HALCON

the power of machine vision

Quick Guide

MVTec Software GmbH

Building Vision for Business

A quick access to the functionality of HALCON, Version 12.0

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)
Edition 3	June 2007	(HALCON 8.0)
Edition 4	December 2008	(HALCON 9.0)
Edition 5	October 2010	(HALCON 10.0)
Edition 6	May 2012	(HALCON 11.0)
Edition 7	November 2014	(HALCON 12.0)

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

Contents

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 HALCON's *IDE* (integrated development environment), consisting of HDevelop and HDevEngine. HDevelop is a highly interactive development tool that enables a quick development of image processing tasks. Via HDevEngine, you can directly execute HDevelop programs and procedures from your C++, C#, Visual Basic, or C application. As an alternative, HDevelop can also export programs and procedures in your programming language.
- 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.
- HALCON can handle images larger than 32k x 32k (HALCON XL).
- 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, 3D matching, 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 2000 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, Linux, and OS X platforms. 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 19). 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 hundreds of industrial cameras and frame grabbers, including GenICam, GigE Vision, and IIDC 1394.

Rapid Prototyping

In many cases it is important to determine quickly if and how a problem can be solved. With HDevelop, HALCON's interactive development environment, 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 25). And if you want to use an image acquisition device that HALCON does not yet support you can use the images directly or create an image acquisition interface for it.

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. The Solution Guide I guides you from the various application areas to HALCON's machine vision methods and introduces them with examples. As an alternative, you can start out in chapter 3 on page 29 or chapter 4 on page 47, which present examples from various industries and application areas and include references to more information about the examples.

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

You can download HALCON from MVTec's web server under `http://www.mvtec.com/download`. Note that you must first register before downloading the software. Installation instructions are provided on the download page.

To evaluate the full power of HALCON you can obtain a temporary license from your local distributor free of charge. Detailed information about licensing can be found in the Installation Guide.

On a Windows system, you can start HDevelop from the start menu. On a Linux system, execute `hdevelop` from a shell. On OS X, start HDevelop from the Applications folder.

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 13.

  The following two chapters present HALCON example programs sorted by machine vision industry (see chapter 3 on page 29) and by application area (see chapter 4 on page 47), respectively.

- Solution Guide I

  This manual describes the main machine vision methods and how to use them in HALCON, including many examples. Chapter 1 on page 19 guides you from the various application areas to suitable methods.

- Installation Guide

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

- HDevelop User's Guide

  This manual is your guide to using HDevelop, HALCON's interactive development environment. 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.

- Image Acquisition Interface Programmer's Manual

  If you want to use an image acquisition interface that is not yet supported by HALCON this manual explains how to create your own image acquisition interface for it.

- Solution Guide II

  Currently, this guide consists of the following Solution Guides:

  Solution Guide II-A (Image Acquisition): This manual shows how to use simple and complex configurations of image acquisition devices, explains the various timing modes, and more.

  Solution Guide II-B (Matching): Here you can find out about how to use HALCON's 2D and perspective matching methods to find objects based on a single model image and to locate them with subpixel accuracy.

  Solution Guide II-C (2D Data Codes): This manual shows how to use HALCON's 2D data code reader.

  Solution Guide II-D (Classification): This manual shows how to classify images or regions with different types of classifiers.

- Solution Guide III

  Currently, this guide consists of the following Solution Guides:

  Solution Guide III-A (1D Measuring): This manual shows how to detect edges and how to measure their position and distance along lines and arcs.

  Solution Guide III-B (2D Measuring): This manual shows how to measure 2D features like position, orientation, angles, and generally the dimensions of and the relations between objects.

  Solution Guide III-C (3D Vision): This manual is your gate into the world of 3D vision.

- Reference Manuals

  The reference of all HALCON operators is available in HDevelop, C++, C#, COM (IDL), 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 directory denoted by the

environment variable %HALCONEXAMPLES% or, if the variable is not set, in the subdirectory examples of the folder into which you have installed HALCON.

- Support

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

- Training Seminars

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

Chapter 2

How to Develop Applications with HALCON

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.

- Image inspection, prototyping of the vision method, and the final development of the vision method are performed within HDevelop. Here, the program is structured into procedures that each represent one sub-task like initialization, processing, or cleanup. The main procedure 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. There are two ways to integrate the HDevelop procedures: Either you export the procedures to your programming language and then import them, e.g., via an include statement. Alternatively,

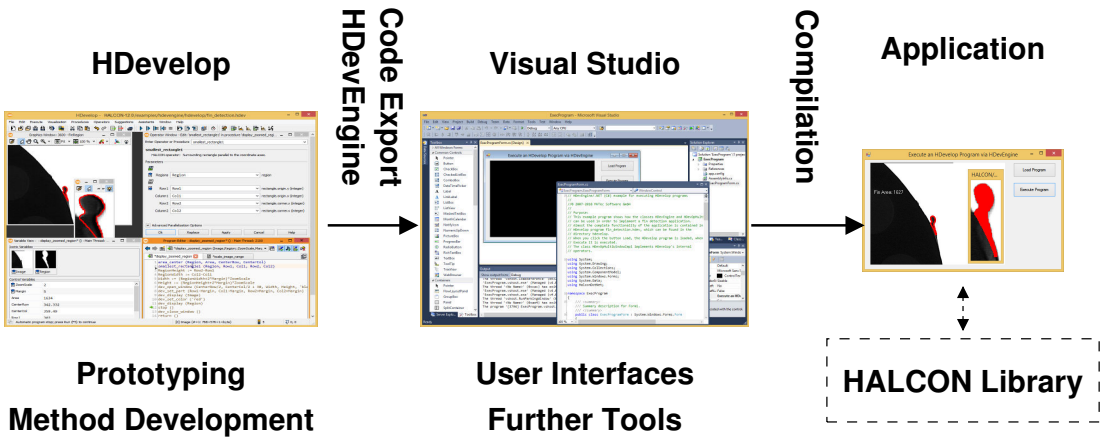


Figure 2.1: Three-step approach for the application development.

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 application represents the solution that can, e.g., be loaded onto the destination machine or sent to a customer.

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. 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 systems.

For basic information on the prototyping with HDevelop see section 2.2 on page 22. How to export code from HDevelop is explained in the HDevelop User's Guide in chapter 9 on page 403. A brief introduction to using HALCON in a programming environment can be found in section 2.3 on page 23. How to use HDevEngine is explained in the Programmer's Guide in part VII on page 207. For using a programming environment like Microsoft Visual C++, see the relevant Microsoft documentation.

Note that in both cases (HDevEngine or export), the application contains "only" the HALCON functionality – HDevelop's powerful development tools like the assistants or other GUI elements cannot be accessed from a programming language.

For C#, however, HALCON provides parts of HDevelop's GUI functionality in form of the so-called HALCON Codelets, which consist of example applications and classes. See the Programmer's Guide, section 17.5 on page 156, for more information.

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 is as short as possible. Furthermore, the operator design has been made such that combinations that are standard sequences or more complex tasks remain efficient.

The open architecture is important in two respects: First, you can use HALCON from many different programming languages. Passing external data to HALCON and accessing internal data of HALCON is also supported. Finally, there are 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 an image acquisition interface without changing the installation of HALCON.

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

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

Figure 2.2: Basic architecture of HALCON.

HALCON's basic architecture is depicted in figure 2.2. The main part is the image processing library, which consists of a huge number of *operators*. You can also develop your own operators in form of

a so-called *Extension Package*. You use the operators in your application via the so-called *language interfaces* like HALCON/C++. These are libraries which allow a direct use of the operators in the typical programming style of the different programming languages. HDevelop also uses a language interface, but this is transparent to the user.

For the access of image acquisition devices, HALCON provides a common interface, the so-called *Image Acquisition Interface*. It allows to use quite different acquisition devices in a common way. The libraries containing the device-specific implementations are loaded dynamically when needed.

Similarly, I/O devices are accessed through device-specific *I/O device interfaces*. These interfaces allow to access different I/O devices in a common way. The libraries are loaded dynamically when needed.

Below, we take a closer look at HALCON's operators and data structures.

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 2000 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++, .NET, 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 25).
- 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-processor or multi-core computer (see section 2.1.3 on page 19).

2.1.2 Parameters and Data Structures

Philosophy:

- HALCON has two basic types of parameters: iconic data (images, regions, XLD contours) and control data (integers, strings, 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 (multi-channel) images, regions, 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 61. 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 (see below).
- 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 the Solution Guide I, Region Of Interest on page 33.

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 directions 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 multi-channel 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'`.

  Note that by default regions are clipped to the current image size. As long as no image has been created or loaded, they are clipped to the default image size (HDevelop: 512×512; programming languages: 128×128). To prevent an undesired clipping, we recommend to create or load an image before creating or loading regions.
- 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 the segmentation of a contour into lines, arcs, polygons or parallels.

2.1.2.4 Handles

Philosophy:

- Handles belong to the control data.
- Handles are references to complex data structures, e.g., a connection to an image acquisition device or a model for the shape-based matching. For efficiency and data security reasons, not the entire structure but only the handle is passed between the operators.
- Handles are unique integer values. 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.

2.1.2.5 Tuple Mode

Philosophy:

- Iconic and control data can contain single objects / values or multiple ones (so-called *tuples*). For example, a control variable can contain none, one, or an arbitrary number of the basic data types (integers, floating-point values, strings), where the types of each element can be different.
- Most operators accept single values (simple mode) as well as multiple values (tuple mode). If, e.g., an operator like `area_center` is called with a single region, one value is returned for the area and the coordinates of the center of the region. When the operator is called with multiple regions, tuples with the corresponding number of values are returned instead. Thus, 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 control tuples range from 0 to the number of values minus 1. In contrast, the index of iconic tuples starts with 1.

2.1.3 HALCON and Parallel Programming

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.

Besides supporting parallel programming, 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 parallel programming using HALCON can be found in the Programmer's Guide, chapter 2 on page 17.

Note that 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 HALCON's parallelization techniques are 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 HALCON supports Compute Devices

HALCON supports GPU processing, which can speed up your application significantly. With HALCON, executing code on the GPU is *transparent*, *easy to use* - just switch it on or off - and it is *based on OpenCL* and therefore independend from manufacturers and not even limited to graphic cards.

We support all new NVIDIA and ATI cards. Lists of compatible cards can be found here:

- NVIDIA: http://www.nvidia.com/object/cuda_learn_products.html
- ATI/AMD: http://developer.amd.com/gpu/atistreamsdk/pages/default.aspx#five

Always install the latest driver. Note that currently for AMD, Stream SDK still needs to be installed.

Many HALCON operators support compute devices. To find out whether an operator supports compute devices or not, please refer to the reference of the operator in the Reference Manual. If the operator supports compute devices, like e.g., `edges_sub_pix`, this is noted in the section 'Parallelization'.

As mentioned before, GPU processing is very easy to use. The GPU is initialized with the operator `init_compute_device`. GPU processing is switched on with `activate_compute_device` and switched off with `deactivate_compute_device`.

Note that to compute on the GPU, the data will be transfered from the CPU to the GPU. Transfering data consumes precious processing time. Therefore, in order to fully profit from GPU processing we recommend to

- use a high-end graphics card,
- use hardware with a high memory bandwidth, and
- design your program well, i.e., group as many GPU-supporting operations as possible, to reduce memory transfers.

However, using compute devices does not always result in an improved runtime. Normal CPU processing may be faster if

- your CPU is already very fast, especially when using multi-core system.
- you are using fast operators.

When computing on the GPU, by default, *one* graphics card will be used. You can, however, also use several compute devices. In that case one thread is used per device.

Please refer to the Reference Manual for more information on the compute devices. Limitations of GPU processing are documented in section 2.5.2 on page 27.

2.1.5 HALCON XL

HALCON is optimized for normal-sized images, i.e., images smaller than 32k x 32k. However, in particular line scan camera applications often provide larger images, typically with the height exceeding 32k. For these applications, you can use HALCON XL, which by itself has no limitation for the image size.

However, there still are other limitations:

- If the image size exceeds 0.5 GB, you cannot process it on a 32 bit platform but must use a 64 bit operating system.
- Independent of the used operating system, the available memory of your system might pose a limitation.

2.1.6 Image Acquisition

Currently, HALCON provides interfaces for more than 50 frame grabbers and hundreds of industrial cameras (analog, Camera Link, USB 2.0, IEEE 1394, and GigE) in the form of dynamically loadable libraries (Windows: DLLs; Unix-like systems: shared libraries). These libraries are installed together with the HALCON libraries. Library names start with the prefix `hAcq`; the libraries ending with the suffix `xl` are used by HALCON XL.

In the following, we give a brief overview of the HALCON image acquisition interface; please refer to the Solution Guide II-A for detailed information about this topic.

The HALCON image acquisition interface libraries form the bridge between software provided by the manufacturer of the image acquisition device and HALCON. They form a common, generic interface that requires a small set of operators only.

If you successfully installed your image acquisition device, all you need to do to access it from HALCON is to call the operator `open_framegrabber`, specifying the name of the image acquisition interface 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 image acquisition 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 manufacturers of image acquisition devices 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.halcon.com/image-acquisition.

2.1.7 I/O devices

HALCON provides interfaces for several I/O devices (sometimes also referred to as I/O modules) to enable data acquisition. These interfaces are available as dynamically loadable libraries (Windows: DLLs; Unix-like systems: shared libraries). Library names start with the prefix `hio`; the libraries ending with the suffix `xl` are used by HALCON XL.

In the following, we give a brief overview of the HALCON I/O device interfaces; please refer to the file `%HALCONEXAMPLES%\io_integration\README.txt` for detailed information.

The HALCON I/O device interfaces provide unified access to different I/O devices using a small set of operators. After you have installed your I/O device, a connection is established using the operator `open_io_device`, specifying the name of the I/O device interface and optionally some device-specific parameters. Once the connection is established, a transmission channel can be opened by calling `open_io_channel`. To read and write values on this channel, use the operators `read_io_channel` and `write_io_channel`, respectively.

Please note that the HALCON I/O device interfaces may change more frequently than the HALCON library itself. You can find the lastest information together with downloadable interfaces (including documentation) under http://www.halcon.com/io.

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. Detailed information about HDevelop can be found in the HDevelop User's Guide.

To start HDevelop under Windows, click on `Start > Programs > MVTec HALCON > HDevelop`. On Windows 8, start HDevelop from your start page. Under Linux, HDevelop is started from the shell by executing `hdevelop`. Under OS X, HDevelop is started from the Applications folder.

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 > Browse Examples ...` 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, click with the mouse to the left of the desired program line so that the program counter (green arrow) is repositioned. 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 the colors that are specified in the menu `Edit > Preferences > User Interfaces > Program Listing`. 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 Guide.

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 install the needed images.
2. Some programs use image acquisition devices for image acquisition or I/O device interfaces for data acquisition. If the corresponding device 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.

2.3 Using HALCON Within Programming Languages

As shown in figure 2.2 on page 15, HALCON provides so-called *language interfaces*. These 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 and the .NET interface can be used with different languages, 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. For .NET applications, the corresponding assembly `halcondotnet.dll` must be referenced.

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++, .NET (C#), COM, and C syntax.

2.3.1 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 III on page 71.

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(0, 0, 512, 512);

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

2.3.2 C# and Visual Basic .NET

C# and Visual Basic .NET use HALCON via the .NET interface.

Analogously to C++, two styles for programming are offered: procedural and object-oriented. For the procedural style, the class `HOperatorSet` provides all HALCON operators, where `HObject` is used to handle iconic data and `HTuple` is used for control data. For the object-oriented style, classes like `HDataCode2d`, `HMeasure`, or `HShapeModel` are provided for the central functionality. In addition, classes for iconic data, e.g., `HImage` or `HRegion`, are available. For more information see the Programmer's Guide, part IV on page 125

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

```
HImage    image = new HImage("monkey");
HRegion   region;

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


2.3.3 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 VI on page 179.

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_size(Monkey, &Width, &Height);
open_window(0, 0, Width, Height, 0, "visible", "", &WindowHandle);
disp_obj(Monkey, WindowHandle);
```

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 2000 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 Image Acquisition Interfaces

Using a similar mechanism, image acquisition interfaces are integrated using dynamic libraries. This allows you to integrate unsupported image acquisition devices without further changes of the rest of the system. How to create and integrate an image acquisition interface is described in the Image Acquisition Interface Programmer's Manual. Furthermore, HALCON is shipped with a template source code that can be used as the basis of an integration.

2.4.3 I/O Device Interfaces

I/O interfaces are integrated using dynamic libraries. This allows you to integrate unsupported I/O devices without further changes of the rest of the system. How to create and integrate an I/O device interface is described in `%HALCONEXAMPLES%\io_integration\README.txt`. Furthermore, HALCON is shipped with a template source code that can be used as the basis of an integration.

2.5 Limitations

2.5.1 General Limitations

With HALCON XL (see section 2.1.5 on page 21), many of HALCON's previous limitations have been removed. Below, we list the remaining ones that are relevant for typical application development with HALCON.

- String processing:
  - The HALCON library does not handle multibyte characters internally. File names containing multibyte characters are supported after calling `set_system('filename_encoding','utf8')`.
- Maximum image size:
  - HALCON: 32 767×32 767
  - HALCON XL: 1 073 741 823×1 073 741 823 ($2^{30}-1 \times 2^{30}-1$)
- Range for coordinates:
  - HALCON: from -32 768 to +32 767
  - HALCON XL: from -1 073 741 824 to 1 073 741 823 (-2^{30} to $2^{30}-1$)
- Maximum number of windows: 600

2.5.2 Limitations Related to Compute Devices

Currently, the following limitations apply to the use of compute devices (see section 2.1.4 on page 20).

Operators that are based on texture are limited to the maximum size of the graphics card. Currently this limit is 8192×8192. This is relevant for the following operators:

- `projective_trans_image` and `projective_trans_image_size`,
- `affine_trans_image` and `affine_trans_image_size`,
- `polar_trans_image`, `polar_trans_image_inv`, and `polar_trans_image_ext`,
- `rotate_image`,
- `mirror_image`,
- `map_image`,
- `image_to_world_plane`, and
- `change_radial_distortion_image`.

Further limitations are listed below.

- All other operators are restricted to the maximum allocated block size, e.g. 200 MB on the Tesla C2050.
- `edges_sub_pix` and `lines_gauss` are not Open CL accelerated for HALCON XL.
- `edges_sub_pix` and `lines_gauss` need a lot of memory on the compute device.

2.5.3 Limitations Related to Image Acquisition

- Maximum number of acquisition interfaces: 128
- Maximum number of open handles per acquisition interface: 256
- Maximum number of channels per image:
  - `grab_image`, `grab_image_async`: 1000
  - otherwise: unlimited
- Maximum number of interface-specific parameters: 2048

2.5.4 Limitations Related to OpenGL

Operators using OpenGL for visualization (e.g., `disp_object_model_3d`) require OpenGL 2.1, GLSL 1.2, and the OpenGL extensions GL_EXT_framebuffer_object and GL_EXT_framebuffer_blit. Therefore, those operators cannot be used via Windows Remote Desktop or SSH forwarding.

2.5.5 Limitations Related to Extension Packages

- maximum number of parameters:

iconic input parameters:	9
iconic output parameters:	9
control input parameters:	20
control output parameters:	20

Chapter 3

Industries

This chapter introduces example programs from various industries:

3.1 Automobile Parts And Manufacturers

3.1.1 Locate Car Door

Example: `hdevelop/Applications/Position-Recognition-3D/locate_car_door.hdev`

The task of this application is to locate a car door placed in different orientations and positions by using the calibrated perspective deformable matching. First a model is defined from a planar area of the car door, then a reference pose is calculated by using an image with a calibration plate in the object plane. For all subsequent takes, the pose can then be determined using the perspective deformable matching as shown in figure 3.1.

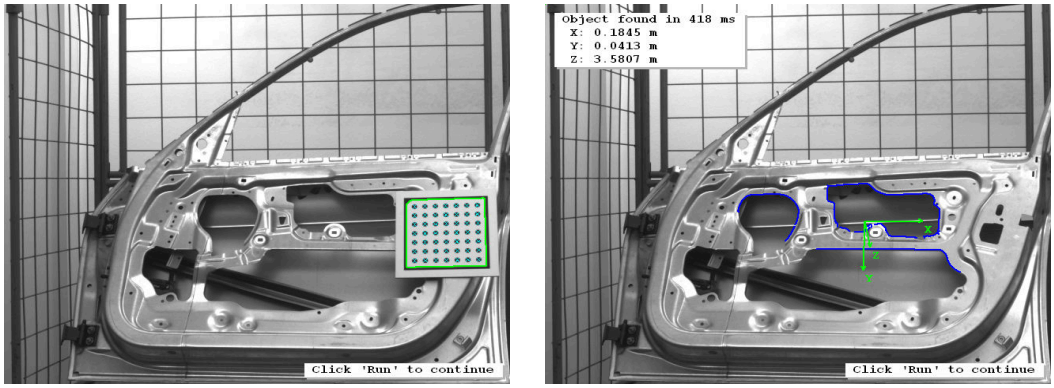


Figure 3.1: (a) Calibration of the reference car door pose; (b) located car door with position values.

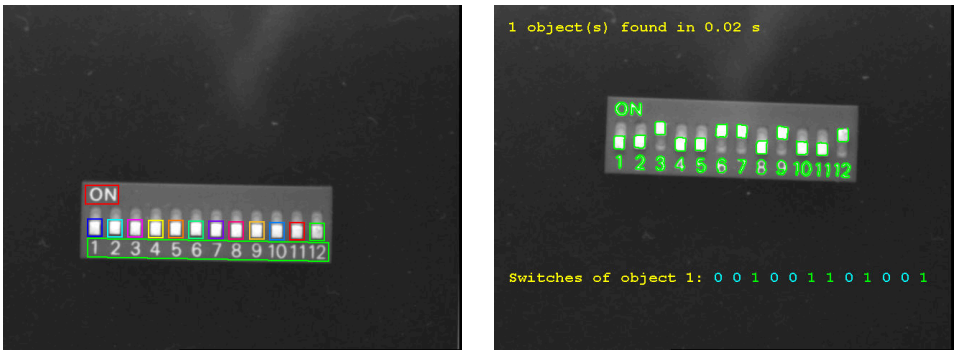
More information on perspective deformable matching and matching in general can be found in the Solution Guide II-B, section 3.6 on page 124 and the Solution Guide I, chapter 9 on page 113.

3.2 Electric Components And Equipment

3.2.1 Check the State of a Dip Switch

Example: `hdevelop/Applications/Position-Recognition-2D/cbm_dip_switch.hdev`

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 3.2). The task is solved using component-based matching.



a) b)

Figure 3.2: (a) Model image of the dip switch with ROIs for the components; (b) located dip switch with state of the switches.

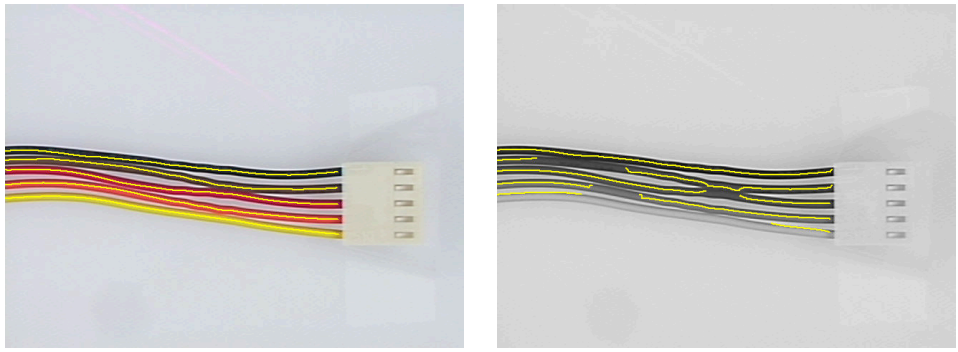
A detailed description of this application can be found in the Solution Guide I on page 129. More information on component-based matching can be found in the Solution Guide II-B, section 3.4 on page 92.

3.2.2 Inspect Power Supply Cables

Example: `hdevelop/Filters/Lines/lines_color.hdev`

The task of this example is to locate and inspect the power supply cables depicted in figure 3.3. 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.

A detailed description of this application can be found in the Solution Guide I on page 199.



a) b)

Figure 3.3: (a) Original color image with cable centers extracted using the color line extractor; (b) corresponding results when using the gray value image.

3.3 Agriculture, Food

3.3.1 "Best Before" Date

Example: `hdevelop/Applications/OCR/bottle.hdev`

The task of this example is to inspect the "best before" date on the bottle depicted in figure 3.4. The task is solved using OCR.

a) b)

Figure 3.4: (a) Original image; (b) read date.

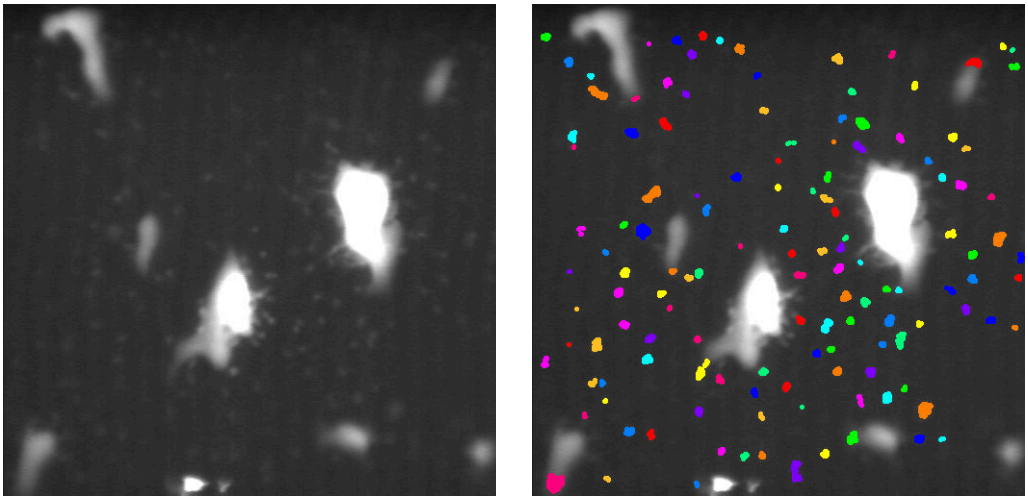
A detailed description of this application can be found in the Solution Guide I on page 274.

3.4 Health Care And Life Science

3.4.1 Analyzing Particles

Example: `hdevelop/Applications/Measuring-2D/particle.hdev`

The task of this application is to analyze particles in a liquid. The main difficulty 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 task is solved by blob analysis.



a) b)

Figure 3.5: Extracting the small particles: (a) original image, (b) result.

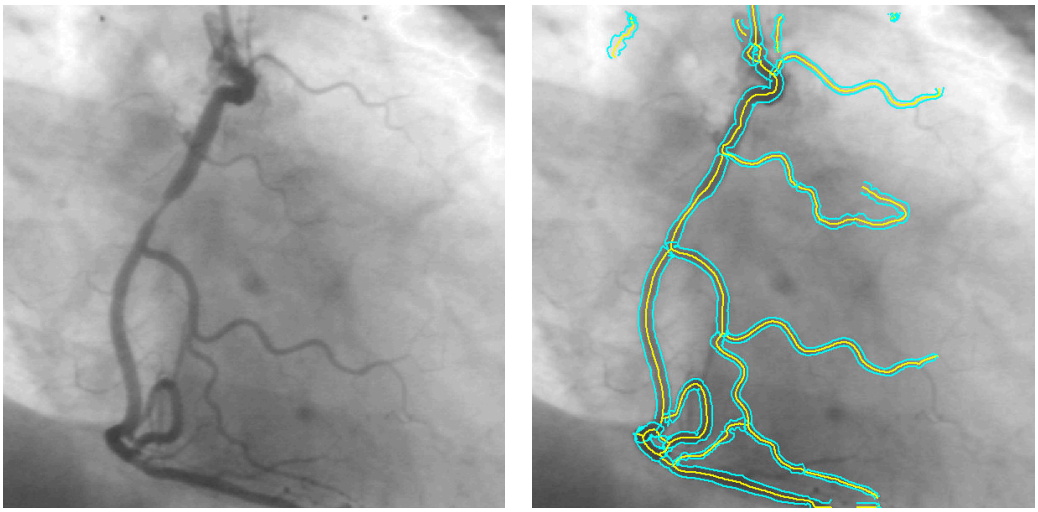
A detailed description of this application can be found in the Solution Guide I on page 52.

3.4.2 Angiography

Example: `hdevelop/Filters/Lines/lines_gauss.hdev`

The task of this example is to extract the blood vessels in the X-ray image of the heart depicted in figure 3.6. 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). The task is solved by using the subpixel-precise line extractor.

A detailed description of this application can be found in the Solution Guide I on page 92.



a) b)

Figure 3.6: (a) X-ray image of the heart; (b) extracted blood vessels.

3.5 Iron, Steel And Metal

3.5.1 Inspect Cast Part

Example: `hdevelop/Applications/Measuring-2D/measure_arc.hdev`

The task of this example is to inspect the distance between elongated holes of a cast part after chamfering (see figure 3.7). The task is solved using 1D measuring.

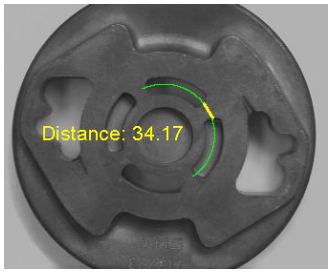


Figure 3.7: Measuring the distance between the holes.

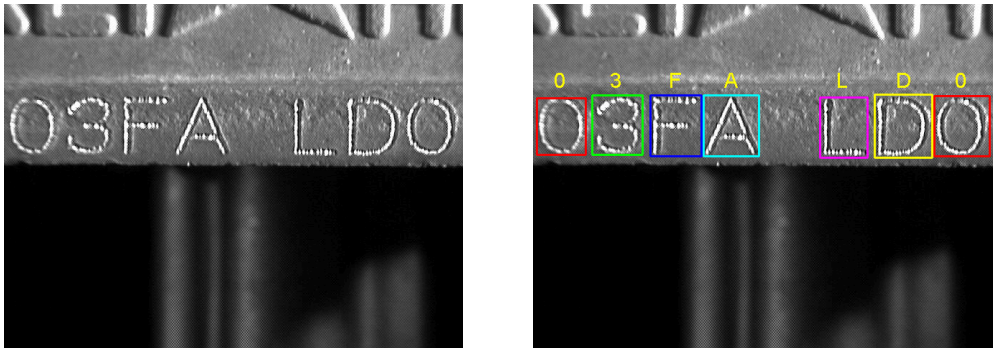
A detailed description of this application can be found in the Solution Guide I on page 68. More information on 1D measuring can be found in the Solution Guide III-A.

3.6 Machinery

3.6.1 Reading Engraved Text

Example: `hdevelop/Applications/OCR/engraved.hdev`

The task of this example is to read the engraved text on the metal surface depicted in figure 3.8. The task is solved using OCR.



a) b)

Figure 3.8: (a) Original image; (b) read characters.

A detailed description of this application can be found in the Solution Guide I on page 275.

3.6.2 Inspecting the Contours of a Tool

Example: `hdevelop/Applications/Measuring-2D/circles.hdev`

The task of this example is to inspect the circular contours of the tool depicted in figure 3.9. The task is solved by contour processing, which is applied to an automatically derived region of interest (ROI).

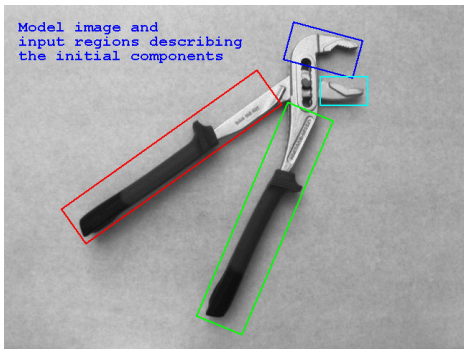
A detailed description of this application can be found in the Solution Guide I on page 39.

Figure 3.9: Fitting circles to the contours of the tool.

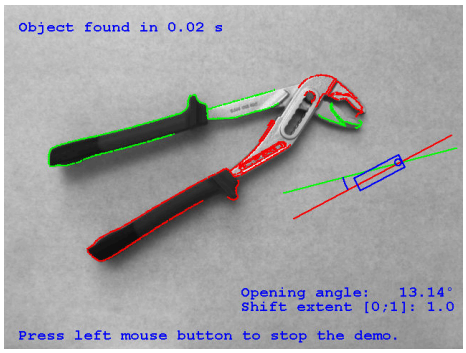
3.6.3 Locating a Pipe Wrench in Different States

Example: `hdevelop/Applications/Position-Recognition-2D/cbm_pipe_wrench.hdev`

The task of this example is to locate a pipe wrench based on four predefined ROIs, two for each rigid part (see figure 3.10). The task is solved using component-based matching.



a)



b)

Figure 3.10: (a) Model image with specified ROIs for the components; (b) located pipe wrench in another state.

A detailed description of this application can be found in the Solution Guide I on page 130. More information on component-based matching can be found in the Solution Guide II-B, section 3.4 on page 92.

3.7 Packaging

3.7.1 Check Mixed Blister

Example: `hdevelop/Applications/Completeness-Check/check_blister_mixed.hdev`

The task of this example is to check the content of manually filled blisters. The first image (reference) shows the correct combination. The subsequent images are then checked against the reference image unsing a GMM classifier as shown in figure 3.11.

Figure 3.11: (a) The GMM classifier is trained with reference blister image; (b) recognition of a defect, a missing pill.

More information on classification can be found in the Solution Guide II-D and in the Solution Guide I, chapter 12 on page 171.

3.7.2 Locate Cookie Box

Example: `hdevelop/Applications/Object-Recognition-2D/locate_cookie_box.hdev`

This example shows how to find a cookie box in different orientations with the descriptor-based matching. First, a model and a reference pose are computed from the reference image, then the model is searched in a sequence of images showing the cookie box in different orientations. Using the resulting pose, we can also check whether the box is placed upside down or not as shown in figure 3.12.

Note that, when running this example in HDevelop, the training of the model takes about half a minute.

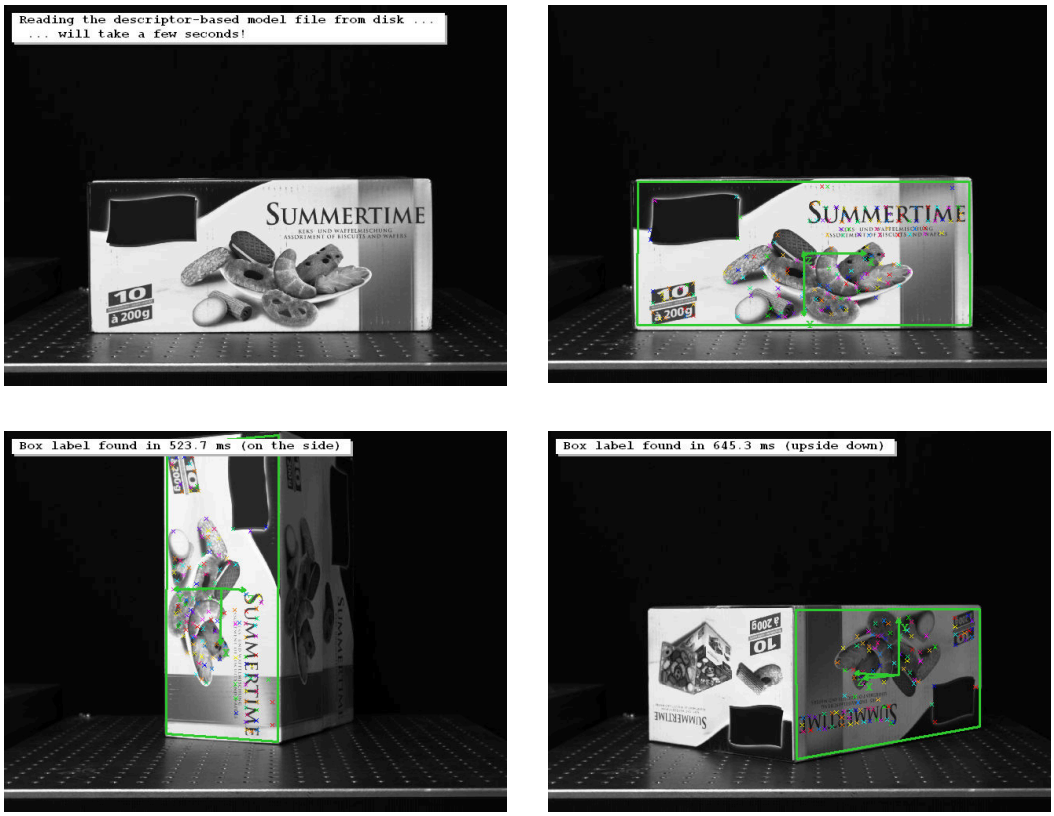


Figure 3.12: A model is trained first and the box side matching this model can be found in subsequent images even if placed upside down.

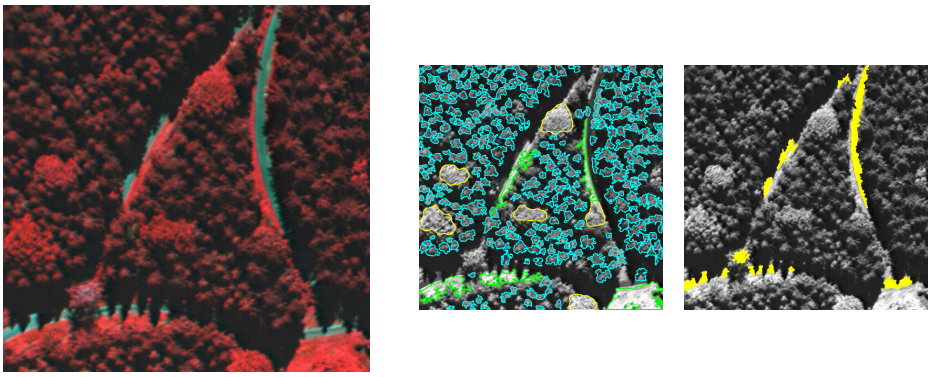
More information on descriptor-based matching and matching in general can be found in the Solution Guide II-B, section 3.7 on page 136 and the Solution Guide I, chapter 9 on page 113.

3.8 Photogrammetry And Remote Sensing

3.8.1 Extracting Forest Features from Color Infrared Image

Example: `hdevelop/Applications/Object-Recognition-2D/forest.hdev`

The task of this example is to detect different object classes in the color infrared image depicted in figure 3.13: trees (coniferous and deciduous), meadows, and roads. The image data is a color infrared image, which allows to extract roads very easily because of their specific color. The taks is solved using color processing and blob analysis.



a) b) c)

Figure 3.13: (a) Original image; (b) extracted trees and meadows; (c) extracted roads.

A detailed description of this application can be found in the Solution Guide I on page 52.

3.8.2 Segmenting a Color Image

Example: `hdevelop/Filters/Edges/edges_color.hdev`

The task of this example is to segment the color image depicted in figure 3.14. The example demonstrates the possibilities of a multi-channel edge filter. When using the corresponding gray value image, the (green) soccer field cannot be distinguished from the surrounding (red) track.

A detailed description of this application can be found in the Solution Guide I on page 82.

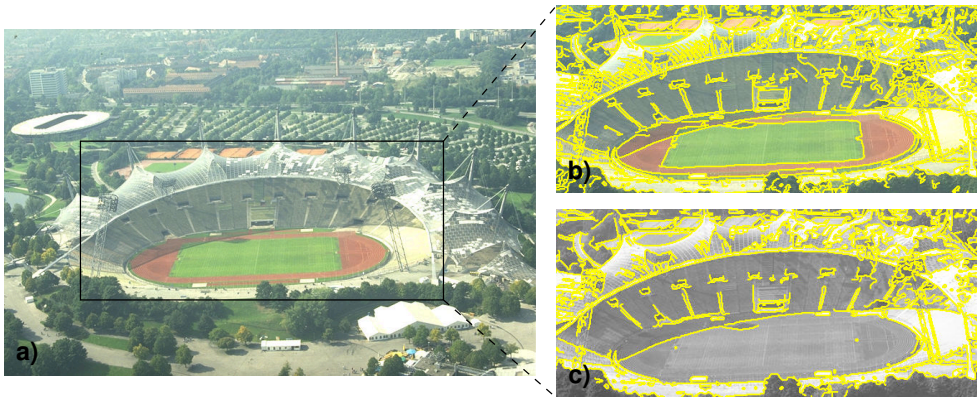


Figure 3.14: (a) Original image; (b) extracted color edges, overlaid on the color image; (c) extracted gray value edges, overlaid on the gray value image.

3.8.3 Extract Roads

Example: `hdevelop/Applications/Object-Recognition-2D/roads.hdev`

The task of this example is to extract the roads in the aerial image depicted in figure 3.15. 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. The task is solved mainly by contour processing.

Figure 3.15: (a) Original image; (b) zoomed image part; (c) extracted roads.

A detailed description of this application can be found in the Solution Guide I on page 107.

3.9 Printing

3.9.1 Reading Bar Codes on a Toner Cartridge

Example: `hdevelop/Identification/Bar-Code/code39.hdev`

The task of this example is to read bar codes of type Code 39 like the two bar codes depicted in figure 3.16. The bar code type Code 39 is very popular on packages and other kind of product descriptions.

Figure 3.16: Detected bar codes.

3.10 Rubber, Synthetic Material, Foil

3.10.1 Checking a Boundary for Fins

Example: `hdevelop/Applications/Measuring-2D/fin.hdev`

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 3.17). The task is solved using blob analysis.

A detailed description of this application can be found in the Solution Guide I on page 54.

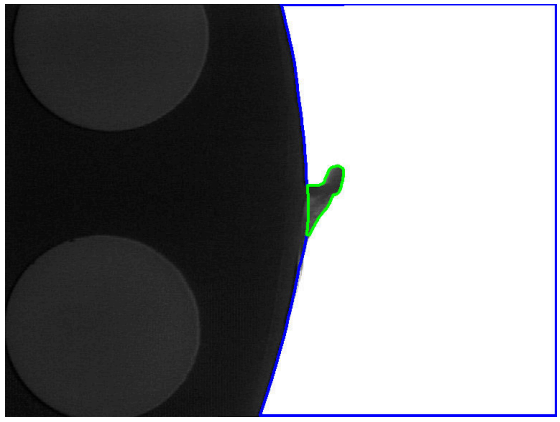


Figure 3.17: Boundary with extracted fin.

3.11 Semiconductors

3.11.1 Bonding Balls

Example: `hdevelop/Applications/Completeness-Check/ball.hdev`

The task of this example is to inspect the diameter of the ball bonds depicted in figure 3.18. The task is solved using blob analysis.

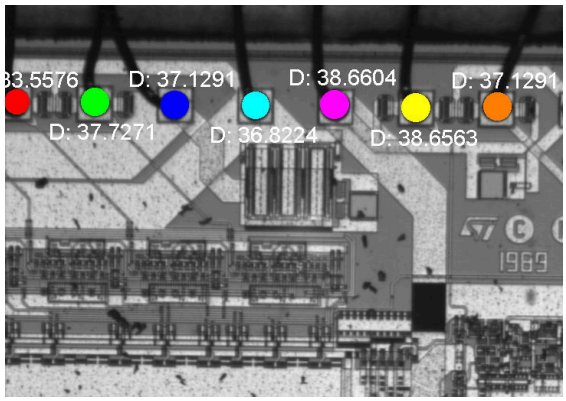


Figure 3.18: Measuring the diameter of ball bonds.

A detailed description of this application can be found in the Solution Guide I on page 55.

3.11.2 Inspecting an IC Using Fuzzy Measuring

Example: `hdevelop/Applications/Measuring-2D/fuzzy_measure_pin.hdev`

The task of this example is to inspect the lead width and the lead distance of the IC depicted in figure 3.19. It is solved using 1D measuring.

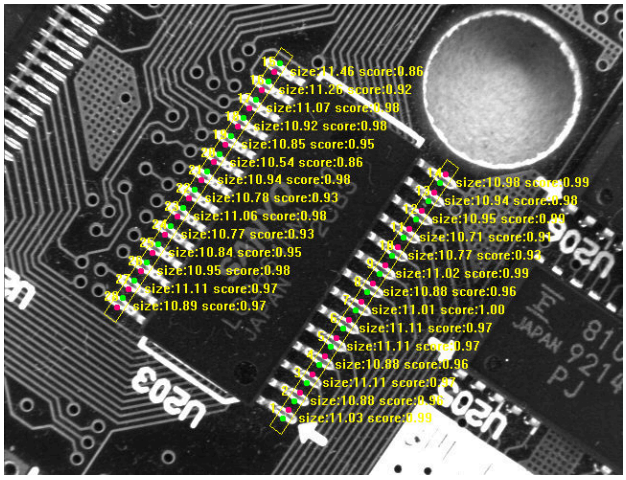


Figure 3.19: Measuring the width and distance of the leads.

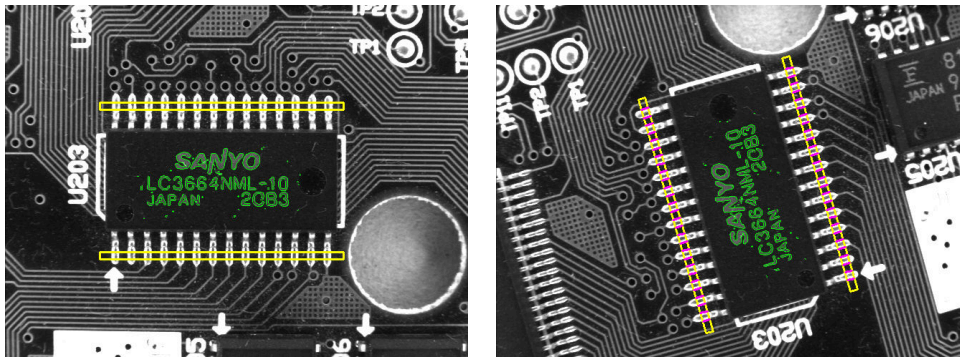
A detailed description of this application can be found in the Solution Guide I on page 69. More information on 1D measuring can be found in the Solution Guide III-A.

3.11.3 Measuring Leads of a Moving IC

Example: `hdevelop/Applications/Measuring-2D/pm_measure_board.hdev`

The task of this example is to measure the positions of the leads of a chip (see figure 3.20). Because the chip can appear at varying positions and angles, the regions of interest used for the measurement must be aligned. The actual measurement is then realized by 1D measuring.

A detailed description of this application can be found in the Solution Guide I on page 70. More information on 1D measuring can be found in the Solution Guide III-A.



a) b)

Figure 3.20: (a) Model image with measurement ROIs; (b) measuring the leads in the aligned ROIs.

3.11.4 Creating a Mosaic Image

Example: `hdevelop/Tools/Mosaicking/gen_projective_mosaic.hdev`

The task of this example is to create an image of the elongated printed circuit board depicted in figure 3.21. Using standard image sizes, most of the image would be empty. The solution is to acquire images from multiple viewpoints and then create a mosaic image using a point-based matching.

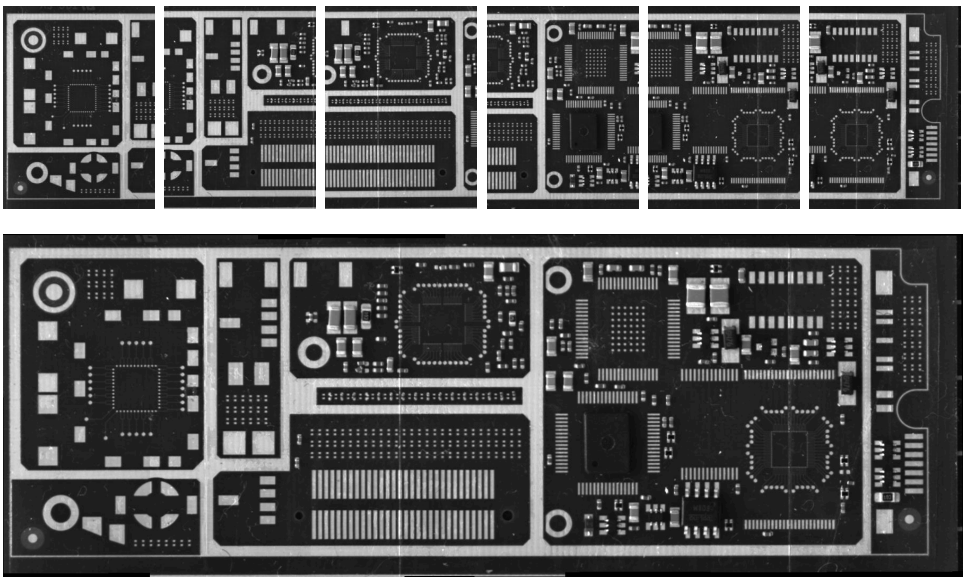


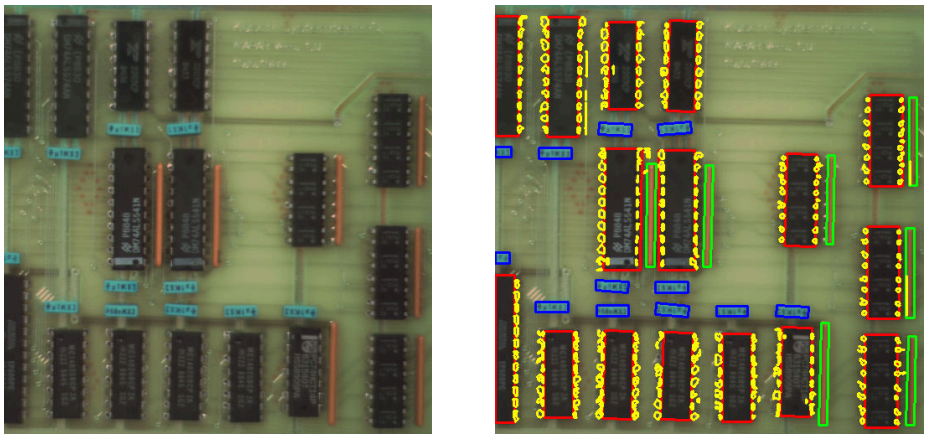
Figure 3.21: Creating a mosaic image from multiple overlapping images.

A detailed description of this application can be found in the Solution Guide I on page 132.

3.11.5 Locating Board Components by Color

Example: `hdevelop/Applications/Completeness-Check/ic.hdev`

The task of this example is to locate all components on the printed circuit board depicted in figure 3.22. It is solved by color processing and blob analysis.



a) b)

Figure 3.22: (a) Original image; (b) extracted ICs, resistors, and capacitors.

A detailed description of this application can be found in the Solution Guide I on page 200.

3.12 Solar, Renewable Energy, Recycling

3.12.1 Inspect Solar Cells

Example: `examples/hdevelop/Applications/Completeness-Check/inspect_solar_panel.hdev`

The task of this example is to extract defect fingers from solar cells. A typical problem is that individual fingers of a solar cell are broken as depicted in figure 3.23. The task is solved using blob analysis and morphology.

More information on blob analysis can be found in the Solution Guide I, chapter 4 on page 45.

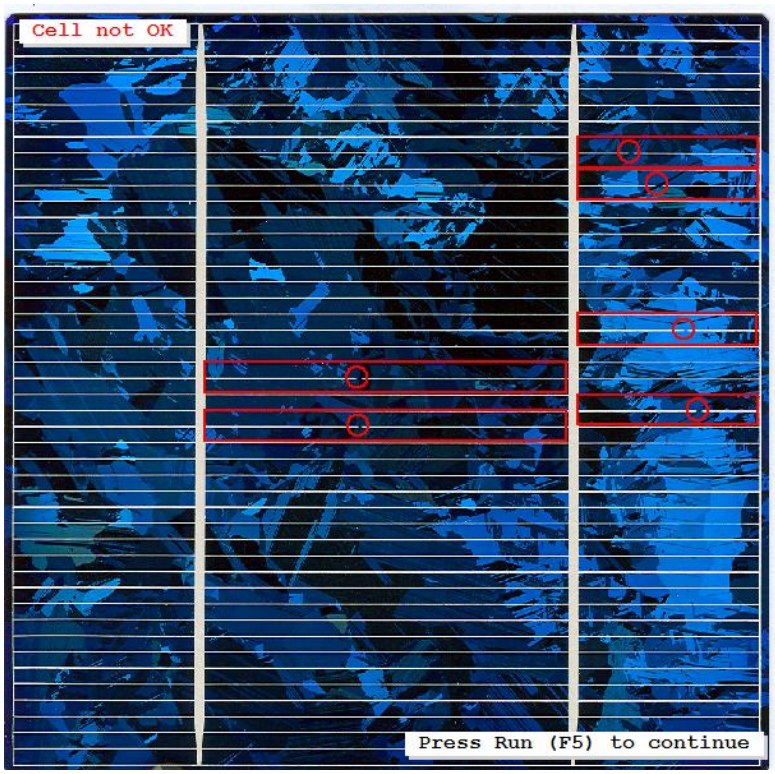


Figure 3.23: Extract broken fingers on a solar cell.

Chapter 4

Application Areas

This chapter introduces example programs from various application areas:

- Identification With Bar Codes (section 4.1),
- Identification With Data Codes (section 4.2 on page 49),
- Completeness Check (section 4.3 on page 50),
- Measuring And Comparison 2D (section 4.4 on page 51),
- Measuring And Comparison 3D (section 4.5 on page 52),
- Optical Character Recognition (section 4.6 on page 53),
- Position Recognition 2D (section 4.7 on page 54),
- Position Recognition 3D (section 4.8 on page 55),
- Object Recognition 2D (section 4.9 on page 56),
- Object Recognition 3D (section 4.10 on page 57),
- Robot Vision (section 4.11 on page 58),
- Print Inspection (section 4.12 on page 59),
- Surface Inspection (section 4.13 on page 60), and
- Traffic (section 4.14 on page 61).

4.1 Identification With Bar Codes

4.1.1 Reading a Bar Code on a CD

Example: `hdevelop/Applications/Bar-Codes/circular_barcode.hdev`

Figure 4.1 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.

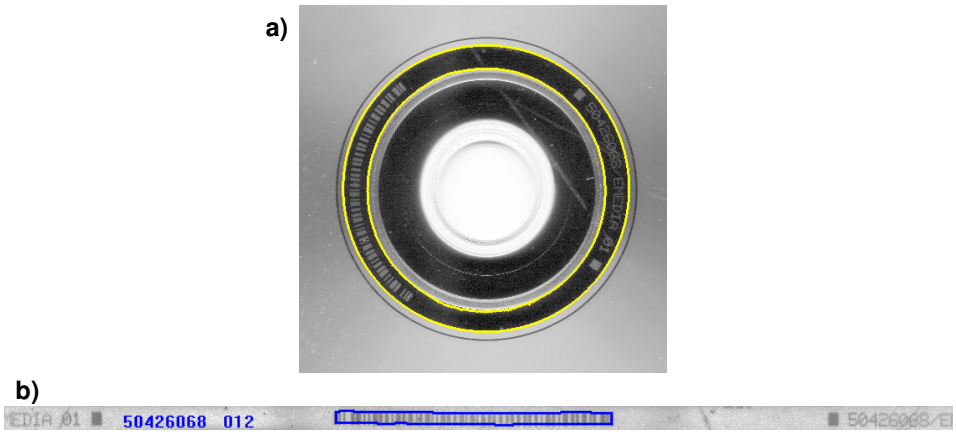


Figure 4.1: (a) Original image with segmented ring; (b) rectified ring with decoded bar code.

A detailed description of this application can be found in the Solution Guide I on page 244.

4.2 Identification With Data Codes

4.2.1 Reading 2D Data Codes on Chips

Example: `hdevelop/Applications/Data-Codes/ecc200_optimized.hdev`

This example program reads 2D data codes (type ECC200) engraved in chips (see figure 4.2).

Figure 4.2: Decoded data code.

A detailed description of this application can be found in the Solution Guide I on page 256. More information on 2D data codes can be found in the Solution Guide II-C.

4.3 Completeness Check

4.3.1 Inspect Razor Blades

Example: `solution_guide/matching/align_measurements.hdev`

The task of this example is to check the razor blades depicted in figure 4.3. The program uses shape-based matching to locate all blades and then uses 1D measuring to inspect the blades. Missing parts are detected and displayed. This example is described in more detail in the Solution Guide II-B, section 2.4.3.2 on page 42.

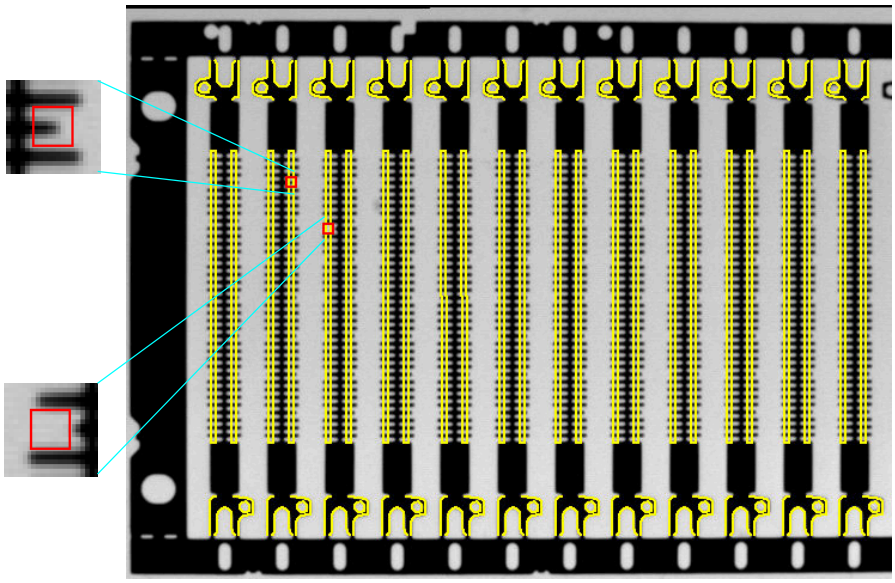


Figure 4.3: Inspecting razor blades.

4.4 Measuring And Comparison 2D

4.4.1 Inspect IC

Example: `hdevelop/Applications/Measuring-2D/measure_pin.hdev`

The task of this example is to inspect major dimensions of an IC (see figure 4.4). The task is solved using 1D measuring.

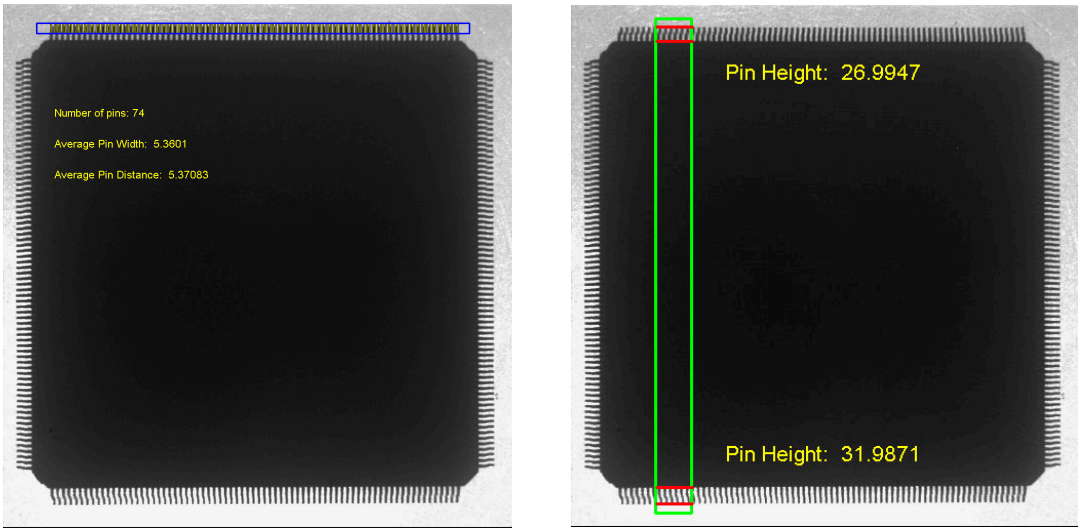


Figure 4.4: Measuring the dimensions of leads: (a) width of the leads and distance between them; (b) length of the leads.

A detailed description of this application can be found in the Solution Guide I on page 71. More information in 1D measuring can be found in the Solution Guide III-A.

4.5 Measuring And Comparison 3D

4.5.1 Inspect IC

Example: `hdevelop/Applications/Measuring-3D/resistor.hdev`

The task of this example is to inspect a soldered component using depth-from-focus (see figure 4.5). There, a sequence of images that are acquired with a continuously varying focus is used to derive a depth image.

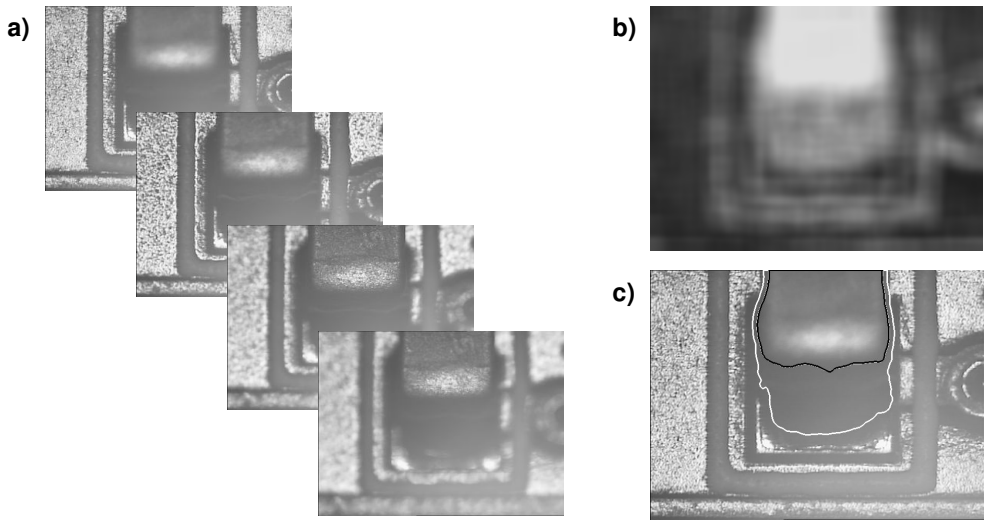


Figure 4.5: Depth-from-focus: (a) sequence of input images; (b) depth image; (c) segmentation of components.

More information on depth-from-focus can be found in the Solution Guide III-C, chapter 7 on page 197.

4.6 Optical Character Recognition

4.6.1 Reading Forms

Example: `hdevelop/Applications/OCR/ocrcolor.hdev`

The task of this example is to extract and read the symbols in the form. A typical problem is that the symbols are not printed in the correct place, as depicted in figure 4.6.

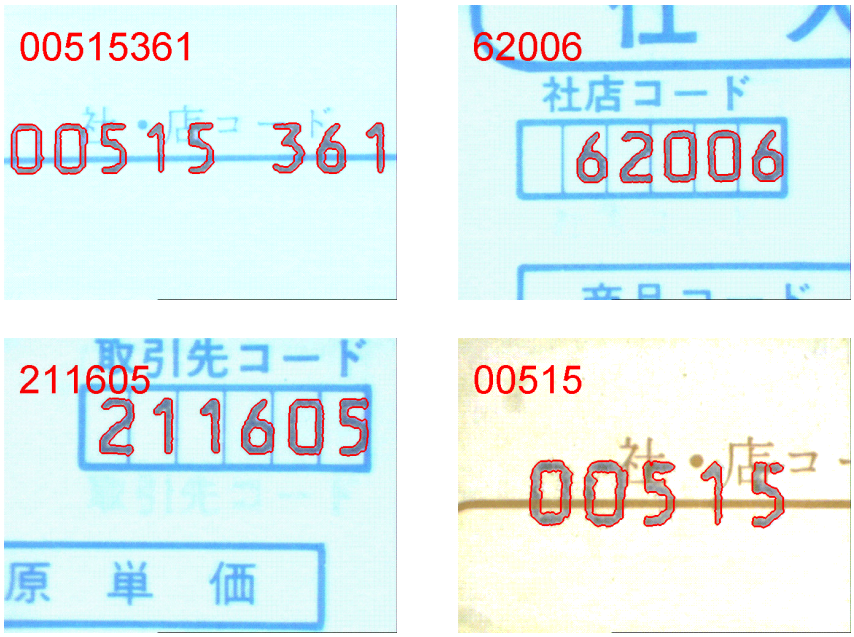


Figure 4.6: Example images for OCR.

A detailed description of this application can be found in the Solution Guide I on page 276.

4.7 Position Recognition 2D

4.7.1 Locate Components on a PCB

Example: `hdevelop/Matching/Component-Based/cbm_modules_simple.hdev`

The task of this example is to locate multiple components on a printed circuit board in one step (see figure 4.7). On a printed circuit board, typically multiple different objects are mounted, whose positions can vary because of the tolerances in the mounting process. To locate all objects quickly and in a robust manner, the component-based matching is used.

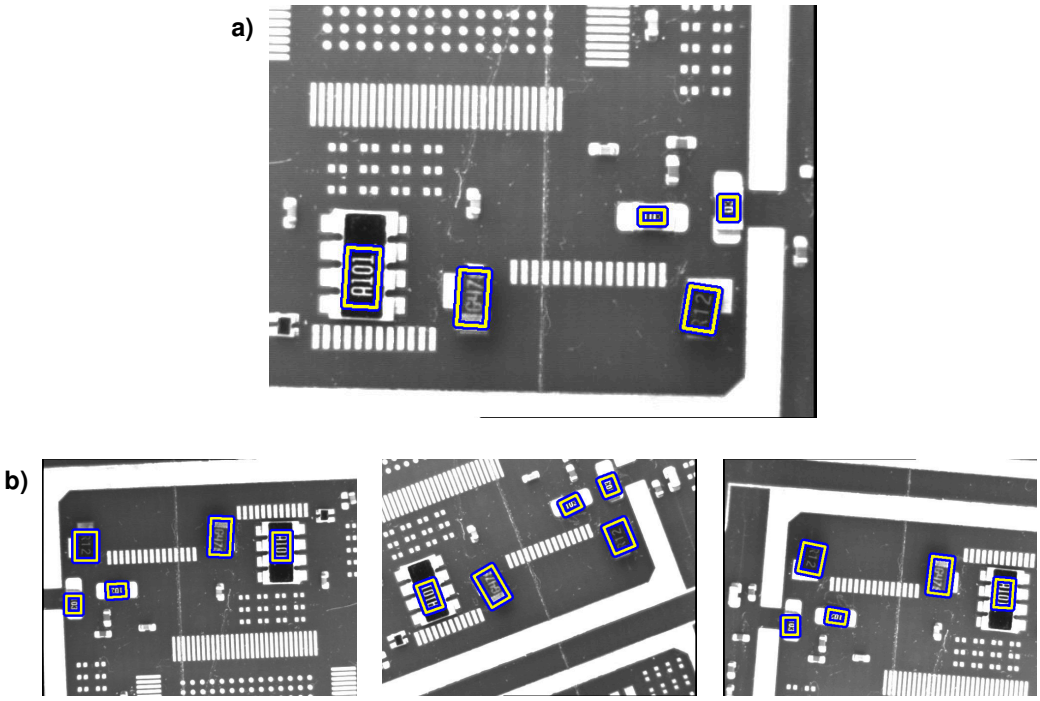


Figure 4.7: Component-based matching: (a) training objects; (b) objects, located in different images where the relation of the objects with respect to each other varies.

A detailed description of this application can be found in the Solution Guide I on page 128. More information on component-based matching can be found in the Solution Guide II-B, section 3.4 on page 92.

4.8 Position Recognition 3D

4.8.1 Get 3D Poses of Work Sheets

Example: `hdevelop/Applications/Position-Recognition-3D/3d_position_of_rectangle.hdev`

The task of this example is to determine the 3D poses of work sheets like the one shown in figure 4.8. The task is solved by a pose estimation for 3D rectangles. How to determine the poses of known 3D objects is described in the Solution Guide III-C, chapter 4 on page 103.

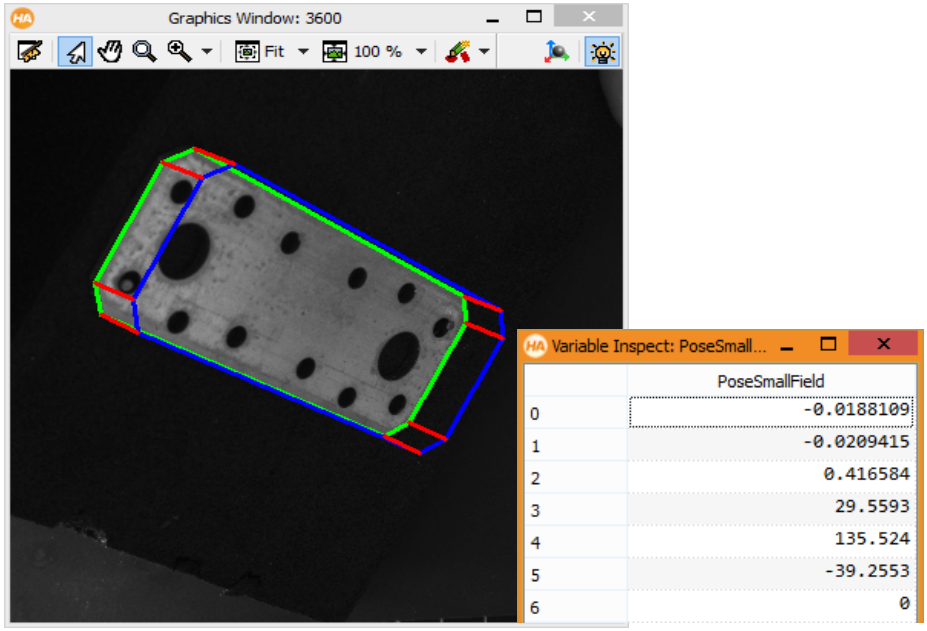


Figure 4.8: Estimating the 3D pose of a rectangular work sheet: (left) pose of the work sheet visualized by extruding the object's border, (right) numerical result for the pose.

4.9 Object Recognition 2D

4.9.1 Distinguishing coins

Example: `solution_guide/basics/matching_coins.hdev`

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 4.9). The task is solved by shape-based matching, using one model for each country symbol.

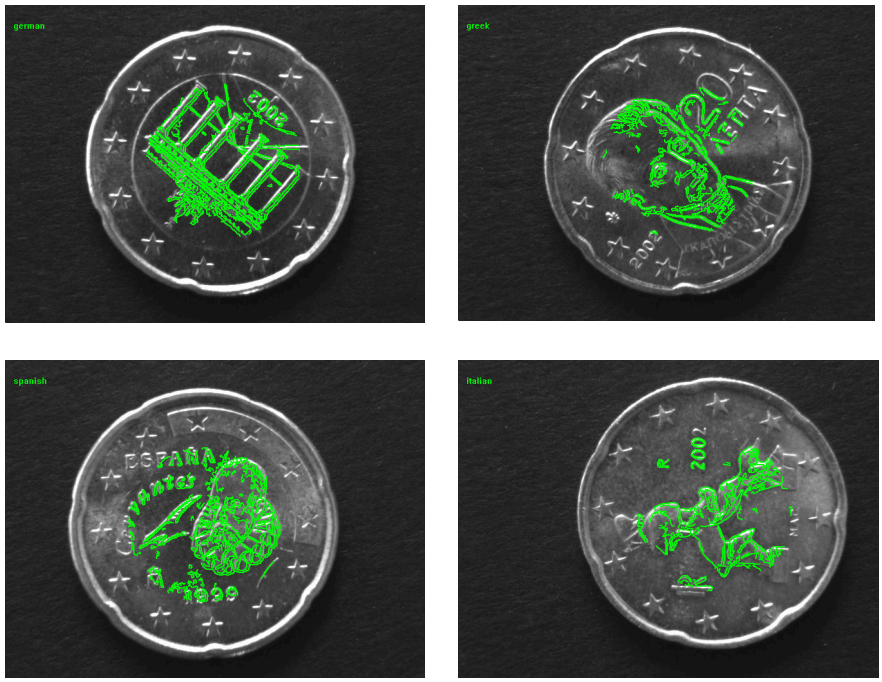


Figure 4.9: Multiple coins that are distinguished.

A detailed description of this application can be found in the Solution Guide I on page 126. More information on shape-based matching can be found in the Solution Guide II-B, section 3.3 on page 64.

4.10 Object Recognition 3D

4.10.1 Find 3D Clamps

Example: `hdevelop/Applications/Position-Recognition-3D/3d_matching_clamps.hdev`

The task of this example is to find the 3D clamps shown in figure 4.10 and determine their poses, i.e., their position and orientation in space. The task is solved by a shape-based 3D matching.

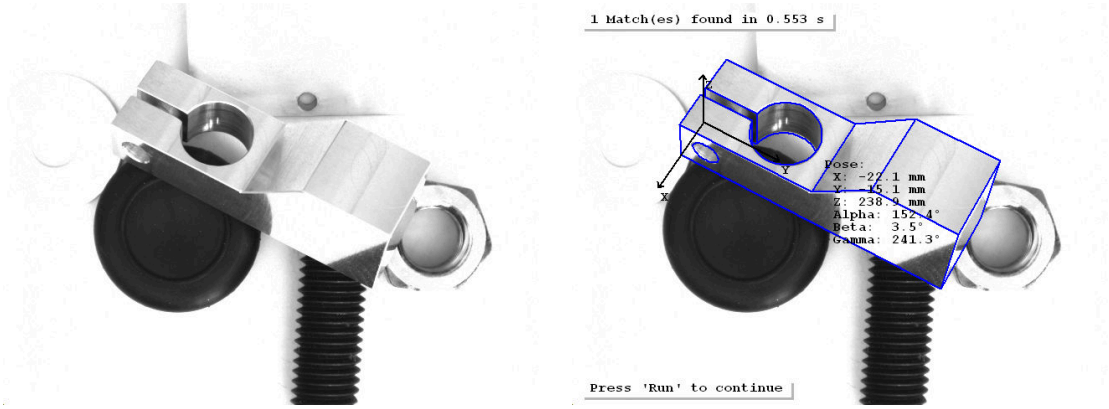


Figure 4.10: 3D matching: (left) original image containing a clamp, (right) 3D model of the found clamp projected in the image and display of its pose.

A detailed description of this application can be found in the Solution Guide I on page 150. More information on shape-based 3D matching can be found in the Solution Guide III-C, section 4.2 on page 109.

4.11 Robot Vision

4.11.1 Grasp Nut

Example: `solution_guide/3d_vision/handeye_stationarycam_grasp_nut.hdev`

The task of this example is to determine the relation between the camera, the pose of the nut shown in figure 4.11, and the pose of a robot that has to grasp the nut. The task is solved by a hand-eye calibration.

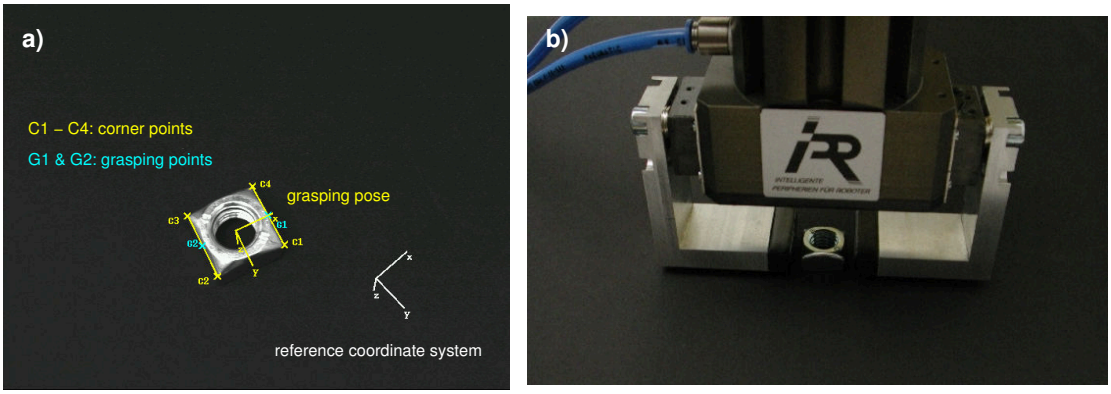


Figure 4.11: (a) Determining the 3D pose for grasping a nut, (b) robot at grasping pose.

A detailed description of this application can be found in the Solution Guide III-C, chapter 8 on page 213.

4.12 Print Inspection

4.12.1 Inspect a Printed Logo

Example: `hdevelop/Applications/Print-Inspection/print_check.hdev`

The task of this example is to inspect the logo printed on the pen clip depicted in figure 4.12. As an additional difficulty, the pens move from image to image. The task is solved using a variation model.

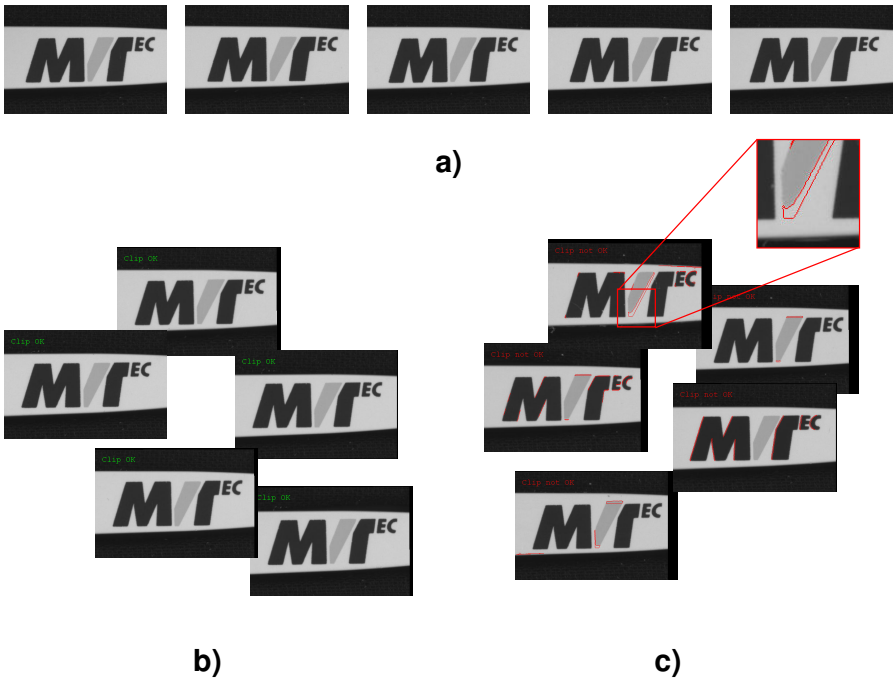


Figure 4.12: Inspection of the logo that is printed on the pen clip: (a) training images; (b) accepted images; (c) rejected images.

A detailed description of this application can be found in the Solution Guide I on page 161.

4.13 Surface Inspection

4.13.1 Surface Scratches

Example: `solution_guide/basics/surface_scratch.hdev`

This example detects scratches on a metal surface (see figure 4.13). The task is solved using blob analysis. The main difficulties for the segmentation are the inhomogenous background and the fact that the scratches are thin structures.

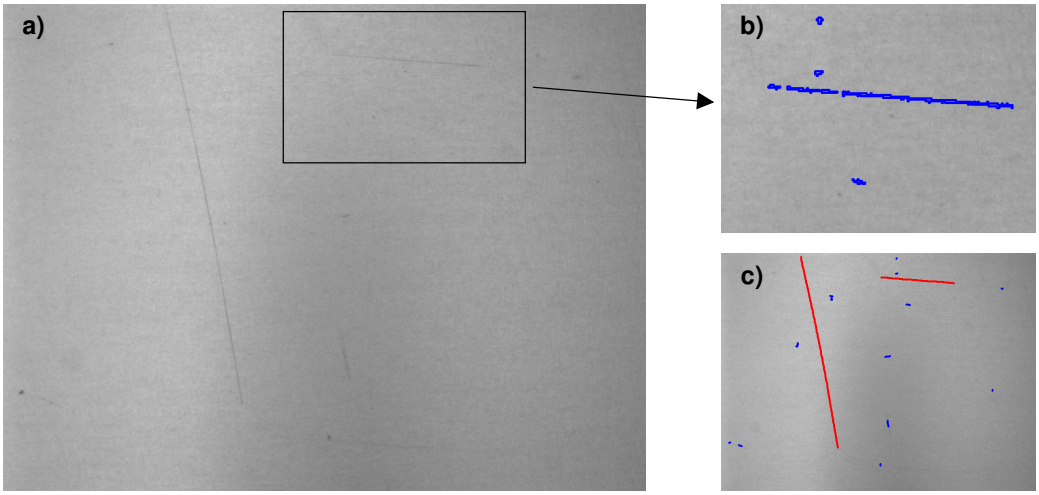


Figure 4.13: Detecting scratches on a metal surface: (a) original image, (b) extracted scratches still partly fractioned, (c) final result with merged scratches.

A detailed description of this application can be found in the Solution Guide I on page 56.

4.14 Traffic

4.14.1 Detect Road Signs

Example: `hdevelop/Applications/Traffic-Monitoring/detect_road_signs.hdev`

A monitoring system in a car checks the sidewalk for road signs to support the driver. The task is solved using perspective deformable matching. To show the imaging process we focus on two road signs, the attention and the dead end road sign. First, the models of both signs are generated and then tracked in a street sequence as shown in figure 4.14.

Figure 4.14: Using the models of the attention and the dead end sign, those signs can be tracked in a street.

A detailed description of this application can be found in the Solution Guide I, chapter 9 on page 113. More information on perspective deformable matching can be found in the Solution Guide II-B, section 3.6 on page 124.