



HALCON

the Power of Machine Vision

Quick Guide

7.1

A quick access to the functionality of HALCON, Version 7.1.4

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.

Edition 1	December 2003	(HALCON 7.0)
Edition 1a	July 2004	(HALCON 7.0.1)
Edition 2	July 2005	(HALCON 7.1)
Edition 2a	April 2006	(HALCON 7.1.1)
Edition 2b	December 2006	(HALCON 7.1.2)

Copyright © 2008 by MVTec Software GmbH, München, Germany



Microsoft, Windows, Windows 95, Windows NT, Windows 2000, Windows XP, Microsoft .NET, Visual C++, Visual Basic, and ActiveX 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/>

Contents

1	Introducing HALCON	1
1.1	Key Features	2
1.2	Who Should Use HALCON?	3
1.3	Required Knowledge	3
1.4	Getting Started with HALCON	4
1.5	Where to Get More Information	5
2	How to Develop Applications with HALCON	7
2.1	A Look Under the Surface of HALCON: Architecture and Data Structures	9
2.2	Quick Start with HDevelop	15
2.3	Using HALCON Within Programming Languages	16
2.4	Extending HALCON	18
2.5	Limitations	19
3	Machine Vision Methods	21
3.1	Image Acquisition	22
3.2	Region Of Interest	27
3.3	Blob Analysis	39
3.4	1D Measuring	53
3.5	Edge Extraction (pixel-precise)	64
3.6	Edge Extraction (subpixel-precise)	74
3.7	Contour Processing	84
3.8	Template Matching	98
3.9	Color Processing	111
3.10	1D Bar Code	125
3.11	2D Data Code	134
3.12	OCR	145
3.13	Stereo	163
3.14	Visualization	173
4	Industries	189
4.1	Electric Components And Equipment	190
4.2	Food	193
4.3	Health Care And Life Science	194
4.4	Iron, Steel And Metal	197

4.5	Machinery	199
4.6	Photogrammetry And Remote Sensing	204
4.7	Printing	208
4.8	Rubber, Synthetic Material, Foil	209
4.9	Semiconductors	211
5	Application Areas	221
5.1	1D Bar Codes	222
5.2	2D Data Codes	223
5.3	Completeness Check	224
5.4	Measuring And Comparison 2D	226
5.5	Measuring And Comparison 3D	231
5.6	Optical Character Recognition	234
5.7	Position Recognition 2D	235
5.8	Print Inspection	237
5.9	Object Recognition 2D	239
5.10	Surface Inspection	241

Chapter 1

Introducing HALCON

HALCON defines the state of the art in machine vision software. It provides a comprehensive vision library and is always based on the latest and most advanced technology. Whatever your task, HALCON will solve it, fast and with highest accuracy.

Vision Development Environment

A professional image processing tool must be more than just a library of image processing operators. Solving image processing tasks is just one part of a complete solution, which comprises other software components like process control or database access, and hardware components from illumination to image acquisition devices and many other mechanical components. Therefore, it is important that the image processing system is easy to use and can be integrated into the development cycle in a flexible manner.

To achieve this, HALCON takes care of all important aspects:

- The *software development* is supported by the integrated development environment HDevelop, which enables a quick development of image processing tasks combined with an easy integration into standard development environments like Microsoft Visual C++ via HDevEngine or automatic code export.
- The *problem-oriented documentation* covers all levels from a quick access to important information up to a detailed discussion of advanced topics.
- These descriptions are combined with hundreds of *examples* for an intuitive understanding of the solutions, which can serve as templates to shorten the development time.
- *Last but not least, HALCON provides open interfaces* for efficient data exchange, to integrate own operators, or to access specialized hardware round off the system.

Vision Library

HALCON fulfills all requirements of a professional vision library:

- It comprises methods for all standard and advanced types of image processing from image acquisition from many different devices up to the advanced shape-based matching.
- Apart from image processing functionality, HALCON provides tools that are typically needed in the context of machine vision applications, e.g., for the communication via sockets or the serial interface, file handling, data analysis, arithmetic operations, or classification.
- HALCON offers flexible ways of parallelization to exploit multi-processor or multi-core hardware to speed up an application.
- The HALCON library that is used in an application will not be visible to the end user and requires only minimum resources in an installation, which makes it perfect for OEM developments.

1.1 Key Features

Leading-Edge Technologies

In addition to the full set of standard machine vision methods, HALCON offers functionality that is outstanding in the field of machine vision libraries, e.g., 3D camera calibration, shape-based and component-based matching, subpixel-precise edge and line extraction, subpixel contour processing, reconstruction via binocular stereo, arbitrary regions of interest, and much more.

Apart from this, many methods that are known from other libraries are offered with a much better performance. An example for this is the *morphology*, which is up to 100 times faster than in other products, and at the same time offers much more flexibility.

One Software for All Applications

Thanks to its more than 1150 operators, HALCON is at home in all areas of research, development, and production where images are processed and analyzed. Numerous customers all over the world already use HALCON to solve their machine vision tasks.

Protection of Investment

By choosing HALCON, you choose independence: Switch to another operating system? HALCON supports a wide range of Windows NT/2000/XP, Linux, and UNIX platforms, including x64 systems. Migrate your applications from C++ to C#? HALCON can be used within various programming languages and environments. Your application grows and needs more computing power? Switch to a multi-processor or multi-core computer and HALCON will [automatically parallelize its execution](#) (see section 2.1.3 on page 14). Last but not least, you are free to choose the image acquisition hardware that fulfills your requirements, because HALCON provides ready-to-use interfaces to a large number of image acquisition devices (analog, digital, IEEE 1394, CameraLink).

Rapid Prototyping

In many cases it is important to determine quickly if and how a problem can be solved. With HDevelop, HALCON's integrated development environment (IDE), you can rapidly develop machine vision applications. Besides being a fully-fledged program interpreter with debug functions, HDevelop assists you

actively, e.g., by suggesting operators and by automatically visualizing the result of an operation. With the help of integrated tools you can inspect images and results and quickly find suitable parameter values that solve your vision task.

Open Architecture

HALCON offers a comprehensive vision library but does not claim to be all-encompassing. Therefore, it is based on an open architecture. Thus, you can extend HALCON by integrating your own vision functionality in form of [new operators](#) (see section 2.4.1 on page 18). And if you want to use a frame grabber that HALCON does not yet support you can use the images directly or create an interface for your frame grabber.

1.2 Who Should Use HALCON?

To put it shortly: everybody in need of a machine vision software.

HALCON is especially designed to be used by:

- *OEMs* to develop machines that include vision components, e.g., for chip or print inspection, or to develop software solutions, e.g., for car plate reading or cell analysis,
- *System integrators* to develop customer-specific machine vision solutions,
- *VARs* that bundle HALCON with other products, and
- *Research groups and universities* that profit from the unbeatable completeness of the library, typically in combination with the integrated development environment HDevelop for intuitive prototyping.

1.3 Required Knowledge

Image Processing

Of course, the more familiar you are with the terminology and the standard methods of image processing, the easier it is to apply HALCON to solve your machine vision task. To make it even easier, this manual briefly describes the standard methods and how to apply them with HALCON in [chapter 3](#) on page 21. As an alternative, you can start out in [chapter 4](#) on page 189 or [chapter 5](#) on page 221, which present examples from various industries and application areas and includes references to the used methods.

Programming


If you want to use HALCON via a programming language, you must be familiar with this language and the corresponding development tools. The [Programmer's Guide](#) concentrates on describing HALCON's language interfaces, i.e., the provided data structures and classes, how to call operators, etc.

Operating System

You will need basic knowledge about your platform's operating system in order to install HALCON and the corresponding license.

1.4 Getting Started with HALCON


To get a first impression of HALCON, we recommend to install the demo version of HDevelop, which can be used without a license. It provides the full image processing functionality of HALCON; the main restrictions are that the frame grabber interface cannot be used and that programs cannot be saved.

 Equipped with the demo version of HDevelop, you can set out and explore the possibilities that HALCON offers you by running the example programs described in the following chapters. See [section 2.2 on page 15](#) for a brief introduction to HDevelop.


You can either download HALCON from MVTec's web server or install it from CD. Note that not all HALCON releases are available on CD.

To evaluate the full power of HALCON, e.g., online image acquisition or using HALCON from a programming language, you can obtain a temporary license from your local distributor free of charge. Detailed information about licensing can be found in the [Installation Guide](#).

1.4.1 How to Install the HALCON Demo Version from CD

 Detailed information about the installation process can be found in the [Installation Guide](#).

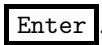
1.4.1.1 Windows NT/2000/XP Platforms

 For the following steps, you need administrator privileges.

- Insert the CD in your CD-ROM drive. If the installation process does not start automatically, run `Setup.exe` in the directory `install-windows`.
- The setup program is self-explanatory and guides you through the installation process.
- The setup allows you to choose between different types of installations; select Demo to install the demo version.

After the installation, you can start the demo version of HDevelop from the Windows start menu without further action.

1.4.1.2 UNIX Platforms

- Insert the CD in your CD-ROM drive and mount the corresponding device. You probably need root privileges to mount the device correctly.
- Run the shell script `install-unix` in the top-level directory of the CD.
- The script asks for the installation path of HALCON and allows to select which optional components are installed. Simply accept the suggested defaults by pressing .
- After the installation, you must set some environment variables. The setup program informs you about the necessary actions.

To start the demo version of HDevelop, execute `hdevelop_demo` from a shell.

1.4.2 How to Install the HALCON Demo Version via WWW

- MVTec's download area can be found under <http://www.mvtec.com/download>. Note that you must first register before downloading the software.
- Select the demo version of HALCON and your operating system. Then, follow the instructions.

On a Windows NT/2000/XP system, you can start the demo version of HDevelop from the start menu. On a UNIX system, execute `hdeveloP_demo` from a shell.

1.5 Where to Get More Information

- *This Manual*

Information about developing applications in HDevelop or in a programming language can be found in [chapter 2](#) on page 7.

[Chapter 3](#) on page 21 describes the main machine vision methods and how to use them in HALCON. It also includes examples.

The following two chapters present HALCON example programs sorted by [machine vision industry](#) (see [chapter 4](#) on page 189) and by [application area](#) (see [chapter 5](#) on page 221), respectively.

- [Installation Guide](#)

This manual explains the different licensing methods and supplies detailed information about installing, upgrading, and uninstalling HALCON.

- [HDevelop User's Manual](#)

This manual is your guide to using HDevelop, HALCON's integrated development environment (IDE). It describes the elements of its graphical user interface, the language used in HDevelop programs, how to export programs to other programming languages, and more.

- [Programmer's Guide](#)

If you want to use HALCON from a programming language consult this manual for detailed information about the provided interfaces and their data types, classes, etc. Furthermore, this manual explains how to use HDevEngine to execute HDevelop programs and procedures from a programming language.

- [Extension Package Programmer's Manual](#)

This manual explains how to integrate your own functionality into HALCON via the so-called extension packages.

- [Frame Grabber Integration Programmer's Manual](#)

If you want to use a frame grabber that is not yet supported by HALCON this manual explains how to create your own frame grabber interface for it.

- [Application Guide](#)

Currently, this guide consists of the following Application Notes:

Application Note on Image Acquisition: This manual shows how to use simple and complex configurations of image acquisition devices, explains the various timing modes, and more.

Application Note on Shape-Based Matching: Here you can find out about how to use HALCON's shape-based matching to find objects based on a single model image and to locate them with subpixel accuracy.

Application Note on 2D Data Codes: This manual shows how to use HALCON's 2D data code reader.

Application Note on 1D Metrology: This manual shows how to detect edges and how to measure their position and distance along lines and arcs.

Application Note on 3D Machine Vision: This manual is your gate into the world of 3D machine vision.

- Reference Manuals

The reference of all HALCON operators is available in [HDevelop](#), [C++](#), [COM](#), and [C](#) syntax.

- Release Notes

If you have used an earlier version of HALCON, please take a look at the release notes, which can be found in the file `release_notes.html` in the directory into which you have installed HALCON.

- Example Programs

HALCON provides an extensive set of example programs, not only for HDevelop but also for different programming languages. These examples can be found in the subdirectory `examples` of the folder into which you have installed HALCON. Many of the examples are described in this manual.

- Support

Your local distributor is the competent contact for all questions about HALCON. Please take a look at <http://www.mvtec.com/halcon/support> for the current list of distributors.

- Training Seminars

Some distributors offer training seminars where you can learn more about HALCON (see <http://www.mvtec.com/halcon/support/training.html> for a list of upcoming seminars).

Chapter 2

How to Develop Applications with HALCON

2.1	A Look Under the Surface of HALCON: Architecture and Data Structures	9
2.2	Quick Start with HDevelop	15
2.3	Using HALCON Within Programming Languages	16
2.4	Extending HALCON	18
2.5	Limitations	19

HALCON offers many ways for the application development. But to make full use of the architecture the mode depicted in [figure 2.1](#) is recommended.

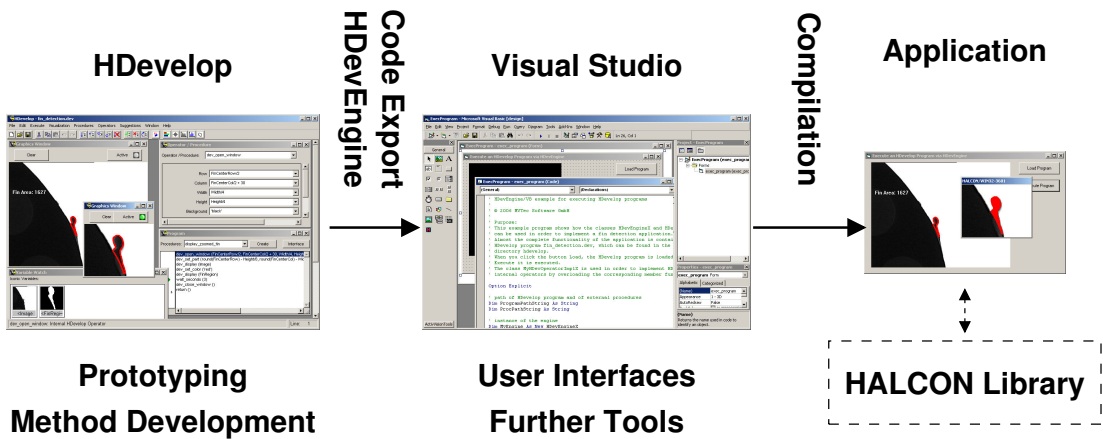


Figure 2.1: Three-step approach for the application development.

- Image inspection, prototyping of the vision method, and the final development of the vision method are done within HDevelop. Here, the program is structured into procedures in which each procedure represents one sub task, like initialization, processing, and cleanup. The main program is used only as a test environment to call the procedures by passing images and receiving the results.
- The complete application is developed in a programming environment like Microsoft Visual Studio. Regarding the integration of the HDevelop procedures, you can choose between two ways: Either you export the procedures to your programming language and then import them, e.g., via an include statement. Alternatively, you can directly execute the HDevelop procedures using HDevEngine. The user interface and other necessary code is implemented using the normal mechanisms offered by the given language. Finally, the project is compiled and linked.
- Together with the HALCON library, the generated program represents the solution that can, e.g., be loaded onto the destination machine or sent to a customer.

An overview of the philosophy of developing with HALCON can be seen in [figure 2.1](#).

The three-step approach has several advantages:

- Whenever needed the vision part can easily be optimized or extended because HDevelop offers much better inspection and debugging facilities for image data than the standard programming environments.
- If you are using HDevEngine, you do not even need to compile and link your application after a change in the HDevelop program (if you did not change the signatures of the procedures). If you are using exported code, you typically do not need to modify the rest of the application but only to compile and link the application again.
- Because the vision part is separated from the general code it can easily be executed in a stand-alone manner. Furthermore, it can be given to others without the need to pass the whole project.

Especially in the case of support questions, the HDevelop program with one or more images can quickly be sent to the distributor.

- Finally, a reuse for other architectures like Linux can easily be achieved because HDevelop runs in exactly the same manner on multiple operating system - quite unlike a program developed with Microsoft tools.

For basic information on the prototyping with HDevelop see [section 2.2](#) on page 15. How to export code from HDevelop is explained in the HDevelop User's Manual in [chapter 4](#) on page 113. A brief introduction to using HALCON in a programming environment can be found in [section 2.3](#) on page 16. How to use HDevEngine is explained in the Programmer's Guide in [part V](#) on page 117. For using a programming environment like Microsoft Visual C++, see the relevant Microsoft documentation.

2.1 A Look Under the Surface of HALCON: Architecture and Data Structures

HALCON's architecture, data structures, and internal mechanisms were developed according to the philosophy that they should be

1. efficient
2. open
3. standardized
4. self-describing

Efficient means that the execution time of each HALCON operator should be as short as possible. Furthermore, the operator design has been made such that combinations that are standard sequences or more complex tasks must still remain efficient.

The open architecture is important in two respects: First, it must be possible to make use of HALCON from many different languages. Here, passing of external data to HALCON and accessing internal data of HALCON must also be supported. Finally, there must be transparent interfaces to integrate user-defined operators and non-standard image acquisition devices. This open architecture allows, e.g., a simple update to a new version of a frame grabber interface without changing the installation of HALCON.

Standardized means that the signatures, naming, and usage of operators and data structures must follow strict rules. This allows a quick learning combined with few possible errors.

Finally, HALCON provides detailed information about each operator and their parameters not only in the documentation but also online via specialized operators.

The basic architecture can be seen in [figure 2.2](#). The main part is the image processing library, which consists of a huge number of operators. They provide all the functionality by performing various operations on the available data structures. These operators are accessed via the so-called language interfaces. They are libraries which allow direct use of the operators in the typical programming style of the different programming languages. Finally, there are extra libraries for dynamically loading extra functionality. On the left side, there are the so-called frame grabber interfaces, while the user extensions can be seen on the right side.

Looking closer into the architecture, two components are important: operators and data structures.



Figure 2.2: Basic architecture of HALCON.

2.1.1 HALCON Operators

Whenever any kind of functionality is used from the HALCON library, it is done via an operator. The current version has more than 1150 of these operators. Most of them comprise multiple methods, which are selected via parameters. A full list of all operators can be found in the Reference Manuals or in the dialog `Operators` of `HDevelop`. Important features of operators are:

- There is no hierarchy among operators. From the software architecture point of view, all operators are on the same level.
- Of course, there are logical groups of operators. This can directly be seen by the classes offered for C++ and COM, where operators processing the same data type are used as members of the corresponding classes.
- Operators have standardized rules for ordering input and output parameters (see [section 2.1.2](#)).
- The design of operators follows the rules of the open architecture. Therefore, you can create your own operators and thus extend HALCON, while getting the same look-and-feel for your own operators (see [section 2.4.1](#) on page 18).
- Many operators can make transparent use of automatic parallelization, which allows an easy way of speeding up the program when using large images on a multi-CPU or multi-core computer (see [section 2.1.3](#) on page 14).

2.1.2 Parameters and Data Structures

Philosophy:

- HALCON has two basic types of parameters: iconic data (images etc.) and control data (integers, handles, etc.), see below.
- The parameters for each operator are arranged in a standardized order: input iconic, output iconic, input control, and output control. Not all of the groups might be needed for a given operator. However, the order remains the same.
- Each operator has a self-describing interface. This description contains, besides the standard documentation, information about parameters like types or value lists, which can be accessed online in the Reference Manuals or in the dialog Operators of HDevelop.
- Input parameters of operators are never modified, which results in a very clear and simple semantics. There are only three operators that do not follow this principle to ensure maximum performance (namely `set_grayval`, `overpaint_gray` and `overpaint_region`).
- The open architecture allows to access internal data and to integrate external data.
- All necessary data structures for 2D image processing like (multichannel) images, region, contours, tuples (a kind of array), etc. are directly supported using an extremely efficient implementation.

A detailed description of the low-level data structures can be found in the Extension Package Programmer's Manual, [chapter 4](#) on page 49. For the corresponding types and classes in the supported programming languages see the [Programmer's Guide](#).

2.1.2.1 Images

Philosophy:

- Images belong to the iconic data.
- The major part of an image are the channels, i.e., matrices containing the gray values of various pixel types.
- For each image, the so-called *domain* specifies which part of the image is processed. It thus acts as a *region of interest* (ROI). The domain is a HALCON region (see [section 2.1.2.2](#)) and can therefore be defined very flexibly (from a simple rectangle to a set of unconnected pixels, see below). For details about ROI handling see [Region Of Interest](#) on page 27.

Pixel Data

An almost arbitrary content is possible, from standard 8-bit gray values to floating-point numbers describing derivatives.

For integer values one, two, and four byte versions (with and without sign) are available. Besides this, floating point and complex images are available. Finally, special data types for describing edge direction or hue values are supported.

Image Channels

A channel corresponds to an image matrix. Each image can have an arbitrary number of channels. All channels of an image have the same size.

Typical cases are: single-channel gray value image, color image with three channels (e.g., RGB), or a multichannel image from a multispectral sensor or as a result of texture filtering.

Coordinate Systems

The origin of an image lies in the center of the pixel in the upper left corner. The individual pixels are accessed using row and column coordinates, like in a matrix. The coordinates range from (0,0) up to (height-1, width-1).

Note that because the origin lies in the *center* of the upper left pixel, the pixels' corners have non-integer coordinates. For example, the pixel in the upper left corner has the corner coordinates (-0.5, -0.5), (-0.5, +0.5), (0.5, -0.5), and (0.5, 0.5).

2.1.2.2 Regions

Philosophy:

- Regions belong to the iconic data.
- A region is defined as a set of pixels. The pixels of a region do not need to be connected. This means that even an arbitrary collection of pixels can be handled as a single region. With the operator `connection` a region can be split into its so-called connected components, i.e., components consisting of connected pixels.
- The implementation of regions is based on an efficient implementation of the so-called runlength encoding. This encoding has many advantages: low memory consumption, efficient processing, and easy handling of regions of interest (domains).
- Because of the implementation based on runlength encoding, it is possible to have overlapping regions, e.g., as the result of a dilation of connected components. This would not be possible with a classical implementation based on label images.
- The coordinates of pixels inside a region are not limited to the coordinates of a given image, the region can be larger than the image, possibly as the result of a dilation operation. Whether a region should be clipped to the maximum image extents can be controlled using the operator `set_system` with the parameter value `'clip_region'`.
- The number of regions for an application is virtually unlimited.

2.1.2.3 XLDs

Philosophy:

- XLD is the abbreviation for eXtended Line Description and comprises all contour and polygon based data.

- XLDs belong to the iconic data.
- Subpixel accurate operators like `edges_sub_pix` return the contours as XLD data.
- A contour is a sequence of 2D control points, which are connected by lines.
- Typically, the distance between control points is about one pixel.
- XLD objects contain, besides the control points, so-called local and global attributes. Typical examples for these are, e.g., the edge amplitude of a control point or the regression parameters of a contour segment.
- Besides the extraction of XLD objects, HALCON supports further processing. Examples for this are the selection of contours based on given feature ranges or segmenting of a contour into lines, arcs, polygons or parallels.

2.1.2.4 Control Tuples

Philosophy:

- Tuples are the generic data type for integer and floating point values as well as strings. A variable of type tuple can be of any of the three basic types.
- Besides single values, arrays of the basic types are supported. Therefore, one variable can contain none, one, or an arbitrary number of values, where the types of each element can be different.
- In most cases, single values are treated in the same way as multiple values. If, e.g., an operator like `area_center` is called with a single region, one value is returned for each result (in the example for the area and the coordinates of the center of the region). When the operator is called with multiple regions, a tuple with the corresponding number of values is returned for each result. For example, if you call `area_center` with four regions, it returns three tuples (one for the area and two for the coordinates of the center), each containing four values corresponding to the four regions.
- The index of tuples range from 0 to the number of values minus 1. **Please note, that iconic tuples start with index 1!**



2.1.2.5 Handles

Philosophy:

- Handles are references to complex data structures, e.g., models for the shape-based matching. For efficiency and data security reasons, not the entire structure but only the handle is passed to the programmer.
- All processing of data is controlled with a unique integer value. These integers are magic numbers that must not be changed and can differ from execution to execution and version to version.
- Examples where handles are used are graphics windows, files, sockets, image acquisition devices, OCR, OCV, measuring, matching, and so on.

2.1.3 Parallel HALCON

To put it in a nutshell, Standard HALCON is optimized for running *sequential programs* on *single-processor or single-core computers*. Under Windows NT/2000/XP, Linux, and Solaris, HALCON is *thread-safe*, i.e., it can be used in multithreaded programs (with some exceptions listed in the Programmer's Guide, [section 1.1](#) on page 3). However, all HALCON operators are executed *exclusively*, thus threads will have to wait for each other.

In contrast, Parallel HALCON supports *parallel programming* (e.g., multithreaded programs) by being thread-safe *and* reentrant. This means that multiple threads can call a HALCON operator simultaneously. Parallel HALCON is available for Windows NT/2000/XP, Linux, and Solaris.

Besides supporting parallel programming, Parallel HALCON *automatically parallelizes operators* if started on multi-processor or multi-core hardware, e.g., a dual-Pentium board. The parallelization mechanism is based on distributing the data which has to be processed, i.e., the images, on multiple threads that run on different processors (so-called *data parallelism*). For example, for a filtering operation on a four-processor board the image will be split into four parts which will then be processed in parallel by four threads executing the (same) filtering operator. Together with HALCON's philosophy for treating images and regions, this form of parallelization is very efficient because images need not to be copied. The degree of parallelization is optimized online to minimize the parallelization overhead. For example, very small images will not be processed in parallel, as the overhead would surpass the parallelization speed-up. Moreover, not all HALCON operators lend themselves to parallelization.

Detailed information on using Parallel HALCON can be found in the Programmer's Guide, [section 1.2](#) on page 4.

Note that Parallel HALCON is designed for *shared-memory systems*, i.e., systems in which multiple processors share a common memory as it is the case for typical multi-processor or multi-core boards. The main reason is that only in a shared-memory system threads can share the HALCON object database and do not need to copy images. This limitation means that Parallel HALCON is not suited to the use on workstation clusters or other multi-processor or multi-core hardware that does not offer shared memory.

2.1.4 Image Acquisition

Currently, HALCON provides interfaces for about forty frame grabbers in the form of dynamically loadable libraries (Windows: DLLs; UNIX: shared libraries). These libraries are installed together with the HALCON libraries. Library names start with the prefix HFG; the libraries starting with parHFG are used by Parallel HALCON.

In the following, we give a brief overview of the HALCON frame grabber interface; please refer to the [Application Note on Image Acquisition](#) for detailed information about this topic.

The HALCON frame grabber interface libraries form the bridge between software provided by the frame grabber's manufacturer and HALCON. They form a common, generic interface that requires a small set of operators only.

If you successfully installed your frame grabber, all you need to do to access it from HALCON is to call the operator `open_framegrabber`, specifying the name of the frame grabber and some additional information, e.g., regarding the connected camera. Then, images can be grabbed by calling the operator `grab_image` (or `grab_image_async`).

Please note that the HALCON frame grabber interfaces may change more frequently than the HALCON library itself. One reason for this is that MVTec continuously develops new interfaces; furthermore, if the software provided by the frame grabber manufacturers changes, e.g., if new features are integrated, the corresponding HALCON interfaces will be adapted. You can find the latest information together with downloadable interfaces (including documentation) under <http://www.mvtec.com/halcon/framegrabber>.

2.2 Quick Start with HDevelop

HDevelop is a powerful integrated development environment for both prototyping and application development. HDevelop is very easy to use. Below, you find a brief introduction. Two tutorials, which show how to run HDevelop programs and how to create them, are located in the subdirectory `doc\html\tutorials` of the folder where you installed HALCON. Detailed information about HDevelop can be found in the [HDevelop User's Manual](#).

To start HDevelop under Windows NT/2000/XP, click on `Start > Programs > MVTec HALCON 7.1 > HDevelop`. Under UNIX, HDevelop is started from the shell by executing `hdevelop`.

To load an example, select the menu `File > Open`. This will open a file selection dialog that shows the main directories of the HDevelop examples (under Windows). For beginners, we recommend to select an example from the directory `Applications`. As an alternative, the menu `File > Open Example Program...` can be used. Here, a dialog is opened that allows you to select examples based on different categories instead of the actual location.

After loading the file, the corresponding program code is displayed in the program window. The used variables - so far not instantiated - can be seen in the variable watch window. The program is now ready for execution.

Steps to run a program:

1. In the toolbar, click the button `Run` to execute the program. To continue at a stop statement, click `Run` again.
2. Besides the button `Run`, HDevelop provides a button to step through a program, executing single lines and displaying the results immediately afterwards. The buttons `Step Into` and `Step Out` are useful if the program contains procedures.
3. To rerun the complete program click the button `Set program to initial state` and then `Run`. To rerun parts only, simply click with the mouse to the left of the desired program line. This will reposition the program counter (green arrow). Now click `Run`.

Hints for understanding the program:

1. At the bottom of the main window, HDevelop provides a status bar. This displays useful information in many cases. Especially during execution, when the program stops to visualize results or waits for a user interaction corresponding instructions are given.

2. For information concerning individual program lines double click them so that the name of the applied operator and its parameters are displayed in the operator/procedure window. There, the help button provides specific information.
3. Many programs will automatically display relevant data in the graphics window. Manual visualization can easily be achieved by double clicking on the icons in the variable watch window.
4. Some examples contain procedures. Program lines containing procedures are marked by green or dark green bars. You can switch between the procedures (including the main procedure) via the combo box `Procedures` in the program window.
5. In many examples expressions like assignments, arithmetic operations (both often represented in the program by “:=”), or control structures like loops occur. If you need help concerning the language used in HDevelop please consult the [HDevelop User’s Manual](#).

Further hints for HDevelop:

1. Depending on the selected installation type, not all images used in an example program might be available. In this case, we recommend to insert the HALCON CD or to install the needed images.
2. Some programs use frame grabbers for image acquisition. If the corresponding frame grabber type is not available, an error message will be raised. In this case, we recommend to either use another example or to modify the parameters to fit to the available hardware. Furthermore, if HDevelop Demo is used, no frame grabber interfaces can be used, including the `File` frame grabber, which reads images from files. If you want to use these programs, please use HDevelop.

2.3 Using HALCON Within Programming Languages

HALCON offers three so-called language interfaces. They are libraries that enable you to call the operators and to use the data types of the HALCON library in an easy way. Two language interfaces are designed for specific languages. These are the C and the C++ interfaces. In contrast, the COM interface is independent of a given language. It can be used, e.g., with Visual Basic, C#, or Delphi.

Independent of which programming language you choose, a suitable interface library (`halconc.*`, `halconcpp.*`, `halconx.*`) together with the HALCON library (`halcon.*`) must be linked to the application. In addition to this, for C and C++ the corresponding include files must be included.

To start the development, we recommend to first check one of the ready-to-run example programs. Here, you can see how the project must be set up and how operators and types are used.

For each language interface, the names of types, classes, the naming conventions of operators, etc. may differ to be compliant with the typical rules that apply for the selected language. For details of the operator signatures, please refer to the corresponding reference manuals, which are available in [HDevelop](#), [C++](#), [COM \(Visual Basic\)](#), and [C](#) syntax.

2.3.1 C

The C interface is the simplest interface supported by HALCON. Each operator is represented by either one or two global functions where the operator name and the parameter sequence is identical to HDevelop. All operators are available in a version that accepts tuples as input. These operators have names that start with "T_". Furthermore, if an operator also accepts single parameters for all control parameters, a simplified operator that only uses basic types (`long`, `double`, and `char*`) is provided. These operators have no "T_" prefix. Two types (`Hobject` and `Htuple`) are offered for iconic and control data. Because C does not offer destructors, you have to free iconic data by calling the operator `clear_obj`. To manipulate, create, destroy, and access tuples, macros are provided. For more information see the Programmer's Guide, [part IV](#) on page 93.

The following code shows a small excerpt of an example program to read an image and to display it in a graphics window.

```
read_image(&Monkey, "monkey");
get_image_pointer1(Monkey, &Pointer, Type, &Width, &Height);
open_window(0, 0, Width, Height, 0, "visible", "", &WindowHandle);
disp_obj(Monkey, WindowHandle);
```

2.3.2 C++

The C++ interface is much more sophisticated than the C interface. Here, the advantages of C++ and object-oriented programming are used, i.e., automatic type conversion, construction and destruction, or grouping functions together with their data into classes. Like in the C interface, global functions for each HALCON operator are provided for a procedural style of programming. Here, the classes `Hobject` and `HTuple` are used. Furthermore, operators can be used as members of classes like `HDataCode2d`, `HMeasure`, or `HShapeModel`. In addition, classes like `HImage` or `HRegion` are used. For more information see the Programmer's Guide, [part II](#) on page 15.

The following code shows a small excerpt of an example program to read an image and to display it in a graphics window and apply some basic blob analysis.

```
HImage      Mandrill("monkey");
HWindow     w(0, 0, 512, 512);

Mandrill.Display(w);
HRegion     Bright = (Mandrill >= 128);
HRegionArray Conn = Bright.Connection();
HRegionArray Large = Conn.SelectShape("area", "and", 500, 90000);
```

2.3.3 Visual Basic

Analogously to C++, two styles for programming are offered: procedural and object-oriented. For the procedural style, the class `HOperatorSetX` provides all HALCON operators, where `HUntypedObjectX`

is used to handle iconic data and the built-in type `Variant` is used for control data. For the object-oriented style, classes like `HDataCode2dX`, `HMeasureX`, or `HShapeModelX` are provided for the central functionality. In addition, classes for iconic data, e.g., `HImageX` or `HRegionX`, are available. For more information see the Programmer's Guide, [part III](#) on page 69

The following code shows a small example program to read an image and apply some basic blob analysis.

```
Dim image As New HImageX
Dim region As HRegionX

Call image.ReadImage("monkey")
Set region = image.Threshold(128, 255)
```

2.3.4 C#

Like Visual Basic, C# uses HALCON via the COM interface. Therefore, everything that was described above also applied to C#. The only difference is that the data type `Variant` is called `Object` in .NET, and hence also in C#.

The following code shows the Visual Basic example given above in C# syntax.

```
HImageX    image = new HImageX();
HRegionX   region;

image.ReadImage("monkey");
region = image.Threshold(128, 255);
```

2.4 Extending HALCON

2.4.1 Extension Packages (User-Defined Operators)

HALCON may easily be extended by new operators. Although HALCON already contains more than 1150 operators for various tasks, you may wish to implement new operators, e.g., in order to access a special hardware or to implement an alternative algorithm. To do so, HALCON provides the Extension Package Interface, which allows the integration of new operators (implemented in C) in the form of so-called extension packages. It contains several predefined routines and macros for the easy handling of image data and memory objects in C. Once a new operator has been successfully integrated, it can be used like any other HALCON operator. The [Extension Package Programmer's Manual](#) contains detailed information about extending the operator library.

2.4.2 Frame Grabber Interfaces

Using a similar mechanism, frame grabber interfaces are integrated using dynamic libraries. This allows you to integrate unsupported frame grabbers without further changes of the rest of the system. How to create and integrate a frame grabber interface is described in the [Frame Grabber Integration Programmer's Manual](#). Furthermore, HALCON is shipped with a template source code that can be used as the basis of an integration.

2.5 Limitations

There are some limitations, which you have to keep in mind, although in most applications you won't come near these limits.

- Maximum image size: 32767×32767
- Maximum number of image matrices in memory: 100000
- Maximum number of iconic objects per parameter: 100000
- Maximum number of channels per image: 1000
- Maximum number of contour points: 200000
- Maximum number of polygon control points: 10000
- Range for coordinates: from -32768 to +32767
- Maximum length of strings: 1024 characters

Chapter 3

Machine Vision Methods

This chapter describes important machine vision methods.

3.1	Image Acquisition	22
3.2	Region Of Interest	27
3.3	Blob Analysis	39
3.4	1D Measuring	53
3.5	Edge Extraction (pixel-precise)	64
3.6	Edge Extraction (subpixel-precise)	74
3.7	Contour Processing	84
3.8	Template Matching	98
3.9	Color Processing	111
3.10	1D Bar Code	125
3.11	2D Data Code	134
3.12	OCR	145
3.13	Stereo	163
3.14	Visualization	173

3.1 Image Acquisition

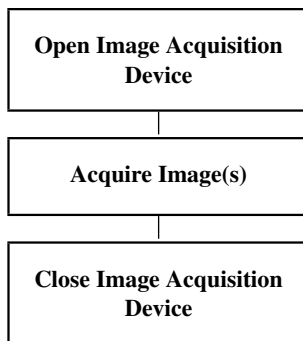
Obviously, the acquisition of images is a task that must be solved in all machine vision applications. Unfortunately, this task mainly consists of interacting with special, non-standardized hardware in the form of the image acquisition device, e.g., a frame grabber board or an IEEE 1394 camera. To let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

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

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

3.1.1 Basic Concept

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



Open Image Acquisition Device

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

There is also a "virtual" frame grabber interface called `File`. As its name suggests, this "frame grabber" reads images from files, and also from so-called image sequence files. The latter are HALCON-specific files, typically with the extension `.seq`; they contain a list of image file names, separated by new lines (you can create it easily using a text editor). If you connect to such a sequence, subsequent calls to `grab_image` return the images in the sequence specified in the file. Alternatively, you can also read all

images from a specific directory. Then, you do not have to create a sequence file, but simply specify the directory name instead of the sequence file as value for the parameter 'CameraType'. Now, subsequent calls to `grab_image` return the images found in the specified image directory. Both approaches are useful if you want to test your application with a sequence of image files and later switch to a real image acquisition device.

Acquire Image(s)

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

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

Close Image Acquisition Device

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

A First Example

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

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

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

```
SequenceName := 'datacode/ecc200/ecc200_cpu_light.seq'
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1,
    'default', SequenceName, 'default', -1, -1, FGHandle)

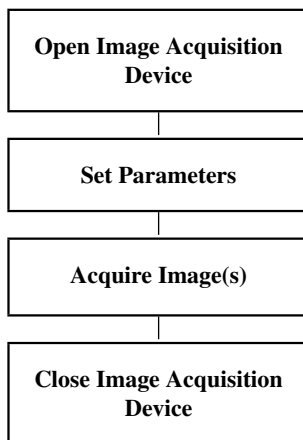
while (1)
    grab_image (Image, FGHandle)

    ... process image ...

endwhile
```

3.1.2 Extended Concept

In real applications, it is typically not enough to tell the camera to acquire an image; instead, it may be important that images are acquired at the correct moment or rate, and that the camera and the frame grabber are configured suitably. Therefore, HALCON allows to further parameterize the acquisition process.



Open Image Acquisition Device

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

With other parameters of `open_framegrabber` you can describe the configuration of image acquisition device(s) and camera(s), which is necessary when using more complex configurations, e.g., multiple cameras connected to different ports on different frame grabber boards. Further parameters allow you to specify the desired image format (size, resolution, pixel type, color space). For most of these parameters there are default values that are used if you specify the values 'default' (string parameters) or -1 (numeric parameters).

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

Detailed information about the parameters of `open_framegrabber` can be found in the Application Note on Image Acquisition (configuring the connection: [section 3](#) on page 7; configuring the acquired image: [section 4](#) on page 12).

Set Parameters

As described above, you already set parameters when connecting to the image acquisition device with `open_framegrabber`. These parameters (configuration of image_acquisition device(s) / camera(s) and image size etc.) are the so-called *general parameters*, because they are common to almost all frame

grabber interfaces. However, image acquisition devices differ widely regarding the provided functionality, leading to many more *special parameters*. These parameters can be customized with the operator `set_framegrabber_param`.

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

Detailed information about setting parameters can be found in the Application Note on Image Acquisition in [section 4](#) on page 12.

Acquire Image(s)

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

HALCON offers many more modes of acquiring images, e.g., triggering the acquisition by external signals or acquiring images simultaneously from multiple cameras. Detailed information about the various modes of acquiring images can be found in the Application Note on Image Acquisition in [section 5](#) on page 16.

3.1.3 Programming Examples

Example programs for all provided frame grabber interfaces can be found in the subdirectory `examples\hdevelop\Image\Framgrabber`. Further examples are described in the [Application Note on Image Acquisition](#).

3.1.4 Selecting Operators

Open Image Acquisition Device

Standard:

`open_framegrabber`

Advanced:

`info_framegrabber`

Set Parameters

Standard:

`set_framegrabber_param`, `get_framegrabber_param`

Acquire Image(s)

Standard:

`read_image`, `grab_image`, `grab_image_async`

Close Image Acquisition Device

Standard:

`close_framegrabber`

3.1.5 Tips & Tricks

Direct Access to External Images in Memory

You can also pass externally created images, i.e., the raw image matrix in the computer's memory, to HALCON using the operators `gen_image1`, `gen_image3`, or `gen_image1_extern`. For an example see the [Application Note on Image Acquisition](#) on page 29.

Unsupported Image Acquisition Devices

If you want to use an image acquisition device that is currently not supported by HALCON, i.e., for which no HALCON frame grabber interface exists, you can create your own interface; how to do this is described in detail in the [Frame Grabber Integration Programmer's Manual](#).

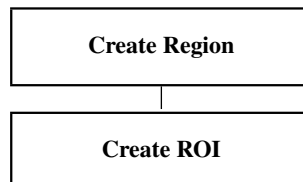
3.2 Region Of Interest

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

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

3.2.1 Basic Concept

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



Create Region

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

Create ROI

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

A First Example

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

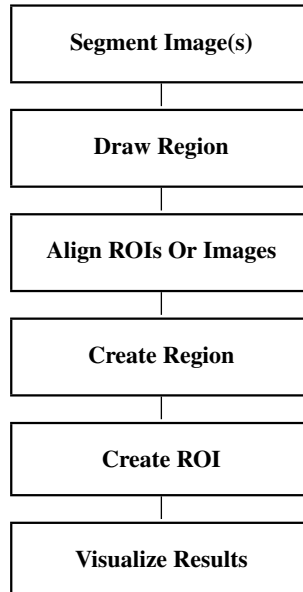
```
read_image (Image, 'mreut')
gen_circle (ROI, 256, 256, 200)
reduce_domain (Image, ROI, ImageReduced)
edges_sub_pix (ImageReduced, Edges, 'lanser2', 0.5, 20, 40)
dev_display (Image)
dev_display (ROI)
dev_display (Edges)
```



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

3.2.2 Extended Concept

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



Segment Image(s)

Very typical for HALCON is the creation of ROIs by a segmentation step. Instead of having a predefined ROI, the parts of the image that are relevant for further processing are extracted from the image using image processing methods. This approach is possible because ROIs are nothing else but normal HALCON regions, and therefore share all their advantages like efficient processing and arbitrary shapes (see [section 2.1.2.2](#) on page 12 for more information on HALCON regions). The segmentation of regions used for ROIs follows the same approach as standard blob analysis. For more details, please refer to the [description of this step](#) on page 40.

Draw Region

The standard way to specify ROIs is to draw the shape interactively using the mouse. To make this easy, HALCON provides special operators for standard shapes and free-form shapes. All operators for this kind of action start with the prefix `draw_`. The drawing is performed by making use of the left mouse button (drawing, picking, and dragging) and finished by clicking the right mouse button. For each such draw-operator HALCON provides operators to generate regions by using the returned parameters (see the description of the step [Create Region](#) on page 27). Operators for mouse interaction can be found in the reference manual in the chapter “[Graphics > Drawing](#)”. More information on user interaction can be also found in the chapter [Visualization](#) on page 173.

Align ROIs Or Images

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

How to perform alignment using shape-based matching is described in the Application Note on Shape-Based Matching in [section 4.3.4](#) on page 36.

Create Region

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

Create ROI

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

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.2.3 Industries

3.2.3.1 Machinery

The example [circles.dev](#) on page 200 shows very well how an ROI can be generated by a segmentation step to reduce the execution time of a subsequent edge extraction.

3.2.3.2 Semiconductors

The program [ic.dev](#) on page 215 from a board inspection application shows very well how one segmentation result can guide the next segmentation by making effective use of ROIs.

3.2.3.3 Photogrammetry And Remote Sensing

In remote sensing, often multiple types of object classes must be extracted. In this case, it is very effective to apply a stepwise segmentation of the image, where one step can focus the next one on specific parts of the image using ROIs. For a corresponding program see [forest.dev](#) on page 204.

3.2.4 Programming Examples

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

3.2.4.1 Processing inside a User Defined Region

Example: [examples\quick_guide\hdevelop\critical_points.dev](#)

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

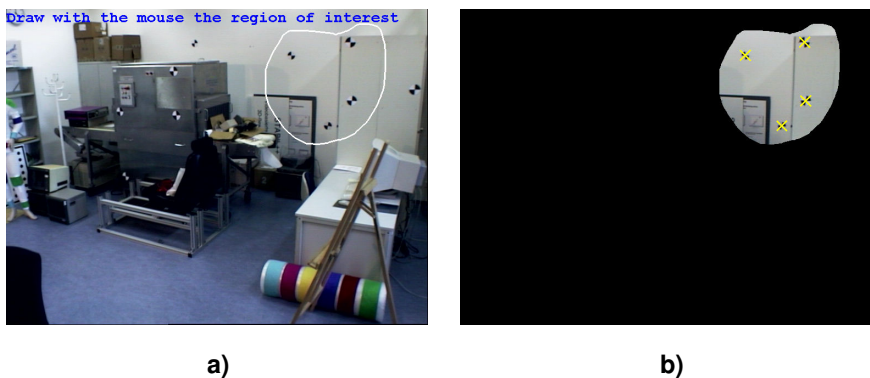


Figure 3.2: (a) Original image with drawn ROI; (b) reduced image with extracted points.

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

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

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

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

3.2.4.2 Interactive Partial Filtering of an Image

Example: `examples\quick_guide\hdevelop\median_interactive.dev`

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



Figure 3.3: Partially filtered image.

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

```

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

```

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

```

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

```

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

```

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

```

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

```

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

```

3.2.4.3 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\3d_information_for_selected_points.dev](#)
Calculating world coordinates for a point in a stereo image pair
→ description in the [Application Note on 3D Machine Vision](#) on page 106
- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_exterior.dev](#)
Measures positions on a caliper rule using camera calibration
→ description in the [Application Note on 3D Machine Vision](#) on page 41
- [examples\application_guide\3d_machine_vision\hdevelop\grid_rectification_ruled_surface.dev](#)
Rectify an arbitrarily distorted image using a regular grid
→ description in the [Application Note on 3D Machine Vision](#) on page 127

- [examples\application_guide\shape_matching\hdevelop\align_measurements.dev](#)
Inspects individual razor blades using shape-based matching to align ROIs for the measure tool
→ description in the [Application Note on Shape-Based Matching](#) on page 36
- [examples\application_guide\shape_matching\hdevelop\first_example_shape_matching.dev](#)
Introduces HALCON's shape-based matching
→ description in the [Application Note on Shape-Based Matching](#) on page 4
- [examples\application_guide\shape_matching\hdevelop\multiple_models.dev](#)
Searching for two types of objects simultaneously
→ description in the [Application Note on Shape-Based Matching](#) on page 25
- [examples\application_guide\shape_matching\hdevelop\multiple_objects.dev](#)
Searches for multiple instances of a security ring
→ description in the [Application Note on Shape-Based Matching](#) on page 34
- [examples\application_guide\shape_matching\hdevelop\multiple_scales.dev](#)
Searches for nuts of different sizes
→ description in the [Application Note on Shape-Based Matching](#) on page 44
- [examples\application_guide\shape_matching\hdevelop\process_shape_model.dev](#)
Creates a model ROI by modifying the result of `inspect_shape_model`
→ description in the [Application Note on Shape-Based Matching](#) on page 10
- [examples\application_guide\shape_matching\hdevelop\reuse_model.dev](#)
Storing and reusing a shape model
→ description in the [Application Note on Shape-Based Matching](#) on page 46
- [examples\hdevelop\Applications\Aerial\forest.dev](#)
Extraction of trees and meadows from forest
→ description [here in the Quick Guide](#) on page 204
- [examples\hdevelop\Applications\Aerial\roads.dev](#)
Extraction of roads from aerial image
→ description [here in the Quick Guide](#) on page 207
- [examples\hdevelop\Applications\Barcode\multiple.dev](#)
Reading multiple bar codes of type Code 39
→ description [here in the Quick Guide](#) on page 222
- [examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev](#)
Measures distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- [examples\hdevelop\Applications\FA\ball.dev](#)
Inspection of ball bonding
→ description [here in the Quick Guide](#) on page 211
- [examples\hdevelop\Applications\FA\ball_seq.dev](#)
Inspection of ball bonding (multiple images)
- [examples\hdevelop\Applications\FA\circles.dev](#)

Fits circles into curved contour segments

→ description [here in the Quick Guide](#) on page 200

- [examples\hdevelop\Applications\FA\holes.dev](#)
Extracts positions and radii of holes
- [examples\hdevelop\Applications\FA\ic.dev](#)
Extracts resistors, capacitors and ICs from board using color information
→ description [here in the Quick Guide](#) on page 215
- [examples\hdevelop\Applications\Medicine\particle.dev](#)
Extracts particles of varying sizes
→ description [here in the Quick Guide](#) on page 194
- [examples\hdevelop\Applications\OCR\ocrcolor.dev](#)
Segmenting and reading numbers using color information
→ description [here in the Quick Guide](#) on page 234
- [examples\hdevelop\Applications\OCR\ocrcolort.dev](#)
Segmenting numbers using color information and training the OCR
- [examples\hdevelop\Applications\OCV\adaption_ocv.dev](#)
Analyzes impact of changes on reported character quality
- [examples\hdevelop\Applications\OCV\print_quality.dev](#)
Inspects quality of letter A in different images
- [examples\hdevelop\Applications\Sequences\autobahn.dev](#)
Fast detection of lane markers
- [examples\hdevelop\Control\for.dev](#)
Uses a for loop to iterate over extracted blobs
- [examples\hdevelop\Filter\Geometric-Transformations\projective_trans_image_reduced.dev](#)
Applying projective transformations to an image and its domain
- [examples\hdevelop\Manuals\HDevelop\ic.dev](#)
Combining different segmentation methods
→ description in the [HDevelop User's Manual](#) on page 144
- [examples\hdevelop\Manuals\HDevelop\particle.dev](#)
Measures small particles
→ description in the [HDevelop User's Manual](#) on page 136
- [examples\hdevelop\Manuals\HDevelop\road_signs.dev](#)
Finds road markings on a motorway
→ description in the [HDevelop User's Manual](#) on page 151
- [examples\hdevelop\Regions\Creation\gen_grid_region.dev](#)
Creating a grid region consisting of lines or points
- [examples\hdevelop\Segmentation\Threshold\dual_threshold.dev](#)
Segmenting signed images into positive and negative regions

- [examples\hdevelop\Tools\2D-Transformations\vector_angle_to_rigid.dev](#)
Matching a pattern and displaying the normalized image
- [examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev](#)
Rectifies image of stadium to simulate overhead view
- [examples\hdevelop\Tools\Grid-Rectification\grid_rectification.dev](#)
Rectifying an arbitrarily distorted image using a regular grid
- [examples\hdevelop\Tools\Hough\hough_lines.dev](#)
Detecting lines in an image using the Hough transform
- [examples\hdevelop\Tools\Hough\hough_lines_dir.dev](#)
Detecting lines in an image using the Hough transform and local gradient directions
- [examples\hdevelop\Tools\Stereo\binocular_disparity_segmentation.dev](#)
Demonstrates stereo results using an artificial image pair
- [examples\hdevelop\XLD\Features\fit_ellipse_contour_xld.dev](#)
Approximating XLD contours with ellipses or elliptic arcs

C++

- [examples\cpp\source\example2.cpp](#)
Accessing image data and applying various tasks
- [examples\cpp\source\example3.cpp](#)
Iterating over a set of image pixels
- [examples\cpp\source\pen.cpp](#)
Inspects the quality of print on a pen using the variation model of HALCON

3.2.5 Selecting Operators

Segment Image(s)

Please refer to the [detailed operator list for the step Segment Image\(s\)](#) on page 50.

Draw Region

Standard:

[draw_circle](#), [draw_rectangle1](#), [draw_rectangle2](#), [draw_region](#)

Advanced:

[draw_circle_mod](#), [draw_rectangle1_mod](#), [draw_rectangle2_mod](#), [draw_xld](#), [draw_xld_mod](#)

Align ROIs Or Images

Operators for aligning ROIs or images are described in the [Application Note on Shape-Based Matching](#).

Create Region

Standard:

[gen_circle](#), [gen_ellipse](#), [gen_rectangle2](#), [gen_region_polygon_filled](#)

Advanced:

[gen_checker_region](#), [gen_grid_region](#)

More operators to generate regions can be found in the reference manual in chapter “[Regions ▷ Creation](#)”.

Create ROI

Standard:

[reduce_domain](#), [rectangle1_domain](#)

Advanced:

[change_domain](#), [full_domain](#), [add_channels](#)

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.2.6 Relation to Other Methods

One class of operators does not follow the standard rules for ROI handling: the operators of the measure tool (see the [description of this method](#) on page 53). Here, the ROI is defined during the creation of a tool ([gen_measure_arc](#) and [gen_measure_rectangle2](#)) by specifying the coordinates as numeric values. The domain defined for the image will be ignored in this case.

3.2.7 Tips & Tricks

Modifying ROI Shapes

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

Storing ROI shapes

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

Speed Up

ROIs are a perfect way to save execution time: The smaller the ROI, the faster the application. This can be used as a general rule. If we consider this in more detail, we also need to think about the shape of ROIs. Because ROIs are based on the HALCON regions they use runlength encoding (see [section 2.1.2.2](#) on page 12 for more information on HALCON regions). This type of encoding is perfect if the runs are long. Therefore, a horizontal line can be both stored and processed more efficiently than a vertical line. This holds as well for the processing time of ROIs. Obviously this type of overhead is very small and can only be of importance with very fast operators like `threshold`.

3.2.8 Advanced Topics

Filter masks and ROIs

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

Binary Images

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

3.3 Blob Analysis

The idea of blob analysis is quite easy: In an image the pixels of the relevant objects (also called foreground) can be identified by their gray value. For example, [figure 3.4](#) shows tissue particles in a liquid. These particles are bright and the liquid (background) is dark. By selecting bright pixels (thresholding) the particles can be detected easily. In many applications this simple condition of dark and bright pixels no longer holds, but the same results can be achieved with extra pre-processing or alternative methods for pixel selection / grouping.

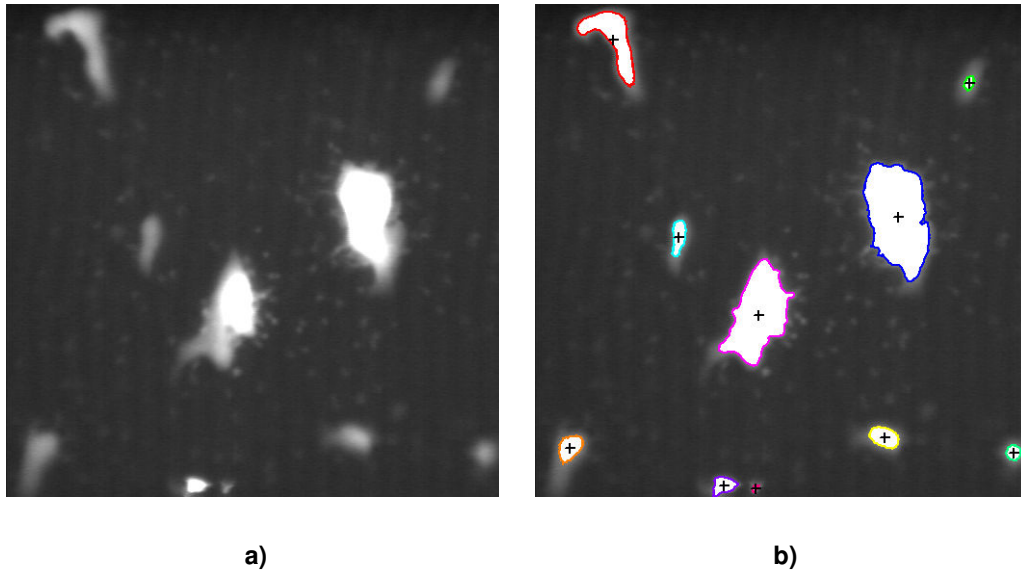
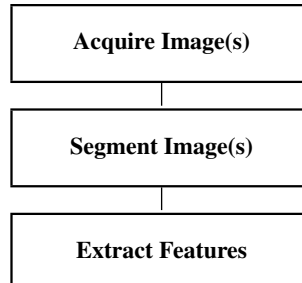


Figure 3.4: Basic idea of blob analysis: (a) original image, (b) extracted blobs with calculated center points.

The advantage of blob analysis is the extreme flexibility that comes from the huge number of operators that HALCON offers in this context. Furthermore, these methods typically have a very high performance. Methods known from blob analysis can also be combined with many other vision tasks, e.g., as a pre-processing step for a flexible generation of regions of interest.

3.3.1 Basic Concept

Blob analysis mainly consists of three parts:



Acquire Image(s)

First, an image is acquired.

For detailed information see the [description of this method](#) on page 22.

Segment Image(s)

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

Extract Features

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

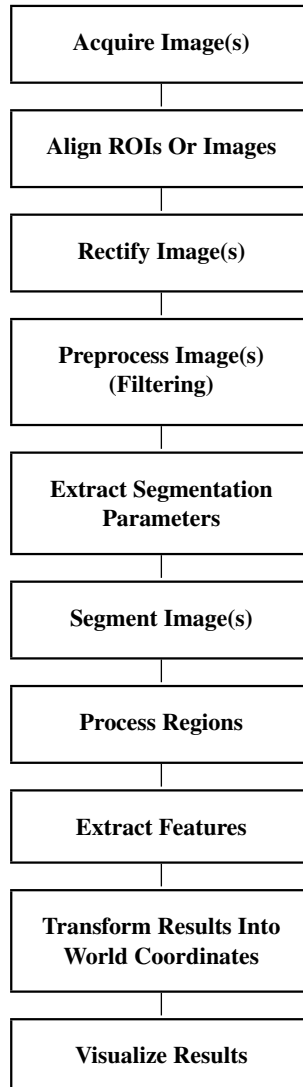
A First Example

An example for this basic concept is the following program, which belongs to the example explained above. Here, the image is acquired from file. All pixels that are brighter than 120 are selected using [threshold](#). Then, an extra step is introduced which is not so obvious: The operator [connection](#) separates the set of all bright pixels into so called connected components. The effect of this step is that we now have multiple regions instead of the single region that is returned by [threshold](#). The last step of this program is the calculation of some features. Here, the operator [area_center](#) determines the size (number of pixels) and the center of gravity. Please note that [area_center](#) returns multiple values for all three feature parameters (one value for each connected component).

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

3.3.2 Extended Concept

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



Align ROIs Or Images

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

How to perform alignment using shape-based matching is described in the Application Note on Shape-Based Matching in [section 4.3.4](#) on page 36.

Rectify Image(s)

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

Detailed information about rectifying images can be found in the Application Note on 3D Machine Vision in [section 3.3](#) on page 49.

Preprocess Image(s) (Filtering)

The next important part is the pre-processing of the image. Here, operators like [mean_image](#) or [gauss_image](#) can be used to eliminate noise. [median_image](#) is useful for suppressing small spots or thin lines. The operator [anisotrope_diff](#) is useful for edge-preserving smoothing, and finally [fill_interlace](#) is used to eliminate defects caused by interlaced cameras.

Extract Segmentation Parameters

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

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

Segment Image(s)

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

Process Regions

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

Blobs with a specific feature can be selected with [select_shape](#), [select_shape_std](#), and [select_shape_proto](#).

Extract Features

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

Transform Results Into World Coordinates

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

How to transform results into world coordinates is described in detail in the Application Note on 3D Machine Vision in [section 3.2](#) on page 44.

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.3.3 Industries

3.3.3.1 Health Care And Life Science

Medicine is the typical application area for blob analysis. Classical objects to be processed are cells or structures in tissues. For a corresponding example see the description of [particle.dev](#) on page 194.

3.3.3.2 Semiconductors

A typical task during the manufacturing of a chip is to inspect the bonding of the wires. For a corresponding example see the description of [ball.dev](#) on page 211.

3.3.3.3 Rubber, Synthetic Material, Foil

For quality assurance the outer boundary of plastic material needs to be checked. For a corresponding example see the description of [fin.dev](#) on page 209.

3.3.3.4 Food

In OCR applications like the inspection of the “best before” date, blob analysis is very useful to extract the symbols in a robust manner. For a corresponding example see the description of program [bottle.dev](#) on page 193.

3.3.3.5 Machinery

A standard task in machinery is to read text that is engraved on metal surfaces. This task can be difficult because of textures and changing illumination conditions. For a corresponding example see the description of [engraved.dev](#) on page 199.

3.3.3.6 Photogrammetry And Remote Sensing

Difficult tasks can also be solved by using methods based on blob analysis. The example program [forest.dev](#) on page 204 shows how to extract different object classes from an aerial image.

3.3.4 Programming Examples

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

3.3.4.1 Crystals

Example: [examples\quick_guide\hdevelop\crystal.dev](#)

Figure 3.5a shows an image taken in the upper atmosphere with collected samples of crystals. The task is to analyze the objects to determine the frequency of specific shapes. One of the important objects are the hexagonally shaped crystals.

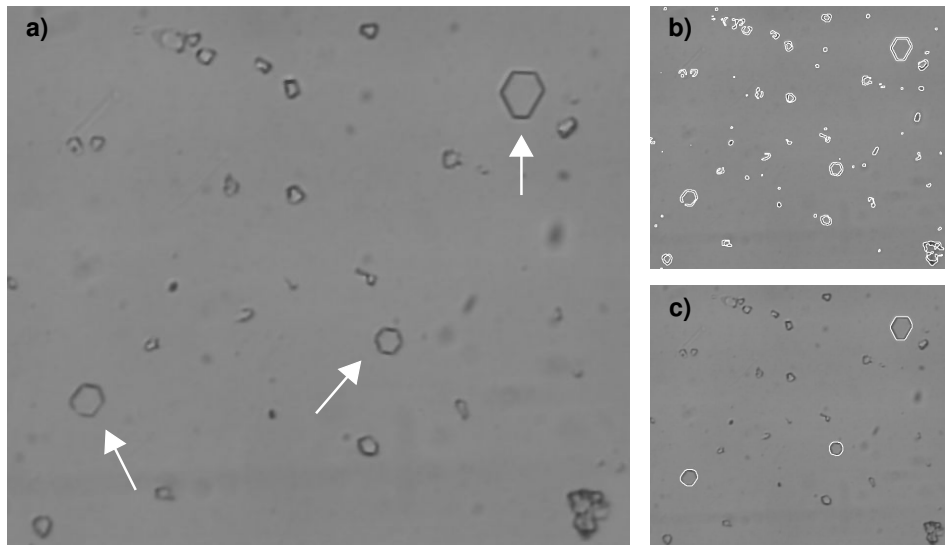


Figure 3.5: Extracting hexagonal crystals: (a) original image with arrows marking the crystals to be extracted, (b) result of the initial segmentation, (c) finally selected blobs.

First, the image is read from file with `read_image`. The segmentation of objects is performed with a local threshold because of the relatively low contrast of the crystals combined with a non-homogeneous background. The background is determined with the average filter `mean_image`. The filter mask size is selected such that it has about three times the width of the dark areas. `dyn_threshold` now compares the smoothed with the original gray values, selecting those pixels that are darker by a contrast of 8 gray values. `connection` separates the objects into connected components. [Figure 3.5b](#) shows the result of this initial segmentation.

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

In the following processing step, the task now is to select only the hexagonally shaped crystals. For this, they are first transformed into their convex hull. This is like putting a rubber band around each region. From these regions, those that are big (`select_shape`) and have a given gray value distribution (`select_gray`) are selected. The parameters for the selection are determined so that only the relevant crystals remain (see [figure 3.5c](#)).

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

3.3.4.2 Atoms

Example: [examples\quick_guide\hdevelop\atoms.dev](#)

Specialized microscopes are able to determine the rough location of single atoms. This is useful, e.g., to analyse the grid change of crystals at a p-n-junction. A segmentation that works perfectly well on images like these is the watershed method. Here, each dark basin is returned as a single region. Because at the outer part of the image atoms are only partially visible, the first task is to extract only those that are not close to the image border. Finally, the irregularity is extracted. This is done by looking for those atoms that have an abnormal (squeezed) shape (see [figure 3.6](#)).

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

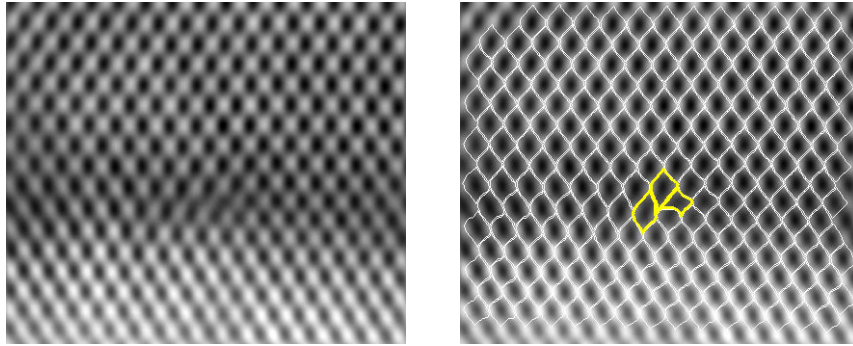


Figure 3.6: Inspecting atom structure.

3.3.4.3 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\handeye_stationarycam_grasp_nut.dev](#)
Calculates pose for grasping a nut based on results of hand-eye calibration for a stationary camera
→ description in the [Application Note on 3D Machine Vision](#) on page 120
- [examples\application_guide\image_acquisition\hdevelop\line_scan.dev](#)
Simulates grabbing from a line scan camera and merges images and extracted regions
→ description in the [Application Note on Image Acquisition](#) on page 34
- [examples\hdevelop\Applications\Aerial\dem_trees.dev](#)
Extraction of trees using texture and a digital elevation model
- [examples\hdevelop\Applications\Aerial\forest.dev](#)
Extraction of trees and meadows from forest
→ description [here in the Quick Guide](#) on page 204
- [examples\hdevelop\Applications\Aerial\texture.dev](#)
Find textured areas (trees and bushes)
- [examples\hdevelop\Applications\FA\ball.dev](#)
Inspection of ball bonding
→ description [here in the Quick Guide](#) on page 211
- [examples\hdevelop\Applications\FA\ball_seq.dev](#)
Inspection of ball bonding (multiple images)
- [examples\hdevelop\Applications\FA\board.dev](#)
Detection of missing solder
- [examples\hdevelop\Applications\FA\clip.dev](#)
Determines the orientation of clips

- [examples\hdevelop\Applications\FA\holes.dev](#)
Extracts positions and radii of holes
- [examples\hdevelop\Applications\FA\hull.dev](#)
Inspects an injection molded nozzle
- [examples\hdevelop\Applications\FA\ic.dev](#)
Extracts resistors, capacitors and ICs from board using color information
→ description [here in the Quick Guide](#) on page 215
- [examples\hdevelop\Applications\Medicine\particle.dev](#)
Extracts particles of varying sizes
→ description [here in the Quick Guide](#) on page 194
- [examples\hdevelop\Applications\Monitoring\movement_col.dev](#)
Extracts moving objects and within them scans for specific color
- [examples\hdevelop\Applications\OCR\stamp_catalogue.dev](#)
Segmenting and grouping characters on a cluttered page
- [examples\hdevelop\Applications\OCV\print_quality.dev](#)
Inspects quality of letter A in different images
- [examples\hdevelop\Control\for.dev](#)
Uses a for loop to iterate over extracted blobs
- [examples\hdevelop\Control\while.dev](#)
Uses a while loop to provide interaction until right mouse button is clicked
- [examples\hdevelop\Filter\Texture\entropy_image.dev](#)
Segmenting an image based on the entropy of gray values
- [examples\hdevelop\Image\Features\gray_features.dev](#)
Calculating standard gray value features
- [examples\hdevelop\Manuals\HDevelop\ball.dev](#)
Inspects bonding of balls
→ description in the [HDevelop User's Manual](#) on page 141
- [examples\hdevelop\Manuals\HDevelop\calib.dev](#)
Extracts circles on a calibration board
→ description in the [HDevelop User's Manual](#) on page 143
- [examples\hdevelop\Manuals\HDevelop\eyes.dev](#)
Performing a basic segmentation on a single image
→ description in the [HDevelop User's Manual](#) on page 149
- [examples\hdevelop\Manuals\HDevelop\ic.dev](#)
Combining different segmentation methods
→ description in the [HDevelop User's Manual](#) on page 144
- [examples\hdevelop\Manuals\HDevelop\marks.dev](#)
Detects circular marks fixed at a person's body
→ description in the [HDevelop User's Manual](#) on page 3

- [examples\hdevelop\Manuals\HDevelop\particle.dev](#)
Measures small particles
→ description in the [HDevelop User's Manual](#) on page 136
- [examples\hdevelop\Manuals\HDevelop\stamps.dev](#)
Finds images in a document
→ description in the [HDevelop User's Manual](#) on page 131
- [examples\hdevelop\Manuals\HDevelop\wood.dev](#)
Determines the age of a tree by counting its annual rings
→ description in the [HDevelop User's Manual](#) on page 139
- [examples\hdevelop\Regions\Features\orientation_region.dev](#)
Calculating the orientation of regions
- [examples\hdevelop\Regions\Transformations\connection.dev](#)
Calculating connected components of regions
- [examples\hdevelop\Regions\Transformations\fill_up_shape.dev](#)
Filling up holes with give shape in regions
- [examples\hdevelop\Regions\Transformations\max_connection.dev](#)
Limiting the number of regions returned by connection operator
- [examples\hdevelop\Segmentation\Threshold\threshold.dev](#)
Selecting gray values lying within an interval
- [examples\quick_guide\hdevelop\color_fuses.dev](#)
Sort fuses by color
→ description [here in the Quick Guide](#) on page 116
- [examples\quick_guide\hdevelop\color_pieces.dev](#)
Completeness check of colored game pieces using MLP classification
→ description [here in the Quick Guide](#) on page 118
- [examples\quick_guide\hdevelop\color_pieces_euclid.dev](#)
Completeness check of game color pieces using Euclidean classification
- [examples\quick_guide\hdevelop\color_simple.dev](#)
Segment color image in HSV color space
→ description [here in the Quick Guide](#) on page 115
- [examples\quick_guide\hdevelop\edge_segments.dev](#)
Extracting connected edges segments
→ description [here in the Quick Guide](#) on page 68
- [examples\quick_guide\hdevelop\surface_scratch.dev](#)
Detects scratches on a surface via local thresholding and morphology
→ description [here in the Quick Guide](#) on page 241

C++

- [examples\cpp\source\example1.cpp](#)
Reading an image and performing a simple segmentation

Visual Basic

- `examples\vb\Manual\eyes.vbp`
Locates the eyes of a monkey
- `examples\vb\Segmentation\monkey.vbp`
Blob analysis with interactive control of some parameters

C

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

3.3.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Align ROIs Or Images

Operators for rectifying images are described in the [Application Note on Shape-Based Matching](#).

Rectify Image(s)

Operators for rectifying images are described in the [Application Note on 3D Machine Vision](#).

Preprocess Image(s) (Filtering)

Standard:

[mean_image](#), [gauss_image](#), [median_image](#)

Advanced:

[smooth_image](#), [anisotrope_diff](#), [fill_interlace](#), [rank_image](#)

Extract Segmentation Parameters

Standard:

[gray_histo_abs](#), [histo_to_thresh](#)

Advanced:

[intensity](#)

Segment Image(s)

Standard:

`threshold`, `fast_threshold`, `bin_threshold`, `dyn_threshold`, `histo_to_thresh`,
`gray_histo`

Advanced:

`watersheds`, `watersheds_threshold`, `regiongrowing`, `regiongrowing_mean`, `var_threshold`

Process Regions

Standard:

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

Advanced:

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

Morphological operators can be found in the Reference Manual in the chapter “[Morphology](#)”.

Extract Features

Standard:

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

Advanced:

`diameter_region`, `inner_circle`, `gray_histo_abs`, `entropy_gray`

Transform Results Into World Coordinates

Standard:

`image_points_to_world_plane`

Advanced:

`gen_contour_region_xld`, `contour_to_world_plane_xld`

More operators for transforming results into world coordinates are described in the [Application Note on 3D Machine Vision](#).

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.3.6 Relation to Other Methods

3.3.6.1 Methods that are Using Blob Analysis

OCR (see [description](#) on page 145)

Blob analysis is typically used as a preprocessing step for OCR to segment the characters.

3.3.6.2 Alternatives to Blob Analysis

Edge Extraction (subpixel-precise) (see [description](#) on page 74)

In blob analysis a region is described by the gray values of its pixels. As an alternative, a region could be described by the change of the gray values at their borders. This approach is called edge detection.

3.3.7 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for blob analysis. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the segmentation is performed can be restricted, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

Connected Components

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

Speed Up

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

- Regions of interest are the standard method to increase the speed by processing only those areas where objects need to be inspected. This can be done using pre-defined regions but also by an online generation of the regions of interest that depend on other objects found in the image.
- If an object has a specific minimum size, the operator [fast_threshold](#) is a fast alternative to [threshold](#). This kind of fast operator can also directly be generated by using operators like [gen_grid_region](#) and [reduce_domain](#) before calling the thresholding operator.

- By default, HALCON performs some data consistency checks. These can be switched off using [set_check](#).
- By default, HALCON initializes new images. Using [set_system](#) with the parameter "init_new_image", this behavior can be changed.

3.3.8 Advanced Topics

Line Scan Cameras

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

High Accuracy

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

3.4 1D Measuring

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

Based on the extracted edges, you can measure the dimensions of parts. For example, you can measure the width of a part by placing a region of interest over it and locating the edges on its left and the right side. The effect of this can be seen in [figure 3.7a](#), whereas [figure 3.7b](#) shows the corresponding gray value profile.

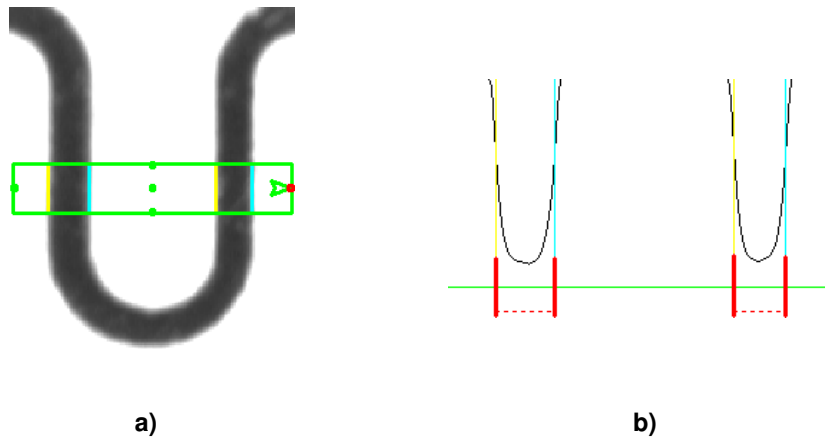


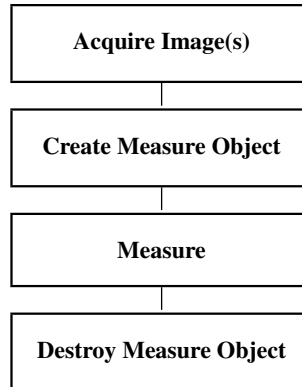
Figure 3.7: (a) Measuring a fuse wire; (b) gray value profile along the region of measurement with extracted edges.

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

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

3.4.1 Basic Concept

Measuring consists of four main steps:



Acquire Image(s)

First, an image is acquired.

For detailed information see the [description of this method](#) on page 22.

Create Measure Object

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

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

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

Measure

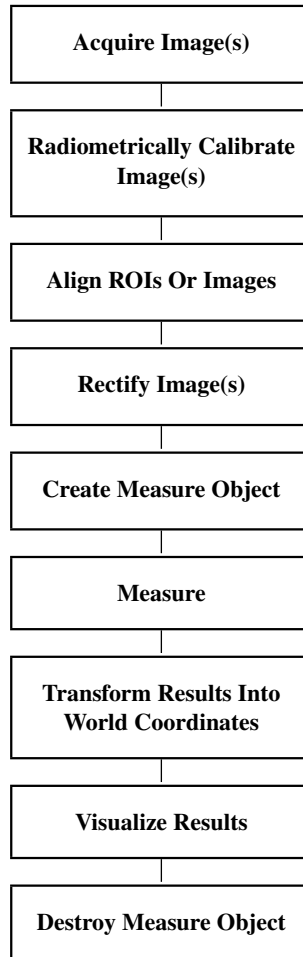
Then, you can apply the measuring by specifying the measure object and some other vision parameters like, e.g., the minimum contrast. You can find detailed information about this step in the Application Note on 1D Metrology in [section 3](#) on page 10.

Destroy Measure Object

When you no longer need the measure object, you destroy it by passing the handle to [close_measure](#).

3.4.2 Extended Concept

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



Radiometrically Calibrate Image(s)

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

Align ROIs Or Images

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

How to perform alignment using shape-based matching is described in the Application Note on Shape-Based Matching in [section 4.3.4](#) on page 36.

Rectify Image(s)

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

Detailed information about rectifying images can be found in the Application Note on 3D Machine Vision in [section 3.3](#) on page 49.

Create Measure Object

You can teach the measurement line or arc interactively with operators like [draw_rectangle2](#) or read its parameters from file ([read_string](#)). As an alternative, its coordinates can be generated based on the results of other vision tools like Blob Analysis (see the [description of this method](#) on page 39). In particular, the measurement line or arc may need to be aligned to a certain object as described above.

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

Please refer to the Application Note on 1D Metrology, [section 2](#) on page 5, for more information.

Transform Results Into World Coordinates

If you have calibrated your vision system, you can easily transform the results of measuring into world coordinates with [image_points_to_world_plane](#). How to do this is described in the Application Note on 1D Metrology in [section 3.5](#) on page 20.

This is described in detail in the Application Note on 3D Machine Vision in [section 3.2](#) on page 44.

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.4.3 Industries

3.4.3.1 Semiconductors

The Quick Guide contains two examples for applying measurements in the semiconductor industry: [measure_pin.dev](#) on page 226 shows how to inspect major dimensions of an IC. [fuzzy_measure_pin.dev](#) on page 212 performs a similar task under difficult illumination conditions.

3.4.3.2 Iron, Steel And Metal

The example [measure_arc.dev](#) on page 197 shows how to inspect the exact distance between elongated holes of a cast part after chamfering.

3.4.4 Programming Examples

The following example gives a brief introduction to using the measure tool of HALCON. The longest parts are pre- and postprocessing; the measurement itself consists only of two operator calls. Further examples are described in the [Application Note on 1D Metrology](#).

3.4.4.1 Inspecting a Fuse

Example: [examples\quick_guide\hdevelop\fuse.dev](#)

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

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

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

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

The last part of the program displays the results by generating a region with the parameters of the measurement line and converting the edge positions to short XLD contours (see [figure 3.8](#)).

```

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

```

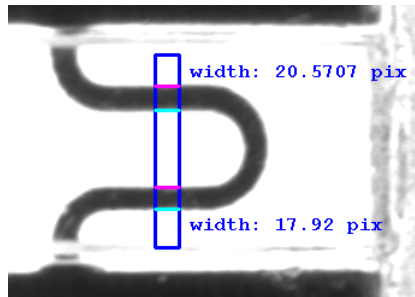


Figure 3.8: Measuring the width of the fuse wire.

3.4.4.2 Other Examples

HDevelop

- [examples\application_guide\1d_metrology\hdevelop\fuzzy_measure_switch.dev](#)
Determine the width of and the distance between the pins of a switch with a fuzzy measure object
→ description in the [Application Note on 1D Metrology](#) on page 25
- [examples\application_guide\1d_metrology\hdevelop\measure_caliper.dev](#)
Measures the distance between the pitch lines of a caliper
→ description in the [Application Note on 1D Metrology](#) on page 16
- [examples\application_guide\1d_metrology\hdevelop\measure_ic_leads.dev](#)
Measures leads of an IC
→ description in the [Application Note on 1D Metrology](#) on page 11
- [examples\application_guide\1d_metrology\hdevelop\measure_ring.dev](#)
Determine the width of cogs with a circular measure object
→ description in the [Application Note on 1D Metrology](#) on page 22
- [examples\application_guide\1d_metrology\hdevelop\measure_switch.dev](#)

Determine the width of and the distance between the pins of a switch with a measure object
→ description in the [Application Note on 1D Metrology](#) on page 4

- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_exterior.dev](#)
Measures positions on a caliper rule using camera calibration
→ description in the [Application Note on 3D Machine Vision](#) on page 41
- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_multi_image.dev](#)
Calibrates the camera and measures positions on a caliper rule
→ description in the [Application Note on 3D Machine Vision](#) on page 39
- [examples\application_guide\shape_matching\hdevelop\align_measurements.dev](#)
Inspects individual razor blades using shape-based matching to align ROIs for the measure tool
→ description in the [Application Note on Shape-Based Matching](#) on page 36
- [examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev](#)
Measures distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- [examples\hdevelop\Applications\FA\pm_measure_board.dev](#)
Locates IC on a board and measures pin distances
→ description [here in the Quick Guide](#) on page 213
- [examples\hdevelop\Applications\Measure\fuzzy_measure_pin.dev](#)
Measures pins of an IC using fuzzy measure
→ description [here in the Quick Guide](#) on page 212
- [examples\hdevelop\Applications\Measure\measure_arc.dev](#)
Measures width of object along circular arc
→ description [here in the Quick Guide](#) on page 197
- [examples\hdevelop\Applications\Measure\measure_online.dev](#)
Measures your object in a live image
- [examples\hdevelop\Applications\Measure\measure_pin.dev](#)
Measures pins of an IC
→ description [here in the Quick Guide](#) on page 226
- [examples\hdevelop\Tools\Measure\gen_measure_arc.dev](#)
Measuring edges perpendicular to a given arc
- [examples\hdevelop\Tools\Measure\gen_measure_rectangle2.dev](#)
Measuring edges perpendicular to a given line
- [examples\quick_guide\hdevelop\close_contour_gaps.dev](#)
Closing gaps in extracted straight contours
→ description [here in the Quick Guide](#) on page 92

C++

- [examples\cpp\source\fuzzy_measure_pin.cpp](#)
Measures pins of an IC using fuzzy measure

- `examples\mfc\Matching\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, creating a HALCON window
- `examples\mfc\MatchingCOM\Matching.cpp`
Locating an IC using HALCON/COM and MFC
- `examples\mfc\MatchingExtWin\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, painting into an existing window
- `examples\motif\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Motif
- `examples\qt\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Qt

Visual Basic

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

Visual Basic .NET

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

C#

- `examples\c#\Matching\Matching.csproj`
Locates an IC on a board and measures pin distances

Delphi

- `examples\delphi\Matching\matching.dpr`
Locates an IC on a board and measures pin distances

3.4.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Radiometrically Calibrate Image(s)

Standard:

[radiometric_self_calibration](#), [lut_trans](#)

Align ROIs Or Images

Operators for aligning ROIs or images are described in the [Application Note on Shape-Based Matching](#).

Rectify Image(s)

Operators for rectifying images are described in the [Application Note on 3D Machine Vision](#).

Create Measure Object

Standard:

```
gen_measure_rectangle2, gen_measure_arc, translate_measure
```

Measure

Standard:

```
measure_pos, measure_pairs
```

Advanced:

```
set_fuzzy_measure, fuzzy_measure_pos, fuzzy_measure_pairs, fuzzy_measure_pairing
```

Transform Results Into World Coordinates

Standard:

```
image_points_to_world_plane
```

Advanced:

```
gen_contour_region_xld, contour_to_world_plane_xld
```

More operators for transforming results into world coordinates are described in the [Application Note on 3D Machine Vision](#).

Visualize Results

Advanced:

```
gen_contour_polygon_xld
```

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

Destroy Measure Object

Standard:

[close_measure](#)

3.4.6 Relation to Other Methods

3.4.6.1 Alternatives to 1D Measuring

Edge Extraction (subpixel-precise) (see [description](#) on page 74)

A very flexible way to measure parameters of edges is to extract the edge contour with [edges_sub_pix](#). The advantage of this approach is that it can handle free-form shapes. Furthermore, it allows to determine attributes like the edge direction for each edge point.

3.4.7 Tips & Tricks

Suppress Clutter or Noise

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

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

Reuse of Measure Objects

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

Use an Absolute Gray Value Threshold

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

3.4.8 Advanced Topics

Fuzzy Measuring

In case there are extra edges that do not belong to the measurement, HALCON offers an extended version of measuring: fuzzy measuring. This tool allows to define so-called fuzzy rules, which describe

the features of good edges. Possible features are, e.g., the position, the distance, the gray values, or the amplitude of edges. These functions are created with [create_funct_1d_pairs](#) and passed to the tool with [set_fuzzy_measure](#). Based on these rules, the tool will select the most appropriate edges.

The advantage of this approach is the flexibility to deal with extra edges even if a very low minimum threshold or smoothing is used. An example for this approach is the example program [fuzzy_measure_pin.dev](#) on page 212.

Please refer to the Application Note on 1D Metrology, [section 4](#) on page 25, for more information.

Evaluation of Gray Values

To have full control over the evaluation of the gray values along the measurement line or arc, you can use [measure_projection](#). The operator returns the projected gray values as an array of numbers, which can then be further processed with HALCON operators for tuple or function processing (see the chapters “[Tuple](#)” and “[Tools > Function](#)” in the Reference Manual). Please refer to the Application Note on 1D Metrology, [section 3.4](#) on page 16, for more information.

3.5 Edge Extraction (pixel-precise)

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

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

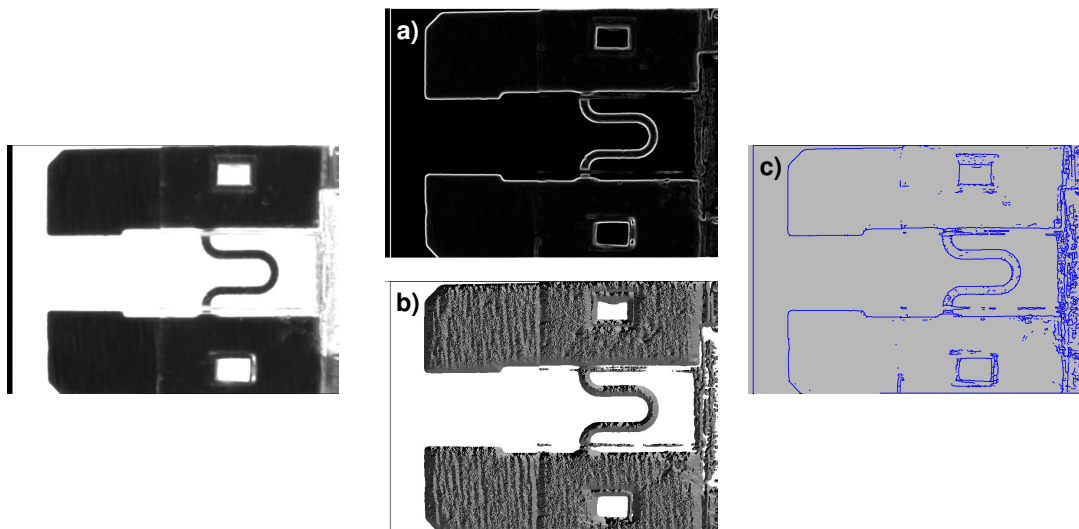
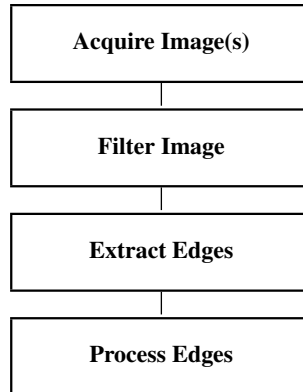


Figure 3.9: Result of applying an edge filter: (a) amplitude, (b) direction, (c) extracted edges.

Please note that in addition to this classical approach, HALCON provides advanced operators for subpixel-precise edge and line extraction (see the [description of this method](#) on page 74) and for successive post-processing and feature extraction.

3.5.1 Basic Concept

Using edge filters typically consists of three basic steps:



Acquire Image(s)

First, an image is acquired.

For detailed information see the [description of this method](#) on page 22.

Filter Image

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

Extract Edges

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

Process Edges

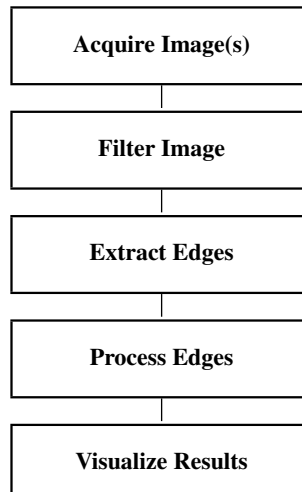
Having extracted the edges, two further processing steps can be applied: The first step is to convert the edge regions into another data structure for a potential further processing and for extracting features. The second step is to extract the regions enclosed by the edges.

A First Example

The following program shows an example for the basic concept of edge filters. As an edge filter, `sobel_amp` is applied with the mode `'thin_sum_abs'` to get thin edges together with a 3x3 filter mask. Then, the operator `threshold` is used to extract all pixels with an edge amplitude higher than 20. The resulting region contains some areas where the edge is wider than one pixel. Therefore, the operator `skeleton` is applied to thin all edges completely. The result is depicted in [figure 3.9c](#) on page 64.

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

3.5.2 Extended Concept



Filter Image

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

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

Extract Edges

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

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

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

Process Edges

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

You can extract the regions enclosed by the edges easily using `background_seg`. If regions merge because of gaps in the edges, the operators `close_edges` or `close_edges_length` can be used in advance to close the gaps before regions are extracted. As an alternative, morphological operators like `opening_circle` can be applied to the output regions of `background_seg`. In general, all operators described for the method [Process Regions](#) on page 42 can be applied here as well.

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.5.3 Industries

3.5.3.1 Photogrammetry And Remote Sensing

Sometimes, objects can no longer be separated because they have similar gray values. When using color information, it is possible to extract the object borders quite easily. The example program `edges_color.dev` on page 205 shows how to separate two areas that have the same luminance but different hues.

3.5.4 Programming Examples

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

3.5.4.1 Aerial Image Interpretation

Example: `examples\quick_guide\hdevelop\edge_segments.dev`

Figure 3.10 shows an image taken from an aeroplane. The task is to extract the edges of roads and buildings as a basis for the image interpretation.

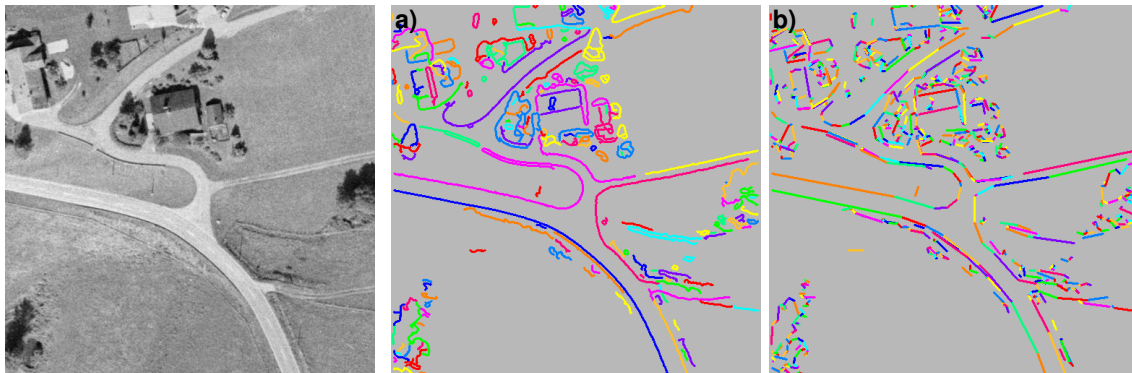


Figure 3.10: (a) Extracting edges and (b) approximating them by segments.

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

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

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


```

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

```

3.5.4.2 Other Examples

HDevelop

- [examples\hdevelop\Applications\Aerial\roads.dev](#)
Extraction of roads from aerial image
→ description [here in the Quick Guide](#) on page 207
- [examples\hdevelop\Applications\FA\sharpness.dev](#)
Determines the sharpness of an image using different approaches
- [examples\hdevelop\Applications\Sequences\autobahn.dev](#)
Fast detection of lane markers
- [examples\hdevelop\Filter\Edges\close_edges.dev](#)
Closing edge gaps using the edge amplitude image
- [examples\hdevelop\Filter\Edges\close_edgeslength.dev](#)
Closing edge gaps using the edge amplitude image
- [examples\hdevelop\Filter\Edges\derivate_gauss.dev](#)
Various usages for convolving an image with derivatives of the Gaussian
- [examples\hdevelop\Filter\Edges\diff_of_gauss.dev](#)
Approximating the LoG operator (Laplace of Gaussian)
- [examples\hdevelop\Filter\Edges\edges_color.dev](#)
Extracting edges using color information
→ description [here in the Quick Guide](#) on page 205
- [examples\hdevelop\Filter\Edges\edges_image.dev](#)
Extracting edges (amplitude and direction) using Deriche, Lanser, Shen, or Canny filters
- [examples\hdevelop\Filter\Edges\frei_amp.dev](#)
Extracting edges (amplitude) using the Frei-Chen operator
- [examples\hdevelop\Filter\Edges\frei_dir.dev](#)
Extracting edges (amplitude and direction) using the Frei-Chen operator

- [examples\hdevelop\Filter\Edges\highpass_image.dev](#)
Extracting high frequency components from an image
- [examples\hdevelop\Filter\Edges\info_edges.dev](#)
Estimate the width of a filter in edges_image
- [examples\hdevelop\Filter\Edges\kirsch_amp.dev](#)
Extracting edges (amplitude) using the Kirsch operator
- [examples\hdevelop\Filter\Edges\kirsch_dir.dev](#)
Extracting edges (amplitude and direction) using the Kirsch operator
- [examples\hdevelop\Filter\Edges\laplace.dev](#)
Extracting edges using the Laplace Operator
- [examples\hdevelop\Filter\Edges\laplace_of_gauss.dev](#)
Extracting edges using the LoG operator (Laplace of Gaussian)
- [examples\hdevelop\Filter\Edges\prewitt_amp.dev](#)
Extracting edges (amplitude) using the Prewitt operator
- [examples\hdevelop\Filter\Edges\prewitt_dir.dev](#)
Extracting edges (amplitude and direction) using the Prewitt operator
- [examples\hdevelop\Filter\Edges\roberts.dev](#)
Extracting edges using the Roberts filter
- [examples\hdevelop\Filter\Edges\robinson_amp.dev](#)
Extracting edges (amplitude) using the Robinson operator
- [examples\hdevelop\Filter\Edges\robinson_dir.dev](#)
Extracting edges (amplitude and direction) using the Robinson operator
- [examples\hdevelop\Filter\Edges\sobel_amp.dev](#)
Extracting edges (amplitude) using the Sobel operator
- [examples\hdevelop\Filter\Edges\sobel_dir.dev](#)
Extracting edges (amplitude and direction) using the Sobel operator
- [examples\hdevelop\Filter\Lines\bandpass_image.dev](#)
Extracting lines using bandpass filter
- [examples\hdevelop\Graphics\Output\disp_xld.dev](#)
Displaying an XLD object
- [examples\hdevelop\Manuals\HDevelop\road_signs.dev](#)
Finds road markings on a motorway
→ description in the [HDevelop User's Manual](#) on page 151
- [examples\hdevelop\Regions\Transformations\background_seg.dev](#)
Calculating connected background components for given foreground regions
- [examples\hdevelop\Segmentation\Edges\hysteresis_threshold.dev](#)
Performing a hysteresis threshold operation on an edge image

- [examples\hdevelop\Segmentation\Edges\nonmax_suppression_amp.dev](#)
Suppressing non-maximum points
- [examples\hdevelop\Segmentation\Edges\nonmax_suppression_dir.dev](#)
Suppress non-maximum points on edges
- [examples\hdevelop\Tools\Hough\hough_lines.dev](#)
Detecting lines in an image using the Hough transform
- [examples\hdevelop\Tools\Hough\hough_lines_dir.dev](#)
Detecting lines in an image using the Hough transform and local gradient directions
- [examples\quick_guide\hdevelop\surface_scratch.dev](#)
Detects scratches on a surface via local thresholding and morphology
→ description [here in the Quick Guide](#) on page 241

C++

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

Visual Basic

- [examples\vb\Online\Movement\movement.vbp](#)
Detecting moving objects

C

- [examples\c\example2.c](#)
Performing some basic segmentation
- [examples\c\example4.c](#)
Applying an LUT for visualization of relevant details

3.5.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Filter Image

Standard:

[sobel_amp](#), [sobel_dir](#), [edges_image](#)

Advanced:

[derivate_gauss](#), [edges_color](#)

Extract Edges

Standard:

[threshold](#), [skeleton](#), [inspect_shape_model](#)

Advanced:

[hysteresis_threshold](#), [nonmax_suppression_dir](#)

Process Edges

Standard:

[background_seg](#), [close_edges](#), [close_edges_length](#), [opening_circle](#),
[split_skeleton_lines](#), [hough_lines_dir](#), [hough_lines](#)

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.5.6 Relation to Other Methods

3.5.6.1 Alternatives to Edge Extraction (pixel-precise)

Blob Analysis (see [description](#) on page 39)

As an alternative to edge extraction, blob analysis can be used. This approach provides many methods from simple thresholding to region growing and watershed methods.

3.5.7 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for edge extraction. With domains, the processing can be focused to a certain area in the image and can thus be sped up. The more the region in which the edge filtering is performed can be restricted, the faster and more robust the extraction will be. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

Speed Up

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

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

3.5.8 Advanced Topics

Subpixel Edge and Line Extraction

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

Contour Processing

As a post-processing step, you can convert the edge region into XLD contours by using, e.g., the operator `gen_contours_skeleton_xld`. The advantage of this approach is the extended set of operators offered for [contour processing](#) on page 84.

3.6 Edge Extraction (subpixel-precise)

In addition to the traditional way of applying an edge filter to get the edge amplitude and thus the edges (see the method [Edge Extraction \(pixel-precise\)](#) on page 64), HALCON provides one-step operators that return subpixel-precise XLD contours. Besides this, not only edges but also lines can be extracted. This approach can also be applied to color images.

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

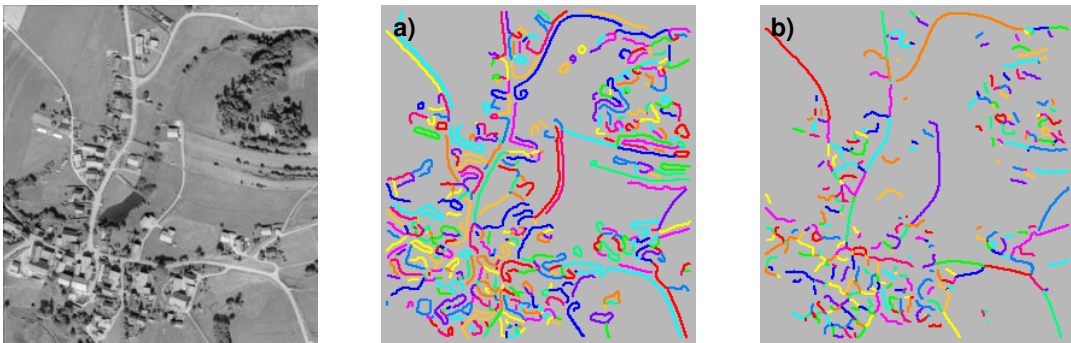
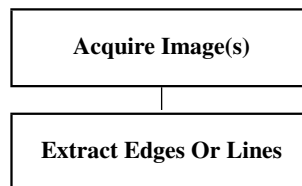


Figure 3.11: Result of contour extraction: (a) edge contours, (b) line contours.

This chapter covers only the extraction of contours. For information about processing them see the method [Contour Processing](#) on page 84.

3.6.1 Basic Concept

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



Acquire Image(s)

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

For detailed information see the [description of this method](#) on page 22.

Extract Edges Or Lines

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

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

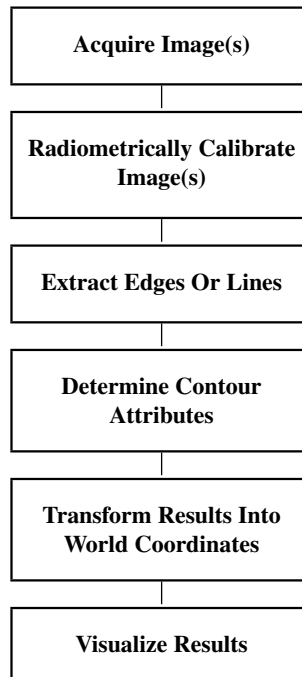
A First Example

The following program explains the basic concept of edge extraction. The only operator needed to extract edge contours is `edges_sub_pix`. It has the image as input and returns the XLD contours. Here, the filter 'lanser2' is selected with a medium-sized smoothing mask. The low value for the parameter Low ensures that contours are tracked even along low-contrast parts. To show that the result consists of multiple contours, the 12-color mode for visualization is selected. The result is depicted in [figure 3.11b](#) on page 74.

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

3.6.2 Extended Concept

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



Radiometrically Calibrate Image(s)

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

Extract Edges Or Lines

The most often used operator for edge contour extraction is [edges_sub_pix](#). You can select various filter methods by specifying the corresponding name with the parameter `Filter`. For standard applications, we recommend to use the values 'canny' (based on a Gaussian convolution) or 'lanser2'. The advantage of 'lanser2' is the recursive implementation which has no increase in execution time when using a large smoothing. As a fast version the parameter value 'sobel' can be used.

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

To extract edges in multi-channel images, e.g., in a color image, HALCON provides the operator [edges_color_sub_pix](#).

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

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

Determine Contour Attributes

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

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

Transform Results Into World Coordinates

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

This is described in detail in the Application Note on 3D Machine Vision in [section 3.2](#) on page 44.

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.6.3 Industries

3.6.3.1 Health Care And Life Science

A standard task in angiography is the inspection of the coronary vessels. In addition to locating the vessels, an important step is to determinate their diameter. This task is solved by the example program `lines_gauss.dev` on page 196.

3.6.3.2 Photogrammetry And Remote Sensing

A very complex task is the extraction of roads from an aerial image. The example program `roads.dev` on page 207 shows how lines and edges can be used in combination to get road segments with a high probability.

3.6.4 Programming Examples

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

3.6.4.1 Measuring the diameter of drilled holes

Example: `examples\quick_guide\hdevelop\rim_simple.dev`

Figure 3.12 shows an image of a car rim. The task is to measure the diameters of the drilled holes.

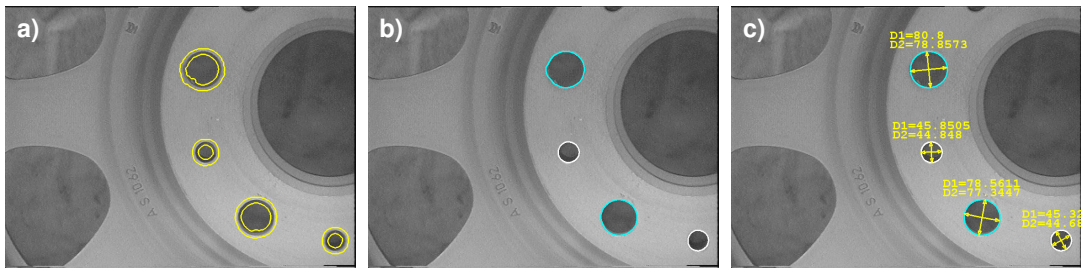


Figure 3.12: (a) automatically determined ROIs; (b) extracted edges; (c) computed ellipses and diameters.

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

Locating the holes is quite easy: First, all dark pixels are selected. After selecting all those connected components that are circular and have a certain size, only the holes remain. Finally, the regions of interest are obtained by accessing the borders of the holes and dilating them. The resulting ROIs are depicted in figure 3.12a.

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

Calling `reduce_domain` changes the domain of the image to the prepared region of interest. Now, the edge extractor can be applied (see figure 3.12b).

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

The extracted contours are further processed to determine their diameter: With `fit_ellipse_contour_xld`, ellipses are fitted to the contours. In other words, those ellipses are determined that fit the extracted contours as closely as possible. The operator returns the parameters of the ellipses. With the operator `gen_ellipse_contour_xld`, the corresponding ellipses are created and displayed (compare [figure 3.12b](#) and [figure 3.12c](#)).

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

The diameters can easily be computed from the ellipse parameters and then be displayed in the image using `write_string` (see [figure 3.12c](#)).

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

3.6.4.2 Other Examples

HDevelop

- [examples\application_guide\1d_metrology\hdevelop\fuzzy_measure_switch.dev](#)
Determine the width of and the distance between the pins of a switch with a fuzzy measure object
→ description in the [Application Note on 1D Metrology](#) on page 25
- [examples\application_guide\1d_metrology\hdevelop\measure_caliper.dev](#)
Measures the distance between the pitch lines of a caliper
→ description in the [Application Note on 1D Metrology](#) on page 16
- [examples\application_guide\1d_metrology\hdevelop\measure_ic_leads.dev](#)
Measures leads of an IC
→ description in the [Application Note on 1D Metrology](#) on page 11
- [examples\application_guide\1d_metrology\hdevelop\measure_ring.dev](#)
Determine the width of cogs with a circular measure object
→ description in the [Application Note on 1D Metrology](#) on page 22
- [examples\application_guide\1d_metrology\hdevelop\measure_switch.dev](#)
Determine the width of and the distance between the pins of a switch with a measure object
→ description in the [Application Note on 1D Metrology](#) on page 4
- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_exterior.dev](#)
Measures positions on a caliper rule using camera calibration
→ description in the [Application Note on 3D Machine Vision](#) on page 41

- [examples\application_guide\3d_machine_vision\hdevelop\radial_distortion.dev](#)
Eliminating radial distortions
→ description in the [Application Note on 3D Machine Vision](#) on page 48
- [examples\hdevelop\Applications\Aerial\roads.dev](#)
Extraction of roads from aerial image
→ description [here in the Quick Guide](#) on page 207
- [examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev](#)
Determine the pose of circles in 3D from their perspective 2D projections
- [examples\hdevelop\Applications\Calibration\lens_distortions.dev](#)
Eliminates distortions of edges extracted from a distorted image
- [examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev](#)
Measures distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- [examples\hdevelop\Applications\FA\circles.dev](#)
Fits circles into curved contour segments
→ description [here in the Quick Guide](#) on page 200
- [examples\hdevelop\Applications\Medicine\angio.dev](#)
Extracts blood vessels and their diameters from an angiogram
- [examples\hdevelop\Filter\Edges\edges_color_sub_pix.dev](#)
Extracting edges with sub-pixel precision using color information
- [examples\hdevelop\Filter\Edges\edges_sub_pix.dev](#)
Extracting edges with sub-pixel precision
- [examples\hdevelop\Filter\Lines\lines_color.dev](#)
Extracting lines using color information
→ description [here in the Quick Guide](#) on page 191
- [examples\hdevelop\Filter\Lines\lines_facet.dev](#)
Extracting lines using the facet model
- [examples\hdevelop\Filter\Lines\lines_gauss.dev](#)
Extracting lines and their widths
→ description [here in the Quick Guide](#) on page 196
- [examples\hdevelop\Graphics\Output\disp_xld.dev](#)
Displaying an XLD object
- [examples\hdevelop\Graphics\Output\dump_window_data.dev](#)
Dump the content of the graphics window to an image object
- [examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev](#)
Rectifies image of stadium to simulate overhead view
- [examples\hdevelop\Tools\Calibration\change_radial_distortion_contours_xld.dev](#)
Eliminating radial distortions from extracted contours

- [examples\hdevelop\XLD\Features\fit_ellipse_contour_xld.dev](#)
Approximating XLD contours with ellipses or elliptic arcs
- [examples\hdevelop\XLD\Features\fit_ellipse_tooth_rim_xld.dev](#)
Approximating the contour of a tooth rim with an ellipse to find its center.
- [examples\hdevelop\XLD\Transformation\clip_contours_xld.dev](#)
Clipping an XLD contour
- [examples\hdevelop\XLD\Transformation\crop_contours_xld.dev](#)
Cropping an XLD contour
- [examples\hdevelop\XLD\Transformation\gen_parallel_contour_xld.dev](#)
Computing the parallel contour of an XLD contour
- [examples\hdevelop\XLD\Transformation\sort_contours_xld.dev](#)
Sorting XLD contours by spatial position
- [examples\hdevelop\XLD\Transformation\union_cocircular_contours_xld.dev](#)
Merging contours belonging to the same circle
- [examples\hdevelop\XLD\Transformation\union_contours_xld.dev](#)
Connecting collinear line segments
- [examples\quick_guide\hdevelop\close_contour_gaps.dev](#)
Closing gaps in extracted straight contours
→ description [here in the Quick Guide](#) on page 92
- [examples\quick_guide\hdevelop\measure_metal_part.dev](#)
Inspects metal part by fitting lines and circles
→ description [here in the Quick Guide](#) on page 90

C++

- [examples\cpp\source\example9.cpp](#)
Extracts roads from aerial images
- [examples\mfc\FGMultiThreading\FGMultiThreading.cpp](#)
Using multiple threads to grab and process images in parallel

3.6.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Radiometrically Calibrate Image(s)

Standard:

[radiometric_self_calibration](#), [lut_trans](#)

Extract Edges Or Lines

Standard:

`edges_sub_pix`, `derivate_gauss`, `lines_gauss`, `lines_facet`

Advanced:

`zero_crossing_sub_pix`, `edges_color_sub_pix`, `lines_color`

Determine Contour Attributes

Standard:

`get_contour_attrib_xld`, `get_contour_global_attrib_xld`,
`query_contour_attribs_xld`, `query_contour_global_attribs_xld`

Transform Results Into World Coordinates

Standard:

`contour_to_world_plane_xld`

More operators for transforming results into world coordinates are described in the [Application Note on 3D Machine Vision](#).

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.6.6 Relation to Other Methods

3.6.6.1 Alternatives to Edge Extraction (subpixel-precise)

Subpixel Thresholding

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

Subpixel Point Extraction

In addition to the contour-based subpixel-accurate data, HALCON offers subpixel-accurate point operators for various applications. In the reference manual, these operators can be found in the chapter [“Filter > Points”](#).

3.6.7 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (HALCON term for region of interest) is very important for contour extraction. With domains, the processing can be focused to a certain area in the image and can thus be sped up. The more the region in which edges or lines are extracted can be restricted, the faster and more robust the extraction will be. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

3.6.8 Advanced Topics

Contour Processing

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

3.7 Contour Processing

One of HALCON's powerful tool sets are the subpixel-accurate contours. Contours belong to the data type XLD (see [section 2.1.2.3](#) on page 12 for more information). These contours are typically the result of some kind of image processing and represent, e.g., the borders of objects. [Figure 3.13a](#) shows such edges overlaid on the original image; [Figure 3.13b](#) zooms into the rectangular area marked in [Figure 3.13a](#) and furthermore highlights the so-called control points of the contours with crosses. Here, you can clearly see the highly accurate positioning of the control points.

HALCON provides operators to perform advanced types of measurements with these contours. For example, the contours can be segmented into lines and circular or elliptic arcs (see [Figure 3.13c](#)). The parameters of these segments, e.g., their angle, center, or radius, can then be determined and used, e.g., in the context of a metrology task.

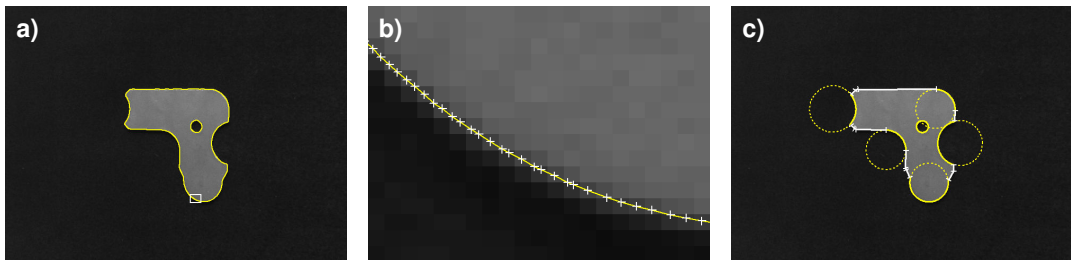
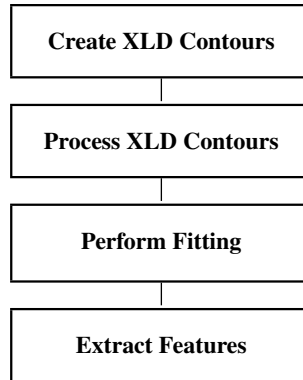


Figure 3.13: XLD contours: (a) edge contours, (b) zoom into rectangular area, (c) segmented lines and elliptic arcs.

The advantage of contour processing is twofold: First, its high accuracy enables reliable measurements. Secondly, the extensive and flexible set of operators provided for this data type enables you to solve problems that cannot be solved with classical methods like metrology.

3.7.1 Basic Concept

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



Create XLD Contours

The most common way to create XLD contours is to apply one of the subpixel-accurate extraction operators described for the method [Extract Edges Or Lines](#) on page 76. As an alternative, an edge filter with some post-processing can be used. The resulting regions are then converted to XLD contours. Please note that this approach is only pixel-accurate. For more information about this approach see the method [Edge Extraction \(pixel-precise\)](#) on page 64.

Process XLD Contours

Typically, only certain contours of an object are used for an inspection task. One possibility to restrict the extraction of contours to the desired ones is to use a well-fitting region of interest as, e.g., depicted in [figure 3.14a](#): The rectangular ROI just covers the upper part of the blades. When applying an edge extractor, exactly one contour on each side of the objects is found.

In many cases, however, not only the desired contours are extracted. An example is depicted in [figure 3.14b](#), where the ROI was chosen too large. Thus, the contours must be processed to obtain the desired parts of the contours. In the example, the contours are segmented into parts and only parallel segments with a certain length are selected (see the result in [figure 3.14c](#)).

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

Perform Fitting

Having obtained contour segments that represent a line or a circular or elliptic arc, you can determine the corresponding parameters, e.g., the coordinates of the end points of a line or the center and radius of a circle, by calling one of the fitting operators. Their goal is to approximate the input contour as closely as possible to a line or elliptic or circular arc. Because the used minimization algorithms are very advanced and all contour points are used for the process, the parameters can be calculated very reliably.

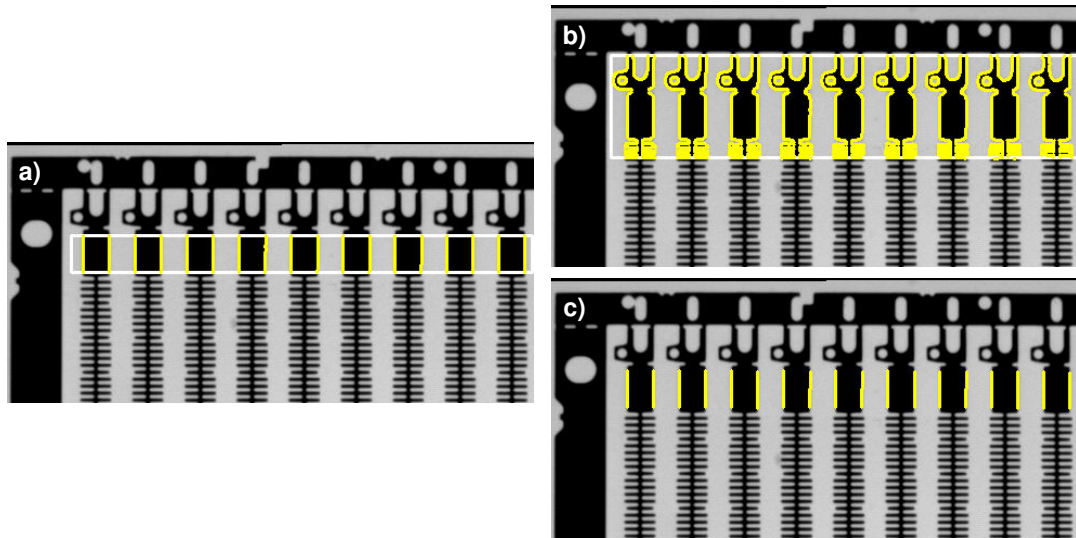


Figure 3.14: Selecting the desired contours: (a) exactly fitting ROI, (b) too many contours because of too large ROI, (c) result of post-processing the contours from (b).

Extract Features

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

A First Example

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

First, an image is acquired from file using `read_image`. The task is to extract the roads, which show up as thin bright lines in the image. For this the operator `lines_gauss` is used. When we look at the result of the contour extraction in [figure 3.15a](#), we see that a lot of unwanted small segments are extracted. They can be suppressed easily by calling `select_contours_xld` with a minimum contour length. A further problem is that some roads are split into more than one segment. They can be combined with the operator `union_collinear_contours_xld`. Looking at the result in [figure 3.15b](#), we see that many fragments have been combined along straight road parts. In curves this method fails because the orientation of the segments differs too much.

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

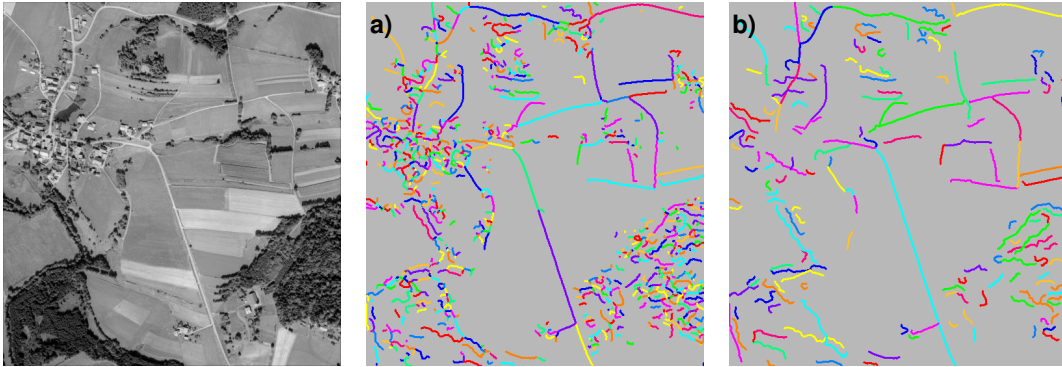
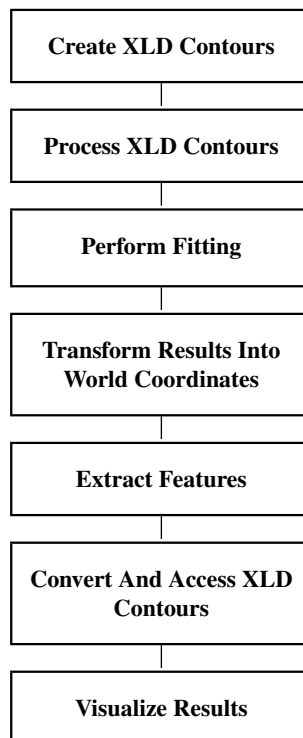


Figure 3.15: Processing XLD contours, (a) extracted contours, (b) processed contours.

3.7.2 Extended Concept

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



Create XLD Contours

The standard method to create contours is to call a contour extraction operator. Contour extraction for edges is performed with `edges_sub_pix`, `edges_color_sub_pix`, or `zero_crossing_sub_pix`. Lines are extracted using `lines_gauss`, `lines_facet`, or `lines_color`. For subpixel blob analysis the operator `threshold_sub_pix` can be used. These operators are described in more detail with the method [Edge Extraction \(subpixel-precise\)](#) on page 74.

If pixel-accuracy is sufficient, you can use an edge filter (like `sobel_amp` or `edges_image`) or a line filter (like `bandpass_image`) followed by thresholding and thinning. The resulting elongated regions are then converted into XLD contours with the operator `gen_contours_skeleton_xld`. For more information on this approach see the method [Edge Extraction \(pixel-precise\)](#) on page 64.

Contours can also be synthesized from different sources, e.g., CAD data, user interaction, or metrology. Having obtained the coordinates of the control points from such a source, the operators `gen_contour_polygon_xld` and `gen_contour_polygon_rounded_xld` convert them to XLD contours. You can also draw XLD contours interactively with the operators `draw_xld` and `draw_xld_mod`.

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

Process XLD Contours

The first method to segment contours is to call `segment_contours_xld`. This operator offers various modes: Splitting into line segments, linear and circular segments, or linear and elliptic segments. The individual contour segments can then be selected with `select_obj` and passed to one of the fitting operators described with the step [Perform Fitting](#) on page 85. Whether a contour segment represents a line, a circular, or an elliptic arc can be queried via the global contour attribute 'cont_approx' using the operator `get_contour_global_attrib_xld`.

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

An important step during contour processing is the suppression of irrelevant contours. This can be accomplished with the operator `select_shape_xld`, which provides almost 30 different shape features. By specifying the desired minimum and maximum value and possibly combining multiple features, contours can be selected very flexibly. As an alternative, you can use the operator `select_contours_xld`, which offers typical features of linear structures. Finally, the operator `select_xld_point` can be used in combination with mouse functions to interactively select contours.

If there are gaps within a contour, the pieces are treated as separate objects, which makes further processing and feature extraction difficult. You can merge linear segments with the operators `union_collinear_contours_xld` or `union_straight_contours_xld`. To handle contours with a complex shape you must first segment them into linear segments (see above).

HALCON also provides an operator for general shape modifications: `shape_trans_xld`. With this operator you can, e.g., transform the contour into its surrounding circle, convex hull, or surrounding rectangle.

Perform Fitting

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

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

Transform Results Into World Coordinates

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

How to transform contours into world coordinates is described in detail in the Application Note on 3D Machine Vision in [section 3.2](#) on page 44.

Extract Features

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

Convert And Access XLD Contours

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

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

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

The operators for edge and line extraction not only return the XLD contours but also so-called attributes. Attributes are numerical values; they are associated either with each control point (called contour attribute) or with each contour as a whole (global contour attribute). The operators `get_contour_attrib_xld` and `get_contour_global_attrib_xld` enable you to access these values by specifying the attribute name. More information on this topic can be found in the description of the step [Determine Contour Attributes](#) on page 77.

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.7.3 Industries

3.7.3.1 Machinery

A typical task in quality control is to inspect the exact dimensions of objects. For objects with complex shapes, which quite often occur in machinery, the flexibility of HALCON's contour processing is ideal. For a corresponding example see [circles.dev](#) on page 200.

3.7.3.2 Electric Components And Equipment

Color is a useful extra information if the gray values are not sufficient to extract components. For cables, it is difficult to locate the centers and the extents if this information is not available. Using the subpixel-accurate color line extractor, both features can be extracted in a robust manner. For a corresponding example see [lines_color.dev](#) on page 191.

3.7.3.3 Photogrammetry And Remote Sensing

A very complex task is the extraction of objects like roads from an aerial image. The example program [roads.dev](#) on page 207 demonstrates the powerful contour processing methods of HALCON. Both lines (as centers of road candidates) and edges (as candidates for road borders) are input for the process. By a stepwise verification and grouping process the road candidates are extracted.

3.7.4 Programming Examples

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

3.7.4.1 Measuring Lines and Arcs

Example: [examples\quick_guide\hdevelop\measure_metal_part.dev](#)

The first example shows how to segment a contour into lines and (circular) arcs and how to determine the corresponding parameters. [Figure 3.16](#) shows the final result of the fitted primitives overlaid on the input image.

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

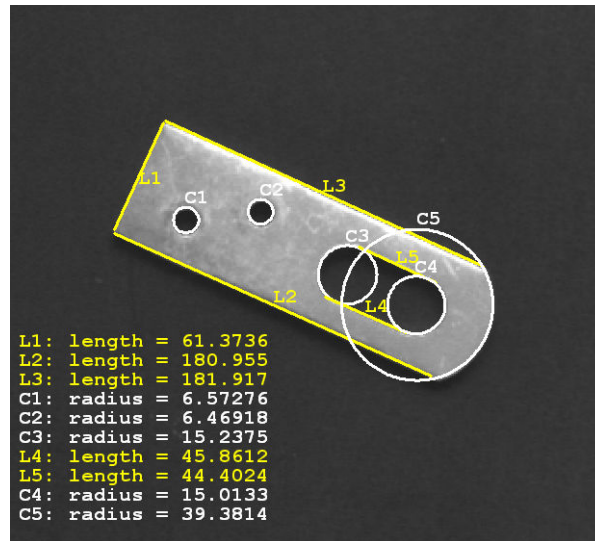


Figure 3.16: Fitted lines and circles.

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

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

```

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

```

3.7.4.2 Close gaps in a contour

Example: [examples\quick_guide\hdevelop\close_contour_gaps.dev](#)

The second example demonstrates how to close gaps in an object contour (see [figure 3.17](#)). The example is based on synthetic data. Instead of using a real image, a light gray square on a dark gray background is generated and a part of its boundary is blurred.

```

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

```

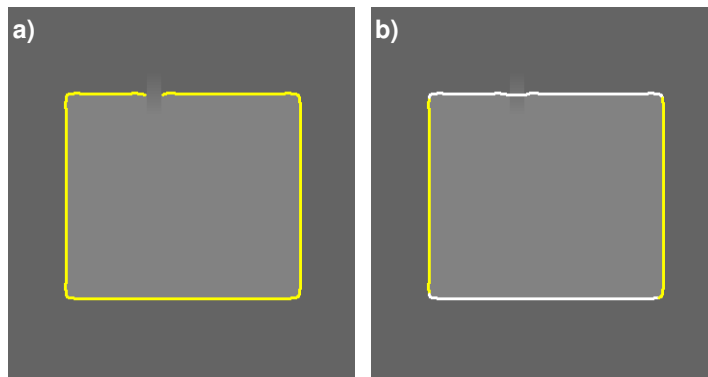


Figure 3.17: Original edges and result of grouping process.

The extraction of contours with `edges_sub_pix` thus results in an interrupted boundary (see [figure 3.17a](#)). Note that the edge extraction is restricted to the inner part of the image, otherwise edges would be extracted at the boundary of the image.

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

A suitable operator for closing gaps in linear segments is `union_collinear_contours_xld`. Before we can apply this operator, some pre-processing is necessary: First, the contours are split into linear segments using `segment_contours_xld`. Then, `regress_contours_xld` is called to determine the regression parameters for each segment. These parameters are stored with each contour and could be accessed with `get_regress_params_xld`. Finally, `union_collinear_contours_xld` is called. Its result is depicted in [figure 3.17b](#) on page 92.

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

3.7.4.3 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_exterior.dev](#)
Measures positions on a caliper rule using camera calibration
→ description in the [Application Note on 3D Machine Vision](#) on page 41
- [examples\application_guide\3d_machine_vision\hdevelop\handeye_stationarycam_grasp_nut.dev](#)
Calculates pose for grasping a nut based on results of hand-eye calibration for a stationary camera
→ description in the [Application Note on 3D Machine Vision](#) on page 120
- [examples\application_guide\3d_machine_vision\hdevelop\radial_distortion.dev](#)
Eliminating radial distortions
→ description in the [Application Note on 3D Machine Vision](#) on page 48
- [examples\hdevelop\Applications\Aerial\roads.dev](#)
Extraction of roads from aerial image
→ description [here in the Quick Guide](#) on page 207
- [examples\hdevelop\Applications\Calibration\lens_distortions.dev](#)
Eliminates distortions of edges extracted from a distorted image
- [examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev](#)
Measures distances between the pitch lines of a caliper rule in a line scan image using camera calibration

- [examples\hdevelop\Applications\FA\circles.dev](#)
Fits circles into curved contour segments
→ description [here in the Quick Guide](#) on page 200
- [examples\hdevelop\Applications\Medicine\angio.dev](#)
Extracts blood vessels and their diameters from an angiogram
- [examples\hdevelop\Filter\Lines\lines_color.dev](#)
Extracting lines using color information
→ description [here in the Quick Guide](#) on page 191
- [examples\hdevelop\Filter\Lines\lines_gauss.dev](#)
Extracting lines and their widths
→ description [here in the Quick Guide](#) on page 196
- [examples\hdevelop\Graphics\Output\disp_xld.dev](#)
Displaying an XLD object
- [examples\hdevelop\Graphics\Output\dump_window_data.dev](#)
Dump the content of the graphics window to an image object
- [examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev](#)
Rectifies image of stadium to simulate overhead view
- [examples\hdevelop\Tools\Calibration\change_radial_distortion_contours_xld.dev](#)
Eliminating radial distortions from extracted contours
- [examples\hdevelop\Tools\Geometry\distance_cc_min.dev](#)
Calculating the distance between two contours
- [examples\hdevelop\XLD\Creation\gen_contour_region_xld.dev](#)
Extracting the contours of regions as XLD objects
- [examples\hdevelop\XLD\Features\fit_circle_contour_xld.dev](#)
Approximating an XLD contour with a circle
- [examples\hdevelop\XLD\Features\fit_ellipse_contour_xld.dev](#)
Approximating XLD contours with ellipses or elliptic arcs
- [examples\hdevelop\XLD\Features\fit_ellipse_tooth_rim_xld.dev](#)
Approximating the contour of a tooth rim with an ellipse to find its center.
- [examples\hdevelop\XLD\Features\fit_line_contour_xld.dev](#)
Approximating XLD contours with line segments
- [examples\hdevelop\XLD\Features\select_xld_point.dev](#)
Selecting XLD contours containing a specified point
- [examples\hdevelop\XLD\Features\shape_trans_xld.dev](#)
Transforming contours into various standard shapes
- [examples\hdevelop\XLD\Features\test_self_intersection_xld.dev](#)
Testing if an XLD contour intersects itself

- [examples\hdevelop\XLD\Features\test_xld_point.dev](#)
Testing if an XLD contour contains a specific point
- [examples\hdevelop\XLD\Transformation\clip_contours_xld.dev](#)
Clipping an XLD contour
- [examples\hdevelop\XLD\Transformation\close_contours_xld.dev](#)
Closing XLD contours
- [examples\hdevelop\XLD\Transformation\crop_contours_xld.dev](#)
Cropping an XLD contour
- [examples\hdevelop\XLD\Transformation\gen_parallel_contour_xld.dev](#)
Computing the parallel contour of an XLD contour
- [examples\hdevelop\XLD\Transformation\projective_trans_contour_xld.dev](#)
This program uses `hom_mat3d_project` and `projective_trans_region` to rotate an XLD contour in 3D
- [examples\hdevelop\XLD\Transformation\sort_contours_xld.dev](#)
Sorting XLD contours by spatial position
- [examples\hdevelop\XLD\Transformation\union_cocircular_contours_xld.dev](#)
Merging contours belonging to the same circle
- [examples\hdevelop\XLD\Transformation\union_contours_xld.dev](#)
Connecting collinear line segments
- [examples\quick_guide\hdevelop\critical_points.dev](#)
Locates saddle point markers in an image
→ description [here in the Quick Guide](#) on page 31

C++

- [examples\cpp\source\example10.cpp](#)
Working with contour data and zooming results
- [examples\cpp\source\example9.cpp](#)
Extracts roads from aerial images

3.7.5 Selecting Operators

Create XLD Contours

Standard:

[gen_contour_polygon_xld](#), [gen_contour_region_xld](#), [gen_ellipse_contour_xld](#),
[draw_xld](#), [draw_xld_mod](#)

Advanced:

[gen_contour_polygon_rounded_xld](#)

Further operators can be found in the following places: [detailed operator list for the step Extract Edges Or Lines](#) on page 82, [operator list for the method Edge Extraction \(pixel-precise\)](#) (see section 3.5.5 on page 71).

Process XLD Contours

Standard:

```
segment_contours_xld, gen_polygons_xld, split_contours_xld, select_shape_xld,  
select_contours_xld, select_xld_point, union_collinear_contours_xld,  
union_straight_contours_xld, shape_trans_xld
```

Advanced:

```
union_collinear_contours_ext_xld
```

Perform Fitting

Standard:

```
fit_line_contour_xld, fit_circle_contour_xld, fit_ellipse_contour_xld
```

Transform Results Into World Coordinates

Standard:

```
contour_to_world_plane_xld, change_radial_distortion_contours_xld
```

More operators for transforming results into world coordinates are described in the [Application Note on 3D Machine Vision](#).

Extract Features

Standard:

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

A full list of operators can be found in the Reference Manual in the chapter “[XLD ▸ Features](#)”.

Convert And Access XLD Contours

Standard:

```
get_contour_xld, gen_region_contour_xld
```

Advanced:

`paint_xld`

More information on operators for accessing XLD data can be found in the Reference Manual in the chapter “XLD ▷ Access”. Operators for determining contour attributes can be found in the [detailed operator list for the step Determine Contour Attributes](#) on page 82.

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.7.6 Relation to Other Methods

3.7.6.1 Alternatives to Contour Processing

Line Processing

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

3.7.7 Advanced Topics

Line Scan Cameras

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

3.8 Template Matching

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

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

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

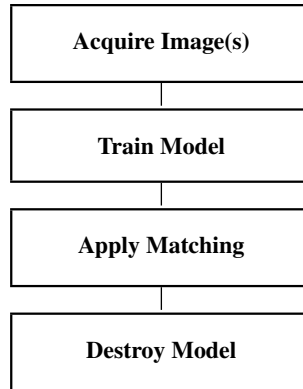
- The *gray-value-based matching* is the classical method. It can be used if the gray values inside the object do not vary and if there are no missing parts and no clutter. The method can handle single instances of objects, which can appear rotated in the search image.
- The *shape-based matching* represents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

How to use shape-based matching is described in detail in the [Application Note on Shape-Based Matching](#).

- The *component-based matching* can be considered as a high-level shape-based matching: The enhancement is that an object can consist of multiple parts that can move (rotate and translate) relative to each other. A simple example of this is a pair of pliers. Logically this is considered as one object, but physically it consists of two parts. The component-based matching allows handling such a compound object in one search step. The advantage is an improved execution time and increased robustness compared to handling the parts as distinct models.
- The *point-based matching* has the intention to combine two overlapping images. This is done by first extracting significant points in both images. These points are the input for the actual matching process. The result of the matching is a mapping from one image to the other, allowing translation, rotation, scaling, and perspective distortions. This mapping is typically used to combine the two images into a single, larger one. Of course, one image can also be treated as a template and the other image as showing an instance of the object that should be found. The advantage is the ability to handle perspective distortions without calibration. The disadvantage is the increased execution time, which comes mainly from the extraction of the significant points.

3.8.1 Basic Concept

Template matching is divided into the following parts:



Acquire Image(s)

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

For detailed information see the [description of this method](#) on page 22.

Train Model

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

Apply Matching

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

Destroy Model

When you no longer need the matching model you should destroy it using `clear_shape_model`.

A First Example

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

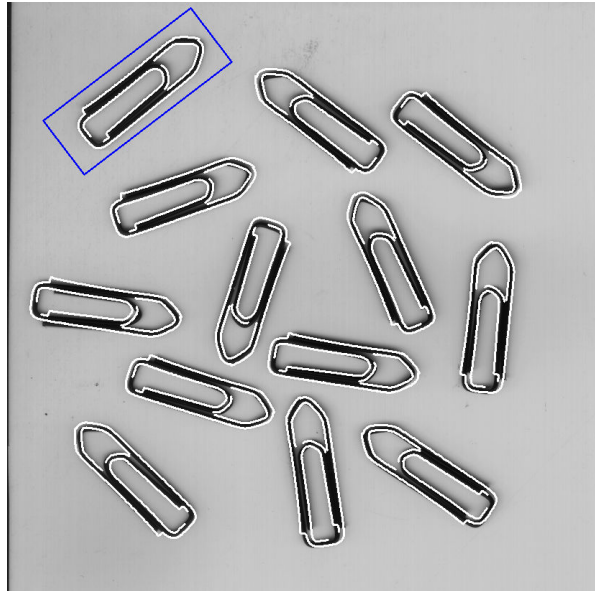


Figure 3.18: Finding all clips in the image.

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

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


3.8.2 Extended Concept

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

Figure 3.19 shows the major steps. The first part is offline and consists of the training of the model, using different input sources that depend on the application. The model can then be stored to be loaded again for later use. For the matching itself, one or more models are used to search for objects in images. The results are the poses, i.e., the position, rotation, and scale of the found objects.

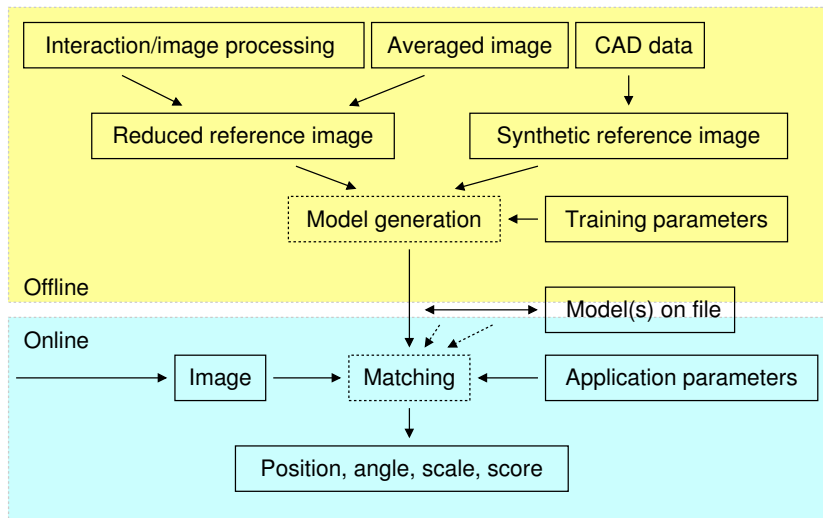
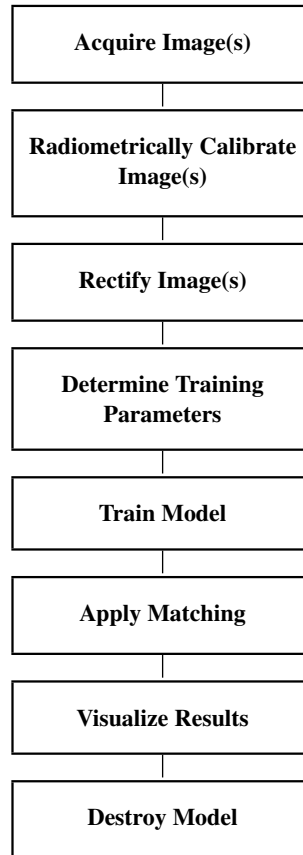


Figure 3.19: Overview of the matching process.



Radiometrically Calibrate Image(s)

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

Rectify Image(s)

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

Detailed information about rectifying images can be found in the Application Note on 3D Machine Vision in [section 3.3](#) on page 49.

Determine Training Parameters

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

The operator `create_scaled_shape_model` allows to use the value 0 for `NumLevels`, `AngleStep`, and `ScaleStep`. This has the effect that the values are determined automatically. `AngleStart`, `AngleExtent`, `ScaleMin`, and `ScaleMax` are typically known from the setup of the system. They can also be determined easily using `HMatchIt` with multiple test images and checking the values in the dialog `Run > Determine Pose Bounds`. If `'no_pregeneration'` is used for the parameter `Optimization` wide ranges can be used and restricted during the finding process. The other training parameters (`Contrast`, `MinContrast`) can be determined using `HMatchIt`. If an automatic method is needed, please contact your local distributor. Their support can provide application-specific procedures to determine the parameters.

Train Model

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

- You can use both real and synthetic images for the training. Using real images is easier. However, if no such image is available or if the real images are too noisy you can create a synthetic image, e.g., with the operators `gen_contour_polygon_rounded_xld`, `gen_image_const`, and `paint_xld`. If the training image is too noisy, you can enhance it, e.g., by smoothing noisy areas or by averaging multiple templates into a single one.
- To speed up the matching process, a so-called image pyramid is created, consisting of the original, full-sized image and a set of downsampled images. The model is then created and searched on the different pyramid levels. You can view the image pyramid and the corresponding models for shape-based matching in `HMatchIt`.
- Each model (except for the point-based matching) is manipulated via a handle. This handle points to the real data, which can consume a large amount of memory. Therefore, it is important to clear the model when it is no longer needed.

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

- To create a model for *gray-value-based matching*, the operator `create_template_rot` is used. The model can then be stored to file with `write_template`
- To create a model for the *shape-based matching*, the operator `create_scaled_shape_model` is used. The other operator `create_shape_model` is just a convenient method if no scaling is needed. The model can then be stored to file with `write_shape_model`
- For the *component-based matching*, the model can be created in different ways: If the components and their relations are known, you can use the operator `create_component_model`. If the components are known, but not their relations, you can train the relations using the operator `train_model_components` with a set of training images, and then create the model using `create_trained_component_model`. Finally, if the components themselves are not known, you can

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

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

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

Apply Matching

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

- You can search objects with *gray-value-based matching* using the operator `create_template_rot`. This operator finds single instances of rotated objects. If the objects are not rotated, the operator `fast_match_mg` allows to find all points with a high score. These locations are returned as a region and can be further processed, e.g., with blob analysis (see the [description of this method](#) on page 39).
- The standard operator for *shape-based matching* is `find_scaled_shape_model`. It allows locating multiple instances of the same template with rotations and scaling. If multiple models are needed, the operator `find_scaled_shape_models` is used.
- For the *component-based matching*, `find_component_model` is used to locate multiple instances of objects, which may be rotated and whose components may move with respect to each other.
- The matching operator for the *point-based matching* is `proj_match_points_ransac`. The input are the two images containing the template *and* the object to be located, together with the significant points. The result of the matching process is a 2D mapping from the template image to the object image. This mapping is the input for operators like `gen_projective_mosaic`, which can be used to combine the two images.

Visualize Results

A typical visualization task is to display the model contours overlaid on the image at the found position. An easy way to do this is to access the model contours by calling `get_shape_model_contours` (for the shape-based matching). With `vector_angle_to_rigid` and `disp_xld`, the contours can be displayed. Displaying the results of the component-based matching is more complex. The easiest way is to call `get_found_component_model`, which returns regions, one for each component. These can directly be displayed with `disp_region`.

For detailed information see the [description of this method](#) on page 173.

3.8.3 Industries

3.8.3.1 Semiconductors

Semiconductor industry is the typical application area for template matching. Here, the main task is to find objects and to determine their position accurately. For a corresponding example see the description of [pm_measure_board.dev](#) on page 213.

Another example from semiconductor industry shows how to locate multiple components on a printed circuit board in one step. For a corresponding example see the description of [cbm_modules_simple.dev](#) on page 235.

A last example from semiconductor industry shows how to combine multiple low-resolution images into one high-resolution image. This allows to cover a large area using standard camera technology and thus to save money. For a corresponding example see the description of [gen_projective_mosaic.dev](#) on page 214.

3.8.3.2 Electric Components And Equipment

An example from the electric components industry shows the possibilities of the component-based matching. The classic approach would be to solve this task with matching for alignment, followed by multiple measurements to get the position of the sub-parts. Here, the inspection is performed in one step (see [cbm_dip_switch.dev](#) on page 190).

3.8.3.3 Printing

Matching is often a useful step to prepare a print inspection task: After locating the object that must be tested, the reference pattern can be aligned and thus be compared easily. For a corresponding example see [print_check.dev](#) on page 237.

3.8.4 Programming Examples

This section gives a brief introduction to using HALCON for template matching. The focus here is on the shape-based matching.

3.8.4.1 Creating a Model for the “Green Dot”

Example: [examples\quick_guide\hdevelop\create_model_green_dot.dev](#)

This example shows how to use the shape-based matching with objects of varying size. [Figure 3.20](#) depicts the training image, which contains the so-called “green dot”, a symbol used in Germany for recycling packages. The template is not defined by a user interaction but by a segmentation step: Using [threshold](#) all dark pixels are selected and the connected component with the appropriate size is chosen ([select_shape](#)). This region is then filled and slightly dilated. [Figure 3.20a](#) depicts the result of the segmentation.

```

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

```

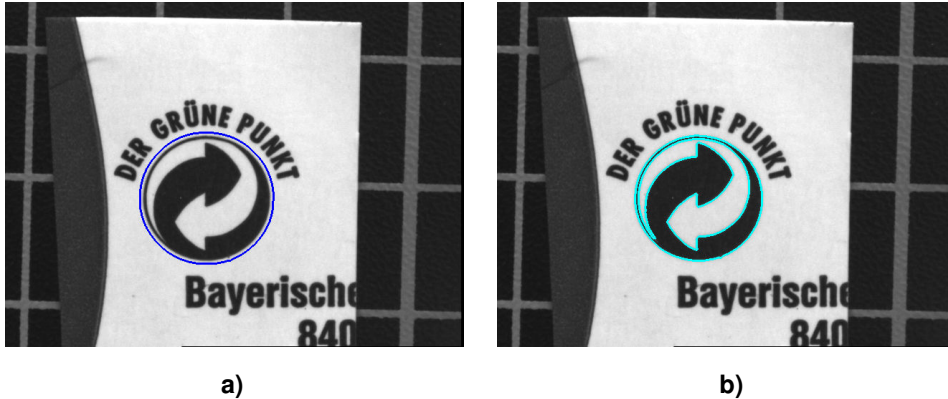


Figure 3.20: Creating a model for template matching: (a) ROI for the template region; (b) model contours.

The extracted region is then combined with the image (`reduce_domain`) to be used as the domain, i.e., as the region of interest. To check whether the value for the parameter `Contrast` has been chosen correctly, `inspect_shape_model` is called. The training is finally applied using `create_scaled_shape_model`. The resulting model contours are depicted in [figure 3.20b](#).

```

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

```

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

```

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

```

3.8.4.2 Locating “Green Dots”

Example: [examples\quick_guide\hdevelop\matching_green_dot.dev](#)

In this example, we use the model created in the previous example to locate the so-called “green dots” in a search image. As you can see in [figure 3.21a](#), the search image contains three “green dots” in different orientations and scales. Furthermore, some of them are partly occluded.



Figure 3.21: Using template matching to locate rotated and scaled objects: (a) search image; (b) matches.

First, the shape model is read from file.

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

Then, the search operator `find_scaled_shape_model` is executed. The result of the operator are the positions, the orientations, and the scales of the “green dots”. To display the model contours overlaid on the image, the contours are accessed with `get_shape_model_contours`. The for-loop is used to handle the contours for each found location. This is done by generating an appropriate transformation and then moving the contours to the correct position. Figure 3.21b depicts the result.

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

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

```
clear_shape_model (ModelID)
```

3.8.4.3 Other Examples

3.8.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Radiometrically Calibrate Image(s)

Standard:

`radiometric_self_calibration, lut_trans`

Rectify Image(s)

Operators for rectifying images are described in the [Application Note on 3D Machine Vision](#).

Determine Training Parameters

Standard:

`determine_shape_model_params, get_shape_model_params, inspect_shape_model`

Train Model

Standard:

`create_template_rot, create_scaled_shape_model, create_component_model, gen_initial_components, train_model_components, create_trained_component_model, points_foerstner, points_harris`

Apply Matching

Standard:

`best_match_mg, find_scaled_shape_model, find_component_model, proj_match_points_ransac, gen_projective_mosaic`

Advanced:

`best_match_pre_mg, fast_match_mg, find_scaled_shape_models, projective_trans_image_size`

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

Destroy Model

Standard:

```
clear_template, clear_shape_model, clear_component_model,  
clear_training_components
```

3.8.6 Relation to Other Methods

3.8.6.1 Methods that are Using Template Matching

1D Measuring (see [description](#) on page 53)

OCR (see [description](#) on page 145)

1D Bar Code (see [description](#) on page 125)

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

A detailed description of alignment can be found in the Application Note on Shape-Based Matching in [section 4.3.4](#) on page 36. For an example of using alignment as a preprocessing for the so-called variation model in a print inspection application see the description of [print_check.dev](#) on page 237.

3.8.6.2 Alternatives to Template Matching

Blob Analysis (see [description](#) on page 39)

In some applications, the object to find can be extracted with classical segmentation methods. With the operators [area_center](#) and [orientation_region](#) you can then determine its pose and use this information, e.g., to align an ROI. With this approach, the execution time can be reduced significantly.

If the objects to be found appear only translated but not rotated or scaled, the morphological operators [erosion1](#) and [opening](#) can be used as binary template matching methods. Unlike in other vision systems, in HALCON these operators are extremely fast.

3.8.7 Tips & Tricks

Use of Domains (Regions of Interest)

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

- The reference point of the template is defined by the center of gravity of the *ROI* used during creation, not by the center of gravity of the contours returned by [inspect_shape_model](#).

- During the search process, only the reference point of the model must fit into the search ROI, not the complete shape. Therefore, sometimes very small ROIs consisting only of a few pixels can be used.

For an overview on how to construct regions of interest and how to combine them with the image see [Region Of Interest](#) on page 27.

Speed Up

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

- Regions of interest are the standard way to increase the speed by processing only those areas where objects need to be inspected. This can be achieved using pre-defined regions, but also by an online generation of the regions of interest that depends on other objects in the image.
- If multiple objects are searched for, it is more efficient to use `find_scaled_shape_models` instead of using `find_scaled_shape_model` multiple times.
- Increasing the values for the parameters `MinContrast` and `Greediness` will decrease the execution time. However, you must make sure that the relevant objects will still be found.
- Using a lower value for the parameter `Contrast` during the training typically results in a better performance because more pyramid levels can be used. If the contrast is too low, irrelevant contours will also be included into the model, which typically causes a lower recognition rate and a decreased accuracy.
- Very small templates cannot be found as quickly as larger ones. The reason for this is the reduced number of significant contours within an image, which makes it harder to distinguish the template from other structures. Furthermore, with smaller templates fewer pyramid levels can be used.

3.8.8 Advanced Topics

High Accuracy

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

3.9 Color Processing

The idea of color processing is to take advantage of the additional information encoded in color or multi-spectral images. Processing color images can simplify many machine vision tasks and provide solutions to certain problems that are simply not possible in gray value images. In HALCON the following approaches in color processing can be distinguished: First, the individual channels of a color image can be processed using standard methods like blob analysis. In this approach the channels of the original image have to be decomposed first. An optional color space transformation is often helpful in order to access specific properties of a color image. Secondly, HALCON can process the color image as a whole by calling specialized operators, e.g., for pixel classification. Advanced applications of color processing include lines and edges extraction.

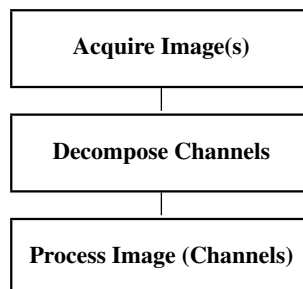


Figure 3.22: Simple color segmentation.

The example illustrated in [figure 3.22](#) shows how to segment blue pieces of plasticine in a color image.

3.9.1 Basic Concept

Simple color processing, which is using the methods of blob analysis, mainly consists of three parts:



Acquire Image(s)

First, an image is acquired.

For detailed information see the [description of this method](#) on page 22.

Decompose Channels

In order to be able to process the individual channels, RGB color images have to be split up into a red, green, and blue channel by using the operator `decompose3`.

Process Image (Channels)

Depending on the application, the individual channels can be processed using standard methods described in this chapter. One of the most frequently used methods is blob analysis (see [Blob Analysis](#) on page 39).

A First Example

An example for this basic concept is the following program, which belongs to the example explained above.

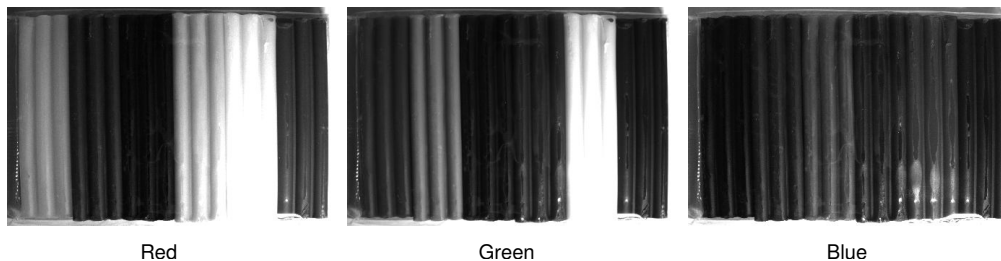


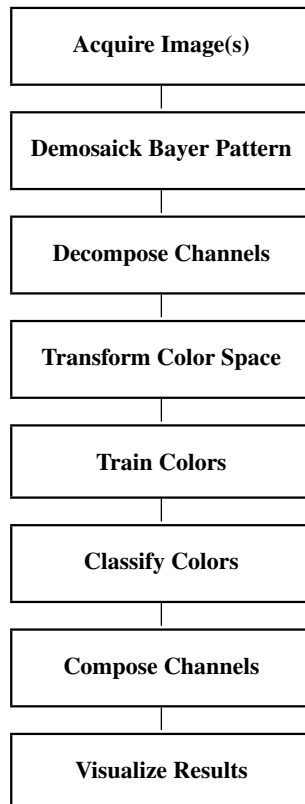
Figure 3.23: Color image decomposed to its red, green, and blue channels.

Here, an RGB image is acquired from file. The image is split into its channels using `decompose3`. The red and green channels are subtracted from the blue channel using `sub_image`. The purpose of this process is to fade out pixels with high values in the other channels, leaving pure blue pixels only. Using `threshold`, the blue pixels with a certain intensity are selected.

```
read_image (Image, 'plasticine')
decompose3 (Image, Red, Green, Blue)
sub_image (Blue, Red, RedRemoved, 1, 0)
sub_image (RedRemoved, Green, RedGreenRemoved, 1, 0)
threshold (RedGreenRemoved, BluePixels, 10, 255)
```

3.9.2 Extended Concept

In many cases the processing of color images will be more advanced than in the above example. Depending on the actual application, the order of the following steps may vary, or some steps may be omitted altogether.



Demosaick Bayer Pattern

If the acquired image is a Bayer image, it can be converted to RGB using the operator `cfa_to_rgb`. The encoding type of the Bayer pattern (the color of the first two pixels of the first row) must be known (see [figure 3.24](#)).

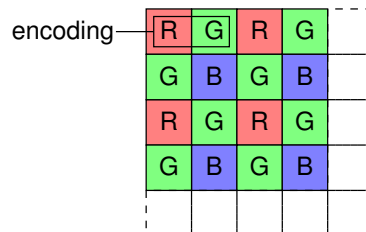


Figure 3.24: Sample Bayer pattern and corresponding encoding.

Transform Color Space

The RGB color space is not always the most appropriate starting point to process color images. If this is the case, a transformation to a different color space might be useful. HALCON supports many important color spaces. Namely, the HSV and HSI color spaces are favorable to select distinct colors independent of their intensity. Therefore, color segmentations in these color spaces are very robust under varying illumination. The `il1i2i3` color space qualifies for color classification, whereas the `cielab` color space is a close match to human perception.

Train Colors

In order to do color classification the colors that need to be distinguished have to be trained. There are different approaches for full color classification including Euclidean, box, and multilayer perceptron (MLP) classification.

Classify Colors

The colors trained in the previous step are used in subsequent images to do the actual classification.

Compose Channels

Any number of channels can be joined to a multi-channel image using the operators `compose2` through `compose7`, or `append_channel`. This way, channels that were processed separately can be composed back to color images for visualization purposes.

Visualize Results

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

For detailed information see the [description of this method](#) on page 173.

3.9.3 Industries

3.9.3.1 Electric Components And Equipment

Many parts in electronic components make use of color coding. For a corresponding example see the description of `lines_color.dev` on page 191.

3.9.3.2 Semiconductors

The inspection of circuit boards can be greatly enhanced when evaluating color images. For a corresponding example see the description of `ic.dev` on page 215.

3.9.3.3 Food

Quality assurance tests in food inspection heavily rely on the use of color image processing.

3.9.3.4 Photogrammetry And Remote Sensing

The example program `forest.dev` on page 204 shows how to extract different object classes from an aerial image.

3.9.4 Programming Examples

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

3.9.4.1 Robust Color Extraction

Example: `examples\quick_guide\hdevelop\color_simple.dev`

The object of this example is to segment the yellow cable in a color image in a robust manner.

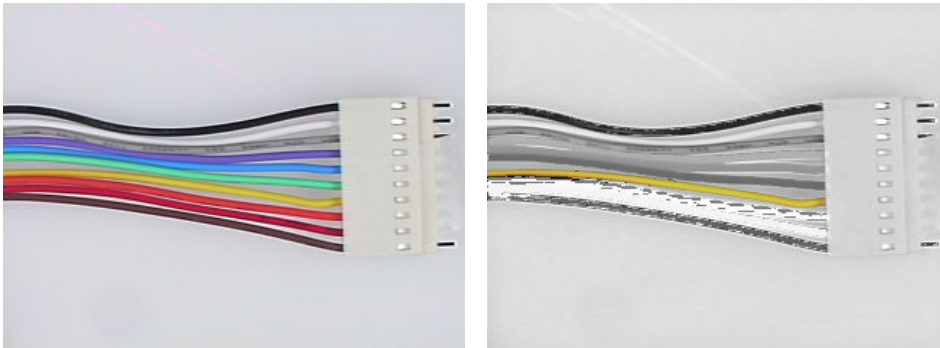


Figure 3.25: Segmentation of a specific color.

Here, an RGB image is acquired from file. The image is split into its channels using `decompose3`. Afterwards, a color space transformation from RGB to HSV is performed using `trans_from_rgb`. This transformation converts the image channels into the separate components hue, saturation and intensity. In the next steps the operator `threshold` selects all pixels with a high saturation value, followed by `reduce_domain` in the hue channel which effectively filters out pale colors and grays. A histogram of the remaining saturated (vivid) colors is displayed in [figure 3.26](#). Each peak in this histogram corresponds to a distinct color. The corresponding color band is shown below the histogram. Finally, the last `threshold` selects the yellowish pixels.

```
read_image (Image, 'cable' + i)
decompose3 (Image, Red, Green, Blue)
trans_from_rgb (Red, Green, Blue, Hue, Saturation, Intensity, 'hsv')
threshold (Saturation, HighSaturation, 100, 255)
reduce_domain (Hue, HighSaturation, HueHighSaturation)
threshold (HueHighSaturation, Yellow, 20, 50)
```

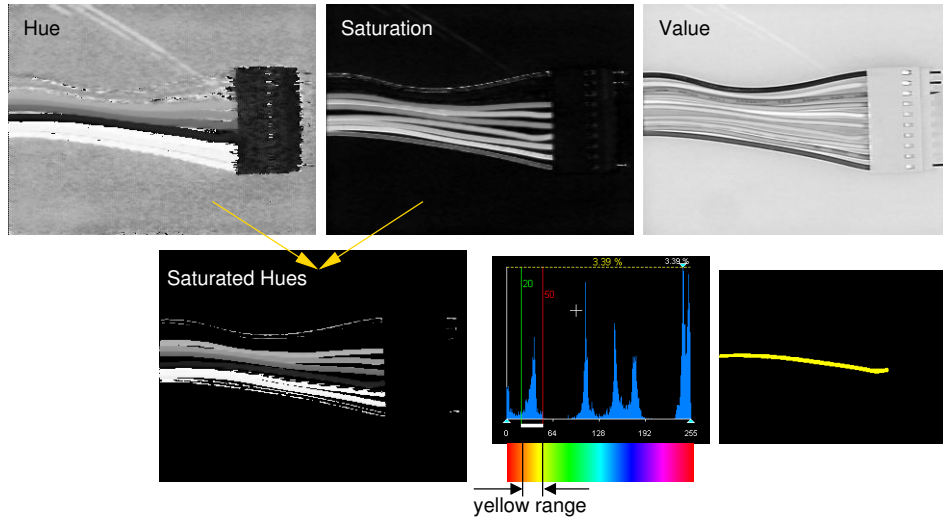


Figure 3.26: Segmentation in HSV color space.

Finding the proper threshold margins is crucial in applications like this. In HDevelop the Gray Histogram tool can be used to determine the values interactively. See the [HDevelop User's Manual](#) for more information. To generate the color band shown in [figure 3.26](#), use the following code snippet:

```
gen_image_gray_ramp (Hue, 0, 1, 128, 32, 128, 256, 64)
gen_image_proto (Hue, White, 255)
trans_to_rgb (Hue, White, White, Red, Green, Blue, 'hsv')
compose3 (Red, Green, Blue, MultiChannelImage)
```

3.9.4.2 Sorting Fuses

Example: [examples\quick_guide\hdevelop\color_fuses.dev](#)

In this example different types of fuses are classified using color images. The applied method is similar to the previous example. A training image has been used to specify ranges of hue for the fuse types that need to be distinguished. The determined ranges are hard-coded in the program.

```
FuseColors := ['Orange', 'Red', 'Blue', 'Yellow', 'Green']
FuseTypes := [5, 10, 15, 20, 30]
* HueRanges: Orange 10-30, Red 0-10...
HueRanges := [10, 30, 0, 10, 125, 162, 30, 64, 96, 128]
```

A sequence of images is acquired from file, converted to the HSV color space, and reduced to contain only saturated colors just like in the previous example. As already mentioned, color selection in this color space is pretty stable under changing illumination. That is why the hard-coded color ranges are



Figure 3.27: Simple classification of fuses by color with varying illumination.

sufficient for a reliable classification. However, it has to be kept in mind that a certain degree of color saturation must be guaranteed for the illustrated method to work.

```
decompose3 (Image, Red, Green, Blue)
trans_from_rgb (Red, Green, Blue, Hue, Saturation, Intensity, 'hsv')
threshold (Saturation, Saturated, 60, 255)
reduce_domain (Hue, Saturated, HueSaturated)
```

The classification iterates over the fuse types and checks for sufficiently large areas in the given hue range. This is done using blob analysis. Afterwards, an additional inner loop labels the detected fuses.

```
for Fuse := 0 to |FuseTypes|-1 by 1
  threshold (HueSaturated, CurrentFuse, HueRanges[Fuse*2],
    HueRanges[Fuse*2+1])
  connection (CurrentFuse, CurrentFuseConn)
  fill_up (CurrentFuseConn, CurrentFuseFill)
  select_shape (CurrentFuseFill, CurrentFuseSel, 'area', 'and', 6000,
    20000)
  area_center (CurrentFuseSel, FuseArea, Row1, Column1)
  dev_set_color ('magenta')
  for i := 0 to |FuseArea|-1 by 1
    set_tposition (WH, Row1[i], Column1[i])
    write_string (WH, FuseColors[Fuse] + ' ' + FuseTypes[Fuse] + ' Ampere')
  endfor
  set_tposition (WH, 24*(Fuse+1), 12)
  dev_set_color ('slate blue')
  write_string (WH, FuseColors[Fuse] + ' Fuses: ' + |FuseArea|)
endfor
stop ()
```

3.9.4.3 Completeness Check of Colored Game Pieces

Example: `examples\quick_guide\hdevelop\color_pieces.dev`

Completeness checks are very common in machine vision. Usually, packages assembled on a production line have to be inspected for missing items. Before this inspection can be done, the items have to be trained. In the example presented here, a package of game pieces has to be inspected. The game pieces come in three different colors, and the package should contain four of each type. The pieces themselves can be of slightly different shape, so shape-based matching is not an option. The solution to this problem is to classify the game pieces by color. The method applied here is a classification using neural nets (MLP classification).

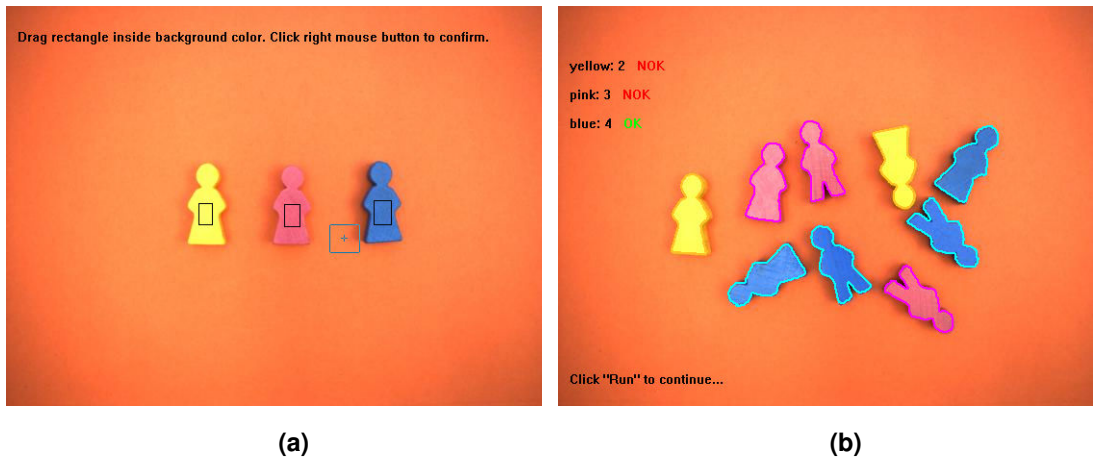


Figure 3.28: Example MLP classification: (a) Training, (b) Result.

In the training phase an image is acquired, which contains the different types of game pieces. The task is to specify sample regions for the game pieces and the background using the mouse (see [Figure 3.28a](#)). This is accomplished by looping over the `draw_rectangle1` and `gen_rectangle1` operators to draw and create the corresponding regions. The tuple `Classes`, which will be used for the actual training, is extended each time.

```
read_image (Image, ImageRootName+'0')
for i := 1 to 4 by 1
  dev_display (Image)
  dev_display (Classes)
  set_tposition (WindowHandle, 24, 12)
  write_string (WindowHandle, 'Drag rectangle inside ' + Regions[i-1] +
    ' color. Click right mouse button to confirm.')
  draw_rectangle1 (WindowHandle, Row1, Column1, Row2, Column2)
  gen_rectangle1 (Rectangle, Row1, Column1, Row2, Column2)
  Classes := [Classes, Rectangle]
endfor
```

Once the classes are specified, a multilayer perceptron is created using `create_class_mlp`. With the

operator `add_samples_image_class_mlp` the training samples from the image are added to the training data of the multilayer perceptron. The actual training is started with `train_class_mlp`. The duration of the training depends on the complexity and sizes of the training regions.

```
create_class_mlp (3, 4, 4, 'softmax', 'normalization', 3, 42, MLPHandle)
add_samples_image_class_mlp (Image, Classes, MLPHandle)
set_tposition (WindowHandle, 100, 12)
write_string (WindowHandle, 'Training...')
train_class_mlp (MLPHandle, 200, 1, 0.01, Error, ErrorLog)
```

After the training has finished, subsequent images are acquired and classified using `classify_image_class_mlp`. The operator returns a classified region.

```
for img := 0 to 3 by 1
  read_image (Image, ImageRootName + img)
  classify_image_class_mlp (Image, ClassRegions, MLPHandle, 0.5)
  stop ()
endfor
```

The returned result is processed further using blob analysis. Each class of the classified region (with the exception of the background class) is accessed using `copy_obj`. The regions of each class are split up using `connection` and reduced to regions of a relevant size (`select_shape`). The remaining few lines of code calculate the number of game pieces found for each class and make a decision whether the result was OK or not.

```
for figure := 1 to 3 by 1
  copy_obj (ClassRegions, ObjectsSelected, figure, 1)
  connection (ObjectsSelected, ConnectedRegions)
  select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 400,
    99999)
  Number := |SelectedRegions|
  dev_set_color (Highlight[figure-1])
  dev_display (SelectedRegions)
  dev_set_color ('black')
  set_tposition (WindowHandle, 24+30*figure, 12)
  write_string (WindowHandle, Regions[figure-1] + ': ' + Number)
  write_string (WindowHandle, ' ')
  dev_set_color ('green')
  if (Number#4)
    dev_set_color ('red')
    write_string (WindowHandle, 'N')
  endif
  write_string (WindowHandle, 'OK')
  dev_set_color ('black')
endfor
```

To illustrate the advantage of using color information, and to compare the classification results, the example program runs an additional training and classification on a converted gray image. As can be seen in [figure 3.29](#) only the yellow region is detected faithfully.

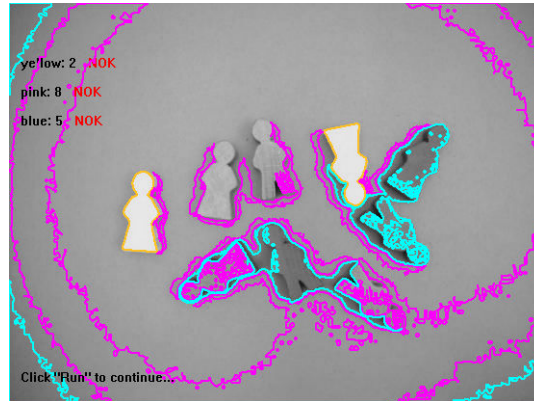


Figure 3.29: Poor classification result when only using the gray scale image.

```
rgb1_to_gray (Image, GrayImage)
compose3 (GrayImage, GrayImage, GrayImage, Image)
```

For more complex (and time-consuming) classifications, it is recommended to save the training data to the file system using `write_class_mlp`. Later, the saved data can be restored using `read_class_mlp`. In order to speed up the classification itself, you can use a different classification method like Euclidean classification (see `examples\quick_guide\hdevelop\color_pieces_euclid.dev`).

3.9.4.4 Other Examples

HDevelop

- `examples\hdevelop\Applications\Aerial\forest.dev`
Extraction of trees and meadows from forest
→ description [here in the Quick Guide](#) on page 204
- `examples\hdevelop\Applications\FA\ic.dev`
Extracts resistors, capacitors and ICs from board using color information
→ description [here in the Quick Guide](#) on page 215
- `examples\hdevelop\Applications\Monitoring\movement_col.dev`
Extracts moving objects and within them scans for specific color
- `examples\hdevelop\Applications\OCR\ocrcolor.dev`
Segmenting and reading numbers using color information
→ description [here in the Quick Guide](#) on page 234
- `examples\hdevelop\Applications\OCR\ocrcolort.dev`
Segmenting numbers using color information and training the OCR
- `examples\hdevelop\Filter\Color\cfa_to_rgb.dev`
Converting a Bayer image (color filter array) into an RGB image

- [examples\hdevelop\Filter\Edges\edges_color.dev](#)
Extracting edges using color information
→ description [here in the Quick Guide](#) on page 205
- [examples\hdevelop\Filter\Edges\edges_color_sub_pix.dev](#)
Extracting edges with sub-pixel precision using color information
- [examples\hdevelop\Filter\Lines\lines_color.dev](#)
Extracting lines using color information
→ description [here in the Quick Guide](#) on page 191
- [examples\hdevelop\Filter\Misc\symmetry.dev](#)
Analyzing symmetry in horizontal direction
- [examples\hdevelop\Manuals\HDevelop\ic.dev](#)
Combining different segmentation methods
→ description in the [HDevelop User's Manual](#) on page 144
- [examples\hdevelop\Segmentation\Classification\class_2dim_sup.dev](#)
Segmenting an image using two-dimensional pixel classification
- [examples\hdevelop\Segmentation\Classification\class_2dim_unsup.dev](#)
Segmenting two images by clustering
- [examples\hdevelop\Segmentation\Classification\class_ndim_box.dev](#)
Classifying pixels using hyper-cuboids
- [examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev](#)
Classifying pixels using hyper-spheres
- [examples\hdevelop\Segmentation\Classification\classify_image_class_mlp.dev](#)
Segmenting an RGB image with an MLP classifier
- [examples\hdevelop\Segmentation\Regiongrowing\regiongrowing_n.dev](#)
Regiongrowing for multi-channel images
- [examples\hdevelop\Segmentation\Topography\pouring.dev](#)
Segmenting an image by "pouring water" over it
- [examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev](#)
Rectifies image of stadium to simulate overhead view
- [examples\quick_guide\hdevelop\color_pieces_euclid.dev](#)
Completeness check of game color pieces using Euclidean classification

C

- [examples\c\example_multithreaded1.c](#)
Using multiple threads with Parallel HALCON

3.9.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Demosaick Bayer Pattern

Standard:

[cfa_to_rgb](#)

Decompose Channels

Standard:

[decompose3](#), [access_channel](#)

Transform Color Space

Standard:

[trans_from_rgb](#), [trans_to_rgb](#)

The following color spaces are supported:

- argyb
- cielab
- ciexyz
- hls
- hsi
- hsv
- ili2i3
- ihs
- rgb
- yiq
- yuv

Train Colors

Standard:

`histo_2dim`, `learn_ndim_norm`, `learn_ndim_box`

Advanced:

`train_class_mlp`

Process Image (Channels)

Standard:

`smooth_image`, `mean_image`, `median_image`

Advanced:

`lines_color`, `edges_color`, `edges_color_sub_pix`

Further operators can be found in the [operator list for the method Blob Analysis](#) (see section 3.3.5 on page 49).

Classify Colors

Standard:

`class_2dim_sup`, `class_2dim_unsup`, `class_ndim_norm`, `class_ndim_box`

Advanced:

`classify_image_class_mlp`

Compose Channels

Standard:

`compose3`, `append_channel`

Visualize Results

Standard:

`disp_color`, `disp_obj`

Further operators can be found in the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.9.6 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is also useful for color processing. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the segmentation is performed can be restricted, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

Speed Up

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

- Of the color processing approaches discussed in this section, the simplest is also the fastest (decompose color image and apply blob analysis). If you want to do color classification, you should consider using [class_ndim_norm](#) or [class_2dim_sup](#) for maximum performance.
- Regions of interest are the standard method to increase the speed by processing only those areas where objects need to be inspected. This can be done using pre-defined regions but also by an online generation of the regions of interest that depend on other objects found in the image.
- By default, HALCON performs some data consistency checks. These can be switched off using [set_check](#).
- By default, HALCON initializes new images. Using [set_system](#) with the parameter "init_new_image", this behavior can be changed.

3.9.7 Advanced Topics

Color Edge Extraction

Similar to the operator [edges_image](#) for gray value images (see [Edge Extraction \(pixel-precise\)](#) on page 64), the operator [edges_color](#) can be applied to find pixel-precise edges in color images. For a subpixel-precise edge extraction in color images, the operator [edges_color_sub_pix](#) is provided. It corresponds to the gray value based operator [edges_sub_pix](#) (see [Edge Extraction \(subpixel-precise\)](#) on page 74). Processing the detected color edges is the same as for gray value images (see [Contour Processing](#) on page 84).

Color Lines Extraction

Similar to color edge extraction, HALCON supports the detection of lines in color images using [lines_color](#). For processing the detected lines, please refer to [Edge Extraction \(subpixel-precise\)](#) on page 74 and [Contour Processing](#) on page 84.

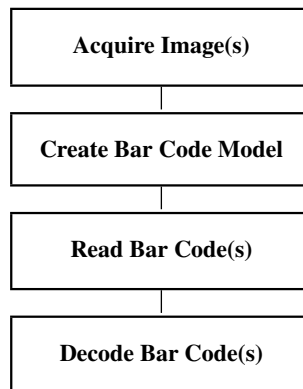
3.10 1D Bar Code

The idea of bar code reading is quite easy. After specifying the type of the desired bar code, the operator for finding the bar code is executed. The result of this is a region that contains the bar code and a tuple that contains the measured element widths. These element widths are the input for the decoding step, which returns the content of the bar code plus the parity (if it is supported).

The advantage of the HALCON bar code reader is its ease of use. No advanced experience in programming or image processing is required. Only a few operators in a clear and simple order are applied. Furthermore, the bar code reader is very powerful and flexible. An example for this is the ability to read codes in any orientation or to decode codes where parts are missing.

3.10.1 Basic Concept

Bar code reading consists mainly of four steps:



Acquire Image(s)

For the online part, i.e., during reading only, images must be acquired.

For detailed information see the [description of this method](#) on page 22.

Create Bar Code Model

The initial part is the creation of a bar code description. This description provides the finder and the reader with all necessary information about the structure of the bar code. For standard codes only the name needs to be provided, and HALCON will supply all desired parameters automatically. For codes that are not directly supported by HALCON, the parameters needed for the search and the extraction of the element widths must be passed by hand.

Read Bar Code(s)

The easiest way to find a bar code is to use the operator `find_1d_bar_code`. It locates the bar code region within the image and returns this region, the bar code orientation, and the element widths.

Decode Bar Code(s)

The final step is the decoding of the element widths. For this, you use the operator `decode_1d_bar_code`. Only bar codes that can be passed to `gen_1d_bar_code_descr` can be decoded. The decoding returns the internal code for each symbol (an index), the corresponding symbol as a string, and the parity. Please note that the parity is optional for some bar codes and can also be interpreted as an extra symbol. In this case, the parity parameter has no meaning.

A First Example

As an example for this basic concept, here is a very simple program, that finds and decodes the EAN 13 bar code depicted in [figure 3.30](#).



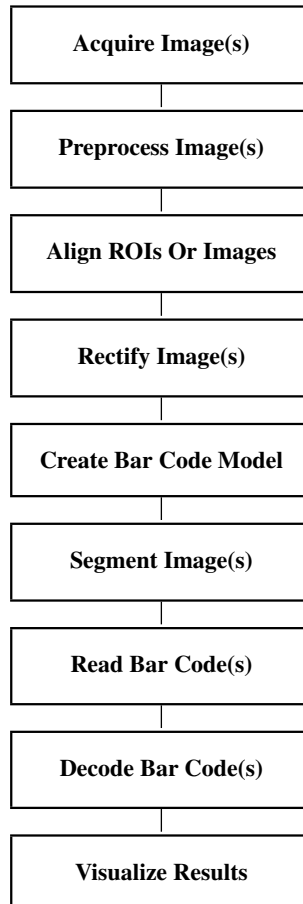
Figure 3.30: Reading a bar code.

First, a test image is acquired from file. Then, the bar code description is generated using `gen_1d_bar_code_descr`. As parameters, the bar code name and the code length are specified. Using `find_1d_bar_code`, HALCON searches for the bar code in the whole image, using all possible orientations. As a result, the bar code region and the element widths are returned. These widths together with the bar code description are input for `decode_1d_bar_code`, which performs the decoding.

```
read_image (image, 'barcode/ean13/ean1301')
gen_1d_bar_code_descr ('EAN 13', 13, 13, BarCodeDescr)
find_1d_bar_code (image, CodeRegion, BarCodeDescr, [], [], BarcodeFound,
                 BarCode, Orientation)
decode_1d_bar_code (BarCode, BarCodeDescr, Characters, Reference, IsCorrect)
```

3.10.2 Extended Concept

In some cases, bar code reading can be more advanced than in the example above. Reasons for this are, e.g., complex illumination conditions. Furthermore, preprocessing like rectification or visualization of results can be necessary.



Preprocess Image(s)

HALCON expects bar codes to be printed dark on a bright background. To read bright bar codes on a dark background you must first invert the image using the operator [invert_image](#).

Align ROIs Or Images

In general, HALCON can read bar codes of any orientation. If the orientation is known, reading becomes more robust. One way to determine the orientation is to use alignment.

How to perform alignment using shape-based matching is described in the Application Note on Shape-Based Matching in [section 4.3.4](#) on page 36.

Rectify Image(s)

Both the finding and reading operators assume that the bar code elements are parallel and that the module width does not vary within the bar code. If the bar code is printed on a cylindrical surface or the camera is not mounted perpendicular to the surface, it might therefore be necessary to rectify the image before applying the bar code reader.

How to perform the rectification for a bar code that is printed radially on a CD is shown in the description of the example program [circular_barcode.dev](#). Detailed information about compensating distortions caused, e.g., by non-perpendicularly mounted cameras, can be found in the Application Note on 3D Machine Vision in [section 3.3](#) on page 49.

Create Bar Code Model

The operator needed to generate a bar code description is called [gen_1d_bar_code_descr](#). It expects the name of the code to be found and the minimum and maximum number of encoded symbols as input and returns a description of the code. The two parameters specifying the code length are not critical, because they are only used as a hint for the finding process.

If a bar code is searched for that is not supported by HALCON, the find operator can still be used for this purpose. To give the system the necessary information about the structure of the code, [gen_1d_bar_code_descr_gen](#) is used. Here, more basic information like the start and stop symbols or the maximum number of modules must be specified. Please note that this information is not sufficient for the decoding, i.e., you cannot use [decode_1d_bar_code](#) but must decode the results of [find_1d_bar_code](#) manually.

Segment Image(s)

With [find_1d_bar_code](#), HALCON provides an operator that locates the bar code regions without programming effort. In some applications, the predefined segmentation of [find_1d_bar_code](#) might fail. In this case, we recommend to use HALCON's powerful segmentation and region processing methods to (roughly) locate the bar code with an application-specific segmentation.

After such a segmentation, the operator [get_1d_bar_code](#) is used to extract the element widths as preparation for the decoding. If the segmentation is not extracting the bar code closely enough, but contains the bar code, [find_1d_bar_code](#) can be used to improve it using the segmentation result as region of interest.

For detailed information see the [description of this method](#) on page 39.

Read Bar Code(s)

If multiple bar codes should be found within one image, the standard operator [find_1d_bar_code](#) can no longer be used. In this case, [find_1d_bar_code_region](#) provides the appropriate functionality. It will return multiple regions, one for each bar code. Here, we must consider two restrictions: First, all bar codes must be of the same type and have the same orientation. If multiple codes and/or orientations are needed, you must call [find_1d_bar_code_region](#) multiple times, possibly with an adapted region of interest.

As a preparation for the decoding, you must extract the element widths of each region returned by [find_1d_bar_code_region](#). For this, you first apply [select_obj](#) and [reduce_domain](#) and then call [get_1d_bar_code](#).

Visualize Results

Finally, you might want to display the images, the bar code regions, and the decoded content.

For detailed information see the [description of this method](#) on page 173.

3.10.3 Industries

3.10.3.1 Printing

Bar codes are widely used in the printing industry, especially the EAN 13 with add-on used for books, magazines and newspapers. For a corresponding example program see [EAN13AddOn5.dev](#) on page 208.

3.10.3.2 Electric Components And Equipment

On packaging, multiple bar codes are often used to identify information needed to describe the content, the addressee, or other information. For a corresponding example program see [multiple.dev](#) on page 222.

3.10.4 Programming Examples

This section gives a brief introduction on the programming of the bar code reader.

3.10.4.1 Reading a Bar Code on a CD

Example: [examples\hdevelop\Applications\Barcode\circular_barcode.dev](#)

Figure 3.31 shows an image of a CD, on which a bar code is printed radially. The task is to read this circular bar code. Because the bar code reader cannot read this kind of print directly, the image first must be transformed such that the elements of the bar code are parallel.

The first step is to segment the dark ring on which the bar code is printed. This is performed using [bin_threshold](#) followed by [connection](#). This segmentation returns all dark areas in the image, including the dark ring. To select the ring, [select_shape](#) is used with corresponding values for the extent.

```
bin_threshold (Image, Region)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, Ring, ['width', 'height'], 'and', [550, 550],
          [750, 750])
```

After that, the parameters of the outer and the inner circle are determined. The outer circle can be determined directly using [shape_trans](#). The inner circle is more complicated to extract. Here, it is calculated by creating the complement region of the ring with appropriate dimensions and then selecting its inner part. With [smallest_circle](#), the parameters of the inner and outer circle are determined.

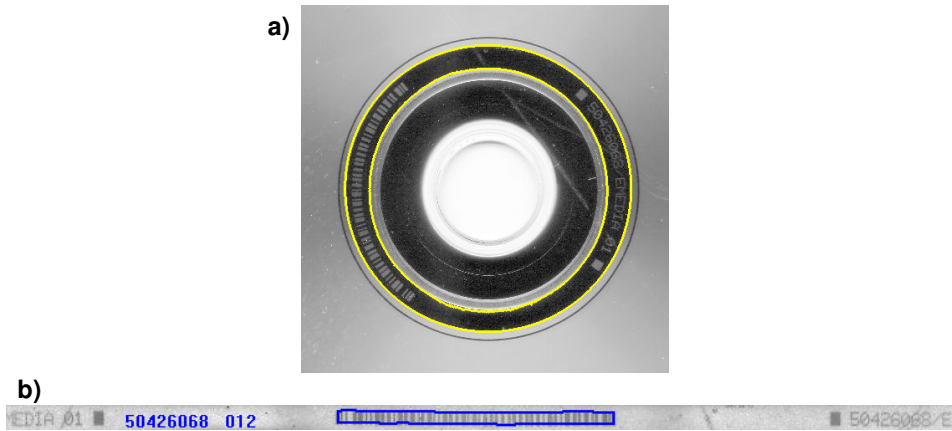


Figure 3.31: (a) Original image with segmented ring; (b) rectified ring with decoded bar code.

```

shape_trans (Ring, OuterCircle, 'outer_circle')
complement (Ring, RegionComplement)
connection (RegionComplement, ConnectedRegions)
select_shape (ConnectedRegions, InnerCircle, ['width','height'], 'and',
    [450,450], [650,650])
smallest_circle (Ring, Row, Column, OuterRadius)
smallest_circle (InnerCircle, InnerRow, InnerColumn, InnerRadius)

```

The parameters for inner and outer circle are the input for the polar transform ([polar_trans_image_ext](#)), which transforms the image inside the ring into a rectangular area. Then, the image is inverted to obtain dark bar code elements on a bright background (see [figure 3.31b](#)).

```

WidthPolar := 1440
HeightPolar := round(OuterRadius-InnerRadius-10)
polar_trans_image_ext (Image, PolarTransImage, Row, Column, rad(360), 0,
    OuterRadius-5, InnerRadius+5, WidthPolar, HeightPolar, 'bilinear')
invert_image (PolarTransImage, ImageInvert)

```

Before reading the bar code, a description is generated with [gen_1d_bar_code_descr](#) by specifying the type, here Code 128. For the actual reading, only two more operators are needed: [find_1d_bar_code](#) and [decode_1d_bar_code](#). The first one locates the bar code region and extracts the widths of the elements. The second one decodes these values.

```

gen_1d_bar_code_descr ('Code 128', 6, 20, BarCodeDescr)
find_1d_bar_code (ImageInvert, CodeRegion, BarCodeDescr,
    ['amplitude_sobel', 'min_size_element', 'dilation_factor',
    'sigma_project'], [30, 5, 2, 0.8], BarcodeFound, BarCodeElements,
    Orientation)
decode_1d_bar_code (BarCodeElements, BarCodeDescr, Characters, Reference,
    IsCorrect)

```

3.10.4.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\grid_rectification_ruled_surface.dev](#)
Rectify an arbitrarily distorted image using a regular grid
→ description in the [Application Note on 3D Machine Vision](#) on page 127
- [examples\hdevelop\Applications\Barcode\25Industry.dev](#)
Reading a bar code of type 2/5 industrial
- [examples\hdevelop\Applications\Barcode\25Interleaved.dev](#)
Reading a bar code of type 2/5 interleaved
- [examples\hdevelop\Applications\Barcode\bar_err.dev](#)
Read a partially covered bar code of type EAN13
- [examples\hdevelop\Applications\Barcode\Codabar.dev](#)
Reading a bar code of type Codabar
- [examples\hdevelop\Applications\Barcode\Code128.dev](#)
Reading a bar code of type Code 128
- [examples\hdevelop\Applications\Barcode\Code39.dev](#)
Reading a bar code of type Code 39
- [examples\hdevelop\Applications\Barcode\defect_barcode.dev](#)
Shows the bar code reader's ability to read defect bar codes
- [examples\hdevelop\Applications\Barcode\EAN13.dev](#)
Reading a bar code of type EAN13
- [examples\hdevelop\Applications\Barcode\EAN13AddOn5.dev](#)
Reading a bar code of type EAN13 Add-On 5
→ description [here in the Quick Guide](#) on page 208
- [examples\hdevelop\Applications\Barcode\EAN8.dev](#)
Reading a bar code of type EAN 8
- [examples\hdevelop\Applications\Barcode\element_width.dev](#)
Explains the requirements for reading bar codes with very narrow elements
- [examples\hdevelop\Applications\Barcode\multiple.dev](#)
Reading multiple bar codes of type Code 39
→ description [here in the Quick Guide](#) on page 222
- [examples\hdevelop\Tools\Grid-Rectification\grid_rectification.dev](#)
Rectifying an arbitrarily distorted image using a regular grid

C++

- [examples\cpp\source\ean13.cpp](#)
Reading a 1d bar code

Visual Basic

- `examples\vb\Online\Barcode\barcode.vbp`
Reading bar codes in a live image

3.10.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Preprocess Image(s)

Standard:

[invert_image](#)

Align ROIs Or Images

Operators for aligning ROIs or images are described in the [Application Note on Shape-Based Matching](#).

Rectify Image(s)

Operators for rectifying images are described in the [Application Note on 3D Machine Vision](#).

Create Bar Code Model

Standard:

[gen_1d_bar_code_descr](#)

Advanced:

[gen_1d_bar_code_descr_gen](#)

Segment Image(s)

Please refer to the [detailed operator list for the step Segment Image\(s\)](#) on page 50.

Read Bar Code(s)

Standard:

[find_1d_bar_code](#)

Advanced:

[find_1d_bar_code_region](#), [get_1d_bar_code](#)

Decode Bar Code(s)

Standard:

[decode_1d_bar_code](#)

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.10.6 Relation to Other Methods

3.10.6.1 Methods that are Useful for 1D Bar Code

1D Measuring (see [description](#) on page 53)

The measure tool can be used to extract the widths of the elements of the bar code using the operator [measure_pos](#). The output of the parameter Distance is almost what is needed for the decoding step. What remains to be done is to negate the distance values for edges with a dark-to-light transition. Because bar codes start with a light to dark transition, this means that all even values, i.e., the second, fourth, etc., must be negated. This transformed tuple can then be used as input for [decode_1d_bar_code](#).

3.10.6.2 Alternatives to 1D Bar Code

OCR (see [description](#) on page 145)

With some bar codes, e.g., the EAN 13, the content of the bar code is printed in plain text below the elements. Here, OCR can be used, e.g., to check the consistency of the reading process.

3.10.7 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is useful for bar code reading. With domains, the processing can be focused to a certain area in the image and thus sped up. The smaller the region in which the bar code is searched for, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

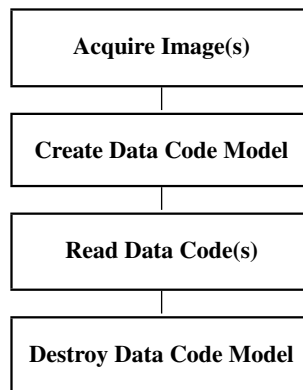
3.11 2D Data Code

Data codes are a special kind of two-dimensional patterns that encode text and numbers. HALCON is able to read the three most popular data codes: Data Matrix ECC 200, QR Code, and PDF417. These codes consist of a so-called finder pattern, which is used to locate the pattern and get basic information about the geometric properties. The code itself contains multiple dots or small squares. Because of the special design of the codes, they can be decoded even if some parts are disturbed.

The advantage of the HALCON data code reader is its ease of use. No advanced experience in programming or image processing is required. Only a few operators in a clear and simple order need to be applied. Furthermore, the data code reader is very powerful and flexible. Examples for this are its ability to read codes in many print styles and the possibility to automatically learn optimal parameters.

3.11.1 Basic Concept

Data code reading consists mainly of four steps:



Acquire Image(s)

For the online part, i.e., during reading, images are acquired.

For detailed information see the [description of this method](#) on page 22.

Create Data Code Model

First, you create a data code model with the operator `create_data_code_2d_model`. This model provides the reader with all necessary information about the structure of the code. For normal printed codes only the name needs to be provided and HALCON will select suitable default parameters. For special cases, you can modify the model by passing specific parameters.

Read Data Code(s)

To read a data code, just one operator is needed: `find_data_code_2d`. It will locate one or more data codes and decode the content.

Destroy Data Code Model

When you no longer need the data code model, you destroy it with the operator `clear_data_code_2d_model`.

A First Example

As an example for this basic concept, here a very simple program, which reads the data code on the chip depicted in [figure 3.32](#), is discussed.



Figure 3.32: Reading a data code.

After reading an image from file, the data code model is generated by calling `create_data_code_2d_model`. As the only required parameter value, the code name 'ECC200' is specified.

```
read_image (Image, 'datacode/ecc200/ecc200_cpu_003')
create_data_code_2d_model ('ECC200', [], [], DataCodeHandle)
```

Then, the data code is read with the operator `find_data_code_2d`.

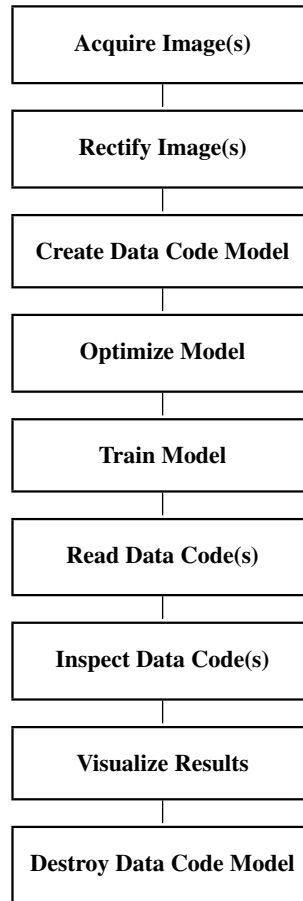
```
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [],
                  ResultHandles, DecodedDataStrings)
```

At the end of the program, the data code model is destroyed using `clear_data_code_2d_model`.

```
clear_data_code_2d_model (DataCodeHandle)
```

3.11.2 Extended Concept

In some cases, data code reading can be more advanced than in the example above. Reasons for this are, e.g., parameter optimization for improved execution time. Furthermore, preprocessing like rectification or the visualization of results might be required. The following sections give a brief overview. More detailed information can be found in the [Application Note on 2D Data Codes](#).



Acquire Image(s)

Optionally, additional images can be acquired for parameter optimization (see the description of the step [Optimize Model](#)).

For detailed information see the [description of this method](#) on page 22.

Rectify Image(s)

HALCON's data code reader is robust against image distortions up to a certain limit. But if the data code is printed on a cylindrical surface or if the camera is tilted relative to the surface, it might be necessary

to rectify the image before applying the data code reader.

Detailed information about rectifying images can be found in the Application Note on 3D Machine Vision in [section 3.3](#) on page 49.

Create Data Code Model

The operator `create_data_code_2d_model` expects the name of the desired code and optionally parameters to specify the geometry and radiometry as input. By default, a parameter set is used that is suitable for data codes that fulfill the following requirements:

- The code must be printed dark on light,
- the contrast must be bigger than 30,
- the size of symbol is smaller or equal to 48x48 modules,
- the width and the height of the modules are between 6 and 20 pixels, and
- there is no or only a small gap in between neighboring modules.

This parameter set is also used if you specify the value `'standard_recognition'` for the parameter `GenParamValues`. In contrast, if you specify the value `'enhanced_recognition'`, a parameter set is used that detects codes that do not follow the rules given above. However, using this parameter set possibly results in a longer processing time.

Optimize Model

Using the default parameters, the data code reader is able to read a wide range of codes. For non-standard codes the parameters can be adapted accordingly. For this, the operator `set_data_code_2d_param` is used.

The easiest way is to use the parameter value `'enhanced_recognition'`, which uses a model that is able to find a very wide range of print styles. An alternative is to specify parameter values separately to adapt the model to the conditions of the used print style.

If a data code symbol is not detected although it is well visible in the image, check whether the symbol's appearance complies with the model. In particular, have a look at the polarity (`'polarity'`: dark-on-light or light-on-dark), the module size (`'module_size'` and `'module_shape'`) and the minimum contrast (`'contrast_min'`). In addition, the parameters `'module_gap'` (allowed gap between modules), `'symbol_size'`, and `'slant_max'` (angle variance of the legs of the finder pattern) should be checked. The current settings of these values can be queried by the operator `get_data_code_2d_param`.

All possible parameter values can be checked in the Reference Manual. Besides this, they can also be queried with the operator `query_data_code_2d_params`.

As an alternative, you can train the model (see below).

Train Model

Instead of modifying the model parameters manually as described above, you can also let HALCON train the model automatically using the operator `find_data_code_2d`. All you need to do is to call

this operator with the parameter values 'train' and 'all'. Then, HALCON will search for the best parameters needed to extract the given code. It is recommended to apply this to multiple example images to ensure that all variations are covered.

As an alternative, you can execute the finder with normal parameters and request the features of the found symbols with `get_data_code_2d_results`. These values can then be used to change the model with `set_data_code_2d_param`.

Read Data Code(s)

The operator `find_data_code_2d` returns for every successfully decoded symbol the surrounding XLD contour in `SymbolXLDs`, a handle to a result structure, which contains additional information about the symbol as well as about the search and decoding process (`ResultHandles`), and the string that is encoded in the symbol (`DecodedDataStrings`). With the result handles and the operators `get_data_code_2d_results` and `get_data_code_2d_objects`, additional data about the extraction process can be accessed.

Inspect Data Code(s)

Using the handles of the successfully decoded symbols returned by `find_data_code_2d`, you can request additional information about the symbol and the finding process using the operators `get_data_code_2d_results` and `get_data_code_2d_objects`. This is useful both for process analysis and for displaying.

In addition, information about rejected candidates can also be queried by requesting the corresponding handles with `get_data_code_2d_results` using, e.g., the parameter values 'all_undecoded' and 'handle'.

The operator `get_data_code_2d_results` gives access to several alphanumerical results that were calculated while searching and reading the data code symbols. Besides basic information like the dimensions of the code, its polarity, or the found contrast, also the raw data can be accessed.

The operator `get_data_code_2d_objects` gives access to iconic objects that were created while searching and reading the data code symbols. Possible return values are the surrounding contours or the regions representing the foreground or background modules.

Visualize Results

Finally, you might want to display the images, the data code regions, and the decoded content.

For detailed information see the [description of this method](#) on page 173.

3.11.3 Industries

3.11.3.1 Semiconductors

Semiconductor industry is the typical application area for data code reading. Especially larger electronic components like chips have a data code printed on them for identification purposes, used, e.g., during the mounting process. For a corresponding example program see `ecc200_optimized.dev` on page 223.

3.11.4 Programming Examples

This section gives a brief introduction to the programming of the data code reader.

3.11.4.1 Training a Data Code Model

Example: `examples\quick_guide\hdevelop\ecc200_training_simple.dev`

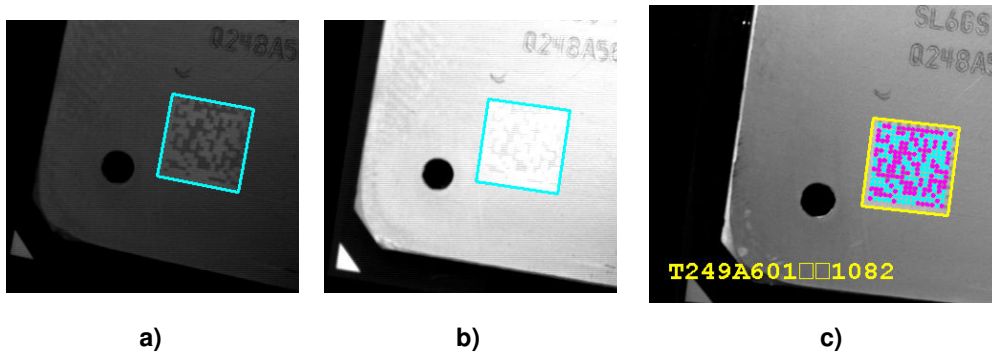


Figure 3.33: (a) Dark training image; (b) bright training image; (c) read code with extracted modules.

In this example we show how easy it is to train a data code model, here to allow changes in the illumination of the images. To prepare the reading of the data codes, three major steps are performed: First, the connection to an image sequence is established by calling `open_framegrabber`.

```
SequenceName := 'datacode/ecc200/ecc200_cpu_light.seq'
open_framegrabber ('File', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1,
  'default', SequenceName, 'default', -1, -1, FGHandle)
```

Then, a model for an ECC 200 is created with `create_data_code_2d_model`.

```
create_data_code_2d_model ('ECC200', [], [], DataCodeHandle)
```

To get the optimal parameters for finding the data code, two sample images are loaded and passed to `find_data_code_2d` with the parameter value 'train'.

```
grab_image (Image, FGHandle)
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all',
  ResultHandles, DecodedDataStrings)
grab_image (Image, FGHandle)
find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, 'train', 'all',
  ResultHandles, DecodedDataStrings)
```

As a preparation for the detection loop, the settings for the find operator are changed with `set_data_code_2d_param` to enable the saving of visual data.

```
set_data_code_2d_param (DataCodeHandle, 'persistence', 1)
```

Inside the while-loop, images are grabbed and `find_data_code_2d` is applied to read the code from the image. After that, iconic results and the read code are visualized. The regions with the foreground and background modules are requested with `get_data_code_2d_objects`.

```
while (1)
  grab_image (Image, FGHandle)
  dev_display (Image)
  find_data_code_2d (Image, SymbolXLDs, DataCodeHandle, [], [],
    ResultHandles, DecodedDataStrings)
  dev_set_color ('yellow')
  dev_display (SymbolXLDs)
  set_tposition (WindowHandle, 430, 80)
  write_string (WindowHandle, DecodedDataStrings)
  get_data_code_2d_objects (Foreground, DataCodeHandle, ResultHandles[0],
    'module_1_rois')
  get_data_code_2d_objects (Background, DataCodeHandle, ResultHandles[0],
    'module_0_rois')
  dev_set_color ('cyan')
  dev_display (Foreground)
  dev_set_color ('magenta')
  dev_display (Background)
endwhile
```

At the end of the program the model for the data code reader is destroyed and the connection to the image sequence is closed.

```
clear_data_code_2d_model (DataCodeHandle)
close_framegrabber (FGHandle)
```

3.11.4.2 Other Examples

HDevelop

- [examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_arbitrary_distortions.dev](#)
Demonstrate distorted symbols of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 37
- [examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_data_access.dev](#)
Access intermediate results when trying to read symbols of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 26
- [examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_enlarge_modules.dev](#)
Increase the resolution of a symbol of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 21

- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_first_example.dev`
Reads symbols of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 5
- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_global_settings.dev`
Switch between standard and enhanced mode for reading symbols of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 8
- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_manual_settings.dev`
Sets parameters manually for reading symbols of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 11
- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_minimize_module_gaps.dev`
Minimize large gaps from a symbol of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 23
- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_minimize_noise.dev`
Minimize noise from a symbol of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 23
- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_rectify_symbol.dev`
Rectify a slanted symbol of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 21
- `examples\application_guide\2d_data_codes\hdevelop\2d_data_codes_training.dev`
Train a 2d data code model for symbols of type ECC200
→ description in the [Application Note on 2D Data Codes](#) on page 10
- `examples\application_guide\2d_data_codes\hdevelop\read_2d_data_code_model.dev`
Read a 2d data code model from file
→ description in the [Application Note on 2D Data Codes](#) on page 19
- `examples\application_guide\2d_data_codes\hdevelop\write_2d_data_code_model.dev`
Train a 2d data code model and write it into a file
→ description in the [Application Note on 2D Data Codes](#) on page 19
- `examples\hdevelop\Applications\Datacode\ecc200_optimized.dev`
Reading 2d data codes of type ECC200
→ description [here in the Quick Guide](#) on page 223
- `examples\hdevelop\Applications\Datacode\ecc200_simple.dev`
Reading 2d data codes of type ECC200

- [examples\hdevelop\Applications\Datacode\pdf417_optimized.dev](#)
Optimizing parameters for reading of 2d data codes of type PDF417
- [examples\hdevelop\Applications\Datacode\pdf417_simple.dev](#)
Reading 2d data codes of type PDF417
- [examples\hdevelop\Applications\Datacode\qrcode_optimized.dev](#)
Reading 2d data codes of type QR code
- [examples\hdevelop\Applications\Datacode\qrcode_simple.dev](#)
Reading 2d data codes of type QR code
- [examples\hdevelop\Tools\Datacode\ecc200_default_settings.dev](#)
Reading a 2d data code of type ECC200
- [examples\hdevelop\Tools\Datacode\ecc200_optimized_settings.dev](#)
Optimizing parameters for reading a 2d data code of type ECC200
- [examples\hdevelop\Tools\Datacode\ecc200_read_model.dev](#)
Loading previously stored parameters for 2d data codes of type ECC200
- [examples\hdevelop\Tools\Datacode\ecc200_training.dev](#)
Training parameters for reading a 2d data code of type ECC200
- [examples\hdevelop\Tools\Datacode\ecc200_write_model.dev](#)
Storing parameters for 2d data codes of type ECC200
- [examples\hdevelop\Tools\Datacode\pdf417_default_settings.dev](#)
Reading various 2d data codes of type PDF417
- [examples\hdevelop\Tools\Datacode\pdf417_optimized_settings.dev](#)
Optimizing parameters for reading various 2d data codes of type PDF417
- [examples\hdevelop\Tools\Datacode\qrcode_default_settings.dev](#)
Reading a 2d data code of type QR Code
- [examples\hdevelop\Tools\Datacode\qrcode_optimized_settings.dev](#)
Optimizing parameters for reading a 2d data code of type QR Code

C++

- [examples\cpp\source\ecc200.cpp](#)
Reading 2d data codes of type ECC200

3.11.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Rectify Image(s)

Operators for rectifying images are described in the [Application Note on 3D Machine Vision](#).

Create Data Code Model

Standard:

```
create_data_code_2d_model
```

Optimize Model

Standard:

```
set_data_code_2d_param, get_data_code_2d_param
```

Advanced:

```
query_data_code_2d_params
```

Train Model

Standard:

```
find_data_code_2d
```

Advanced:

```
get_data_code_2d_results, set_data_code_2d_param
```

Read Data Code(s)

Standard:

```
find_data_code_2d
```

Inspect Data Code(s)

Standard:

```
get_data_code_2d_results, get_data_code_2d_objects
```

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

Destroy Data Code Model

Standard:

```
clear_data_code_2d_model
```

3.11.6 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is useful for data code reading. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the data code region is searched for can be restricted, the faster and more robust the search. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

3.12 OCR

Optical Character Recognition (OCR) is the technical term for reading, i.e., identifying symbols. In HALCON, OCR is defined as the task to assign an interpretation to regions of an image. These regions typically represent single characters and therefore we consider this as reading single symbols.

In an offline phase, the characters are trained by presenting several samples for each character. In the online phase, the image is segmented to extract the regions representing the characters and then the OCR reader is applied to get the interpretation for each character.

Figure 3.34 shows the principal steps. The first part is offline and consists of collecting training samples and, after that, applying the training. The online part consists of extracting the characters and then reading them.

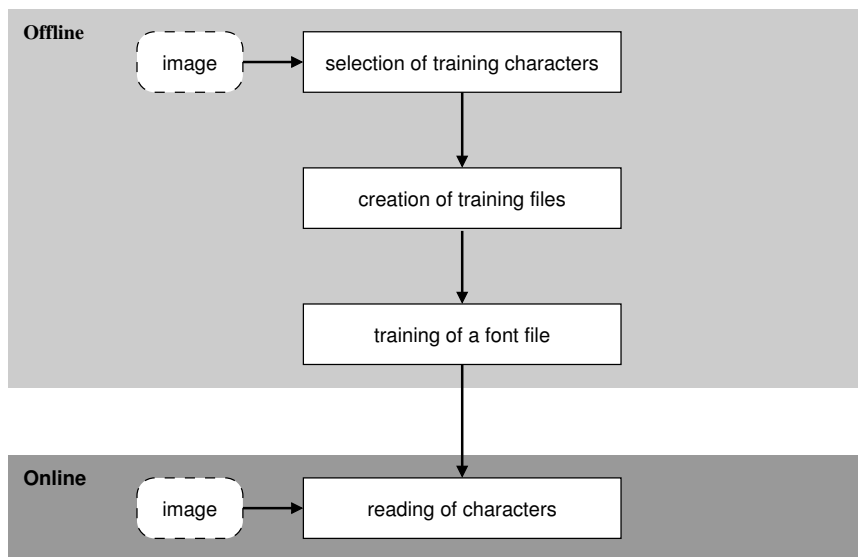


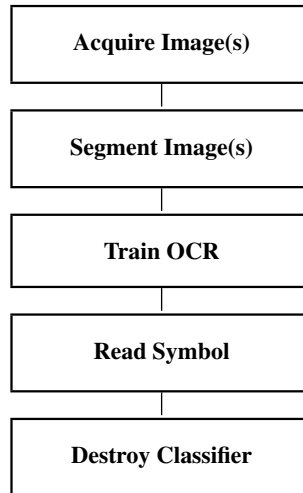
Figure 3.34: Main steps of OCR.

The advantage of OCR is the flexibility of the training, which allows to select features optimized for an application. Furthermore, the used classifier is based on the latest neural network technology, providing best possible performance.

As a further advantage, HALCON provides you with a set of pretrained fonts, which are based on a large amount of training data from various application areas. These fonts allow you to read text in documents, on pharmaceutical or industrial products, dot prints, and even handwritten numbers. Furthermore, HALCON includes pretrained fonts for OCR-A and OCR-B.

3.12.1 Basic Concept

The OCR is split into two major parts: training and reading. Each of these major parts requires additional preparation steps:



Acquire Image(s)

Both for the generation of training data and for the OCR itself images must be acquired.

For detailed information see the [description of this method](#) on page 22.

Segment Image(s)

Both for the training samples and for the online reading process, characters must be extracted from the image. This step is called segmentation. This means that the OCR operators like [do_ocr_single_class_mlp](#) do not search for the characters within a given region of interest, but expect a segmented region, which then will be classified.

If the samples for training are taken from real application images, the same segmentation method will be applied for both training and reading. If the training images are more “artificial”, a simpler method might be used to segment the training images.

Train OCR

The training consists of two important steps: First, a number of samples for each character are selected and stored in so-called training files. In the second step, these files are input for a newly created OCR classifier.

As already noted, HALCON provides pretrained fonts, which already solve many OCR applications. These fonts can be found in the subdirectory `ocr` of the folder where you installed HALCON.

Read Symbol

The only task that must be solved for the reading is the segmentation of the characters. The corresponding regions will then be input for the trained classifier. The classifier will be read from disk beforehand.

Destroy Classifier

When you no longer need the classifier, you destroy it with the operator `clear_ocr_class_mlp` or `close_ocr`.

3.12.2 Extended Concept

When we look more closely at the OCR, many possibilities for adaptation to specific applications become apparent. For example, we have to consider an efficient way of collecting samples as well as correct parameters for the training. In the online phase, an optimal segmentation method is required to extract all characters in a robust manner.

Align ROIs Or Images

Because the reading of characters is not invariant to rotation, it may be necessary to correct the orientation of the image. This is achieved by locating another object. Then, the part of the image containing the characters is cropped and aligned using the orientation of the found object.

How to perform alignment using shape-based matching is described in the Application Note on Shape-Based Matching in [section 4.3.4](#) on page 36.

Rectify Image(s)

Similarly to alignment, it may be necessary to rectify the image, e.g., to remove perspective distortions.

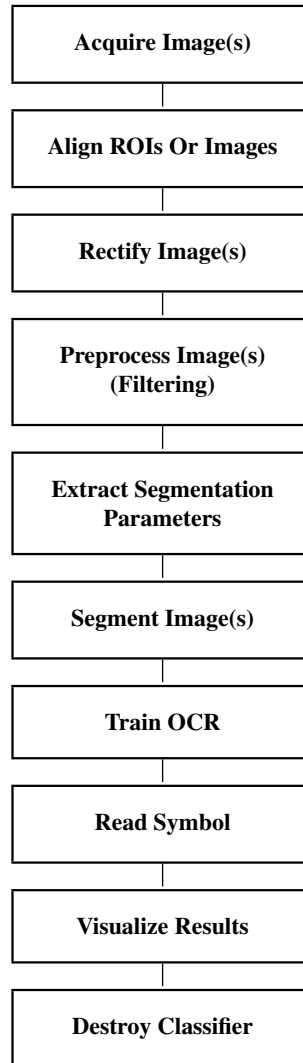
Detailed information about rectifying images can be found in the Application Note on 3D Machine Vision in [section 3.3](#) on page 49.

Preprocess Image(s) (Filtering)

Sometimes, the characters may be difficult to extract because of noise, texture, or overlaid structures. Here, operators like `mean_image` or `gauss_image` can be used to eliminate noise. `median_image` is helpful for suppressing small spots or thin lines. The operator `dots_image` is optimized to emphasize a dot-print while suppressing other structures in the image. Gray value morphology can be used to eliminate noise structures and to adapt the stroke width of characters.

Extract Segmentation Parameters

Instead of using fixed threshold values, they can be extracted dynamically for each image. This approach is identical to the one used for standard blob analysis. For more details, please refer to the [description of this step](#) on page 42.



Segment Image(s)

For the segmentation various methods can be used. The most simple method is the operator `threshold`, with one or more gray value ranges specifying the regions that belong to the foreground objects. Another very common method is `dyn_threshold`. Here, a second image is passed as a reference. With this approach a local instead of a global threshold is used for each position.

Like the previous step, segmentation is described in more detail with the method `blob analysis` on page 42.

Train OCR

[Figure 3.35](#) shows an overview on the generation of the training files: First, the characters from sample

images must be extracted using a segmentation method (see above). To each of the single characters a name must be assigned. This can be done either by typing it in, by a programmed input in the case of a well-structured image (having, e.g., multiple samples of each character in different lines), or by reading the character names from file. Then, the regions together with their names are written into training files. The most convenient operator to do this is `append_ocr_trainf`. Before applying the training, we recommend to check the correctness of the training files. This can, e.g., be achieved by using the operator `read_ocr_trainf` combined with visualization operators.

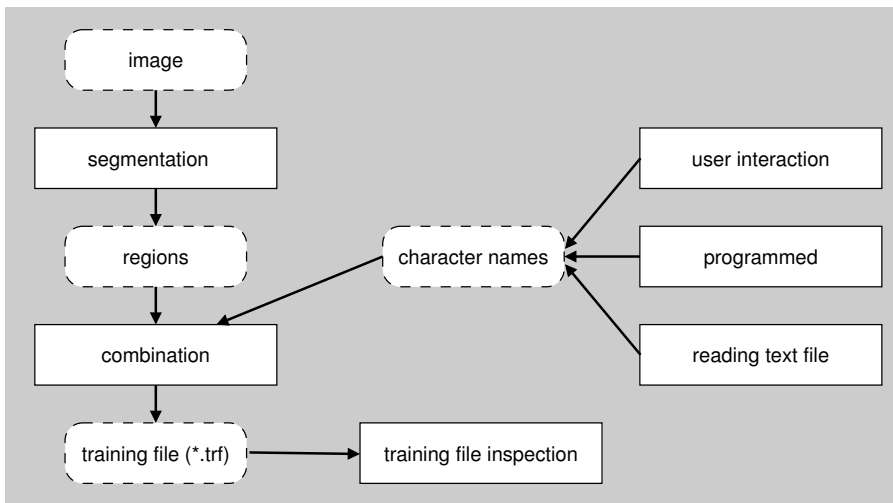


Figure 3.35: Creating training files.

The actual training is depicted in figure 3.36. First, a new classifier is created using either `create_ocr_class_box` or `create_ocr_class_mlp`. Then, the training is applied using `trainf_ocr_class_box` or `trainf_ocr_class_mlp`, respectively. After the training, you typically save the classifier to disk for later use for the reading with the operators `write_ocr` or `write_ocr_class_mlp`, respectively.

There are two different OCR tools available. The difference is very simple: The box classifier has the advantage of step-by-step training. This means that single samples can be added to an already trained classifier. This is very useful with interactive programs where an immediate improvement after showing extra samples is required. The neural network classifier (multi-layer perceptron) has better classification results but requires all samples simultaneously during the (single) training.

Read Symbol

Figure 3.37 shows an overview on the reading process. First, the characters must be extracted using an appropriate segmentation method. Here, you must use a method that returns the characters in a form similar to the ones used for training. After reading the classifier (font file) from file (`read_ocr` or `read_ocr_class_mlp`), the classifier can be used for reading.

For reading multiple operators are provided: In the easiest case, multiple characters are passed to the reading operators (`do_ocr_multi` or `do_ocr_multi_class_mlp`). Here, for each region the corresponding name and the confidence are returned. Sometimes, it can be necessary not only to obtain the

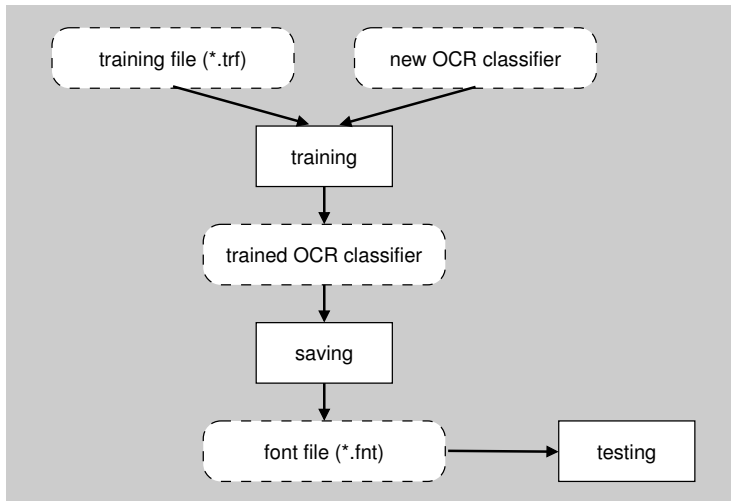


Figure 3.36: Training an OCR classifier.

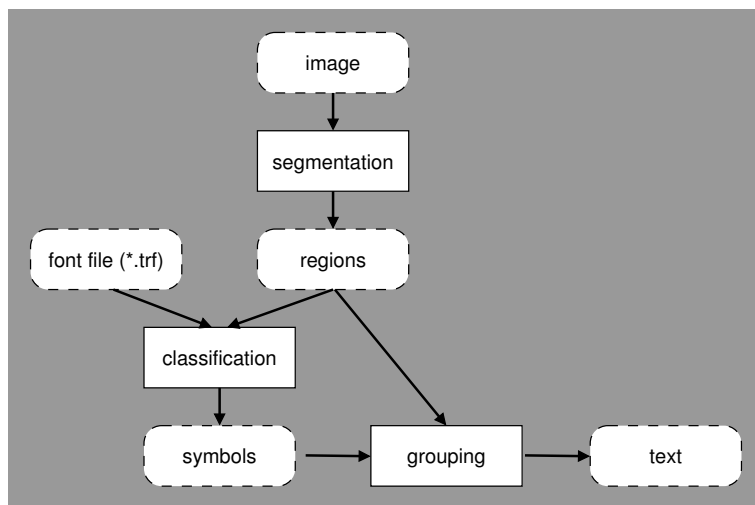


Figure 3.37: Reading characters.

characters with the highest confidence but also others with lower confidences. A zero, e.g., might easily be mistaken for the character “O”. This information is returned by the operators `do_ocr_single` and `do_ocr_single_class_mlp`.

As a final step it might be necessary to group digits to numbers or characters to words. This can be realized with the region processing operators like those described for the method `blob analysis` on page 42.

Visualize Results

Finally, you might want to display the images, the blob (regions), and the result of the reading process.

For detailed information see the [description of this method](#) on page 173.

3.12.3 Industries

3.12.3.1 Food

A typical OCR application is to check the “best before” date. For a corresponding example see the description of [bottle.dev](#) on page 193.

3.12.3.2 Machinery

A standard task in machinery is to read text that is engraved on metal surfaces. This task can be difficult because of textures and changing illumination conditions. For a corresponding example see the description of [engraved.dev](#) on page 199.

3.12.3.3 Transport And Logistics

A typical but not easy task is the reading of forms. For the corresponding example see the description of [ocrcolor.dev](#) on page 234.

3.12.4 Programming Examples

This section gives a brief introduction to using HALCON for OCR. All important steps from training file generation over training to reading are presented.

3.12.4.1 Generating a Training File

Example: [examples\quick_guide\hdevelop\gen_training_file.dev](#)

[Figure 3.38](#) shows a training image from which the characters in the fourth line are used as training samples. For this example image, the segmentation is very simple because the characters are significantly darker than the background. Therefore, [threshold](#) can be used.

The number of the line of characters that is used for training is specified by the variable `TrainingLine`. To select this line, first the operator [closing_rectangle1](#) is used to combine characters horizontally into lines. These lines are then converted to their connected components with [connection](#). Out of all lines the relevant one is selected using [select_obj](#). By using [intersection](#) with the original segmentation and the selected line as input, the characters for training are returned. These are sorted from left to right, using [sort_region](#).

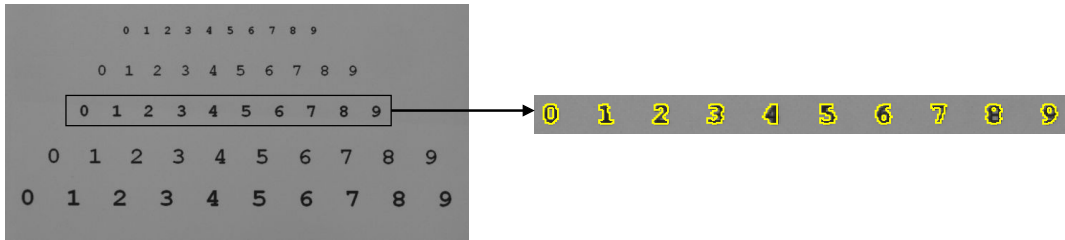


Figure 3.38: Collecting characters for a training file.

```

TrainingLine := 3
threshold (Image, Region, 0, 125)
closing_rectangle1 (Region, RegionClosing, 70, 10)
connection (RegionClosing, Lines)
Training := Lines[TrainingLine]
intersection (Training, Region, TrainingChars)
connection (TrainingChars, ConnectedRegions)
sort_region (ConnectedRegions, SortedRegions, 'first_point', 'true',
            'column')

```

Now, the characters can be stored in the training file. As a preparation step a possibly existing older training file is deleted. Within a loop over all characters the single characters are selected. The variable `Chars` contains the names of the characters as a tuple of strings. With the operator `append_ocr_trainf` the selected regions, together with the gray values and the corresponding name, are added to the training file.

```

Chars := ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
TrainFile := 'numbers.trf'
dev_set_check ('~give_error')
delete_file (TrainFile)
dev_set_check ('give_error')
for i := 1 to 10 by 1
    TrainSingle := SortedRegions[i]
    append_ocr_trainf (TrainSingle, Image, Chars[i-1], TrainFile)
endfor

```

3.12.4.2 Creating and Training an OCR Classifier

Example: [examples\quick_guide\hdevelop\simple_training.dev](#)

Having prepared the training file, the creation and training of an OCR classifier is very simple. First, the names of the training file and the final font file are determined. Typically, the same base with different extensions is used. We recommend to use “.trf” for training files and “.fnt” for font files, i.e., for OCR classifiers.

To create an OCR classifier, some parameters need to be determined. The most important one is the

list of all possible character names. This list can easily be extracted from the training file by using the operator `read_ocr_trainf_names`.

```
TrainFile := 'numbers.trf'
read_ocr_trainf_names (TrainFile, CharacterNames, CharacterCount)
```

Another important parameter is the number of nodes in the hidden layer of the neural network. In this case, it is set to 20. As a rule of thumb, this number should be in the same order as the number of different symbols. Besides these two parameters, here only default values are used for `create_ocr_class_mlp`. The training itself is applied using `trainf_ocr_class_mlp`. We recommend to simply use the default values here as well.

```
NumHidden := 20
create_ocr_class_mlp (8, 10, 'constant', 'default', CharacterNames,
    NumHidden, 'none', 1, 42, OCRHandle)
trainf_ocr_class_mlp (OCRHandle, TrainFile, 200, 1, 0.01, Error, ErrorLog)
```

Finally, the classifier is stored to disk and the memory is freed.

```
FontFile := 'numbers.fnt'
write_ocr_class_mlp (OCRHandle, FontFile)
clear_ocr_class_mlp (OCRHandle)
```

3.12.4.3 Reading Numbers

Example: `examples\quick_guide\hdevelop\simple_reading.dev`

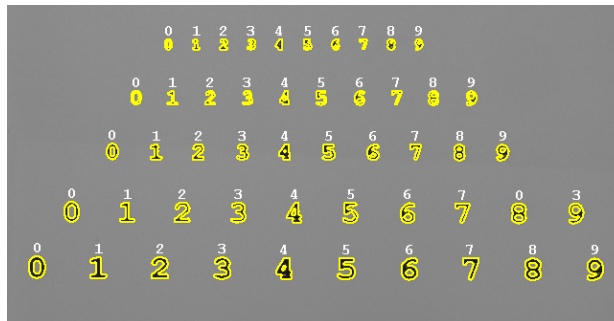


Figure 3.39: Applying the classifier.

This example uses the font trained in the previous example to read the numbers in the images depicted in [figure 3.38](#). First, the trained font is read using `read_ocr_class_mlp`.

```
FontFile := 'numbers.fnt'
read_ocr_class_mlp (FontFile, OCRHandle)
```

Then, the numbers are segmented using `threshold` and `connection`. Because the order is irrelevant here, no further processing is applied.

```
threshold (Image, Region, 0, 125)
connection (Region, Characters)
```

Finally, the numbers are read in a for-loop. The operator `do_ocr_single_class_mlp` takes the single region, the image, and the OCR handle as input. As a result the best and the second best interpretation together with the confidences are returned.

```
Number := |Characters|
for i := 1 to Number by 1
  SingleChar := Characters[i]
  do_ocr_single_class_mlp (SingleChar, Image, OCRHandle, 2, Class,
    Confidence)
endfor
```

3.12.4.4 Other Examples

HDevelop

- `examples\hdevelop\Applications\OCR\bottle.dev`
Segmenting and reading numbers on a beer bottle
→ description [here in the Quick Guide](#) on page 193
- `examples\hdevelop\Applications\OCR\bottlet.dev`
Segmenting and training numbers on a beer bottle
- `examples\hdevelop\Applications\OCR\dotprt.dev`
Segmenting a dot print
- `examples\hdevelop\Applications\OCR\engraved.dev`
Segmenting and reading characters on a metal surface
→ description [here in the Quick Guide](#) on page 199
- `examples\hdevelop\Applications\OCR\engravedt.dev`
Segmenting and training characters on a metal surface
- `examples\hdevelop\Applications\OCR\font.dev`
Segmenting and reading printed characters
- `examples\hdevelop\Applications\OCR\fontt.dev`
Segmenting and training printed characters
- `examples\hdevelop\Applications\OCR\letter.dev`
Segmenting and reading printed characters
- `examples\hdevelop\Applications\OCR\letters_mlp.dev`
Trains an MLP OCR classifier and reclassifies the training samples

- [examples\hdevelop\Applications\OCR\letttert.dev](#)
Segmenting and training printed characters
- [examples\hdevelop\Applications\OCR\ocrcolor.dev](#)
Segmenting and reading numbers using color information
→ description [here in the Quick Guide](#) on page 234
- [examples\hdevelop\Applications\OCR\ocrcolort.dev](#)
Segmenting numbers using color information and training the OCR
- [examples\hdevelop\Applications\OCR\rotchar.dev](#)
Estimating small inclinations of text lines
- [examples\hdevelop\Applications\OCR\stamp_catalogue.dev](#)
Segmenting and grouping characters on a cluttered page
- [examples\hdevelop\Filter\Points\dots_image.dev](#)
Segmenting a dot print using dots_image

C++

- [examples\cpp\source\bottle.cpp](#)
Reads numbers on a beer bottle
- [examples\cpp\source\engraved.cpp](#)
Segmenting and reading characters on a metal surface

3.12.5 Selecting Operators

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Align ROIs Or Images

Operators for aligning ROIs or images are described in the [Application Note on Shape-Based Matching](#).

Rectify Image(s)

Operators for rectifying images are described in the [Application Note on 3D Machine Vision](#).

Preprocess Image(s) (Filtering)

Standard:

[mean_image](#), [gauss_image](#), [median_image](#), [gray_opening_shape](#), [gray_closing_shape](#),
[dots_image](#), [gray_range_rect](#)

Advanced:

[gray_dilation_shape](#), [gray_erosion_shape](#), [anisotrope_diff](#)

Extract Segmentation Parameters

Please refer to the [detailed operator list for the step Extract Segmentation Parameters](#) on page 49.

Segment Image(s)

Please refer to the [detailed operator list for the step Segment Image\(s\)](#) on page 50.

Train OCR

Standard:

```
read_string, append_ocr_trainf, delete_file, read_ocr_trainf,  
create_ocr_class_box, trainf_ocr_class_box, write_ocr, create_ocr_class_mlp,  
trainf_ocr_class_mlp, write_ocr_class_mlp
```

Advanced:

```
traind_ocr_class_box, get_prep_info_ocr_class_mlp
```

Read Symbol

Standard:

```
read_ocr, do_ocr_multi, do_ocr_single, close_ocr, read_ocr_class_mlp,  
do_ocr_multi_class_mlp, do_ocr_single_class_mlp, clear_ocr_class_mlp
```

Advanced:

```
dilation_rectangle1, closing_circle, partition_rectangle, partition_dynamic
```

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

Destroy Classifier

Standard:

```
clear_ocr_class_mlp, close_ocr
```

3.12.6 Relation to Other Methods

3.12.6.1 Alternatives to OCR

(see [description](#) on page ??)

As an alternative to classical OCR, matching can be used to read single characters or more complex symbols. In this case, one model for each character must be generated. The advantage of matching is the invariance to rotation. Furthermore, it is not necessary to segment the characters prior to the classification. Therefore, the matching approach should be considered when there is no robust way to separate the characters from the background.

Classification

You can consider the OCR tool as a convenient way of using a classifier. The OCR automatically derives invariant features and passes them to the underlying classifier. If the features offered by the OCR do not fulfill the needs of your application, you can create an “extended” OCR classifier by calculating the features using normal HALCON feature extraction operators and then using them with one of the two classifiers that HALCON offers (see the chapters “Regions ▸ Features” and “Classification” in the Reference Manual).

3.12.7 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is also useful for OCR. With domains, the processing can be focused to a certain area in the image and thus sped up. The more the region in which the characters are searched can be restricted, the faster and more robust the search will be. For an overview on how to construct regions of interest and how to combine them with the image see the method [Region Of Interest](#) on page 27.

Composed Symbols

Some characters and symbols are composed of multiple sub-symbols, like an “i”, “%”, or “!”. For the OCR, these sub-symbols must be combined into a single region. This can be achieved by calling [closing_rectangle1](#) after thresholding, typically using a small width but a larger height. After calling [connection](#) to separate the characters, you use the operator [intersection](#) to get the original segmentation (input parameter 2), while preserving the correct connected components from [connection](#) (input parameter 1).

3.12.8 Advanced Topics

Line Scan Cameras

In general, line scan cameras are treated like normal area sensors. But in some cases, not single images but an “infinite” sequence of images showing objects, e.g., on a conveyor belt, must be processed. In this case, the end of one image is the beginning of the next one. This means that text or numbers, which partially lie in both images, must be combined into one object. For this purpose, HALCON provides the operator [merge_regions_line_scan](#). This operator is called after the segmentation of one image, and combines the current objects with those of previous images. For more information see the [Application Note on Image Acquisition](#).

Circular Prints

In some cases the symbols are not printed as straight lines but along arcs, e.g., on a CD. To read these, the (virtual) center and radius of the corresponding circle are extracted. Using the operator `po-lar_trans_image_ext`, the image is then unwrapped.

OCR Features

HALCON offers many different features for the OCR. Most of these are for advanced use only. In most cases it is recommended to use the feature combination `'default'`. This combination is based on the gray values within the surrounding rectangle of the character. In case that the background of the characters cannot be used, e.g., if it varies because of texture, the features `'pixel_binary'`, `'ratio'`, and `'anisometry'` are good combinations. Here, only the region is used, the underlying gray values are ignored.

3.12.9 Pretrained OCR Fonts

The following sections shortly introduce you to the pretrained OCR fonts provided by HALCON. You can access them in the subdirectory `ocr` of the folder where you installed HALCON.

Nomenclature for the Ready-to-Use OCR Fonts

There are several groups of OCR fonts available. The members of each group differ as they contain different symbol sets. The content of an OCR font is described by its name. For the names of the pretrained OCR fonts the following nomenclature is applied:

The name starts with the group name, e.g., `Document` or `DotPrint`, followed by indicators for the set of symbols contained in the OCR font. The meaning of the indicators is the following:

- 0-9: The OCR font contains the digits 0 to 9.
- A-Z: The OCR font contains the uppercase characters A to Z.
- + : The OCR font contains special characters. The list of special characters varies slightly over the individual OCR fonts. It is given below for each OCR font separately.

If the name of the OCR font does not contain any of the above indicators, typically, the OCR font contains the digits 0 to 9, the uppercase characters A to Z, the lowercase characters a to z, and special characters. Some of the OCR fonts do not contain lowercase characters or the full set of uppercase characters (e.g., MICR). For these OCR fonts, the contained symbols are named explicitly.

Ready-to-Use OCR Font `'Document'`

The OCR font `Document` can be used to read characters printed in fonts like Arial, Courier, or Times New Roman. These are typical fonts for printing documents or letters.

Available special characters: `- = + < > . # $ % & () @ * , $ €`

The following OCR fonts with different symbol sets are available:

- Document
- Document_0-9
- Document_0-9A-Z

Ready-to-Use OCR Font 'DotPrint'

The OCR font DotPrint can be used to read characters printed with dot printers (see [figure 3.40](#)).

It contains no lowercase characters.

Available special characters: - / . * , :

The following OCR fonts with different symbol sets are available:

- DotPrint
- DotPrint_0-9
- DotPrint_0-9+
- DotPrint_0-9A-Z

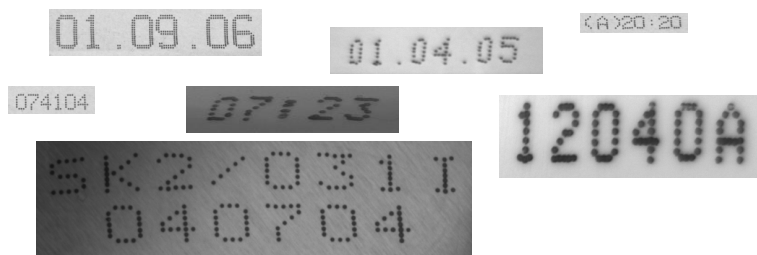


Figure 3.40: Examples for dot prints.

Ready-to-Use OCR Font 'HandWritten_0-9'

The OCR font HandWritten_0-9 can be used to read handwritten numbers (see [figure 3.41](#)).

It contains the digits 0-9.

Available special characters: none

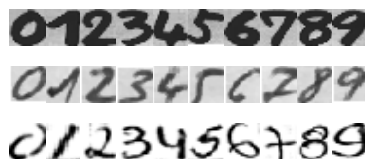


Figure 3.41: Examples for handwritten numbers.

Ready-to-Use OCR Font 'Industrial'

The OCR font `Industrial` can be used to read characters printed in fonts like Arial, OCR-B, or other sans-serif fonts (see [figure 3.42](#)). These fonts are typically used to print, e.g., labels.

Available special characters: - / + . \$ % * , €

The following OCR fonts with different symbol sets are available:

- `Industrial`
- `Industrial_0-9`
- `Industrial_0-9+`
- `Industrial_0-9A-Z`

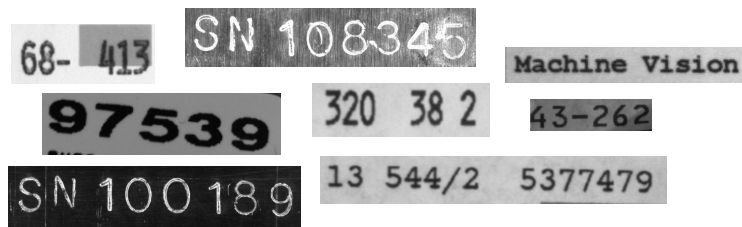


Figure 3.42: Examples for industrial prints.

Ready-to-Use OCR Font 'MICR'

The OCR font `MICR` can be used to read characters printed in the font `MICR` (see [figure 3.43](#)).

It contains the digits 0-9 and the characters A-D.

Available special characters: none

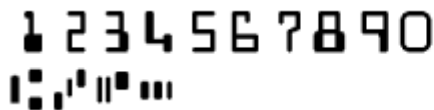


Figure 3.43: The MICR font.

Ready-to-Use OCR Font 'OCR-A'

The OCR font `OCR-A` can be used to read characters printed in the font `OCR-A` (see [figure 3.44](#)).

Available special characters: - ? ! / \ { } = + < > . # \$ % & () @ *

The following OCR fonts with different symbol sets are available:

- `OCRA`

- OCRA_0-9
- OCRA_0-9A-Z

0123456789
 ABCDEFGHIJKLM
 NOPQRSTUVWXYZ
 abcdefghijklm
 nopqrtsuvwxyz
 - ? ! / \ = + < > . # \$ % & () @ *

Figure 3.44: The OCR-A font.

Ready-to-Use OCR Font 'OCR-B'

The OCR font OCR-B can be used to read characters printed in the font OCR-B (see [figure 3.45](#)).

Available special characters: - ? ! / \ { } = + < > . # \$ % & () @ *

The following OCR fonts with different symbol sets are available:

- OCRB
- OCRB_0-9
- OCRB_0-9A-Z

0123456789
 ABCDEFGHIJKLM
 NOPQRSTUVWXYZ
 abcdefghijklm
 nopqrtsuvwxyz
 - ? ! / \ = + < > . # \$ % & () @ *

Figure 3.45: The OCR-B font.

Ready-to-Use OCR Font 'Pharma'

The OCR font Pharma can be used to read characters printed in fonts like Arial, OCR-B, and other fonts that are typically used in the pharmaceutical industry (see [figure 3.46](#)).

This OCR font contains no lowercase characters.

Available special characters: - / . () :

The following OCR fonts with different symbol sets are available:

- Pharma
- Pharma_0-9
- Pharma_0-9A-Z

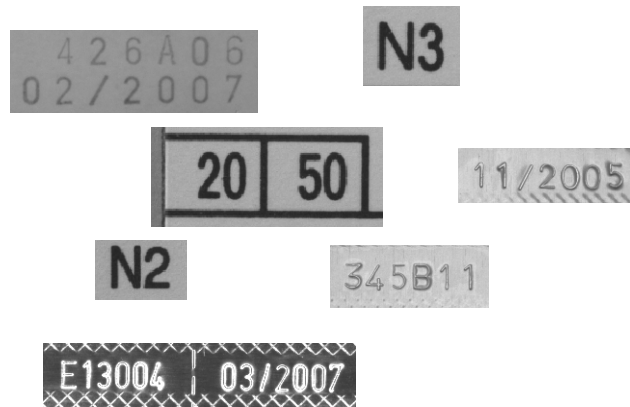


Figure 3.46: Examples for pharmaceutical labels.

3.13 Stereo

The basic principle of binocular stereo is that 3D coordinates of object points are determined from two images that are acquired simultaneously from different points of view. For example, the top row of [figure 3.47](#) shows the stereo image pair of a board. With binocular stereo, the heights of the individual components of the board can be determined. The bottom row of [figure 3.47](#) contains the height map for the entire overlapping area of the stereo image pair. In addition, a part of the height map is shown as a 3D plot.

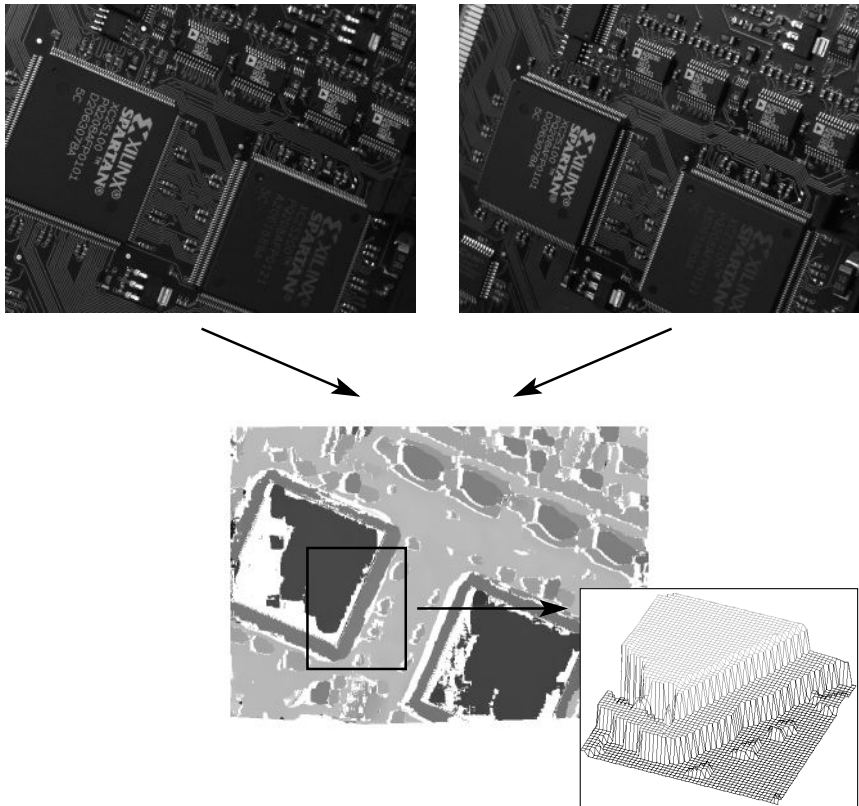


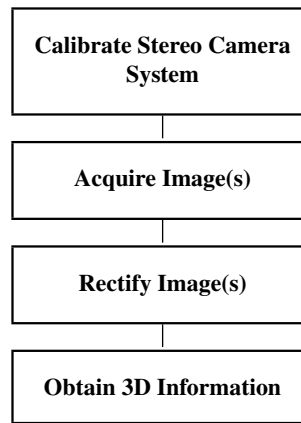
Figure 3.47: Basic principle of binocular stereo. Top: stereo image pair; bottom: height map.

The advantage of binocular stereo is that 3D information of the surface of arbitrarily shaped objects can be determined from images. Binocular stereo can also be combined with other vision methods, e.g., as a preprocessing step for blob analysis, which can then be used to extract objects from the depth image.

Note that this section describes the fully calibrated version of stereo vision. HALCON also provides an uncalibrated version. Please refer to the Application Note on 3D Machine Vision, [section 7.5](#) on page [108](#), for more information.

3.13.1 Basic Concept

The derivation of 3D information with a binocular stereo system consists of four main steps:



Calibrate Stereo Camera System

First, the stereo camera system is calibrated. For this, a number of stereo calibration images must be acquired, each showing the HALCON calibration plate in a different position and orientation. The object coordinates of the calibration marks can be read from the calibration plate description file. The image coordinates of the calibration marks must be extracted from the calibration images. Then, the parameters of the stereo setup are determined. With this, the rectification maps for the rectification of the stereo images can be determined.

The calibration process is described in detail in the Application Note on 3D Machine Vision in [section 7.3](#) on page 93.

Acquire Image(s)

Stereo images are acquired with the calibrated stereo camera system.

Rectify Image(s)

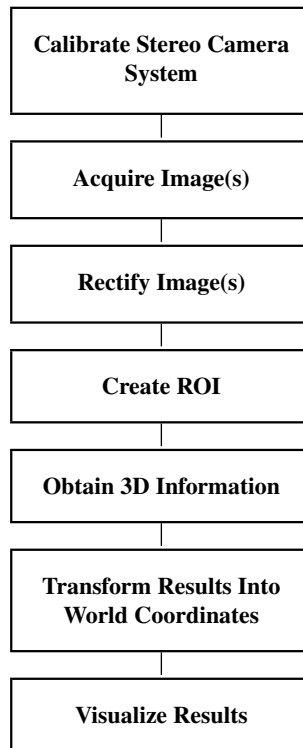
Having obtained a pair of stereo images, they must be rectified such that corresponding points (conjugate points) lie on the same row in both rectified images. For this, the rectification map that has been determined above must be used.

Obtain 3D Information

In the final step, for each point of the first rectified image the conjugate point in the second rectified image will be determined (stereo matching). For these points, either the disparity or the distance to the stereo camera system can be calculated and returned as a gray value image. The reference plane to which these distances are related can be changed (see the Application Note on 3D Machine Vision, [section 7.4.3](#) on page 104). It is also possible to derive the distance or the 3D coordinates for selected points for which the disparity is known. What is more, 3D coordinates can be determined from the image coordinates of each pair of conjugate points directly.

3.13.2 Extended Concept

In many cases the derivation of 3D information with a binocular stereo system will involve more steps than described above. Reasons for this are, e.g., the need to restrict the stereo reconstruction to an ROI. Furthermore, postprocessing, like transforming the 3D coordinates into another coordinate system or visualization of results, is often required.



Create ROI

A region of interest can be created to reduce the image domain for which the stereo matching will be performed. This will reduce the processing time.

For detailed information see the [description of this method](#) on page 27.

Transform Results Into World Coordinates

In some applications, the 3D coordinates must be transformed into a given world coordinate system. For this, the relation between the given world coordinate system and the stereo camera system must be determined. Then, the 3D coordinates can be transformed as requested.

How to transform results into world coordinates is described in detail in the Application Note on 3D Machine Vision in [section 3.2](#) on page 44.

Visualize Results

Finally, you might want to visualize the disparity or distance images, e.g., as a 3D plot or as contour lines, or you may need to visualize the 3D coordinates.

For detailed information see the [description of this method](#) on page 173.

3.13.3 Industries

3.13.3.1 Semiconductors

Typical tasks where binocular stereo can be used comprise, but are not limited to, completeness checks and the inspection of ball grid arrays.

3.13.3.2 Automobile Parts And Manufacturers

Binocular stereo can be used for the contact-free measurement of the 3D surface of arbitrarily shaped objects.

3.13.3.3 Machinery

In this industry as well, binocular stereo can be used for the contact-free measurement of the 3D surface of arbitrarily shaped objects.

3.13.4 Programming Examples

This section gives a brief introduction to using HALCON for binocular stereo.

3.13.4.1 Segment the Components of a Board

Example: [examples\hdevelop\Applications\Stereo\board_components.dev](#)

[Figure 3.48](#) shows a stereo image pair of a board together with the result of a segmentation of raised objects. The segmentation has been carried out based on the distance image that was derived with binocular stereo.

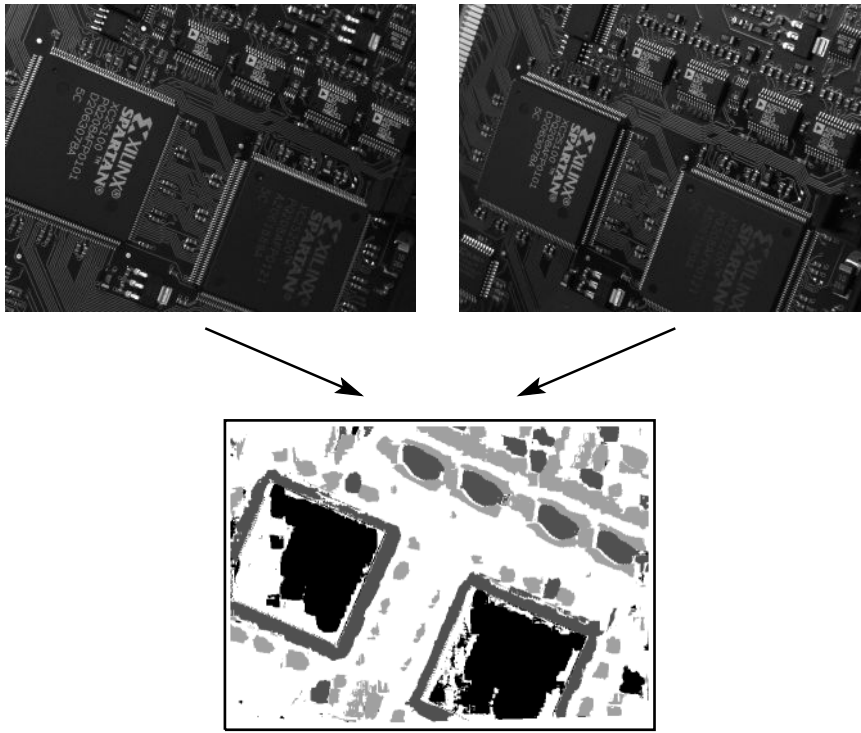


Figure 3.48: Segment board components based on their height.

First, the stereo camera system must be calibrated. For this, a number of calibration image pairs must be acquired. The calibration plate must be completely visible in each calibration image. A subset of these calibration images is shown in [figure 3.49](#).

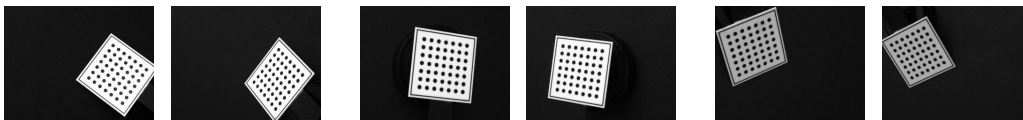


Figure 3.49: A subset of the calibration images that are used for the calibration of the stereo camera system.

The approximate position of the calibration plate is determined and the image coordinates of the calibration marks are extracted from the individual calibration images. The coordinates and the initial values for the poses of the calibration plates must be accumulated.

```

for imgnr := 1 to Number by 1
  find_caltab (ImageL, CaltabL, CalDescrFile, SizeGauss, MarkThresh,
              MinDiamMarks)
  find_caltab (ImageR, CaltabR, CalDescrFile, SizeGauss, MarkThresh,
              MinDiamMarks)
  find_marks_and_pose (ImageL, CaltabL, CalDescrFile, StartCamParL,
                      StartThresh, DeltaThresh, MinThresh, Alpha, MinContLength,
                      MaxDiamMarks, RCoordL, CCoordL, StartPoseL)
  find_marks_and_pose (ImageR, CaltabR, CalDescrFile, StartCamParR,
                      StartThresh, DeltaThresh, MinThresh, 0.7, MinContLength,
                      MaxDiamMarks, RCoordR, CCoordR, StartPoseR)
  RowsL := [RowsL,RCoordL]
  ColsL := [ColsL,CCoordL]
  StartPosesL := [StartPosesL,StartPoseL]
  RowsR := [RowsR,RCoordR]
  ColsR := [ColsR,CCoordR]
  StartPosesR := [StartPosesR,StartPoseR]
endfor

```

The object coordinates of the calibration marks can be read from the calibration plate description file.

```
caltab_points (CalDescrFile, X, Y, Z)
```

With this, the actual calibration of the stereo camera system can be performed.

```

binocular_calibration (X, Y, Z, RowsL, ColsL, RowsR, ColsR, StartCamParL,
                    StartCamParR, StartPosesL, StartPosesR, 'all', CamParamL, CamParamR,
                    NFinalPoseL, NFinalPoseR, cLPcR, Errors)

```

Now, the rectification maps for the rectification of the stereo image pair can be generated.

```

gen_binocular_rectification_map (MapL, MapR, CamParamL, CamParamR, cLPcR,
                                1, 'geometric', 'bilinear', RectCamParL, RectCamParR, CamPoseRectL,
                                CamPoseRectR, RectLPosRectR)

```

Then, each stereo image pair acquired with the calibrated stereo camera system can be rectified. This has the effect that conjugate points have the same row coordinate in both images. The rectified images are displayed in [figure 3.50](#)

```

map_image (ImageL, MapL, ImageRectifiedL)
map_image (ImageR, MapR, ImageRectifiedR)

```

From the rectified images, a distance image can be derived in which the gray values represent the distance of the respective object point to the stereo camera system. This step is the core of the stereo approach. Here, the stereo matching, i.e., the determination of the conjugate points takes place.



Figure 3.50: Rectified images.

```
binocular_distance (ImageRectifiedL, ImageRectifiedR, DistanceImage,
    ScoreImageDistance, RectCamParL, RectCamParR, RectLPosRectR, 'ncc',
    MaskWidth, MaskHeight, TextureThresh, MinDisparity, MaxDisparity,
    NumLevels, ScoreThresh, 'left_right_check', 'interpolation')
```

Finally, the distance image can, e.g., be corrected such that a given object plane receives a specified distance value, e.g., zero. Objects that deviate from the given object plane can thus be segmented very easily with a threshold operation.

```
threshold (HeightAboveReferencePlaneReduced, Range1, Height1_Min,
    Height1_Max)
threshold (HeightAboveReferencePlaneReduced, Range2, Height2_Min,
    Height2_Max)
threshold (HeightAboveReferencePlaneReduced, Range3, Height3_Min,
    Height3_Max)
```

3.13.4.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\3d_information_for_selected_points.dev](#)
Calculating world coordinates for a point in a stereo image pair
→ description in the [Application Note on 3D Machine Vision](#) on page 106
- [examples\application_guide\3d_machine_vision\hdevelop\height_above_reference_plane_from_stereo.dev](#)
Extracts chips using height information from binocular stereo
→ description in the [Application Note on 3D Machine Vision](#) on page 100
- [examples\application_guide\3d_machine_vision\hdevelop\stereo_calibration.dev](#)

Calibrates a stereo system

→ description in the [Application Note on 3D Machine Vision](#) on page 95

- [examples\hdevelop\Applications\Stereo\board_segmentation_uncalib.dev](#)
Segmentation of board components by height using uncalibrated binocular stereo
- [examples\hdevelop\Applications\Stereo\disparity.dev](#)
Shows disparity results for several stereo image pairs
- [examples\hdevelop\Filter\Inpainting\harmonic_interpolation.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_aniso.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_ced.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_mcf.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Tools\Stereo\binocular_calibration.dev](#)
Calibrating stereo systems and rectifying image pairs
- [examples\hdevelop\Tools\Stereo\binocular_disparity.dev](#)
Calculating disparities from epipolar image pairs
- [examples\hdevelop\Tools\Stereo\binocular_disparity_segmentation.dev](#)
Demonstrates stereo results using an artificial image pair
- [examples\hdevelop\Tools\Stereo\disparity_to_point_3d.dev](#)
Reconstructing 3D information from disparity
- [examples\hdevelop\Tools\Stereo\intersect_lines_of_sight.dev](#)
Reconstructing 3D information by intersecting lines of sight
- [examples\hdevelop\Tools\Stereo\uncalib_stereo_boxes.dev](#)
Determine the surfaces of boxes using uncalibrated binocular stereo.

3.13.5 Selecting Operators

Calibrate Stereo Camera System

Standard:

[caltab_points](#), [find_caltab](#), [find_marks_and_pose](#), [binocular_calibration](#),
[gen_binocular_rectification_map](#)

Acquire Image(s)

Please refer to the [operator list for the method Image Acquisition](#) (see section 3.1.4 on page 25).

Rectify Image(s)

Standard:

[map_image](#)

Create ROI

Standard:

[reduce_domain](#)

Further operators can be found in the [operator list for the method Region Of Interest](#) (see section 3.2.5 on page 36).

Obtain 3D Information

Standard:

[binocular_disparity](#), [binocular_distance](#), [disparity_to_distance](#),
[disparity_to_point_3d](#), [distance_to_disparity](#), [intersect_lines_of_sight](#)

Transform Results Into World Coordinates

Operators for transforming results into world coordinates are described in the [Application Note on 3D Machine Vision](#).

Visualize Results

Please refer to the [operator list for the method Visualization](#) (see section 3.14.5 on page 184).

3.13.6 Relation to Other Methods

3.13.6.1 Methods that are Using Stereo

Blob Analysis (see [description](#) on page 39)

The results of stereo can be used as the input for blob analysis. This may be useful if locally high structures must be extracted that cannot be reliably detected from the original image. By applying blob analysis to the distance image, such structures may be extracted very easily.

3.13.7 Tips & Tricks

Use of Domains (Regions of Interest)

The concept of domains (the HALCON term for a region of interest) is very important for stereo. With domains, the processing can be focused to a certain area in the image and thus sped up. For example, you can restrict the computation of 3D information to a certain region. For an overview on how to construct regions of interest and how to combine them with the image, see the method [Region Of Interest](#) on page 27.

Speed Up

Many online applications require maximum speed. Although the stereo matching is a very complex task, HALCON offers ways to speed up this process.

- Regions of interest are the standard method to increase the speed by processing only those areas where objects must be inspected. This can be done by using pre-defined regions but also by an online generation of the region of interest that depends on other objects found in the image.
- The use of image pyramids for the stereo matching reduces the processing time. The matching is started on the specified level of the image pyramid. The respective results are used as initial values for the matching on the next lower level of the image pyramid.

3.13.8 Advanced Topics

High Accuracy

Sometimes very high accuracy is required. To achieve a high distance resolution, i.e., a high accuracy with which the distance of the object surface from the stereo camera system can be determined, special care should be taken of the configuration of the stereo camera setup. The setup should be chosen such that the distance between the cameras as well as the focal length are large, and that the stereo camera system is placed as close as possible to the object. For more information, please refer to the Application Note on 3D Machine Vision, [section 7.2](#) on page 92.

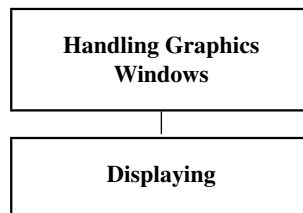
3.14 Visualization

Displaying data in HALCON is quite easy: In the graphics windows provided by HALCON, all supported data types can be visualized directly using specific display operators. Both the creation of these windows and the displaying requires only little programming effort because the functionality is optimized for the use in machine vision.

Making use of the HALCON visualization provides several advantages. First of all, it saves a lot of time during the development because all important visualization methods are already predefined. Furthermore, the functionality is independent of the operating system: Writing a program under Windows using the visualization of HALCON can be ported easily to Linux because the visualization operators behave identically and therefore only user code making use of operating system functions needs to be rewritten.

3.14.1 Basic Concept

For visualization there are two important aspects to consider: The graphics windows and the data that must be visualized.



Handling Graphics Windows

HALCON provides an easy-to-use operator to create a graphics window for visualization: [open_window](#). This operator takes all necessary parameters to specify the dimensions, the mode, and the relation to a potential parent window. As a result, a `WindowHandle` is returned with which you refer to the window when displaying into it or when visualization parameters are changed. The most important operators for controlling a window are [clear_window](#) to reset it to its background color, [set_part](#) to specify the display coordinate system and [close_window](#) when the window is no longer needed.

Displaying

For each HALCON data type, specific operators for displaying are provided. The most convenient operator for iconic data (images, regions, and XLD) is [disp_obj](#). This operator automatically handles gray or color images, regions, and XLDs. To control the way how data is presented in the window, operators with the prefix `set_` (or `dev_set_` for the visualization in HDevelop) are used. They allow to control the color, the draw mode, the line width, and many other parameters.

A First Example

An example for this basic concept is the following program, which shows how to visualize an image overlaid with a segmentation result. Here, the visualization operators provided in HDevelop are used.

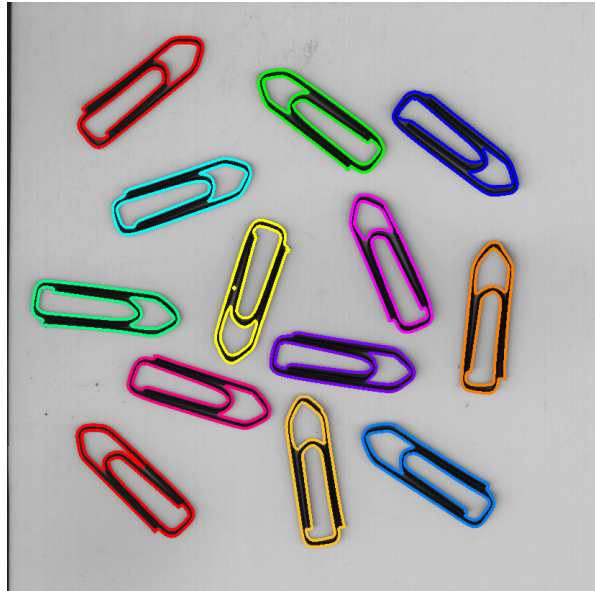


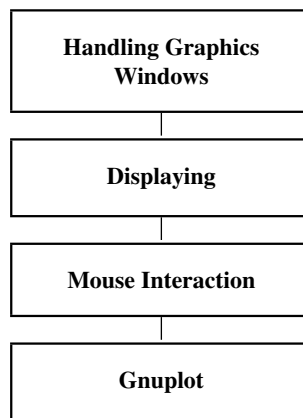
Figure 3.51: Visualizing the segmented clips.

After reading the image from file the dark clips are selected with `bin_threshold`, which automatically selects the threshold value. After determining the connected components and selecting regions with the appropriate size, the result is visualized. First, the image is displayed. After this, the parameters for regions are set to multi-colors (12) and margin mode. Finally, the regions are displayed with these settings.

```
read_image (Image, 'clip')
bin_threshold (Image, Dark)
connection (Dark, Single)
select_shape (Single, Selected, 'area', 'and', 5000, 10000)
dev_display (Image)
dev_set_colored (12)
dev_set_draw ('margin')
dev_display (Selected)
```

3.14.2 Extended Concept

In advanced applications it is required to gain complete control over the visualization process. This can include using graphics windows in programs developed with Microsoft Visual Basic or Microsoft C++, changing the behavior of the graphics windows, making use of buffered output, or even using external programs for the visualization. HALCON provides full control over all these topics to provide advanced flexibility, in addition to the ease of use.



Handling Graphics Windows

In this section we consider the concept of graphics windows in more detail.

- The graphics windows are designed such that each one stores all the corresponding parameters in a kind of graphics context. Whenever an operator like `set_draw` is called, this context is modified to be used for this window until the value is overwritten. All operators that modify the graphics context start with the prefix `set_`. The current value of the context can be requested using the corresponding operator with the prefix `get_`, e.g., `get_draw`.
- Besides display parameters, each graphics window has a coordinate system that can be defined with `set_part`. The upper left corner of an image is (0,0), the lower right corner is (height-1,width-1). The size of an image can be requested using `get_image_pointer1`. Please note that unlike in HDevelop this coordinate system must be specified by calling `set_part`. Otherwise, the part of the image that is visualized is undefined.
- The operator `open_window` has a parameter called `Father`. By default, the value 0 is used, which means that the window has no parent and floats as an independent instance. You can construct hierarchies of windows by passing the handle of one window as parent to the other. Besides this, it is also possible to use the parent mechanism to embed the graphics windows into other forms (see [section 3.14.7](#) on page 185).

Displaying

After having opened a graphics window, the returned window handle is used to communicate with it. Typically, first the visualization parameters are specified before data is displayed. To control images

only few operators are needed: `set_paint` (for profiles and 3D plots), `set_lut` (for look-up-tables), and `set_part_style` (for the zooming interpolation). To specify the output for regions, many parameters are available. The most important ones are `set_draw` (for filled or margin mode), `set_color` (for the pen color), `set_line_width` (for the pen width), and `set_colored` (for multi-color modes). To display XLD data, the same parameters (except `set_draw`) are used.

The visualization itself is performed with operators like `disp_image`, `disp_color`, `disp_region`, or `disp_xld`. The most convenient way is to use `disp_obj`, which automatically uses the correct method.

For text output, you first specify the font with `set_font`. The desired position in the graphics window is determined with `set_tposition`. Writing text into the window is performed with `write_string`

Mouse Interaction

The most important thing to know is that HALCON does *not* use an event-driven approach for mouse handling. Each operator is designed such that the mouse interaction starts when the operator is called and finishes when the operator returns. If an event-driven mode is needed one has to use the standard mechanisms provided by the operating system.

Interacting with the mouse mainly involves two tasks:

- The first task is to request the position of the mouse. This can be achieved using `get_mposition`. This operator returns immediately and has the position and the mouse button as result. An alternative is `get_mbutton`. This operator only returns when a mouse button has been clicked.
- The second important action is drawing shapes with the mouse. This is done with special operators whose names start with `draw_`. Operators for many different shapes (like circles or rectangles) are provided. Furthermore, different data types like regions or XLD contours can be used for the result.

Gnuplot

To visualize numeric values, HALCON provides an interface to the public domain software Gnuplot. This tool is especially useful to plot tuples of values like distributions and also provides extended versions of 3D plots of images.

3.14.3 Industries

3.14.3.1 Semiconductors

The program `pm_measure_board.dev` on page 213 from a chip inspection application is a good example to demonstrate the visualization of images, regions, XLD, and text.

3.14.3.2 Health Care And Life Science

The example `lines_gauss.dev` on page 196 from a medical application shows how to display XLD data including its attribute values.

3.14.3.3 Machinery

The example [cbm_pipe_wrench.dev](#) on page 201 shows how to handle visualization of complex data like a component model and the matching results including text, symbolic data, and model contours.

3.14.4 Programming Examples

This section gives a brief introduction to using the visualization operators provided by HALCON.

3.14.4.1 Displaying HALCON data structures

Example: [examples\quick_guide\hdevelop\display_operators.dev](#)

The example program is designed to show the major features of displaying images, regions, XLD, and text. It consists of a small main program that calls procedures handling the four different data types. The program is written for HDevelop and uses its specific display operators. This is done in a way that naturally ports (e.g., with the automatic export) to other language interfaces like C++, C#, or Visual Basic.

The main procedure contains five procedures: `open_graphics_window`, `display_image`, `display_regions`, `display_xld`, and `display_text`. To switch between the programs of the individual procedures you can use the combo box Procedures in the program window. Following, selected parts of each procedure are explained.

```
read_image (Image, 'fabrik')
open_graphics_window (Image, WindowHandle)
display_image (Image, WindowHandle)
regiongrowing (Image, Regions, 1, 1, 3, 100)
display_regions (Image, Regions, WindowHandle)
edges_sub_pix (Image, Edges, 'lanser2', 0.5, 10, 30)
display_xld (Image, Edges, WindowHandle)
display_text (Image, Regions, WindowHandle)
```

`open_graphics_window` is a support procedure to open a graphics window that has the same size as the image. This is done by calling [get_image_pointer1](#) to access the image dimensions. Before opening the new window, the existing window is closed. To adapt the coordinate system accordingly, [dev_set_part](#) is called. This would be done automatically in HDevelop, but for the other programming environments this step is necessary. Finally, the default behavior of HDevelop of displaying each result automatically is switched off. This has the effect that only programmed output will be visible.

```
get_image_pointer1 (Image, _, _, Width, Height)
dev_close_window ()
dev_open_window (0, 0, Width, Height, 'white', WindowHandle)
dev_set_part (0, 0, Height-1, Width-1)
dev_update_window ('off')
```

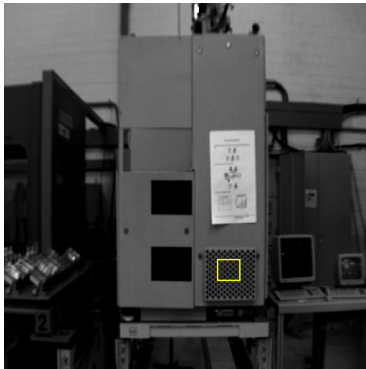
Then, the display procedure for images is called: `display_image`. It has the `Image` and the `WindowHandle` as input parameters. First, the window is activated, which again is not needed for HDevelop,

but is important for other programming environments. Now, the image is displayed in the graphics window. To change the look-up-table (LUT), `dev_set_lut` is called and the effect will become visible after calling `dev_display` once again.

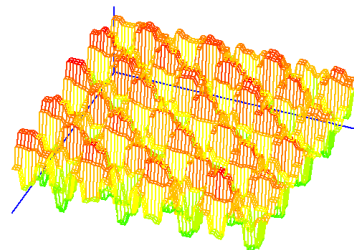
```
dev_set_window (WindowHandle)
dev_display (Image)
dev_set_lut ('temperature')
dev_display (Image)
```

Next, a part of the image is displayed using a so-called 3D plot (see [figure 3.52](#)). Here, the gray values are treated as height information. For this mode another LUT is used.

```
gen_rectangle1 (Rectangle, 358, 298, 387, 329)
dev_set_draw ('margin')
dev_set_color ('yellow')
dev_display (Rectangle)
dev_set_part (358, 298, 387, 329)
dev_set_lut ('twenty_four')
dev_set_paint (['3D-plot_hidden',7,1,110,160,600,0,0])
dev_display (Image)
```



a)



b)

Figure 3.52: (a) Original image with ROI; (b) 3D plot of image within the ROI.

The procedure to display regions is called `display_regions`. It first displays the image as background and then sets the display parameters for the regions. `dev_set_draw` specifies that only the region border is visualized. `dev_set_colored` activates the multi-color mode, where each region is displayed with different colors (which change cyclically). As an alternative to simply showing the original shape of the regions, HALCON enables you to modify the shape using `dev_set_shape`. In the given example the equivalent ellipses are chosen. The result is depicted in [figure 3.53](#).

```

dev_display (Image)
dev_set_draw ('margin')
dev_set_colored (6)
dev_display (Regions)
dev_display (Image)
dev_set_shape ('ellipse')
dev_display (Regions)

```

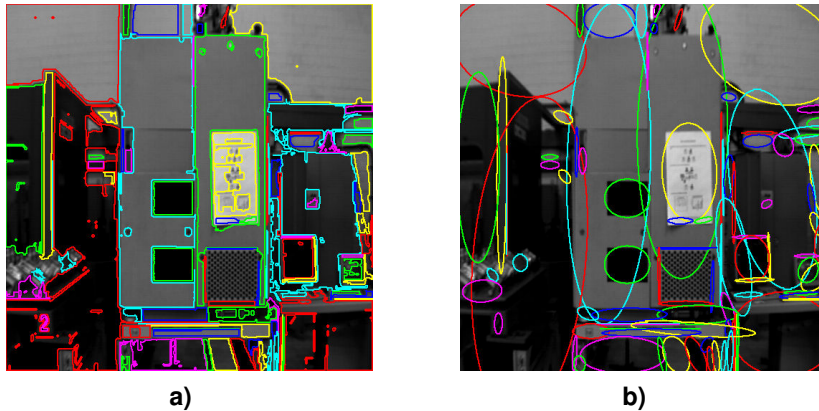


Figure 3.53: (a) Regions and (b) their equivalent ellipses.

The procedure `display_xld` first shows all contours overlaid on the image using the multi-color mode. Then, a zoom is defined using `dev_set_part`. This zoom mode allows to inspect the sub-pixel accurate contours easily. To give additional information, contours with a given size are selected and for each of these the control points are extracted using `get_contour_xld`. The coordinates are here returned as tuples of real values. For each of these control points, a cross is generated using `gen_cross_contour_xld`, which is then overlaid onto the contour.

```

dev_display (Contours)
gen_rectangle1 (Rectangle, 239, 197, 239+17, 197+17)
dev_set_part (239, 197, 239+17, 197+17)
select_shape_xld (Contours, SelectedXLD, 'area', 'and', 2000, 3000)
for k := 1 to Number by 1
    SingleContour := SelectedXLD[k]
    get_contour_xld (SingleContour, Row, Col)
    for i := 0 to |Row|-1 by 1
        gen_cross_contour_xld (Cross, Row[i], Col[i], 0.8, rad(45))
        dev_display (Cross)
    endfor
endfor

```

The last part of the program is a procedure called `display_text`. It shows how to handle the mouse and text output. The task is to click with the mouse into the graphics window to select a specific region and to calculate features, which are then displayed in the graphics window. First, the font is selected. This is the

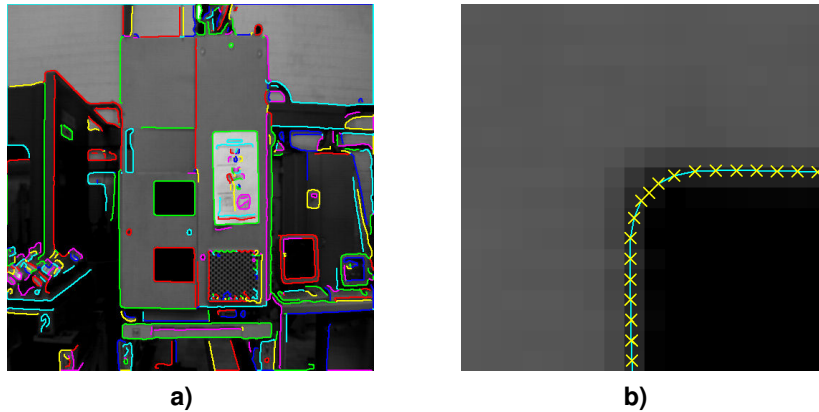


Figure 3.54: Subpixel accurate contour control points: (a) original image; (b) zoomed image part.

only part of the visualization where the parameters differ between Linux/UNIX and Windows because of the incompatible font name handling. However, as can be seen, the program can still be written so that it ports without problems.

```
OpSystem := environment('OS')
if (OpSystem='Windows_NT')
    set_font (WindowHandle, '-Courier New-16-*-*-*-*--')
else
    set_font (WindowHandle,
              '-*-courier-bold-r-normal--16-*-*-*-*--iso8859-1')
endif
```

The rest of the program consists of a while-loop, which terminates as soon as the right mouse button is pressed. The mouse operator `get_mbutton` waits until the user clicks with the mouse into the graphics window and then returns the coordinate and the button value. This coordinate is used to select the region that contains this point using `select_region_point`. For this region, the size and the center of gravity are calculated with `area_center`. First, the text cursor is positioned with `set_tposition` and the values are displayed using `write_string`. Here, it can be seen how conveniently strings can be composed using the "+" operator.

```
Button := 0
while (Button # 4)
    get_mbutton (WindowHandle, Row, Column, Button)
    select_region_point (Regions, DestRegions, Row, Column)
    area_center (DestRegions, Area, RowCenter, ColumnCenter)
    if (|Area| > 0)
        set_tposition (WindowHandle, Row, Column)
        dev_set_color ('yellow')
        write_string (WindowHandle, '('+RowCenter+', '+ColumnCenter+') = '+Area)
    endif
endwhile
```

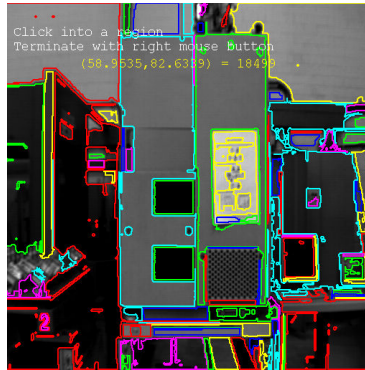



Figure 3.55: Center of gravity and area of selected region.

3.14.4.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_multi_image.dev](#)
Calibrates the camera and measures positions on a caliper rule
→ description in the [Application Note on 3D Machine Vision](#) on page 39
- [examples\application_guide\3d_machine_vision\hdevelop\handeye_movingcam_calibration.dev](#)
Performs hand-eye calibration for a hand-eye system with a moving camera
→ description in the [Application Note on 3D Machine Vision](#) on page 112
- [examples\application_guide\3d_machine_vision\hdevelop\handeye_stationarycam_calibration.dev](#)
Performs hand-eye calibration for a hand-eye system with a stationary camera
→ description in the [Application Note on 3D Machine Vision](#) on page 112
- [examples\application_guide\3d_machine_vision\hdevelop\handeye_stationarycam_grasp_nut.dev](#)
Calculates pose for grasping a nut based on results of hand-eye calibration for a stationary camera
→ description in the [Application Note on 3D Machine Vision](#) on page 120
- [examples\application_guide\3d_machine_vision\hdevelop\height_above_reference_plane_from_stereo.dev](#)
Extracts chips using height information from binocular stereo
→ description in the [Application Note on 3D Machine Vision](#) on page 100
- [examples\application_guide\shape_matching\hdevelop\multiple_scales.dev](#)
Searches for nuts of different sizes
→ description in the [Application Note on Shape-Based Matching](#) on page 44
- [examples\hdevelop\Applications\FA\ball.dev](#)

Inspection of ball bonding

→ description [here in the Quick Guide](#) on page 211

- [examples\hdevelop\Applications\FA\cbm_caliper.dev](#)
Measures the setting of a caliper using component-based matching in a perspective distorted image
- [examples\hdevelop\Applications\FA\circles.dev](#)
Fits circles into curved contour segments
→ description [here in the Quick Guide](#) on page 200
- [examples\hdevelop\Applications\FA\clip.dev](#)
Determines the orientation of clips
- [examples\hdevelop\Applications\FA\pm_measure_board.dev](#)
Locates IC on a board and measures pin distances
→ description [here in the Quick Guide](#) on page 213
- [examples\hdevelop\Applications\FA\pm_multiple_models.dev](#)
Finds multiple different models in a single pass using shape-based matching
- [examples\hdevelop\Applications\FA\pm_world_plane.dev](#)
Recognizes planar objects using shape-based matching in perspective distorted images
- [examples\hdevelop\Applications\Measure\fuzzy_measure_pin.dev](#)
Measures pins of an IC using fuzzy measure
→ description [here in the Quick Guide](#) on page 212
- [examples\hdevelop\Applications\Measure\measure_arc.dev](#)
Measures width of object along circular arc
→ description [here in the Quick Guide](#) on page 197
- [examples\hdevelop\Applications\Measure\measure_pin.dev](#)
Measures pins of an IC
→ description [here in the Quick Guide](#) on page 226
- [examples\hdevelop\Applications\Medicine\particle.dev](#)
Extracts particles of varying sizes
→ description [here in the Quick Guide](#) on page 194
- [examples\hdevelop\Applications\OCR\bottle.dev](#)
Segmenting and reading numbers on a beer bottle
→ description [here in the Quick Guide](#) on page 193
- [examples\hdevelop\Develop\dev_clear_window.dev](#)
Clearing a graphics window in HDevelop
- [examples\hdevelop\Develop\dev_close_window.dev](#)
Closing graphics windows in HDevelop
- [examples\hdevelop\Develop\dev_display.dev](#)
Displaying image objects in a graphics window in HDevelop

- [examples\hdevelop\Develop\dev_open_window.dev](#)
Opening graphics windows in HDevelop
- [examples\hdevelop\Develop\dev_set_paint.dev](#)
Painting data using different representations into a graphics window in HDevelop
- [examples\hdevelop\Develop\dev_set_part.dev](#)
Setting the part of an image to be displayed (zoomed) in a graphics window in HDevelop
- [examples\hdevelop\Develop\dev_set_window_extents.dev](#)
Setting size and position of a graphics window in HDevelop
- [examples\hdevelop\Filter\Geometric-Transformations\projective_trans_image_reduced.dev](#)
Applying projective transformations to an image and its domain
- [examples\hdevelop\Filter\Lines\lines_color.dev](#)
Extracting lines using color information
→ description [here in the Quick Guide](#) on page 191
- [examples\hdevelop\Manuals\HDevelop\exception.dev](#)
Querying the mouse position (with error handling)
→ description in the [HDevelop User's Manual](#) on page 150
- [examples\hdevelop\Regions\Geometric-Transformations\projective_trans_region.dev](#)
Applying projective transformations to regions
- [examples\hdevelop\Tools\2D-Transformations\gen_projective_mosaic.dev](#)
Combining several images of a PCB into a large mosaic image
→ description [here in the Quick Guide](#) on page 214
- [examples\hdevelop\Tools\2D-Transformations\projective_trans_pixel.dev](#)
Applying a projective transformation to rotate an image in 3D
- [examples\hdevelop\Tools\2D-Transformations\projective_trans_point_2d.dev](#)
Applying a projective transformation to rotate images and points in 3D
- [examples\hdevelop\Tools\2D-Transformations\vector_to_proj_hom_mat2d.dev](#)
Rectifies image of stadium to simulate overhead view
- [examples\hdevelop\Tools\Datacode\ecc200_optimized_settings.dev](#)
Optimizing parameters for reading a 2d data code of type ECC200
- [examples\hdevelop\Tools\Datacode\pdf417_default_settings.dev](#)
Reading various 2d data codes of type PDF417
- [examples\hdevelop\Tools\Datacode\pdf417_optimized_settings.dev](#)
Optimizing parameters for reading various 2d data codes of type PDF417
- [examples\hdevelop\Tools\Datacode\qrcode_optimized_settings.dev](#)
Optimizing parameters for reading a 2d data code of type QR Code
- [examples\hdevelop\XLD\Transformation\projective_trans_contour_xld.dev](#)

This program uses `hom_mat3d_project` and `projective_trans_region` to rotate an XLD contour in 3D

- [examples\quick_guide\hdevelop\critical_points.dev](#)
Locates saddle point markers in an image
→ description [here in the Quick Guide](#) on page 31

C++

- `examples\mfc\Matching\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, creating a HALCON window
- `examples\mfc\MatchingCOM\Matching.cpp`
Locating an IC using HALCON/COM and MFC
- `examples\mfc\MatchingExtWin\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, painting into an existing window
- `examples\motif\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Motif
- `examples\qt\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Qt

C#

- `examples\c#\Matching\Matching.csproj`
Locates an IC on a board and measures pin distances

Delphi

- `examples\delphi\Matching\matching.dpr`
Locates an IC on a board and measures pin distances

3.14.5 Selecting Operators

Handling Graphics Windows

Standard:

[open_window](#), [clear_window](#), [close_window](#), [open_window](#)

Displaying

Standard:

[set_paint](#), [set_lut](#), [set_part_style](#), [set_draw](#), [set_line_width](#), [set_color](#), [set_colored](#), [disp_obj](#), [set_font](#), [set_tposition](#), [write_string](#)

Advanced:

[disp_image](#), [disp_color](#), [disp_region](#), [disp_xld](#)

Mouse Interaction

Standard:

`get_mposition`, `get_mbutton`, `draw_region`, `draw_circle`

Gnuplot

Advanced:

`gnuplot_open_file`, `gnuplot_open_pipe`, `gnuplot_plot_ctrl`, `gnuplot_plot_image`

3.14.6 Tips & Tricks

Saving Window Content

HALCON provides an easy way to save the content of a graphics window to a file. This can, e.g., be useful for documentation purposes. The corresponding operator is called `dump_window`. It takes the window handle and the file name as input. The parameter `Device` allows to select amongst different file formats.

Execution Time

One fact concerning visualization is often underestimated: the influence of the bit depth on the display time. In most cases, 32 bits are chosen without giving any thoughts to it. However, depending on the graphics card, the different bit depths often vary dramatically in the execution time. Therefore, it is recommended to simply try different bit depths, to perform typical display operators like `disp_image`, `disp_color`, `disp_region`, or `disp_xld`, and to measure the execution time with `count_seconds`.

3.14.7 Advanced Topics

Programming Environments

The handling of graphics windows differs in the different programming environments. Especially HDevelop has a specific way of working with the visualization.

- Because HDevelop is an interactive environment, the handling of windows must be as easy as possible. In particular, it is important to display data without any need of programming. Therefore, the concept of window handles is used only if needed. Normally, the window where the output has to go to is not specified explicitly. Instead, HDevelop makes use of the activation status of the graphics windows. As a consequence, the display operators of HDevelop do not have a parameter for a window handle. Visualization operators in HDevelop look identical to their counterparts of HALCON except that their name starts with `dev_` and that the parameter for the window handle is suppressed. When exporting the code, this missing handle will automatically be inserted.

The second difference is that the windows in HDevelop have a history to automatically redisplay the data when the window has been resized. This is not the case with standard HALCON graphics windows.

The last difference is that HDevelop automatically sets the coordinate system according to the current image, whereas you must do this explicitly with programmed code when using HALCON. To make an export to, e.g., C++, C#, or Visual Basic transparent, we recommend to use `dev_set_window` when working with multiple graphics windows and to call `dev_set_part` to specify the coordinate system.

- When using HALCON windows in MFC, the usual way is to use the windows as a subwindow of a parent form. This can easily be achieved by using the window handle of the current form as the father. The handle must be converted to a long value that can then be passed to `open_window`. Note that `set_check` is needed to change to exception handling of HALCON in this context.

```
set_window_attr("border_width",0);
set_check("~father");
long lWWindowID = (long)m_hWnd;
open_window(0,0,640,480,lWWindowID,"visible","",&m_lWindowID);
set_check("father");
```

- Opening a HALCON window in Visual Basic is similar to the approach used for MFC. Here, you use the `memberhWnd` of the form or of another subwindow like a picture box. As an alternative, the `HWindowXCtrl` can be used.

```
Call sys.SetCheck("~father")
Call op.OpenWindow(8, 8, 320, 240, form.hWnd, "", "", WindowHandle)
Call sys.SetCheck("father")
```

Buffered Output

For a flicker-free visualization, a sequential call of display operators is not suitable, because after each call the new data will immediately be flushed on the visible screen, which may cause flickering results. Under Windows, this can easily be fixed by calling `set_system` with the parameter value `'flush_graphic'` set to `'false'`, then performing all display operators - except the last one - then calling `set_system` again with `'flush_graphic'` set to `'true'`, and finally the last display call of the display sequence. This will have the effect that the whole data will become visible in one step when applying the last display call. Under Linux/UNIX, for this purpose a buffer window must be used in addition to the visible window. A buffer window is opened with `open_window`, while setting `Mode` to `'buffer'`. Then, all drawing operations are performed into the buffer window. Finally, the buffer window contents are copied into the visible window using `copy_rectangle`. Of course, this mechanism also works under Windows.

Remote Visualization

In some applications, the computer used for processing differs from the computer used for visualization. Such applications can easily be created with HALCON using the socket communication. Operators like

[send_image](#) or [receive_tuple](#) allow a transparent transfer of the relevant data to the control computer to apply visualization there.

Programmed Visualization

Sometimes it might be necessary not to apply the standard HALCON visualization operators, but to use a self-programmed version. This can be achieved by using the access functions provided for all data types. Examples for these are [get_image_pointer1](#), [get_region_runs](#), or [get_contour_xld](#). Operators like these allow full access to all internal data types. Furthermore, they provide the data in various forms (e.g., runlength encoding, points, or contours) to make further processing easier. Based on this data, a self-programmed visualization can be developed easily.

As an alternative, with [set_window_type](#) the window type 'pixmap' can be chosen. In this case, all displayed data is painted into an internal buffer that can be accessed with [get_window_pointer3](#). The returned pointers reference the three color channels of the buffer. This buffer can then easily be transferred (e.g., to another system) and/or transformed into the desired format. One example for a conversion is to call [gen_image3](#) to create a HALCON color image.

Chapter 4

Industries

This chapter describes example programs from various industries.

4.1	Electric Components And Equipment	190
4.2	Food	193
4.3	Health Care And Life Science	194
4.4	Iron, Steel And Metal	197
4.5	Machinery	199
4.6	Photogrammetry And Remote Sensing	204
4.7	Printing	208
4.8	Rubber, Synthetic Material, Foil	209
4.9	Semiconductors	211

4.1 Electric Components And Equipment

4.1.1 Check the State of a Dip Switch

Example: [examples\hdevelop\Applications\FA\cbm_dip_switch.dev](#)

The task of this example is to check a dip switch, i.e., to determine the positions of the single switches relative to the casing (see [figure 4.1](#)).

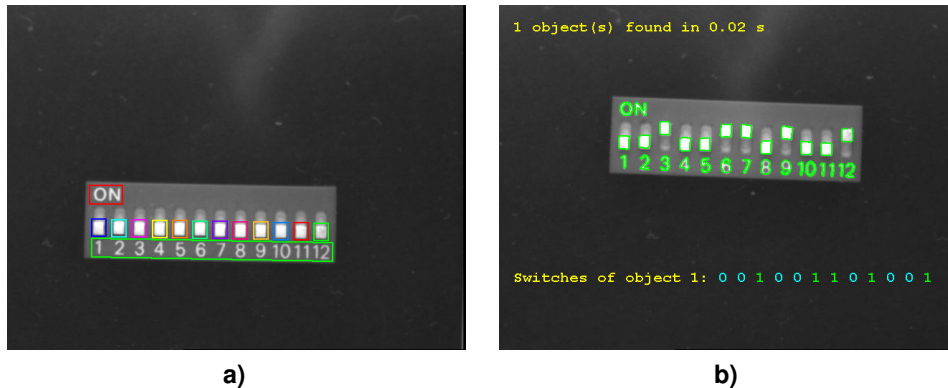


Figure 4.1: (a) Model image of the dip switch with ROIs for the components; (b) located dip switch with state of the switches.

The task is solved using component-based matching. The dip switch consists of 14 components: 12 for the switches and two for the printed text on the casing (see [figure 4.1a](#)). The training is performed using training images that show all possible positions of the switches. From this, the system learns the possible movements of the switches relative to the casing.

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

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

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

Now, the model can be created.

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

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

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

4.1.2 Inspect Power Supply Cables

Example: [examples\hdevelop\Filter\Lines\lines_color.dev](#)

The task of this example is to locate and inspect the power supply cables depicted in [figure 4.2](#).

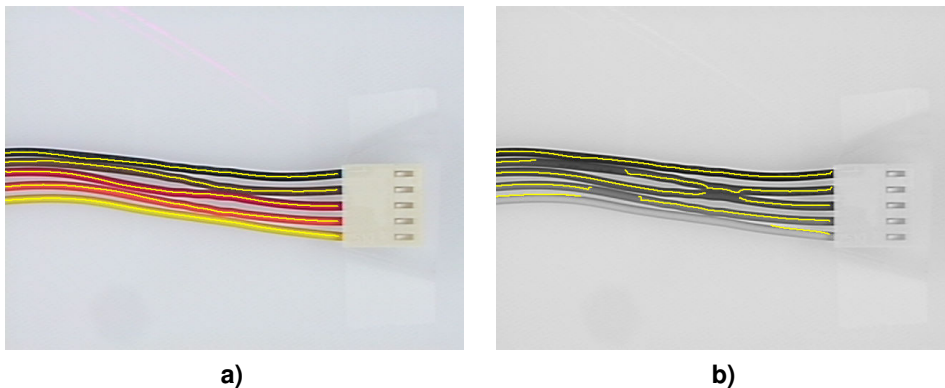


Figure 4.2: (a) Original color image with cable centers extracted using the color line extractor; (b) corresponding results when using the gray value image.

The input for the program are sample images of colored power supply cables. The task is to extract the centers of each cable together with the width. This is performed using the subpixel-precise color line extractor. To remove irrelevant structures, contours that are too short are removed.

```
lines_color (Image, Lines, 3.5, 0, 12, 'true', 'false')
select_contours_xld (Lines, LongLines, 'contour_length', 450, 100000, 0,
    0)
```

The cable width is determined by accessing the line width attribute. For display purposes, a contour is generated for each side.

```

Number := |LongLines|
EdgesL := []
EdgesR := []
for K := 1 to Number by 1
  Line := LongLines[K]
  get_contour_xld (Line, Row, Col)
  get_contour_attrib_xld (Line, 'angle', Angle)
  get_contour_attrib_xld (Line, 'width_right', WidthR)
  get_contour_attrib_xld (Line, 'width_left', WidthL)
  EdgeRR := Row+cos(Angle)*WidthR
  EdgeRC := Col+sin(Angle)*WidthR
  EdgeLR := Row-cos(Angle)*WidthL
  EdgeLC := Col-sin(Angle)*WidthL
  gen_contour_polygon_xld (EdgeR, EdgeRR, EdgeRC)
  gen_contour_polygon_xld (EdgeL, EdgeLR, EdgeLC)
  EdgesL := [EdgesL,EdgeL]
  EdgesR := [EdgesR,EdgeR]
endfor

```

To compare this result with the classical approach, a line extractor is also applied to the gray value image. The result is depicted in [figure 4.2b](#). Here, it becomes obvious how hard it is to extract the cable using the luminance only.

```

rgb1_to_gray (Image, GrayImage)
lines_gauss (GrayImage, LinesGray, 3.5, 0, 0.7, 'dark', 'true', 'true',
            'false')

```

4.1.3 Other Examples

HDevelop

- [examples\application_guide\1d_metrology\hdevelop\fuzzy_measure_switch.dev](#)
Determine the width of and the distance between the pins of a switch with a fuzzy measure object
→ description in the [Application Note on 1D Metrology](#) on page 25
- [examples\application_guide\1d_metrology\hdevelop\measure_switch.dev](#)
Determine the width of and the distance between the pins of a switch with a measure object
→ description in the [Application Note on 1D Metrology](#) on page 4
- [examples\hdevelop\Applications\Datacode\pdf417_optimized.dev](#)
Optimizing parameters for reading of 2d data codes of type PDF417
- [examples\hdevelop\Applications\Datacode\pdf417_simple.dev](#)
Reading 2d data codes of type PDF417
- [examples\hdevelop\Applications\FA\cbm_bin_switch.dev](#)
Locates a switch and tests its state using component-based matching
- [examples\hdevelop\Applications\FA\holes.dev](#)
Extracts positions and radii of holes

- `examples\hdevelop\Filter\Inpainting\harmonic_interpolation.dev`
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- `examples\hdevelop\Filter\Inpainting\inpainting_aniso.dev`
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- `examples\hdevelop\Filter\Inpainting\inpainting_ced.dev`
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- `examples\hdevelop\Filter\Inpainting\inpainting_mcf.dev`
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- `examples\hdevelop\Regions\Features\rectangularity.dev`
Calculating the rectangularity of regions
- `examples\quick_guide\hdevelop\color_fuses.dev`
Sort fuses by color
→ description [here in the Quick Guide](#) on page 116
- `examples\quick_guide\hdevelop\color_simple.dev`
Segment color image in HSV color space
→ description [here in the Quick Guide](#) on page 115
- `examples\quick_guide\hdevelop\fuse.dev`
Measures the thickness of a fuse wire
→ description [here in the Quick Guide](#) on page 57

4.2 Food

4.2.1 "Best Before" Date

Example: `examples\hdevelop\Applications\OCR\bottle.dev`

The task of this example is to inspect the "best before" date on the bottle depicted in [figure 4.3](#).



Figure 4.3: (a) Original image; (b) read date.

The task is solved in multiple steps. First, dark areas are extracted and post-processed to eliminate structures that are too thin.

```
threshold (Bottle, RawSegmentation, 0, 95)
fill_up_shape (RawSegmentation, RemovedNoise, 'area', 1, 5)
opening_circle (RemovedNoise, ThickStructures, 2.5)
fill_up (ThickStructures, Solid)
```

Next, the region is split into the individual characters; even the characters that are so close that they touch each other can be separated.

```
opening_rectangle1 (Solid, Cut, 1, 7)
connection (Cut, ConnectedPatterns)
intersection (ConnectedPatterns, ThickStructures, NumberCandidates)
select_shape (NumberCandidates, Numbers, 'area', 'and', 300, 9999)
sort_region (Numbers, FinalNumbers, 'first_point', 'true', 'column')
```

Finally, the actual reading is performed.

```
read_ocr_class_mlp (FontName, OCRHandle)
do_ocr_multi_class_mlp (FinalNumbers, Bottle, OCRHandle, RecNum, Confidence)
```

4.2.2 Other Examples

HDevelop

- [examples\hdevelop\Applications\OCR\bottlet.dev](#)
Segmenting and training numbers on a beer bottle

C++

- [examples\cpp\source\bottle.cpp](#)
Reads numbers on a beer bottle

4.3 Health Care And Life Science

4.3.1 Analyzing Particles

Example: [examples\hdevelop\Applications\Medicine\particle.dev](#)

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

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

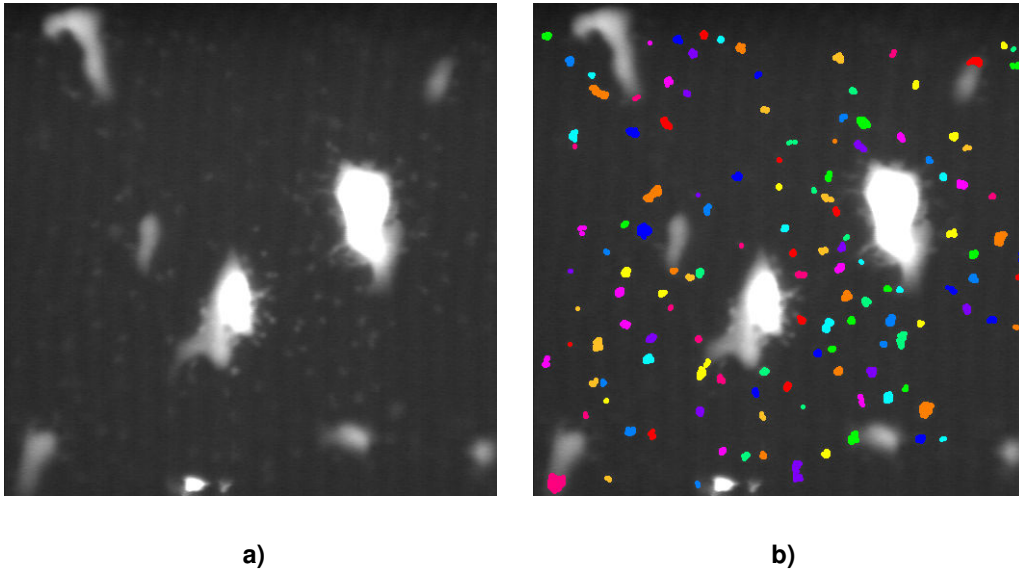


Figure 4.4: Extracting the small particles: (a) original image, (b) result.

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

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

4.3.2 Angiography

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

The task of this example is to extract the blood vessels in the X-ray image of the heart depicted in [figure 4.5](#). The vessels are emphasized by using a contrast medium. For the diagnosis it is important to extract the width of the vessels to determine locally narrowed parts (stenoses).

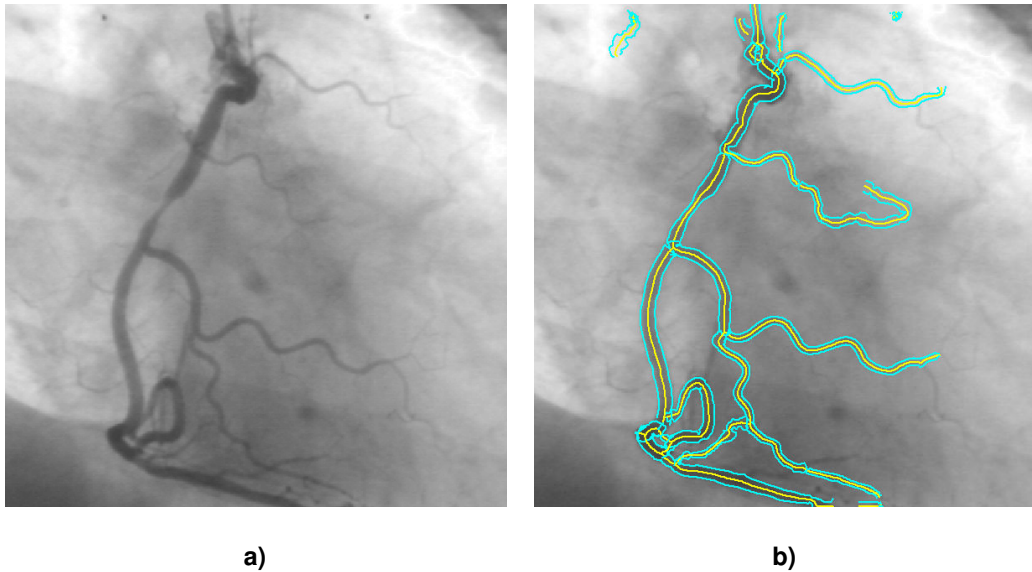


Figure 4.5: (a) X-ray image of the heart; (b) extracted blood vessels.

The vessels are extracted using `lines_gauss`. The result of this operator are the centers of the vessels in the form of XLD contours. Besides this, attributes are associated with the contour points, one of which is the local line width. This width is requested and displayed as contours.

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


4.3.3 Other Examples

HDevelop

- [examples\hdevelop\Applications\Medicine\angio.dev](#)
Extracts blood vessels and their diameters from an angiogram
- [examples\hdevelop\Applications\Medicine\vessel.dev](#)
Segmentation and measurement of a blood vessel
- [examples\hdevelop\Manuals\HDevelop\particle.dev](#)
Measures small particles
→ description in the [HDevelop User's Manual](#) on page 136
- [examples\hdevelop\Manuals\HDevelop\vessel.dev](#)
Extracts a capillary vessel
→ description in the [HDevelop User's Manual](#) on page 133

C++

- [examples\cpp\source\example5.cpp](#)
Analyzes the distribution of cell sizes

4.4 Iron, Steel And Metal

4.4.1 Inspect Cast Part

Example: [examples\hdevelop\Applications\Measure\measure_arc.dev](#)

The task of this example is to inspect the distance between elongated holes of a cast part after chamfering (see [figure 4.6](#)). Note that to achieve best accuracy it would be recommended to use backlight combined with a telecentric lens instead of the depicted setup.

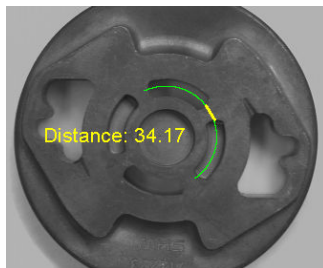


Figure 4.6: Measuring the distance between the holes.

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

```

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

```

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

```

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

```

4.4.2 Other Examples

HDevelop

- [examples\application_guide\shape_matching\hdevelop\align_measurements.dev](#)
Inspects individual razor blades using shape-based matching to align ROIs for the measure tool
→ description in the [Application Note on Shape-Based Matching](#) on page 36
- [examples\application_guide\shape_matching\hdevelop\multiple_models.dev](#)
Searching for two types of objects simultaneously
→ description in the [Application Note on Shape-Based Matching](#) on page 25
- [examples\application_guide\shape_matching\hdevelop\multiple_objects.dev](#)
Searches for multiple instances of a security ring
→ description in the [Application Note on Shape-Based Matching](#) on page 34
- [examples\application_guide\shape_matching\hdevelop\multiple_scales.dev](#)
Searches for nuts of different sizes
→ description in the [Application Note on Shape-Based Matching](#) on page 44
- [examples\application_guide\shape_matching\hdevelop\reuse_model.dev](#)
Storing and reusing a shape model
→ description in the [Application Note on Shape-Based Matching](#) on page 46
- [examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev](#)
Determine the pose of circles in 3D from their perspective 2D projections
- [examples\hdevelop\Applications\FA\cbm_pipe_wrench.dev](#)
Locates a pipe wrench that consists of two components
→ description [here in the Quick Guide](#) on page 201
- [examples\hdevelop\Applications\FA\pm_multiple_models.dev](#)
Finds multiple different models in a single pass using shape-based matching
- [examples\hdevelop\Applications\FA\pm_world_plane.dev](#)
Recognizes planar objects using shape-based matching in perspective distorted images

- [examples\hdevelop\Applications\OCR\engraved.dev](#)
Segmenting and reading characters on a metal surface
→ description [here in the Quick Guide](#)
- [examples\hdevelop\Applications\OCR\engravedt.dev](#)
Segmenting and training characters on a metal surface
- [examples\hdevelop\Matching\Gray-Value-Based\best_match_mg.dev](#)
Finding the best match of a gray value template in a pyramid
- [examples\hdevelop\Tools\Geometry\distance_pc.dev](#)
Calculating the distance between a point and a contour
- [examples\quick_guide\hdevelop\measure_metal_part.dev](#)
Inspects metal part by fitting lines and circles
→ description [here in the Quick Guide](#) on page 90
- [examples\quick_guide\hdevelop\surface_scratch.dev](#)
Detects scratches on a surface via local thresholding and morphology
→ description [here in the Quick Guide](#) on page 241

4.5 Machinery

4.5.1 Reading Engraved Text

Example: [examples\hdevelop\Applications\OCR\engraved.dev](#)

The task of this example is to read the engraved text on the metal surface depicted in [figure 4.7](#).

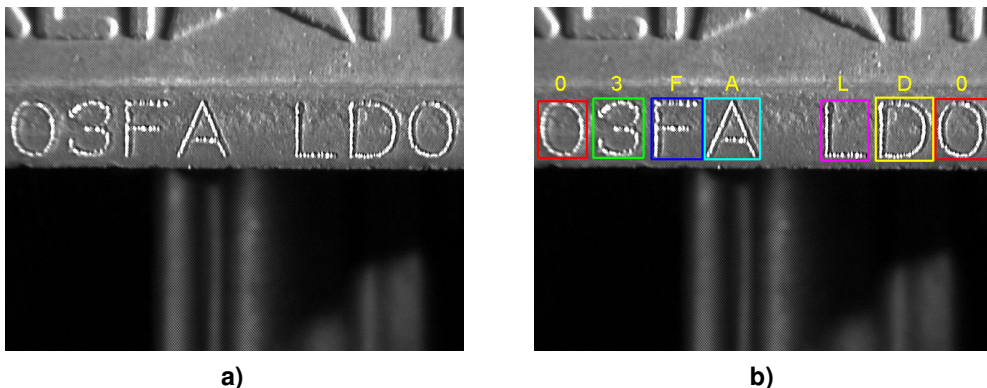


Figure 4.7: (a) Original image; (b) read characters.

The task is solved by using advanced blob analysis: The characters cannot simply be extracted by selecting dark or light pixels. Instead, a simple segmentation would yield only fractions of the characters together with noise objects. Preprocessing the image using gray value morphology allows to segment the real characters.

```

gray_range_rect (Image, ImageResult, 7, 7)
invert_image (ImageResult, ImageInvert)
threshold (ImageResult, Region, 128, 255)
connection (Region, ConnectedRegions)
select_shape (ConnectedRegions, SelectedRegions, 'area', 'and', 1000, 99999)
sort_region (SelectedRegions, SortedRegions, 'first_point', 'true',
            'column')

```

Finally, the actual reading is performed.

```

read_ocr_class_mlp (FontName, OCRHandle)
for i := 1 to Number by 1
    ObjectSelected := SortedRegions[i]
    do_ocr_single_class_mlp (ObjectSelected, ImageInvert, OCRHandle, 1,
                          Class, Confidence)
endfor
clear_ocr_class_mlp (OCRHandle)

```

4.5.2 Inspecting the Contours of a Tool

Example: [examples\hdevelop\Applications\FA\circles.dev](#)

The task of this example is to inspect the contours of the tool depicted in [figure 4.8](#).



Figure 4.8: Fitting circles to the contours of the tool.

Because the subpixel-precise contour extraction is time-consuming, in a first step the edges are located roughly based on standard blob analysis: With a threshold operator the object to be measured is extracted. This region is converted to its boundary, omitting the pixels at the image border.

```

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

```

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

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

For the segments that represent arcs, the corresponding circle parameters are determined. For inspection purposes, circles with the same parameters are generated and overlaid on the image.

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

4.5.3 Locating a Pipe Wrench in Different States

Example: [examples\hdevelop\Applications\FA\cbm_pipe_wrench.dev](#)

The task of this example is to locate a pipe wrench based on four predefined ROIs, two for each rigid part (see [figure 4.9](#)).

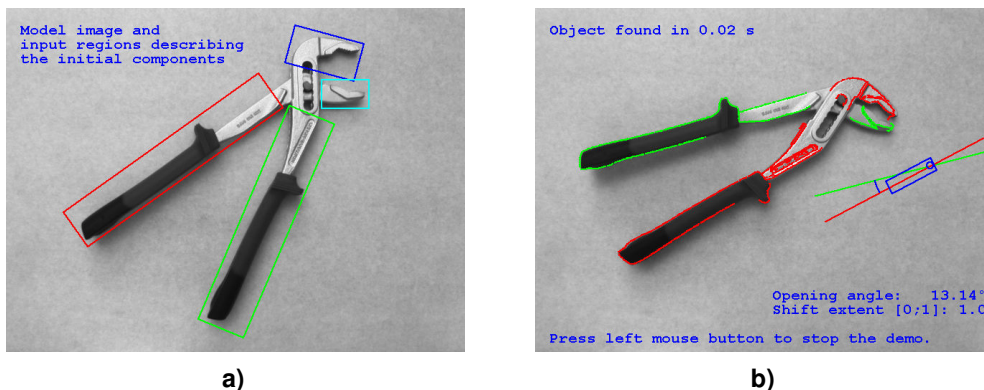


Figure 4.9: (a) Model image with specified ROIs for the components; (b) located pipe wrench in another state.

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

```

read_image (ModelImage, 'pipe_wrench/pipe_wrench_model')
gen_rectangle2 (InitialComponentRegions, 212, 233, 0.62, 167, 29)
gen_rectangle2 (Rectangle2, 298, 363, 1.17, 162, 34)
gen_rectangle2 (Rectangle3, 63, 444, -0.26, 50, 27)
gen_rectangle2 (Rectangle4, 120, 473, 0, 33, 20)
InitialComponentRegions := [InitialComponentRegions,Rectangle2]
InitialComponentRegions := [InitialComponentRegions,Rectangle3]
InitialComponentRegions := [InitialComponentRegions,Rectangle4]
for i := 1 to 4 by 1
    read_image (TrainingImage, 'pipe_wrench/pipe_wrench_training_'+i)
    TrainingImages := [TrainingImages,TrainingImage]
endfor
train_model_components (ModelImage, InitialComponentRegions,
    TrainingImages, ModelComponents, 22, 60, 30, 0.65, -1, -1, rad(60),
    'speed', 'rigidity', 0.2, 0.4, ComponentTrainingID)

```

The components and their relations are displayed as follows:

```

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

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

```

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

```

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

```

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

```

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

```

4.5.4 Other Examples

HDevelop

- [examples\application_guide\1d_metrology\hdevelop\measure_ring.dev](#)
Determine the width of cogs with a circular measure object
→ description in the [Application Note on 1D Metrology](#) on page 22
- [examples\application_guide\shape_matching\hdevelop\multiple_models.dev](#)
Searching for two types of objects simultaneously
→ description in the [Application Note on Shape-Based Matching](#) on page 25
- [examples\application_guide\shape_matching\hdevelop\multiple_objects.dev](#)
Searches for multiple instances of a security ring
→ description in the [Application Note on Shape-Based Matching](#) on page 34
- [examples\application_guide\shape_matching\hdevelop\multiple_scales.dev](#)
Searches for nuts of different sizes
→ description in the [Application Note on Shape-Based Matching](#) on page 44
- [examples\application_guide\shape_matching\hdevelop\reuse_model.dev](#)
Storing and reusing a shape model
→ description in the [Application Note on Shape-Based Matching](#) on page 46
- [examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev](#)
Determine the pose of circles in 3D from their perspective 2D projections
- [examples\hdevelop\Applications\Measure\measure_arc.dev](#)
Measures width of object along circular arc
→ description [here in the Quick Guide](#) on page 197
- [examples\hdevelop\Applications\OCR\engravedt.dev](#)
Segmenting and training characters on a metal surface

- [examples\hdevelop\XLD\Features\fit_ellipse_tooth_rim_xld.dev](#)
Approximating the contour of a tooth rim with an ellipse to find its center.

4.6 Photogrammetry And Remote Sensing

4.6.1 Extracting Forest Features from Color Infrared Image

Example: [examples\hdevelop\Applications\Aerial\forest.dev](#)

The task of this example is to detect different object classes in the color infrared image depicted in [figure 4.10](#): trees (coniferous and deciduous), meadows, and roads.

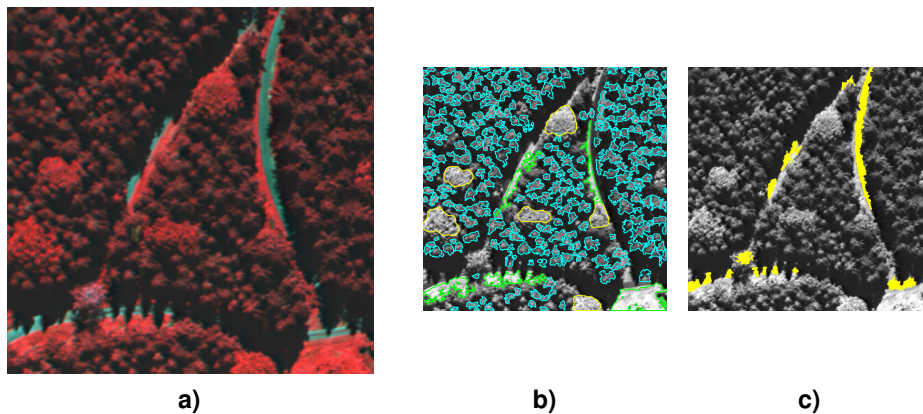


Figure 4.10: (a) Original image; (b) extracted trees and meadows; (c) extracted roads.

The image data is a color infrared image, which allows to extract roads very easily because of their specific color.

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

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


```

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

```

Meadows have similar spectral properties, but are slightly brighter.

```

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

```

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

```

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

```

4.6.2 Segmenting a Color Image

Example: [examples\hdevelop\FILTER\Edges\edges_color.dev](#)

The task of this example is to segment the color image depicted in [figure 4.11](#).

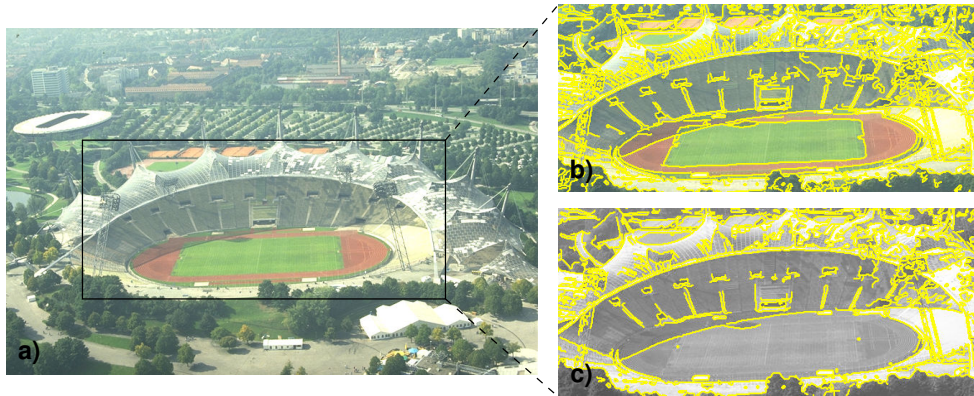


Figure 4.11: (a) Original image; (b) extracted color edges, overlaid on the color image; (c) extracted gray value edges, overlaid on the gray value image.

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

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

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

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

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

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

4.6.3 Extract Roads

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

The task of this example is to extract the roads in the aerial image depicted in [figure 4.12](#)

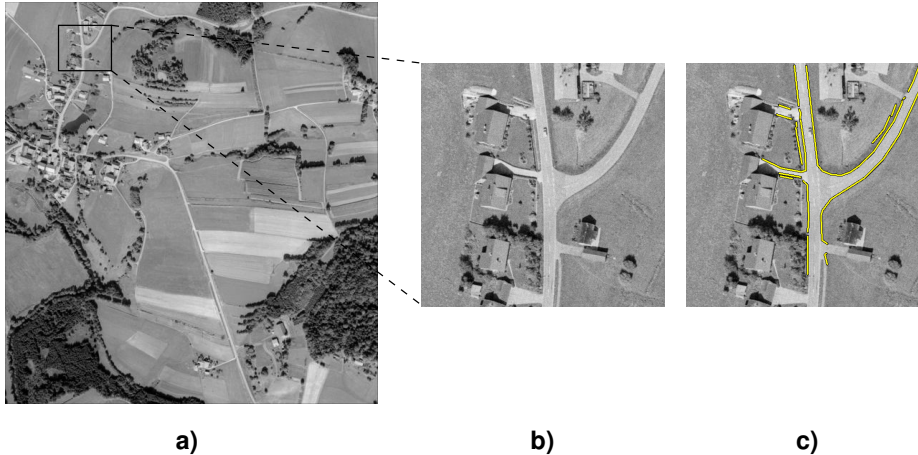


Figure 4.12: (a) Original image; (b) zoomed image part; (c) extracted roads.

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

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

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

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

4.6.4 Other Examples

HDevelop

- `examples\hdevelop\Applications\Aerial\dem.dev`
Extraction of high objects from a digital elevation model
- `examples\hdevelop\Applications\Aerial\dem_trees.dev`
Extraction of trees using texture and a digital elevation model
- `examples\hdevelop\Applications\Aerial\high.dev`
Different methods to extract high objects
- `examples\hdevelop\Applications\Aerial\texture.dev`
Find textured areas (trees and bushes)
- `examples\hdevelop\Manuals\HDevelop\dtm.dev`
Extracts high objects from a digital elevation model

4.7 Printing

4.7.1 Reading a Bar Code

Example: `examples\hdevelop\Applications\Barcode\EAN13AddOn5.dev`

The task of this example is to read the bar code depicted in [figure 4.13](#).

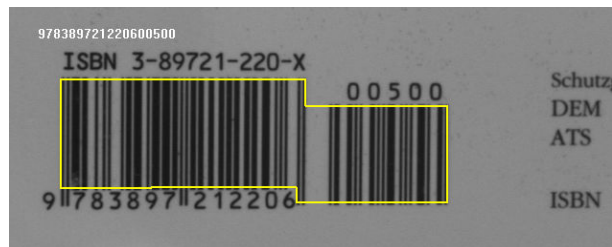


Figure 4.13: Decoded bar code.

This example shows how easily bar codes - even compound bar codes - can be read with HALCON. First, the bar code type is specified.

```
gen_1d_bar_code_descr ('EAN 13 Add-On 5', 13, 13, BarCodeDescr)
```

After reading an image from file, the bar code finder is called. The result of this operator, i.e., the element widths, are then passed to the decoding operator.

```
read_image (image, 'barcode/ean13addon5/ean13addon5'+(i$.2'))
find_1d_bar_code (image, CodeRegion, BarCodeDescr, 'dilation_factor', 2,
    BarcodeFound, BarCode, Orientation)
decode_1d_bar_code (BarCode, BarCodeDescr, Characters, Reference, IsCorrect)
```

4.7.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\grid_rectification_arbitrary_distortion.dev](#)
Determine differences between two printed pages, even if there are distortions in the vertical direction
→ description in the [Application Note on 3D Machine Vision](#) on page 131
- [examples\hdevelop\Applications\OCR\stamp_catalogue.dev](#)
Segmenting and grouping characters on a cluttered page
- [examples\hdevelop\Applications\OCV\adaption_ocv.dev](#)
Analyzes impact of changes on reported character quality
- [examples\hdevelop\Applications\OCV\print_quality.dev](#)
Inspects quality of letter A in different images
- [examples\hdevelop\Tools\OCV\write_ocv.dev](#)
Writing OCV data to file (and reading it in again)

4.8 Rubber, Synthetic Material, Foil

4.8.1 Checking a Boundary for Fins

Example: [examples\hdevelop\Applications\FA\fin.dev](#)

The task of this example is to check the outer boundary of a plastic part. In this case, some objects show fins that are not allowed for faultless pieces (see [figure 4.14](#)).

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

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

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

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

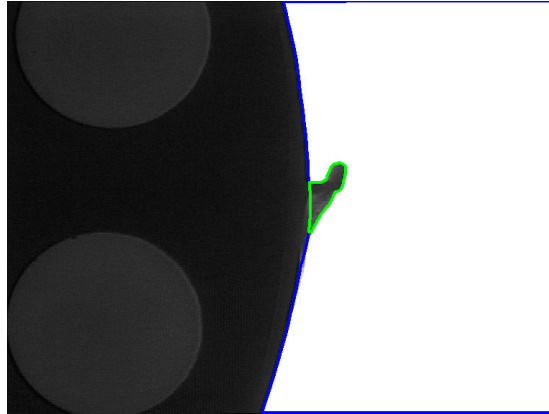


Figure 4.14: Boundary with extracted fin.

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

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

4.8.2 Other Examples

HDevelop

- [examples\hdevelop\Applications\FA\hull.dev](#)
Inspects an injection molded nozzle

C++

- [examples\hdevengine\cpp\source\exec_extproc.cpp](#)
Executes an external HDevelop procedure for fin detection using HDevEngine
→ description in the [Programmer's Guide](#) on page 123
- [examples\hdevengine\cpp\source\exec_program.cpp](#)
Executes an HDevelop program for fin detection using HDevEngine
→ description in the [Programmer's Guide](#) on page 121

Visual Basic

- [examples\hdevengine\vb\ExecExtProc\ExecExtProc.vbp](#)
Executes an external HDevelop procedure for fin detection using HDevEngine
→ description in the [Programmer's Guide](#) on page 138
- [examples\hdevengine\vb\ExecProgram\ExecProgram.vbp](#)
Executes an HDevelop program for fin detection using HDevEngine
→ description in the [Programmer's Guide](#) on page 136

Visual Basic .NET

- `examples\hdevengine\vb.net\ExecExtProc\ExecExtProc.vbproj`
Executes an external HDevelop procedure for fin detection using HDevEngine
→ description in the [Programmer's Guide](#) on page 138
- `examples\hdevengine\vb.net\ExecProgram\ExecProgram.vbproj`
Executes an HDevelop program for fin detection using HDevEngine
→ description in the [Programmer's Guide](#) on page 136

4.9 Semiconductors

4.9.1 Bonding Balls

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

The task of this example is to inspect the diameter of the ball bonds depicted in [figure 4.15](#).

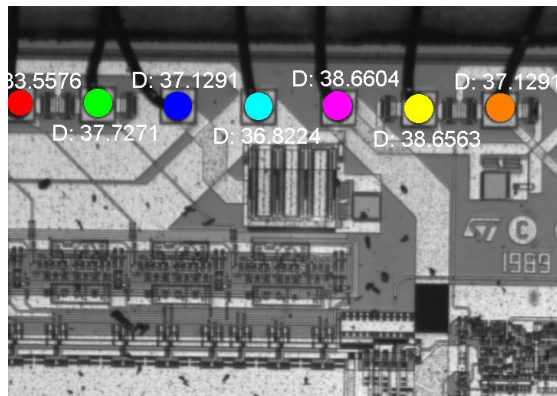


Figure 4.15: Measuring the diameter of ball bonds.

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

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

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

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

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

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

4.9.2 Inspecting an IC Using Fuzzy Measuring

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

The task of this example is to inspect the lead width and the lead distance of the IC depicted in [figure 4.16](#).

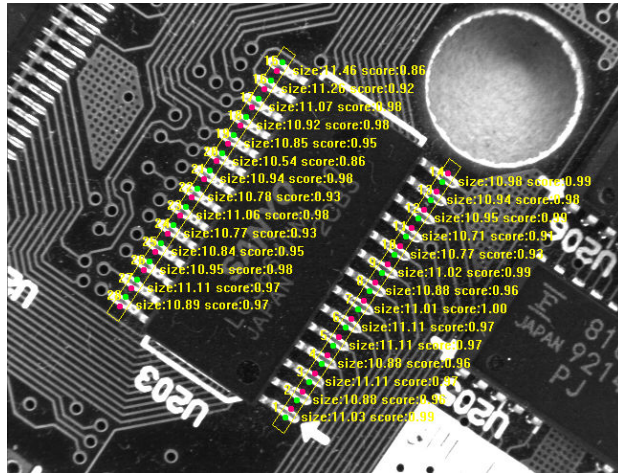


Figure 4.16: Measuring the width and distance of the leads.

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

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


4.9.3 Measuring Leads of a Moving IC

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

The task of this example is to measure the positions of the leads of a chip (see [figure 4.17](#)). Because the chip can appear at varying positions and angles, the regions of interest used for the measurement must be aligned.

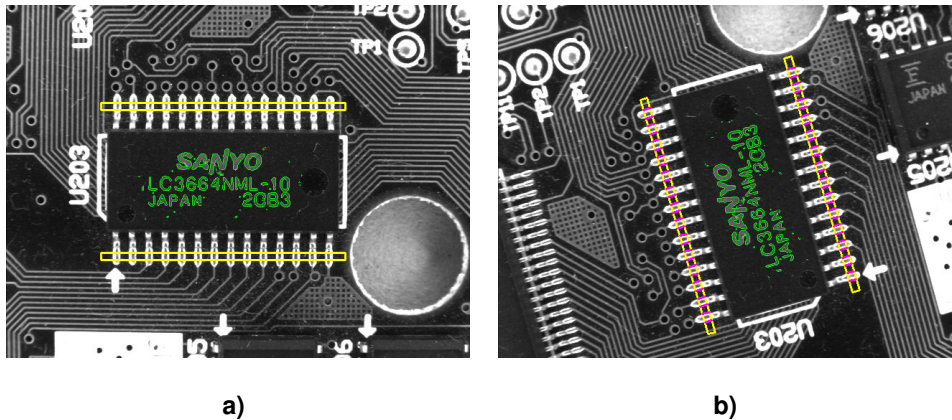


Figure 4.17: (a) Model image with measurement ROIs; (b) measuring the leads in the aligned ROIs.

In this case, the alignment is achieved by searching for the print on the chip using the shape-based matching.

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

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

```
find_shape_model (ImageCheck, ModelID, 0, rad(360), 0.7, 1, 0.5,
    'least_squares', 4, 0.9, RowCheck, ColumnCheck, AngleCheck, Score)

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

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

```

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

```

4.9.4 Creating a Mosaic Image

Example: [examples\hdevelop\Tools\2D-Transformations\gen_projective_mosaic.dev](#)

The task of this example is to create an image of the elongated printed circuit board depicted [figure 4.18](#). Using standard image sizes, most of the image would be empty. The solution is to acquire images from multiple viewpoints and then to create a mosaic image.

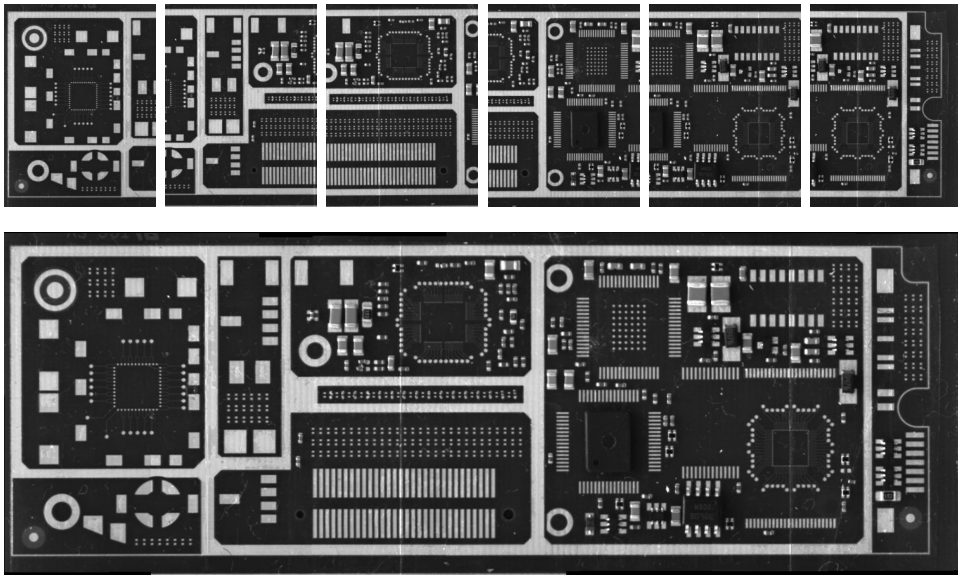


Figure 4.18: Creating a mosaic image from multiple overlapping images.

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

```

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

```

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

```

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

```

4.9.5 Locating Board Components by Color

Example: [examples\hdevelop\Applications\FA\ic.dev](#)

The task of this example is to locate all components on the printed circuit board depicted in [figure 4.19](#).

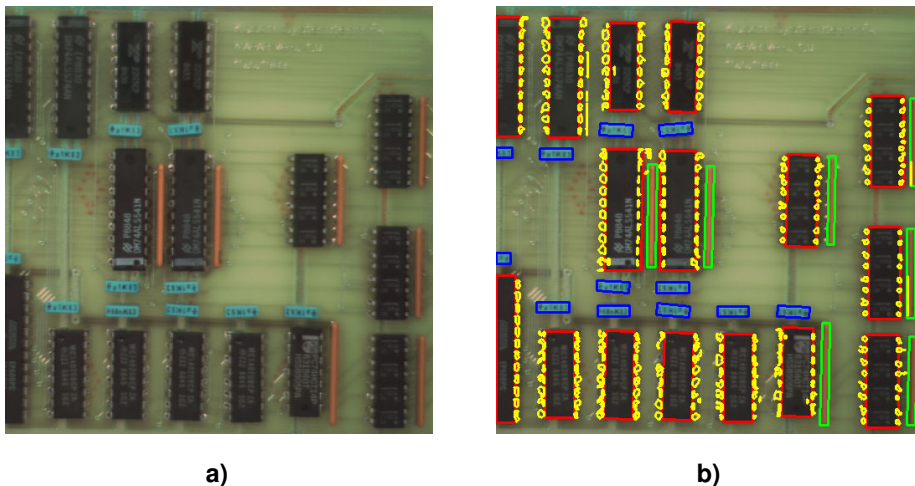


Figure 4.19: (a) Original image; (b) extracted ICs, resistors, and capacitors.

The input data is a color image, which allows locating components like capacitors and resistors very easily by their significant color: Using a color space transformation, the hue values allow the selection of the corresponding components. The following code extracts the resistors; the extraction of the capacitors is performed along the same lines.

```

decompose3 (Image, Red, Green, Blue)
trans_from_rgb (Red, Green, Blue, Hue, Saturation, Intensity, 'hsv')
threshold (Saturation, Colored, 100, 255)
reduce_domain (Hue, Colored, HueColored)
threshold (HueColored, Red, 10, 19)
connection (Red, RedConnect)
select_shape (RedConnect, RedLarge, 'area', 'and', 150.000000, 99999.000000)
shape_trans (RedLarge, Resistors, 'rectangle2')

```

The extraction of the ICs is more difficult because of the bright imprints, which do not allow a simple thresholding in one step. Instead of this, dark areas are selected first, which are then combined using a dilation.

```

threshold (Intensity, Dark, 0, 50)
dilation_rectangle1 (Dark, DarkDilation, 14, 14)
connection (DarkDilation, ICLarge)

```

After this, the segmentation is repeated inside the thus extracted connected components.

```

add_channels (ICLarge, Intensity, ICLargeGray)
threshold (ICLargeGray, ICDark, 0, 50)
shape_trans (ICDark, IC, 'rectangle2')

```

To locate the contact points, small ROIs are generated on the left and right side of each IC.

```

dilation_rectangle1 (IC, ICDilation, 5, 1)
difference (ICDilation, IC, SearchSpace)
dilation_rectangle1 (SearchSpace, SearchSpaceDilation, 14, 1)
union1 (SearchSpaceDilation, SearchSpaceUnion)

```

Inside these areas, locally bright spots are detected.

```

reduce_domain (Intensity, SearchSpaceUnion, SearchGray)
mean_image (SearchGray, SearchMean, 15, 15)
dyn_threshold (SearchGray, SearchMean, PinsRaw, 5.000000, 'light')
connection (PinsRaw, PinsConnect)
fill_up (PinsConnect, PinsFilled)
select_shape (PinsFilled, Pins, 'area', 'and', 10, 100)

```

4.9.6 Other Examples

HDevelop

- [examples\application_guide\1d_metrology\hdevelop\measure_ic_leads.dev](#)
Measures leads of an IC
→ description in the [Application Note on 1D Metrology](#) on page 11

- [examples\application_guide\3d_machine_vision\hdevelop\bundle_adjusted_mosaicking.dev](#)
Uses bundle adjusted mosaicking to merge partial images of a BGA into one large image
→ description in the [Application Note on 3D Machine Vision](#) on page 82
- [examples\application_guide\3d_machine_vision\hdevelop\height_above_reference_plane_from_stereo.dev](#)
Extracts chips using height information from binocular stereo
→ description in the [Application Note on 3D Machine Vision](#) on page 100
- [examples\application_guide\3d_machine_vision\hdevelop\mosaicking.dev](#)
Uses mosaicking to merge partial images of a BGA into one large image
→ description in the [Application Note on 3D Machine Vision](#) on page 70
- [examples\application_guide\shape_matching\hdevelop\first_example_shape_matching.dev](#)
Introduces HALCON's shape-based matching
→ description in the [Application Note on Shape-Based Matching](#) on page 4
- [examples\application_guide\shape_matching\hdevelop\process_shape_model.dev](#)
Creates a model ROI by modifying the result of `inspect_shape_model`
→ description in the [Application Note on Shape-Based Matching](#) on page 10
- [examples\application_guide\shape_matching\hdevelop\synthetic_circle.dev](#)
Uses a synthetic model (circle) to search for capacitors on a board
→ description in the [Application Note on Shape-Based Matching](#) on page 18
- [examples\hdevelop\Applications\Datacode\pdf417_optimized.dev](#)
Optimizing parameters for reading of 2d data codes of type PDF417
- [examples\hdevelop\Applications\Datacode\pdf417_simple.dev](#)
Reading 2d data codes of type PDF417
- [examples\hdevelop\Applications\FA\ball_seq.dev](#)
Inspection of ball bonding (multiple images)
- [examples\hdevelop\Applications\FA\board.dev](#)
Detection of missing solder
- [examples\hdevelop\Applications\FA\cbm_dip_switch.dev](#)
Locates dip switches and tests their state using component-based matching
→ description [here in the Quick Guide](#) on page 190
- [examples\hdevelop\Applications\Measure\measure_pin.dev](#)
Measures pins of an IC
→ description [here in the Quick Guide](#) on page 226
- [examples\hdevelop\Applications\Stereo\board_components.dev](#)
Segments board components by height using binocular stereo
→ description [here in the Quick Guide](#) on page 166
- [examples\hdevelop\Applications\Stereo\board_segmentation_uncalib.dev](#)
Segmentation of board components by height using uncalibrated binocular stereo

- [examples\hdevelop\Filter\Inpainting\harmonic_interpolation.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_aniso.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_ced.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_mcf.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Manuals\HDevelop\ball.dev](#)
Inspects bonding of balls
→ description in the [HDevelop User's Manual](#) on page 141
- [examples\hdevelop\Manuals\HDevelop\ic.dev](#)
Combining different segmentation methods
→ description in the [HDevelop User's Manual](#) on page 144
- [examples\hdevelop\Matching\Component-Based\cbm_modules_simple.dev](#)
Locates modules on a board using component-based matching
→ description [here in the Quick Guide](#) on page 235
- [examples\hdevelop\Matching\Component-Based\cbm_sbm.dev](#)
Compares component-based matching to shape-based matching
- [examples\hdevelop\Morphology\Gray-Values\pcb_inspection.dev](#)
Finds defects on a PCB using gray value morphology
- [examples\hdevelop\Regions\Features\rectangularity.dev](#)
Calculating the rectangularity of regions
- [examples\hdevelop\Segmentation\Classification\class_2dim_sup.dev](#)
Segmenting an image using two-dimensional pixel classification
- [examples\hdevelop\Segmentation\Classification\class_ndim_box.dev](#)
Classifying pixels using hyper-cuboids
- [examples\hdevelop\Segmentation\Classification\class_ndim_norm.dev](#)
Classifying pixels using hyper-spheres
- [examples\hdevelop\XLD\Transformation\union_contours_xld.dev](#)
Connecting collinear line segments

C++

- [examples\cpp\source\fuzzy_measure_pin.cpp](#)
Measures pins of an IC using fuzzy measure
- [examples\mfc\Matching\Matching.cpp](#)
Locating an IC using HALCON/C++ and MFC, creating a HALCON window
- [examples\mfc\MatchingCOM\Matching.cpp](#)
Locating an IC using HALCON/COM and MFC

- `examples\mfc\MatchingExtWin\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, painting into an existing window
- `examples\motif\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Motif
- `examples\qt\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Qt

Visual Basic

- `examples\vb\Tools\Matching\matching.vbp`
Locates an IC on a board and measures pin distances
- `examples\vb\Tools\Measure\measure.vbp`
Measuring pins with interactive control of parameters

Visual Basic .NET

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

C#

- `examples\c#\Matching\Matching.csproj`
Locates an IC on a board and measures pin distances

C

- `examples\c\example_multithreaded1.c`
Using multiple threads with Parallel HALCON

Delphi

- `examples\delphi\Matching\matching.dpr`
Locates an IC on a board and measures pin distances

Chapter 5

Application Areas

This chapter describes example programs from various application areas.

5.1	1D Bar Codes	222
5.2	2D Data Codes	223
5.3	Completeness Check	224
5.4	Measuring And Comparison 2D	226
5.5	Measuring And Comparison 3D	231
5.6	Optical Character Recognition	234
5.7	Position Recognition 2D	235
5.8	Print Inspection	237
5.9	Object Recognition 2D	239
5.10	Surface Inspection	241

5.1 1D Bar Codes

5.1.1 Reading Multiple Bar Codes on a Toner Cartridge

Example: [examples\hdevelop\Applications\Barcode\multiple.dev](#)

The task of this example is to read the two bar codes depicted in [figure 5.1](#).



Figure 5.1: Detected bar codes.

In this example, the Code 39 is used, which is very popular on packages and other kind of product descriptions. The task is solved by calling the HALCON operator that returns all bar codes of a given type within the image. For each of the found regions, the decoding is called separately.

```
gen_1d_bar_code_descr ('code 39', 6, 12, BarCodeDescr)
find_1d_bar_code_region (image, CodeRegion, BarCodeDescr, [], [],
    Orientation)
for i := 1 to Number by 1
    ObjectSelected := CodeRegion[i]
    reduce_domain (image, ObjectSelected, ImageReduced)
    get_1d_bar_code (ImageReduced, BarCodeDescr, [], [], Orientation[i-1],
        BarCode)
    decode_1d_bar_code (BarCode, BarCodeDescr, Characters, Reference,
        IsCorrect)
endfor
```

5.1.2 Other Examples

Please refer to the list of examples for the method [1D Bar Code](#) (see section [3.10.4.2](#) on page [131](#)).

5.2 2D Data Codes

5.2.1 Reading 2D Data Codes on Chips

Example: `examples\hdevelop\Applications\Datacode\ecc200_optimized.dev`

This example program reads 2D data codes (type ECC200) engraved in chips (see [figure 5.2](#)).



Figure 5.2: Decoded data code.

The example shows how to set optimized parameters for efficient data code reading. The code printed on a chip is always light on dark in this application and has a given size and number of modules. Also, the contrast is within a predefined range. By specifying these values for the model, the execution time can be sped up significantly.

```
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DataCodeHandle)
set_data_code_2d_param (DataCodeHandle,
    ['module_size_min', 'module_size_max'], [4,7])
set_data_code_2d_param (DataCodeHandle, 'module_gap', 'no')
set_data_code_2d_param (DataCodeHandle, 'polarity', 'light_on_dark')
set_data_code_2d_param (DataCodeHandle, 'mirrored', 'no')
set_data_code_2d_param (DataCodeHandle, 'contrast_min', 10)
set_data_code_2d_param (DataCodeHandle, 'symbol_size', 18)

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

5.2.2 Other Examples

Please refer to the list of examples for the method [2D Data Code](#) (see section [3.11.4.2](#) on page [140](#)).

5.3 Completeness Check

5.3.1 Inspect Razor Blades

Example: [examples\application_guide\shape_matching\hdevelop\align_measurements.dev](#)

The task of this example is to check the razor blades depicted in [figure 5.3](#). The program uses shape-based matching to locate all blades and then uses metrology to inspect the blades. Missing parts are detected and displayed. This example is described in more detail in the [Application Note on Shape-Based Matching](#) on page 36.

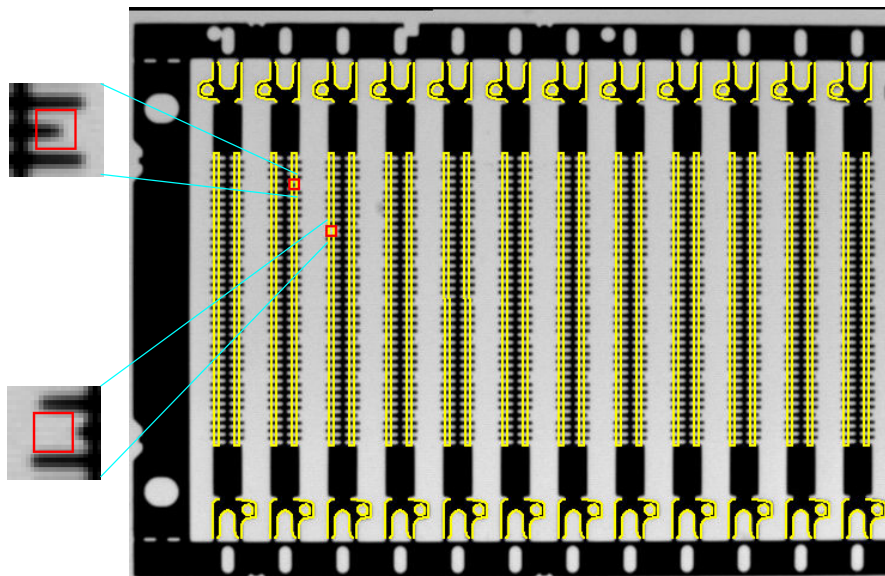


Figure 5.3: Inspecting razor blades.

5.3.2 Other Examples

HDevelop

- [examples\hdevelop\Applications\FA\ball.dev](#)
Inspection of ball bonding
→ description [here in the Quick Guide](#) on page 211
- [examples\hdevelop\Applications\FA\ball_seq.dev](#)
Inspection of ball bonding (multiple images)
- [examples\hdevelop\Applications\FA\board.dev](#)
Detection of missing solder

- `examples\hdevelop\Applications\FA\ic.dev`
Extracts resistors, capacitors and ICs from board using color information
→ description [here in the Quick Guide](#) on page 215
- `examples\hdevelop\Applications\FA\pm_measure_board.dev`
Locates IC on a board and measures pin distances
→ description [here in the Quick Guide](#) on page 213
- `examples\hdevelop\Manuals\HDevelop\ball.dev`
Inspects bonding of balls
→ description in the [HDevelop User's Manual](#) on page 141
- `examples\hdevelop\Manuals\HDevelop\ic.dev`
Combining different segmentation methods
→ description in the [HDevelop User's Manual](#) on page 144
- `examples\hdevelop\Matching\Component-Based\cbm_modules_simple.dev`
Locates modules on a board using component-based matching
→ description [here in the Quick Guide](#) on page 235
- `examples\quick_guide\hdevelop\color_pieces.dev`
Completeness check of colored game pieces using MLP classification
→ description [here in the Quick Guide](#) on page 118
- `examples\quick_guide\hdevelop\color_pieces_euclid.dev`
Completeness check of game color pieces using Euclidean classification

C++

- `examples\mfc\Matching\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, creating a HALCON window
- `examples\mfc\MatchingCOM\Matching.cpp`
Locating an IC using HALCON/COM and MFC
- `examples\mfc\MatchingExtWin\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, painting into an existing window
- `examples\motif\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Motif
- `examples\qt\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Qt

Visual Basic

- `examples\vb\Tools\Matching\matching.vbp`
Locates an IC on a board and measures pin distances

Visual Basic .NET

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

C#

- `examples\c#\Matching\Matching.csproj`
Locates an IC on a board and measures pin distances

C

- `examples\c\example_multithreaded1.c`
Using multiple threads with Parallel HALCON

Delphi

- `examples\delphi\Matching\matching.dpr`
Locates an IC on a board and measures pin distances

5.4 Measuring And Comparison 2D

5.4.1 Inspect IC

Example: [examples\hdevelop\Applications\Measure\measure_pin.dev](#)

The task of this example is to inspect major dimensions of an IC (see [figure 5.4](#)).

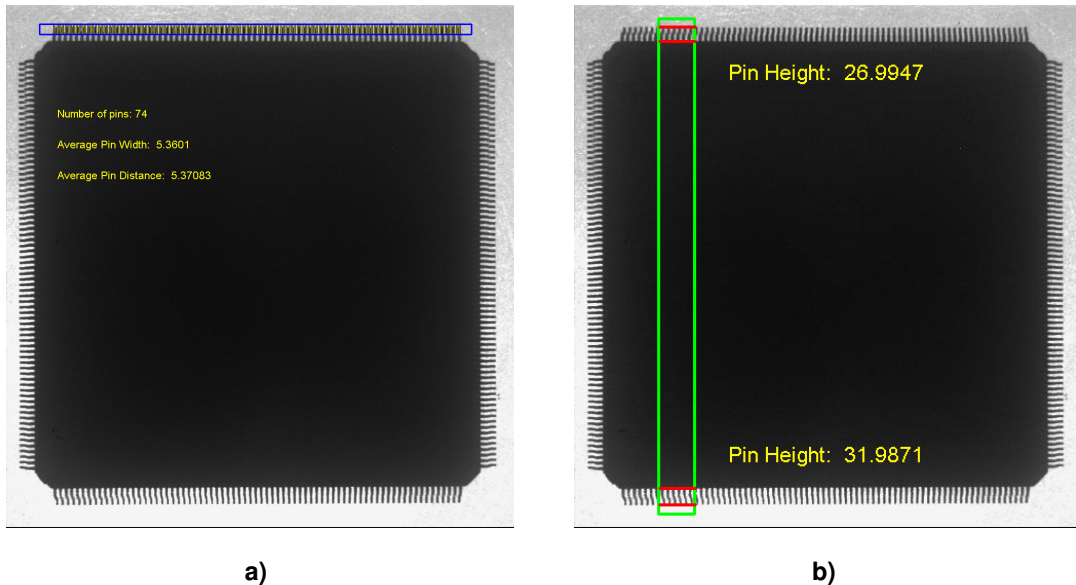


Figure 5.4: Measuring the dimensions of leads: (a) width of the leads and distance between them; (b) length of the leads.

In the first step the extent of each lead and - more importantly - the distance between the leads is measured. For this, a rectangle that contains the leads is defined (see [figure 5.4a](#)), which is used to generate

the measure object is generated. It is used to extract pairs of straight edges that lie perpendicular to the major axis of the rectangle.

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

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

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

The second part shows the power of the measure tool: The length of the leads are determined. This is possible although each lead has a width of only a few pixels. For this, a new measure object is generated based on a rectangle that contains the leads on two opposite sides of the IC (see [figure 5.4b](#)). The distance between the first and the second found edge is the length of the upper leads, and the distance between the third and the fourth edge is the length of the lower leads.

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

5.4.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_exterior.dev](#)
Measures positions on a caliper rule using camera calibration
→ description in the [Application Note on 3D Machine Vision](#) on page 41
- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_multi_image.dev](#)
Calibrates the camera and measures positions on a caliper rule
→ description in the [Application Note on 3D Machine Vision](#) on page 39
- [examples\application_guide\3d_machine_vision\hdevelop\height_displacement.dev](#)
Calculating height displacements (for known heights)
→ description in the [Application Note on 3D Machine Vision](#) on page 57
- [examples\application_guide\shape_matching\hdevelop\align_measurements.dev](#)
Inspects individual razor blades using shape-based matching to align ROIs for the measure tool
→ description in the [Application Note on Shape-Based Matching](#) on page 36

- [examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev](#)
Measures distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- [examples\hdevelop\Applications\FA\circles.dev](#)
Fits circles into curved contour segments
→ description [here in the Quick Guide](#) on page 200
- [examples\hdevelop\Applications\FA\clip.dev](#)
Determines the orientation of clips
- [examples\hdevelop\Applications\FA\holes.dev](#)
Extracts positions and radii of holes
- [examples\hdevelop\Applications\FA\hull.dev](#)
Inspects an injection molded nozzle
- [examples\hdevelop\Applications\FA\pm_measure_board.dev](#)
Locates IC on a board and measures pin distances
→ description [here in the Quick Guide](#) on page 213
- [examples\hdevelop\Applications\Measure\fuzzy_measure_pin.dev](#)
Measures pins of an IC using fuzzy measure
→ description [here in the Quick Guide](#) on page 212
- [examples\hdevelop\Applications\Measure\measure_arc.dev](#)
Measures width of object along circular arc
→ description [here in the Quick Guide](#) on page 197
- [examples\hdevelop\Applications\Measure\measure_online.dev](#)
Measures your object in a live image
- [examples\hdevelop\Applications\Medicine\angio.dev](#)
Extracts blood vessels and their diameters from an angiogram
- [examples\hdevelop\Applications\Medicine\particle.dev](#)
Extracts particles of varying sizes
→ description [here in the Quick Guide](#) on page 194
- [examples\hdevelop\Applications\Medicine\vessel.dev](#)
Segmentation and measurement of a blood vessel
- [examples\hdevelop\Filter\Lines\lines_color.dev](#)
Extracting lines using color information
→ description [here in the Quick Guide](#) on page 191
- [examples\hdevelop\Image\Features\area_center_gray.dev](#)
Analyzes the accuracy of calculating the gray area and center of gravity
- [examples\hdevelop\Image\Features\elliptic_axis_gray.dev](#)
Analyzes the accuracy of calculating the gray value moments (elliptic_axis_gray)
- [examples\hdevelop\Manuals\HDevelop\particle.dev](#)
Measures small particles

→ description in the [HDevelop User's Manual](#) on page 136

- [examples\hdevelop\Manuals\HDevelop\wood.dev](#)
Determines the age of a tree by counting its annual rings
→ description in the [HDevelop User's Manual](#) on page 139
- [examples\hdevelop\Manuals\HDevelop\wood_cells.dev](#)
Analyzes the cell density during a tree's growth
→ description in the [HDevelop User's Manual](#) on page 147
- [examples\hdevelop\Segmentation\Topography\saddle_points_sub_pix.dev](#)
Detecting saddle points with subpixel accuracy
- [examples\hdevelop\Tools\Calibration\camera_calibration.dev](#)
Determining camera parameters using a special calibration plate
- [examples\hdevelop\Tools\Geometry\angle_ll.dev](#)
Calculating the angle between two lines
- [examples\hdevelop\Tools\Geometry\angle_lx.dev](#)
Calculating the angle between a line and the vertical axis
- [examples\hdevelop\Tools\Geometry\distance_cc_min.dev](#)
Calculating the distance between two contours
- [examples\hdevelop\Tools\Geometry\distance_lr.dev](#)
Calculating the distance between a line and a region
- [examples\hdevelop\Tools\Geometry\distance_pc.dev](#)
Calculating the distance between a point and a contour
- [examples\hdevelop\Tools\Geometry\distance_pl.dev](#)
Calculating the distances between points and a line
- [examples\hdevelop\Tools\Geometry\distance_pp.dev](#)
Calculating the distance between two points
- [examples\hdevelop\Tools\Geometry\distance_pr.dev](#)
Calculating the distance between a point and a region
- [examples\hdevelop\Tools\Geometry\distance_ps.dev](#)
Calculating the distance between a point and a line segment
- [examples\hdevelop\Tools\Geometry\distance_sl.dev](#)
Calculating the distance between a line segment and a line
- [examples\hdevelop\Tools\Geometry\distance_sr.dev](#)
Calculating the distance between a line segment and a region
- [examples\hdevelop\Tools\Geometry\distance_ss.dev](#)
Calculating the distances between line segments
- [examples\hdevelop\Tools\Measure\gen_measure_arc.dev](#)
Measuring edges perpendicular to a given arc

- `examples\hdevelop\Tools\Measure\gen_measure_rectangle2.dev`
Measuring edges perpendicular to a given line
- `examples\hdevelop\XLD\Features\fit_ellipse_tooth_rim_xld.dev`
Approximating the contour of a tooth rim with an ellipse to find its center.
- `examples\quick_guide\hdevelop\atoms.dev`
Locates irregularities in an atomic grid structure
→ description [here in the Quick Guide](#) on page 45
- `examples\quick_guide\hdevelop\critical_points.dev`
Locates saddle point markers in an image
→ description [here in the Quick Guide](#) on page 31
- `examples\quick_guide\hdevelop\crystal.dev`
Extracts hexagonally shaped crystals via local thresholding and region post-processing
→ description [here in the Quick Guide](#) on page 44
- `examples\quick_guide\hdevelop\fuse.dev`
Measures the thickness of a fuse wire
→ description [here in the Quick Guide](#) on page 57
- `examples\quick_guide\hdevelop\measure_metal_part.dev`
Inspects metal part by fitting lines and circles
→ description [here in the Quick Guide](#) on page 90

C++

- `examples\cpp\source\example5.cpp`
Analyzes the distribution of cell sizes
- `examples\cpp\source\fuzzy_measure_pin.cpp`
Measures pins of an IC using fuzzy measure
- `examples\mfc\Matching\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, creating a HALCON window
- `examples\mfc\MatchingCOM\Matching.cpp`
Locating an IC using HALCON/COM and MFC
- `examples\mfc\MatchingExtWin\Matching.cpp`
Locating an IC using HALCON/C++ and MFC, painting into an existing window
- `examples\motif\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Motif
- `examples\qt\Matching\matching.cpp`
Locating an IC using HALCON/C++ and Qt

Visual Basic

- `examples\vb\Online\Measure\measure.vbp`
Measuring edge positions in a live image

- `examples\vb\Tools\Matching\matching.vbp`
Locates an IC on a board and measures pin distances
- `examples\vb\Tools\Measure\measure.vbp`
Measuring pins with interactive control of parameters

Visual Basic .NET

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

C#

- `examples\c#\Matching\Matching.csproj`
Locates an IC on a board and measures pin distances

Delphi

- `examples\delphi\Matching\matching.dpr`
Locates an IC on a board and measures pin distances

5.5 Measuring And Comparison 3D

5.5.1 Inspect IC

Example: [examples\hdevelop\Applications\DFP\resistor.dev](#)

The task of this example is to inspect a soldered component using depth-from-focus (see [figure 5.5](#)).

First, a sequence of images acquired with continuously varying focus ([figure 5.5a](#)) is read and stored into a multi-channel image.

```
for i := 1 to 10 by 1
  Names := [Names, 'dff/focus_' + (i$.2')]
endfor
read_image (Image, Names)
```

Then, a depth image is derived with the depth-from-focus approach.

```
depth_from_focus (Image, Depth, Confidence, 'highpass', 'next_maximum')
```

From the smoothed depth image ([figure 5.5b](#)), the resistor and the solder can be segmented easily by thresholding. The extracted outlines of the resistor and of the solder are overlaid to one of the input images in [figure 5.5c](#).

```
threshold (DepthMean, Resistor, 158, 255)
threshold (DepthMean, Solder, 125, 158)
```

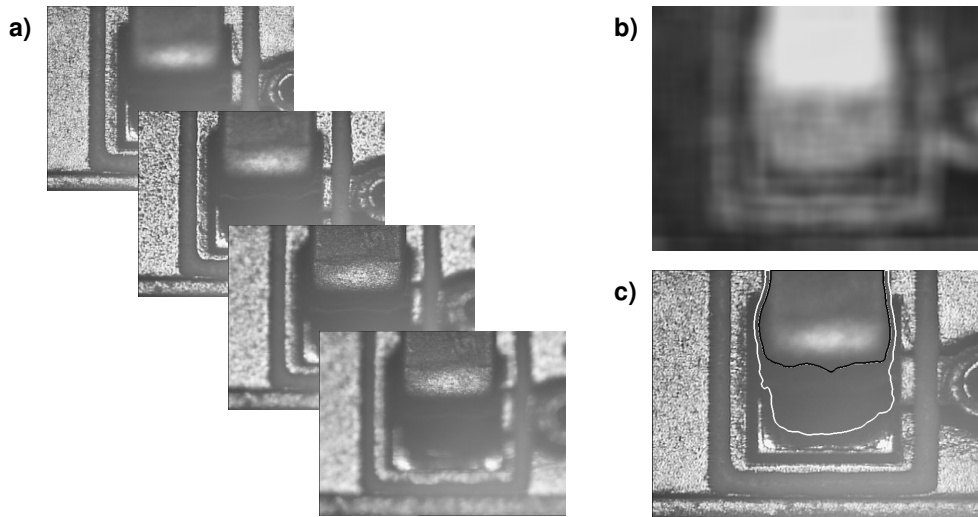


Figure 5.5: Depth-from-focus: (a) sequence of input images; (b) depth image; (c) segmentation of components.

5.5.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\3d_information_for_selected_points.dev](#)
 Calculating world coordinates for a point in a stereo image pair
 → description in the [Application Note on 3D Machine Vision](#) on page 106
- [examples\application_guide\3d_machine_vision\hdevelop\camera_calibration_exterior.dev](#)
 Measures positions on a caliper rule using camera calibration
 → description in the [Application Note on 3D Machine Vision](#) on page 41
- [examples\application_guide\3d_machine_vision\hdevelop\height_above_reference_plane_from_stereo.dev](#)
 Extracts chips using height information from binocular stereo
 → description in the [Application Note on 3D Machine Vision](#) on page 100
- [examples\application_guide\3d_machine_vision\hdevelop\height_displacement.dev](#)
 Calculating height displacements (for known heights)
 → description in the [Application Note on 3D Machine Vision](#) on page 57
- [examples\application_guide\3d_machine_vision\hdevelop\pose_of_known_3d_object.dev](#)
 Determines pose of object by extracting three or more reference points
 → description in the [Application Note on 3D Machine Vision](#) on page 84

- [examples\hdevelop\Applications\Aerial\dem.dev](#)
Extraction of high objects from a digital elevation model
- [examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev](#)
Determine the pose of circles in 3D from their perspective 2D projections
- [examples\hdevelop\Applications\Calibration\world_coordinates_line_scan.dev](#)
Measures distances between the pitch lines of a caliper rule in a line scan image using camera calibration
- [examples\hdevelop\Applications\FA\cbm_caliper.dev](#)
Measures the setting of a caliper using component-based matching in a perspective distorted image
- [examples\hdevelop\Applications\Stereo\board_components.dev](#)
Segments board components by height using binocular stereo
→ description [here in the Quick Guide](#) on page 166
- [examples\hdevelop\Applications\Stereo\board_segmentation_uncalib.dev](#)
Segmentation of board components by height using uncalibrated binocular stereo
- [examples\hdevelop\Applications\Stereo\disparity.dev](#)
Shows disparity results for several stereo image pairs
- [examples\hdevelop\Filter\Inpainting\harmonic_interpolation.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_aniso.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_ced.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Filter\Inpainting\inpainting_mcf.dev](#)
Fill up unreconstructed areas in a distance image created by stereo reconstruction
- [examples\hdevelop\Manuals\HDevelop\dtm.dev](#)
Extracts high objects from a digital elevation model
- [examples\hdevelop\Tools\Calibration\get_circle_pose.dev](#)
Determine the pose of circles in 3D from their perspective 2D projections
- [examples\hdevelop\Tools\Shape-from\depth_from_focus.dev](#)
Extracting depth using multiple focus levels
- [examples\hdevelop\Tools\Shape-from\phot_stereo.dev](#)
Reconstructing a surface by illuminating it from three different directions
- [examples\hdevelop\Tools\Stereo\binocular_disparity.dev](#)
Calculating disparities from epipolar image pairs
- [examples\hdevelop\Tools\Stereo\uncalib_stereo_boxes.dev](#)
Determine the surfaces of boxes using uncalibrated binocular stereo.

Visual Basic

- `examples\vb\Tools\Calibration\calibration.vbp`
Performing a camera calibration

5.6 Optical Character Recognition

5.6.1 Reading Forms

Example: `examples\hdevelop\Applications\OCR\ocrcolor.dev`

The task of this example is to extract the symbols in the form. A typical problem is that the symbols are not printed in the correct place, as depicted in [figure 5.6](#).

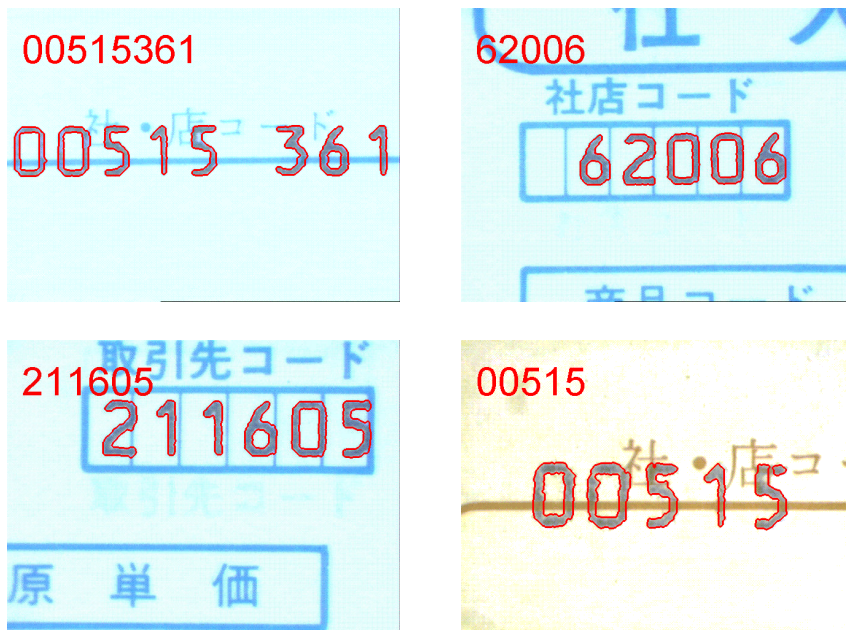


Figure 5.6: Example images for OCR.

To solve the problem of numbers printed on lines, color is used here: The hue value of the characters differs from the hue of the form. The color classification method is a very simple way to save execution time: In contrast to more difficult color processing problems, here it is sufficient to consider the difference of the red and green channel combined with the intensity.

```

threshold (Green, ForegroundRaw, 0, 220)
sub_image (RedReduced, GreenReduced, ImageSub, 2, 128)
mean_image (ImageSub, ImageMean, 3, 3)
bin_threshold (ImageMean, Cluster1)
difference (Foreground, Cluster1, Cluster2)
Cluster := [Cluster1, Cluster2]
opening_circle (Cluster, Opening, 2.5)

```

The selected pixels are grouped and post-processed with morphological operators.

```

closing_rectangle1 (NumberRegion, NumberCand, 1, 20)
difference (Image, NumberCand, NoNumbers)
connection (NumberRegion, NumberParts)
intensity (NumberParts, Green, MeanIntensity, _)
expand_gray_ref (NumberParts, Green, NoNumbers, Numbers, 20, 'image',
                MeanIntensity, 48)
union1 (Numbers, NumberRegion)
connection (NumberRegion, Numbers)

```

For reading, it is important not to use the gray values of the background because of the changes in color. To solve this, only region features are used for the font.

```

paint_region (NoNumbers, Green, ImageOCRRaw, 255, 'fill')
paint_region (NumberRegion, ImageOCRRaw, ImageOCR, 0, 'fill')
do_ocr_multi_class_mlp (FinalNumbers, ImageOCR, OCRHandle, RecChar,
                       Confidence)

```

5.6.2 Other Examples

Please refer to the list of examples for the method [OCR](#) (see section [3.12.4.4](#) on page [154](#)).

5.7 Position Recognition 2D

5.7.1 Locate Components on a PCB

Example: [examples\hdevelop\Matching\Component-Based\cbm_modules_simple.dev](#)

The task of this example is to locate multiple components on a printed circuit board in one step (see [figure 5.7](#)). On a printed circuit board, typically multiple different objects are mounted, whose positions can vary because of the tolerances in the mounting process. To locate all objects quickly and in a robust manner, the component-based matching is used.

In the training step, each object is marked with a region of interest ([figure 5.7a](#)). The possible movement and rotation of the objects is known from the manufacturing process and is passed as a parameter to the training.

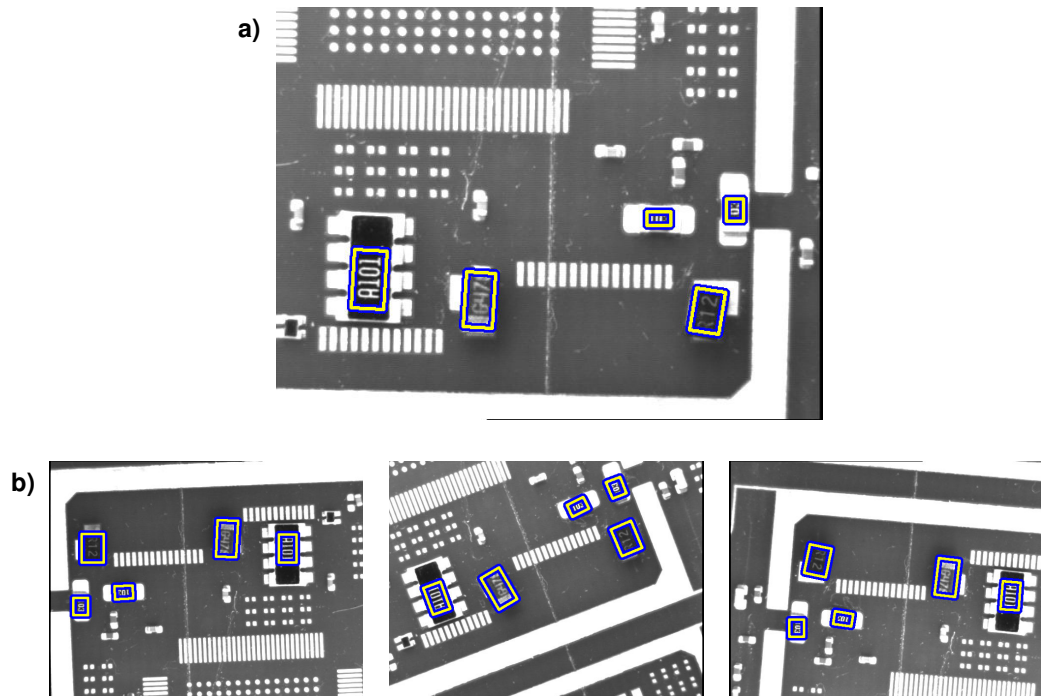


Figure 5.7: Component-based matching: (a) training objects; (b) objects, located in different images where the relation of the objects with respect to each other varies.

```

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

```

Now, the component-based matching is able to find all objects in one step (figure 5.7a), returning the relative positions of each object.

```

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

```


5.8 Print Inspection

5.8.1 Inspect a Printed Logo

Example: [examples\hdevelop\Applications\FA\print_check.dev](#)

The task of this example is to inspect the logo printed on the pen clip depicted in [figure 5.8](#). As an additional difficulty, the pens move from image to image.

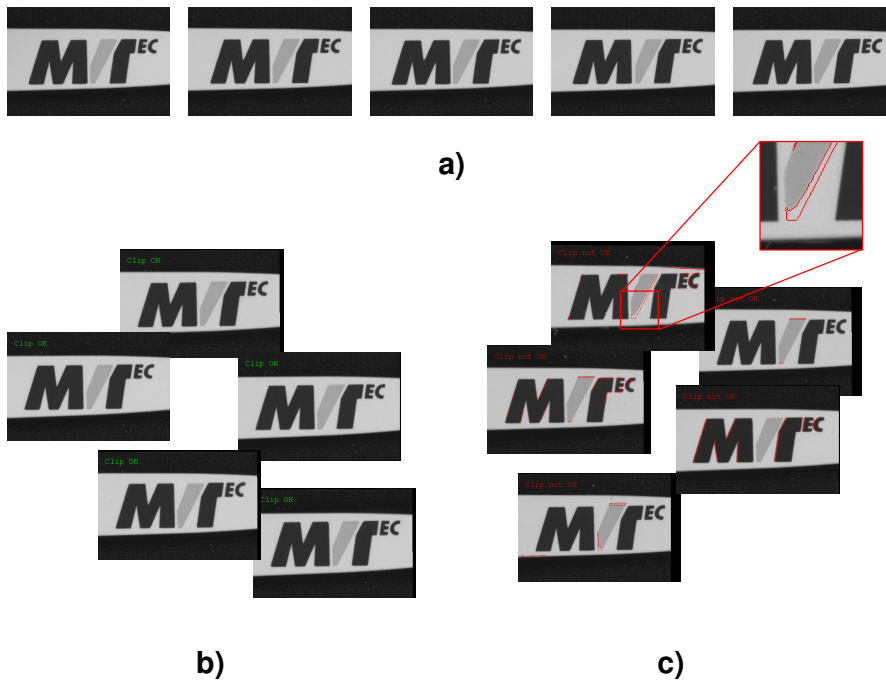


Figure 5.8: Inspection of the logo that is printed on the pen clip: (a) training images; (b) accepted images; (c) rejected images.

To solve this task, a variation model is generated and trained with multiple images. For the training of the variation model, the matching is used to locate the logo and to align the image appropriately. Then, the variation model is prepared for the comparison with the images.

```

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

```

During the inspection phase, again the matching is used to align the image; then, the variation model is applied to check the print for errors. Finally, the erroneous regions are displayed.

```

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

```

5.8.2 Other Examples

HDevelop

- [examples\application_guide\3d_machine_vision\hdevelop\grid_rectification_arbitrary_distortion.dev](#)
Determine differences between two printed pages, even if there are distortions in the vertical direction
→ description in the [Application Note on 3D Machine Vision](#) on page 131
- [examples\hdevelop\Applications\OCR\rotchar.dev](#)
Estimating small inclinations of text lines
- [examples\hdevelop\Applications\OCV\adaption_ocv.dev](#)
Analyzes impact of changes on reported character quality

- [examples\hdevelop\Applications\OCV\print_quality.dev](#)
Inspects quality of letter A in different images
- [examples\hdevelop\Manuals\HDevelop\stamps.dev](#)
Finds images in a document
→ description in the [HDevelop User's Manual](#) on page 131
- [examples\hdevelop\Tools\OCV\write_ocv.dev](#)
Writing OCV data to file (and reading it in again)

C++

- [examples\cpp\source\multi_chars.cpp](#)
Inspecting the quality of printed characters
- [examples\cpp\source\pen.cpp](#)
Inspects the quality of print on a pen using the variation model of HALCON

5.9 Object Recognition 2D

5.9.1 Distinguishing coins

Example: [examples\quick_guide\hdevelop\matching_coins.dev](#)

The task of this application is to distinguish different types of Euro coins, depending on the country of origin. The coins differ on one side, having a specific symbol for each country (see [figure 5.9](#)). The task is solved by shape-based matching, using one model for each country symbol.

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

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

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



Figure 5.9: Multiple coins that must be distinguished.

```

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

```

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

```

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

```

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

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

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

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

```
locate_coin (Image, Coin)
area_center (Coin, _, Row, Column)
gen_circle (Circle, Row, Column, 30)
reduce_domain (Image, Circle, ImageReduced)
find_shape_models (ImageReduced, Models, 0, rad(360), 0.6, 1, 0,
    'least_squares', 0, 1, Row, Column, Angle, Score, Model)
```

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

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

Finally, the model is destroyed.

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

5.10 Surface Inspection

5.10.1 Surface Scratches

Example: [examples\quick_guide\hdevelop\surface_scratch.dev](#)

This example detects scratches on a metal surface (see [figure 5.10](#)).

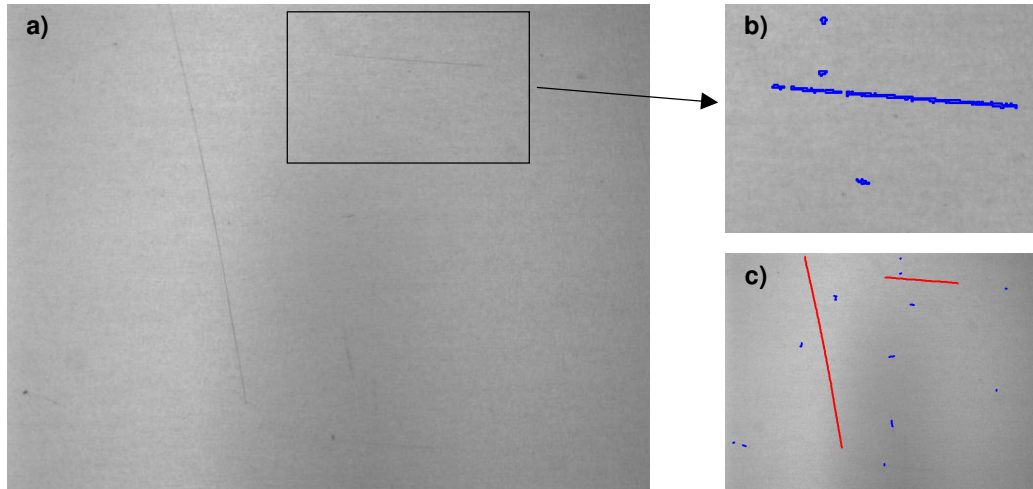


Figure 5.10: Detecting scratches on a metal surface: (a) original image, (b) extracted scratches still partly fractionated, (c) final result with merged scratches.

The main difficulties for the segmentation are the inhomogenous background and the fact that the scratches are thin structures. Both problems can be solved using a local threshold, i.e., the operators `mean_image` and `dyn_threshold`. After `connection`, the small objects that are mainly noise are removed (see [figure 5.10b](#)).

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

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

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

The last step is to distinguish between small dots and scratches on the surface. This is achieved with `select_shape`, using the size as feature. [Figure 5.10c](#) depicts the result.

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