HALCON

a product of MVTec

Solution Guide III-A

1D Measuring



HALCON 21.05 *Progress*

MVTec Software GmbH

Building Vision for Business

How to measure along lines or arcs with high accuracy, Version 21.05.0.0

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

Copyright © 2005-2021 by MVTec Software GmbH, München, Germany



Protected by the following patents: US 7,062,093, US 7,239,929, US 7,751,625, US 7,953,290, US 7,953,291, US 8,260,059, US 8,379,014, US 8,830,229. Further patents pending.

Microsoft, Windows, Windows Server 2008/2012/2012 R2/2016, Windows 7/8/8.1/10, Microsoft .NET, Visual C++, and Visual Basic are either trademarks or registered trademarks of Microsoft Corporation.

All other nationally and internationally recognized trademarks and tradenames are hereby recognized.

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

About This Manual

With HALCON, you can perform highly accurate measurements in 1D (i.e., measure distances and angles along lines and arcs), 2D (i.e., measure features like the size and orientation of objects), and 3D (i.e., measure in 3D world coordinates). This Solution Guide describes 1D measuring, a fast and easy-to-use tool that enables you to measure the dimensions of objects with high accuracy.

A first example and a short characterization of the various methods is given in chapter 1 on page 7. A description of how to create a measure object as well as the basic concepts of 1D measuring can be found in chapter 2 on page 9. Then, the methods to perform 1D measuring are described in detail (chapter 3 on page 15).

For more complex measurement tasks, it might be necessary to control the selection of the edges. To do this, the measure object can be extended to a fuzzy measure object (chapter 4 on page 27) where the selection of the edges is controlled by fuzzy membership functions.

To ease the measuring process, you can also use HDevelop's Measure Assistant, which helps you to perform measurements with just a few mouse clicks. It is described in more detail in the HDevelop User's Guide, section 7.4 on page 249.

The HDevelop example programs that are presented in this Solution Guide can be found in the specified subdirectories of the directory `%HALCONEXAMPLES%`.

Contents

1 First Example and Overview **7**

2 The Basics of Measure Objects **9**
2.1 The Process of 1D Edge Extraction . . . 9
2.2 Creating a Measure Object . . . 11

3 Using the Measure Object **15**
3.1 The Position of Edges . . . 15
3.2 The Position of Edge Pairs . . . 17
3.3 The Position of Points With a Particular Gray Value . . . 19
3.4 The Gray Projection . . . 19
3.5 Measuring in World Coordinates . . . 22
3.6 Compensate for Lens Distortions . . . 23
3.7 Changing the Position and Orientation of the Measure Object . . . 23
3.8 Tips and Tricks . . . 24
3.8.1 Distances for Circular ROIs . . . 24
3.8.2 ROIs That Lie Partially Outside the Image . . . 25
3.8.3 Accuracy vs. Speed . . . 25
3.8.4 Image Type . . . 26

4 Advanced Measuring: The Fuzzy Measure Object **27**
4.1 A First Example . . . 27
4.2 Controlling the Selection of Edges and Edge Pairs . . . 29
4.2.1 Features that Can Be Used to Control the Selection of Edges and Edge Pairs . . . 29
4.2.2 Defining the Constraints on the Feature Values with Fuzzy Membership Functions . . . 30
4.3 Creating the Fuzzy Measure Object . . . 31
4.3.1 Specifying a Fuzzy Membership Function . . . 32
4.3.2 Specifying a Normalized Fuzzy Membership Function . . . 32
4.3.3 Specifying Multiple Fuzzy Membership Functions . . . 35
4.3.4 Changing and Resetting a Fuzzy Measure Object . . . 36
4.4 Using the Standard Measure Object Functionality . . . 36

A Slanted Edges **37**

B Examples for Fuzzy Membership Functions **43**
B.1 Set Type *contrast* . . . 44
B.1.1 Subtype *'contrast'* . . . 44
B.2 Set Type *position* . . . 45
B.2.1 Subtype *'position'* . . . 45
B.2.2 Subtype *'position_center'* . . . 46
B.2.3 Subtype *'position_end'* . . . 47
B.2.4 Subtype *'position_first_edge'* . . . 48
B.2.5 Subtype *'position_last_edge'* . . . 49
B.3 Set Type *position_pair* . . . 50
B.3.1 Subtype *'position_pair'* . . . 50
B.3.2 Subtype *'position_pair_center'* . . . 51
B.3.3 Subtype *'position_pair_end'* . . . 52
B.3.4 Subtype *'position_first_pair'* . . . 53

B.3.5 Subtype *'position_last_pair'* 54
B.4 Set Type *size* 55
B.4.1 Subtype *'size'* 55
B.4.2 Subtype *'size_diff'* 56
B.4.3 Subtype *'size_abs_diff'* 57
B.5 Set Type *gray* 58
B.5.1 Subtype *gray* 58
B.6 Set Type *position_pair* combined with *size* 59

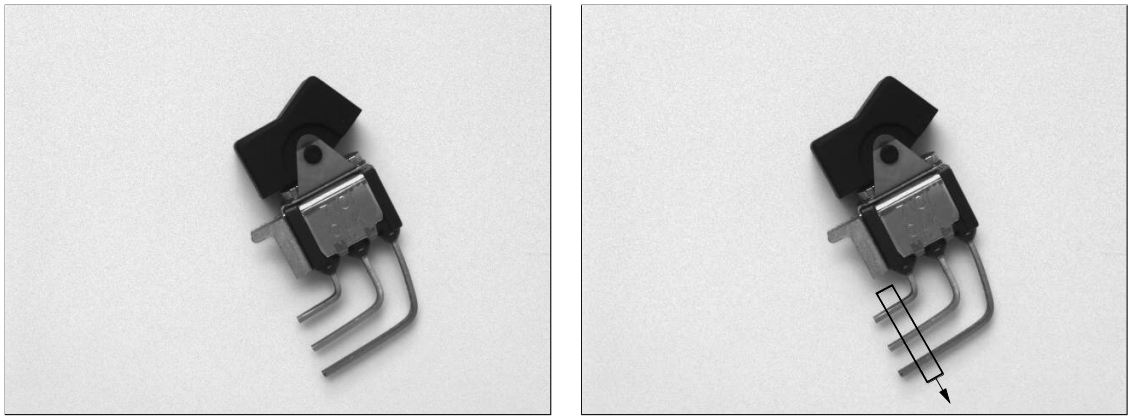
C Definition of the Geometric Mean **61**

Index **63**

Chapter 1

First Example and Overview

The example program %HALCONEXAMPLES%\solution_guide\1d_measuring\measure_switch.hdev is a simple application where the width of and the distance between the pins of a switch (see figure 1.1a) must be measured. This task can easily be solved by 1D measuring because positions and distances are measured along a line.



(a) (b)

Figure 1.1: Image of a switch: (a) The width of the pins and their distance from each other must be measured; (b) the rectangular ROI within which the edges are determined.

First, a measure object is created by specifying a rectangle that lies over the pins as depicted in figure 1.1b. The operator gen_measure_rectangle2 returns a handle to the created measure object.

```
gen_measure_rectangle2 (Row, Column, Phi, Length1, Length2, Width, Height, \
                        Interpolation, MeasureHandle)
```

To perform the measurement, you pass this handle to the measuring operators. To inspect the switch, we use measure_pairs, which extracts the edge pairs corresponding to the pins and returns their width (distance between the edges of a pair, IntraDistance) and their distance (distance between edges of subsequent pairs, InterDistance).

```
measure_pairs (Image, MeasureHandle, Sigma, Threshold, Transition, Select, \
               RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, \
               RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, \
               IntraDistance, InterDistance)
```

The resulting edges are displayed in figure 1.2 together with the measured width of the pins and the distance between them.

The following sections take a closer look at these steps and explain how to solve more complex tasks: Chapter 2 on page 9 describes the basics of measuring, i.e., how the edges are determined (section 2.1 on page 9) and how to create measure objects for rectangular and circular ROIs (section 2.2 on page 11).

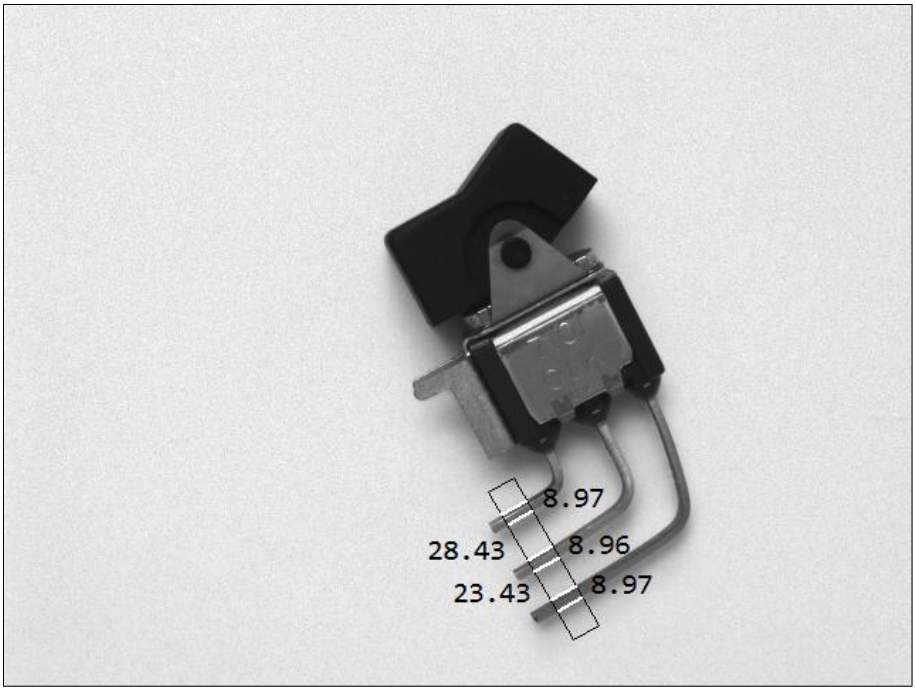


Figure 1.2: The result of measuring the width of and the distance between the pins of a switch.

Chapter 3 on page 15 explains how to use the measure object to extract edges (section 3.1 on page 15), edge pairs (section 3.2 on page 17), points that have a particular gray value (section 3.3 on page 19), and the gray value profile (section 3.4 on page 19).

Often, it is necessary to determine the dimensions of an object in world coordinates, e.g., in μm. The transformation of the results of a measure object into world coordinates is described in section 3.5 on page 22.

Section 3.7 on page 23 describes how to change the position and orientation of measure objects. Some tips and tricks for the use of measure objects are given in section 3.8 on page 24.

Chapter 4 on page 27 shows how to solve more complex measurement tasks, e.g., if the object shows different gray levels or if parts of the image are disturbed by reflections or by shadows that are cast on the object. For this purpose, it is possible to introduce special weighting functions to suppress unwanted edges. This extension is called fuzzy measure object.

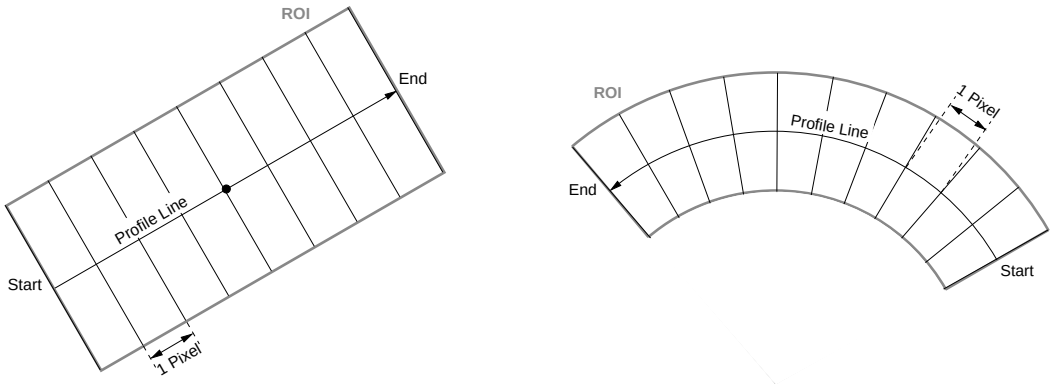
Chapter 2

The Basics of Measure Objects

The main idea behind the measure object is to extract edges that run approximately perpendicular to the line or arc of measurement. This process is described in section 2.1. It is important to understand this process because you influence it with parameters both when creating the measure object and when applying it. Section 2.2 on page 11 shows how to specify the shape of the measuring ROI. Please note that in contrast to other HALCON methods, measuring ROIs are not created using `reduce_domain`, but by specifying their dimensions when creating a measure object. A measure object can be used to determine the position of straight edges that run approximately perpendicular to the given ROI.

2.1 The Process of 1D Edge Extraction

HALCON determines the position of 1D edges by the following approach: First, equidistant lines of projection are constructed perpendicular to the line or arc of measurement (also called profile line) with a length equal to the width of the ROI (figure 2.1).



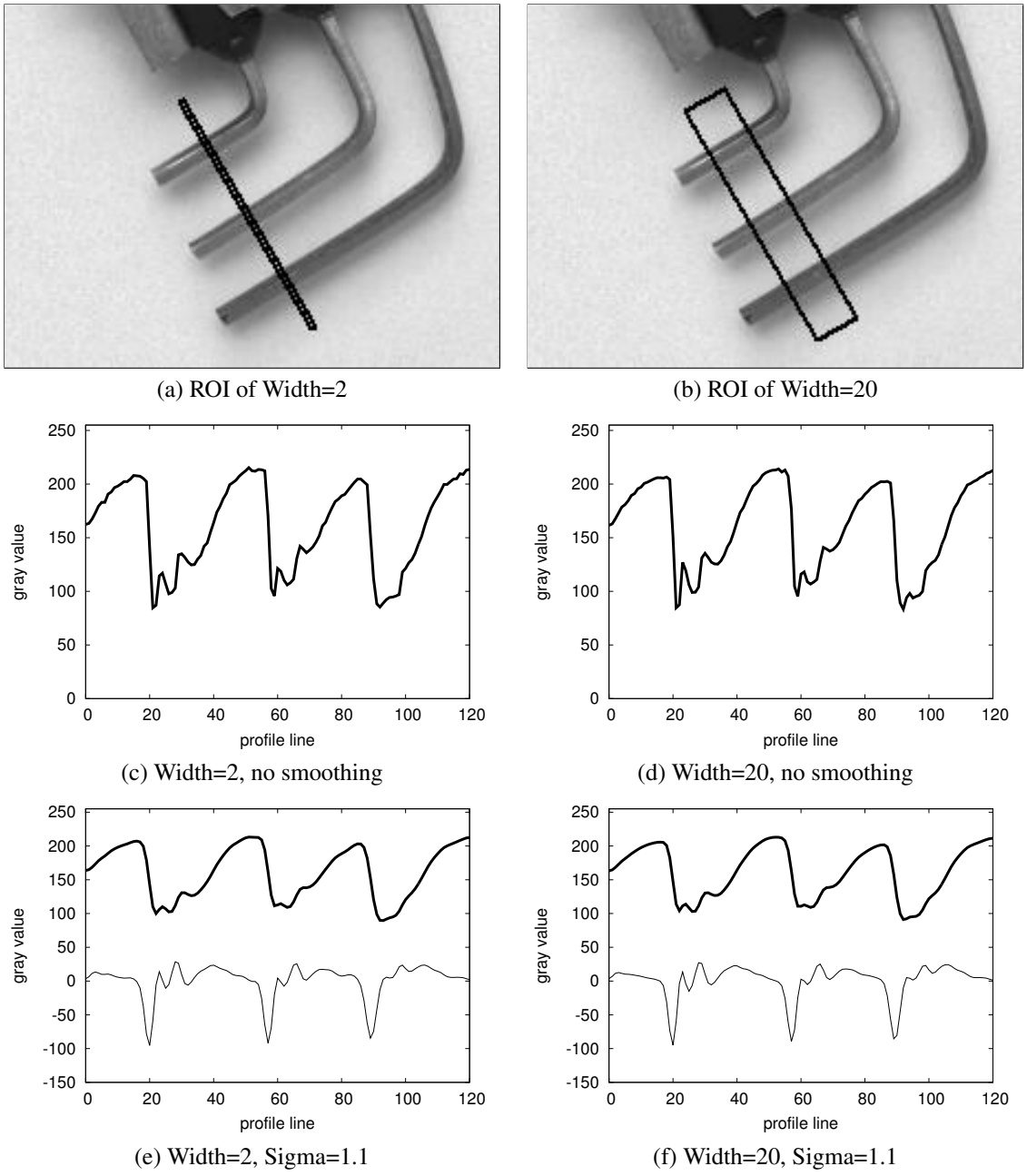
(a) (b)

Figure 2.1: Lines of projection for a (a) rectangular ROI and (b) circular ROI.

Then, the mean gray value along each line of projection is calculated. The sequence of these mean values is called the profile. If the lines of projection are not oriented horizontally or vertically, the pixel values along them must be interpolated. You can select the interpolation mode with the parameter `Interpolation` of the operators for the creation of measure objects (e.g., `gen_measure_rectangle2`, see chapter 1 on page 7). If `Interpolation` = *'nearest_neighbor'*, the gray values in the measurement are obtained from the gray values of the closest pixel, i.e., by constant interpolation. This is the fastest method. However, the geometric accuracy is slightly lower in this mode. For `Interpolation` = *'bilinear'*, bilinear interpolation is used, while for `Interpolation` = *'bicubic'*, bicubic interpolation is used. The bicubic interpolation yields the most accurate results. However, it is the slowest method.

The length of the lines of projection, i.e., the width of the ROI, defines the degree of averaging in the direction perpendicular to the profile line. The thick lines in figure 2.2c and d show the profiles for the two ROIs displayed in

figure 2.2a and b. It can be seen that the profile that corresponds to the wider ROI is less noisy. Therefore, as long as the edges run approximately perpendicular to the profile line, the ROI should be chosen as wide as possible. If the edges do not run perpendicular to the profile line, the width of the ROI must be chosen smaller to allow the detection of edges. Note that in this case, the profile will contain more noise. Consequently, the edges will be determined less accurate (see appendix A on page 37).



(a) ROI of Width=2

(b) ROI of Width=20

(c) Width=2, no smoothing

(d) Width=20, no smoothing

(e) Width=2, Sigma=1.1

(f) Width=20, Sigma=1.1

Figure 2.2: Profiles (thick lines) and derivatives (thin lines) of the profiles for two ROIs with different width with and without smoothing the profiles (the orientation of the ROIs is from top left down to the right).

The profile is then smoothed with a Gaussian smoothing filter (see the thick lines in figure 2.2e and f), whose standard deviation is specified with the parameter `Sigma` of the measuring operators (e.g., `measure_pairs`). Now, the first derivative of the smoothed profile is calculated (thin lines in figure 2.2e-f). Note that the amplitude of the derivative is scaled with a factor $\sqrt{2\pi} \cdot$ `Sigma`. The subpixel precise positions of all local extrema of the first derivative are edge candidates. In figure 2.3a, these edge candidates are indicated by vectors pointing from the respective position of the derivative (thin line) to the smoothed gray value profile (thick line). Only those edge candidates that have an absolute value of the first derivative larger than a given `Threshold` (another parameter of the measuring operators) are considered as edges in the image (see figure 2.3b).

For each edge, the position of its intersection with the profile line is returned. Figure 2.4 shows the detected edges,

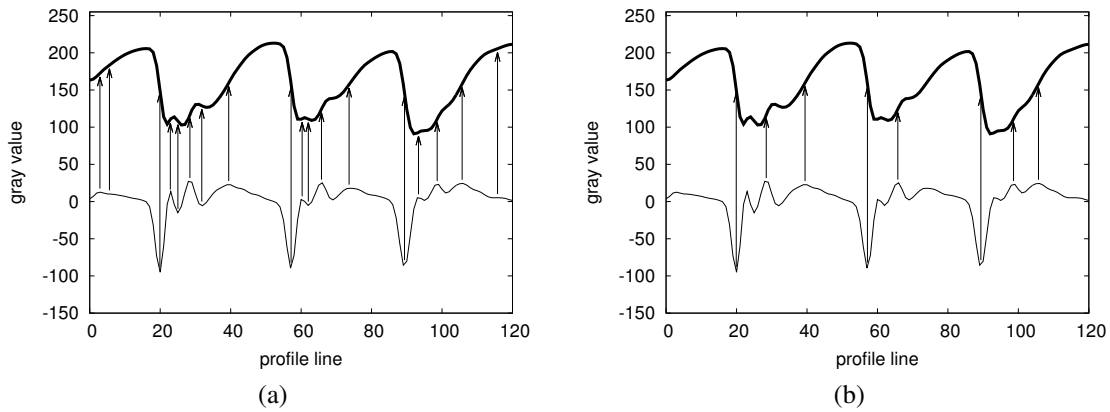


Figure 2.3: Positions of (a) edge candidates and (b) edges along the profile. The profile was derived from the 20 pixel wide ROI, it was smoothed with `Sigma` = 1.1.

displayed by straight lines with a length similar to the width of the ROI.

Note that the above described approach for the extraction of 1D edges from a profile differs significantly from the 2D edge extraction performed, e.g., with the operator `edges_sub_pix` (compare figure 2.5). In the case of the 1D edge extraction, only the positions of the intersections of the edges with the profile line are determined as described above whereas the 2D edge extraction returns the complete edges as contours. If the edges are straight and run approximately perpendicular to the profile line, the results of the 1D edge extraction are similar to the intersection of the 2D edges with the profile line. The more the edges are curved or not perpendicular to the profile line, the more different the results of the two approaches will be. An example where the results of the two approaches are completely different is given in section 3.1 on page 15.

Keep in mind that measure objects are designed to measure the position of straight edges that run approximately perpendicular to the profile line. This means that measure objects are not suited to measure the position of, e.g., curved edges. In those cases, the measure position will typically not coincide with the intersection of the gray value edge with the profile line. An example for such a case is given in figure 2.6. Here, the edges are curved, which leads to wrong measurement results. This effect will be reduced if the width of the ROI is chosen smaller.

2.2 Creating a Measure Object

Measure objects can be created with two different shapes: Rotated rectangles and circular arcs. They are created with the operators `gen_measure_rectangle2` or `gen_measure_arc`.

To create a rectangular measure object, you use the operator `gen_measure_rectangle2`:

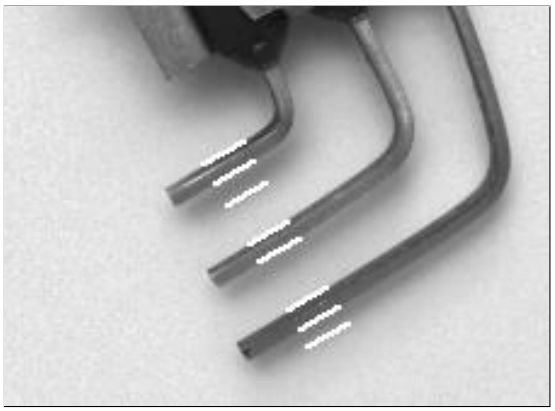


Figure 2.4: Edges detected from the 20 pixel wide ROI. The profile was smoothed with `Sigma` = 1.1 and thresholded with a `Threshold` of 20.

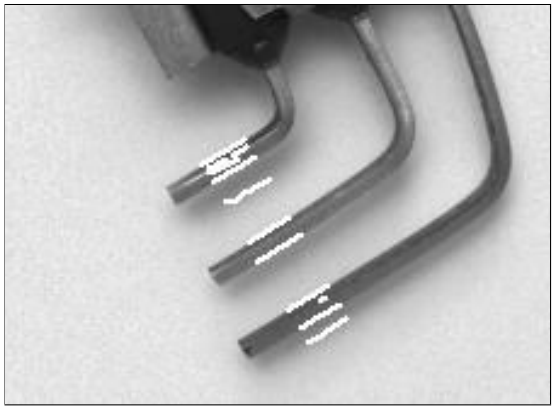


Figure 2.5: Edges detected in 2D with the operator edges_sub_pix.

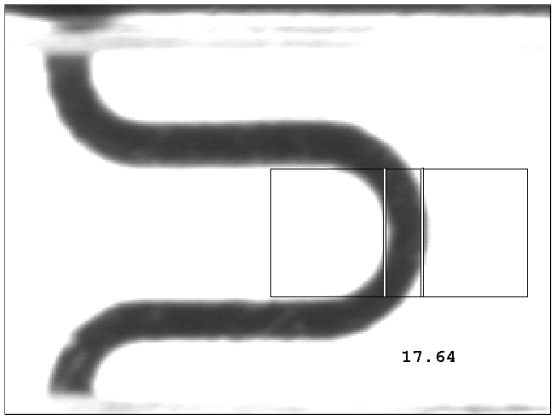


Figure 2.6: An example for an *incorrect* use of the measure object. The ROI is oriented from left to right.

```
gen_measure_rectangle2 (Row, Column, Phi, Length1, Length2, Width, Height, \
                        Interpolation, MeasureHandle)
```

A rotated rectangle is defined by its center (Row, Column), its orientation Phi, and its half edge lengths Length1 and Length2 (see figure 2.7). The parameter Phi is given in radians in mathematically positive sense. All the other parameters are given in pixels.

To create a circular measure object, you use the operator gen_measure_arc:

```
gen_measure_arc (CenterRow, CenterCol, Radius, AngleStart, AngleExtent, \
                 AnnulusRadius, Width, Height, Interpolation, \
                 MeasureHandle)
```

The circular arc is defined by the following parameters (see figure 2.8): The position and size are defined by the center (CenterRow, CenterColumn) and the radius (Radius). The segment of the circle is defined by its start angle (AngleStart) and the angular extent (AngleExtent). The width of the circular ROI is defined by the annulus radius (AnnulusRadius). Again, all parameters are given in pixels, except for the angles (AngleStart and AngleExtent), which are given in radians in mathematically positive sense.

The first parameters of both operators for the creation of measure objects define the position, size, and orientation of the respective ROI. The parameters Width and Height are the size of the images, on which the measure can be applied. The parameter Interpolation sets the interpolation mode (see section 2.1 on page 9). Note that when using interpolation, not only the ROI but additionally the margin around the ROI must fit into the image. The width of the margin (in all four directions) has to be at least 1 pixel for a bilinear interpolation and two pixels for a bicubic interpolation. For lines of projection that do not fulfill this condition, no result is obtained.

The result for both operators is a handle for the newly created measure object (MeasureHandle), which can then be used to specify the measure object, e.g., in calls to the operator measure_pos. Note that if you use HALCON's

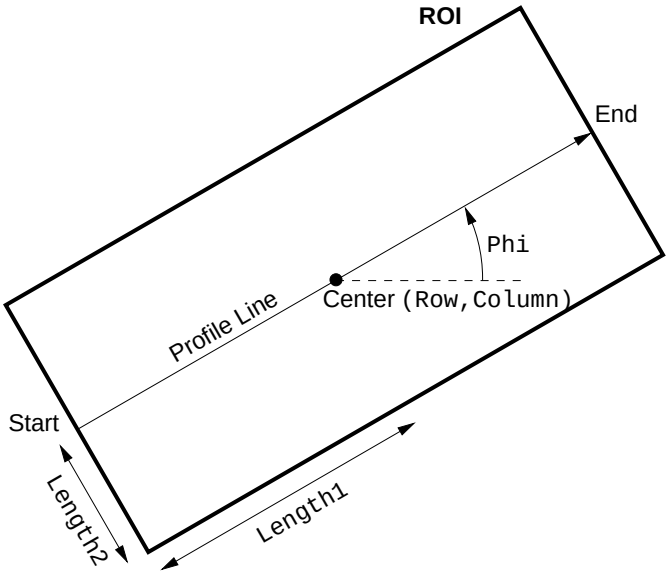


Figure 2.7: Rectangular region of interest (ROI).

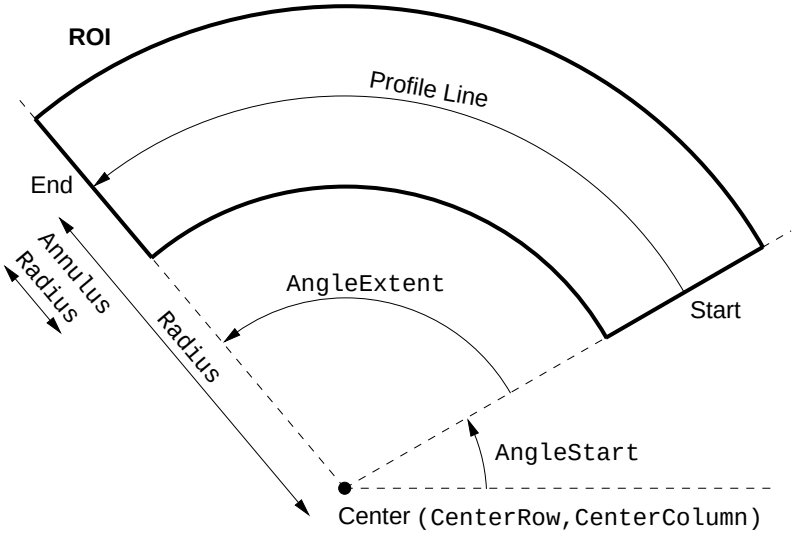


Figure 2.8: Circular region of interest (ROI).

C++ interface and call the operator via the classes HMeasureX or HMeasure, no handle is returned because the instance of the class itself acts as your handle.

Chapter 3

Using the Measure Object

For different tasks, depending on your application, you can measure the position of individual edges and the distance between them (see section 3.1). If the object to inspect has two edges (like the edge pairs in chapter 1 on page 7), you can let HALCON group the edges to edge pairs (see section 3.2 on page 17). Furthermore, the position of points on the profile that have a particular gray value can be determined (section 3.3 on page 19). It is also possible to access the gray value profile itself (section 3.4 on page 19).

Often, it is necessary to determine the dimensions of an object in world coordinates, e.g., in μm. The transformation of the results of a measure object into world coordinates is described in section 3.5 on page 22.

Section 3.7 on page 23 describes how to change the position and orientation of measure objects. Some tips and tricks for the use of measure objects are given in section 3.8 on page 24.

3.1 The Position of Edges

You measure the position of the individual edges with the operator `measure_pos`. The edges are determined with the approach described in section 2.1 on page 9. With this approach, it is also possible to extract edges that appear only if the image is smoothed in a direction perpendicular to the ROI.

In the example program `%HALCONEXAMPLES%\solution_guide\1d_measuring\measure_ic_leads.hdev`, we use this operator to measure the length of the leads of the IC that is displayed in figure 3.1a. At a first glance, it is not clear how to solve this task because the individual leads are not enclosed by two edges. However, we can exploit the averaging perpendicular to the profile line: If the ROI encloses multiple leads, as depicted in figure 3.1a, the averaged gray value of the leads is darker than the background, but lighter than the body of the IC (see figure 3.1b and figure 3.1c).

Let's take a look at the corresponding code. First, a measure object is created for the ROI at the top of the image. The ROI must have a width such that it contains several leads.

```
gen_measure_rectangle2 (Row, Column, Phi, Length1, Length2, Width, Height, \
                        Interpolation, MeasureHandle)
```

Now, all edges that are perpendicular to the ROI are extracted.

```
measure_pos (Image, MeasureHandle, Sigma, Threshold, Transition, Select, \
             RowEdge, ColumnEdge, Amplitude, Distance)
```

The resulting edges enclose the leads (see figure 3.2). Their distance is returned in the parameter `Distance`. In the example, only two edges are returned. Therefore, the parameter `Distance` contains only one element, which is the length of the leads at the top of the image. To determine the length of the leads at the bottom of the image, the same measurements are carried out within the second ROI.

The parameter `Sigma` determines the degree of smoothing of the profile along the profile line. The parameter `Threshold` defines the threshold that is applied to the derivative of the profile for the selection of edges. Both parameters are described in section 2.1 on page 9.

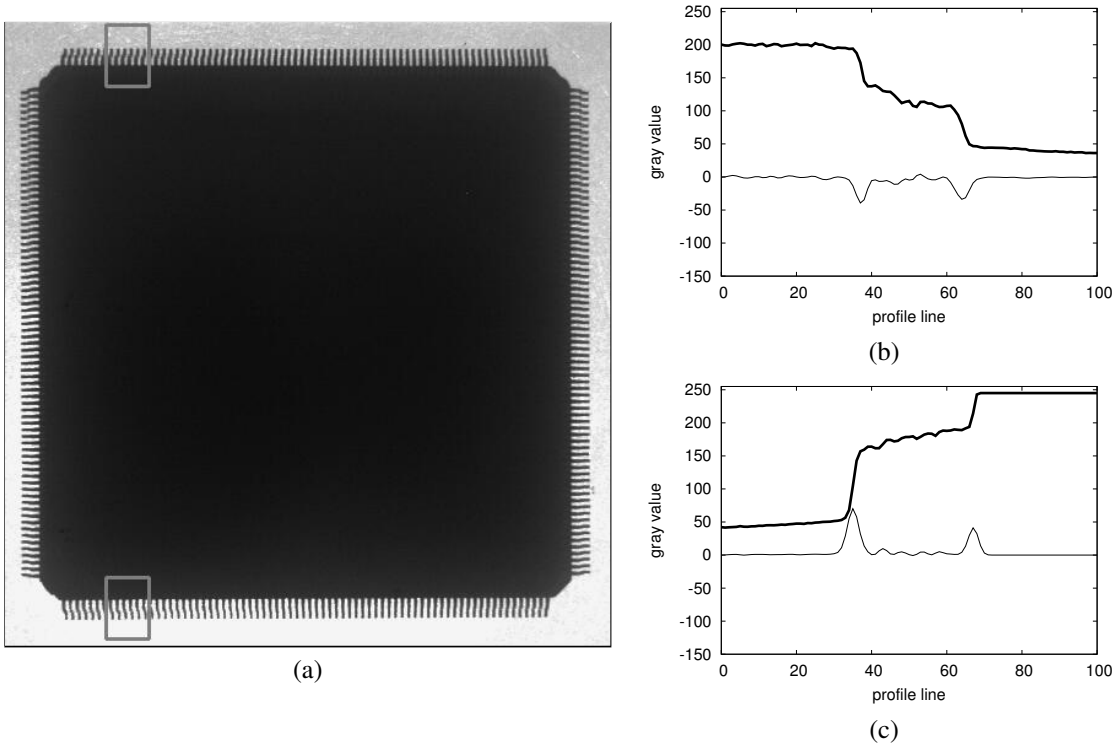


Figure 3.1: (a) Image of an IC with the ROIs that are used for the determination of the lead length; (b) and (c) show the profile (thick line) and its derivation (thin line) for (b) the ROI at the top and (c) the ROI at the bottom of the image in (a). The ROIs are oriented from the top to the bottom of the image.

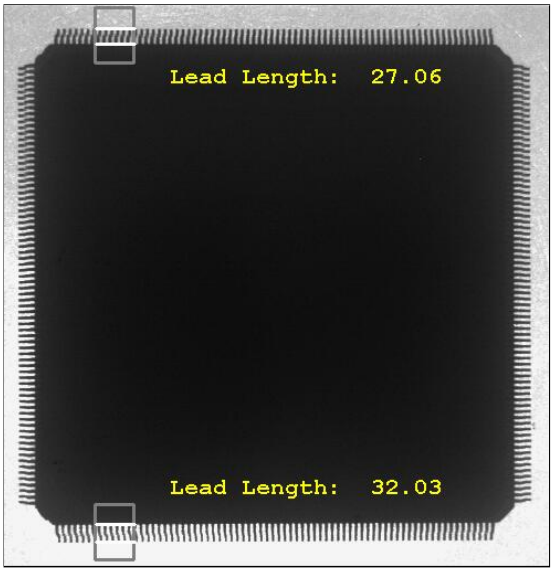


Figure 3.2: The result of the determination of the lead length.

The parameter `Transition` can be used to select edges with a particular transition. If `Transition` is set to *'negative'*, only edges at which the gray values change from light to dark will be returned. In the current example, the two edges in the ROI at the top of the image have a negative transition. If `Transition` is set to *'positive'*, only edges at which the gray values change from dark to light (e.g., the two edges in the ROI at the bottom of the image) will be returned. To determine all edges, `Transition` must be set to *'all'* in this application.

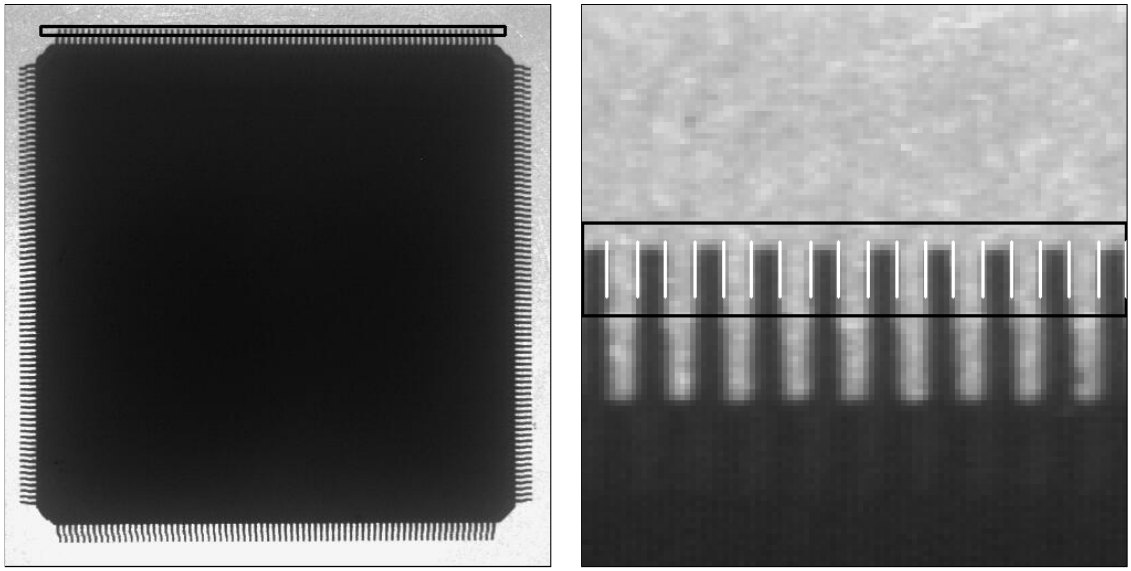
The parameter Select can be used to restrict the result to the *'first'* or *'last'* edge. In the current example, Select must be set to *'all'*.

Besides the distance of the edges, measure_pos also returns their position in the output parameters RowEdge and ColumnEdge as well as their amplitude (Amplitude). In the example, the position is used to visualize the edges:

```
disp_line (WindowHandle, RowEdge, ColumnEdge - Length2, RowEdge, \
           ColumnEdge + Length2)
```

3.2 The Position of Edge Pairs

As already noted in chapter 1 on page 7, whenever the object to measure is enclosed by two edges, you can use the operator measure_pairs, which groups edges to edge pairs. In the following example, we use this operator to determine the width of the leads from the image displayed in figure 3.1a on page 16.



(a) (b)

Figure 3.3: (a) The ROI for and (b) a part of the result of the determination of the lead width.

First, a suitable ROI must be defined. Because we want to determine the edges of the individual leads, the ROI must be oriented perpendicular to the leads (see figure 3.3a). Here, it is oriented from the left to the right. With this ROI, a measure object is created as described in chapter 2 on page 9.

Now, all edge pairs that enclose dark image regions are extracted (%HALCONEXAMPLES%\solution_guide\1d_measuring\measure_ic_leads.hdev):

```
measure_pairs (Image, MeasureHandle, Sigma, Threshold, Transition, Select, \
               RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, \
               RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, \
               IntraDistance, InterDistance)
```

The resulting edges are displayed in figure 3.3b, which shows only a part of the image.

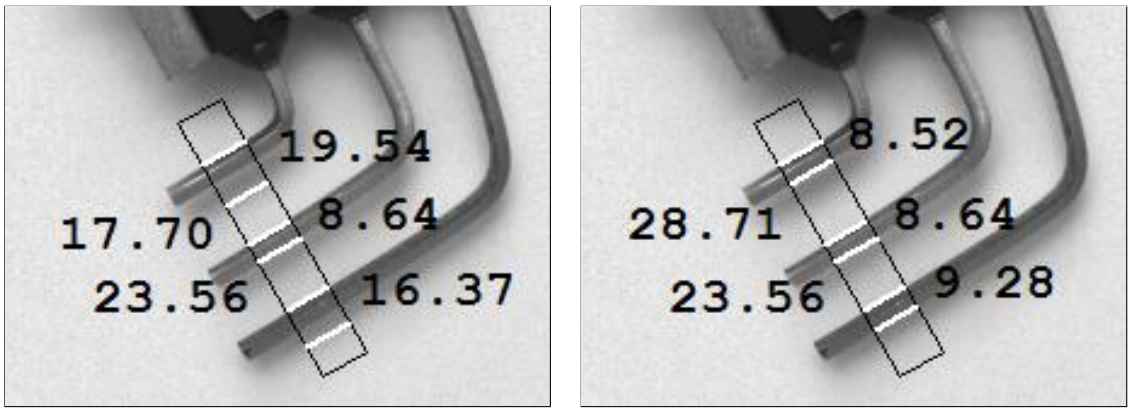
The parameters of measure_pairs are partly identical to the ones of measure_pos (see section 3.1 on page 15): Sigma determines the degree of smoothing of the profile. The parameter Threshold defines the threshold that is applied to the derivative of the profile for the selection of edges. Both parameters are described in section 2.1 on page 9.

The parameter Transition can be used to select edge pairs with a particular transition. If Transition is set to *'negative'*, as in the example, only edge pairs where the first edge has a negative transition and the second edge has a positive transition, i.e., edge pairs that enclose a dark image region will be returned. If Transition is set

to *'positive'*, the behavior is exactly opposite, i.e., edge pairs that enclose a bright image region will be returned. If `Transition` is set to *'all'*, the transition is determined by the first edge. I.e., dependent on the positioning of the measure object, edge pairs with a light-dark-light transition or edge pairs with a dark-light-dark transition are returned. This is suited, e.g., to measure objects with different brightness relative to the background.

If more than one consecutive edge with the same transition is found, only the first one will be used as a pair element. This behavior may cause problems in applications in which the `Threshold` cannot be selected high enough to suppress consecutive edges of the same transition. This problem can be demonstrated, e.g., by defining the ROI in the example program `%HALCONEXAMPLES%\solution_guide\1d_measuring\measure_switch.hdev` (see chapter 1 on page 7) the other way round, i.e., from bottom right to top left. Then, the first edge with a negative transition corresponds to the shadow of the pin and the edge that corresponds to the pin itself is not used as a pair element (see figure 3.4a).

For these applications, an advanced pairing mode exists that only selects the strongest edges of a sequence of consecutive edges of the same transition. This mode is selected by appending *'_strongest'* to any of the above modes for `Transition`, e.g., *'negative_strongest'*. Figure 3.4b shows the result of setting `Transition` to *'negative_strongest'*.



(a) (b)

Figure 3.4: The results of measuring the width of and the distance between the pins of a switch if the ROI is defined from bottom right to top left: (a) `Transition` = *'negative'*, (b) `Transition` = *'negative_strongest'*.

The parameter `Select` can be used to restrict the result to the *'first'* or *'last'* edge pair. To determine all edge pairs, `Select` must be set to *'all'*.

The operator `measure_pairs` returns the coordinates of the edge positions separately for the first and the second edge of each edge pair. The output parameters `RowEdgeFirst` and `ColumnEdgeFirst` contain the coordinates of the first edges of the edge pairs and the parameters `RowEdgeSecond` and `ColumnEdgeSecond` contain the coordinates of the second edges. With this, the center of the pairs can easily be determined by calculating the mean of the respective first and second edge positions:

```
RowPairCenter := (RowEdgeFirst + RowEdgeSecond) / 2.0
ColumnPairCenter := (ColumnEdgeFirst + ColumnEdgeSecond) / 2.0
```

The output parameters `IntraDistance` and `InterDistance` contain the distance between the two edges of the edge pairs and the distance between the edge pairs.

3.3 The Position of Points With a Particular Gray Value

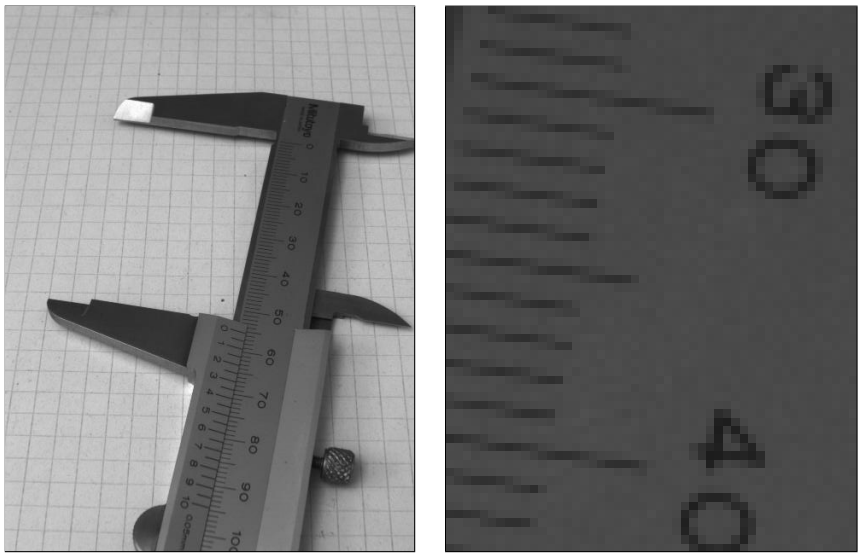
With the operator measure_thresh, it is also possible to determine the subpixel precise positions where the gray value profile has a given value. This 1D thresholding works like the 2D subpixel precise thresholding operator threshold_sub_pix.

This operator can be useful to extract particular edges in applications where the lighting conditions can be controlled very accurately. Note that the measured position depends heavily on the brightness of the image, i.e., the measured position will vary if the brightness level of the image changes, e.g., due to illumination changes.

3.4 The Gray Projection

For applications that require the determination of special 1D features, it is possible to access the gray value profile directly via the operator measure_projection. It returns the original, i.e., unsmoothed gray value profile as a tuple in the output parameter GrayValues. On the profile, points that satisfy a particular condition can be determined. Then, the pixel coordinates of these points can be calculated.

In the example program %HALCONEXAMPLES%\solution_guide\1d_measuring\measure_caliper.hdev, the distances between the pitch lines of a caliper are determined. The used image (figure 3.5a) shows the pitch lines as approximately one pixel wide lines (see, e.g., figure 3.5b). To determine the line positions in the gray value profile, a 1D line extraction is implemented based on the gray value profile.



(a) (b)

Figure 3.5: (a) The image of a caliper; (b) detail of (a).

First, the measure object is created. It is positioned such that only the longest pitch lines, i.e., those which indicate full centimeters run through the measuring ROI.

```
gen_measure_rectangle2 (Row, Column, Phi, Length1, Length2, Width, Height, \
                        'bilinear', MeasureHandle)
```

Then, the profile is determined with the operator measure_projection, which returns the profile in the output parameter GrayValues:

```
measure_projection (Image, MeasureHandle, GrayValues)
```

To reduce the noise, the profile must be smoothed because the operator measure_projection returns the original, i.e., unsmoothed profile.

```
Sigma := 0.3
create_funct_1d_array (GrayValues, Function)
smooth_funct_1d_gauss (Function, Sigma, SmoothedFunction)
```

The smoothed profile is displayed in figure 3.6.

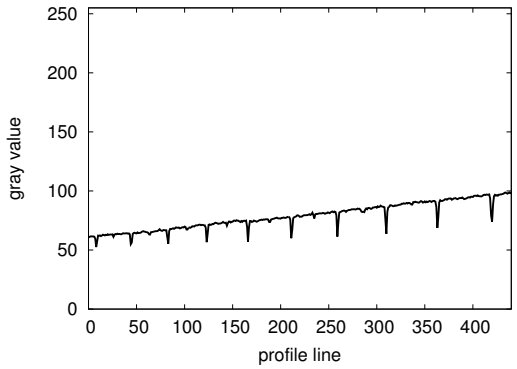
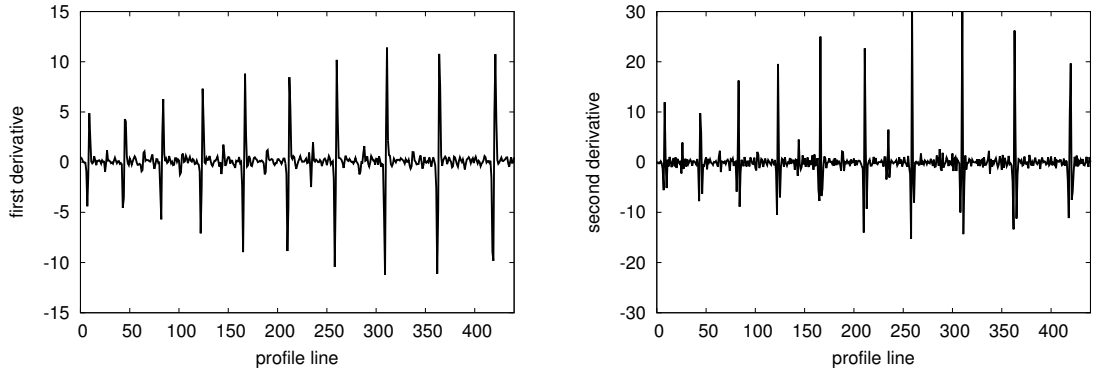


Figure 3.6: The smoothed profile.

To detect line points on the profile, it is sufficient to determine the points where the first derivative of the smoothed profile vanishes. The first derivative of the smoothed profile is determined with the operator `derivate_funct_1d` with the control parameter `Mode` set to *'first'*; the points where the first derivative vanishes are determined with the operator `zero_crossings_funct_1d`:

```
derivate_funct_1d (SmoothedFunction, 'first', FirstDerivative)
zero_crossings_funct_1d (FirstDerivative, ZeroCrossings)
```

The first derivative of the smoothed profile is displayed in figure 3.7a.



(a) (b)

Figure 3.7: The (a) first and (b) second derivative of the smoothed profile.

Because of noise, the first derivative shows many more zero crossings than just in the middle of each line, i.e., between the large negative and positive spikes in figure 3.7a. We can select the desired zero crossings by looking at the magnitude of the second derivative: Bright lines on a dark background will have a negative second derivative while dark lines on a bright background will have a positive second derivative. The second derivative of the smoothed profile is determined with the operator `derivate_funct_1d` with the control parameter `Mode` set to *'second'*:

```
derivate_funct_1d (SmoothedFunction, 'second', SecondDerivative)
```

The second derivative of the smoothed profile is displayed in figure 3.7b.

In our case, the pitch lines appear as dark lines on a bright background. Therefore, we have to select those line positions, where the second derivative has a large positive value: For each line position, i.e., zero crossing of the first derivative, the value of the second derivative is determined with the operator get_y_value_funct_1d. If this value is larger than a user defined threshold, the line position will be stored in a tuple that holds the positions of all salient lines.

```
MinimumMagnitudeOfSecondDerivative := 8
PositionOfSalientLine := []
for i := 0 to |ZeroCrossings| - 1 by 1
    get_y_value_funct_1d (SecondDerivative, ZeroCrossings[i], 'constant', Y)
    if (Y > MinimumMagnitudeOfSecondDerivative)
        PositionOfSalientLine := [PositionOfSalientLine,ZeroCrossings[i]]
    endif
endfor
```

Finally, the positions of the lines on the profile must be transformed into pixel coordinates because measure_projection returns just the gray value profile along the profile line. Note that internally, the coordinates of the center of the measuring ROI are rounded and the length of the measuring ROI is set to the largest integer value that is smaller or equal to the given value.

The pixel coordinates of the start point of the profile line can be determined as follows (see figure 3.8):

$$\begin{aligned} \texttt{RowStart} &= \lfloor(\texttt{Row} + 0.5)\rfloor + \lfloor\texttt{Length1}\rfloor \cdot \sin(\texttt{Phi}) \\ \texttt{ColStart} &= \lfloor(\texttt{Column} + 0.5)\rfloor - \lfloor\texttt{Length1}\rfloor \cdot \cos(\texttt{Phi}) \end{aligned}$$

This can be realized by the following lines of code:

```
RowStart := floor(Row + 0.5) + floor(Length1) * sin(Phi)
ColStart := floor(Column + 0.5) - floor(Length1) * cos(Phi)
```

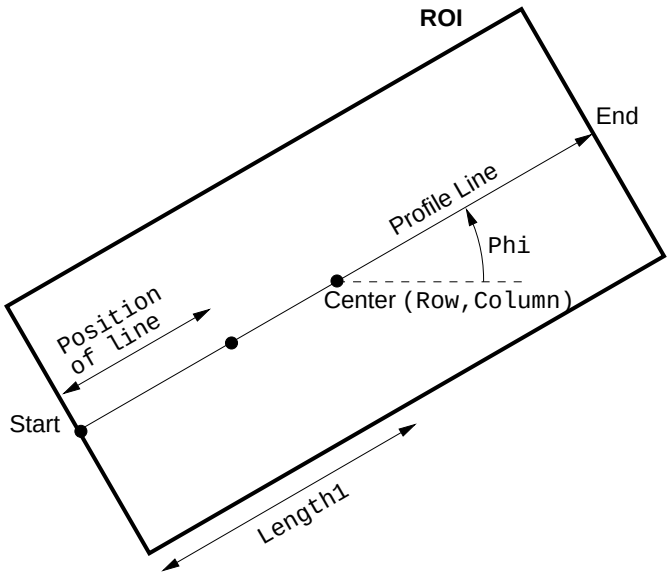


Figure 3.8: The measuring ROI with the position of a point on the profile.

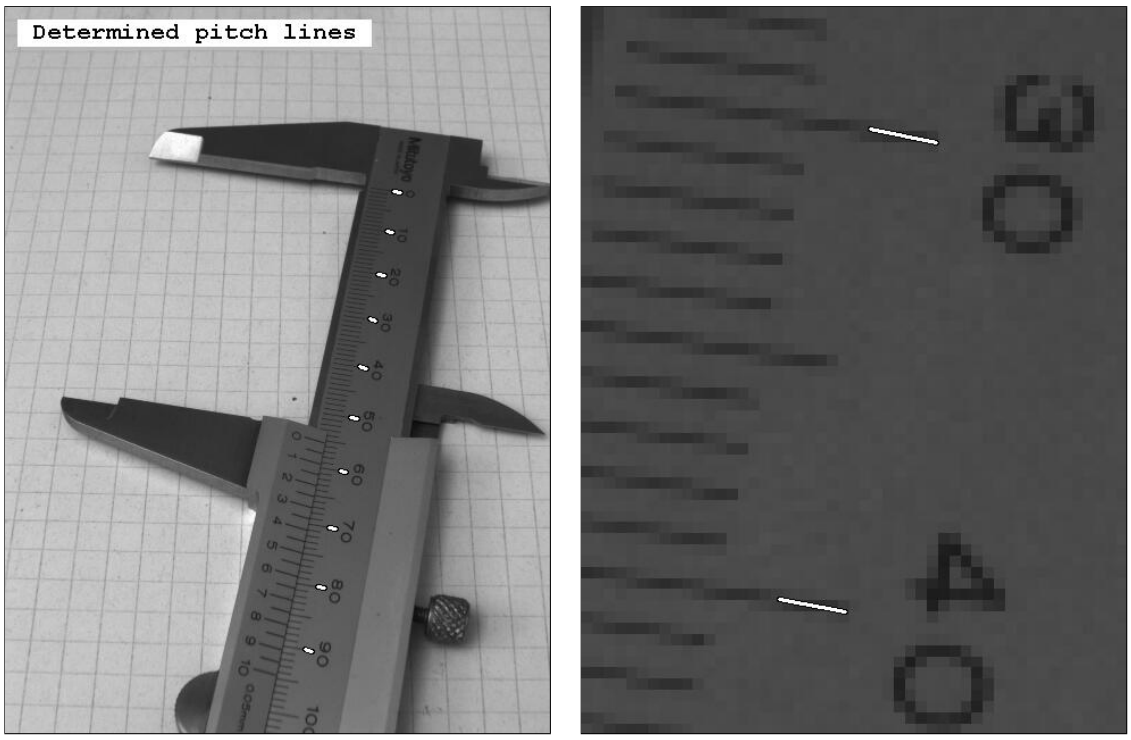
Now, the pixel coordinates of the positions of the salient lines on the profile can be determined as follows:

$$\begin{aligned} \texttt{RowLine} &= \texttt{RowStart} - \texttt{PositionOfSalientLine} \cdot \sin(\texttt{Phi}) \\ \texttt{ColLine} &= \texttt{ColStart} + \texttt{PositionOfSalientLine} \cdot \cos(\texttt{Phi}) \end{aligned}$$

In HDevelop, this can be expressed by the following lines of code:

```
RowLine := RowStart - PositionOfSalientLine * sin(Phi)
ColLine := ColStart + PositionOfSalientLine * cos(Phi)
```

Figure 3.9 shows the positions of the determined lines.



(a) (b)

Figure 3.9: (a) The determined lines; (b) detail of (a).

In the following section, the pixel coordinates of the pitch lines and the distance between them are transformed into world coordinates.

If you are using a circular measure object, which has been created with the operator gen_measure_arc, the following formulae must be used to determine the pixel coordinates RowPoint and ColPoint of a point on the profile:

$$\begin{aligned} \texttt{RowPoint} &= \lfloor(\texttt{CenterRow} + 0.5)\rfloor - \texttt{Radius} \cdot \sin(\texttt{AngleStart} + \texttt{Position}) \\ \texttt{ColPoint} &= \lfloor(\texttt{CenterColumn} + 0.5)\rfloor + \texttt{Radius} \cdot \cos(\texttt{AngleStart} + \texttt{Position}) \end{aligned}$$

Here, Position is the position of the point on the profile and CenterRow, CenterColumn, Radius, and AngleStart are the parameters used for the creation of the measure object with the operator gen_measure_arc.

3.5 Measuring in World Coordinates

In many applications, the dimensions of the measured object must be determined in world coordinates, e.g., in μm. This can either be achieved by transforming the original measurement results into world coordinates (see Solution Guide III-C, section 3.3 on page 75) or, by first rectifying the image and applying the measurement in the transformed image (see Solution Guide III-C, section 3.4 on page 78). As a prerequisite, the camera must be calibrated (Solution Guide III-C, section 3.2 on page 59).

In the example program %HALCONEXAMPLES%\solution_guide\1d_measuring\measure_caliper.hdev, first the positions of the pitch lines of a caliper are determined (see section 3.4 on page 19). These positions are now transformed into world coordinates with the operator image_points_to_world_plane:

```
image_points_to_world_plane (CamParam, WorldPose, RowLine, ColLine, 'mm', X, \
                             Y)
```

The parameter CamParam contains the camera parameters and the parameter WorldPose the position and orientation of the world plane in which the measurement takes place. Both parameters can be determined during

the camera calibration (Solution Guide III-C, section 3.2 on page 59). In the current example program, they are assumed to be known.

Now, the distance between the pitch lines can be calculated with the operator `distance_pp`:

```
Num := |X|
distance_pp (X[0:Num - 2], Y[0:Num - 2], X[1:Num - 1], Y[1:Num - 1], \
             Distance)
```

The resulting distances are displayed in figure 3.10.

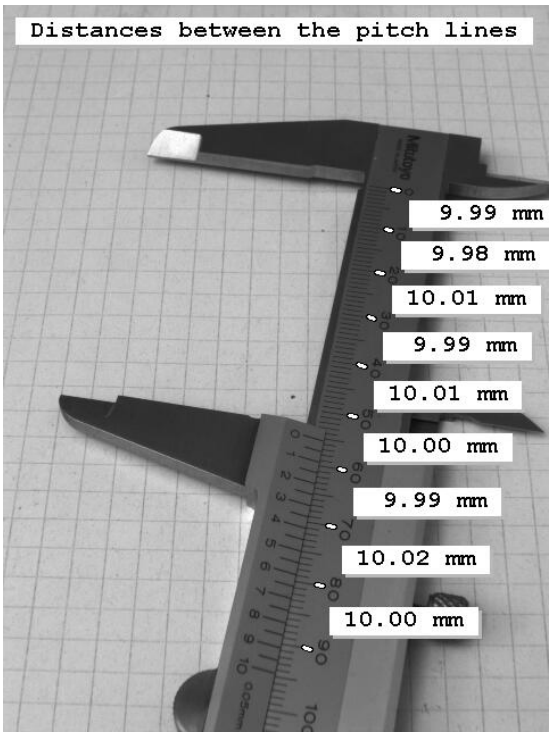


Figure 3.10: The distances between the pitch lines in world coordinates.

3.6 Compensate for Lens Distortions

If the image suffers from heavy lens distortions, a correction of the image coordinates or a rectification of the image will be necessary.

A detailed description of how to correct the image coordinates can be found in Solution Guide III-C, section 3.3.6 on page 77. The description of how to rectify images can be found in the Solution Guide III-C, section 3.4.2 on page 84.

3.7 Changing the Position and Orientation of the Measure Object

Often, the position of the objects to be measured varies from image to image. In this case, the ROI of the measure must be adapted accordingly. This process is also called alignment.

The position and orientation of the object to be measured can be determined, e.g., with shape-based matching. For a detailed description of shape-based matching, please refer to the Solution Guide II-B. This manual also contains an example for aligning a measure ROI (see Solution Guide II-B, section 2.4.3.2 on page 34).

If only the position of the measure ROI must be changed, the operator `translate_measure` can be used. If also the orientation must be adapted as in the above mentioned example in the Solution Guide II-B, it is necessary to transform the parameters that define the measure ROI and to create a new measure object with these parameters.

3.8 Tips and Tricks

3.8.1 Distances for Circular ROIs

There are multiple representations for the distance between two edges that are determined with a measure object that is based on a circular ROI (see figure 3.11):

- arc length,
- linear distance, and
- angular distance.

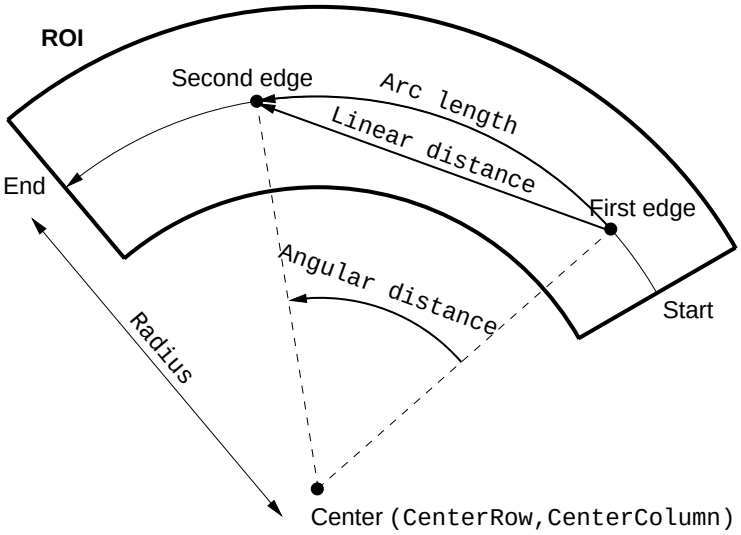


Figure 3.11: Distances for circular ROIs.

Measure objects based on a circular ROI return the distances as arc lengths.

The example program `%HALCONEXAMPLES%\solution_guide\1d_measuring\measure_ring.hdev` shows that it is very easy to transform the measurements into another representation. In this example, the size of the cogs of a cogwheel must be measured (see figure 3.12).

First, a measure object is created based on a circular ROI:

```
gen_measure_arc (CenterRow, CenterCol, Radius, AngleStart, AngleExtent, \
                 AnnulusRadius, Width, Height, Interpolation, \
                 MeasureHandle)
```

Then, the edges of the cogs are determined with the operator `measure_pairs`:

```
measure_pairs (Image, MeasureHandle, Sigma, Threshold, Transition, Select, \
               RowEdgeFirst, ColumnEdgeFirst, AmplitudeFirst, \
               RowEdgeSecond, ColumnEdgeSecond, AmplitudeSecond, \
               IntraDistance, InterDistance)
```

The size of the cogs, i.e., the distance between the edges of the edge pairs is returned in the output parameter *'IntraDistance'* as arc length.

The linear distance can be calculated from the coordinates of the edges with the operator `distance_pp`:

```
distance_pp (RowEdgeFirst, ColumnEdgeFirst, RowEdgeSecond, ColumnEdgeSecond, \
             LinearDistance)
```

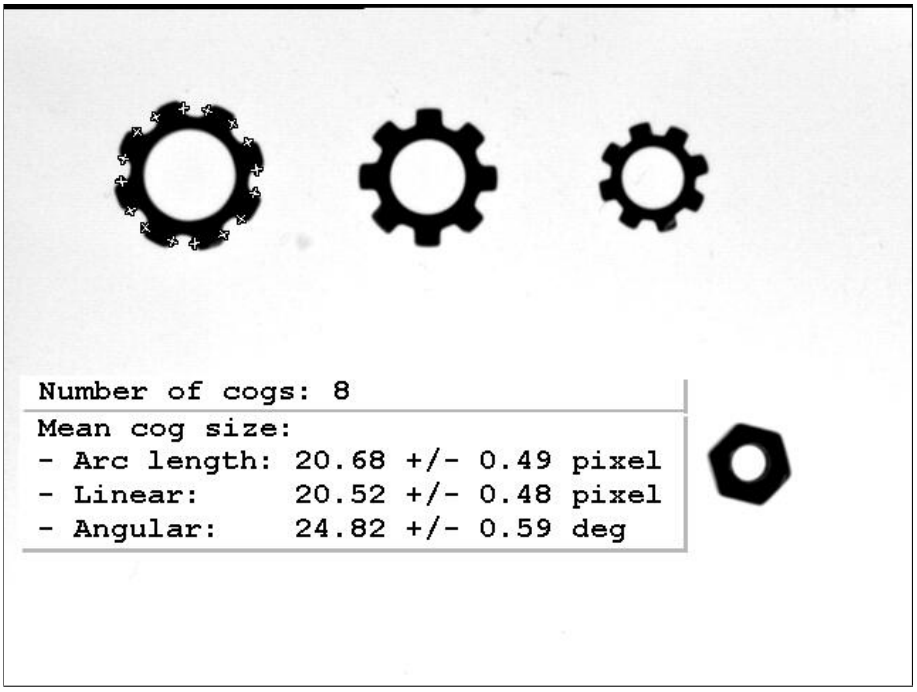


Figure 3.12: The size of the cogs of the ring in the upper left corner of the image given in different representations.

The angular distance can be derived from the arc lengths as follows:

$$\texttt{AngularDistance} = \frac{\texttt{Arc length}}{\texttt{Radius}}$$

which is implemented in the HDevelop example program as follows:

```
AngularDistance := deg(IntraDistance / Radius)
```

Note that here the angular distance is determined in degrees.

3.8.2 ROIs That Lie Partially Outside the Image

Gray value information exists only for positions on the profile line where at least a part of the respective line of projection (see section 2.1 on page 9) lies inside the image. If parts of an ROI lie outside the image, the edges inside the image and the distances between them will still be determined correctly.

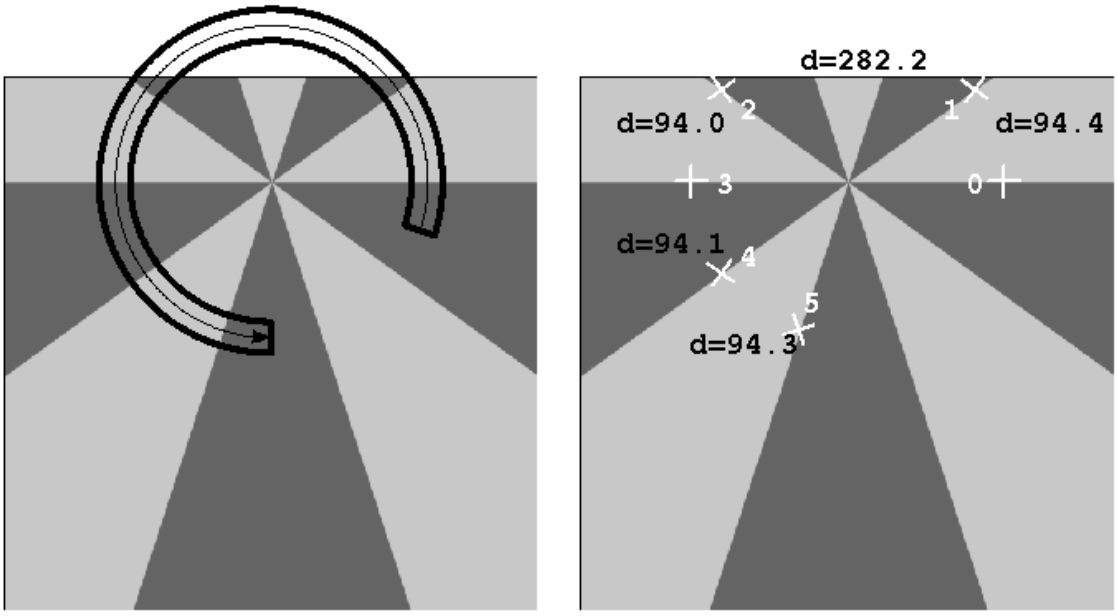
The example presented in figure 3.13 demonstrates such a case. Here, the edges of a Siemens star must be determined with a measure object centered at the center of the Siemens star. In figure 3.13a, the image as well as the outline of the ROI are shown. The upper part of the ROI lies outside the image. The distances between the edges as returned by the measure object are displayed in figure 3.13b.

Note that at positions where no gray value information is available, the profile contains zeros. Figure 3.14 shows the gray value profile determined with the operator `measure_projection` from the measure object that was created with the above described circular ROI (see figure 3.13a).

3.8.3 Accuracy vs. Speed

By default, the averaging of gray values (see section 2.1 on page 9) is performed using fixed point arithmetic, to achieve maximum speed. You can trade some of the speed against accuracy by setting the system parameter *'int_zooming'* to *'false'* with the operator `set_system`. Then the internal calculations are performed using floating point arithmetic. Note that the gain in accuracy is minimal.

See also the description of the parameter `Interpolation` at section 2.1 on page 9.



(a) (b)

Figure 3.13: Image of a Siemens star: (a) The circular ROI lies partially outside the image; (b) the measured distances between the edges.

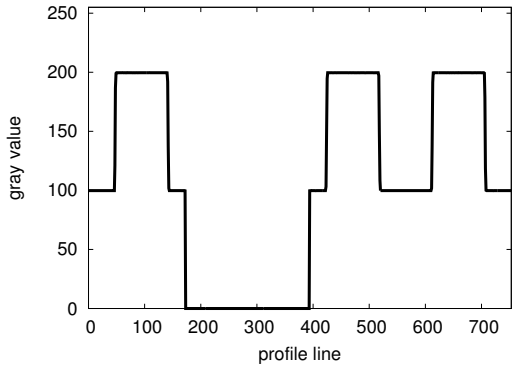


Figure 3.14: The gray value profile from the circular ROI that lies partially outside the image.

3.8.4 Image Type

Measure objects can be applied to images of type *'byte'*, *int1*, *int2*, *'uint2'*, *int4*, and *float*.

Note that if a multi-channel image is passed to the measuring operators (e.g., `measure_pos`), only the first channel of the image will be used.

Chapter 4

Advanced Measuring: The Fuzzy Measure Object

Some applications require that the selection of the edges or edge pairs can be controlled in more detail. With the standard measure object (see above), edges or edge pairs can be selected only based on their contrast and transition. With a fuzzy measure object it is possible to select edges also based on their contrast and position. The selection of edge pairs can be additionally controlled based on their position and size, as well as based on the gray value between the two edges of the pair. The name fuzzy measure object does not mean that the measurements are "fuzzy"; it comes from the so-called "fuzzy membership functions" that control the selection of edges (see section 4.2.2 on page 30).

In the following section, a first example demonstrates how to create and use a fuzzy measure object. The features that can be used to control the selection of edges and edge pairs are described in section 4.2 on page 29 as well as the fuzzy membership functions, which are used to formalize the desired constraints. Finally, section 4.3 on page 31 shows how to create a fuzzy measure object from a standard measure object and one or more fuzzy membership functions.

Alternatively, you can use HDevelop's Measure Assistant, which is described in the HDevelop User's Guide, section 7.4 on page 249, to quickly guide you through the fuzzy measuring process.

4.1 A First Example

The fuzzy measure object is an extension to the standard measure object. It allows to control the selection of edges by specifying weighting functions.

The example program `%HALCONEXAMPLES%\solution_guide\1d_measuring\fuzzy_measure_switch.hdev` shows a possible usage of the fuzzy measure object. Again, the width of and the distance between the pins of a switch must be measured. In this case, there are reflections on the middle pin (see figure 4.1), which will lead to wrong results if the standard measure object is used (figure 4.2a).

The selection of edges can be improved if additional information is available. In the example, the pins are approximately 9 pixels wide. This information can be translated into a fuzzy membership function. Figure 4.3 shows this fuzzy membership function. It returns 1.0 for the desired pair size and 0.0 if the size of the pairs deviates more than two pixels from the desired pair size. The intermediate values are interpolated linearly (see figure 4.3). In HALCON, this fuzzy membership function is created with the operator `create_funct_1d_pairs`:

```
create_funct_1d_pairs ([7,9,11], [0.0,1.0,0.0], SizeFunction)
```

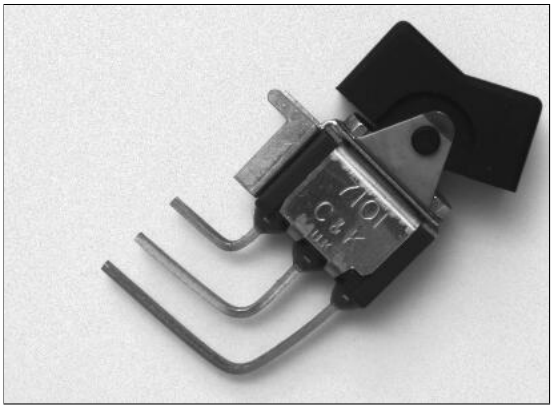
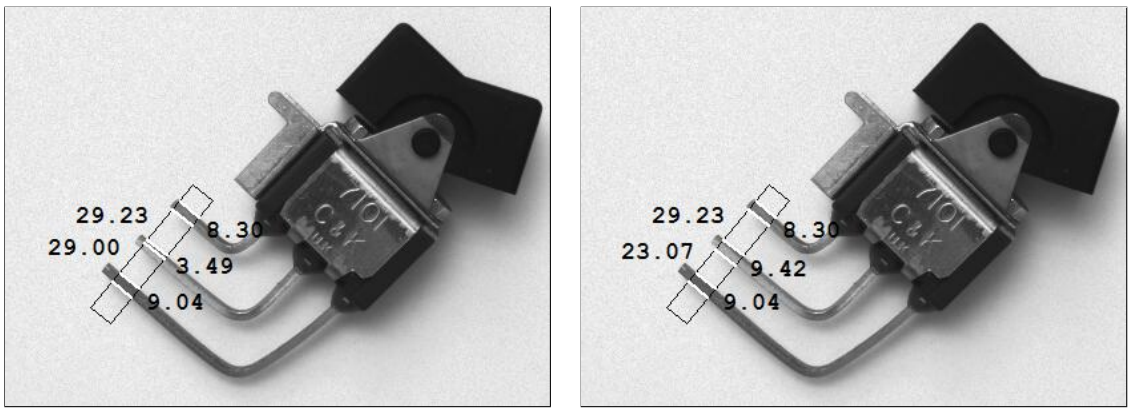


Figure 4.1: Image of a switch with reflections on the middle pin.



(a) (b)

Figure 4.2: The results of measuring the width of and the distance between the pins of the switch with (a) a standard measure object and (b) a fuzzy measure object.

The standard measure object is transformed into a fuzzy measure object with the operator `set_fuzzy_measure`. As parameters, you pass the handle of the standard measure object, the fuzzy membership function, and the name of the feature to which the fuzzy membership function applies.

```
SetType := 'size'
set_fuzzy_measure (MeasureHandle, SetType, SizeFunction)
```

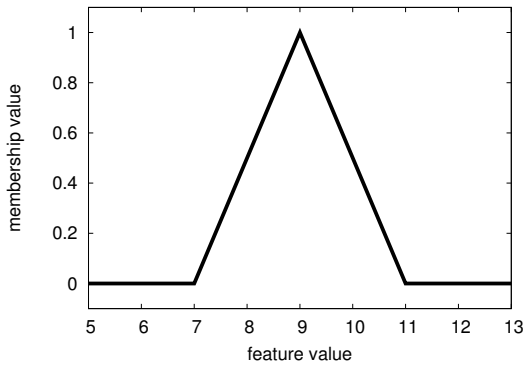


Figure 4.3: The fuzzy membership function that is used in the current example to describe the constraint on the pair width.

The resulting fuzzy measure object can be used to extract edge pairs that are approximately 9 pixels wide, with the term “approximately” being defined by the fuzzy membership function.

Then, the fuzzy measure object is applied to extract the desired edge pairs:

```
Sigma := 1.1
AmpThresh := 20
FuzzyThresh := 0.5
Transition := 'negative'
fuzzy_measure_pairs (Image, MeasureHandle, Sigma, AmpThresh, FuzzyThresh, \
                     Transition, RowEdgeFirst, ColumnEdgeFirst, \
                     AmplitudeFirst, RowEdgeSecond, ColumnEdgeSecond, \
                     AmplitudeSecond, RowEdgeCenter, ColumnEdgeCenter, \
                     FuzzyScore, IntraDistance, InterDistance)
```

Internally, the specified fuzzy membership function is used to weight all the possible edge pairs. The width of each possible edge pair is determined and transformed into a fuzzy value. If this fuzzy value lies above the user defined threshold `FuzzyThresh`, the edge pair is accepted, otherwise it is rejected. The result is displayed in figure 4.2b on page 28. With this, the width of all pins is measured correctly.

4.2 Controlling the Selection of Edges and Edge Pairs

The selection of edges and edge pairs can be controlled based on various features. Section 4.2.1 gives an overview over the possible features. The constraints on the feature values are defined with fuzzy membership functions. A short introduction into the concept of fuzzy logic and how fuzzy membership functions are created in HALCON is given in section 4.2.2.

4.2.1 Features that Can Be Used to Control the Selection of Edges and Edge Pairs

In the example program `%HALCONEXAMPLES%\solution_guide\1d_measuring\fuzzy_measure_switch.hdev` described above, the selection of edge pairs has been controlled by the approximately known pair size. There are further features according to which the edges or edge pairs can be selected. These features are grouped into so-called set types (parameter `SetType` of the operators `set_fuzzy_measure` and `set_fuzzy_measure_norm_pair`).

The set types that describe positions or sizes can be specified either with fuzzy membership functions that directly describe the constraint on the feature value like the one displayed in figure 4.3 on page 28 (see section 4.2.2.1) or with *normalized* fuzzy membership functions, i.e., fuzzy membership functions that are defined relative to the desired pair width (section 4.3.2 on page 32). The first method uses the operator `set_fuzzy_measure` to specify the fuzzy membership function. The second method uses the operator `set_fuzzy_measure_norm_pair`, which enables a generalized usage of the fuzzy membership function because the pair size is set independently of the fuzzy membership function.

The set types that refer to features of edge pairs will only take effect if edge pairs are determined, i.e., if the operators `fuzzy_measure_pairs` or `fuzzy_measure_pairing` are used.

Note that for the extraction of edge pairs all possible pairs of edges are internally analyzed according to the specified fuzzy membership functions. If only partial knowledge about the edge pairs to be determined is specified with the fuzzy membership functions, this might possibly lead to unintuitive results. For example, if the position of the edge pairs is specified but not their size, some small edge pairs might possibly be omitted because of a large edge pair that consists of the first edge of the first small edge pair and the second edge of the last small edge pair and that has a membership value equal to or higher than that of the small edge pairs. Therefore, all the available knowledge about the edge pairs to be extracted should be specified with fuzzy membership functions (see, e.g., appendix B.6 on page 59).

4.2.2 Defining the Constraints on the Feature Values with Fuzzy Membership Functions

The constraints for the selection of edges and edge pairs are defined with fuzzy membership functions. These functions transform the feature values into membership values. The decision whether an edge or an edge pair is accepted is made based on these membership values. For this so-called defuzzyfication, the membership values are compared with a given threshold.

This section describes the concept of fuzzy membership functions and how they are created in HALCON as well as the defuzzyfication.

4.2.2.1 Fuzzy Membership Functions

Fuzzy logic handles the concept of partial truth, i.e., truth values between "completely true" and "completely false". A fuzzy set is a set whose elements have degrees of membership. The degree of membership is defined by the feature value of the element and by the fuzzy membership function, which transforms the feature value into a membership value, i.e., the degree of membership.

With this, fuzzy logic provides a framework for the representation of knowledge that is afflicted with uncertainties or can only be described vaguely. This applies often to knowledge that is expressed in natural language, e.g., "the distance between the two edges of a pair is approximately 9 pixel," or "the edges are strong and they are located at the beginning of the profile line."

For example, a fuzzy membership function for the requirement that the edges must be strong could look like the one displayed in figure 4.4a. Here, any edge that has an amplitude of less than 50 will not be part of the fuzzy set of strong edges. Edges with an amplitude higher than 100 are full members of the fuzzy set, i.e, they have a membership value of 1.0. Edges with amplitudes between 50 and 100 are partial members of the fuzzy set, e.g., an edge with an amplitude of 75 will have a membership value of 0.5.

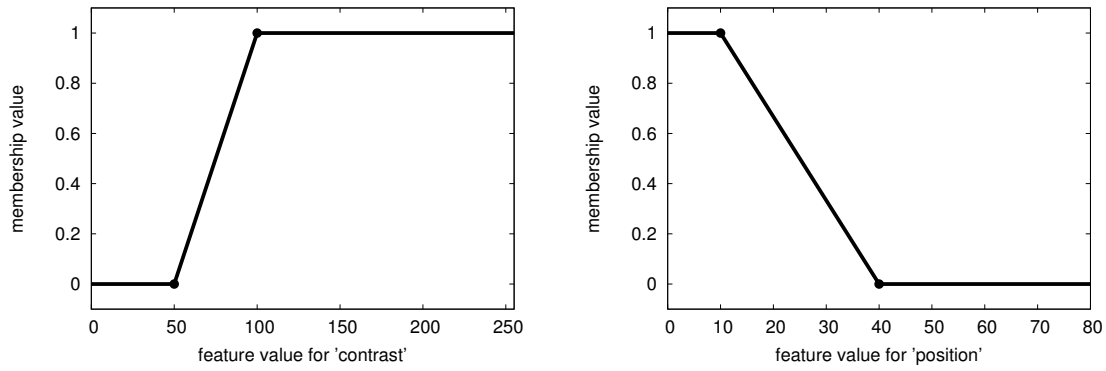


Figure 4.4: Examples for fuzzy membership functions. They can, e.g., be used to select (a) strong edges or (b) edges at the beginning of the profile line. The dots indicate the points to pass to `create_funct_1d_pairs`.

In general, a fuzzy membership function is a mathematical function that defines the degree of an element's membership in a fuzzy set.

In HALCON, fuzzy membership functions are defined as piecewise linear functions by at least two pairs of values, sorted in an ascending order by their x value. They are created with the operator `create_funct_1d_pairs`:

```
create_funct_1d_pairs (XValues, YValues, FuzzyMembershipFunction)
```

The x values represent the feature of the edges or edge pairs and must lie within the parameter space of the set type, i.e., in case of 'contrast' and 'gray' features and, e.g., byte images within the range $0.0 \leq x \leq 255.0$. For 'size', x has to satisfy $0.0 \leq x$, whereas for 'position' and 'position_pair', x can be any real number. The y values of the fuzzy membership function represent the resulting membership value for the corresponding feature value and

have to satisfy $0.0 \leq y \leq 1.0$. Outside of the function's interval, defined by the smallest and the largest specified x value, the y values of the interval borders are continued with constant values equal to the closest defined y value.

Therefore, the fuzzy membership function displayed in figure 4.4a is defined by the following code, which specifies the coordinates of the two control points of the fuzzy membership function.

```
create_funct_1d_pairs ([50,100], [0,1], FuzzyMembershipFunction1)
```

A fuzzy membership function describing the condition that the edge must be at the beginning of the profile line could look like the one displayed in figure 4.4b on page 30 (assuming that the profile line has a length of 80 pixels). It can be defined as follows:

```
Length := 80
create_funct_1d_pairs ([Length / 8.0,Length / 2.0], [1,0], \
                       FuzzyMembershipFunction2)
```

Further examples for fuzzy membership functions can be found in the appendix B on page 43.

4.2.2.2 Defuzzyfication

Finally, to decide whether an edge or an edge pair meets the constraint, the membership value must be "defuzzyfied". This is done by calculating a fuzzy score between 0 and 1 from the fuzzy membership values of all used fuzzy sets (as explained in section 4.3.3 on page 35), and comparing it against a user-defined fuzzy threshold. If the fuzzy score lies below the fuzzy threshold, the edge (pair) will be rejected. Otherwise, it will be accepted.

If, e.g., a fuzzy threshold of 0.5 is assumed as in figure 4.5, all edges that have a membership value of at least 0.5 will be accepted. For the fuzzy membership function displayed in figure 4.5a, these are all edges with a feature value of at least 75; for the fuzzy membership function of figure 4.5b, edges will only be accepted if their feature value is at most 25.

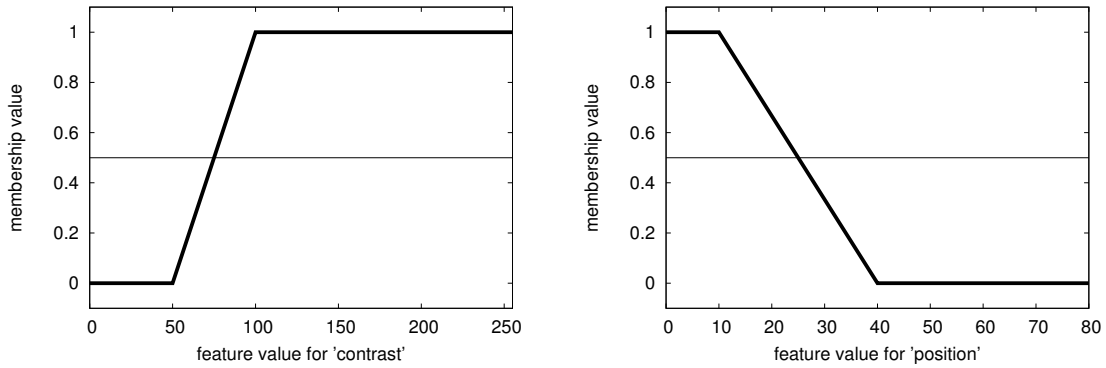


Figure 4.5: Examples for fuzzy membership functions (thick lines) together with a fuzzy threshold at 0.5 (thin lines).

The specification of the fuzzy membership function and the definition of the fuzzy threshold are separated for fuzzy measure objects. The fuzzy membership function is specified with the operator `set_fuzzy_measure` or `set_fuzzy_measure_norm_pair`, respectively, i.e., during the creation of the fuzzy measure object (see section 4.3). In contrast, the fuzzy threshold is defined when the fuzzy measure object is used with one of the operators `fuzzy_measure_pos`, `fuzzy_measure_pairs`, or `fuzzy_measure_pairing`. This separation between the specification of the fuzzy membership function and the definition of the fuzzy threshold allows a flexible use of the fuzzy measure object: Often, it will be sufficient to adjust only the fuzzy threshold to varying imaging conditions.

4.3 Creating the Fuzzy Measure Object

This section describes how to create a fuzzy measure object from an existing measure object. Fuzzy measure objects can be created by specifying a fuzzy membership function as described in section 4.3.1. Fuzzy measure

objects can also be created with normalized fuzzy membership functions, i.e., fuzzy membership functions that are defined relative to the desired pair width (see section 4.3.2). Section 4.3.3 on page 35 shows that it is even possible to specify multiple fuzzy membership functions for different set types of one fuzzy measure object.

4.3.1 Specifying a Fuzzy Membership Function

To transform a standard measure object into a fuzzy measure object, you specify a fuzzy membership function with the operator `set_fuzzy_measure`.

For example, you can create a fuzzy measure object for the extraction of edge pairs that have a size of approximately 10 pixels with the following lines of code:

```
create_funct_1d_pairs ([5,10,15], [0,1,0], \
                       FuzzyMembershipFunctionPairSize10)
set_fuzzy_measure (MeasureHandle, 'size', FuzzyMembershipFunctionPairSize10)
```

The fuzzy membership function used in this example is displayed in figure 4.6.

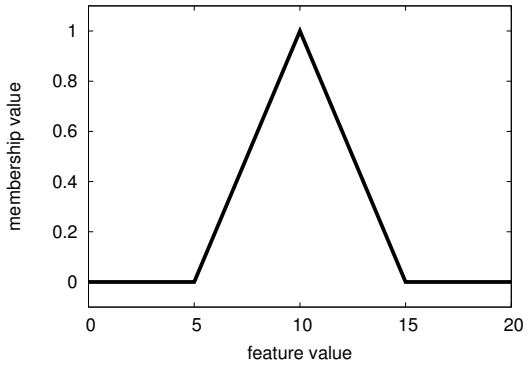


Figure 4.6: Fuzzy membership function, which can be used to extract edge pairs that are approximately 10 pixels wide.

4.3.2 Specifying a Normalized Fuzzy Membership Function

Fuzzy measure objects can also be created with normalized fuzzy membership functions, i.e., fuzzy membership functions that are defined relative to the desired pair width. For this, the operator `set_fuzzy_measure_norm_pair` must be used.

To create a fuzzy measure object that is identical with the one created in the above example (see section 4.3.1), the following code can be used:

```
create_funct_1d_pairs ([0.5,1,1.5], [0,1,0], \
                       FuzzyMembershipFunctionPairSizeNormalized)
PairSize := 10
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, 'size', \
                             FuzzyMembershipFunctionPairSizeNormalized)
```

Here, the normalized fuzzy membership function is used, which is displayed in figure 4.7. Internally, the operator `set_fuzzy_measure_norm_pair` multiplies the x values of the normalized fuzzy membership function with the given pair size.

One major advantage of using the normalized form of fuzzy membership functions is that the application can very easily be adapted to varying sizes of the object to be measured. If, e.g., larger objects must be measured or if a camera with a higher resolution is used, it will suffice to adapt the pair size.

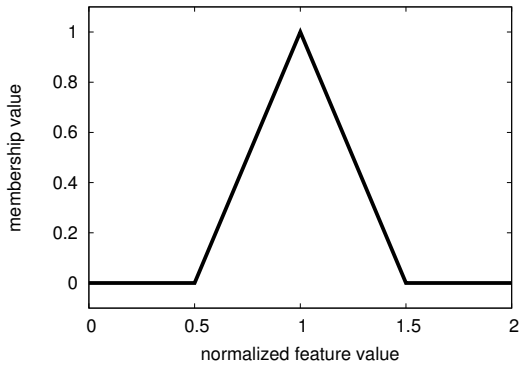


Figure 4.7: Normalized fuzzy membership function, which can be used for the extraction of edge pairs of various sizes, according to the defined pair size.

If, e.g., the above example for the extraction of 10 pixel wide edge pairs must be adapted to the extraction of 20 pixel wide edge pairs, simply the definition of the pair size must be changed:

```
PairSize := 20
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, 'size', \
                             FuzzyMembershipFunctionPairSizeNormalized)
```

This creates a fuzzy measure object that uses the fuzzy membership function displayed in figure 4.8, internally.

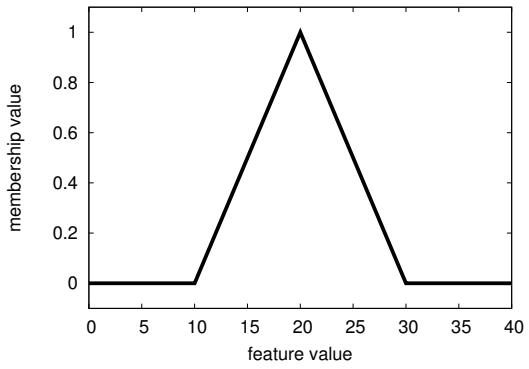


Figure 4.8: Fuzzy membership function, which can be used for the extraction of approximately 20 pixel wide edge pairs.

To achieve such a fuzzy measure object without normalized fuzzy membership functions, the definition of the fuzzy membership function itself must be adapted:

```
create_funct_1d_pairs ([10,20,30], [0,1,0], \
                       FuzzyMembershipFunctionPairSize20)
set_fuzzy_measure (MeasureHandle, 'size', FuzzyMembershipFunctionPairSize20)
```

In the following, a list of the possible set types is given. It is possible to use multiple set types. Details can be found in section 4.3.3 on page 35. Note that the subtypes of each set are mutually exclusive, i.e., only one subtype of each set can be specified for a fuzzy measure object.

Set Type	Subtype	Function Type		Short Description
		Direct	Norm.	
contrast	*'contrast'*	•	-	Evaluates the amplitude of the edges.
position	*'position'*	•	•	Evaluates the signed distance of the edges to the reference point of the fuzzy measure object. The reference point is located at the start of the profile line (see figure 2.7 on page 13 and figure 2.8 on page 13).
	'position_center'	•	•	Like *'position'* with the reference point located at the center of the profile line.
	'position_end'	•	•	Like *'position'* with the reference point located at the end of the profile line.
	'position_first_edge'	•	•	Like *'position'* with the reference point located at the first edge.
	'position_last_edge'	•	•	Like *'position'* with the reference point located at the last edge.
position_pair (for edge pairs only)	*'position_pair'*	•	•	Evaluates the signed distance of the edge pairs to the reference point of the fuzzy measure object. The position of an edge pair is defined as the center of the two edges. The reference point is located at the start of the profile line (see figure 2.7 on page 13 and figure 2.8 on page 13).
	'position_pair_center'	•	•	Like *'position_pair'* with the reference point located at the center of the profile line.
	'position_pair_end'	•	•	Like *'position_pair'* with the reference point located at the end of the profile line.
	'position_first_pair'	•	•	Like *'position_pair'* with the reference point located at the position of the first edge pair.
	'position_last_pair'	•	•	Like *'position_pair'* with the reference point located at the position of the last edge pair.
size (for edge pairs only)	*'size'*	•	•	Evaluates the size of the edge pairs, i.e., the distance between the two edges.
	'size_diff'	-	•	Evaluates the signed difference between the desired `PairSize` and the actual size of the edge pairs.
	'size_abs_diff'	-	•	Evaluates the absolute difference between the desired `PairSize` and the actual size of the edge pairs.
gray (for edge pairs only)	*'gray'*	•	-	Evaluates the mean gray value between the two edges of edge pairs.

4.3.3 Specifying Multiple Fuzzy Membership Functions

You can also specify fuzzy membership functions for different set types by repeatedly calling the operator `set_fuzzy_measure` or `set_fuzzy_measure_norm_pair`, respectively:

```
set_fuzzy_measure (MeasureHandle, 'contrast', FuzzyMembershipFunction1)
set_fuzzy_measure (MeasureHandle, 'position', FuzzyMembershipFunction2)
```

If more than one set is defined in this way, the individual membership values will be aggregated with a fuzzy and operator: The overall membership value is derived by the geometric mean (see appendix C on page 61) of the individual membership values of each set.

For example, the two set types *'contrast'* and *'position'* are used to achieve the selection of strong edges at the beginning of the profile line. For this, the two fuzzy membership functions as displayed in figure 4.4a and b on page 30 are used.

Assuming an edge at the position 20 along the profile line that has an amplitude of 80, the membership value is determined as follows. First, the membership values of the individual fuzzy sets are determined:

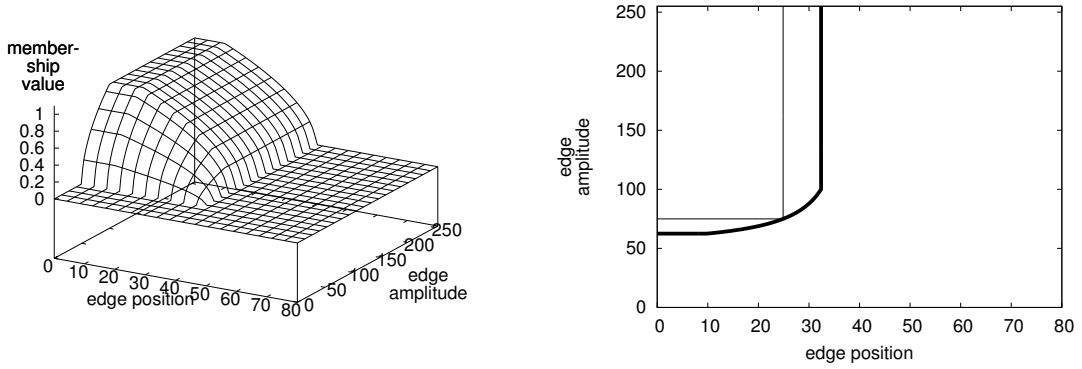
$$\begin{aligned} \text{membership value}_{position=20} &= 0.67 \\ \text{membership value}_{contrast=80} &= 0.60 \end{aligned}$$

Then, the geometric mean is calculated:

$$\begin{aligned} \text{membership value}_{20/80} &= \sqrt{\text{membership value}_{position=20} \cdot \text{membership value}_{contrast=80}} \\ &= 0.63 \end{aligned}$$

This value is compared to the fuzzy threshold. If it lies above this threshold, the respective edge will be accepted, otherwise it will be rejected.

Figure 4.9a shows a plot of the membership value over the two feature values for *'contrast'* and *'position'*. Assuming a fuzzy threshold of 0.5, the thick line in figure 4.9b marks the boundary of the domain of the feature values that lead to an acceptance of the respective edge. All edges with position/amplitude combinations that lie in this domain in the upper left corner will be accepted. The thin line marks the domain that results if both constraints must be satisfied individually. As can be seen, the domain that results from the combination of several set types is a bit larger. In particular, if one feature value meets the requirement very well, the other value may be slightly worse than allowed by the respective individual constraint.



(a) (b)

Figure 4.9: Membership value as a function of two feature values: (a) The membership value; (b) the boundary of the domain where edges will be accepted (thick line) compared to the boundary of the domain if the two constraints must be satisfied individually.

To illustrate this, an edge at the position 10 having an amplitude of 65 is assumed. The individual membership values are:

$$\begin{aligned} \text{membership value}_{position=10} &= 1.00 \\ \text{membership value}_{contrast=65} &= 0.30 \end{aligned}$$

With this, the membership value can be calculated:

$$\begin{aligned} \text{membership value}_{10/65} &= \sqrt{\text{membership value}_{position=10} \cdot \text{membership value}_{contrast=65}} \\ &= 0.54 \end{aligned}$$

The overall membership value lies above 0.5 although the individual membership value for the contrast of the edge lies well below this value.

Note that it is not possible to specify multiple fuzzy membership functions for one and the same set type. Setting a further fuzzy membership function to a set discards the previously defined function and replaces it by the new one.

4.3.4 Changing and Resetting a Fuzzy Measure Object

Fuzzy membership functions that have been specified for a particular set type can be changed and reset. To change a fuzzy membership function for a set type, simply specify the new fuzzy membership function for this set type with one of the operators `set_fuzzy_measure` and `set_fuzzy_measure_norm_pair`, as described in section 4.3.1 on page 32.

A fuzzy measure object can be reset with the operator `reset_fuzzy_measure`. This discards the fuzzy membership function of the specified fuzzy set `SetType`. For example, the fuzzy membership function of the set type *'contrast'* is discarded with the following lines of code:

```
SetType := 'contrast'
reset_fuzzy_measure (MeasureHandle, SetType)
```

4.4 Using the Standard Measure Object Functionality

All operators that provide functionality for the standard measure object can also be applied to fuzzy measure objects. This includes especially the operators `measure_projection`, `measure_thresh`, `translate_measure`, and `close_measure`. The operators for the standard measure object ignore the specified fuzzy membership functions and return the same results as if they were applied to a standard measure object.

Appendix A

Slanted Edges

This section shows the effect of the width of the ROI and the angle between the profile line and the edges. In the figures on the next pages, the images on the left hand show the size and orientation of the ROI and the diagrams on the right hand show the resulting profile (thick line) and its derivative (thin line, `Sigma` = 1.1).

As long as the profile line runs approximately perpendicular to the edges, the ROI should be chosen as wide as possible because then the profile is less noisy (compare, e.g., figure A.2a and figure A.2c).

If the profile line does not run perpendicular to the edges (see figure A.3 on page 39 and figure A.4 on page 40), the width of the ROI must be chosen smaller to allow the detection of edges. Note that in this case, the profile will contain more noise. Consequently, the edges will be determined less accurately. What is more, the distance between the determined edges does not represent the perpendicular, i.e., the shortest distance between the respective image edges (see figure A.1). If the angle δ between the profile line and the perpendicular of the edges is known, the perpendicular distance can be determined from the distance between the determined edges (`Distance`) as follows:

$$\text{perpendicular distance} \quad = \quad \cos(\delta) \cdot \texttt{Distance}$$

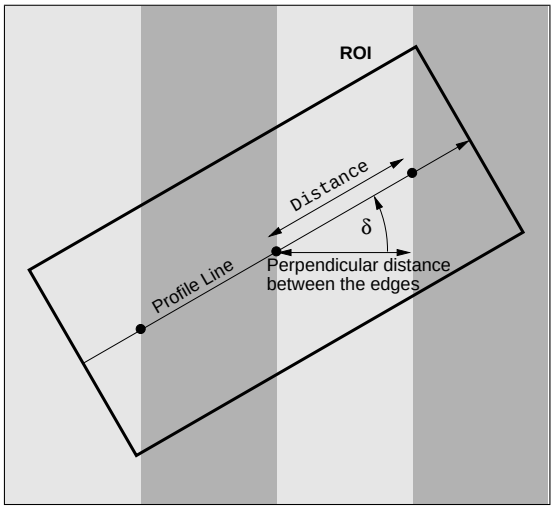
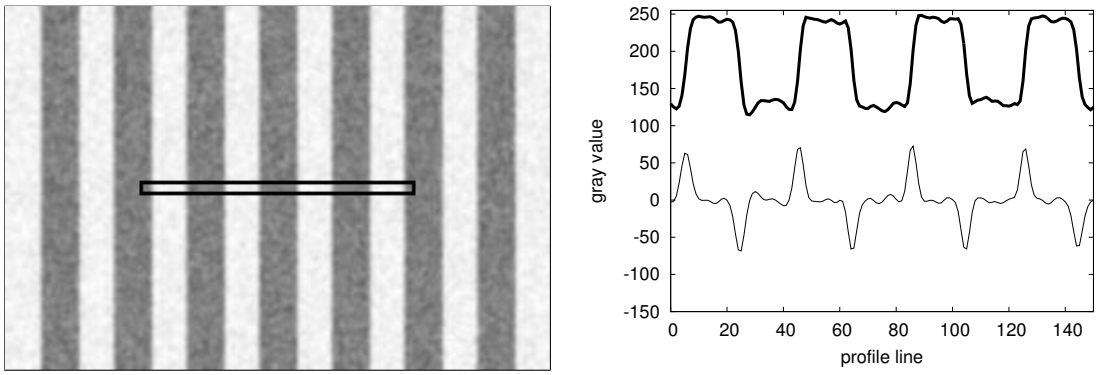
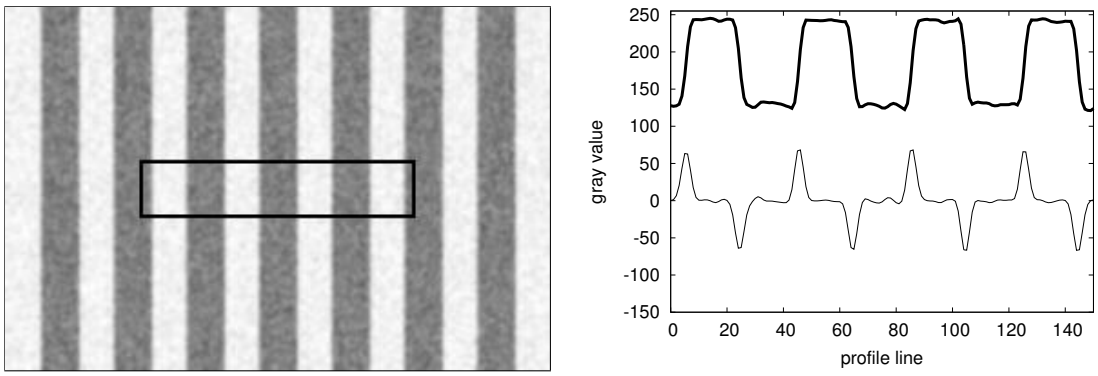


Figure A.1: Relationship between the determined `Distance` and the perpendicular distance between edges.

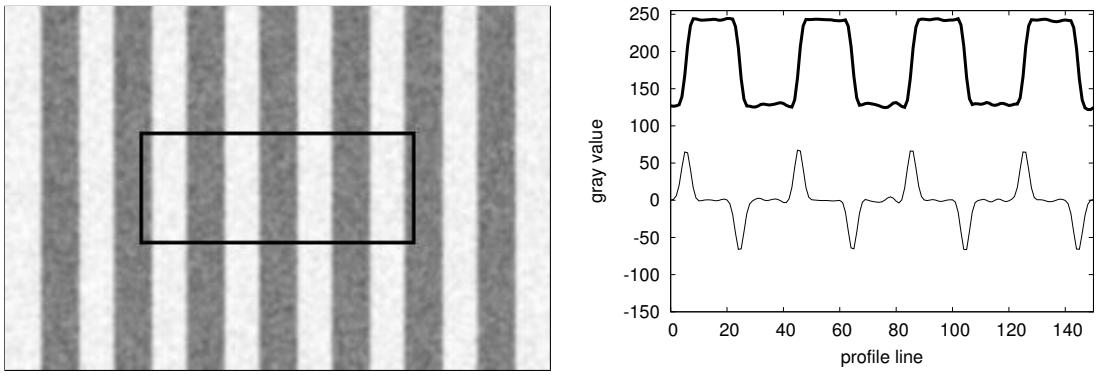
If the profile line is heavily slanted with respect to the edges, it may become impossible to determine the edges reliably, even with a very narrow ROI (see, e.g., figure A.5a on page 41).



(a) Width of the ROI: 6 pixels.

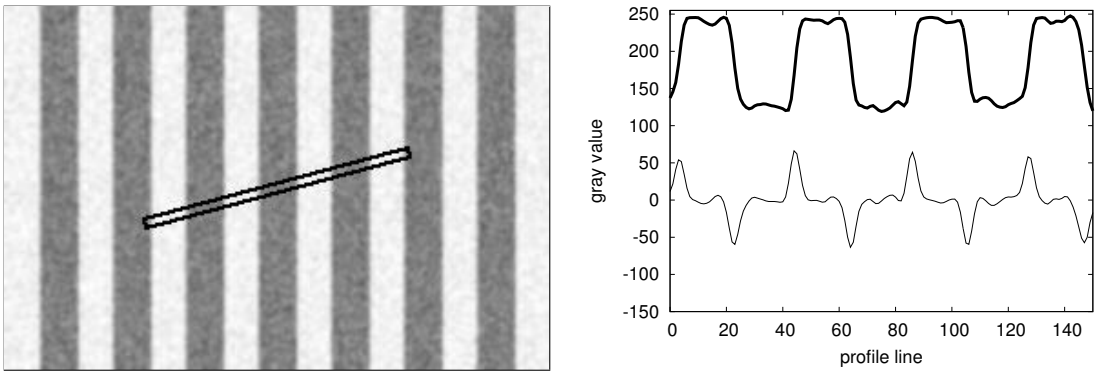


(b) Width of the ROI: 30 pixels.

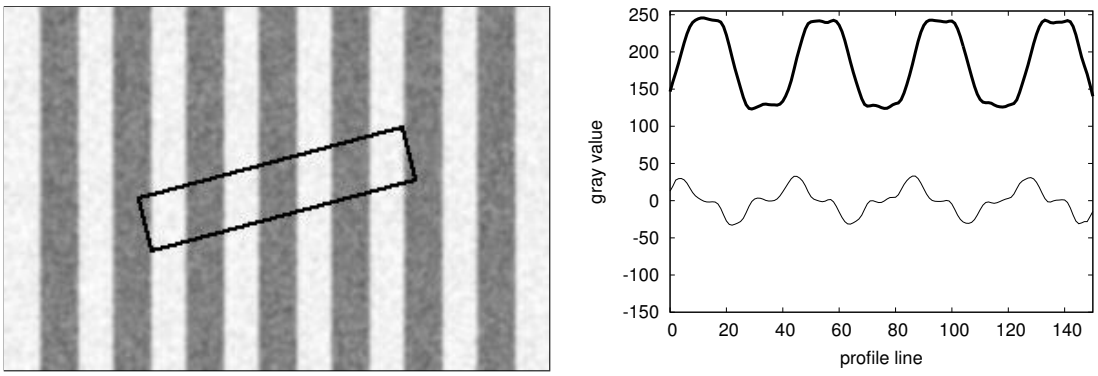


(c) Width of the ROI: 60 pixels.

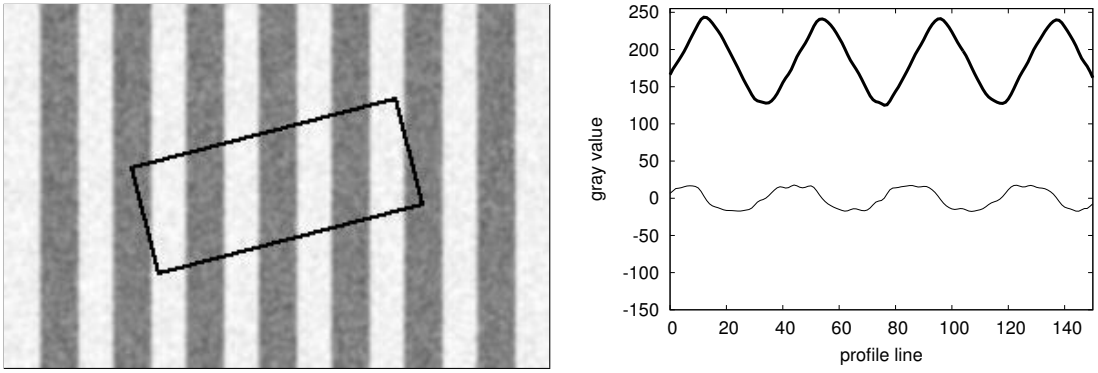
Figure A.2: Angle between the profile line and the perpendicular of the edges: 0°.



(a) Width of the ROI: 6 pixels.

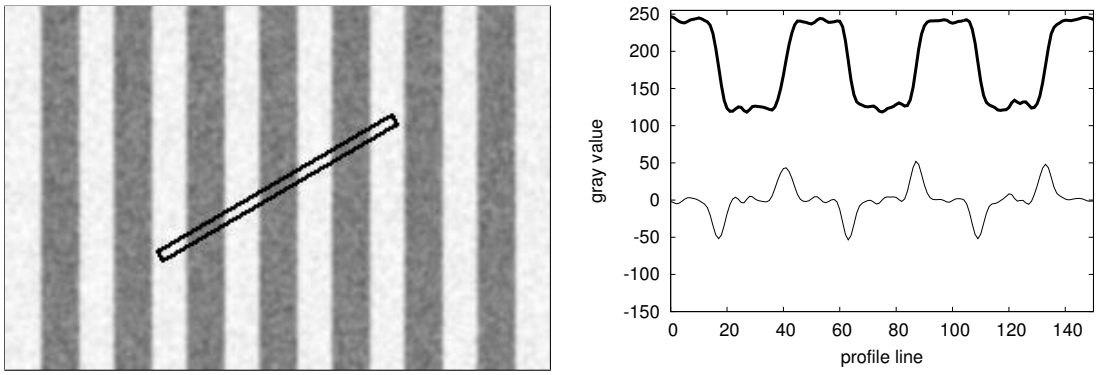


(b) Width of the ROI: 30 pixels.

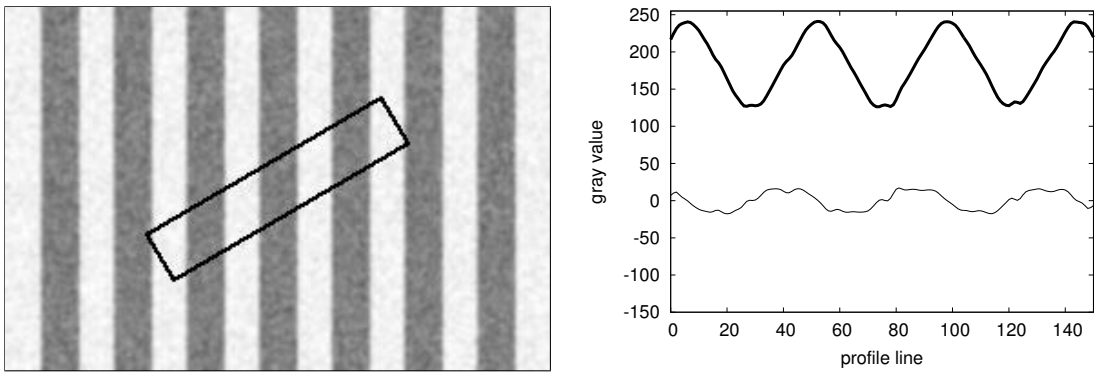


(c) Width of the ROI: 60 pixels.

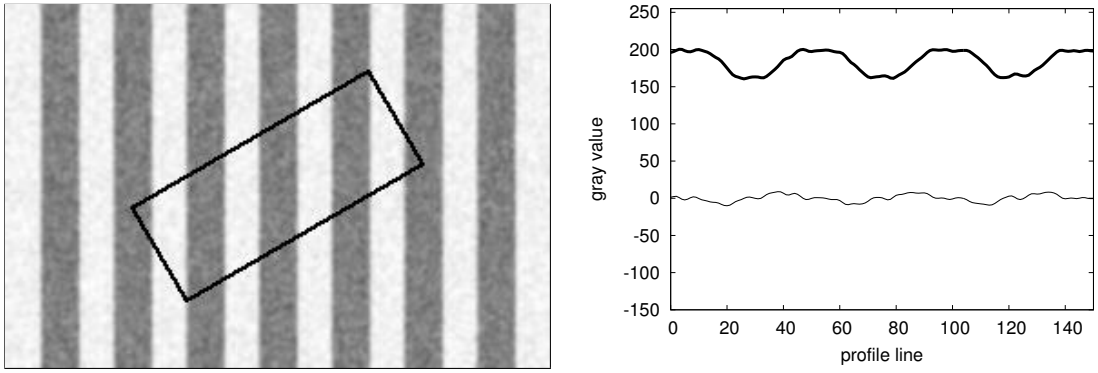
Figure A.3: Angle between the profile line and the perpendicular of the edges: 15°.



(a) Width of the ROI: 6 pixels.

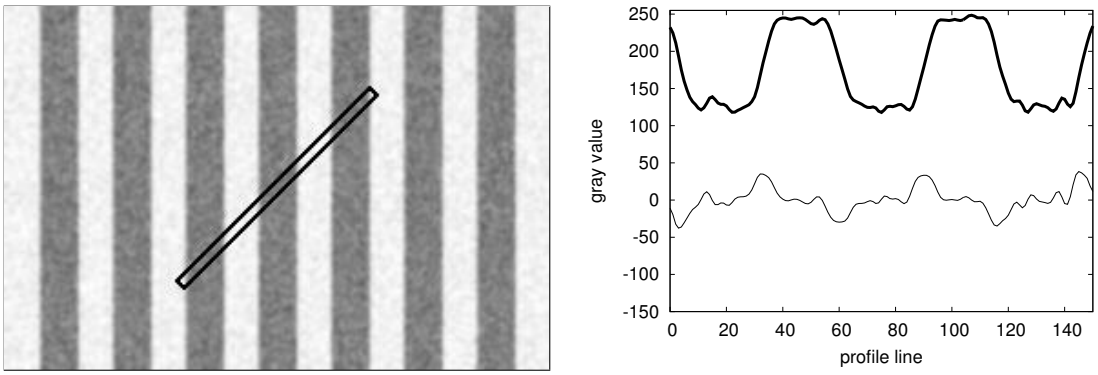


(b) Width of the ROI: 30 pixels.

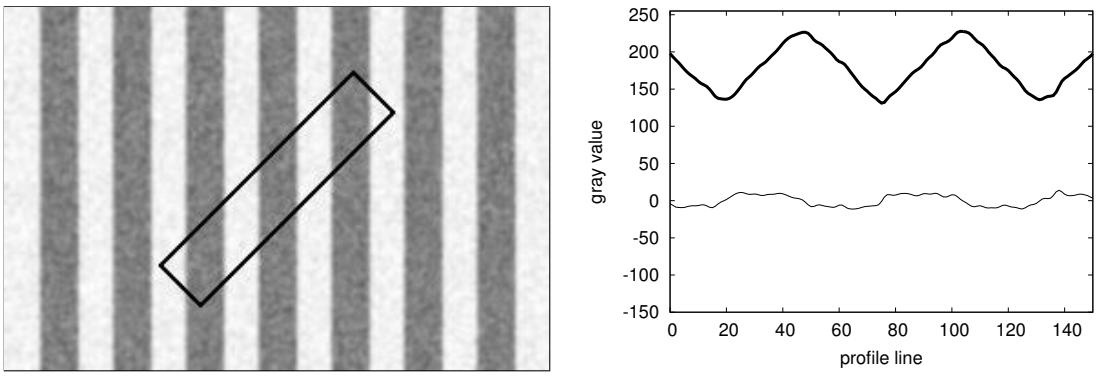


(c) Width of the ROI: 60 pixels.

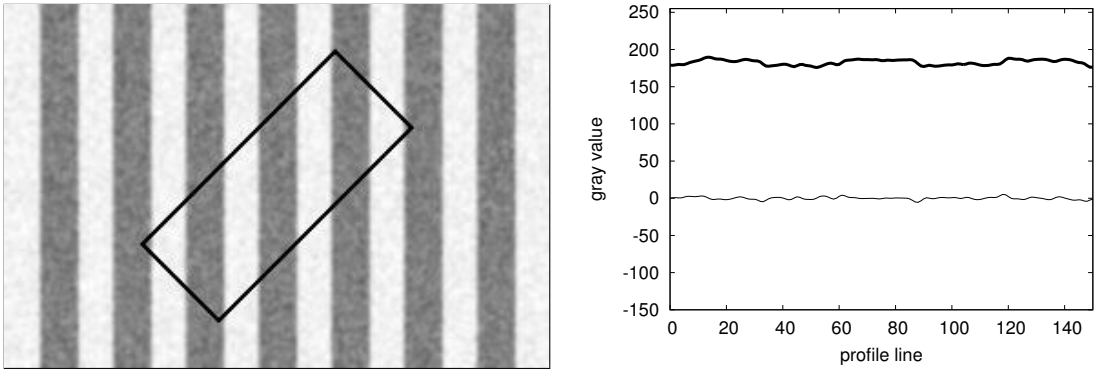
Figure A.4: Angle between the profile line and the perpendicular of the edges: 30°.



(a) Width of the ROI: 6 pixels.



(b) Width of the ROI: 30 pixels.



(c) Width of the ROI: 60 pixels.

Figure A.5: Angle between the profile line and the perpendicular of the edges: 45°.

Appendix B

Examples for Fuzzy Membership Functions

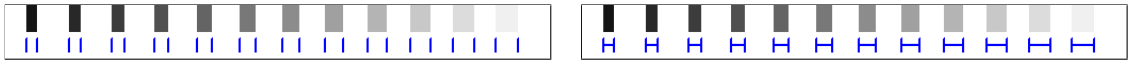
In this section, examples for fuzzy membership functions are given. For each set type (see section 4.2.1 on page 29), the typical range of the x values of the fuzzy membership functions as well as one fuzzy membership function is given. If the set type allows the creation of the fuzzy measure object with fuzzy membership functions *and* normalized fuzzy membership functions, both fuzzy membership functions are displayed. In this case, the normalized fuzzy membership function is defined such that, together with the stated `PairSize`, it creates the same fuzzy measure object as the direct fuzzy membership function. Then, mostly a `PairSize` of 10.0 is used to make it easy to compare the two fuzzy membership functions.

The effect of the specification of the fuzzy membership function is shown by means of the results of the fuzzy measure object applied to the test image shown in figure B.1.



Figure B.1: The test image that is used to show the effect of the different fuzzy membership functions.

The results of the standard measure object applied to this test image are shown in figure B.2.



(a) (b)

Figure B.2: Result of the standard measure object applied to the test image: (a) Edges determined with the operator `measure_pos`, (b) edge pairs determined with the operator `measure_pairs`.

Figure B.2a shows the edges determined with the operator `measure_pos` and figure B.2b shows the edge pairs determined with the operator `measure_pairs`. Both operators were applied with the following parameter settings:

Parameter	Value
`Sigma`	*1.0*
`Threshold`	*10.0*
`Transition`	*'all'*
`Select`	*'all'*
`FuzzyThresh`	*0.5*

The following examples shall give an impression of the effect of different fuzzy membership functions specified for different set types. They are in no way complete. Note that all the edge pairs are extracted with the operator `fuzzy_measure_pairing`, not with the operator `fuzzy_measure_pairs`.

B.1 Set Type *contrast*

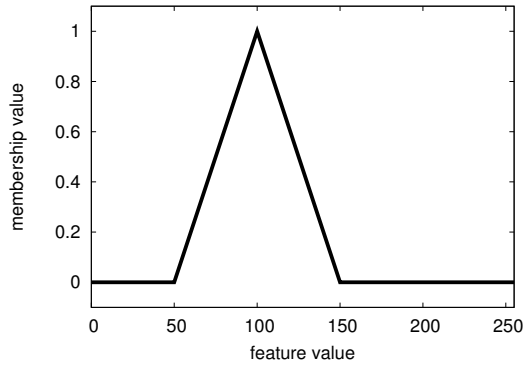
B.1.1 Subtype *'contrast'*

Evaluates the amplitude of the edges.

The x values of fuzzy membership functions for the sub set type *'contrast'* must lie within the range of gray values, e.g.,

$0.0 \leq x \leq 255.0$ for *'byte'* images and
$0.0 \leq x \leq 65535.0$ for *'uint2'* images.

Figure B.3 shows a fuzzy membership function that can be used to extract fairly strong edges in *'byte'* images.



(a)

The sub set type *'contrast'* cannot be specified with a normalized fuzzy membership function.

(b)

Figure B.3: A fuzzy membership function that can be used to extract fairly strong edges in *'byte'* images (`SetType` = *'contrast'*).

Figure B.4 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.3. With this fuzzy measure object, only those edges are extracted that have a contrast between 75 and 125.

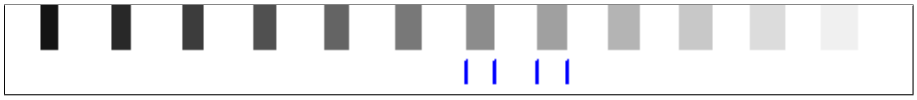


Figure B.4: The extracted edges.

The fuzzy membership function displayed in figure B.3a can be created with the following HDevelop code:

```
SetType := 'contrast'
create_funct_1d_pairs ([50,100,150], [0,1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

B.2 Set Type *position*

B.2.1 Subtype *'position'*

Evaluates the signed distance of the edges to the reference point of the fuzzy measure object. The reference point is located at the start of the profile line (see figure 2.7 on page 13 and figure 2.8 on page 13).

The x values of fuzzy membership functions for the sub set type *'position'* must lie within the range

$$0.0 \leq x \leq \text{length of the ROI.}$$

Figure B.5 shows fuzzy membership functions that can be used to extract edges at the beginning of the profile line.

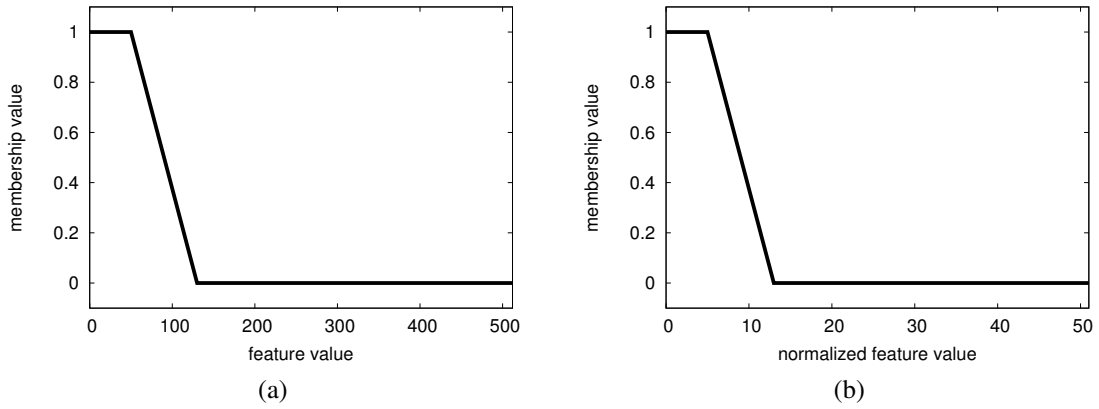


Figure B.5: Fuzzy membership functions that can be used to extract edges at the beginning of the profile line (`SetType` = *'position'*, `PairSize` = 10).

Figure B.6 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.5. Only the first four edges are returned.

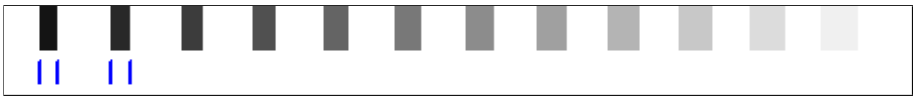


Figure B.6: The extracted edges.

The fuzzy membership function displayed in figure B.5a can be created with the following HDevelop code:

```
SetType := 'position'
create_funct_1d_pairs ([50,130], [1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.5b can be created with the following HDevelop code:

```
SetType := 'position'
PairSize := 10
create_funct_1d_pairs ([5,13], [1,0], NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

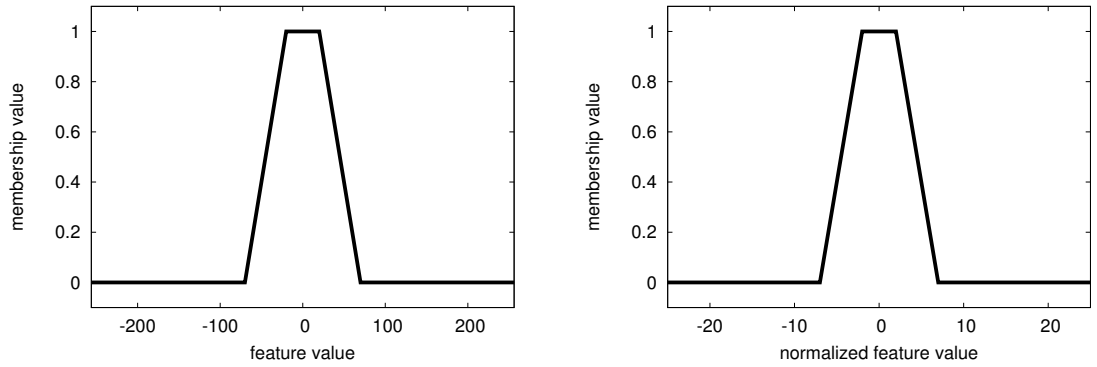
B.2.2 Subtype *'position_center'*

'position_center' behaves like *'position'* with the reference point located at the center of the profile line.

The x values of fuzzy membership functions for the sub set type *'position_center'* must lie within the range

$$-\frac{\text{length of the ROI}}{2} \leq x \leq \frac{\text{length of the ROI}}{2}.$$

Figure B.7 shows fuzzy membership functions that can be used to extract edges in the center of the profile line.



(a) (b)

Figure B.7: Fuzzy membership functions that can be used to extract edges in the center of the profile line (`SetType` = *'position_center'*, `PairSize` = 10).

Figure B.8 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.7. Only the edges in the center of the profile line are returned.

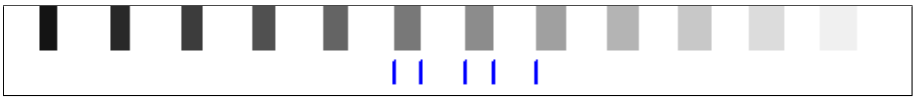


Figure B.8: The extracted edges.

The fuzzy membership function displayed in figure B.7a can be created with the following HDevelop code:

```
SetType := 'position_center'
create_funct_1d_pairs ([-70,-20,20,70], [0,1,1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.7b can be created with the following HDevelop code:

```
SetType := 'position_center'
PairSize := 10
create_funct_1d_pairs ([-7,-2,2,7], [0,1,1,0], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

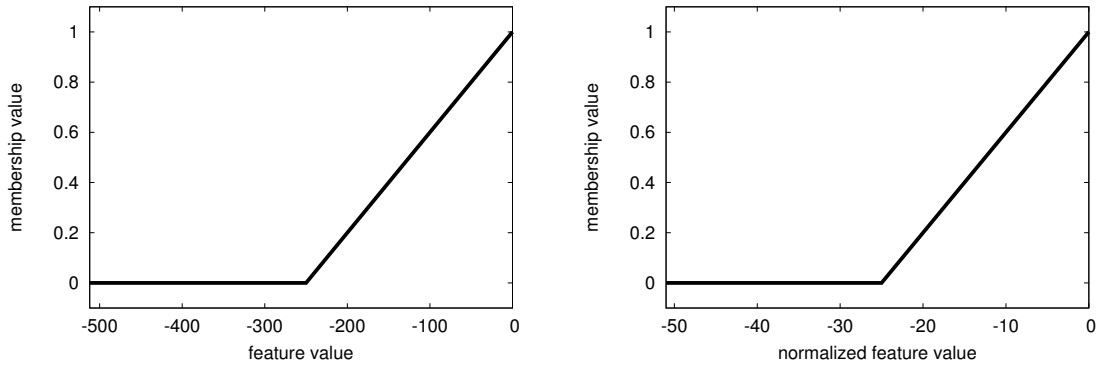
B.2.3 Subtype *'position_end'*

'position_end' behaves like *'position'* with the reference point located at the end of the profile line.

The x values of fuzzy membership functions for the sub set type *'position_end'* must lie within the range

$$-\text{length of the ROI} \leq x \leq 0.0.$$

Figure B.9 shows fuzzy membership functions that can be used to extract edges at the end of the profile line.



(a) (b)

Figure B.9: Fuzzy membership functions that can be used to extract edges at the end of the profile line (`SetType` = *'position_end'*, `PairSize` = 10).

Figure B.10 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.9. Only the edges at the end of the profile line are returned.



Figure B.10: The extracted edges.

The fuzzy membership function displayed in figure B.9a can be created with the following HDevelop code:

```
SetType := 'position_end'
create_funct_1d_pairs ([-250,0], [0,1], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.9b can be created with the following HDevelop code:

```
SetType := 'position_end'
PairSize := 10
create_funct_1d_pairs ([-25,0], [0,1], NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

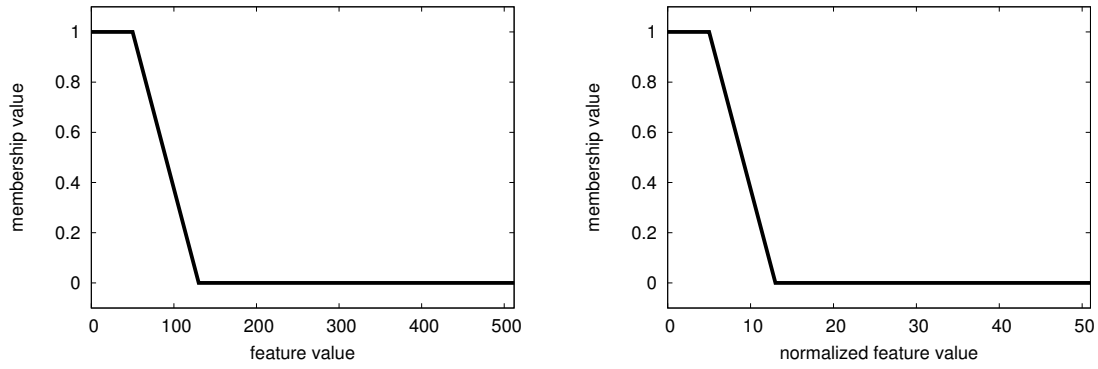
B.2.4 Subtype *'position_first_edge'*

'position_first_edge' behaves like *'position'* with the reference point located at the first edge.

The x values of fuzzy membership functions for the sub set type *'position_first_edge'* must lie within the range

$$0.0 \leq x \leq \text{length of the ROI.}$$

Figure B.11 shows fuzzy membership functions that can be used to extract edges close to the first edge.



(a) (b)

Figure B.11: Fuzzy membership functions that can be used to extract edges close to the first edge (`SetType` = *'position_first_edge'*, `PairSize` = 10).

Figure B.12 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.11. Only the edges at the beginning of the profile line are returned. Note that even though the same fuzzy membership function has been used as in figure B.5 on page 45, one more edge has been extracted. This is because the reference point is located at the first edge instead of the start of the profile line.



Figure B.12: The extracted edges.

The fuzzy membership function displayed in figure B.11a can be created with the following HDevelop code:

```
SetType := 'position_first_edge'
create_funct_1d_pairs ([50,130], [1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.11b can be created with the following HDevelop code:

```
SetType := 'position_first_edge'
PairSize := 10
create_funct_1d_pairs ([5,13], [1,0], NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.2.5 Subtype *'position_last_edge'*

'position_last_edge' behaves like *'position'* with the reference point located at the last edge.

The x values of fuzzy membership functions for the sub set type *'position_last_edge'* must lie within the range

$$-\text{length of the ROI} \leq x \leq 0.0.$$

Figure B.13 shows fuzzy membership functions that can be used to extract edges close to the last edge.

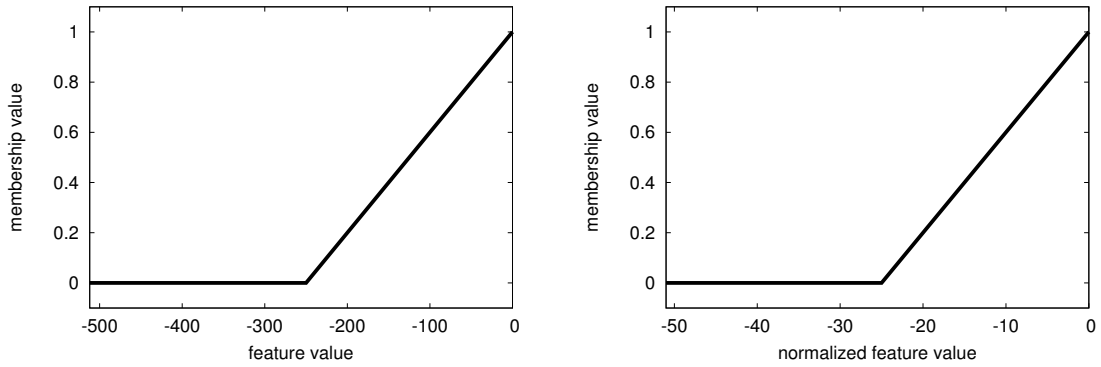


Figure B.13: Fuzzy membership functions that can be used to extract edges close to the last edge (`SetType` = *'position_last_edge'*, `PairSize` = 10).

Figure B.14 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.13. Only the edges at the end of the profile line are returned. Note that even though the same fuzzy membership function has been used as in figure B.9 on page 47, more edges have been extracted. This is because the reference point is located at the last edge instead of the end of the profile line.

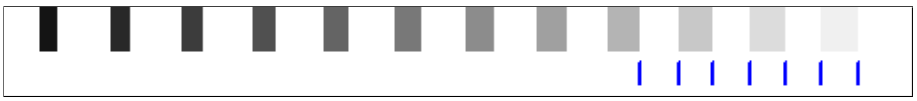


Figure B.14: The extracted edges.

The fuzzy membership function displayed in figure B.13a can be created with the following HDevelop code:

```
SetType := 'position_last_edge'
create_funct_1d_pairs ([-250,0], [0,1], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.13b can be created with the following HDevelop code:

```
SetType := 'position_last_edge'
PairSize := 10
create_funct_1d_pairs ([-25,0], [0,1], NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.3 Set Type *position_pair*

Note that for the extraction of edge pairs all possible pairs of edges are internally analyzed according to the specified fuzzy membership functions. If only partial knowledge about the edge pairs to be determined is specified with the fuzzy membership functions, this might possibly lead to unintuitive results. For example, if the position of the edge pairs is specified but not their size, some small edge pairs might possibly be omitted because of a large edge pair that consists of the first edge of the first small edge pair and the second edge of the last small edge pair and that has a membership value equal to or higher than that of the small edge pairs. Therefore, all the available knowledge about the edge pairs to be extracted should be specified with fuzzy membership functions (see, e.g., appendix B.6 on page 59).

B.3.1 Subtype *'position_pair'*

Evaluates the signed distance of the edge pairs to the reference point of the fuzzy measure object. The position of an edge pair is defined as the center of the two edges. The reference point is located at the start of the profile line (see figure 2.7 on page 13 and figure 2.8 on page 13).

The x values of fuzzy membership functions for the sub set type *'position_pair'* must lie within the range

$$0.0 \leq \text{x} \leq \text{length of the ROI.}$$

Figure B.15 shows fuzzy membership functions that can be used to extract edge pairs in approximately the first half of the profile line.

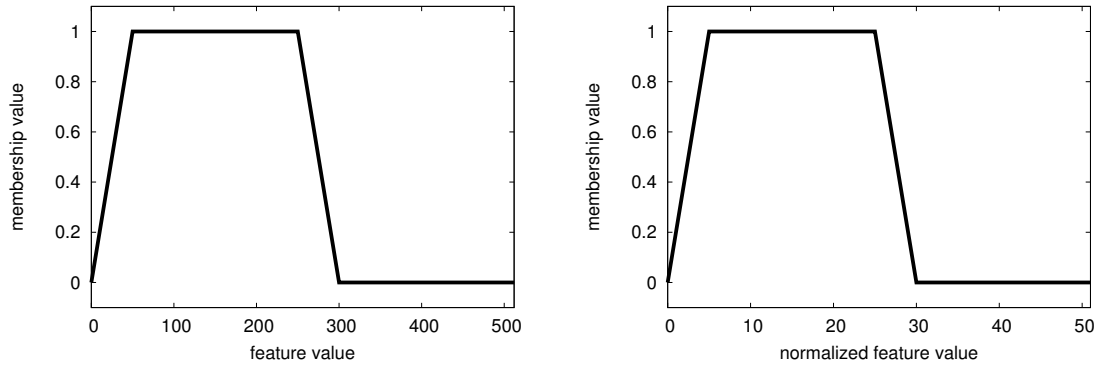


Figure B.15: Fuzzy membership functions that can be used to extract edge pairs in approximately the first half of the profile line (`SetType` = *'position_pair'*, `PairSize` = 10).

Figure B.16 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.15. Only edge pairs in approximately the first half of the profile line are returned. See appendix B.6 on page 59 for an example where in addition a fuzzy membership function for the size of the edge pairs is specified.



Figure B.16: The extracted edge pairs.

The fuzzy membership function displayed in figure B.15a can be created with the following HDevelop code:

```
SetType := 'position_pair'
create_funct_1d_pairs ([0,50,250,300], [0,1,1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.15b can be created with the following HDevelop code:

```
SetType := 'position_pair'
PairSize := 10
create_funct_1d_pairs ([0,5,25,30], [0,1,1,0], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.3.2 Subtype *'position_pair_center'*

'position_pair_center' behaves like *'position_pair'* with the reference point located at the center of the profile line.

The x values of fuzzy membership functions for the sub set type *'position_pair_center'* must lie within the range

$$-\frac{\text{length of the ROI}}{2} \leq \text{x} \leq \frac{\text{length of the ROI}}{2}.$$

Figure B.17 shows fuzzy membership functions that can be used to extract edge pairs in the middle of the profile line.

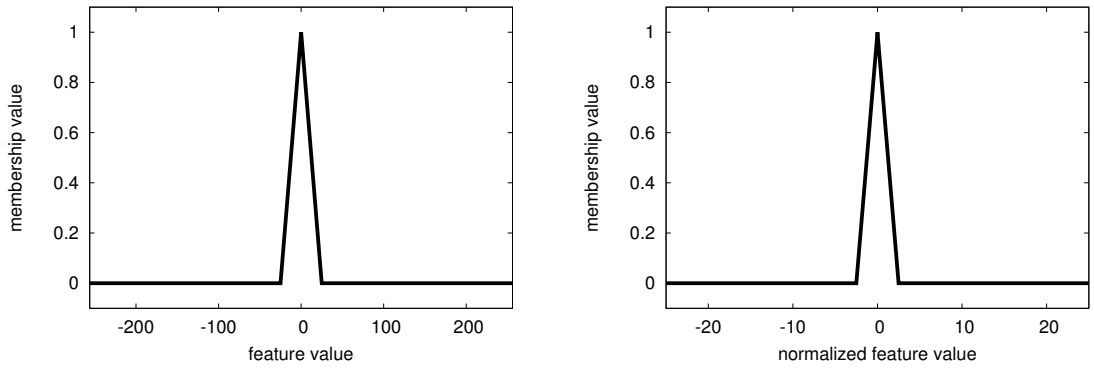


Figure B.17: Fuzzy membership functions that can be used to extract edge pairs in the middle of the profile line (`SetType` = *'position_pair_center'*, `PairSize` = 10).

Figure B.18 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.17. Only one edge pair in the middle of the profile line is returned. Note that only the position of the edge pair, i.e., the position of the center between the two edges of the edge pair is used to restrict the extraction of edge pairs. The position of the individual edges of the edge pair is not restricted. In this case, one large edge pair with its center in the middle of the profile line is returned.



Figure B.18: The extracted edge pairs.

The fuzzy membership function displayed in figure B.17a can be created with the following HDevelop code:

```
SetType := 'position_pair_center'
create_funct_1d_pairs ([-25,0,25], [0,1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.17b can be created with the following HDevelop code:

```
SetType := 'position_pair_center'
PairSize := 10
create_funct_1d_pairs ([-2.5,0,2.5], [0,1,0], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

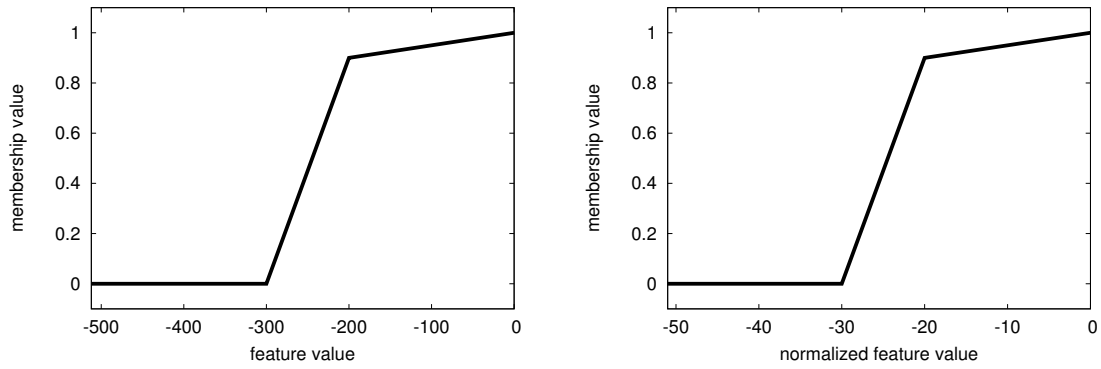
B.3.3 Subtype *'position_pair_end'*

'position_pair_end' behaves like *'position_pair'* with the reference point located at the end of the profile line.

The x values of fuzzy membership functions for the sub set type *'position_pair_end'* must lie within the range

$$-\text{length of the ROI} \leq x \leq 0.0.$$

Figure B.19 shows fuzzy membership functions that can be used to extract edge pairs at the end of the profile line.



(a) (b)

Figure B.19: Fuzzy membership functions that can be used to extract edge pairs at the end of the profile line (`SetType` = *'position_pair_end'*, `PairSize` = 10).

Figure B.20 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.19. Only edge pairs at the end of the profile line are returned.



Figure B.20: The extracted edge pairs.

The fuzzy membership function displayed in figure B.19a can be created with the following HDevelop code:

```
SetType := 'position_pair_end'
create_funct_1d_pairs ([-300,-200,0], [0,0.9,1], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.19b can be created with the following HDevelop code:

```
SetType := 'position_pair_end'
PairSize := 10
create_funct_1d_pairs ([-30,-20,0], [0,0.9,1], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.3.4 Subtype *'position_first_pair'*

'position_first_pair' behaves like *'position_pair'* with the reference point located at the position of the first edge pair.

The x values of fuzzy membership functions for the sub set type *'position_first_pair'* must lie within the range

$$0.0 \leq x \leq \text{length of the ROI.}$$

Figure B.21 shows fuzzy membership functions that can be used to extract edge pairs in approximately the first half of the profile line.

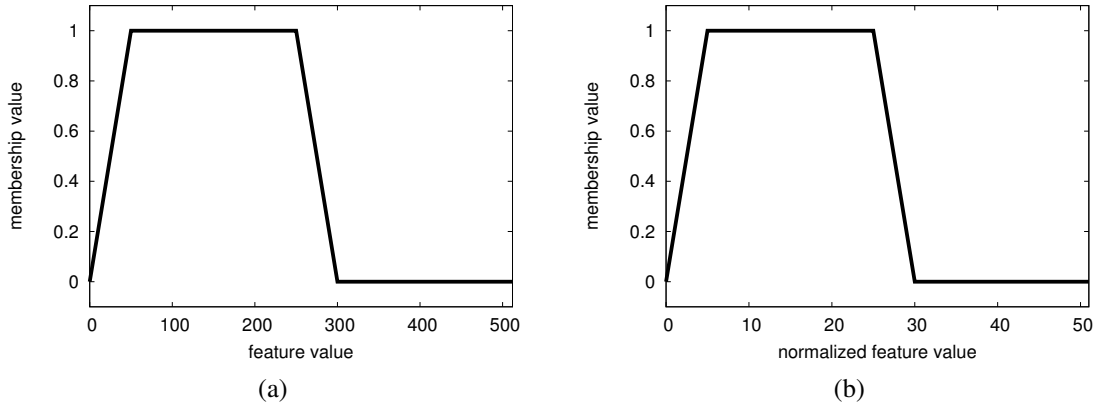


Figure B.21: Fuzzy membership functions that can be used to extract edge pairs in approximately the first half of the profile line (`SetType` = *'position_first_pair'*, `PairSize` = 10).

Figure B.22 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.21. Only edge pairs are returned that have a distance from the first edge pair that corresponds to the specified values. See appendix B.6 on page 59 for an example where in addition to a fuzzy membership function for the position of the edge pairs a fuzzy membership function for their size is specified.



Figure B.22: The extracted edge pairs.

The fuzzy membership function displayed in figure B.21a can be created with the following HDevelop code:

```
SetType := 'position_first_pair'
create_funct_1d_pairs ([0,50,250,300], [0,1,1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.21b can be created with the following HDevelop code:

```
SetType := 'position_first_pair'
PairSize := 10
create_funct_1d_pairs ([0,5,25,30], [0,1,1,0], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.3.5 Subtype *'position_last_pair'*

'position_last_pair' behaves like *'position_pair'* with the reference point located at the position of the last edge pair.

The x values of fuzzy membership functions for the sub set type *'position_last_pair'* must lie within the range

$$-\text{length of the ROI} \leq x \leq 0.0.$$

Figure B.23 shows fuzzy membership functions that can be used to extract edge pairs close to the last edge pair.

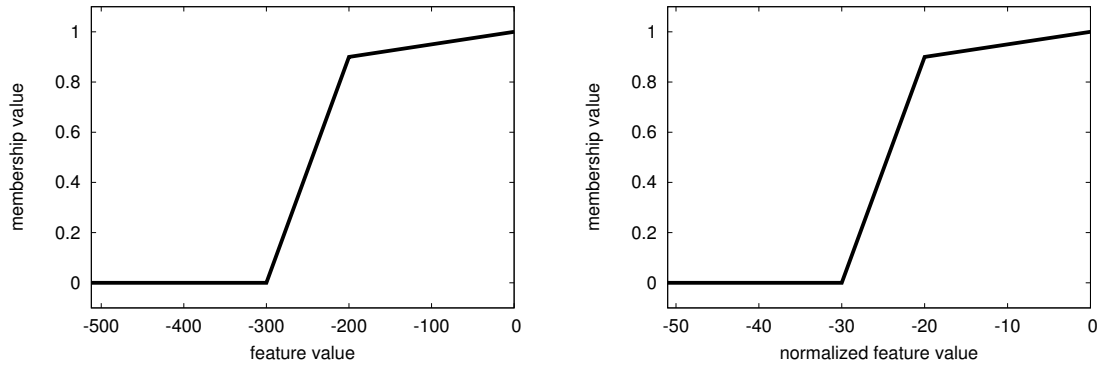


Figure B.23: Fuzzy membership functions that can be used to extract edge pairs close to the last edge pair (`SetType` = *'position_last_pair'*, `PairSize` = 10).

Figure B.24 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.23. Only edge pairs in the second half of the profile line are returned. Note that even though the same fuzzy membership function has been used as in appendix B.3.3 on page 52, more edge pairs are extracted. This is because the reference point is located at the last edge pair instead of the end of the profile line.

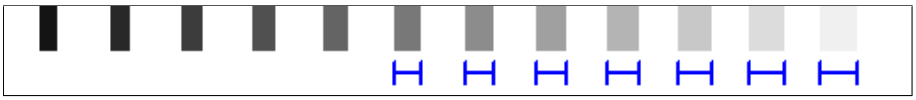


Figure B.24: The extracted edge pairs.

The fuzzy membership function displayed in figure B.23a can be created with the following HDevelop code:

```
SetType := 'position_last_pair'
create_funct_1d_pairs ([-300,-200,0], [0,0.9,1], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.23b can be created with the following HDevelop code:

```
SetType := 'position_last_pair'
PairSize := 10
create_funct_1d_pairs ([-30,-20,0], [0,0.9,1], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.4 Set Type *size*

B.4.1 Subtype *'size'*

Evaluates the size of the edge pairs, i.e., the distance between the two edges.

The x values of fuzzy membership functions for the sub set type *'size'* must lie within the range

$$x \geq 0.0.$$

Figure B.25 shows fuzzy membership functions that can be used to extract edge pairs that are approximately 50 pixels wide.

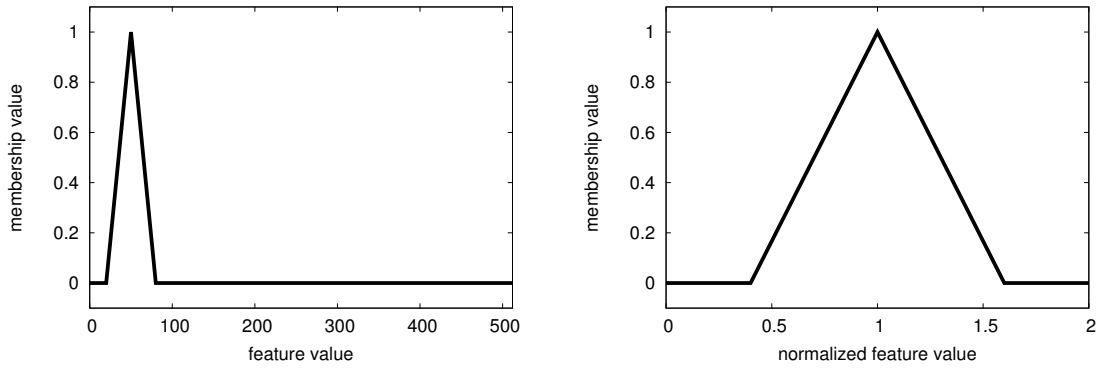


Figure B.25: Fuzzy membership functions that can be used to extract edge pairs that are approximately 50 pixels wide (`SetType` = *'size'*, `PairSize` = 50).

Figure B.26 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.25. Only edge pairs that are approximately 50 pixels wide are returned.



Figure B.26: The extracted edge pairs.

The fuzzy membership function displayed in figure B.25a can be created with the following HDevelop code:

```
SetType := 'size'
create_funct_1d_pairs ([20,50,80], [0,1,0], FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

The equivalent normalized fuzzy membership function, which is displayed in figure B.25b can be created with the following HDevelop code:

```
SetType := 'size'
PairSize := 50
create_funct_1d_pairs ([0.4,1,1.6], [0,1,0], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

B.4.2 Subtype *'size_diff'*

Evaluates the signed difference between the desired `PairSize` and the actual size of the edge pairs.The signed difference is defined by:

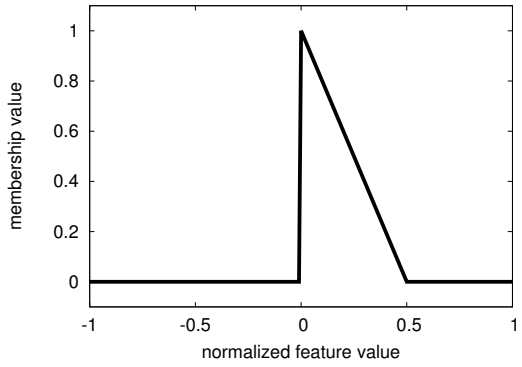
$$x = \frac{\texttt{PairSize} - \text{actual size of a pair}}{\texttt{PairSize}}$$

The x values of fuzzy membership functions for the sub set type *'size_diff'* must lie within the range

$$x \leq 1.0.$$

Figure B.27 shows a fuzzy membership function that can be used to extract edge pairs that have the specified `PairSize` or are a little bit smaller.

The sub set type *'size_diff'* can only be specified with a normalized fuzzy membership function.



(a) (b)

Figure B.27: A fuzzy membership function that can be used to extract edge pairs that have the specified `PairSize` or are a little bit smaller (`SetType` = *'size_diff'*, `PairSize` = 15).

Figure B.28 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.27. Only edge pairs that are between 11 pixels and 15 pixels wide are returned.



Figure B.28: The extracted edge pairs.

The normalized fuzzy membership function displayed in figure B.27b can be created with the following HDevelop code:

```
SetType := 'size_diff'
PairSize := 15
create_funct_1d_pairs ([-0.01,0,0.5], [0,1,0], \
                       NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

Note that with the subtype *'size_diff'* asymmetrical fuzzy membership functions can be specified for the difference between the `PairSize` and the actual size of an edge pair. This is not possible with the subtype *'size_abs_diff'* (see appendix B.4.3).

B.4.3 Subtype *'size_abs_diff'*

Evaluates the absolute difference between the desired `PairSize` and the actual size of the edge pairs.The absolute difference is defined by:

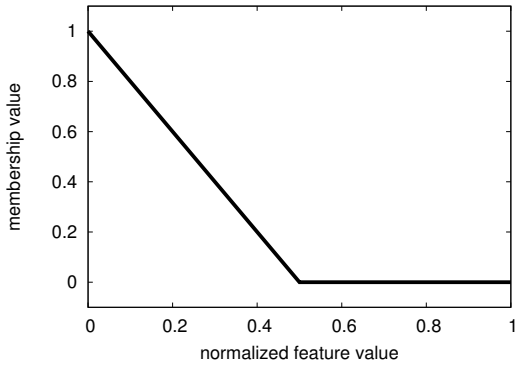
$$x = \frac{|\texttt{PairSize} - \text{actual size of a pair}|}{\texttt{PairSize}}$$

The x values of fuzzy membership functions for the sub set type *'size_abs_diff'* must lie within the range

$$x \geq 0.0.$$

Figure B.29 shows a fuzzy membership function that can be used to extract edge pairs that are approximately as wide as the specified `PairSize`.

The sub set type *'size_abs_diff'* can only be specified with a normalized fuzzy membership function.



(a) (b)

Figure B.29: A fuzzy membership function that can be used to extract edge pairs that are approximately as wide as the specified `PairSize` (`SetType` = *'size_abs_diff'*, `PairSize` = 15).

Figure B.30 shows the positions of the edge pairs that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.29. Only edge pairs that are between 11 pixels and 19 pixels wide are returned. Note that in addition to the edge pairs that are extracted in appendix B.4.2 on page 56 also larger edge pairs are returned.



Figure B.30: The extracted edge pairs.

The normalized fuzzy membership function displayed in figure B.29b can be created with the following HDevelop code:

```
SetType := 'size_abs_diff'
PairSize := 15
create_funct_1d_pairs ([0,0.5], [1,0], NormalizedFuzzyMembershipFunction)
set_fuzzy_measure_norm_pair (MeasureHandle, PairSize, SetType, \
                             NormalizedFuzzyMembershipFunction)
```

Note that with the subtype *'size_abs_diff'* only symmetrical fuzzy membership functions can be specified for the difference between the `PairSize` and the actual size of an edge pair. To specify asymmetrical fuzzy membership functions, use the subtype *'size_abs_diff'* (see appendix B.4.2 on page 56).

B.5 Set Type *gray*

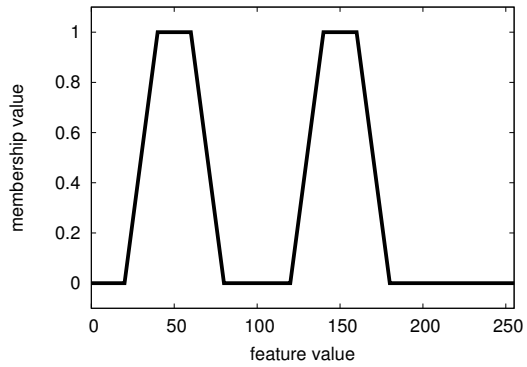
B.5.1 Subtype *gray*

Evaluates the mean gray value between the two edges of edge pairs.

The x values of fuzzy membership functions for the sub set type *'gray'* must lie within the range of gray values, e.g.,

$0.0 \leq x \leq 255.0$ for *'byte'* images and
$0.0 \leq x \leq 65535.0$ for *'uint2'* images.

Figure B.31 shows a fuzzy membership function that can be used to extract edge pairs that enclose areas with a gray value of approximately 50 or 150.



(a)

The sub set type *'gray'* cannot be specified with a normalized fuzzy membership function.

(b)

Figure B.31: A fuzzy membership function that can be used to extract edge pairs that enclose areas with a gray value of approximately 50 or 150 (`SetType` = *'gray'*).

Figure B.32 shows the positions of the edges that will be extracted if the fuzzy measure object is created with the fuzzy membership function displayed in figure B.31. With this fuzzy measure object, only those edge pairs are returned that enclose areas of the specified gray values.

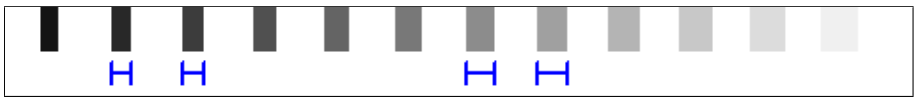


Figure B.32: The extracted edges.

The fuzzy membership function displayed in figure B.31a can be created with the following HDevelop code:

```
SetType := 'gray'
create_funct_1d_pairs ([20,40,60,80,120,140,160,180], [0,1,1,0,0,1,1,0], \
                       FuzzyMembershipFunction)
set_fuzzy_measure (MeasureHandle, SetType, FuzzyMembershipFunction)
```

B.6 Set Type *position_pair* combined with *size*

Figure B.33 shows two fuzzy membership functions that can be used to determine edge pairs in approximately the first half of the profile line. In addition to only specifying the position of the edge pairs (see appendix B.3.1 on page 50), also a fuzzy membership function for the size of the edge pairs is specified (see appendix B.4.1 on page 55). With this, it is possible to define precisely which kind of edge pairs must be returned.

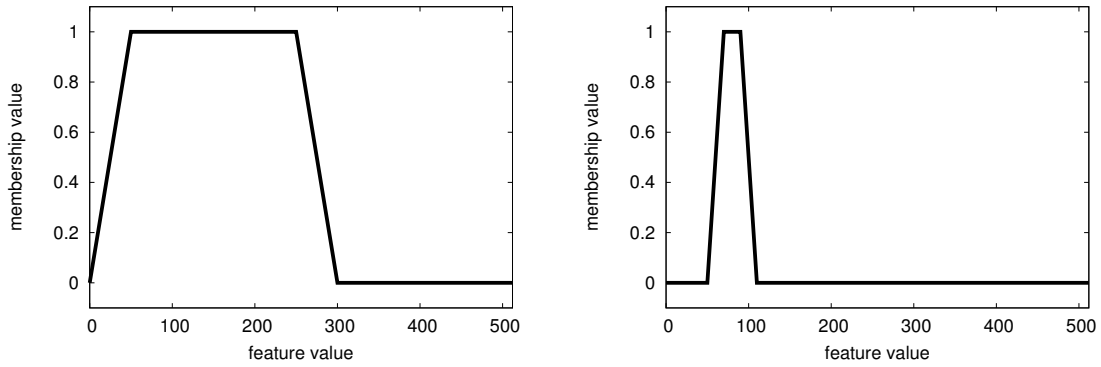


Figure B.33: Fuzzy membership functions that can be used to extract edge pairs in approximately the first half of the profile line: (a) The restriction for the position of the pairs (`SetType` = *'position_pair'*) and (b) a fuzzy membership function that favors edge pairs with a width of about 80 pixel (`SetType` = *'size'*).

Figure B.34 shows the positions of the edge pairs that will be extracted if the the fuzzy measure object is created with the fuzzy membership functions displayed in figure B.33. Only larger edge pairs in approximately the first half of the profile line are returned. Note that the smaller edge pairs returned in the example in appendix B.3.1 on page 50 are not extracted. This is because of the constraint on the size of the edge pairs, which is introduced by the fuzzy membership function for the size of the edge pairs (figure B.33b).



Figure B.34: The extracted edge pairs.

The fuzzy membership function displayed in figure B.33a can be created with the following HDevelop code:

```
create_funct_1d_pairs ([0,50,250,300], [0,1,1,0], \
                       FuzzyMembershipFunctionPositionPair)
```

The fuzzy membership function displayed in figure B.33b can be created with the following HDevelop code:

```
create_funct_1d_pairs ([50,70,90,110], [0,1,1,0], \
                       FuzzyMembershipFunctionSize)
```

Then, the fuzzy measure object can be created:

```
set_fuzzy_measure (MeasureHandle, 'position_pair', \
                   FuzzyMembershipFunctionPositionPair)
set_fuzzy_measure (MeasureHandle, 'size', FuzzyMembershipFunctionSize)
```

Appendix C

Definition of the Geometric Mean

The geometric mean of a sequence $\{a_i\}_{i=1}^n$ is defined by

$$G(a_1, \ldots, a_n) \equiv \left(\prod_{i=1}^{n} a_i \right)^{1/n}$$

Thus,

$$\begin{aligned} G(a_1, a_2) &= \sqrt{(a_1 a_2)} \\ G(a_1, a_2, a_3) &= (a_1 a_2 a_3)^{1/3} \end{aligned}$$

and so on.

Index

1D edge extraction, 9
1D measuring, 7
1D measuring with slanted edges, 37

accuracy of 1D measuring, 25
align regions of interest or images for 1D measuring, 23

create fuzzy measure object, 31
create measure object for 1D measuring, 11

defuzzify value, 31

fuzzy measuring, 27
fuzzy measuring contrast (set sub type), 44
fuzzy measuring contrast (set type), 44
fuzzy measuring gray (set sub type), 58
fuzzy measuring gray (set type), 58
fuzzy measuring position (set sub type), 45
fuzzy measuring position (set type), 45
fuzzy measuring position center (set sub type), 46
fuzzy measuring position end (set sub type), 47
fuzzy measuring position first edge (set sub type), 48
fuzzy measuring position first pair (set sub type), 53
fuzzy measuring position last edge (set sub type), 49
fuzzy measuring position last pair (set sub type), 54
fuzzy measuring position pair (set sub type), 50
fuzzy measuring position pair (set type), 50
fuzzy measuring position pair center (set sub type), 51
fuzzy measuring position pair combined with size (set type), 59
fuzzy measuring position pair end (set sub type), 52
fuzzy measuring set type
  overview, 29
fuzzy measuring size (set sub type), 55
fuzzy measuring size (set type), 55
fuzzy measuring size abs diff (set sub type), 57
fuzzy measuring size diff (set sub type), 56
fuzzy membership functions, 30, 43

geometric mean, 61
gray-value profile, 19

image types for 1D measuring, 26

measure (1D measuring), 15
measure along region of interest partially outside image, 25
measure distances along circular arcs (1D measuring), 24
measure edge pair positions (1D measuring), 17
measure edge position (1D measuring), 15
measure object, 9
measure object (fuzzy 1D measuring), 27
measure with gray-value threshold (1D measuring), 19

rectify lens distortion for 1D measuring, 23
reset fuzzy measure function, 36

set fuzzy measure function, 32
set multiple fuzzy measure functions, 35
set normalized fuzzy measure function, 32
speed up 1D measuring, 25

transform results of 1D measuring into 3D (world) coordinates, 22