HALCON
a product of MVTec

Solution Guide II-C
2D Data Codes



HALCON 21.05 *Progress*

MVTec Software GmbH

Building Vision for Business

Finding and decoding 2D data codes, 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 © 2006-2021 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

2D data codes are used in various application areas and get more and more important. This Solution Guide guides you to the handling of 2D data codes using the operators of HALCON.

In section 1 on page 7 we introduce you to 2D data codes in general, including a description of the different symbol types supported by HALCON, namely PDF417, Data Matrix ECC 200, QR Code, Micro QR Code, Aztec Code, and DotCode. A first example in section 2 on page 9 shows the main steps needed to read a standard 2D data code. To read non-standard 2D data codes as well or to enhance the run time, section 3 on page 11 describes the different ways to change the 2D data code model, which is used to guide the search process of the 2D data code reader.

Although the 2D data code reader of HALCON is rather powerful, there are some symbol representations that cannot be decoded for various reasons. Some problems can be solved by using image preprocessing methods. Section 4 on page 25 shows a selection of these problems and describes the corresponding preprocessing steps. A deeper insight into the handling of problems is given in section 5 on page 29. There, an approach for debugging the search process is provided. This can be used on the one hand to locate specific defects of symbols that are not decoded, and on the other hand to get information about successfully decoded symbols, so that the run time can be enhanced by a better model adaptation. Some problems strictly have to be avoided already during the image acquisition. Besides their introduction, the requirements and limitations concerning the appearance of the symbols are summarized for the individual symbol types. In some cases, the print quality is asked for. In section 6 on page 45 the quality standards supported by HALCON are introduced. First the ISO/IEC 15415:2011 standard is explained for a basic understanding of the quality inspection. Then the print quality verification using the AIM DPM-1-2006 standard is shown.

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

Contents

1 Introduction to 2D Data Codes 7

2 A First Example 9

3 Model Adaptation 11
- 3.1 Global Parameter Settings . . . 11
- 3.2 Training . . . 13
  - 3.2.1 Train the Model . . . 13
  - 3.2.2 Inspect the Changes . . . 14
- 3.3 Specific Parameter Settings . . . 15
  - 3.3.1 Shape and Size of the Symbols . . . 16
  - 3.3.2 Format (only Aztec Code) . . . 17
  - 3.3.3 Appearance of the Modules . . . 17
  - 3.3.4 Model Control Parameters . . . 21
- 3.4 Miscellaneous . . . 21
  - 3.4.1 Speeding up `find_data_code_2d` . . . 21
  - 3.4.2 Store the 2D Data Code Model . . . 23

4 Preprocessing Difficult Images 25
- 4.1 Slanted Symbol (Perspective Distortion) . . . 25
- 4.2 Large Module Gaps . . . 26
- 4.3 Noise . . . 27

5 Problem Handling 29
- 5.1 Data Access for Debugging . . . 29
  - 5.1.1 General Information About Data Access . . . 29
  - 5.1.2 Parameters to Access for Successfully Decoded Symbols . . . 31
  - 5.1.3 Parameters to Access for Symbols that are not Decoded . . . 33
- 5.2 Selected Problems and Tips to Avoid Them . . . 37
  - 5.2.1 Geometric Distortions . . . 38
  - 5.2.2 Radiometric Distortions . . . 39
- 5.3 Requirements and Limitations . . . 39
  - 5.3.1 Main Rules to Follow . . . 39
  - 5.3.2 Valid Parameter Ranges . . . 41

6 Check for Print Quality 45
- 6.1 The ISO/IEC 15415:2011 Standard . . . 45
  - 6.1.1 Grades . . . 45
  - 6.1.2 Acquire Raw Data . . . 50
- 6.2 The AIM DPM-1-2006 Standard . . . 50
  - 6.2.1 Acquire Raw Data . . . 51
  - 6.2.2 How to Apply the AIM DPM-1-2006 Standard . . . 52
  - 6.2.3 How to Adapt the Example Programs To Your Image Acquisition Device . . . 52
  - 6.2.4 Reflectance Calibration . . . 54
  - 6.2.5 Inspecting Print Quality . . . 54

Index 57

Chapter 1

Introduction to 2D Data Codes

HALCON provides means to read 2D data codes of type Portable Data Format 417 (PDF417), Data Matrix ECC 200, QR Code, Micro QR Code, Aztec Code, and DotCode. 2D data codes, which are also called 2D bar codes or 2D symbologies, are used in various areas. Similar to 1D bar codes, they encode characters and numbers in graphical symbols that are constructed by dark and light bars or dots that are called modules. 1D bar codes use black bars and the spaces in between as modules. As the individual bars or spaces have a constant width along their height, you can read a 1D bar code in a single scanning line along the symbol's width. In contrast to 1D bar codes, for the symbols of 2D data codes changes occur along both directions. Thus, the same information can be encoded in smaller symbols. The actual size of a symbol, i.e., the number of modules in both directions, mainly depends on the length of the encoded message and the level of the applied error correction. The latter is needed in order to completely decode a symbol even when it has small defects, e.g., if some of the modules are not visible. There are different types of 2D data codes. Two common types are the so-called stacked codes and matrix codes.

Of the stacked codes, HALCON supports the PDF417 (see figure 1.1). Its symbol is built up by several 1D bar codes, which are arranged in rows and columns. Each 1D bar code encodes an individual 'codeword'. According to the name of the symbol type, each codeword consists of four dark as well as four light bars (spaces) and is built up by 17 modules. It always starts with a dark and ends with a light bar. The number of rows and columns of a PDF417 symbol is variable in a range of 3 to 90 rows and 1 to 30 columns. Start and stop patterns frame the symbol on the left and right border. The first and the last columns of codewords are called left and right row indicators. These codewords provide important information for the decoding, like the number of rows and columns, the error correction level etc. Around the symbol's border, a homogeneous frame is placed, which is called quiet zone. For small symbols, a variant of PDF417 exists, where no right row indicator exists and the stop pattern is reduced to a one module wide bar. It is called compact or truncated PDF417. HALCON operators can read both conventional and truncated PDF417.

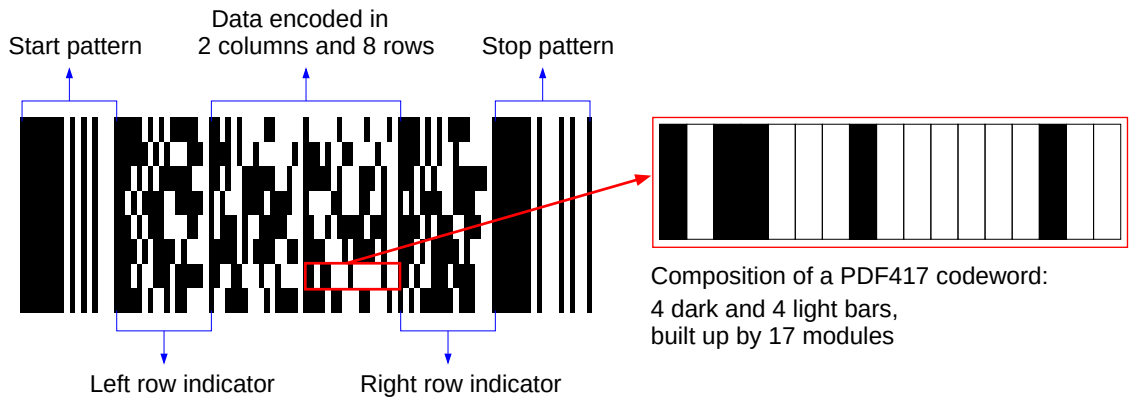

Figure 1.1: Stacked code of type PDF417.

Matrix codes supported by HALCON are Data Matrix ECC 200, QR Code, Micro QR Code, Aztec Code, and DotCode (see figure 1.2; note that these images are only exemplary for the supported codes; variants of the shown codes, like 'full-range' and 'rune' for the Aztec code, are supported as well). Matrix codes use graphical patterns. They consist of three components: a so-called finder element or finder pattern (except for DotCode symbols),

which is needed to find the symbol and its orientation in an image, the data patterns, which consist of binary modules grouped relative to the finder pattern, and a quiet zone, similar to the one needed for PDF417 symbols. The data code modules can be square, rectangular, or circular and are ordered in rows and columns. The number of rows and columns define the size of a symbol, which for a QR Code is directly linked to its version number (the higher the version number, the bigger the symbol). The finder element of a Data Matrix ECC 200 consists of an L-shape, and alternating dark and light modules on the opposite borders. The finder element of a QR Code consists of three squares, which are called 'position detection patterns'. The smaller Micro QR Code has only one finder pattern which is also square shaped. This 'position detection pattern' is located in the upper left corner of the symbol. The finder element of the Aztec Code consists of squares that are nested into one another and are located in the middle of the code. The DotCode does not contain a finder pattern.

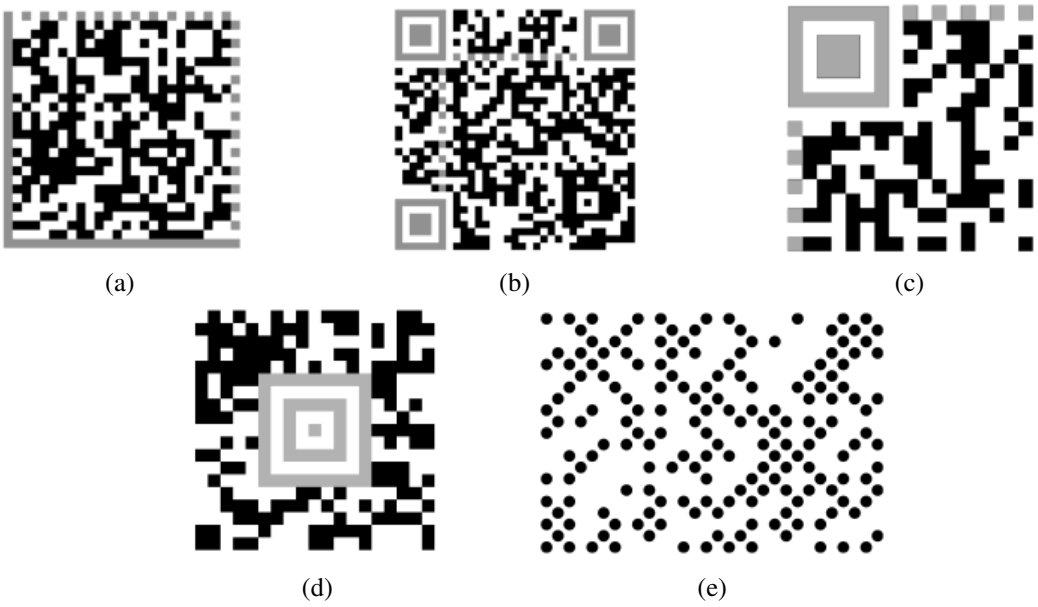

Figure 1.2: Types of matrix codes (finder patterns, if contained, are marked in gray): Data Matrix ECC 200 (a), QR Code (b), Micro QR Code (c), Aztec Code (d), and DotCode (e).

Whereas stacked codes can also be read row by row by a 1D bar code reader, matrix codes can only be decoded by inspecting images, i.e., a camera is needed. The HALCON operators for reading 2D data codes assume images as the source for matrix codes as well as for stacked codes. A basic goal of HALCON is that all 2D data code operators are applied as easily as possible. Every 2D data code operator can be applied to all supported symbol types; only the assigned parameters vary. Additionally, finding, reading, and decoding of a symbol can be done with a single operator call. The following chapters show when and how to apply the different 2D data code operators and give suggestions how to handle common problems, e.g., when facing irregular symbols or images of bad quality.

Chapter 2

A First Example

This section shows basic steps for the 2D data code reading. To follow the example actively, start the HDevelop program %HALCONEXAMPLES%\hdevelop\Applications\Data-Codes\ecc200_simple.hdev, which reads symbols of type 'Data Matrix ECC 200' in different images; the steps described below start after the initialization of the application (press Run once to reach this point).

Step 1: Specify the 2D data code model

```
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
```

First, the operator create_data_code_2d_model specifies the SymbolType that is to be read. Supported types are 'PDF417', 'Data Matrix ECC 200', 'QR Code', 'Micro QR Code', 'Aztec Code', and 'DotCode'. Here, 'Data Matrix ECC 200' is chosen. As a result, the operator returns a handle to access the created 2D data code model.

Step 2: Find, read, and decode the symbol

```
    find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                       ResultHandles, DecodedDataStrings)
```

Now, the images are read and for each image the operator find_data_code_2d searches for symbols of the specified type and, if found, reads and decodes them. The input to the operator is the image and the data code handle. If you work with images that contain more than one symbol per image you additionally must add the general parameter 'stop_after_result_num' with the number of expected symbols. In this example, the number of symbols per image is 1, which is the default value. Because no additional parameters are set in this example, two empty tuples [] are passed in the operator call. The operator find_data_code_2d returns the XLD contour representing the border of the found symbol, a result handle for further investigations concerning the search process, as well as a string containing the encoded message(s). In the program, the string is visualized by the procedure disp_message and the XLD contour is visualized by the operator dev_display (see figure 2.1).

```
    dev_display (Image)
    dev_display (SymbolXLDs)
    dev_disp_text (DecodedDataStrings, 'window', 40, 12, 'black', [], [])
    *
    * If no data code could be found.
    if (|DecodedDataStrings| == 0)
        dev_disp_text ('No data code found.\nPlease adjust the parameters.', \
                       'window', 40, 12, 'red', [], [])
    endif
```

When running the program, the applied operators assume default values for the parameters used by the specified 2D data code model. As these default values are chosen according to a certain standard, not all symbols can be found and decoded. The next chapter describes how the parameters can be adapted to better suit a specific application.

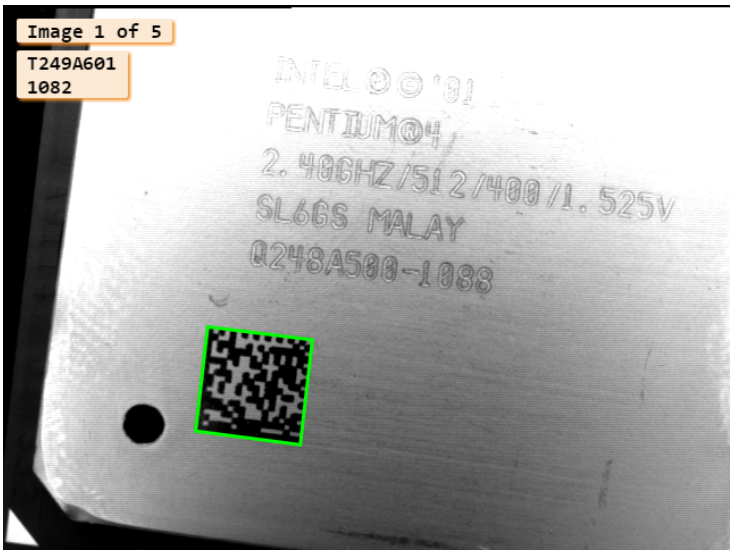

Figure 2.1: Visualization of the result: surrounding XLD contour of the symbol and decoded string.

Chapter 3

Model Adaptation

To be able to find a symbol in an image, the operator `find_data_code_2d` needs a set of parameters. In our first HDevelop program `%HALCONEXAMPLES%\hdevelop\Applications\Data-Codes\ecc200_simple.hdev` we did not explicitly set any parameters besides the `SymbolType` we were looking for. Thus, for all parameters needed for the chosen type, default values were used. These work for symbols fulfilling the following requirements:

- The code must be printed dark on light,
- the contrast value must be bigger than `30`,
- the sizes of symbol and modules are in a certain range (which depends on the selected symbol type),
- there is no or only a small gap between neighboring modules of matrix codes (for PDF417 no gap is allowed),
- for QR Codes, additionally all three position detection patterns must be visible,
- for Micro QR Codes, the single position detection pattern must be visible, and
- for Aztec Codes, the position detection pattern must be visible as well.

In many cases, you face symbols that do not fulfill all the requirements and therefore are not found. In this case, and also if you want to enhance the run time of your application, you have to modify the parameters in order to adapt the 2D data code model to your specific set of images. HALCON provides three different methods to modify the parameters. They are described in detail in the following sections. Additionally, tips concerning the run time and the storing of the 2D data code model are given. In particular, the next sections show how to

- adjust the model to read a wider range of symbols (see section 3.1),
- train the model automatically with a set of representative images (see section 3.2),
- optimize the model by setting specific parameters manually (see section 3.3),
- enhance the run time and save the modified model into a file (see section 3.4).

3.1 Global Parameter Settings

There are three predefined sets of parameters for the 2D data code model. The first one is used in the already mentioned standard mode, which is chosen by default when no other parameter settings are applied. It uses a restricted range of values for each parameter, is rather fast, and works fine for many, but not all, 2D data codes. The second one is used in the enhanced mode. There, a large range of values common for each parameter is checked. Therefore, it is not as fast as the standard mode, but now almost all readable symbols can be read. In particular, using this mode

- the symbols may also appear light on dark,

- the contrast can be lower ($\geq$10),
- the size of the modules can be smaller,
- a bigger gap between neighboring modules is allowed for matrix codes,
- for Data Matrix ECC 200 the symbol may be slanted up to 0.5235 (30 degrees)
- and the size of the individual modules may vary in a specific range,
- for a QR Code only two position detection patterns must be visible,
- for a Micro QR Code, the single position detection pattern must be visible, and
- for Aztec Codes the position detection pattern must be visible as well.

The third mode is the maximum recognition mode. There, additionally to the enhancements of the enhanced mode, the size of the modules may be smaller and for Data Matrix ECC 200 the robustness in the present of strong local contrast variations is improved. For Aztec Code the finder pattern may be distorted or partially occluded and for Data Matrix ECC 200 the finder pattern may also be distorted or even missing. But note that this mode should be used only in very rare cases as it is even slower and needs significantly more memory space than the enhanced mode. Additionally, it should be used only with additional parameter adjustments, especially concerning the size of the symbol.

All modes can be set using the operator set_data_code_2d_param, although for the standard mode this is only necessary to reset a 2D data code model after using another 2D data code model. The HDevelop program %HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_default_settings.hdev is similar to the first example, but now three data code models are created, one for each mode, and the results are compared.

```
create_data_code_2d_model (SymbolType[Index], 'default_parameters', \
                           'standard_recognition', \
                           DataCodeHandleStandard)
create_data_code_2d_model (SymbolType[Index], 'default_parameters', \
                           'enhanced_recognition', \
                           DataCodeHandleEnhanced)
create_data_code_2d_model (SymbolType[Index], 'default_parameters', \
                           'maximum_recognition', \
                           DataCodeHandleMaximum)
```

When running the program, the data codes are searched with each data code model and thus in all three modes. Symbols that were not found in standard mode are correctly decoded in enhanced or maximum mode. Even symbols that were not found in enhanced mode are correctly decoded in maximum mode. Switching from standard mode to enhanced mode or maximum mode is the easiest way to find symbols that do not fulfill the requirements of the standard mode. However, because for each parameter more alternatives have to be checked, the time needed for the search process increases, especially when no code is found at all. The following code lines measure the time needed to run the operator find_data_code_2d in standard mode. The time measurements in enhanced mode and in maximum mode are done in the same way. Thus, you can compare the run time needed for reading the data code with each of the global parameter sets. For example, in figure 3.1 the run time needed in enhanced mode is approximately twice the time needed in standard mode and the run time needed in maximum mode is nearly five times longer than in standard mode.

```
count_seconds (T1)
    find_data_code_2d (Image, SymbolXLDs, DataCodeHandleStandard, \
                       [], [], ResultHandles, DecodedDataStrings)
endfor
count_seconds (T2)
TimeStandard := 1000 * (T2 - T1) / real(NLoops)
```

Because of the increased run time and memory usage in enhanced mode and maximum mode, the model should always be specified as exactly as possible. For ECC 200 symbols, for example, the symbol size should be specified as exactly as possible if the maximum recognition mode is chosen.



However, to decrease the run time, you have to adapt the parameters to your specific images. Thus, **it is strongly**



Figure 3.1: Run times required on an Intel(R) Core (TM) i7-6820HQ processor (2.7 GHz CPU): 1.09ms in standard mode, 1.82ms in enhanced mode, and 6.58ms in maximum mode.

recommended to use the standard mode and additionally adjust the parameters either by applying an automatic training (see next section) or by setting specific parameters manually (see section 3.3). But note that if your symbols cannot be decoded in enhanced or maximum mode, apart from a few exceptions, a further adaptation of the model will not work either. In such a case, you should enhance the quality of your images. This can be done either during the image acquisition (pay attention, e.g., to the lighting conditions, see section 5.2 on page 37), which is recommended, or by a preprocessing (see section 4 on page 25).

3.2 Training

If you have a set of symbols you want to find, read, and decode, in most cases the individual symbols have similar attributes. This similarity can be used to train your 2D data code model, i.e., you use a subset of your symbols to automatically obtain an individual set of parameters suited best for your specific application. After the training, you use this parameter set to find the symbols in your remaining images. By this means, you can find the symbols also if they do not fulfill the requirements of the standard mode. But opposite to using the enhanced mode or even the maximum recognition mode, in most cases more restricted parameter values are checked and therefore the search process becomes faster.

3.2.1 Train the Model

In the HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\ecc200_training.hdev`, the 2D data code model created by `create_data_code_2d_model` is trained with two different images. For training, the operator `find_data_code_2d`, i.e., the same operator as for reading a 2D data code, is applied. This time, the additional parameter `'train'` is assigned. Its value determines the group of parameters that will be affected by the training. Here, we choose `'all'`, i.e., all available model parameters are trained. If you want to train only specific groups like `'symbol_size'`, `'module_size'`, or `'contrast'`, you can combine them using tuples. Inside a tuple, you can also choose the value `'all'` and exclude specific groups from the training by using their name with a preceding `'~'`. A complete list of valid values for the parameter `'train'` is provided in the description of the operator `find_data_code_2d` in the Reference Manual. If you work with images that contain more than one symbol, you can use all symbols for training by setting the additional parameter `'stop_after_result_num'` to the correct number of expected symbols. If you want to use only specific symbols, you can reduce the domain to a region of interest (ROI) containing only the specific symbols. For a short introduction to ROIs see section 3.4.1. Extensive descriptions can be found in the Solution Guide I, chapter 3 on page 27.

The operator `find_data_code_2d` is applied to each training image individually. The training of the first image restricts the model to the parameter values needed to find, read, and decode the symbol of that specific image.

All following training images are used to extend the restricted model again, so that the resulting model suits the whole set of images. If all symbols have the same size, no gaps between the modules, foreground and background modules of the same size, no distinct texture in the background, and a similar contrast, one image is sufficient for the training. But for most applications, several training images are recommended. For a satisfying training result, you should use your most different images. Here, we choose two images with different contrast values (see figure 3.2).

```
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
* Dark image
read_image (Image, 'datacode/ecc200/ecc200_cpu_007')
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all', \
                   ResultHandles, DecodedDataStrings)
* Bright image
read_image (Image, 'datacode/ecc200/ecc200_cpu_008')
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all', \
                   ResultHandles, DecodedDataStrings)
```



Figure 3.2: Images with different contrast values for training the 2D data code model.

The training leads to a new 2D data code model, which is now used to find, read, and decode the symbols in the remaining images. To do this, the operator `find_data_code_2d` is applied again, this time without the parameter `'train'`.

```
for Index := 7 to 16 by 1
    read_image (Image, ImageFiles + Index$'.2d')
    find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                       ResultHandles, DecodedDataStrings)
endfor
```

3.2.2 Inspect the Changes

In the program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\ecc200_training.hdev`, the training is framed by additional code lines, which are not necessary for the training or the following reading of the symbols, but help to understand how the training changes the 2D data code model. Before the training, the operator `query_data_code_2d_params` gets a list of parameters valid for symbols of the type specified in the 2D data code handle. In this case all model parameters that can be set for symbols of type `'Data Matrix ECC 200'` are queried. The current values of these parameters are obtained by the operator `get_data_code_2d_param`.

```
query_data_code_2d_params (DataCodeHandle, 'get_model_params', GenParamName)
get_data_code_2d_param (DataCodeHandle, GenParamName, ModelBeforeTraining)
```

After the training, the parameter values are checked again and the changes between the untrained and the trained model, stored in the variable `'ModelAdaptation'`, are displayed (see figure 3.3).

```
get_data_code_2d_param (DataCodeHandle, GenParamName, ModelAfterTraining)
ModelAdaption := (GenParamName + ':')$'-35' + ModelBeforeTraining$'15' + '  \
                 -> ' + ModelAfterTraining
disp_message (WindowHandle, \
              'Model parameters before and after the training:', 'window', \
              12, 12, 'white', 'false')
disp_message (WindowHandle, ModelAdaption, 'window', 60, 12, 'white', \
              'false')
```

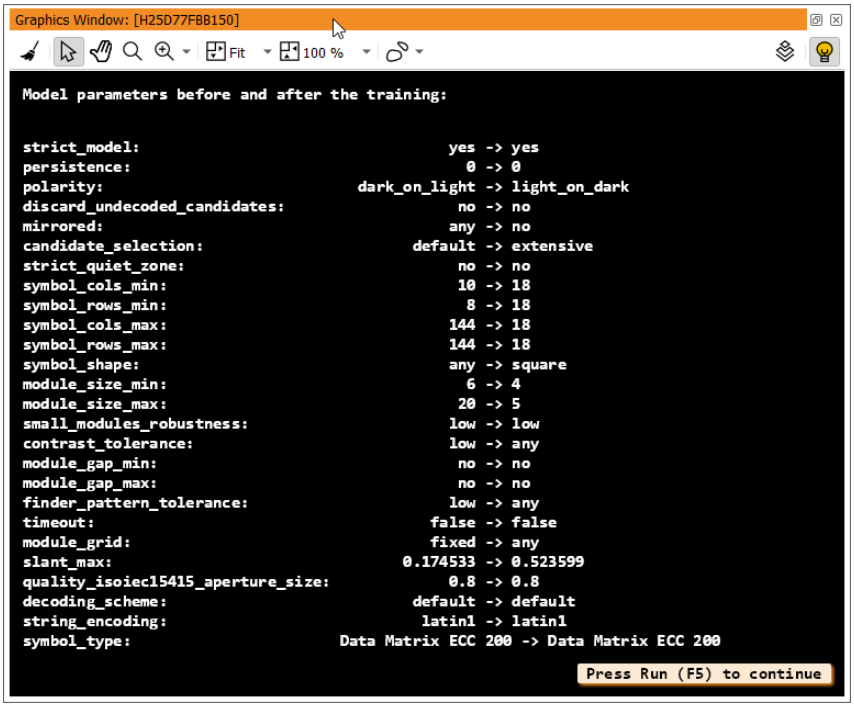

Figure 3.3: Displaying changes of parameter values.

In some cases, e.g., when you want to read additional symbols that are not similar to the symbols used for the training, you have to adapt selected parameters manually. The next section shows how to apply individual modifications and goes deeper into specific groups of parameters available for the 2D data code operators provided by HALCON.

3.3 Specific Parameter Settings

The third and most complex way to modify a 2D data code model is to change specific parameters manually. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Applications\Data-Codes\ecc200_optimized_settings.hdev` shows how to modify parameters and introduces several operators that provide a deeper insight into a specific 2D data code model. The modification of a model can be done either within the operator `create_data_code_2d_model` or by the operator `set_data_code_2d_param`, as already introduced in section 3.1. The latter operator can be called several times and therefore is used if several tuples of parameters that cannot be combined in the same operator call have to be modified. Instead of setting all parameters at once by specifying a global parameter set, we now modify the individual parameters separately. The following sections go deeper into the specific groups of model parameters, particularly concerning

- the shape and size of the symbols (see section 3.3.1),
- the appearance of the modules, e.g., their contrast or size (see section 3.3.3),
- the general behavior of the 2D data code model, i.e., the strictness of the symbol search and the storing of intermediate results (see section 3.3.4).

For the first two groups a concise overview to the value ranges specific to each individual symbol type is provided in section 5.3.2 on page 41. The complete list of parameter names is provided in the description of the operator `set_data_code_2d_param` in the Reference Manual.

3.3.1 Shape and Size of the Symbols

One group of parameters used for a 2D data code model is related to the size, e.g., the number of rows and columns, but also the shape or type of a symbol.

3.3.1.1 Symbol Shape (only Data Matrix ECC 200)

For Data Matrix ECC 200, the parameter `'symbol_shape'` specifies the shape of the symbol. If it is set to `'rectangle'`, the number of rows and columns differs. If it is set to `'square'`, the number of rows and columns is equal. If the value `'any'` is passed, both shapes are searched for. Since HALCON 7.1.1, the parameter `'symbol_shape'` still can be queried but now the search algorithm is the same for both shapes. Thus, for the symbol search it is of no importance for the parameter settings anymore.

3.3.1.2 Symbol Size

Parameters related to the symbol size set the minimum and maximum number of rows and columns allowed for a searched symbol. For matrix codes (Data Matrix ECC 200, QR Code, Micro QR Code, Aztec Code, and DotCode), rows and columns correspond to modules, whereas for the stacked code (PDF417), they correspond to codewords (excluding the codewords of the start and stop patterns as well as those of the left and right row indicators). QR Codes and Micro QR Codes are always square and therefore have the same number for rows and columns. For them, instead of setting the number explicitly, it can also be set implicitly by specifying the version number. Aztec Codes are also always square. For Data Matrix ECC 200, PDF417, and DotCode the number of rows and columns may differ. Note that for DotCode symbols one of the two dimensions needs to be even while the other one needs to be odd.

In the example program, the size of the symbol is the first attribute to be modified. By default, the size of the Data Matrix ECC 200 symbol can lie between 8 to 144 rows and 10 to 144 columns (see figure 3.4). The parameter `'symbol_size'` assumes a square symbol and therefore expects a single value for the number of rows and columns. So, by setting it to `18`, we speed up the search process by restricting the model to square shaped symbols with 18 rows and 18 columns.

```
set_data_code_2d_param (DataCodeHandle, 'symbol_size', 18)
```

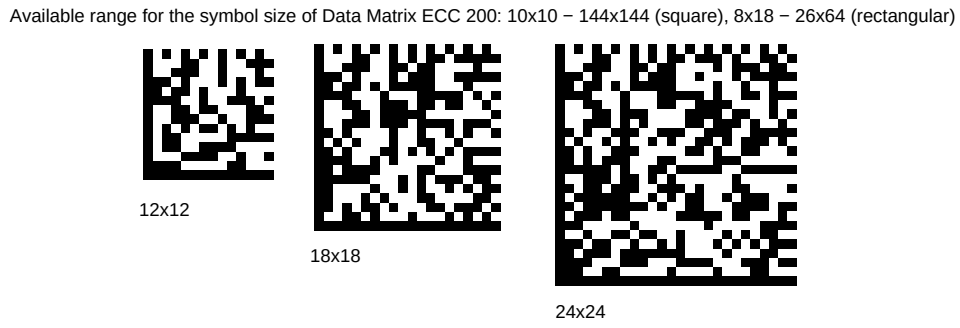

Figure 3.4: Examples for symbol sizes of Data Matrix ECC 200.

3.3.1.3 Model Type (only QR Code)

Two types of QR Code are differentiated: the old Model 1 and the new Model 2 (see figure 3.5). For both model types the smallest symbol consists of 21 rows and columns and is specified as Version 1. The largest symbol consists of 73 rows and columns (Version 14) for Model 1 and 177 rows and columns (Version 40) for Model 2. In

addition to the position detection patterns, symbols of Version 2 or larger contain extension patterns for Model 1 and alignment patterns for Model 2. When working with QR Codes, the model should be restricted to the correct model by setting `'model_type'` to 1 or 2. If the parameter is set to `'any'`, both types are searched for.

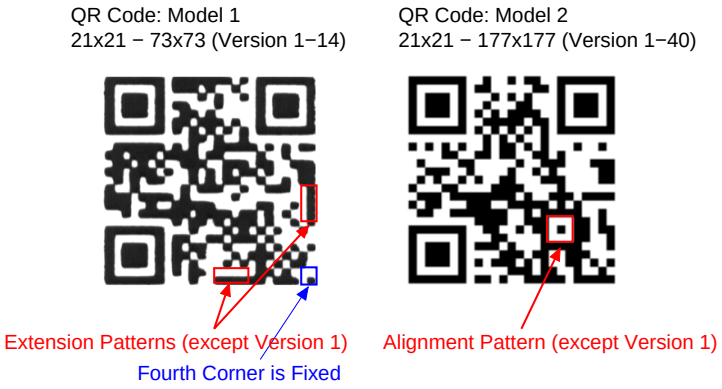

Figure 3.5: QR Codes: (left) Model 1, (right) Model 2.

Note that HALCON also supports Micro QR Codes. These codes are, however, treated as an own code type.

3.3.2 Format (only Aztec Code)

The Aztec Code can appear in three different formats, which can be set with the parameter `'format'`. This parameter can have the values `'compact'`, `'full_range'` or `'rune'`.

Those values define the following sizes:

- **compact**: 15 x 15 to 27 x 27 (increasing in steps of 4 modules)
- **full_range**: 19 x 19 to 151 x 151 (increasing in steps of 4 modules)
- **rune**: 11 x 11 (fixed)

The default is `'compact full_range'`. If an Aztec Code cannot be read, special attention should be paid to the setting of `'format'`.

The example program `%HALCONEXAMPLES%\hdevelop\Applications\Data-Codes\aztec_optimized_settings.hdev` shows how to optimize parameters especially for reading Aztec Codes, explaining in which cases parameter adaptions are especially advantageous.

3.3.3 Appearance of the Modules

The modules of 2D data code symbols can differ significantly in their general appearance. In the program, the settings for some of the main attributes are modified.

3.3.3.1 Polarity

The parameter `'polarity'` determines if the foreground modules are darker or brighter than the background (see figure 3.6) or if both polarities are checked. By default, the value `'dark_on_light'` is set. If it is set to `'any'`, both polarities are checked. In the program we change it to `'light_on_dark'` to adapt it to our specific symbols.

```
set_data_code_2d_param (DataCodeHandle, 'polarity', 'light_on_dark')
```

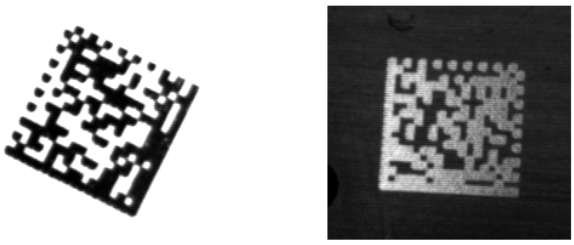

Figure 3.6: Symbols of different polarity: (left) dark on light and (right) light on dark.

3.3.3.2 Mirrored

Especially when reading symbols on transparent surfaces, it may occur that the symbol's representation is mirrored (see figure 3.7). By default, the parameter `'mirrored'` is set to `'any'`, i.e., mirrored and non-mirrored symbols are searched for. If you have only mirrored symbols you can restrict the search by setting the parameter to `'yes'`. Here, all of our symbols are not mirrored, so we restrict the search by setting the parameter to `'no'`.

```
set_data_code_2d_param (DataCodeHandle, 'mirrored', 'no')
```

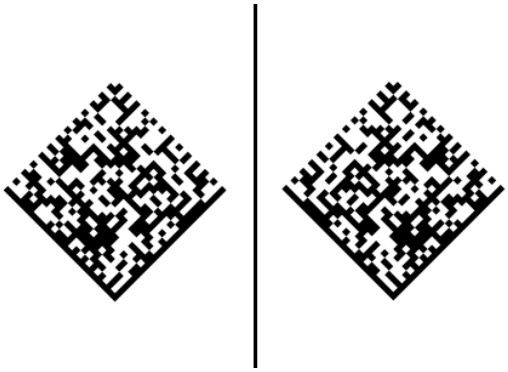

Figure 3.7: Alignment of rows and columns: (left) non-mirrored and (right) mirrored symbol.

3.3.3.3 Contrast

The contrast roughly corresponds to the difference between the gray values of the foreground and the background of a symbol. Internally, the contrast is calculated using the gradient amplitude between foreground and background. This implies that the contrast is lower for symbols with blurry edges, e.g., if the camera is out of focus or the symbol is printed badly.

See figure 3.8 for symbols with varying contrast.

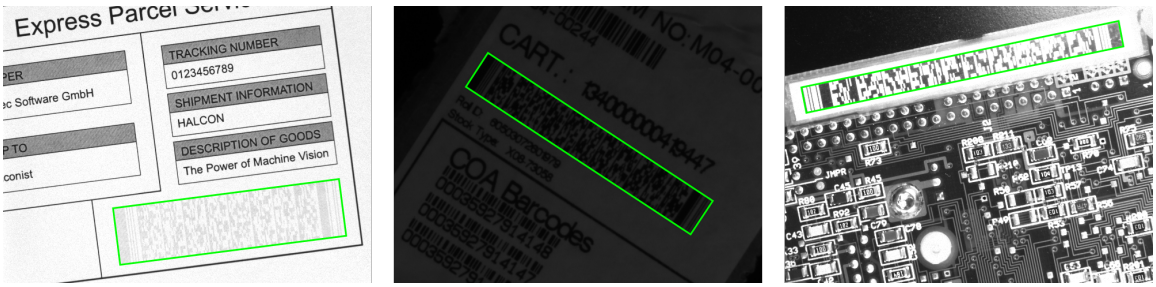

Figure 3.8: Symbols of different contrast: (left) `11`, (center) `8`, and (right) varying contrast.

Minimum Contrast

The minimum contrast value must be set lower than the contrast of the data code. In standard mode, the minimum contrast is set to 30. If the contrast is lower, e.g because the symbol is blurred or because the lighting conditions are bad, we have to set the parameter `'contrast_min'`. In this case we use the default value of the enhanced mode, i.e., a value of 10. Note, this parameter is not applicable for symbols of type `'Data Matrix ECC 200'` and `'DotCode'`.

```
set_data_code_2d_param (DataCodeHandle, 'contrast_min', 10)
```

Contrast Tolerance (only Data Matrix ECC 200)

If local contrast variations appear, e.g. in the presence of glare or reflections, the robustness of the data code reader can be improved by setting the parameter `'contrast_tolerance'` to `'high'` as in the example `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\ecc200_contrast_tolerance.hdev`.

```
set_data_code_2d_param (DataCodeHandleHigh, 'contrast_tolerance', 'high')
```

3.3.3.4 Module Size

Parameters related to the module size restrict the size of a module in pixels to speed up the search process. This is useful if the modules of all symbols are of similar size. For the matrix codes, the size is described by the width and height of a module, whereas for PDF417 codes, it is described by the module width and the module aspect ratio, which is the module height divided by the module width (see figure 3.9). The default values for the different ranges depend on the chosen symbol type. For symbols of type `'Data Matrix ECC 200'`, the default range in standard mode is 6 to 20 pixels. Because we have rather small modules in our images, we restrict the range to 4 to 7 pixels.

```
set_data_code_2d_param (DataCodeHandle, ['module_size_min', \
                        'module_size_max'], [4,7])
```

Note that for modules that are smaller than two pixels, i.e., if `'module_size_min'` is set to 1, the parameter `'small_modules_robustness'` must be set to `'high'` to enable a successful reading. But then, the run time and the needed memory increases significantly. `'small_modules_robustness'` can not be set for DotCode models.

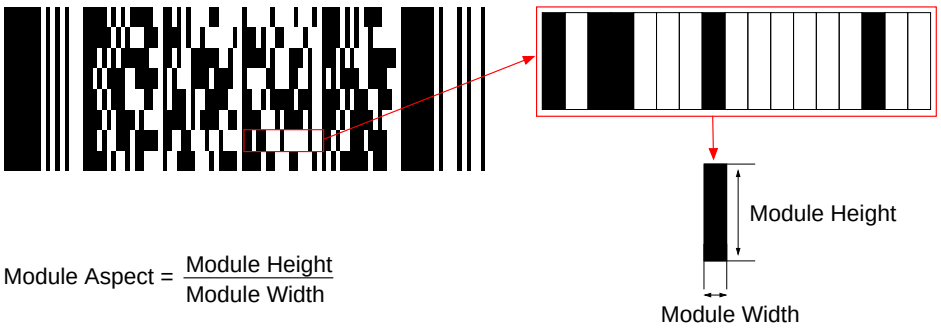

Figure 3.9: Module aspect ratio for PDF417.

3.3.3.5 Module Gap (only Matrix Codes)

In contrast to PDF417 codes, the modules of a matrix code are not necessarily connected. If a gap between the modules exists, you can specify the range of its size for both x- and y-direction, `'small'` ($\leq$ 10% of the module

size) or 'big' ($\leq$ 50% of the module size; not available for DotCode). In standard mode no or only a small gap is allowed. In the example program, we have no gaps in all directions of our symbols, so we restrict the parameter 'module_gap' to 'no'. Figure 3.10 illustrates the different gap sizes.

```
set_data_code_2d_param (DataCodeHandle, 'module_gap', 'no')
```

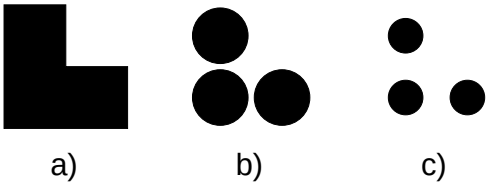

Figure 3.10: Size of module gaps: a) no gaps, b) small gaps, c) big gaps.

3.3.3.6 Maximum Slant (only Data Matrix ECC 200)

For Data Matrix ECC 200, the L-shaped finder pattern is assumed to be right-angled, but a certain slant of the symbol can be coped with (see figure 3.11). In standard mode the maximum slant angle is 0.1745 (10 degrees) and in enhanced mode the angle can be up to 0.5235 (30 degrees). Here, we keep the default value of the standard mode.

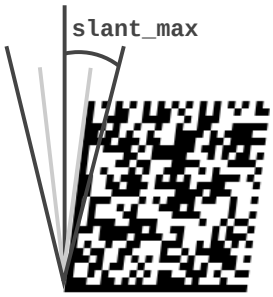

Figure 3.11: Slant angle.

3.3.3.7 Module Grid (only Data Matrix ECC 200)

The parameter 'module_grid' determines which algorithm is used for the calculation of the module positions of a Data Matrix ECC 200 symbol. If it is set to 'fixed', the modules of the symbol have to be arranged in a regular grid with similar distance between the modules. If it is set to 'variable', the grid is aligned to the alternating side of the finder pattern and the size of the modules may vary in a specific range, in particular they now may deviate up to a modules size from the regular grid. With 'any', both approaches are tested one after the other. Note that 'module_grid' is set to 'fixed' automatically for the case that the parameter 'finder_pattern_tolerance' is set to 'high', which is needed for symbols with distorted or even missing finder patterns, and which is set automatically when selecting the predefined parameter set 'maximum_recognition' for ECC 200 symbols (see section 3.1 on page 11).

3.3.3.8 Strict Quiet Zone (all codes except Aztec Code)

All code standards (except Aztec Code) require a quiet zone (i.e. a region with undisturbed background) around the code symbol. Nevertheless, HALCON is able to read codes, even if the quiet zone shows defects. The example %HALCONEXAMPLES%\hdevelop\Identification\Data-Code\micro_qr_strict_quiet_zone.hdev demonstrates how the reading of such codes can be suppressed by setting the parameter 'strict_quiet_zone' to 'yes'.

```
set_data_code_2d_param (DataCodeHandle, 'strict_quiet_zone', 'yes')
```

3.3.3.9 Number of Position Detection Patterns (only QR Code)

When working with QR Codes, you can set the parameter 'position_pattern_min' for the number of position detection patterns that are at least required to be present to 3 or 2. 3 is the default value of the standard mode and means that all position detection patterns have to be visible. 2 is used in enhanced mode. There, one of the patterns may be missing.

3.3.4 Model Control Parameters

Besides the size and appearance of the symbol, you can also specify the general behavior of a 2D data code model.

3.3.4.1 Persistence

The parameter 'persistence' determines how intermediate results, obtained while searching for symbols by the operator find_data_code_2d, are stored in the 2D data code model. By default, they are stored only temporarily ('persistence' set to 0) in order to reduce the allocated memory. Setting the parameter 'persistence' to 1, the results are stored persistently. Due to the high memory requirements, we recommend to do this only if some of the intermediate results are to be visualized or used for debugging purposes, e.g., when a symbol can not be read. Further information about debugging is given in section 5.1 on page 29. If only the decoded 2D data code and no further information is needed, 'persistence' can also be set to -1. Then, only the data that is necessary for retrieving the decoded data is stored in the model and the allocated memory is reduced to a minimum.

3.3.4.2 Strictness

Sometimes, symbols can be read but nevertheless do not fit the model restrictions on the size of the symbols. The parameter 'strict_model' controls if the operator find_data_code_2d rejects such symbols or returns them as a result independent of their size and the size specified in the model. For the first case, which is the default, the parameter is set to 'yes'. This is reasonable if only symbols of a certain size are to be found and other symbols, which may be contained in the image as well, should be ignored. If you set the parameter 'strict_model' to 'no', it may occur that also symbols that do not strictly fulfill the restrictions are found.

3.3.4.3 Timeout

It may be necessary to interrupt find_data_code_2d after a defined period, for example, when a maximum cycle time has to be ensured. This period can be defined with the parameter 'timeout' in milliseconds. All results gained until the interrupt can be accessed by get_data_code_2d_results. Passing values less or equal to zero implies a deactivation of the timeout, which is the default. The temporal accuracy is about 10 ms. It depends on several factors including the speed of your computer and the 'timer_mode' set via set_system. find_data_code_2d does not raise an exception if a timeout occurs. To check whether find_data_code_2d has been interrupted, check the parameter 'timeout_occurred' in get_data_code_2d_results. Note that the timeout is ignored if find_data_code_2d runs in training mode.

3.4 Miscellaneous

3.4.1 Speeding up find_data_code_2d

You can speed up the run time of the operator find_data_code_2d in two ways. One way is to restrict the search space, the other way is to restrict the value ranges of the parameters used by the 2D data code model. In the following, both approaches are described in more detail.

3.4.1.1 Region of Interest

The standard HALCON approach to speed up the processing is to restrict the search space. For this, you define a region of interest (ROI) where the symbol is searched in. If, e.g., all of your images have the symbol placed in the upper left corner, you define a region that covers the upper left corner in a way that it contains the symbols in all images. An ROI can be created, e.g., by generating a rectangle with the operator `gen_rectangle1`. After reading an image, you reduce the image domain to this specific rectangular region using the operator `reduce_domain`. Instead of the original image you then use the reduced one in the operator `find_data_code_2d`. Further information about ROIs can be found in the Solution Guide I, chapter 3 on page 27.

3.4.1.2 Restricted Model

In section 3.1 on page 11 it was already mentioned that adjusting the model parameters significantly affects the run time of the operator `find_data_code_2d`. So, the second way to speed it up is to restrict all parameters to the minimum range of values needed for your specific symbol representations. To understand the importance of restricting the ranges of values for specific parameters, we have a closer look at the functionality of the symbol search.

The search takes place in several passes, starting at the highest pyramid level, i.e., the level where symbols with the maximum module size are still visible. The minimum module size determines the lowest pyramid level to investigate (see figure 3.12). Reducing the range of values for the module size, we reduce the number of passes needed for the symbol search and thus enhance the run time.

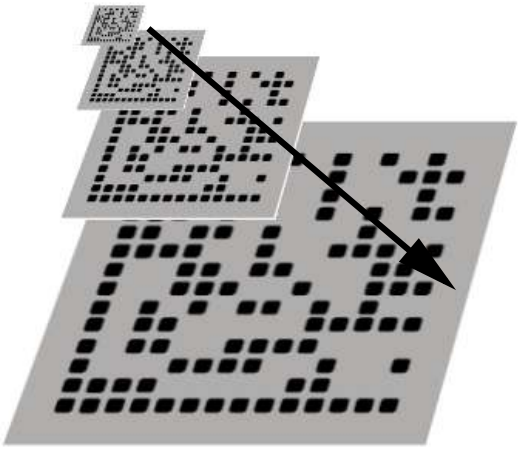

Figure 3.12: Pyramid levels.

In each pyramid level specific parameters are checked. If, e.g., the `'polarity'` is set to `'any'`, in a first pass `'dark_on_light'` symbols are searched for. If none are found, a second pass searches for `'light_on_dark'` symbols. Therefore, restricting the polarity significantly increases the speed.

Each pass consists of two phases, the search phase and the evaluation phase. The search phase is used to look for finder patterns and generate symbol candidates for every detected finder pattern. The evaluation phase is used to investigate the candidates in a lower pyramid level and, if possible, to read them. The operator `find_data_code_2d` terminates when the required number of symbols was successfully decoded, or when the last pass was performed. This explains why the symbol search is rather fast when the right parameter values are checked first and takes much longer when a wide range of parameter values has to be checked, but the requested number of symbols is not found.

Note that for the Aztec Code you can even add more pyramid levels in order to increase the robustness. This setting does however increase runtime, too.

In summary, if run time matters to your application, you should pay special attention to the model parameters for the following attributes:

- polarity,

- minimum module size,
- number of symbol rows (for PDF417, especially for strongly cluttered or textured images),
- module gaps (for matrix codes with very small modules),
- the minimum number of position detection patterns (for QR codes), and
- the use of additional pyramid levels (`additional_levels`) as well as the setting of the tolerance while looking for the finder pattern (`finder_pattern_tolerance` (for Aztec Codes)).

If these parameters are not set correctly or with a range that is unnecessarily wide, the search process slows down, especially when the requested number of symbols cannot be found. The actual values of the found symbols can be queried with the operator `get_data_code_2d_results` as described in section 5.1.2 on page 31.

3.4.2 Store the 2D Data Code Model

After changing the parameter setting by an automatic training or by a manual parameter setting, the new model can be stored in a file using the operator `write_data_code_2d_model`. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\ecc200_training.hdev` shows the main steps for storing a trained 2D data code model. First, like in the previous example programs, a 2D data code handle for symbols of the type `'Data Matrix ECC 200'` is created. Then, the model is trained by applying the operator `find_data_code_2d` to selected images using the parameter `'train'`. This step changes the 2D data code model, which then is stored into the default file `'2d_data_code_model.dcm'` using the operator `write_data_code_2d_model`.

```
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all', \
                   ResultHandles, DecodedDataStrings)
write_data_code_2d_model (DataCodeHandle, '2d_data_code_model.dcm')
```

Using the saved file, you can restore the saved model in a later session. Then instead of calling the operator `create_data_code_2d_model` at the beginning of the program, the operator `read_data_code_2d_model` is used to create a 2D data code handle, which loads the parameter settings described in the file `'2d_data_code_model.dcm'`.

```
read_data_code_2d_model ('2d_data_code_model.dcm', DataCodeHandle)
for Index := 7 to 16 by 1
    read_image (Image, ImageFiles + Index$'.2d')
    find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [], \
                       ResultHandles, DecodedDataStrings)
endfor
```

Chapter 4

Preprocessing Difficult Images

The different methods for changing the model parameters follow two complementary goals. The global parameter settings of the enhanced mode extend the ranges for the parameter values of the 2D data code model so that almost any symbol can be read. An automatic training or a manual parameter setting on the other hand restricts the ranges to enhance the run time for the images of a specific application. Therefore, if a symbol cannot be read in enhanced mode it is most likely not readable with other parameter settings as well (the few exceptional parameters that may be out of the range specified for the enhanced mode are listed in section 5.3.2 on page 42). Reasons for a failure of the 2D data code reader comprise various irreparable distortions of the symbol, which will be introduced in section 5.2 on page 37, and several problems that occur because of the bad quality of the symbol's appearance. Both problems should be avoided already when acquiring the image. But if you nevertheless have to work with images of bad quality, at least the following problems have a chance to be solved by preprocessing the image before applying the operator `find_data_code_2d`:

- The symbol cannot be read because it is slanted beyond the allowed slant angle (for troubleshooting see section 4.1),
- the gaps between the modules of a matrix code are larger than the biggest allowed module gap (for troubleshooting see section 4.2),
- there is too much noise in the image so that foreground and background of the symbol cannot be clearly distinguished anymore (for troubleshooting see section 4.3).

4.1 Slanted Symbol (Perspective Distortion)

In section 3.3 on page 20 it was mentioned that small slant angles are allowed for the L-shaped finder element of a Data Matrix ECC 200 code. PDF417, QR Code, Micro QR Code, Aztec Code, and DotCode can also be read with slight perspective or radial distortions. Large slant angles may lead to problems for all symbols, independent of the symbol type. If a symbol is strongly skewed because of radial or perspective distortions, a PDF417, a QR Code, a Micro QR Code, or an Aztec Code cannot be decoded and a Data Matrix ECC 200 is not even detected as a valid symbol. To nevertheless decode those symbols, you have to rectify the image before applying the operator `find_data_code_2d`. If you have a set of images with slanted symbols we recommend a calibrated rectification. Comprehensive information can be found in the Solution Guide III-C, section 3.4 on page 78. For single symbols or a set of symbols lying in the same plane, the HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_rectify_symbol.hdev` provides a fast solution for an uncalibrated rectification of a perspective distortion, which requires a little manual effort.

After reading an image with a slanted symbol, we use the online zooming tool of HDevelop to obtain the coordinates of all four corners of the symbol. Then, we define the coordinates of the four corners of the rectified symbol we want to obtain as result of the rectification. The shape of the rectified symbol depends on the ratio of rows and columns for the specified symbol type. As the symbol in our image is a Data Matrix ECC 200 with an equal number of rows and columns, we choose a square shape and define its corner coordinates. The operator `hom_vector_to_proj_hom_mat2d` uses the coordinates of the slanted symbol and the coordinates of the rectified symbol to compute a transformation matrix, which is then used by the operator `projective_trans_image`

to transform the slanted symbol into a square symbol with the defined extent. If you have a set of symbols lying in the same plane, the transformation matrix only has to be computed once since it can be used to rectify the other symbols as well.

```
hom_vector_to_proj_hom_mat2d (XCoordCorners, YCoordCorners, [1,1,1,1], [70, \
                              270,270,70], [100,100,300,300], [1,1,1,1], \
                               'normalized_dlt', HomMat2D)
projective_trans_image (Image_slanted, Image_rectified, HomMat2D, \
                        'bilinear', 'false', 'false')
```

The rectified symbol can be found, read, and decoded as described in section 2 on page 9 and section 3 on page 11. Figure 4.1 shows the slanted symbol before and after the rectification and decoding.

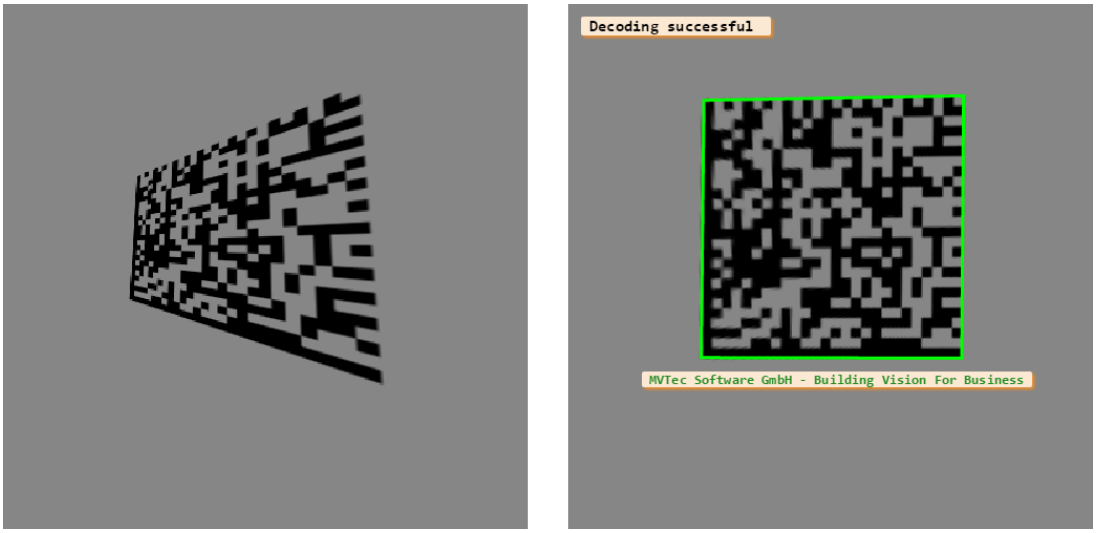

Figure 4.1: Uncalibrated rectification of an individual image with small manual effort: (left) symbol with perspective distortion, (right) decoded symbol after rectification.

4.2 Large Module Gaps

For matrix codes, gaps between modules are allowed in a certain range. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_minimize_module_gaps.hdev` shows how to read a Data Matrix ECC 200 with very large gaps. First, we try to adapt the parameters for the 2D data code model as described in section 3 on page 19, i.e., we adapt the parameter `'module_gap_min'` to the biggest allowed module gap `'big'` before trying to read the code. The adaptation can be done either with the operator `set_data_code_2d_param` or as shown in the following code lines within the operator `create_data_code_2d_model`.

```
create_data_code_2d_model ('Data Matrix ECC 200', ['module_gap_min', \
                           'module_gap_max'], ['no','big'], DataCodeHandle)
```

Because the gaps in our image are larger than the biggest allowed module gap, i.e., bigger than 50% of the module size, the reading of the symbol fails. Getting no result by adapting the parameters of the model, we have to adapt the image to suit the model. Here, we use gray value morphology, in particular a gray value erosion with a rectangular structuring element, to enlarge the foreground modules and thus minimize the size of the gaps. Figure 4.2 shows the symbol before and after the gray value morphology. After the preprocessing, the symbol is found, read, and decoded.

```
    gray_erosion_shape (Image, ImageMin, 15, 15, 'rectangle')
```

Note that this procedure mainly works for matrix codes, i.e., Data Matrix ECC 200, QR Code, Micro QR Code, Aztec Code, and DotCode. For them, the distance between the modules only have to become smaller, whereas for

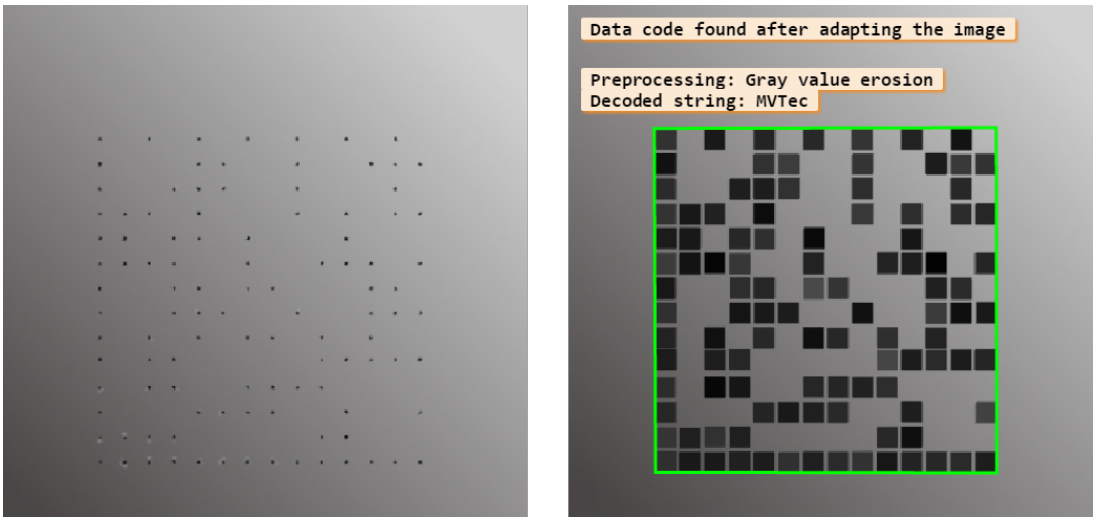

Figure 4.2: Preprocessing symbols with large module gaps: (left) original symbol with large gaps between the modules and (right) decoded symbol after gray value morphology.

PDF417 no gaps are allowed at all and the exact closing of the gaps is rather challenging, especially with gaps of slightly different size.

4.3 Noise

For the detection of 2D data code symbols, background and foreground should be clearly distinguishable, i.e., the modules should consist of homogeneous or at least low-textured regions. If you cannot read a symbol because there is too much texture or noise in the image you can try to preprocess the image with gray value morphology, a median filter, or a combination of both. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_minimize_noise.hdev` reads a symbol with several parts distorted by noise. Especially the quiet zone in the lower part of the symbol is distorted by a noisy streak. We apply both preprocessing steps separately. For the gray value morphology, the operator `gray_opening_shape` with a rectangular structuring element is used. It partly closes the gaps and, what is more important here, reduces noise (see figure 4.3).

```
gray_opening_shape (Image, ImageOpening, 7, 7, 'octagon')
```

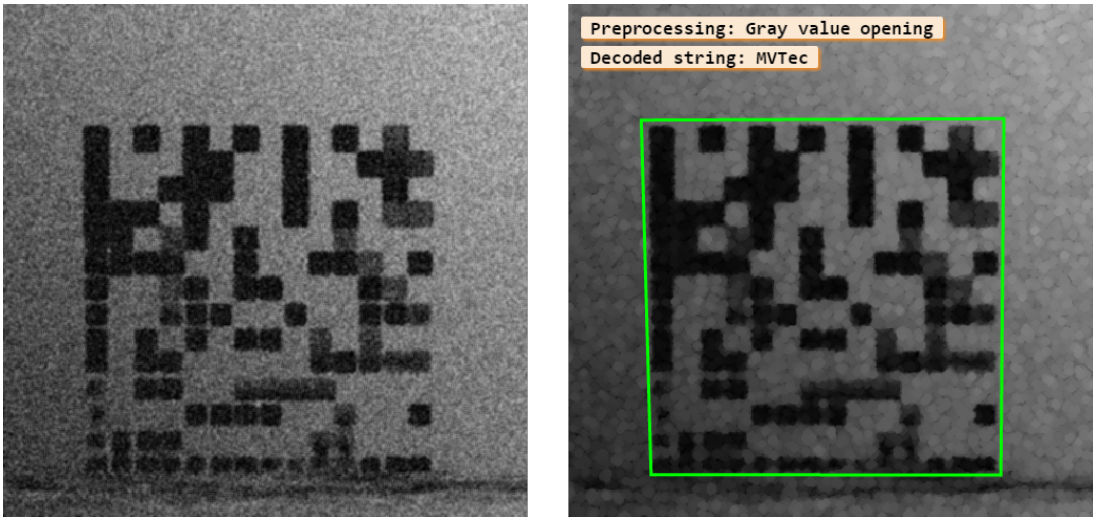

Figure 4.3: Preprocessing a noisy image: (left) noise at the symbol's lower border, (right) decoded after gray value morphology.

The median filter smoothes edges and also reduces noise (see figure 4.4).

```
median_image (Image, ImageMedian, 'circle', 3, 'continued')
```

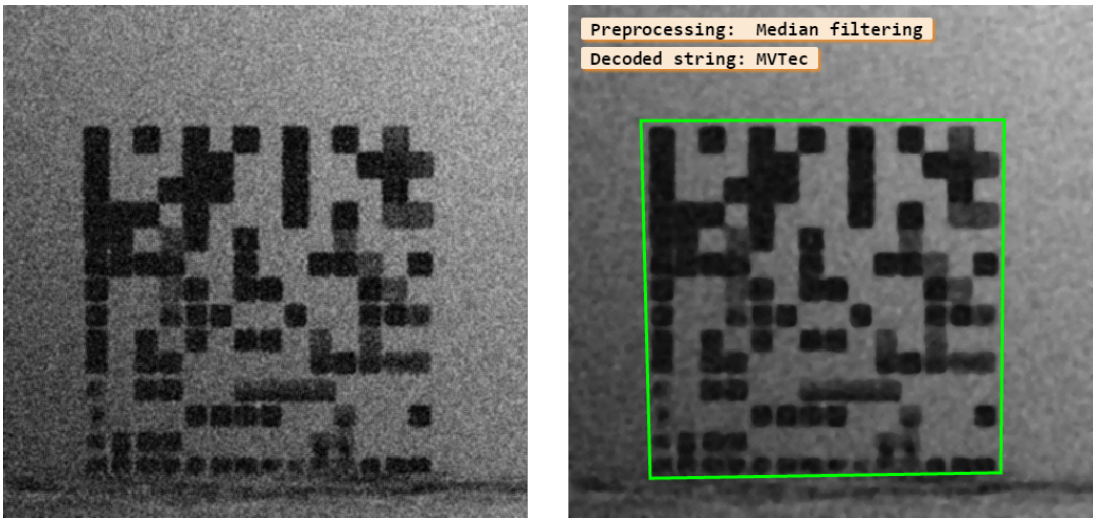

Figure 4.4: Preprocessing a noisy image: (left) noise at the symbol's lower border, (right) decoded after median filtering.

For both procedures, because of the noise reduction, the difference between the symbol and the disturbing streak in the quiet zone becomes more obvious and the reading is successful. So, in this case both preprocessing steps lead to a successful 2D data code reading.

Chapter 5

Problem Handling

The previous section proposed a selection of preprocessing steps for common problems when working with 2D data codes. Now, we will go deeper into the handling of problems.

- In section 5.1 we introduce you to the debugging of the operator `find_data_code_2d`, which on the one hand helps to enhance the run time of successfully decoded symbols and on the other hand is used for locating problems with symbols that are not decoded. To identify a problem often leads to ideas how an undecoded symbol can be preprocessed.
- Some situations exist where a preprocessing yields no success. These situations and tips how to avoid them are presented in section 5.2.
- In section 5.3, the requirements and limitations for the 2D data code reader are summarized concisely.

5.1 Data Access for Debugging

During the search process of the operator `find_data_code_2d`, various results besides the decoded data string and the XLD contour of the successfully decoded symbol are available. These results provide hints how the search process can be enhanced with respect to run time, or why a symbol is not found or decoded. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_data_access.hdev` shows how to access results for various reasons. The following sections describe the single steps of the program in detail. In particular, they introduce you to

- data access in general, as well as a selection of results useful for all debugging purposes (see section 5.1.1),
- a selection of results that are useful when facing symbols that are decoded successfully but slowly (see section 5.1.2),
- a selection of results that may give you a hint why a symbol is not decoded (see section 5.1.3).

The program concentrates on symbols of type Data Matrix ECC 200. Some characteristics of PDF417, QR Codes, Micro QR Codes, Aztec Codes, and DotCode that deviate from the presented results are introduced in section 5.1.3 on page 36.

5.1.1 General Information About Data Access

The results obtained from the operator `find_data_code_2d` are divided into iconic and alphanumeric results. Iconic results, i.e., objects like images or regions, can be queried with the operator `get_data_code_2d_objects`. The input parameters for the operator are the `DataCodeHandle`, a `CandidateHandle`, and the `ObjectName`. Alphanumeric results are obtained by the operator `get_data_code_2d_results`. Here, the input parameters are the `DataCodeHandle`, the `CandidateHandle`, and the `ResultNames`. The candidate handle needed for both iconic as well as for alphanumeric results specifies either an individual candidate for a symbol or a group of

several candidates, for which the results or objects are queried. In a single operator call, you can combine a group of candidates with an individual result or an individual candidate with a tuple of results. A list of all predefined groups of candidates as well as all available object and result names for each SymbolType can be found in the Reference Manual at the descriptions of the individual operators.

In the HDevelop program %HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_data_access.hdev, we first specify the general settings, i.e., we create a 2D data code model for symbols of type 'Data Matrix ECC 200' and set the default parameters to the enhanced mode so that all undamaged symbols can be decoded. By default, some of the results are stored only temporarily in the 2D data code model. Therefore, we set the model parameter 'persistence' to 1 to keep all intermediate results in memory (see section 3.3.4 on page 21).

```
create_data_code_2d_model ('Data Matrix ECC 200', 'default_parameters', \
                           'enhanced_recognition', DataCodeHandle)
set_data_code_2d_param (DataCodeHandle, 'persistence', 1)
```

The lists of the available alphanumeric result names and iconic object names are obtained by the operator query_data_code_2d_params with the parameters 'get_result_params' and 'get_result_objects' (see also section 3.2.2 on page 14).

```
query_data_code_2d_params (DataCodeHandle, 'get_result_params', \
                           GenParamName)
query_data_code_2d_params (DataCodeHandle, 'get_result_objects', \
                           GenObjectNames)
```

After reading an image, we apply the operator find_data_code_2d, which now stores all of its intermediate results, so that we can investigate them further.

For the group of alphanumeric results, we differentiate between general results and results associated with a specific symbol candidate or a group of candidates. General results are used, e.g., to get information about the number of candidates related to each individual group of candidates. In the program, we set the CandidateHandle to 'general' and pass a tuple containing all available general ResultNames. In detail, we want to access the lowest and highest pyramid level that is searched for symbols ('min_search_level' and 'max_search_level'), the number of passes that were completed ('pass_num'), the number of all successfully decoded symbols ('result_num'), the number of all investigated candidates ('candidate_num'), the number of candidates that were identified as symbols but could not be read ('undecoded_num'), and the number of candidates that could not be identified as valid candidates ('aborted_num'). The first three values provide us with information about the performance of the search process. All result values that are stored in the variable 'GenResult' are shown in the variable window of HDevelop. For better inspection, we display them in a new graphics window (see figure 5.1).

```
GenResultNames := ['min_search_level','max_search_level','pass_num', \
                   'result_num','candidate_num','undecoded_num', \
                   'aborted_num']
GenResultValues := []
get_data_code_2d_results (DataCodeHandle, 'general', GenResultNames, \
                          GenResultValues)
GenResult := (GenResultNames + ': ')$'-20' + GenResultValues$'2'
CandidateNum := GenResultValues[4]
UndecodedNum := GenResultValues[5]
AbortedNum := GenResultValues[6]
```

If candidates are found the program queries and displays two iconic results of the search process, the search image, and the process image. The search image is the pyramid image in which a candidate is found, whereas the process image is the pyramid image in which it is investigated more closely. A visual inspection of search image and process image often leads to ideas why a symbol is not found or decoded, or why the decoding process takes too much time.

```
get_data_code_2d_objects (SearchImage, DataCodeHandle, 0, \
                          'search_image')
dev_display (SearchImage)
get_data_code_2d_objects (ProcessImage, DataCodeHandle, 0, \
                          'process_image')
dev_display (ProcessImage)
```

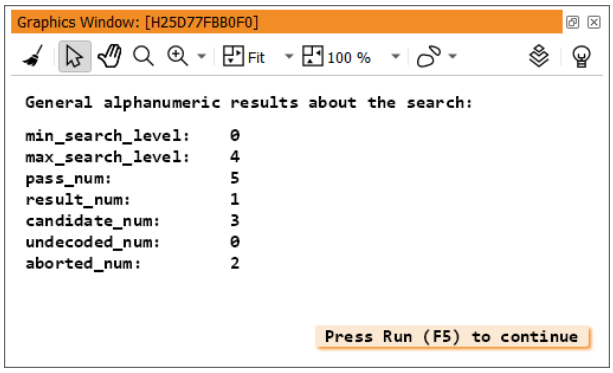

Figure 5.1: Display of general results.

Figure 5.2 shows the search image and the process image of the image we already tried to read in section 4.3 on page 27. The reason why the noisy streak in the quiet zone cannot be distinguished from the symbol becomes more obvious in the low-resolution search image than in the high-resolution original image.

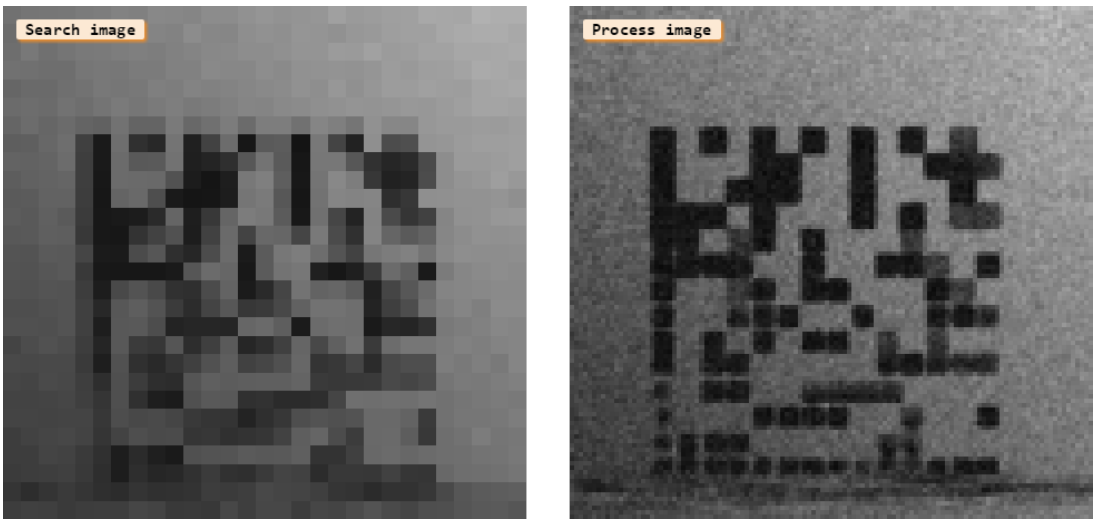

Figure 5.2: Pyramid images: (left) search image and (right) process image.

Using the information on the number of candidates in each candidate group, we now discuss two different situations. On the one hand, we investigate successfully decoded symbols to get information about their current parameter values so we can enhance the run time (see section 5.1.2). On the other hand, we investigate symbols which are not decoded to find out why they are not decoded and if specific preprocessing steps are suitable (see section 5.1.3).

5.1.2 Parameters to Access for Successfully Decoded Symbols

The main reason for the debugging of successfully decoded symbols is to enhance the run time of the search process, i.e., to reduce the number of the needed passes. For that, you have to restrict the ranges for the parameter values of the 2D data code model to a minimum (see section 3.2 on page 13 and section 3.3 on page 15). To determine a reasonable range for a specific application, it is helpful to query the current parameter values of the symbols using the operator `get_data_code_2d_results`. The success of the model adaptation can be controlled by comparing the number of completed passes queried in section 5.1.1 on page 30 before and after the adaptation.

In the HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_data_access.hdev`, we query a tuple of results which provide us with information about the symbol's appearance and additional information concerning the successful search process (see figure 5.3).

The latter comprises the actual pass in which the symbol was generated and processed (’pass’), a status message (’status’), and the decoded data string (’decoded_string’). Status messages provide you with the information whether a symbol was decoded successfully or why and at which point of the evaluation process the search was aborted for a specific candidate. Here, we query the results for successfully decoded symbols, so the status message in the tuple ’VariousResults’ is always ’successfully decoded’.

```
ResultVariousNames := ['polarity','module_gap','pass','status', \
                       'mirrored','module_height', \
                       'module_width','slant','contrast', \
                       'decoded_string']
ResultVariousValues := []
get_data_code_2d_results (DataCodeHandle, ResultHandles[J], \
                          ResultVariousNames, \
                          ResultVariousValues)
VariousResults := (ResultVariousNames + ':')$'-18' +  \
                  ResultVariousValues
disp_message (ResultWindow, \
              'Alphanumeric results about the symbol:', \
              'window', 12, 12, 'black', 'false')
disp_message (ResultWindow, VariousResults, 'window', 30, 12, \
              'black', 'false')
```

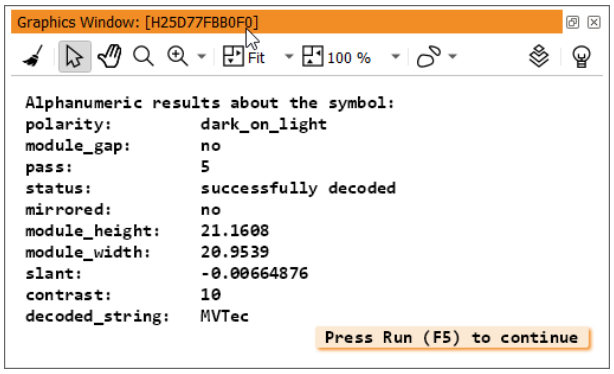

Figure 5.3: Display of various individual results.

To control the classification of the modules that are determined in the search process the program queries two different arrays of regions, in particular the iconic representations of the foreground and background modules (’module_1_rois’ and ’module_0_rois’). The returned tuples of regions (’Foreground’ and ’Background’) are displayed in figure 5.4.

```
get_data_code_2d_objects (Foreground, DataCodeHandle, \
                          ResultHandles[J], 'module_1_rois')
dev_set_color ('blue')
dev_display (Foreground)
get_data_code_2d_objects (Background, DataCodeHandle, \
                          ResultHandles[J], 'module_0_rois')
dev_set_color ('yellow')
dev_display (Background)
```

When working with rather large codes, you can query an unrestricted tuple containing all individual numbers and characters of the decoded data as ASCII code by passing the result name ’decoded_data’ to the operator get_data_code_2d_results. In the program, the resulting tuple ’ResultASCIICode’ is displayed in formatted form in a graphics window for a better view (see figure 5.5).

```
get_data_code_2d_results (DataCodeHandle, ResultHandles[J], \
                          'decoded_data', ResultASCIICode)
```

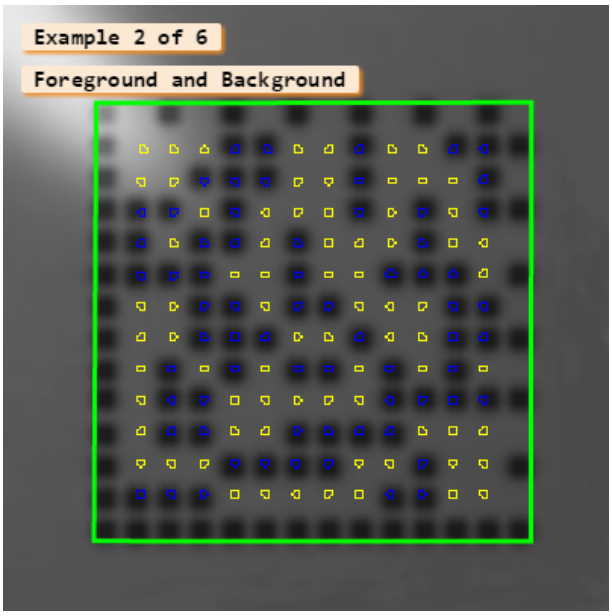

Figure 5.4: Visualization of the individual modules of a successfully decoded symbol.

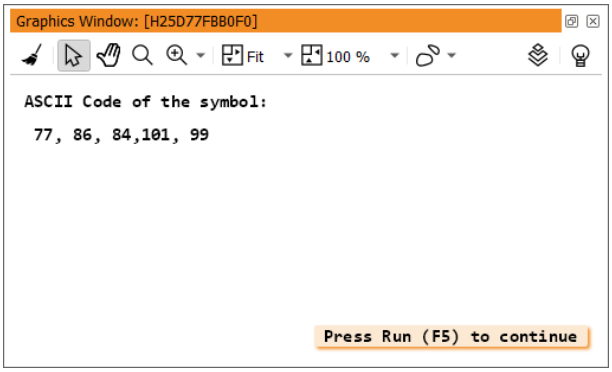

Figure 5.5: Display of the decoded data ('MVTec') as ASCII code.

5.1.3 Parameters to Access for Symbols that are not Decoded

For symbols that are not decoded, we differentiate further between symbols that are found but not decoded and symbols for which no candidate is classified as a valid symbol. If the minimum size of the modules is not set the modules themselves are often determined as candidates. To reduce the number of candidates and thus make the investigations more concise, we adapt the model to the minimum module size before applying the 2D data code reader to the problematic images (of index `'Index'`).

```
if (Index == 3)
    set_data_code_2d_param (DataCodeHandle, 'module_size_min', 24)
endif
if (Index == 4)
    set_data_code_2d_param (DataCodeHandle, 'module_size_min', 10)
endif
```

Another method to reduce the number of the candidates, especially if the image contains other objects with right angles, is to reduce the domain of the image to an ROI containing the complete symbol (including its quiet zone). For the creation of ROIs see section 3.4.1 on page 21.

To access all individual symbol candidates that are found but not decoded, we create the handle `'HandlesUndecoded'` by passing the candidate handle `'all_undecoded'` and the result name `'handle'` to the operator `get_data_code_2d_results`.

```
            get_data_code_2d_results (DataCodeHandle, 'all_undecoded', \
                                      'handle', HandlesUndecoded)
```

Then, we query the XLD contour and the corresponding status message for the undecoded symbol candidates. These provide important information about the reason why a candidate was not found or decoded. For successfully decoded symbols we received the XLD contour explicitly by the operator `find_data_code_2d`. For symbols that are not decoded, we have to query the XLD contours using the operator `get_data_code_2d_objects` with the result name `'candidate_xld'`.

```
            for J := 0 to |HandlesUndecoded| - 1 by 1
                dev_display (Image)
                get_data_code_2d_results (DataCodeHandle, \
                                          HandlesUndecoded[J], 'status', \
                                          StatusValue)
                disp_message (ResultWindow, \
                              'Alphanumeric results about the symbol:', \
                              'window', 12, 12, 'black', 'false')
                disp_message (ResultWindow, 'Status: ' + StatusValue, \
                              'window', 40, 12, 'black', 'false')
                get_data_code_2d_objects (DataCodeObject, DataCodeHandle, \
                                          HandlesUndecoded[J], \
                                          'candidate_xld')
                dev_display (DataCodeObject)
```

Additionally, we visualize the regions of the modules, this time to check, e.g., if a great amount of modules is missing or if the modules deviate from the regular grid. Perhaps the modules are too small or the quiet zone is distorted by streaks. In these cases, the obtained grid may be translated relative to the symbol. The problem with small modules can be solved by adjusting the parameter `'small_modules_robustness'` (except for DotCode symbols). Other problems often can be solved by preprocessing the image as described in section 4 on page 25.

```
                get_data_code_2d_objects (Foreground, DataCodeHandle, \
                                          HandlesUndecoded[J], \
                                          'module_1_rois')
                dev_set_color ('blue')
                dev_display (Foreground)
                get_data_code_2d_objects (Background, DataCodeHandle, \
                                          HandlesUndecoded[J], \
                                          'module_0_rois')
                dev_set_color ('yellow')
                dev_display (Background)
```

Besides the iconic results, alphanumeric information about the modules can be obtained. During the search process, the modules are read row by row. The value `0` defines a certain background module and the value `100` defines a certain foreground module. Often, modules have a value somewhere in between. An automatically chosen threshold divides the foreground from the background modules. By inspecting the values, or more precisely their deviations from `0` or `100`, you can evaluate the quality of the module classification.

```
                get_data_code_2d_results (DataCodeHandle, \
                                          HandlesUndecoded[J], \
                                          'bin_module_data', \
                                          ResultBinModules)
            endfor
```

Figure 5.6 shows a part of a symbol with very small modules. Because the modules are smaller than the specified minimum module size the computed grid cannot be fitted correctly to the symbol and each computed region contains parts of several modules, i.e., the individual regions contain dark and light parts at the same time.

The binary values of all regions that are stored in the tuple 'ResultBinModules' (see figure 5.7) confirm the visual impression, since for most of the regions the values strongly deviate from `0` or `100`. Hence, the decision whether a region belongs to foreground or background is not reliable. As noted before, the problems caused by a small module size can be solved by adapting the parameter `'small_modules_robustness'`.

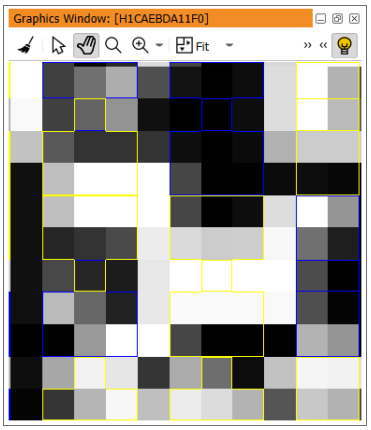

Figure 5.6: Regions of the approximated grid contain parts of multiple modules, which leads to an unreliable classification.

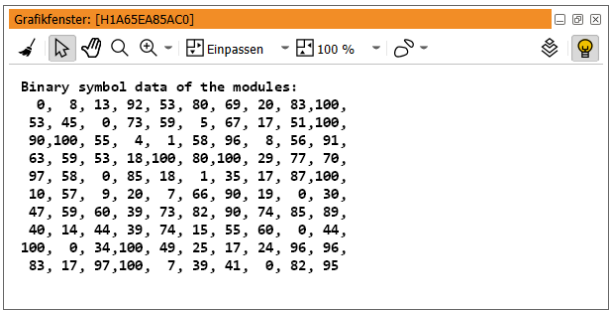

Figure 5.7: Display of the binary values for each module.

For candidates that are not detected as valid symbols, no module regions can be obtained. Therefore, after creating a handle for 'all_aborted' candidates called 'HandlesAborted', the debug information is reduced to the status message and the XLD contour for each candidate.

```
get_data_code_2d_results (DataCodeHandle, 'all_aborted', \
                          'handle', HandlesAborted)
for J := 0 to |HandlesAborted| - 1 by 1
        dev_display (Image)
        get_data_code_2d_objects (DataCodeObject, \
                                  DataCodeHandle, \
                                  HandlesAborted[J], \
                                  'candidate_xld')
        dev_set_color ('yellow')
        dev_display (DataCodeObject)
        get_data_code_2d_results (DataCodeHandle, \
                                  HandlesAborted[J], \
                                  'status', StatusValue)
        Substrings := regexp_replace(StatusValue,'(:  \
                      )',':\n')
        disp_message (ResultWindow, \
                      'Alphanumeric results about the symbol:', \
                      'window', 12, 12, 'black', 'false')
        disp_message (ResultWindow, \
                      'status:\n\'' + Substrings + '\'', \
                      'window', 40, 12, 'black', 'false')
endfor
```

For example, the status message tells us that the X-border of the finder element cannot be adjusted. A didactic illustration for such a candidate is shown in figure 5.8. By looking at the corresponding XLD contour, it becomes obvious that it cannot be adjusted because of the disturbing streak in the quiet zone near the X-border. To dif-

ferentiate clearly between streak and symbol, we have to remove the noise as described in section 4.3 on page 27.

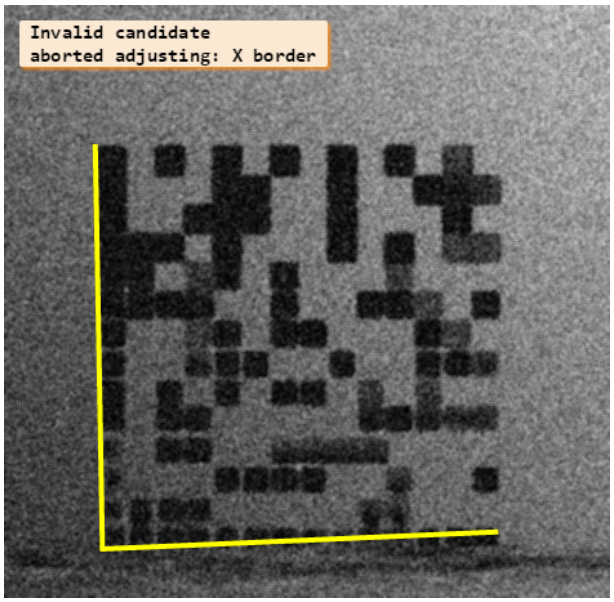

Figure 5.8: A border of the finder pattern cannot be adjusted because of the streak in the quiet zone.

Sometimes no candidate can be found at all. If this happens check visually if some of the problems introduced in section 4 on page 25 occur. Perhaps the gaps between the modules are so large that no connection between the individual modules is recognizable for the 2D data code reader, or the symbol is so noisy that the modules cannot be separated from each other. In these cases, you should try to solve the problems by applying the proposed preprocessing steps. Sometimes the symbols are damaged and cannot be read at all. Some of these situations and tips how to avoid them already at the image acquisition are introduced in section 5.2.

In the program, we concentrated on a selection of important results common for Data Matrix ECC 200. If you work with PDF417, QR Codes, Micro QR Codes, or Aztec Codes, some results will differ. An example for rather different debugging results concerns slanted symbols. For the Data Matrix ECC 200 symbol that we already presented in section 4.1 on page 25 the 2D data code reader searches for a rectangular finder pattern. Because the symbol is slanted the right angles are lost and the status messages state problems related to the border of the symbol or the finder pattern. A didactic example to visualize such a case is shown in figure 5.9. Since no valid finder pattern is identified all symbol candidates are aborted.

For the PDF417 code in figure 5.10 no rectangular finder pattern is searched for. Therefore, several symbol candidates are found and adjusted in relation to the start or stop pattern of the symbol. Because these are degenerated and do not correspond to the outline of the actual symbol no data modules are found and the symbol is not decoded.

For QR Codes, similar difficulties may occur. A didactic example to visualize such a case is shown in figure 5.11. Again, several symbol candidates are found, because the finder pattern, i.e., at least two of the three position detection patterns, are found. However, as the candidates are adjusted in relation to those two position detection patterns, the candidate does not fit the slanted symbol. Here, at least some of the modules are detected and visualize the square of the incorrect symbol candidate. Since inside the candidate the modules are in the wrong place the error correction fails.

Additionally, further results are available, e.g., the actual error correction level for PDF417 and QR Codes, queried by `get_data_code_2d_results` with the parameter `'error_correction_level'`. For PDF417, a value between `0` and `8` can be obtained. `0` means that errors are only detected but not corrected. The values `1` to `8` describe an increasing error correction capacity. For QR Codes and Micro QR Codes, the increasing levels `'L'`, `'M'` and `'Q'` are available. The level `'H'` is available for QR Codes but not for Micro QR Codes. As mentioned before, the complete list of results specific for each `SymbolType` can be found in the Reference Manual at the description of the operators `get_data_code_2d_results` and `get_data_code_2d_objects`. To learn more about the error correction levels, also refer to the corresponding standard ISO/IEC CD 18004, Information technology - Automatic identification and data capture techniques - QR Code 2005 bar code symbology specification.

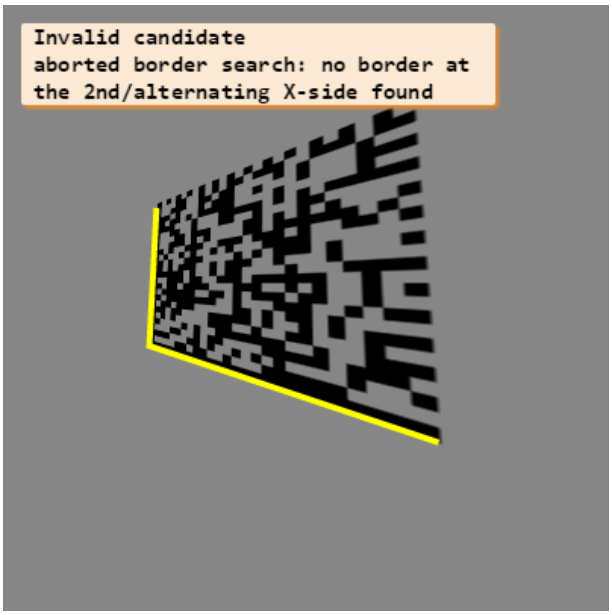

Figure 5.9: The alternating border of a rectangular finder pattern cannot be reconstructed.

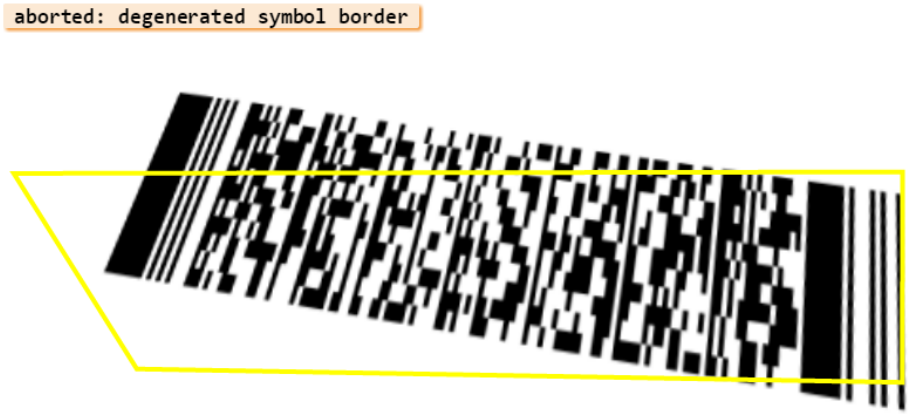

Figure 5.10: The symbol candidate is related to the stop pattern and does not fit the actual symbol outline.

5.2 Selected Problems and Tips to Avoid Them

The 2D data code reader of HALCON is a rather powerful tool, which can be used to read also partly distorted symbols. But sometimes a symbol is distorted so much that it cannot be decoded even after preprocessing the image. Some distortions are because of damaged symbols, e.g., symbols that are not printed correctly or for which a great amount of modules is missing for various reasons. Other distortions occur during the image acquisition. These can be avoided by the right acquisition conditions. In the following, we introduce you to examples for distortions that result either from

- a bad geometry, i.e., the modules of the symbol are not placed on a regular grid (see section 5.2.1),
- or a bad radiometry, i.e., due to bad lighting conditions the individual modules cannot be classified correctly (see section 5.2.2).

We recommend to avoid both situations by flattening the symbols to a level surface and using diffuse light at the image acquisition.

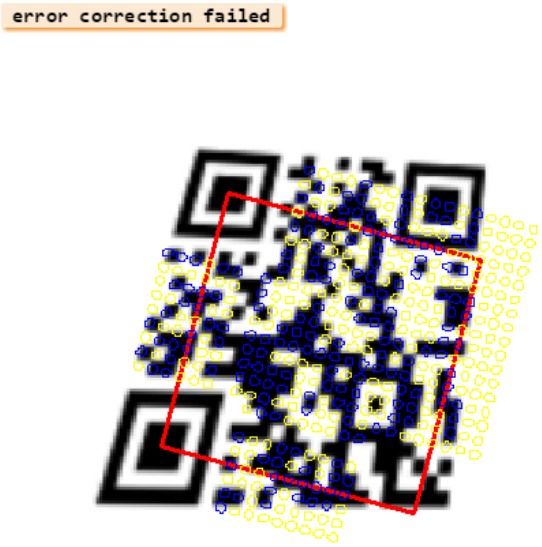

Figure 5.11: The symbol candidate is related to two position detection patterns and does not fit the actual symbol outline.

5.2.1 Geometric Distortions

The modules of an ideal symbol are placed on a regular grid. For matrix codes the grid must be regular along the whole symbol, whereas for PDF417 a regular grid is necessary only within the individual columns. While searching for a symbol, the operator `find_data_code_2d` searches for the finder pattern (or start and stop pattern for PDF417) of the specified symbol. If found, it approximates a grid for the modules, which is adjusted in relation to the finder pattern. Some errors, i.e., small deviations of the modules from the grid (up to a displacement of half a modules size, for Data Matrix ECC 200 up to a whole modules size when `'module_grid'` is set to `'variable'`), can be coped with. For stronger deviations the 2D data code reading fails. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_arbitrary_distortions.hdev` reads symbols with various distortions. In Figure 5.12, e.g., six symbols are printed on paper. The paper is crumpled, so that arbitrary distortions occur to the symbols and for four of them the modules deviate so much from the regular grid, that they cannot be decoded.

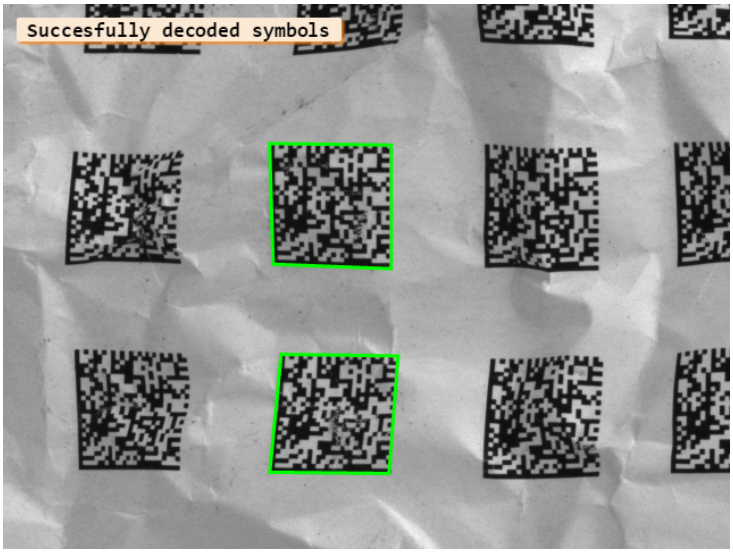

Figure 5.12: Four symbols cannot be read because of arbitrary geometric distortions.

In contrast to the regular distortion in section 4.1 on page 25, which can be removed by a rectification, here the distortions are arbitrary and irregular. Therefore, no preprocessing step can be proposed. You have to prevent such a situation already when acquiring the images by attending that the symbols are as flat as possible. If your symbols

are printed on a flexible surface like paper, you can, e.g., flatten them with a glass plate. But then, you have to be very careful and consider the right lighting conditions. Otherwise you may get problems because of reflections as described in the next section.

5.2.2 Radiometric Distortions

Besides the right geometry of the symbol's grid, the contrast and a uniform appearance of the symbol's modules, especially with regard to their polarity, are essential for the decoding of 2D data codes. The HDevelop program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\2d_data_codes_arbitrary_distortions.hdev` searches for the symbols of two more images. These contain symbols printed on a reflecting surface. Figure 5.13 shows an image where the reflections are so strong that for some parts of the symbols the contrast approaches 0, i.e., the information about the modules of these parts is not available anymore. Thus, the symbols cannot be reconstructed by preprocessing and the 2D data code reading fails.

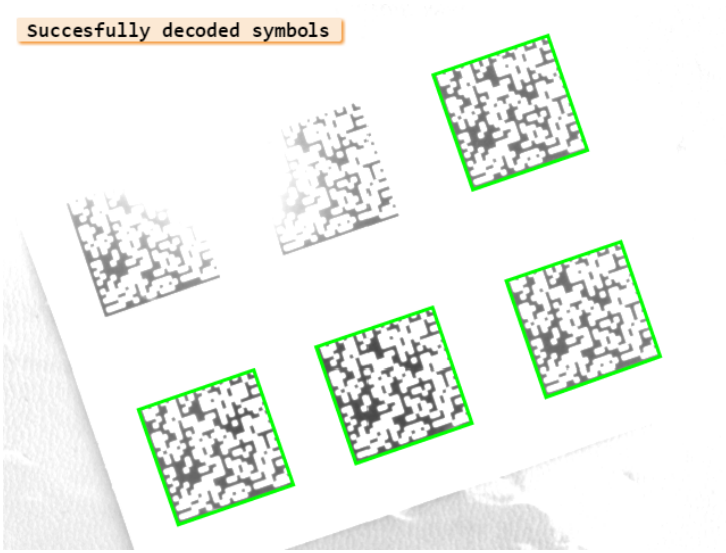

Figure 5.13: Two symbols cannot be read because parts of the symbols are not visible because of reflections.

Figure 5.14 illustrates another problem that occurs because of bad lighting conditions. Here, because of reflections the polarity changes within two of the symbols. This leads to problems, since in most cases a change of the appearance of the modules inside a symbol cannot be coped with.

For both cases, no preprocessing is reasonable to improve the image in a way that makes the symbols readable again. Therefore, it is especially important to avoid strong reflections by using diffuse light at the image acquisition.

5.3 Requirements and Limitations

For a successful 2D data code reading, a symbol's representation must fulfill certain requirements. Some of them were already stated in the preceding sections. Generally, the value ranges specified for the individual parameters should not be exceeded. Most of the limits are soft, i.e., sometimes symbols can be read although their parameter values do not completely lie in the specified ranges. But since this cannot be ensured you should try to adhere to the rules summarized in section 5.3.1. A concise list of the value ranges allowed for the parameter groups related to the size and appearance of the symbols is given in section 5.3.2.

5.3.1 Main Rules to Follow

All symbol types:

- The symbol (including the quiet zone if it is required) must be contained completely in the image.

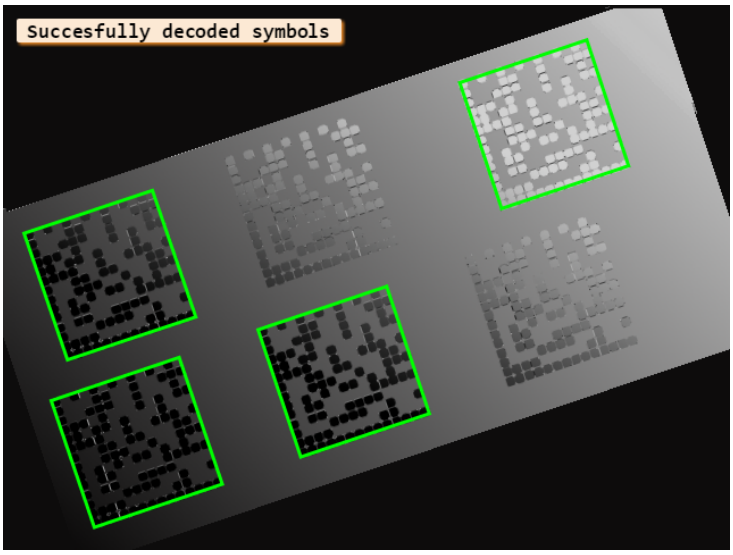

Figure 5.14: Two symbols cannot be read because of a changed module appearance (polarity).

- The modules must fit a regular grid and therefore should have approximately the same size. For matrix codes, the grid has to be regular along the whole symbol, whereas for PDF417 codes it has to be regular only within the individual columns. Some errors up to a displacement of half a modules size (for Data Matrix ECC 200 up to a whole modules size when `'module_grid'` is set to `'variable'`) can be coped with, whereas errors at the finder pattern are worse than errors at the data modules.
- The appearance of the modules of a symbol should be uniform. Especially the polarity must not change within a symbol.
- Although a minimum contrast of `1` is allowed, for a stable result the symbols should have a minimum contrast of `10` between foreground and background.

PDF417:

- For a stable result, the minimum width of the modules should not fall below 3 pixels.
- At the border of the symbol, there should be a quiet zone of at least 2 modules width in each direction.
- Gaps between the modules are not allowed.

Data Matrix ECC 200:

- For a stable result, the minimum size of the modules should not fall below 4 pixels.
- At the border of the symbol, there should be a quiet zone of at least 1 module's width in each direction.
- Gaps between the modules are allowed, but should not exceed 50% of the module's size.
- Although theoretically allowed, a slant angle of maximum `0.5235` (30 degrees) should not be exceeded.

QR Code:

- For a stable result, the minimum size of the modules should not fall below 4 pixels.
- At the border of the symbol, there should be a quiet zone of at least 4 module's width in each direction.
- Gaps between the modules are allowed, but should not exceed 50% of the module's size.
- At least two of three position detection patterns have to be visible.

Micro QR Code:

- For a stable result, the minimum size of the modules should not fall below 4 pixels.
- At the border of the symbol, there should be a quiet zone of at least 2 module's width in each direction.
- Gaps between the modules are allowed, but should not exceed 50% of the module's size.
- The position detection pattern has to be visible.

Aztec Code:

- For a stable result, the minimum size of the modules should not fall below 4 pixels.
- No quiet zone is required at the border of the symbol.
- Gaps between the modules are allowed, but should not exceed 50% of the module's size.

DotCode:

- For a stable result, the minimum size of the modules should not fall below 4 pixels.
- At the border of the symbol, there should be a quiet zone of at least 3 module's width in each direction.
- Gaps between the modules are allowed, but should not exceed 10% of the module's size.

5.3.2 Valid Parameter Ranges

The following list concisely summarizes the value ranges valid for the parameter groups related to the size and appearance of the individual symbol types. A complete list of parameter names that belong to each group can be found at the description of the operator `set_data_code_2d_param` in the Reference Manual.

Symbol Type	PDF417	Data Matrix ECC 200	QR Code	Micro QR Code	Aztec Code	DotCode
Symbol Size - Columns - Rows	 1 - 30 code-words 3 - 90 mod-ules	 10 - 144 modules, even 8 - 144 mod-ules, even	 21 - 177 modules 21 - 177 modules	 11 - 17 mod-ules 11 - 17 mod-ules	 11 - 151 modules 11 - 151 modules	 5 - 999 mod-ules 5 - 999 mod-ules
Symbol Shape	—	rectangle, square, any	—		—	—
Model Type	—	—	1, 2, any	—	—	—
Version	—	—	1 - 40	1 - 4	—	—
Format	—	—	—	—	compact, full_range, rune	—
Polarity	dark_on_light, light_on_dark, any	dark_on_light, light_on_dark, any	dark_on_light, light_on_dark, any	dark_on_light, light_on_dark, any	dark_on_light, light_on_dark, any	dark_on_light, light_on_dark, any
Mirrored	yes, no, any	yes, no, any	yes, no, any	yes, no, any	yes, no, any	yes, no, any
Minimum Contrast	1 - 100	—	1 - 100	1- 100	1 - 100	—
Module Size	—	1 - 100	1 - 100	1 - 100	1 - 100	1 - 100
Module As-pect Ratio	0.5 - 20	—	—	—	—	—
Module Gaps	no gaps	no, small, big	no, small, big	no, small, big	no, small, big	no, small
Maximum Slant	—	0 - 0.7	—	—	—	—
Module Grid	—	fixed, vari-able, any	—	—	—	—
Minimum Position Patterns	—	—	2, 3	—	—	—
Small Mod-ule Robust-ness	low, high	low, high	low, high	low, high	low, high	—
Finder Pattern Tolerance	—	low, high, any	—	—	low, high	—
Contrast Tol-erance	—	low, high, any	—	—	—	—
Additional Levels	—	—	—	—	0 - 2	—

Note that in enhanced mode (see section 3.1 on page 11) the 2D data code reader checks the whole range of values above, except for following parameters:

- the minimum contrast for all symbol types (allowed range: 1 - 100, default in enhanced mode: 10),
- the maximum module size for matrix codes (allowed range: 2 - 200, default in enhanced mode: 100),
- the module aspect ratio for PFD417 (allowed range: 0.5 - 20, default in enhanced mode for the minimum module aspect: 1.0, default in enhanced mode for the maximum module aspect: 10),
- the maximum slant for Data Matrix ECC200 (allowed range: 0 - 0.7, default in enhanced mode: 0.5235),
- the small modules robustness (allowed range: low, high, default in enhanced mode: low),
- the finder pattern tolerance for ECC 200 symbols (allowed range: low, high, any, default in enhanced mode: low).

- the contrast tolerance for ECC 200 symbols (allowed range: low, high, any, default in enhanced mode: low).

For a stable result, it is recommended not to exceed the value ranges specified for the enhanced parameter set.

Chapter 6

Check for Print Quality

In some cases, the print quality of a symbol is asked for. There are various standards available that are related to the quality verification of 2D data codes. Their focus differs depending, e.g., on the way the symbol is applied to a surface. HALCON supports the quality standards ISO/IEC 15415:2011, AIM DPM-1-2006, and SEMI T10-0701. Here, the standards ISO/IEC 15415:2011 and AIM DPM-1-2006, which are common for printed 2D data code symbols, are described in detail. As the AIM DPM-1-2006 is an extension to the ISO/IEC 15415:2011 standard, the last one will be explained first to get a basic understanding of how the quality inspection works. The print quality grading using the AIM DPM-1-2006 standard is different in its usage as it requires interactive image acquisition. Therefore, the new standard is best explained at the end of this chapter, using the program itself.

Note that the print quality inspection can be performed for all symbol types except DotCode.

6.1 The ISO/IEC 15415:2011 Standard

With `get_data_code_2d_results` and the parameter `'quality_isoiec15415'` you can query a set of values that numerically describe the quality grade for different features of a successfully read symbol in a descending order from `4` (highest quality) to `0` (lowest quality). As a tuple of values is returned, the print quality has to be queried for each symbol candidate individually. The calculation of the quality is applied after reading the symbol with `find_data_code_2d` and does not affect the run time of the actual reading process. Note that **with HALCON you can read many 2D data code symbols even if their quality according to the standard is low.**

For symbols of type Data Matrix ECC 200, QR Code, Micro QR Code or Aztec Code, the print quality grades eleven different features. To query the quality grades, you have to apply `find_data_code_2d` and then call `get_data_code_2d_results` (with the parameter `'quality_isoiec15415'`) to get the quality grades for the features `'Overall Quality'`, `'Contrast'`, `'Modulation'`, `'Fixed Pattern Damage'`, `'Decode'`, `'Axial Non-Uniformity'`, `'Grid Non-Uniformity'`, `'Unused Error Correction'`, `'Reflectance Margin'`, `'Print Growth'`, and `'Contrast Uniformity'`.

For symbols of type PDF417, you can query the quality grades for the seven features `'Overall Quality'`, `'Start/Stop Pattern'`, `'Codeword Yield'`, `'Unused Error Correction'`, `'Modulation'`, `'Decodability'`, and `'Defects'`. The quality grades also comply to the standard ISO/IEC 15415:2011 but involve also techniques that are common for 1D bar codes and which are described in detail in the standard ISO/IEC 15416:2016. Basically, a set of scan lines measures the reflectance profiles along the symbol and the individual profiles are evaluated for the quality grades of the different features.

6.1.1 Grades

The individual features that are evaluated for their quality grade are shortly introduced in the following sections. Additionally, they are demonstrated in the HDevelop example `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\ecc200_print_quality.hdev`. For detailed information, we recommend to read the specifications for the standards ISO/IEC 15415:2011, for symbols of type PDF417 also the standard ISO/IEC 15416:2016 and for symbols of the type Aztec Code the standard ISO/IEC 24778.

6.1.1.1 Overall Quality (all Symbol Types except DotCode)

The overall quality is calculated for all symbol types that are supported by HALCON, except for DotCode symbols, and is always the first element in the tuple of quality grades returned by `get_data_code_2d_results`. The overall quality is the lowest quality grade obtained for the complete set of features that are evaluated for the selected type of 2D data code. If, e.g., for one of the features the value `0` is returned, the overall quality is graded with the value `0` even if all other features have the value `4`.

6.1.1.2 Contrast (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The quality grade for the contrast of a symbol of type Data Matrix ECC 200, QR Code, Micro QR Code, or Aztec Code tests whether the two reflectance states in the symbol, namely dark and light, are sufficiently distinct. The quality grade is obtained by calculating the range between the minimum and maximum pixel intensities (maximum amplitudes) in the 2D data code domain. A strong contrast leads to a good quality grade.

Figure 6.1 shows two symbols with different quality grades for their contrast. The symbol with black and white modules has a high value, whereas the symbol with black and gray modules has a lower contrast and therefore also a lower quality grade.

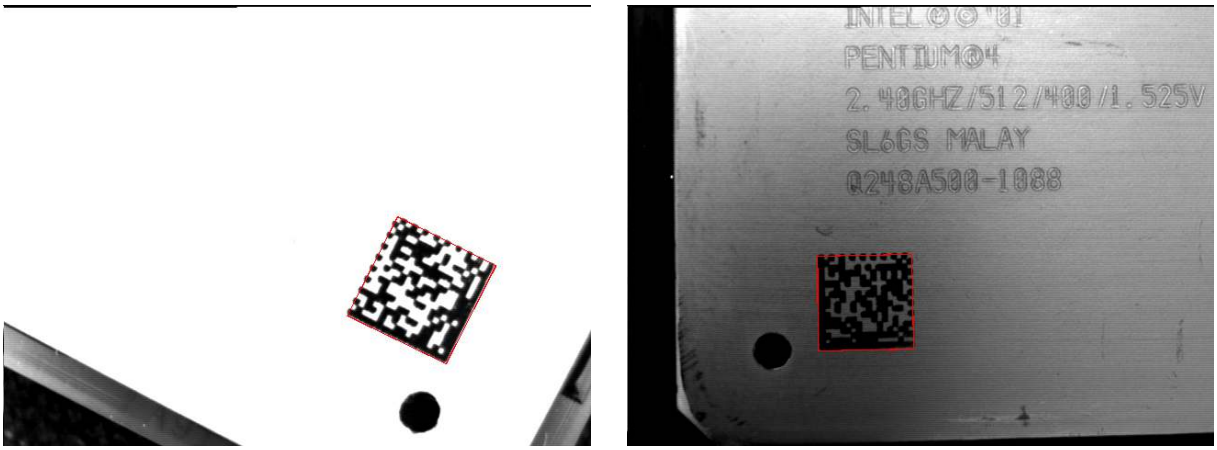

Figure 6.1: Quality grades for the contrast: (left) `4` and (right) `2`.

6.1.1.3 Modulation (all Symbol Types except DotCode)

The modulation is evaluated for symbols of type Data Matrix ECC 200, QR Code, Micro QR Code, or Aztec Code as well as for symbols of type PDF417. It measures the uniformity of the amplitudes of the modules inside the symbol. That is, not only the difference between the minimum and maximum pixel intensities of the symbol is evaluated but the relation between the minimum local contrast for adjacent elements and the contrast of the complete symbol. Large amplitudes along the whole symbol result in a good quality grade. If the modulation is insufficient, the probability that modules are not correctly identified as dark or light modules increases.

Figure 6.2 shows three symbols with different modulations. The first one consists only of black and white modules. It is clear which modules belong to the dark modules and which belong to the light modules, so the modulation is graded with the value `4`. For the other symbols, the contrast for a local part of the symbol changes so that one part of the symbol consists of black and white modules and another part consists of gray and white modules. In the second symbol the gray values of the black and gray modules differ only slightly and thus, the gray modules still have a good chance to be identified correctly and the quality is graded with value `2`. For the third symbol, the gray modules have a gray value that is somewhere in between black and white, i.e., their amplitude is rather low (see also figure 6.3). A reliable identification of the modules is difficult and the quality of the modulation is graded with the value `0`.

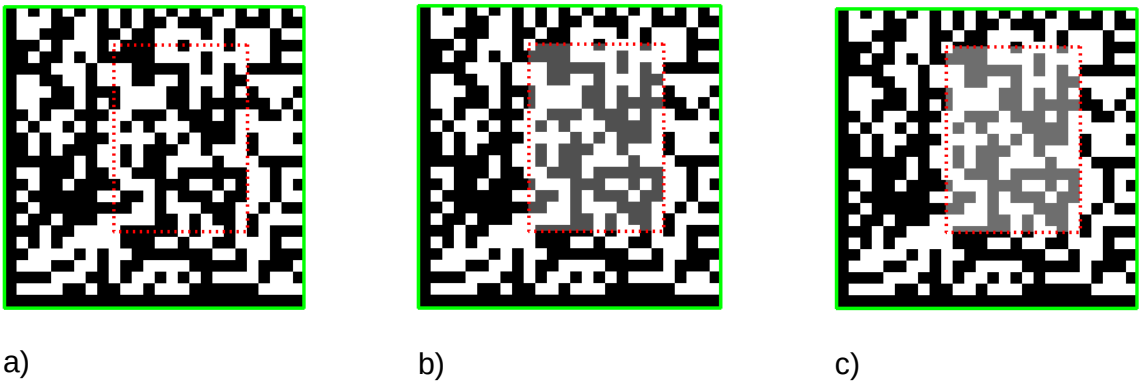

Figure 6.2: Quality grades for the modulation: (a) 4, (b) 2, and (c) 0.

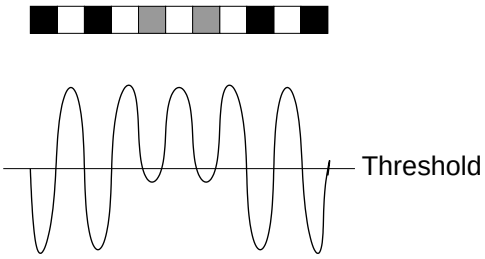

Figure 6.3: Amplitudes for black, white, and gray modules. The gray modules have a lower amplitude and with a slightly different threshold they would be incorrectly classified.

6.1.1.4 Fixed Pattern Damage (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The fixed pattern damage describes how reliable a symbol can be located and identified in the image. To locate and identify a symbol in an image, the finder pattern as well as the quiet zone of the symbol are needed. If one of these elements is damaged, the quality grade decreases dependent on the amount of the damage.

Figure 6.4 shows three symbols of type Data Matrix ECC 200. The first symbol has a damaged quiet zone, the second one is damaged at the L-shaped part of the finder pattern, and the third one is damaged at the alternating part of the finder pattern. For all three kinds of damage, the amount of the damage is big enough to lead to a quality grade of value 0.

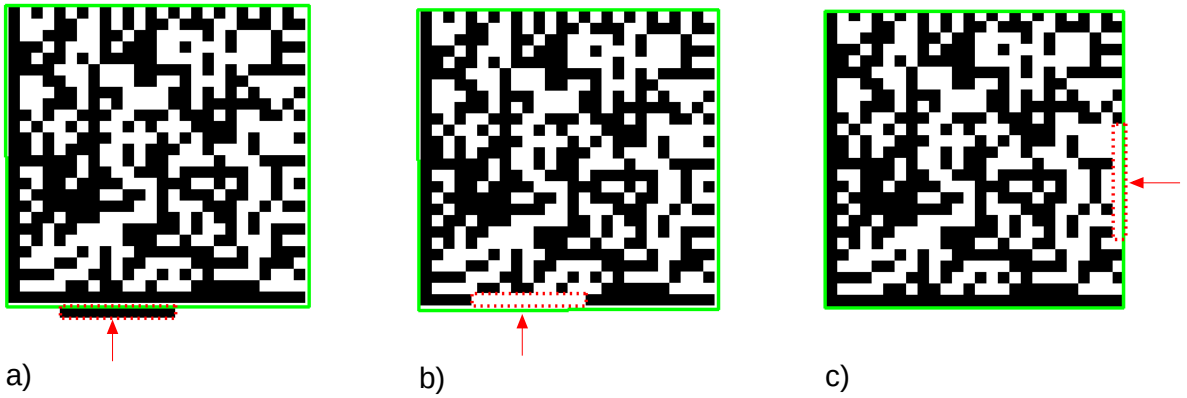

Figure 6.4: Quality grades with value 0 for fixed pattern damage: (a) damaged quiet zone, (b) damage at the L-shape of the finder pattern, and (c) damage at the alternating side of the finder pattern.

6.1.1.5 Decode (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The quality grade for the decode feature describes whether a symbol can be successfully read with HALCON or not. As only two possibilities exist (either the 2D data code can be read or not), opposite to the other features that are evaluated, no four grades are available. If the data code reading was successful, the value is 4. If not, the value theoretically would be 0, but in that case no quality grades can be obtained at all, because the quality grades can only be queried for successfully read symbols.

6.1.1.6 Axial Non-Uniformity (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The axial non-uniformity describes the aspect ratio of the symbol, i.e., the squareness of the modules or, respectively, of the grid that is built by the centers of the modules. Normally, the distance between the module's center positions is the same in horizontal and vertical direction. If a symbol is scaled only in one direction, i.e., only in its width or height, the quality grade decreases.

Figure 6.5 shows a symbol of type Data Matrix ECC 200 that is scaled only in vertical direction. The first symbol is not scaled, so that the value for the quality grade is 4. The other two symbols are scaled for different amounts, so that dependent on the scale, the value is 2 and 0.

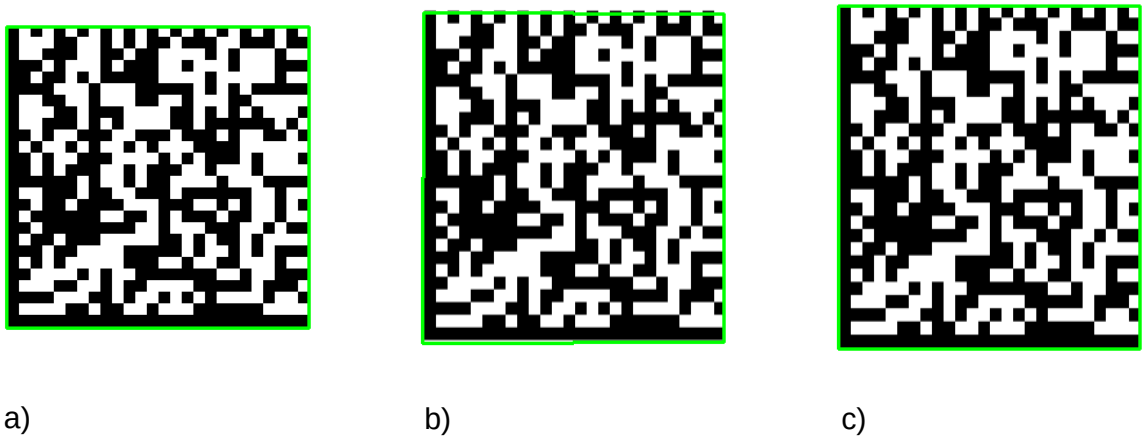

Figure 6.5: Quality grades for axial non-uniformity: (a) 4, (b) 2, and (c) 0.

6.1.1.7 Grid Non-Uniformity (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The quality grade for the grid non-uniformity quantifies the deviation of the modules from their ideal grid. Figure 6.6 shows three symbols of type Data Matrix ECC 200. For the first one, the modules are arranged in a regular, rectangular grid and the value for the quality grade is 4. For the other two symbols, the grid structure is not rectangular anymore, i.e., the individual modules deviate from the ideal grid that is assumed for a 2D data code symbol. Dependent on the amount of the deviation, the values for the quality grade decrease to 2 and 0.

Note that a bad value might also be caused by problems with the finder pattern, in which case the ideal grid may not be positioned correctly.

6.1.1.8 Reflectance Margin (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The quality grade for the reflectance margin indicates how strong the amplitudes of the data code modules are and therefore how good the modules can be distinguished into dark and light modules. In contrast to the quality grade for the modulation, the calculation of the reflectance margin considers the knowledge about the correct classification of the modules into dark and light modules, which is available after the decoding of the symbol.

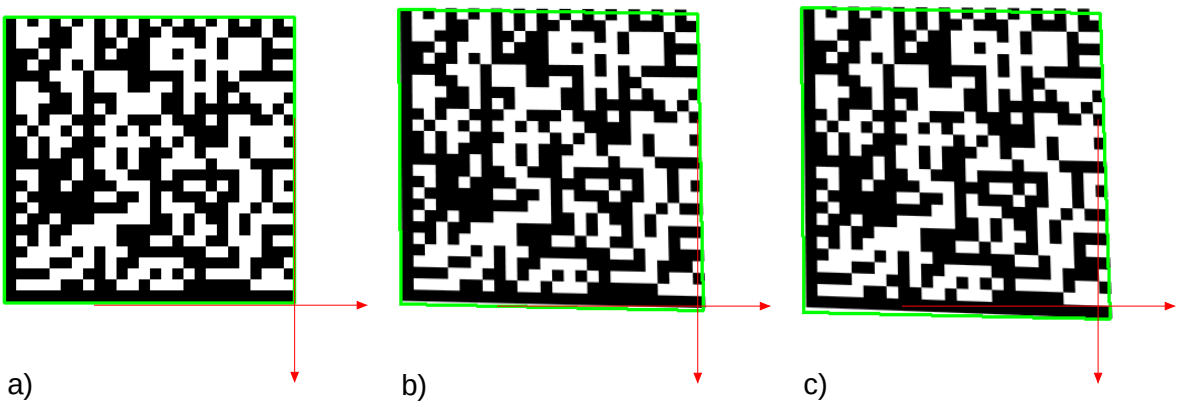

Figure 6.6: Quality grades for grid non-uniformity: (a) 4, (b) 2, and (c) 0.

6.1.1.9 Print Growth (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

The print growth is no quality grade but indicates to which extent the dark and light modules fill out their module boundaries. For example, if too much toner was used when printing the data code symbol, the dark modules may be very large compared to the light modules. The print growth does not influence the overall quality. Note that the calculation of the print growth is not conform with the ISO/IEC 15415:2011 standard but depends on the standards for the specific data code types. Further information is provided with the description of `get_data_code_2d_results`.

6.1.1.10 Contrast Uniformity (Data Matrix ECC 200, QR Code, Micro QR Code, and Aztec Code)

In contrast to modulation, which is a quality grade that concerns the complete symbol, contrast uniformity describes the minimum value of the modulations that were calculated for the individual modules in the data region of the symbol. The contrast uniformity is no quality grade and thus does not influence the overall quality.

6.1.1.11 Unused Error Correction (all Symbol Types)

The quality grade for unused error correction quantifies the reserve in error correction that is still available after reading the data code. A perfect data code requires no error correction at all, resulting in a quality grade of value 4 (see figure 6.7, a). The more defective a data code becomes, the more the inherent error correction needs to be applied to still be able to read the data code. Figure 6.7 shows two symbols (b and c) for which degradations occur inside the data patterns. Dependent on the amount of the degradations, the values for the quality grade decrease to 2 and 0.

6.1.1.12 Start/Stop Pattern (PDF417)

The quality grade for the start/stop pattern of a symbol of type PDF417 evaluates the quality of the reflectance profile as well as the correctness of the sequence of the light and dark bars for the start and stop patterns.

6.1.1.13 Codeword Yield (PDF417)

The quality grade for the codeword yield evaluates how many of the attempts to read the individual codewords were successful.

6.1.1.14 Decodability (PDF417)

The quality grade for the decodability describes to which amount the light and dark bars are metrically correct, i.e., it measures the deviations between the actual widths of the bars from the theoretical widths that can be derived from the size of the symbol and the known number of bars (17) per codeword.

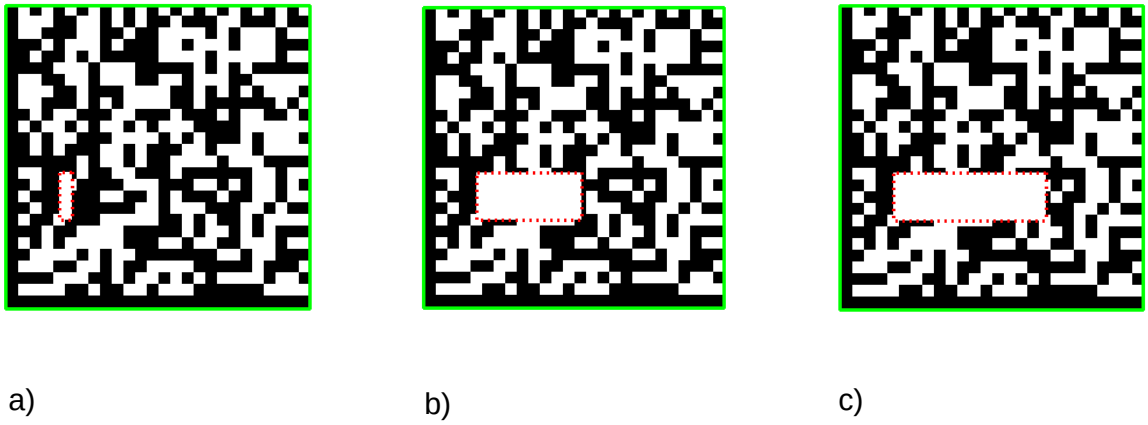

Figure 6.7: Quality grades for unused error correction: (a) 4, (b) 2, and (c) 0.

6.1.1.15 Defects (PDF417)

The quality grade for defects describes the amount of radiometric degradations inside the individual modules or the quiet zone, i.e., dark spots inside the light modules, the light parts of the finder pattern, or the quiet zone, and light spots inside the dark modules or the dark parts of the finder pattern (see figure 6.8).

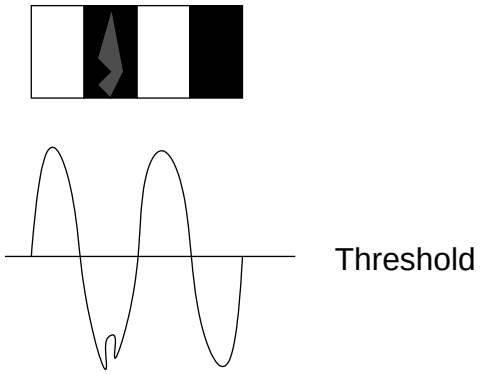

Figure 6.8: Small defect in a black bar.

6.1.2 Acquire Raw Data

When investigating the reason for quality defects, it can be useful to query the data that was used to calculate the values for the print quality elements. For this, you call the operator `get_bar_code_result` with the parameter value `'quality_isoiec_15415_values'`. The operator then returns a tuple with the following raw values:

ECC200, QR Code, Micro QR Code, Aztec Code: `'Contrast'`, `'Axial Non-Uniformity'`, `'Unused Error Correction'`;

PDF417: `'Codeword Yield'`, `'Unused Error Correction'`;

The parameter `'quality_aimdpm_1_2006'` always returns a tuple with a value for each grade. It therefore displays `'N/A'` if a grade is not on the above list and no raw value is consequently available.

6.2 The AIM DPM-1-2006 Standard

Another possibility of evaluating the print quality of ECC 200, QR data codes, Micro QR Codes and Aztec Codes is using HALCON's data code quality inspection following the AIM DPM-1-2006 standard, an extension to ISO/IEC

15415:2011. It returns a tuple representing the twelve print quality elements

- 'Overall Quality',
- 'Cell Contrast',
- 'Cell Modulation',
- 'Fixed Pattern Damage',
- 'Decode',
- 'Axial Non-Uniformity',
- 'Grid Non-Uniformity',
- 'Unused Error Correction',
- 'Mean Light',
- 'Reflectance Margin',
- 'Pront Growth',
- and 'Contrast Uniformity'.

All grades except 'Mean Light' have the same meaning as the ISO/IEC 15415:2011 grades with two exceptions: 'Contrast' and 'Modulation' are renamed 'Cell Contrast' and 'Cell Modulation', respectively, to reflect differences in the methods specified with both standards for estimating those values.

'Mean Light' is not a grade but an estimation for the quality of the processed image computed as the mean gray-scale value of the centers of the bright data code modules, a value between 0.0 and 1.0, corresponding to 0% to 100% of the maximum gray-scale value (255 for byte images). The mean light is estimated to reach reproducible results, which can be achieved because

- the mean light condition guarantees a unified image quality among different systems and
- the calibration compares the 'light' used to evaluate a known calibration object and the 'light' used to evaluate a physical data code in the quality inspection process. Codes with a contrast that is too large or too weak are rejected.

Note that the AIM DPM-1-2006 standard recommends special lighting environments. These include diffuse on-axis illumination (90), diffuse off-axis illumination (D) and directional illumination (30Q, 30T, 30S). For more information about the lighting environments refer to the comments within the HDevelop example program %HALCONEXAMPLES%\hdevelop\Identification\Data-Code\calibration_aimdpm_1_2006.hdev or the more detailed description in the document AIM DPM-1-2006 with the title 'Directed Part Mark (DPM) Quality Guideline'.

6.2.1 Acquire Raw Data

When investigating the reason for quality defects, it can be useful to query the data that was used to calculate the values for the print quality elements. For this, you call the operator get_bar_code_result with the parameter value 'quality_aimdpm_1_2006_values'. The operator then returns a tuple with the following raw values:

ECC200, QR Code, Micro QR Code, Aztec Code: 'Contrast', 'Axial Non-Uniformity', 'Unused Error Correction';

PDF417: 'Codeword Yield', 'Unused Error Correction';

The parameter 'quality_aimdpm_1_2006' always returns a tuple with a value for each grade. It therefore displays 'N/A' if a grade is not on the above list and no raw value is consequently available.

6.2.2 How to Apply the AIM DPM-1-2006 Standard

The AIM DPM-1-2006 requires interactive image acquisition. Much of the functionality cannot be provided within an operator. Instead, HALCON provides two example applications:

- `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\calibration_aimdpm_1_2006.hdev` performs the so-called reflectance calibration and stores the calibration results (see section 6.2.4 on page 54).
- `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\print_quality_aimdpm_1_2006.hdev` performs the print quality inspection based on the calibration results (see section 6.2.5 on page 54).

Figure 6.9 shows an overview of the complete progress of reflectance calibration and print quality inspection.

6.2.3 How to Adapt the Example Programs To Your Image Acquisition Device

Note that both example programs contain parts that must be adapted to your image acquisition device / image acquisition interface. In the programs, they are marked with a special comment (how to search for specific text within HDevelop programs is explained in the HDevelop User's Guide in section 6.2.2 on page 59).

```
* PLEASE ADAPT THESE VARIABLES TO YOUR APPLICATION
```

First, you can decide whether you just want to see what the program does first, and therefore set `Offline = true`, or whether you want to perform the actual calibration or print quality inspection and set `Offline = false`. If you decide to use the online mode, proceed to set your parameters. First, adapt the name of the used image acquisition interface in the variable `InterfaceName`.

The AIM DPM-1-2006 standard evaluates the 'light' used for calibration and print quality inspection based on the so-called system response. This system response is influenced by the parameters

- `ExposureTime`, which controls how much light reaches the camera chip, and
- `Gain`, which controls how the incoming light is transferred to gray values on the chip.

In the example programs, you must adapt the corresponding operator calls to your image acquisition device / interface. In particular, replace the parameter names `'GainSelector'`, `'Gain'`, and `'ExposureTime'` with the ones used by your image acquisition interface. In the program, starting values are set first:

```
set_framegrabber_param (AcqHandle, 'GainSelector', 'All')
set_framegrabber_param (AcqHandle, 'ExposureTime', 4000.0)
```

Then, adapt the procedure `adjust_system_response` to your image acquisition device / interface. The procedure first queries the available ranges of values:

```
get_framegrabber_param (AcqHandle, 'ExposureTime_range', ExposureRange)
get_framegrabber_param (AcqHandle, 'Gain_range', GainRange)
```

and the current values:

```
get_framegrabber_param (AcqHandle, 'Gain', GainValue)
get_framegrabber_param (AcqHandle, 'ExposureTime', ExposureValue)
```

Depending on the value of the parameter `IncreaseLight`, the procedure then increases (`IncreaseLight = 1`) or decreases (`IncreaseLight = -1`) the system response. Again, you must replace the parameter names used in the calls to `set_framegrabber_param` with the ones of your image acquisition interface. If your interface does not provide parameters for exposure and gain, you might try to adapt the complete code of the procedure or ask your distributor.

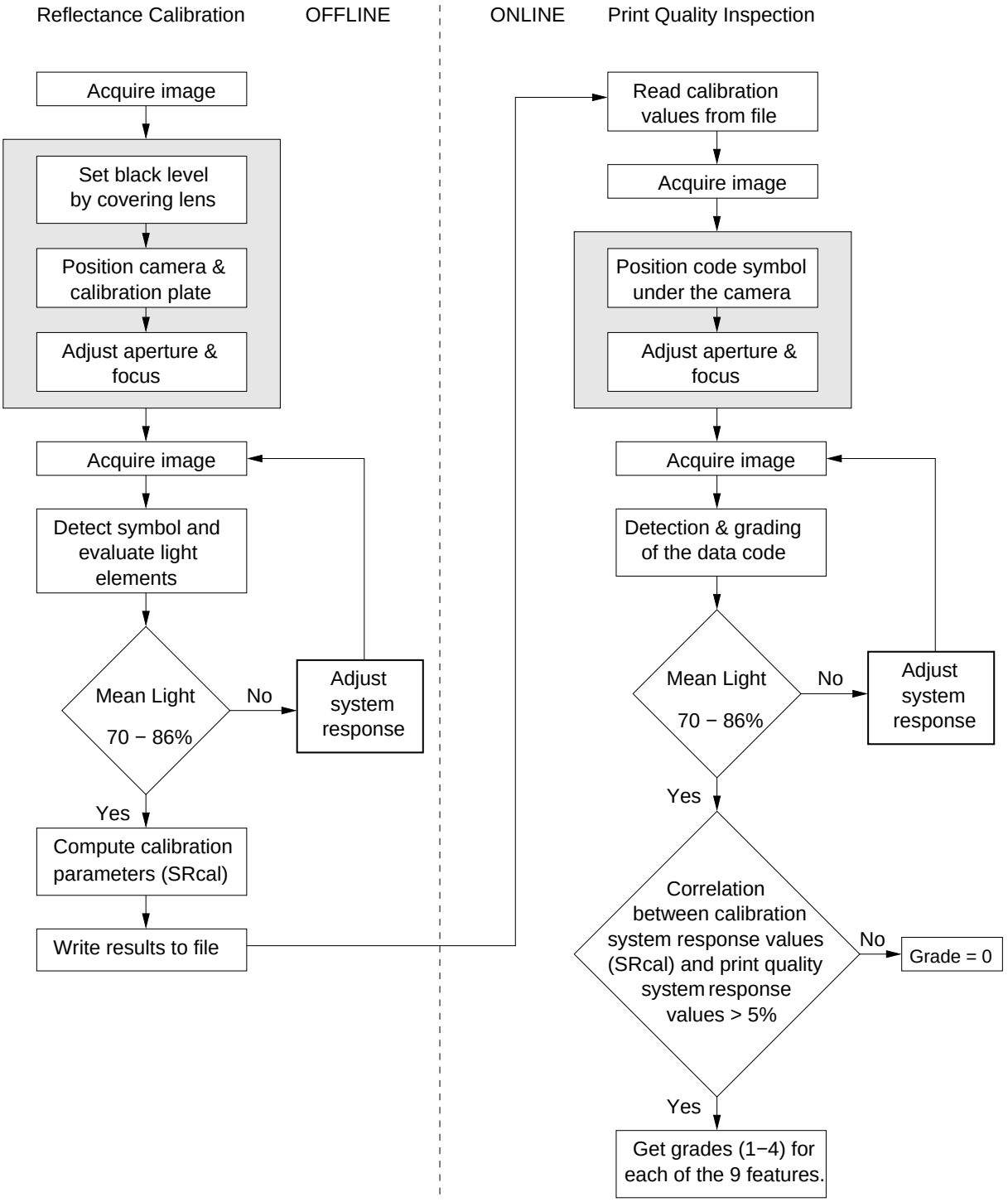

Figure 6.9: Flow chart of the reflectance calibration and the print quality inspection. The sections that are highlighted with gray require user interaction.

After adapting the procedure, we recommend to make it external by unchecking the box `Local` in the procedure box as described in the HDevelop User's Guide in the section 6.4.5.1 on page 121 and then saving the program. Now, if you want to perform data code quality inspection with your adapted procedure and the program `%HALCONEXAMPLES%\hdevelop\Identification\Data-Code\print_quality_aimdpm_1_2006.hdev`, just delete the existing `adjust_system_response` procedure and the program will automatically use the external procedure with the same name.

6.2.4 Reflectance Calibration

To calibrate your system, use the example program %HALCONEXAMPLES%\hdevelop\Identification\Data-Code\calibration_aimdpm_1_2006.hdev. The step described there is called reflectance calibration. To perform this calibration you need a calibrated conformance standard test card (e.g., a NIST-traceable EAN/UPC calibrated conformance test card). Currently, there are only cards available with printed 1D bar code symbols, so they are also used for the reflectance calibration of 2D data code symbols. At first you have to setup your system. The example program contains all the code necessary for the reflectance calibration. However, it is also an example that can be run as it is and for this purpose, it uses a stored sequence of images. Thus, you must first adapt the example to your application.

```
* Set the value of code type to the code type printed on the
* calibration plate.
CodeType := 'EAN-13'
* Set the value of the reference aperture of the laser (see
* calibration plate). If no reference aperture is denoted, set
* the X Dimension value to 1.
RefAperture := 1
* Set the value of X Dimension printed on the calibration plate.
* If no X Dimension is denoted, set the X Dimension value to 1
XDimension := 1
* Set initial value of bar code handle to invalid value -1.
```

The program guides you through the necessary steps. First, it asks you to cover the lens for setting the black level. If your image interface does not provide black-level compensation, please ask your distributor.

```
min_max_gray (Image, Image, 0, Min, Max, Range)
set_framegrabber_param (AcqHandle, 'BlackLevelSelector', 'All')
```

Then, position the code horizontally in the center of the image and adjust focus and aperture (F-number) so that the code is imaged with a good contrast.

Now, the interactive part of the calibration is finished.

During the calibration, the code symbol on the calibration plate is detected, its bright code elements are segmented and, using the gray values in the center of each element, the value of mean light is calculated from the maximum gray value.

If the gray value does not lie in a range between 70% and 86%, the system response is adjusted by calling the procedure adjust_system_response. If the mean light value lies over 86%, then the symbol is too bright and the system response values are adapted to reduce the brightness. The contrary happens if the mean light value lies below 70%. This procedure repeats until the mean light is within the requested range or the system response values cannot be adjusted any more. Then, the calibration values are returned and written to a file.

```
write_calib_results (A, MLCal, Rcal, SRexp, SRgain, ResultFileName, \
                     ResultValueNames)
```

6.2.5 Inspecting Print Quality

The program %HALCONEXAMPLES%\hdevelop\Identification\Data-Code\print_quality_aimdpm_1_2006.hdev is used to demonstrate the print quality inspection. It starts by reading the calibration values from file.

```
read_reflectance_calib_file (ReflectanceCalibFile, \
                             ReflectanceCalibResultsValues)
```

In the next step you are asked to position the code symbol parallel to the axis of the image plane and in the center of the field of view under the camera and to adjust the aperture and the focus. The code is detected in the current image and graded according to the AIM DPM-1-2006 grades as listed above. Furthermore, the program returns raw values for some quality grades to investigate the reason for quality defects. Again, the mean light value is checked and the system response is adjusted if the value does not lie between 70% and 86%.

```
elseif (MeanLight < 0.7)
    IncreaseLight := 1
elseif (MeanLight > 0.86)
    IncreaseLight := -1
endif
    adjust_system_response (AcqHandle, IncreaseLight, AdjustParams, \
                            MAXSTEPS, EXPSTEP, AdjustmentImpossible, \
                            AdjustParams)
```

Once the mean light value lies within the specifications, the system response values of the calibration and of the print quality inspection are compared. If the correlation between those values is higher than 5%, a grade is returned for each print quality value. Otherwise, the grade 0 is returned, indicating that either the print quality of the code is too low or your setup is not suitable for the code that is to be inspected.

```
if (Rtarget > 0.05)
    Grade := 1
    disp_message (WindowHandle, \
                  'The quality grades of the 2d data code symbol:', \
                  'window', 10, 10, 'black', 'true')
    get_data_code_2d_results (DataCodeHandle, ResultHandles[0], \
                              'quality_aimdpm_1_2006_labels', QALabels)
else
    Grade := 0
```

Index

axial non-uniformity (data code), 48

check print quality of data code, 45
contrast (data code), 46
contrast uniformity (data code), 49

data code contrast, 18
data code contrast tolerance, 19
data code maximum slant, 20
data code minimum contrast, 19
data code model, 11
data code types, 7
decode feature (data code), 48

fixed pattern damage (data code), 47
format (Aztec Code), 17

global data code model parameters, 11
grid non-uniformity (data code), 48

inspect data code(s), 29
inspect results for speed up (data code), 31
inspect results for troubleshooting (data code), 33

model parameter ranges (data code), 41
model type (data code), 16, 17
modulation (data code), 46
module gap (data code), 19
module grid (data code), 20
module mirrored (data code), 18
module polarity (data code), 17
module size (data code), 19

optimize model of data code, 11
overall (data code), 46

position detection (data code), 21
print growth (data code), 49

re-use data code model, 23
read difficult data codes, 25
read with geometric distortions (data code), 38
read with large module gaps (data code), 26
read with noise (data code), 27
read with radiometric distortions (data code), 39
read with slant or perspective distortion (data code), 25
reflectance margin (data code), 48
requirements (data code), 39
result persistence (data code), 21
result strictness (data code), 21
result timeout (data code), 21

specific data code model parameters, 15
speed up data code reader, 21
start stop pattern (data code), 49
symbol shape (data code), 16
symbol size (data code), 16

train model of 2D data code, 13
troubleshooting (data code), 29