying

After having opened a graphics window, the returned window handle is used to communicate with it. Typically, first the visualization parameters are specified before data is displayed. To control images only few operators are needed: set_paint (for profiles and 3D plots), set_lut (for look-up-tables), and set_part_style (for the zooming interpolation). To specify the output for regions, many parameters are available. The most important ones are set_draw (for filled or margin mode), set_color (for the pen color), set_line_width (for the pen width), and set_colored (for multi-color modes). To display XLD data, the same parameters (except set_draw) are used.

The visualization itself is performed with operators like disp_image, disp_color, disp_region, or disp_xld. The most convenient way is to use disp_obj, which automatically uses the correct method.

For text output, you first specify the font with set_font. The desired position in the graphics window is determined with set_tposition. Writing text into the window is performed with write_string

For a flicker-free display see section 18.6 on page 285.

18.2.3 Mouse Interaction

The most important thing to know is that HALCON does *not* use an event-driven approach for mouse handling. Each operator is designed such that the mouse interaction starts when the operator is called and finishes when the operator returns. If an event-driven mode is needed one has to use the standard mechanisms provided by the operating system.

Interacting with the mouse mainly involves two tasks:

- The first task is to request the position of the mouse. This can be achieved using get_mposition. This operator returns immediately and has the position and the mouse button as result. An alternative is get_mbutton. This operator only returns when a mouse button has been clicked.
- The second important action is drawing shapes with the mouse. This is done with special operators whose names start with draw_. Operators for many different shapes (like circles or rectangles) are provided. Furthermore, different data types like regions or XLD contours can be used for the result.

18.2.4 Gnuplot

To visualize numeric values, HALCON provides an interface to the public domain software Gnuplot. This tool is especially useful to plot tuples of values like distributions and also provides extended versions of 3D plots of images.

18.3 Programming Examples

This section gives a brief introduction to using the visualization operators provided by HALCON.

18.3.1 Displaying HALCON data structures

Example: examples\solution_guide\basics\display_operators.dev

The example program is designed to show the major features of displaying images, regions, XLD, and text. It consists of a small main program that calls procedures handling the four different data types. The program is written for HDevelop and uses its specific display operators. This is done in a way that naturally ports (e.g., with the automatic export) to other language interfaces like C++, C#, or Visual Basic.

The main procedure contains five procedures: `open_graphics_window`, `display_image`, `display_regions`, `display_xld`, and `display_text`. To switch between the programs of the individual procedures you can use the combo box `Procedures` in the program window. Following, selected parts of each procedure are explained.

```
read_image (Image, 'fabrik')
open_graphics_window (Image, WindowHandle)
display_image (Image, WindowHandle)
regiongrowing (Image, Regions, 1, 1, 3, 100)
display_regions (Image, Regions, WindowHandle)
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 10, 30)
display_xld (Image, Edges, WindowHandle)
display_text (Image, Regions, WindowHandle)
```

`open_graphics_window` is a support procedure to open a graphics window that has the same size as the image. This is done by calling `get_image_pointer1` to access the image dimensions. Before opening the new window, the existing window is closed. To adapt the coordinate system accordingly, `dev_set_part` is called. This would be done automatically in HDevelop, but for the other programming environments this step is necessary. Finally, the default behavior of HDevelop of displaying each result automatically is switched off. This has the effect that only programmed output will be visible.

```
get_image_pointer1 (Image, _, _, Width, Height)
dev_close_window ()
dev_open_window (0, 0, Width, Height, 'white', WindowHandle)
dev_set_part (0, 0, Height-1, Width-1)
dev_update_window ('off')
```

Then, the display procedure for images is called: `display_image`. It has the `Image` and the `WindowHandle` as input parameters. First, the window is activated, which again is not needed for HDevelop, but is important for other programming environments. Now, the image is displayed in the graphics window.

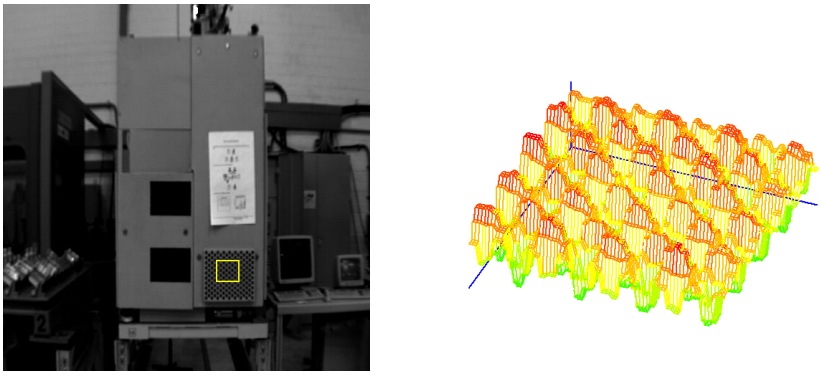
```
dev_set_window (WindowHandle)
dev_display (Image)
```

To change the look-up-table (LUT), `dev_set_lut` is called and the effect will become visible after calling `dev_display` once again.

```
dev_set_lut ('temperature')
dev_display (Image)
```

Next, a part of the image is displayed using a so-called 3D plot (see figure 18.2). Here, the gray values are treated as height information. For this mode another LUT is used.

```
gen_rectangle1 (Rectangle, 358, 298, 387, 329)
dev_set_draw ('margin')
dev_set_color ('yellow')
dev_display (Rectangle)
dev_set_part (358, 298, 387, 329)
dev_set_lut ('twenty_four')
dev_set_paint (['3D-plot_hidden',7,1,110,160,600,0,0])
dev_display (Image)
```



a) b)

Figure 18.2: (a) Original image with ROI; (b) 3D plot of image within the ROI.

The procedure to display regions is called `display_regions`. It first displays the image as background and then sets the display parameters for the regions. `dev_set_draw` specifies that only the region border is visualized. `dev_set_colored` activates the multi-color mode, where each region is displayed with different colors (which change cyclically). As an alternative to simply showing the original shape of the

regions, HALCON enables you to modify the shape using dev_set_shape. In the given example the equivalent ellipses are chosen. The result is depicted in figure 18.3.

```
dev_display (Image)
dev_set_draw ('margin')
dev_set_colored (6)
dev_display (Regions)
dev_display (Image)
dev_set_shape ('ellipse')
dev_display (Regions)
```

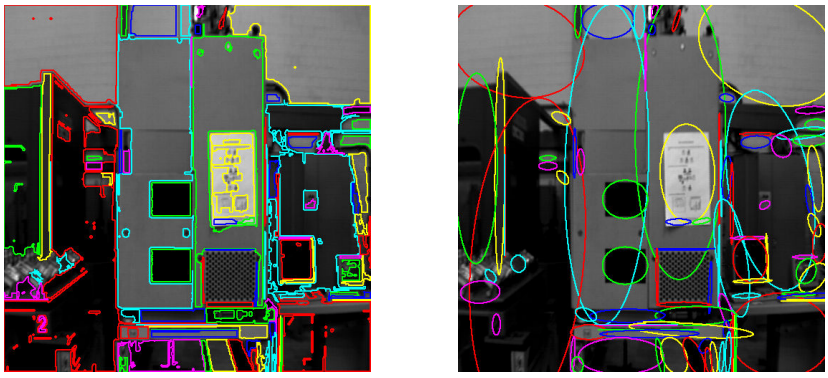
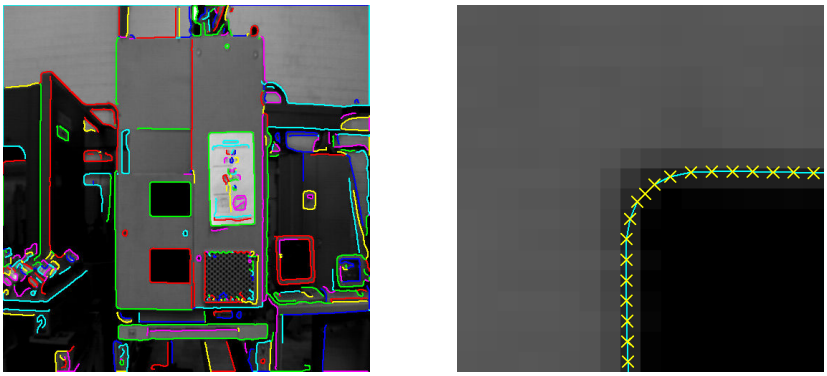


Figure 18.3: (a) Regions and (b) their equivalent ellipses.

The procedure display_xld first shows all contours overlaid on the image using the multi-color mode. Then, a zoom is defined using dev_set_part. This zoom mode allows to inspect the sub-pixel accurate contours easily. To give additional information, contours with a given size are selected and for each of these the control points are extracted using get_contour_xld. The coordinates are here returned as tuples of real values. For each of these control points, a cross is generated using gen_cross_contour_xld, which is then overlaid onto the contour.

```
dev_display (Contours)
gen_rectangle1 (Rectangle, 239, 197, 239+17, 197+17)
dev_set_part (239, 197, 239+17, 197+17)
select_shape_xld (Contours, SelectedXLD, 'area', 'and', 2000, 3000)
count_obj (SelectedXLD, Number)
for k := 1 to Number by 1
  select_obj (SelectedXLD, SingleContour, k)
  get_contour_xld (SingleContour, Row, Col)
  for i := 0 to |Row|-1 by 1
    gen_cross_contour_xld (Cross, Row[i], Col[i], 0.8, rad(45))
    dev_display (Cross)
  endfor
endfor
```



a) b)

Figure 18.4: Subpixel accurate contour control points: (a) original image; (b) zoomed image part.

The last part of the program is a procedure called `display_text`. It shows how to handle the mouse and text output. The task is to click with the mouse into the graphics window to select a specific region and to calculate features, which are then displayed in the graphics window. First, the font is selected. This is the only part of the visualization where the parameters differ between Linux/UNIX and Windows because of the incompatible font name handling. However, as can be seen, the program can still be written so that it ports without problems.

```
OpSystem := environment('OS')
if (OpSystem='Windows_NT')
  set_font (WindowHandle, '-Courier New-16-*-*-*-*-*-')
else
  set_font (WindowHandle,
            '-*-courier-bold-r-normal--16-*-*-*-*-*-iso8859-1')
endif
```

The rest of the program consists of a `while`-loop, which terminates as soon as the right mouse button is pressed. The mouse operator `get_mbutton` waits until the user clicks with the mouse into the graphics window and then returns the coordinate and the button value. This coordinate is used to select the region that contains this point using `select_region_point`. For this region, the size and the center of gravity are calculated with `area_center`. First, the text cursor is positioned with `set_tposition` and the values are displayed using `write_string`. Here, it can be seen how conveniently strings can be composed using the `"+"` operator.

```
Button := 0
while (Button #  4)
  get_mbutton (WindowHandle, Row, Column, Button)
  select_region_point (Regions, DestRegions, Row, Column)
  area_center (DestRegions, Area, RowCenter, ColumnCenter)
  if (|Area| > 0)
    set_tposition (WindowHandle, Row, Column)
    dev_set_color ('yellow')
    write_string (WindowHandle, '('+RowCenter+',
                  '+ColumnCenter+') = '+Area)
  endif
endwhile
```

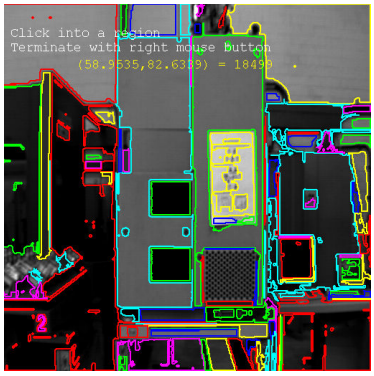


Figure 18.5: Center of gravity and area of selected region.

18.3.2 Other Examples

HDevelop

- examples\hdevelop\Applications\Aerial\dem.dev
  Extract high objects from a digital elevation model
- examples\hdevelop\Applications\Aerial\roads.dev
  Extract roads from aerial image
  → description here in the Solution Guide Basics on page 106
- examples\hdevelop\Applications\Barcode\get_rectangle_pose_barcode.dev
  Estimate 3D pose of bar codes
- examples\hdevelop\Applications\Calibration\3d_position_of_rectangle.dev
  Estimate 3D pose of rectangular objects
- examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev
  Measure distances between the pitch lines of a caliper rule in a line scan image using camera calibration

- examples\hdevelop\Applications\FA\3d_matching_clamps.dev
  Recognize 3D objects in images using a 3D DXF model
  → description here in the Solution Guide Basics on page 151
- examples\hdevelop\Applications\FA\ball.dev
  Inspect ball bondings
  → description here in the Solution Guide Basics on page 43
- examples\hdevelop\Applications\FA\cbm_bin_switch.dev
  Locate a switch and test its state using component-based matching
- examples\hdevelop\Applications\FA\cbm_caliper.dev
  Measure the setting of a caliper using component-based matching in a perspectively distorted image
- examples\hdevelop\Applications\FA\cbm_dip_switch.dev
  Locate dip switches and test their state using component-based matching
  → description here in the Solution Guide Basics on page 129
- examples\hdevelop\Applications\FA\cbm_pipe_wrench.dev
  Locate a pipe wrench that consists of two components
  → description here in the Solution Guide Basics on page 130
- examples\hdevelop\Applications\FA\circles.dev
  Fit circles into curved contour segments
  → description here in the Solution Guide Basics on page 25
- examples\hdevelop\Applications\FA\clip.dev
  Determine the position and orientation of clips
- examples\hdevelop\Applications\FA\matching_image_border.dev
  Locate multiple dies even if they extend beyond the image border
- examples\hdevelop\Applications\FA\matching_multi_channel_clamp.dev
  Locate upper clamp from a pile using color shape-based matching
- examples\hdevelop\Applications\FA\matching_multi_channel_yogurt.dev
  Find yogurts of different flavors using color shape-based matching
- examples\hdevelop\Applications\FA\pm_illu_rectify.dev
  Find a rotated pattern using a gray value template and align the image to the pattern
- examples\hdevelop\Applications\FA\pm_measure_board.dev
  Locate IC on a board and measure pin distances
  → description here in the Solution Guide Basics on page 64
- examples\hdevelop\Applications\FA\pm_world_plane.dev
  Recognize planar objects using shape-based matching in perspectively distorted images
- examples\hdevelop\Applications\FA\rim.dev
  Inspect holes and characters on a rim
- examples\hdevelop\Applications\FA\sharpness.dev
  Determine the sharpness of an image using different approaches

- examples\hdevelop\Applications\Measure\measure_arc.dev
  Measure the width of a metal part along a circular arc
  → description here in the Solution Guide Basics on page 62
- examples\hdevelop\Applications\Measure\measure_pin.dev
  Measure pins of an IC
  → description here in the Solution Guide Basics on page 65
- examples\hdevelop\Applications\Medicine\angio.dev
  Extract blood vessels and their diameters from an angiogram
- examples\hdevelop\Applications\Medicine\particle.dev
  Extract particles of varying sizes
  → description here in the Solution Guide Basics on page 39
- examples\hdevelop\Applications\Medicine\vessel.dev
  Extract and measure a blood vessel
- examples\hdevelop\Applications\Monitoring\optical_flow.dev
  Monitor traffic using optical flow
- examples\hdevelop\Applications\Monitoring\optical_flow_bicycle.dev
  Monitor moving bicycle using optical flow
- examples\hdevelop\Applications\OCR\bottle.dev
  Read numbers on a beer bottle
  → description here in the Solution Guide Basics on page 243
- examples\hdevelop\Applications\OCR\engraved.dev
  Read characters on a metal surface
  → description here in the Solution Guide Basics on page 244
- examples\hdevelop\Applications\OCR\font.dev
  Read printed characters with interactive selection
- examples\hdevelop\Applications\OCR\letter.dev
  Read printed characters with interactive selection
- examples\hdevelop\Applications\OCR\ocr_gradient_feature.dev
  Compare OCR features
- examples\hdevelop\Applications\Stereo\board_components.dev
  Extract board components by height using binocular stereo
  → description here in the Solution Guide Basics on page 262
- examples\hdevelop\Applications\Stereo\board_segmentation_uncalib.dev
  Extract board components by height using uncalibrated binocular stereo
- examples\hdevelop\Classification\Support-Vector-Machines\
  get_support_vector_class_svm.dev
  Read out the support vectors from an SVM and visualize them along with the class boundaries
- examples\hdevelop\Develop\dev_clear_window.dev
  Clear a graphics window in HDevelop

- examples\hdevelop\Develop\dev_close_window.dev
  Close graphics windows in HDevelop
- examples\hdevelop\Develop\dev_display.dev
  Display image objects in graphics windows in HDevelop
- examples\hdevelop\Develop\dev_open_window.dev
  Open graphics windows in HDevelop
- examples\hdevelop\Develop\dev_set_paint.dev
  Paint data using different representations into a graphics window in HDevelop
- examples\hdevelop\Develop\dev_set_part.dev
  Set the part of an image to be displayed (zoomed) in a graphics window in HDevelop
- examples\hdevelop\Develop\dev_set_window_extents.dev
  Set size and position of a graphics window in HDevelop
- examples\hdevelop\Filter\Geometric-Transformations\projective_trans_image_reduced.dev
  Apply projective transformations to an image and its domain
- examples\hdevelop\Filter\Lines\lines_color.dev
  Extract lines using color information
  → description here in the Solution Guide Basics on page 201
- examples\hdevelop\Regions\Geometric-Transformations\projective_trans_region.dev
  Apply projective transformations to regions
- examples\hdevelop\Tools\2D-Transformations\gen_projective_mosaic.dev
  Combine several images of a PCB into a large mosaic image
  → description here in the Solution Guide Basics on page 132
- examples\hdevelop\Tools\2D-Transformations\projective_trans_pixel.dev
  Apply a projective transformation to rotate an image in 3D
- examples\hdevelop\Tools\2D-Transformations\projective_trans_point_2d.dev
  Apply a projective transformation to rotate images and points in 3D
- examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev
  Rectify image of stadium to simulate overhead view
- examples\hdevelop\Tools\Calibration\get_circle_pose.dev
  Determine the pose of circles in 3D from their perspective 2D projections
- examples\hdevelop\Tools\Datacode\ecc200_optimized_settings.dev
  Optimize parameters for reading a 2d data code of type ECC200
- examples\hdevelop\Tools\Datacode\pdf417_default_settings.dev
  Read various 2d data codes of type PDF417
- examples\hdevelop\Tools\Datacode\pdf417_optimized_settings.dev
  Optimize parameters for reading various 2d data codes of type PDF417

- examples\hdevelop\Tools\Datacode\qrcode_optimized_settings.dev
  Optimize parameters for reading a 2d data code of type QR Code
- examples\hdevelop\Tools\Function\compose_funct_1d.dev
  Compose two 1D functions and compare the result with the analytical composition result
- examples\hdevelop\Tools\Geometry\distance_pc.dev
  Determine the minimum and maximum radius of drill holes by calculating the distance between a point and a contour
- examples\hdevelop\XLD\Features\statistics_points_xld.dev
  Analyze 2D statistical data using *_points_xld operators
- examples\hdevelop\XLD\Transformation\projective_trans_contour_xld.dev
  Rotate an XLD contour in 3D using hom_mat3d_project and projective_trans_region
- examples\solution_guide\1d_measuring\fuzzy_measure_switch.dev
  Determine the width of and the distance between the pins of a switch with a fuzzy measure object
  → description in the Solution Guide II-D on page 27
- examples\solution_guide\1d_measuring\measure_ic_leads.dev
  Measure leads of an IC
  → description in the Solution Guide II-D on page 12
- examples\solution_guide\1d_measuring\measure_ring.dev
  Determine the width of cogs with a circular measure object
  → description in the Solution Guide II-D on page 23
- examples\solution_guide\1d_measuring\measure_switch.dev
  Determine the width of and the distance between the pins of a switch
  → description in the Solution Guide II-D on page 5
- examples\solution_guide\2d_data_codes\2d_data_codes_manual_settings.dev
  Set parameters manually for reading symbols of type ECC200
  → description in the Solution Guide II-C on page 13
- examples\solution_guide\2d_data_codes\2d_data_codes_training.dev
  Train a 2D data code model for symbols of type ECC200
  → description in the Solution Guide II-C on page 11
- examples\solution_guide\2d_data_codes\write_2d_data_code_model.dev
  Train a 2D data code model and write it into a file
  → description in the Solution Guide II-C on page 21
- examples\solution_guide\2d_measuring\measure_chip.dev
  Get extents and positions of rectangular chip components
  → description in the Solution Guide II-E on page 49
- examples\solution_guide\2d_measuring\measure_circles.dev
  Get radii and positions of circular shapes
  → description in the Solution Guide II-E on page 52
- examples\solution_guide\2d_measuring\measure_grid.dev
  Get junctions of a grid that separates keys

→ description in the Solution Guide II-E on page 42

- examples\solution_guide\2d_measuring\measure_metal_part_extended.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 39
- examples\solution_guide\2d_measuring\measure_metal_part_first_example.dev
  Measure several features of a metal part
  → description in the Solution Guide II-E on page 6
- examples\solution_guide\2d_measuring\measure_metal_part_id.dev
  Inspect metal part for missing or deviating circular shapes
  → description in the Solution Guide II-E on page 64
- examples\solution_guide\2d_measuring\measure_pump.dev
  Measure positions and radii of circular shapes
  → description in the Solution Guide II-E on page 54
- examples\solution_guide\2d_measuring\measure_screw.dev
  Measure several features of a screw
  → description in the Solution Guide II-E on page 31
- examples\solution_guide\3d_machine_vision\bundle_adjusted_mosaicking.dev
  Use bundle-adjusted mosaicking to merge partial images of a BGA into one large image
  → description in the Solution Guide II-F on page 84
- examples\solution_guide\3d_machine_vision\camera_calibration_multi_image.dev
  Calibrate the camera and measures positions on a caliper rule
  → description in the Solution Guide II-F on page 39
- examples\solution_guide\3d_machine_vision\handeye_movingcam_calibration.dev
  Perform hand-eye calibration for a hand-eye system with a moving camera
  → description in the Solution Guide II-F on page 115
- examples\solution_guide\3d_machine_vision\handeye_stationarycam_calibration.dev
  Perform hand-eye calibration for a hand-eye system with a stationary camera
  → description in the Solution Guide II-F on page 115
- examples\solution_guide\3d_machine_vision\handeye_stationarycam_grasp_nut.dev
  Calculate pose for grasping a nut based on results of hand-eye calibration for a stationary camera
  → description in the Solution Guide II-F on page 120
- examples\solution_guide\3d_machine_vision\height_above_reference_plane_from_stereo.dev
  Extract chips using height information from binocular stereo
  → description in the Solution Guide II-F on page 103
- examples\solution_guide\3d_machine_vision\mosaicking.dev
  Use mosaicking to merge partial images of a BGA into one large image
  → description in the Solution Guide II-F on page 72
- examples\solution_guide\basics\critical_points.dev
  Locate saddle point markers in an image
  → description here in the Solution Guide Basics on page 23

- examples\solution_guide\shape_matching\multiple_scales.dev
  Search for nuts of different sizes
  → description in the Solution Guide II-B on page 44

C++

- examples\mfc\Matching\Matching.cpp
  Locate an IC using HALCON/C++ and MFC, creating a HALCON window
- examples\mfc\MatchingCOM\Matching.cpp
  Locate an IC using HALCON/COM and MFC
- examples\mfc\MatchingExtWin\Matching.cpp
  Locate an IC using HALCON/C++ and MFC, paint into an existing window
- examples\motif\Matching\source\matching.cpp
  Locate an IC using HALCON/C++ and Motif
- examples\qt\Matching\matching.cpp
  Locate an IC using HALCON/C++ and Qt

C#

- examples\c#\Matching\vs.net\Matching.csproj
  Locate an IC on a board and measure pin distances

Visual Basic .NET

- examples\vb.net\Matching\vs.net\Matching.vbproj
  Locate an IC on a board and measures pin distances

Delphi

- examples\delphi\Matching\matching.dpr
  Locate an IC on a board and measure pin distances

18.4 Selecting Operators

18.4.1 Handling Graphics Windows

Standard:

open_window, clear_window, close_window

18.4.2 Displaying

Standard:

set_paint, set_lut, set_part_style, set_draw, set_line_width, set_color, set_colored, disp_obj, set_font, set_tposition, write_string

Advanced:

disp_image, disp_color, disp_region, disp_xld

18.4.3 Mouse Interaction

Standard:

get_mposition, get_mbutton, draw_region, draw_circle

18.4.4 Gnuplot

Advanced:

gnuplot_open_file, gnuplot_open_pipe, gnuplot_plot_ctrl, gnuplot_plot_image

18.5 Tips & Tricks

18.5.1 Saving Window Content

HALCON provides an easy way to save the content of a graphics window to a file. This can, e.g., be useful for documentation purposes. The corresponding operator is called dump_window. It takes the window handle and the file name as input. The parameter Device allows to select amongst different file formats.

18.5.2 Execution Time

Generally, visualization takes time. To reduce time, it is recommended to visualize only when it is really needed for the specific task. When using HDevEngine (see the Programmer's Guide in part VI on page 161), by default the execution of all dev_* operators is supressed. Thus, you can use as much display operators as you need while developing with HDevelop but save time when executing the final application from a programming language.

Further influences on the run time concern the graphics card and the bit depth. Choosing a bit depth of 16 instead of 32 in many cases speeds up the program execution significantly. So, if speed is important for you, we recommend to simply try different bit depths, perform typical display operators

like disp_image, disp_color, disp_region, or disp_xld, and measure the execution time with count_seconds. Additionally, although it has generally only a small influence on the run time, cases exist where you can speed up the visualization also by choosing a lower screen resolution.

18.6 Advanced Topics

18.6.1 Programming Environments

The handling of graphics windows differs in the different programming environments. Especially HDevelop has a specific way of working with the visualization.

- Because HDevelop is an interactive environment, the handling of windows must be as easy as possible. In particular, it is important to display data without any need of programming. Therefore, the concept of window handles is used only if needed. Normally, the window where the output has to go to is not specified explicitly. Instead, HDevelop makes use of the activation status of the graphics windows. As a consequence, the display operators of HDevelop do not have a parameter for a window handle. Visualization operators in HDevelop look identical to their counterparts of HALCON except that their name starts with dev_ and that the parameter for the window handle is suppressed. When exporting the code, this missing handle will automatically be inserted.

  The second difference is that the windows in HDevelop have a history to automatically redisplay the data when the window has been resized. This is not the case with standard HALCON graphics windows.

  The last difference is that HDevelop automatically sets the coordinate system according to the current image, whereas you must do this explicitly with programmed code when using HALCON. To make an export to, e.g., C++, C#, or Visual Basic transparent, we recommend to use dev_set_window when working with multiple graphics windows and to call dev_set_part to specify the coordinate system.

- When using HALCON windows in MFC, the usual way is to use the windows as a subwindow of a parent form. This can easily be achieved by using the window handle of the current form as the father. The handle must be converted to a long value that can then be passed to open_window. Note that set_check is needed to change to exception handling of HALCON in this context.

  set_window_attr("border_width",0); set_check(" father"); long lWWindowID = (long)m_hWnd; open_window(0,0,640,480,lWWindowID,"visible","",&m_lWindowID); set_check("father");

- Opening a HALCON window in Visual Basic is similar to the approach used for MFC. Here, you use the memberhWnd of the form or of another subwindow like a picture box. As an alternative, the HWindowXCtrl can be used.

  Call sys.SetCheck(" father") Call op.OpenWindow(8, 8, 320, 240, form.hWnd, "", "", WindowHandle) Call sys.SetCheck("father")

18.6.2 Flicker-Free Display

For a flicker-free visualization, a sequential call of display operators is not suitable, because after each call the new data will immediately be flushed on the visible screen, which may cause flickering results.

Under Linux/UNIX or if you want to create a platform-independent application, this can be fixed by using a buffer window in addition to the visible window. A buffer window is opened with open_window, while setting Mode to 'buffer'. Then, all drawing operations are performed into the buffer window. Finally, the buffer window contents are copied into the visible window using copy_rectangle. An example using a buffer window is provided under examples\hdevelop\Tools\2D-Transformations\projective_trans_pixel.dev.

When working exclusively with Windows, a faster approach is available. Here, you can call set_system with the parameter value 'flush_graphic' set to 'false', perform all display operators - except the last one - and then call set_system again with 'flush_graphic' set to 'true'. When finally applying the last display call of the display sequence, the whole data becomes visible in one step. An example using this approach is provided under examples\solution_guide\basics\median_interactive.dev.

18.6.3 Remote Visualization

In some applications, the computer used for processing differs from the computer used for visualization. Such applications can easily be created with HALCON using the socket communication. Operators like send_image or receive_tuple allow a transparent transfer of the relevant data to the control computer to apply visualization there.

18.6.4 Programmed Visualization

Sometimes it might be necessary not to apply the standard HALCON visualization operators, but to use a self-programmed version. This can be achieved by using the access functions provided for all data types. Examples for these are get_image_pointer1, get_region_runs, or get_contour_xld. Operators like these allow full access to all internal data types. Furthermore, they provide the data in various forms (e.g., runlength encoding, points, or contours) to make further processing easier. Based on this data, a self programmed visualization can be developed easily.

As an alternative, with set_window_type the window type 'pixmap' can be chosen. In this case, all displayed data is painted into an internal buffer that can be accessed with get_window_pointer3. The returned pointers reference the three color channels of the buffer. This buffer can then easily be transferred (e.g., to another system) and/or transformed into the desired format. One example for a conversion is to call gen_image3 to create a HALCON color image.