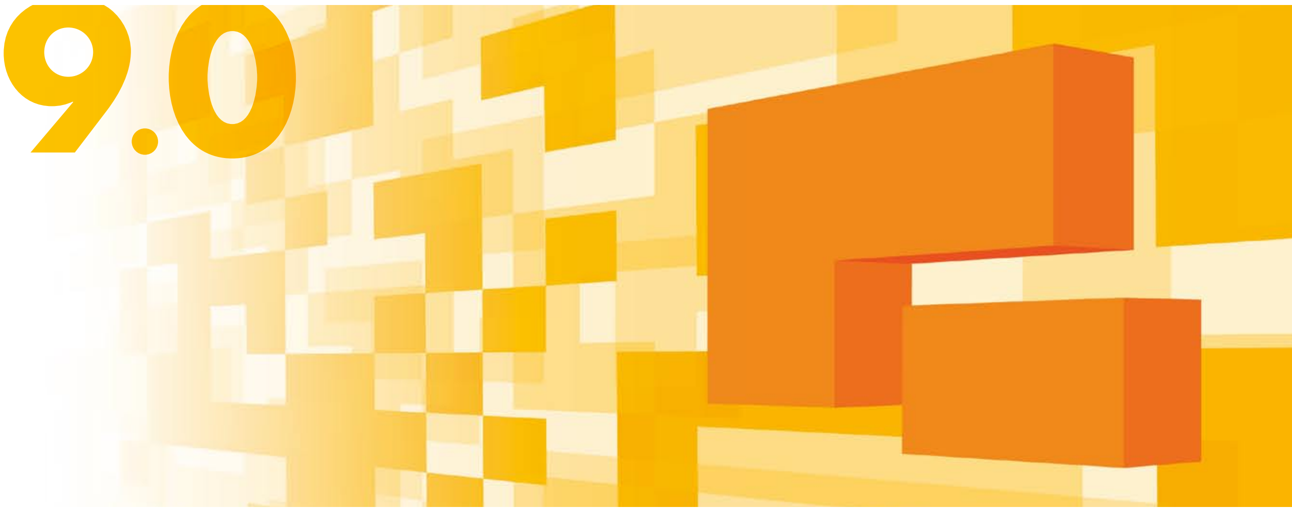
HALCON

the Power of Machine Vision

Solution Guide III-C

3D Vision

9.0



MVTec Software GmbH

Building Vision for Business

Overview

Measurements in 3D become more and more important. HALCON provides many methods to perform 3D measurements. This Solution Guide gives you an overview over these methods, and it assists you with the selection and the correct application of the appropriate method.

A short characterization of the various methods is given in section 1 on page 5. Principles of 3D transformations and poses as well as the description of the camera model can be found in section 2 on page 8. Afterwards, the methods to perform 3D measurements are described in detail.

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

Machine Vision in 3D World Coordinates, Version 9.0.4

Edition 1	December 2003	(HALCON 7.0)
Edition 1a	July 2004	(HALCON 7.0.1)
Edition 1b	April 2005	(HALCON 7.0.2)
Edition 2	July 2005	(HALCON 7.1)
Edition 2a	December 2006	(HALCON 7.1.2)
Edition 3	June 2007	(HALCON 8.0)
Edition 3a	October 2007	(HALCON 8.0.1)
Edition 4	December 2008	(HALCON 9.0)
Edition 4a	June 2009	(HALCON 9.0.1)
Edition 4b	March 2010	(HALCON 9.0.2)

Contents

1 Can You Really Perform 3D Vision with HALCON?

Do you have an application at hand where it is necessary to perform 3D measurements from images? Then, HALCON is exactly the right software solution for you. This Solution Guide will introduce you to the world of 3D vision with HALCON.

What makes HALCON very powerful in the area of 3D measurements is its ability to model the whole imaging process in 3D with the camera calibration (section 3.1 on page 30). Among other things, this allows to transform the image processing results into arbitrary 3D coordinate systems and with this to derive metrical information from the images, regardless of the position and orientation of the camera with respect to the object.

In general, no 3D measurements are possible if only one image of the object is available. But nevertheless, by using HALCON's camera calibration you can perform inspection tasks in 3D coordinates in specified object planes (section 3 on page 29). These planes can be oriented arbitrarily with respect to the camera. This is, e.g., useful if the camera cannot be mounted such that it looks perpendicular to the object surface. In any case, you can perform the image processing in the original images. Afterwards, the results can be transformed into 3D coordinates.

It is also possible to rectify the images such that they appear as if they were acquired from a camera that has no lens distortions and that looks exactly perpendicular onto the object surface (section 3.3 on page 54). This is useful for tasks like OCR or the recognition and localization of objects, which rely on images that are not distorted too much with respect to the training images.

If the object is too large to be covered by one image with the desired resolution, multiple images, each covering only a part of the object, can be combined into one larger mosaic image. This can be done either based on a calibrated camera setup with very high precision (section 4 on page 64) or highly automated for arbitrary and even varying image configurations (section 5 on page 76).

The position and orientation of 3D objects with respect to a given 3D coordinate system can be determined with the HALCON camera calibration (section 6 on page 90). This is, e.g., necessary for pick-and-place applications (3D alignment).

If you need to determine the 3D shape of arbitrary objects, you can use HALCON's binocular stereo vision functionality (section 7 on page 100). The 3D coordinates of any point on the object surface can be determined based on two images that are acquired suitably from different points of view. Thus, 3D reconstruction and 3D inspection becomes possible.

A typical application area for 3D vision is robot vision, i.e., using the results of machine vision to command a robot. In such applications you must perform an additional calibration: the so-called hand-eye calibration, which determines the relation between camera and robot coordinates. Again, this calibration must be performed only once (offline), its results allow you to quickly transform machine vision results from camera into robot coordinates.

Thus, we can answer the question that was posed at the beginning: Yes, you can perform 3D vision with HALCON, but you have to calibrate your camera first. Don't be afraid of the calibration process: In HALCON, this can be done with just a few lines of code.

What is more, if you want to achieve accurate results it is essential to calibrate the camera. It is of no use to extract edges with an accuracy of 1/40 pixel if the lens distortion of the uncalibrated camera accounts for a couple of pixels. This also applies if you use cameras with telecentric lenses.

We propose to read section 2.2 on page 19 first, as the camera model is described there. Then, depending on the task at hand, you can step into the appropriate section. To find this section, you can use the overview given in figure 1 and figure 2. For details on 3D transformations and poses, please refer to section 2.1 on page 8.

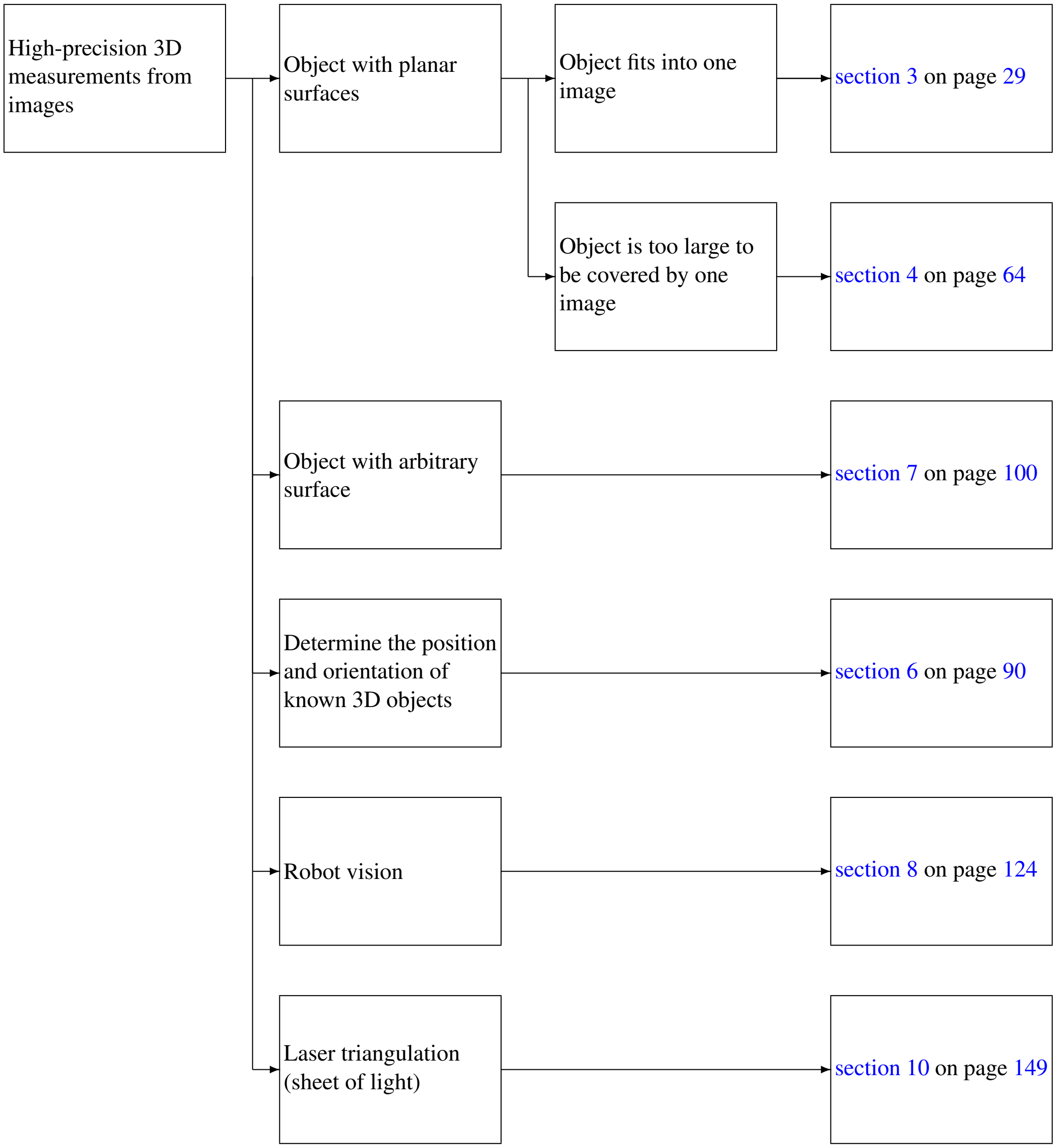

High-precision 3D measurements from images
Object with planar surfaces
Object fits into one image
section 3 on page 29
Object is too large to be covered by one image
section 4 on page 64
Object with arbitrary surface
section 7 on page 100
Determine the position and orientation of known 3D objects
section 6 on page 90
Robot vision
section 8 on page 124
Laser triangulation (sheet of light)
section 10 on page 149

Figure 1: How to find the appropriate section of this Solution Guide (part 1).

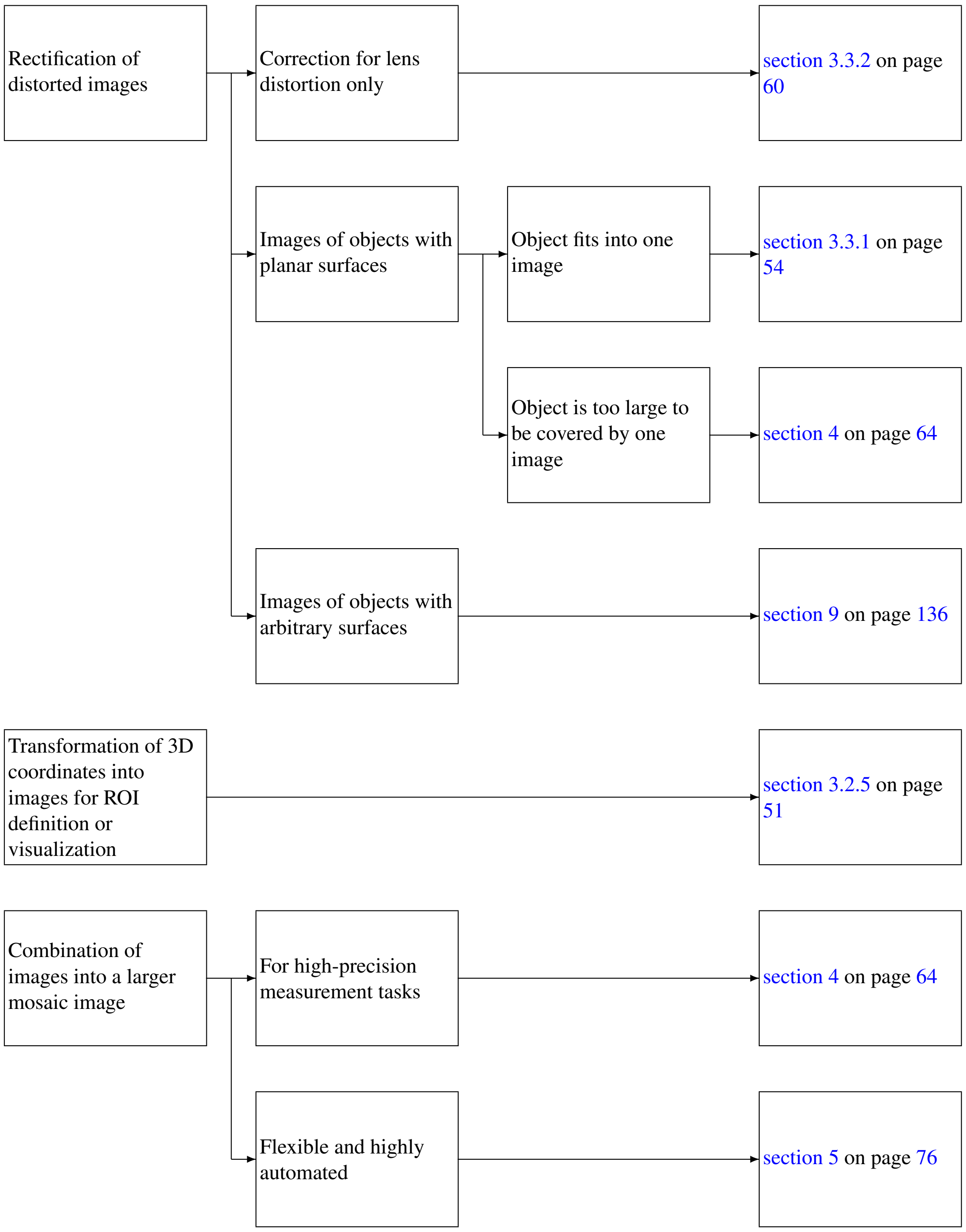

Rectification of distorted images
Correction for lens distortion only
section 3.3.2 on page 60
Images of objects with planar surfaces
Object fits into one image
section 3.3.1 on page 54
Object is too large to be covered by one image
section 4 on page 64
Images of objects with arbitrary surfaces
section 9 on page 136
Transformation of 3D coordinates into images for ROI definition or visualization
section 3.2.5 on page 51
Combination of images into a larger mosaic image
For high-precision measurement tasks
section 4 on page 64
Flexible and highly automated
section 5 on page 76

Figure 2: How to find the appropriate section of this Solution Guide (part 2).

2 Basics

2.1 3D Transformations and Poses

Before we start explaining how to perform 3D vision with HALCON, we take a closer look at some basic questions regarding the use of 3D coordinates:

- How to describe the transformation (translation and rotation) of points and coordinate systems,
- how to describe the position and orientation of one coordinate system relative to another, and
- how to determine the coordinates of a point in different coordinate systems, i.e., how to transform coordinates between coordinate systems.

In fact, all these tasks can be solved using one and the same means: homogeneous transformation matrices and their more compact equivalent, 3D poses.

2.1.1 3D Coordinates

The position of a 3D point P is described by its three coordinates (x_p, y_p, z_p). The coordinates can also be interpreted as a 3D vector (indicated by a bold-face lower-case letter). The coordinate system in which the point coordinates are given is indicated to the upper right of a vector or coordinate. For example, the coordinates of the point P in the camera coordinate system (denoted by the letter c) and in the world coordinate system (denoted by the letter w) would be written as:

$$\mathbf{p}^c = \begin{pmatrix} x_p^c \\ y_p^c \\ z_p^c \end{pmatrix} \qquad \mathbf{p}^w = \begin{pmatrix} x_p^w \\ y_p^w \\ z_p^w \end{pmatrix}$$

Figure 3 depicts an example point lying in a plane where measurements are to be performed and its coordinates in the camera and world coordinate system, respectively.

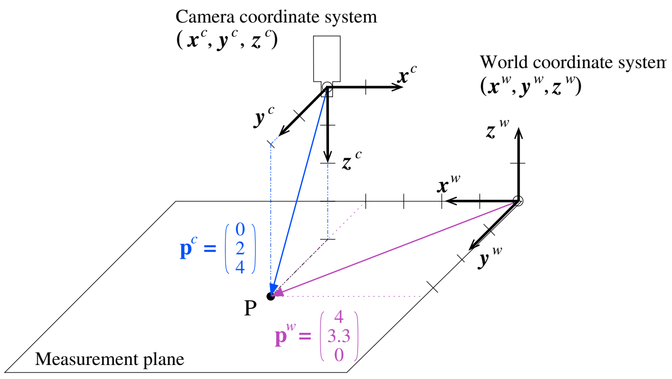


Figure 3: Coordinates of a point in two different coordinate systems.

2.1.2 Translation

Translation of Points

In figure 4, our example point has been translated along the x-axis of the camera coordinate system.

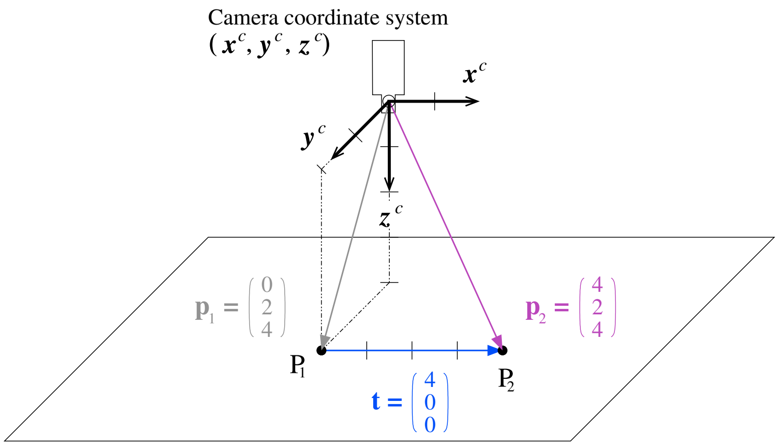

Camera coordinate system
(x^c, y^c, z^c)
x^c
y^c
z^c
$p_1 = \begin{pmatrix}0\\2\\4\end{pmatrix}$
$p_2 = \begin{pmatrix}4\\2\\4\end{pmatrix}$
P_1
P_2
$t = \begin{pmatrix}4\\0\\0\end{pmatrix}$

Figure 4: Translating a point.

The coordinates of the resulting point P_2 can be calculated by adding two vectors, the coordinate vector $\mathbf{p}_1$ of the point and the translation vector $\mathbf{t}$:

$$\mathbf{p}_2 = \mathbf{p}_1 + \mathbf{t} = \begin{pmatrix} x_{p_1} + x_t \\ y_{p_1} + y_t \\ z_{p_1} + z_t \end{pmatrix} \tag{1}$$

Multiple translations are described by adding the translation vectors. This operation is *commutative*, i.e., the sequence of the translations has no influence on the result.

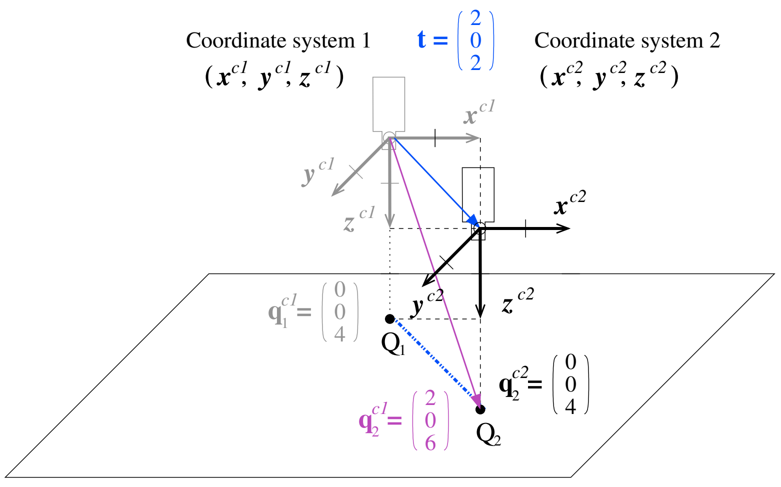
Translation of Coordinate Systems

Coordinate systems can be translated just like points. In the example in figure 5, the coordinate system c_1 is translated to form a second coordinate system, c_2. Then, the position of c_2 in c_1, i.e., the coordinate vector of its origin relative to c_1 ($\mathbf{o}_{c_2}^{c_1}$), is identical to the translation vector:

$$\mathbf{t}^{c_1} = \mathbf{o}_{c_2}^{c_1} \tag{2}$$

Coordinate Transformations

Let's turn to the question how to transform point coordinates between (translated) coordinate systems. In fact, the translation of a point can also be thought of as translating it together with its local coordinate system. This is depicted in figure 5: The coordinate system c_1, together with the point Q_1, is translated



Coordinate system 1
(x^{c1}, y^{c1}, z^{c1})
$\mathbf{t} = \begin{pmatrix} 2 \\ 0 \\ 2 \end{pmatrix}$
Coordinate system 2
(x^{c2}, y^{c2}, z^{c2})
x^{c1}
y^{c1}
z^{c1}
x^{c2}
y^{c2}
z^{c2}
$\mathbf{q}_1^{c1} = \begin{pmatrix} 0 \\ 0 \\ 4 \end{pmatrix}$
Q_1
$\mathbf{q}_2^{c2} = \begin{pmatrix} 0 \\ 0 \\ 4 \end{pmatrix}$
$\mathbf{q}_2^{c1} = \begin{pmatrix} 2 \\ 0 \\ 6 \end{pmatrix}$
Q_2

Figure 5: Translating a coordinate system (and point).

by the vector $\mathbf{t}$, resulting in the coordinate system c_2 and the point Q_2. The points Q_1 and Q_2 then have the same coordinates relative to their local coordinate system, i.e., $\mathbf{q}_1^{c_1} = \mathbf{q}_2^{c_2}$.

If coordinate systems are only translated relative to each other, coordinates can be transformed very easily between them by adding the translation vector:

$$\mathbf{q}_2^{c_1} = \mathbf{q}_2^{c_2} + \mathbf{t}^{c_1} = \mathbf{q}_2^{c_2} + \mathbf{o}_{c_2}^{c_1} \tag{3}$$

In fact, figure 5 visualizes this equation: $\mathbf{q}_2^{c_1}$, i.e., the coordinate vector of Q_2 in the coordinate system c_1, is composed by adding the translation vector $\mathbf{t}$ and the coordinate vector of Q_2 in the coordinate system c_2 ($\mathbf{q}_2^{c_2}$).

The downside of this graphical notation is that, at first glance, the direction of the translation vector appears to be contrary to the direction of the coordinate transformation: The vector points from the coordinate system c_1 to c_2, but transforms coordinates from the coordinate system c_2 to c_1. According to this, the coordinates of Q_1 in the coordinate system c_2, i.e., the inverse transformation, can be obtained by subtracting the translation vector from the coordinates of $Q1$ in the coordinate system c_1:

$$\mathbf{q}_1^{c_2} = \mathbf{q}_1^{c_1} - \mathbf{t}^{c_1} = \mathbf{q}_1^{c_1} - \mathbf{o}_{c_2}^{c_1} \tag{4}$$

Summary

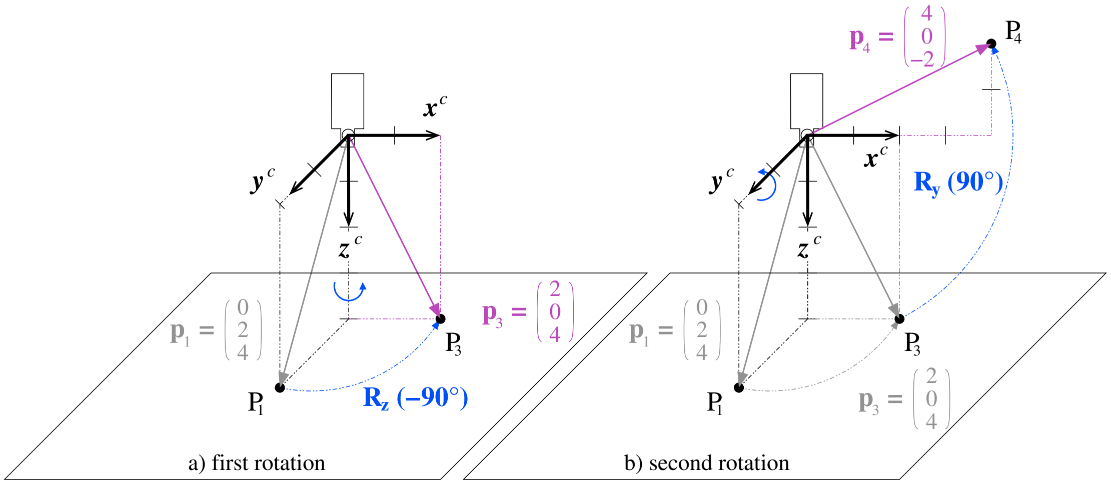
- Points are translated by adding the translation vector to their coordinate vector. Analogously, coordinate systems are translated by adding the translation vector to the position (coordinate vector) of their origin.
- To transform point coordinates from a translated coordinate system c_2 into the original coordinate system c_1, you apply the same transformation to the points that was applied to the coordinate system, i.e., you add the translation vector used to translate the coordinate system c_1 into c_2.

- Multiple translations are described by adding all translation vectors; the sequence of the translations does not affect the result.

2.1.3 Rotation

Rotation of Points

In figure 6a, the point $\mathbf{p}_1$ is rotated by $-90°$ around the z-axis of the camera coordinate system.



$\mathbf{p}_4 = \begin{pmatrix} 4 \\ 0 \\ -2 \end{pmatrix}$
P_4
x^c
y^c
z^c
$\mathbf{R}_y\ (90°)$
$\mathbf{p}_1 = \begin{pmatrix} 0 \\ 2 \\ 4 \end{pmatrix}$
$\mathbf{p}_3 = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$
P_3
P_1
$\mathbf{R}_z\ (-90°)$
a) first rotation
b) second rotation

Figure 6: Rotate a point: (a) first around z-axis; (b) then around y-axis.

Rotating a point is expressed by multiplying its coordinate vector with a 3×3 rotation matrix $\mathbf{R}$. A rotation around the z-axis looks as follows:

$$\mathbf{p}_3 = \mathbf{R}_z(\gamma) \cdot \mathbf{p}_1 = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x_{p_1} \\ y_{p_1} \\ z_{p_1} \end{pmatrix} = \begin{pmatrix} \cos\gamma \cdot x_{p_1} - \sin\gamma \cdot y_{p_1} \\ \sin\gamma \cdot x_{p_1} + \cos\gamma \cdot y_{p_1} \\ z_{p_1} \end{pmatrix} \tag{5}$$

Rotations around the x- and y-axis correspond to the following rotation matrices:

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \qquad \mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{6}$$

Chain of Rotations

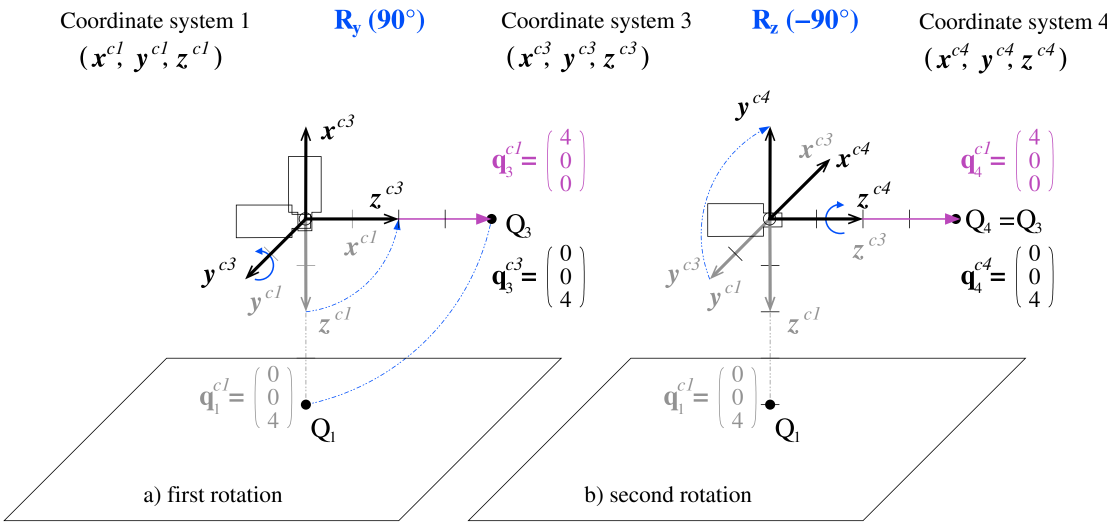
In figure 6b, the rotated point is further rotated around the y-axis. Such a chain of rotations can be expressed very elegantly by a chain of rotation matrices:

$$\mathbf{p}_4 = \mathbf{R}_y(\beta) \cdot \mathbf{p}_3 = \mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma) \cdot \mathbf{p}_1 \tag{7}$$

Note that in contrast to a multiplication of scalars, the multiplication of matrices is not commutative, i.e., if you change the sequence of the rotation matrices, you get a different result.

Rotation of Coordinate Systems

In contrast to points, coordinate systems have an orientation relative to other coordinates systems. This orientation changes when the coordinate system is rotated. For example, in figure 7a the coordinate system c_3 has been rotated around the y-axis of the coordinate system c_1, resulting in a different orientation of the camera. Note that in order to rotate a coordinate system in your mind's eye, it may help to image the points of the axis vectors being rotated.



Coordinate system 1
(x^{c1}, y^{c1}, z^{c1})
$\mathbf{R}_y\,(90°)$
Coordinate system 3
(x^{c3}, y^{c3}, z^{c3})
$\mathbf{R}_z\,(-90°)$
Coordinate system 4
(x^{c4}, y^{c4}, z^{c4})
x^{c3}
z^{c3}
y^{c3}
x^{c1}
y^{c1}
z^{c1}
$\mathbf{q}_3^{c1} = \begin{pmatrix}4\\0\\0\end{pmatrix}$
Q_3
$\mathbf{q}_3^{c3} = \begin{pmatrix}0\\0\\4\end{pmatrix}$
$\mathbf{q}_1^{c1} = \begin{pmatrix}0\\0\\4\end{pmatrix}$
Q_1
a) first rotation
y^{c4}
x^{c3}
x^{c4}
z^{c4}
z^{c3}
y^{c3}
y^{c1}
z^{c1}
$\mathbf{q}_4^{c1} = \begin{pmatrix}4\\0\\0\end{pmatrix}$
$Q_4 = Q_3$
$\mathbf{q}_4^{c4} = \begin{pmatrix}0\\0\\4\end{pmatrix}$
$\mathbf{q}_1^{c1} = \begin{pmatrix}0\\0\\4\end{pmatrix}$
Q_1
b) second rotation

Figure 7: Rotate coordinate system with point: (a) first around z-axis; (b) then around y-axis.

Just like the position of a coordinate system can be expressed directly by the translation vector (see equation 2 on page 9), the orientation is contained in the rotation matrix: The columns of the rotation matrix correspond to the axis vectors of the rotated coordinate system in coordinates of the original one:

$$\mathbf{R} = \begin{bmatrix} \mathbf{x}_{c_3}^{c_1} & \mathbf{y}_{c_3}^{c_1} & \mathbf{z}_{c_3}^{c_1} \end{bmatrix} \tag{8}$$

For example, the axis vectors of the coordinate system c_3 in figure 7a can be determined from the corresponding rotation matrix $\mathbf{R}_y(90°)$ as shown in the following equation; you can easily check the result in the figure.

$$\mathbf{R}_y(90°) = \begin{bmatrix} \cos(90°) & 0 & \sin(90°) \\ 0 & 1 & 0 \\ -\sin(90°) & 0 & \cos(90°) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$$

$$\Rightarrow \quad \mathbf{x}_{c_3}^{c_1} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \quad \mathbf{y}_{c_3}^{c_1} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad \mathbf{z}_{c_3}^{c_1} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Coordinate Transformations

Like in the case of translation, to transform point coordinates from a rotated coordinate system c_3 into the original coordinate system c_1, you apply the same transformation to the points that was applied to the coordinate system c_3, i.e., you multiply the point coordinates with the rotation vector used to rotate the coordinate system c_1 into c_3:

$$\mathbf{q}_3^{c_1} = {}^{c_1}\mathbf{R}_{c_3} \cdot \mathbf{q}_3^{c_3} \tag{9}$$

This is depicted in figure 7 also for a chain of rotations, which corresponds to the following equation:

$$\mathbf{q}_4^{c_1} = {}^{c_1}\mathbf{R}_{c_3} \cdot {}^{c_3}\mathbf{R}_{c_4} \cdot \mathbf{q}_4^{c_4} = \mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma) \cdot \mathbf{q}_4^{c_4} = {}^{c_1}\mathbf{R}_{c_4} \cdot \mathbf{q}_4^{c_4} \tag{10}$$

In Which Sequence and Around Which Axes are Rotations Performed?

If you compare the chains of rotations in figure 6 and figure 7 and the corresponding equations 7 and 10, you will note that two different sequences of rotations are described by the same chain of rotation matrices: In figure 6, the point was rotated *first* around the *z-axis* and *then* around the *y-axis*, whereas in figure 7 the coordinate system is rotated *first* around the *y-axis* and *then* around the *z-axis*. Yet, both are described by the chain $\mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma)$!

The solution to this seemingly paradox situation is that in the two examples the chain of rotation matrices can be "read" in different directions: In figure 6 it is read from the right to left, and in figure 7 from left to the right.

However, there still must be a difference between the two sequences because, as we already mentioned, the multiplication of rotation matrices is not commutative. This difference lies in the second question in the title, i.e., around which axes the rotations are performed.

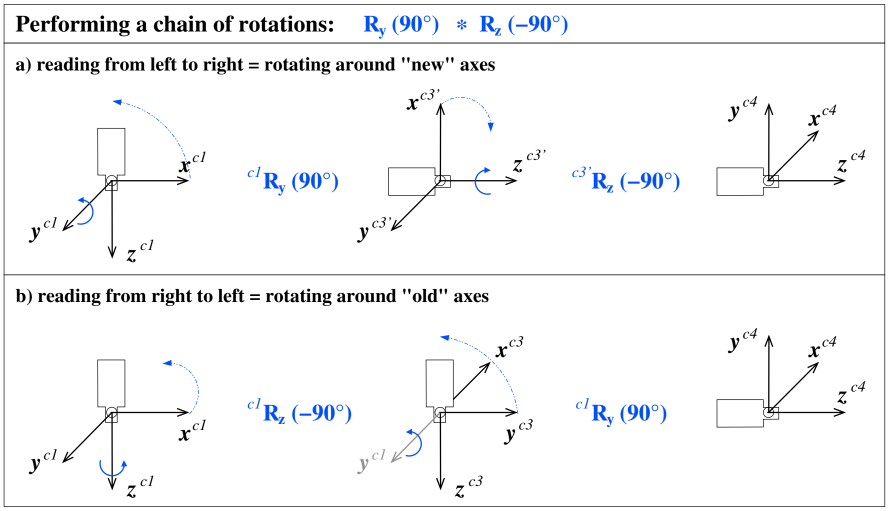

Performing a chain of rotations: $\mathbf{R}_y$ (90°) * $\mathbf{R}_z$ (–90°)
a) reading from left to right = rotating around "new" axes
x^{c1}
y^{c1}
z^{c1}
${}^{c1}\mathbf{R}_y$ (90°)
$x^{c3'}$
$z^{c3'}$
$y^{c3'}$
${}^{c3'}\mathbf{R}_z$ (–90°)
y^{c4}
x^{c4}
z^{c4}
b) reading from right to left = rotating around "old" axes
x^{c1}
y^{c1}
z^{c1}
${}^{c1}\mathbf{R}_z$ (–90°)
x^{c3}
y^{c3}
y^{c1}
z^{c3}
${}^{c1}\mathbf{R}_y$ (90°)
y^{c4}
x^{c4}
z^{c4}

Figure 8: Performing a chain of rotations (a) from left to the right, or (b) from right to left.

Let's start with the second rotation of the coordinate system in figure 7b. Here, there are two possible sets of axes to rotate around: those of the "old" coordinate system c_1 and those of the already rotated, "new" coordinate system c_3. In the example, the second rotation is performed around the "new" z-axis.

In contrast, when rotating points as in figure 6, there is only one set of axes around which to rotate: those of the "old" coordinate system.

From this, we derive the following rules:

- When reading a chain from the **left to right**, rotations are performed around the **"new"** axes.
- When reading a chain from the **right to left**, rotations are performed around the **"old"** axes.

As already remarked, point rotation chains are always read from right to left. In the case of coordinate systems, you have the choice how to read a rotation chain. In most cases, however, it is more intuitive to read them from left to right.

Figure 8 shows that the two reading directions really yield the same result.

Summary

- Points are rotated by multiplying their coordinate vector with a rotation matrix.
- If you rotate a coordinate system, the rotation matrix describes its resulting orientation: The column vectors of the matrix correspond to the axis vectors of the rotated coordinate system in coordinates of the original one.
- To transform point coordinates from a rotated coordinate system c_3 into the original coordinate system c_1, you apply the same transformation to the points that was applied to the coordinate system, i.e., you multiply them with the rotation matrix that was used to rotate the coordinate system c_1 into c_3.
- Multiple rotations are described by a chain of rotation matrices, which can be read in two directions. When read from left to right, rotations are performed around the "new" axes; when read from right to left, the rotations are performed around the "old" axes.

2.1.4 Rigid Transformations and Homogeneous Transformation Matrices

Rigid Transformation of Points

If you combine translation and rotation, you get a so-called rigid transformation. For example, in figure 9, the translation and rotation of the point from figures 4 and 6 are combined. Such a transformation is described as follows:

$$\mathbf{p}_5 = \mathbf{R} \cdot \mathbf{p}_1 + \mathbf{t} \tag{11}$$

For multiple transformations, such equations quickly become confusing, as the following example with two transformations shows:

$$\mathbf{p}_6 = \mathbf{R}_a \cdot (\mathbf{R}_b \cdot \mathbf{p}_1 + \mathbf{t}_b) + \mathbf{t}_a = \mathbf{R}_a \cdot \mathbf{R}_b \cdot \mathbf{p}_1 + \mathbf{R}_a \cdot \mathbf{t}_b + \mathbf{t}_a \tag{12}$$

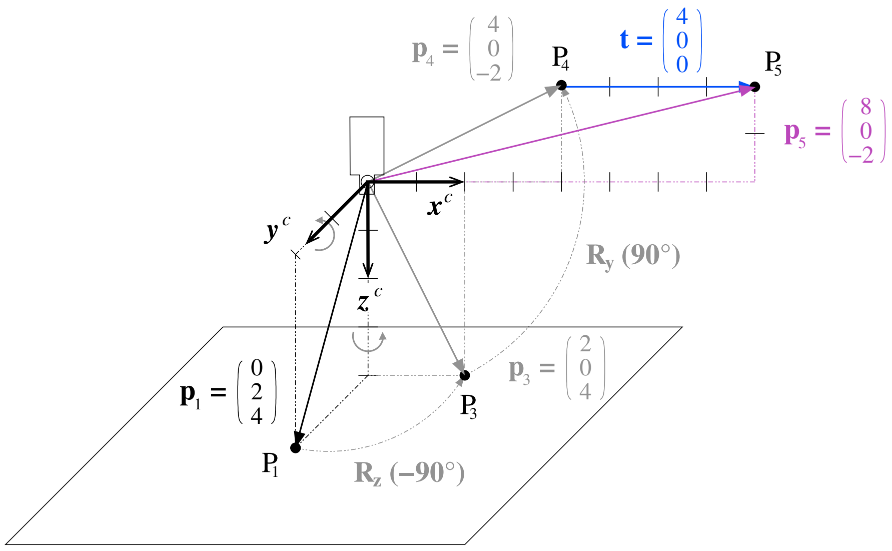

$\mathbf{p}_4 = \begin{pmatrix} 4 \\ 0 \\ -2 \end{pmatrix}$
P_4
$\mathbf{t} = \begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix}$
P_5
$\mathbf{p}_5 = \begin{pmatrix} 8 \\ 0 \\ -2 \end{pmatrix}$
x^c
y^c
z^c
$R_y\ (90°)$
$\mathbf{p}_3 = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$
P_3
$\mathbf{p}_1 = \begin{pmatrix} 0 \\ 2 \\ 4 \end{pmatrix}$
P_1
$R_z\ (-90°)$

Figure 9: Combining the translation from figure 4 on page 9 and the rotation of figure 6 on page 11 to form a rigid transformation.

An elegant alternative is to use so-called *homogeneous transformation matrices* and the corresponding homogeneous vectors. A homogeneous transformation matrix $\mathbf{H}$ contains both the rotation matrix and the translation vector. For example, the rigid transformation from equation 11 can be rewritten as follows:

$$\begin{pmatrix} \mathbf{p}_5 \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0\,0\,0 & 1 \end{bmatrix} \cdot \begin{pmatrix} \mathbf{p}_1 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} \cdot \mathbf{p}_1 + \mathbf{t} \\ 1 \end{pmatrix} = \mathbf{H} \cdot \mathbf{p}_1 \tag{13}$$

The usefulness of this notation becomes apparent when dealing with sequences of rigid transformations, which can be expressed as chains of homogeneous transformation matrices, similarly to the rotation chains:

$$\mathbf{H}_1 \cdot \mathbf{H}_2 = \begin{bmatrix} \mathbf{R}_a & \mathbf{t}_a \\ 0\,0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}_b & \mathbf{t}_b \\ 0\,0\,0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_a \cdot \mathbf{R}_b & \mathbf{R}_a \cdot \mathbf{t}_b + \mathbf{t}_a \\ 0\,0\,0 & 1 \end{bmatrix} \tag{14}$$

As explained for chains of rotations, chains of rigid transformation can be read in two directions. When reading from left to right, the transformations are performed around the "new" axes, when read from right to left around the "old" axes.

In fact, a rigid transformation is already a chain, since it consists of a translation and a rotation:

$$\mathbf{H} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0\,0\,0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \mathbf{t} \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} & 0 \\ \mathbf{R} & 0 \\ & 0 \\ 0\,0\,0 & 1 \end{bmatrix} = \mathbf{H}(\mathbf{t}) \cdot \mathbf{H}(\mathbf{R}) \tag{15}$$

If the rotation is composed of multiple rotations around axes as in figure 9, the individual rotations can also be written as homogeneous transformation matrices:

$$\mathbf{H} = \begin{bmatrix} \mathbf{R}_y(\beta)\cdot\mathbf{R}_z(\gamma) & \mathbf{t} \\ 0\,0\,0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \mathbf{t} \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} & 0 \\ \mathbf{R}_y(\beta) & 0 \\ & 0 \\ 0\,0\,0 & 1 \end{bmatrix} \cdot \begin{bmatrix} & 0 \\ \mathbf{R}_z(\gamma) & 0 \\ & 0 \\ 0\,0\,0 & 1 \end{bmatrix}$$

Reading this chain from right to left, you can follow the transformation of the point in figure 9: First, it is rotated around the z-axis, then around the ("old") y-axis, and finally it is translated.

Rigid Transformation of Coordinate Systems

Rigid transformations of coordinate systems work along the same lines as described for a separate translation and rotation. This means that the homogeneous transformation matrix ${}^{c_1}\mathbf{H}_{c_5}$ describes the transformation of the coordinate system c_1 into the coordinate system c_5. At the same time, it describes the position and orientation of coordinate system c_5 relative to coordinate system c_1: Its column vectors contain the coordinates of the axis vectors and the origin.

$${}^{c_1}\mathbf{H}_{c_5} = \begin{bmatrix} \mathbf{x}_{c_5}^{c_1} & \mathbf{y}_{c_5}^{c_1} & \mathbf{z}_{c_5}^{c_1} & \mathbf{o}_{c_5}^{c_1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{16}$$

As already noted for rotations, chains of rigid transformations of coordinate systems are typically read from left to right. Thus, the chain above can be read as first translating the coordinate system, then rotating it around its "new" y-axis, and finally rotating it around its "newest" z-axis.

Coordinate Transformations

As described for the separate translation and the rotation, to transform point coordinates from a rigidly transformed coordinate system c_5 into the original coordinate system c_1, you apply the same transformation to the points that was applied to the coordinate system c_5, i.e., you multiply the point coordinates with the homogeneous transformation matrix:

$$\begin{pmatrix} \mathbf{p}_5^{c_1} \\ 1 \end{pmatrix} = {}^{c_1}\mathbf{H}_{c_5} \cdot \begin{pmatrix} \mathbf{p}_5^{c_5} \\ 1 \end{pmatrix} \tag{17}$$

Typically, you leave out the homogeneous vectors if there is no danger of confusion and simply write:

$$\mathbf{p}_5^{c_1} = {}^{c_1}\mathbf{H}_{c_5} \cdot \mathbf{p}_5^{c_5} \tag{18}$$

Summary

- Rigid transformations consist of a rotation and a translation. They are described very elegantly by homogeneous transformation matrices, which contain both the rotation matrix and the translation vector.
- Points are transformed by multiplying their coordinate vector with the homogeneous transformation matrix.

- If you transform a coordinate system, the homogeneous transformation matrix describes the coordinate system's resulting position and orientation: The column vectors of the matrix correspond to the axis vectors and the origin of the coordinate system in coordinates of the original one. Thus, you could say that a homogeneous transformation matrix "is" the position and orientation of a coordinate system.
- To transform point coordinates from a rigidly transformed coordinate system c_5 into the original coordinate system c_1, you apply the same transformation to the points that was applied to the coordinate system, i.e., you multiply them with the homogeneous transformation matrix that was used to transform the coordinate system c_1 into c_5.
- Multiple rigid transformations are described by a chain of transformation matrices, which can be read in two directions. When read from left to the right, rotations are performed around the "new" axes; when read from the right to left, the transformations are performed around the "old" axes.

HALCON Operators

As we already anticipated at the beginning of section 2.1 on page 8, homogeneous transformation matrices are the answer to all our questions regarding the use of 3D coordinates. Because of this, they form the basis for HALCON's operators for 3D transformations. Below, you find a brief overview of the relevant operators. For more details follow the links into the Reference Manual.

- `hom_mat3d_identity` creates the identical transformation
- `hom_mat3d_translate` translates along the "old" axes: $\mathbf{H}_{2} = \mathbf{H}(\mathbf{t}) \cdot \mathbf{H}_{1}$
- `hom_mat3d_translate_local` translates along the "new" axes: $\mathbf{H}_{2} = \mathbf{H}_{1} \cdot \mathbf{H}(\mathbf{t})$
- `hom_mat3d_rotate` rotates around the "old" axes: $\mathbf{H}_{2} = \mathbf{H}(\mathbf{R}) \cdot \mathbf{H}_{1}$
- `hom_mat3d_rotate_local` rotates around the "new" axes: $\mathbf{H}_{2} = \mathbf{H}_{1} \cdot \mathbf{H}(\mathbf{R})$
- `hom_mat3d_compose` multiplies two transformation matrices: $\mathbf{H}_{3} = \mathbf{H}_{1} \cdot \mathbf{H}_{2}$
- `hom_mat3d_invert` inverts a transformation matrix: $\mathbf{H}_{2} = \mathbf{H}_{1}{}^{-1}$
- `affine_trans_point_3d` transforms a point using a transformation matrix: $\mathbf{p}_2 = \mathbf{H}_{0} \cdot \mathbf{p}_1$

2.1.5 3D Poses

Homogeneous transformation matrices are a very elegant means of describing transformations, but their content, i.e., the elements of the matrix, are often difficult to read, especially the rotation part. This problem is alleviated by using so-called *3D poses*.

A 3D pose is nothing more than an easier-to-understand representation of a rigid transformation: Instead of the 12 elements of the homogeneous transformation matrix, a pose describes the rigid transformation with 6 parameters, 3 for the rotation and 3 for the translation: (`TransX`, `TransY`, `TransZ`, `RotX`, `RotY`, `RotZ`). The main principle behind poses is that even a rotation around an arbitrary axis can always be represented by a sequence of three rotations around the axes of a coordinate system.

In HALCON, you create 3D poses with `create_pose`; to transform between poses and homogeneous matrices you can use `hom_mat3d_to_pose` and `pose_to_hom_mat3d`.

Sequence of Rotations

However, there is more than one way to represent an arbitrary rotation by three parameters. This is reflected by the HALCON operator create_pose, which lets you choose between different pose types with the parameter OrderOfRotation. If you pass the value *'gba'*, the rotation is described by the following chain of rotations:

$$\mathbf{R}_{gba} = \mathbf{R}_x(\texttt{RotX}) \cdot \mathbf{R}_y(\texttt{RotY}) \cdot \mathbf{R}_z(\texttt{RotZ}) \tag{19}$$

You may also choose the inverse order by passing the value *'abg'*:

$$\mathbf{R}_{abg} = \mathbf{R}_z(\texttt{RotZ}) \cdot \mathbf{R}_y(\texttt{RotY}) \cdot \mathbf{R}_x(\texttt{RotX}) \tag{20}$$

For example, the transformation discussed in the previous sections can be represented by the homogeneous transformation matrix

$$\mathbf{H} = \left[\begin{array}{cc} \mathbf{R}_y(\beta) \cdot \mathbf{R}_z(\gamma) & \mathbf{t} \\ 0 \quad 0 \quad 0 & 1 \end{array} \right] = \left[\begin{array}{cccc} \cos\beta \cdot \cos\gamma & -\cos\beta \cdot \sin\gamma & \sin\beta & x_t \\ \sin\gamma & \cos\gamma & 0 & y_t \\ -\sin\beta \cdot \cos\gamma & \sin\beta \cdot \sin\gamma & \cos\beta & z_t \\ 0 & 0 & 0 & 1 \end{array} \right]$$

The corresponding pose with the rotation order *'gba'* is much easier to read:

$$(\texttt{TransX} = x_t, \texttt{TransY} = y_t, \texttt{TransZ} = z_t, \texttt{RotX} = 0, \texttt{RotY} = 90°, \texttt{RotZ} = 90°)$$

If you look closely at figure 7 on page 12, you can see that the rotation can also be described by the sequence $\mathbf{R}_z(90°) \cdot \mathbf{R}_x(90°)$. Thus, the transformation can also be described by the following pose with the rotation order *'abg'*:

$$(\texttt{TransX} = x_t, \texttt{TransY} = y_t, \texttt{TransZ} = z_t, \texttt{RotX} = 90°, \texttt{RotY} = 0, \texttt{RotZ} = 90°)$$

HALCON Operators

Below, the relevant HALCON operators for dealing with 3D poses are briefly described. For more details follow the links into the Reference Manual.

- create_pose creates a pose
- hom_mat3d_to_pose converts a homogeneous transformation matrix into a pose
- pose_to_hom_mat3d converts a pose into a homogeneous transformation matrix
- convert_pose_type changes the pose type
- write_pose writes a pose into a file
- read_pose reads a pose from a file
- set_origin_pose translates a pose along its “new” axes

How to Determine the Pose of a Coordinate System

The previous sections showed how to describe known transformations using translation vectors, rotations matrices, homogeneous transformation matrices, or poses. Sometimes, however, there is another task: How to describe the position and orientation of a coordinate system with a pose. This is necessary, e.g., when you want to use your own calibration object and need to determine starting values for the external camera parameters as described in section 3.1.4 on page 35.

Figure 10 shows how to proceed for a rather simple example. The task is to determine the pose of the world coordinate system from figure 3 on page 8 relative to the camera coordinate system.

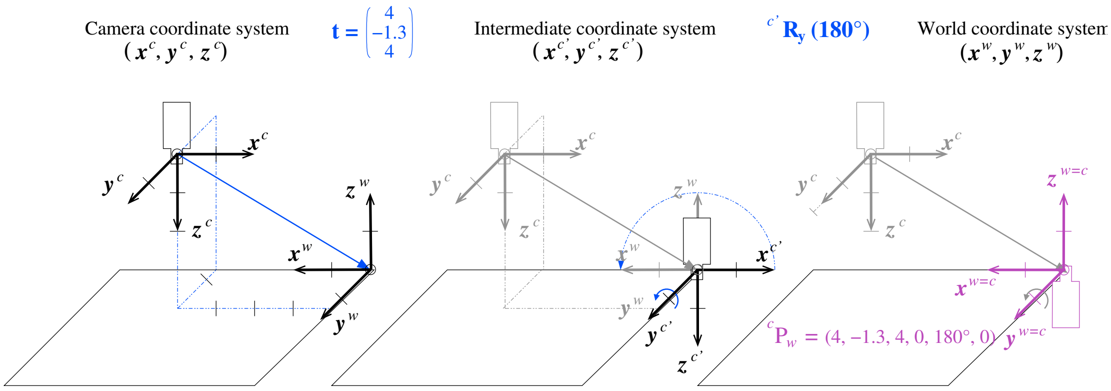

Camera coordinate system
(x^c, y^c, z^c)
$\mathbf{t} = \begin{pmatrix} 4 \\ -1.3 \\ 4 \end{pmatrix}$
Intermediate coordinate system
$(x^{c'}, y^{c'}, z^{c'})$
$^{c'}\mathbf{R}_y\,(180°)$
World coordinate system
(x^w, y^w, z^w)
$^{c}P_w = (4, -1.3, 4, 0, 180°, 0)$

Figure 10: Determining the pose of the world coordinate system in camera coordinates.

In such a case, we recommend to build up the rigid transformation from individual translations and rotations from left to right. Thus, in figure 10 the camera coordinate system is first translated such that its origin coincides with that of the world coordinate system. Now, the y-axes of the two coordinate systems coincide; after rotating the (translated) camera coordinate system around its (new) y-axis by 180°, it has the correct orientation.

2.2 Camera Model and Parameters

If you want to derive *accurate* world coordinates from your imagery, you first have to calibrate your camera. To calibrate a camera, a model for the mapping of the 3D points of the world to the 2D image generated by the camera, lens, and frame grabber is necessary.

HALCON supports the calibration of two different kinds of cameras: area scan cameras and line scan cameras. While area scan cameras acquire the image in one step, line scan cameras generate the image line by line (see Solution Guide II-A, section 6.3 on page 40). Therefore, the line scan camera must move relative to the object during the acquisition process.

Two different types of lenses are relevant for machine vision tasks. The first type of lens effects a perspective projection of the world coordinates into the image, just like the human eye does. With this type of lens, objects become smaller in the image the farther they are away from the camera. This combination of camera and lens is called a *pinhole camera model* because the perspective projection can also be achieved if a small hole is drilled in a thin planar object and this plane is held parallel in front of another plane (the image plane).

The second type of lens that is relevant for machine vision is called a telecentric lens. Its major difference is that it effects a parallel projection of the world coordinates onto the image plane (for a certain range of distances of the object from the camera). This means that objects have the same size in the image independent of their distance to the camera. This combination of camera and lens is called a *telecentric camera model*.

In the following, first the camera model for area scan cameras is described in detail, then, the camera model for line scan cameras is explained.

2.2.1 Area scan cameras

Figure 11 displays the perspective projection effected by a pinhole camera graphically. The world point P is projected through the optical center of the lens to the point P' in the image plane, which is located at a distance of f (the focal length) behind the optical center. Actually, the term "focal length" is not quite correct and would be appropriate only for an infinite object distance. To simplify matters, in the following always the term "focal length" is used even if the "image distance" is meant. Note that the focal length and thus the focus must not be changed after applying the camera calibration.

Although the image plane in reality lies behind the optical center of the lens, it is easier to pretend that it lies at a distance of f in front of the optical center, as shown in figure 12. This causes the image coordinate system to be aligned with the pixel coordinate system (row coordinates increase downward and column coordinates to the right) and simplifies most calculations.

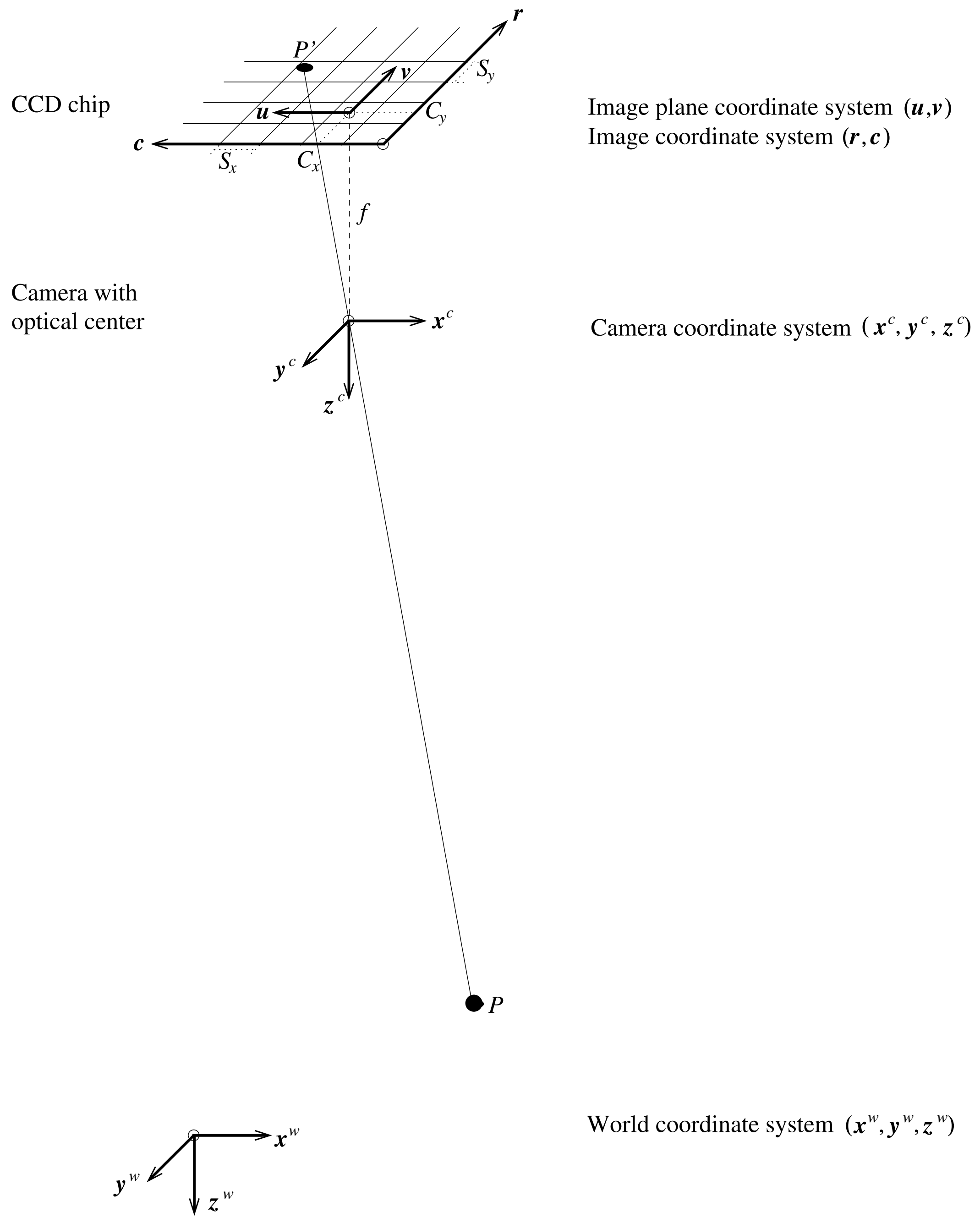
Transformation into camera coordinates (external camera parameters)

With this, we are now ready to describe the projection of objects in 3D world coordinates to the 2D image plane and the corresponding camera parameters. First, we should note that the points P are given in a world coordinate system (WCS). To make the projection into the image plane possible, they need to be transformed into the camera coordinate system (CCS). The CCS is defined so that its x and y axes are parallel to the column and row axes of the image, respectively, and the z axis is perpendicular to the image plane.

The transformation from the WCS to the CCS is a rigid transformation, which can be expressed by a pose or, equivalently, by the homogeneous transformation matrix ${}^{c}\mathbf{H}_{w}$. Therefore, the camera coordinates $\mathbf{p}^c = (\mathbf{x}^c, \mathbf{y}^c, \mathbf{z}^c)^T$ of point P can be calculated from its world coordinates $\mathbf{p}^w = (\mathbf{x}^w, \mathbf{y}^w, \mathbf{z}^w)^T$ simply by

$$\mathbf{p}^c = {}^{c}\mathbf{H}_{w} \cdot \mathbf{p}^w \tag{21}$$

The six parameters of this transformation (the three translations t_x, t_y, and t_z and the three rotations α, β, and γ) are called the *external camera parameters* because they determine the position of the camera with respect to the world. In HALCON, they are stored as a pose, i.e, together with a code that describes the order of translation and rotations.



CCD chip
Image plane coordinate system $(\boldsymbol{u},\boldsymbol{v})$
Image coordinate system $(\boldsymbol{r},\boldsymbol{c})$
P'
$\boldsymbol{r}$
$\boldsymbol{v}$
$\boldsymbol{u}$
$\boldsymbol{c}$
S_y
C_y
S_x
C_x
f
Camera with optical center
Camera coordinate system $(\boldsymbol{x}^c, \boldsymbol{y}^c, \boldsymbol{z}^c)$
$\boldsymbol{x}^c$
$\boldsymbol{y}^c$
$\boldsymbol{z}^c$
P
World coordinate system $(\boldsymbol{x}^w, \boldsymbol{y}^w, \boldsymbol{z}^w)$
$\boldsymbol{x}^w$
$\boldsymbol{y}^w$
$\boldsymbol{z}^w$

Figure 11: Perspective projection by a pinhole camera.

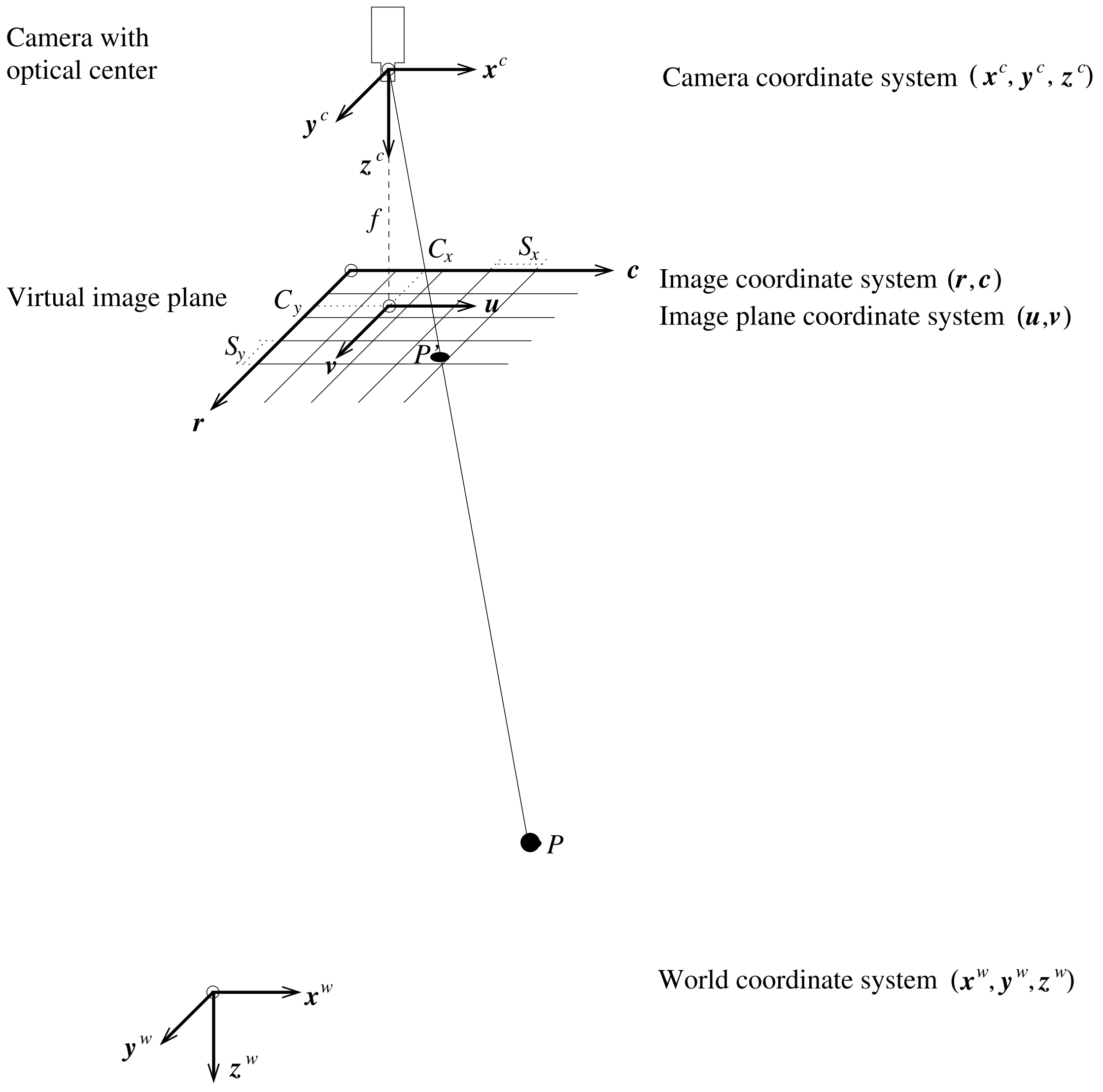

Camera with
optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
f
C_x
S_x
c
Virtual image plane
C_y
u
Image coordinate system (r, c)
Image plane coordinate system (u, v)
S_y
v
P'
r
P
World coordinate system (x^w, y^w, z^w)
x^w
y^w
z^w

Figure 12: Image plane and virtual image plane.

Projection

The next step is the projection of the 3D point given in the CCS into the image plane coordinate system (IPCS). For the pinhole camera model, the projection is a perspective projection, which is given by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{f}{\mathbf{z}^c} \begin{pmatrix} \mathbf{x}^c \\ \mathbf{y}^c \end{pmatrix} \tag{22}$$

For the telecentric camera model, the projection is a parallel projection, which is given by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \mathbf{x}^c \\ \mathbf{y}^c \end{pmatrix} \tag{23}$$

As can be seen, there is no focal length f for telecentric cameras. Furthermore, the distance z of the object to the camera has no influence on the image coordinates.

Lens distortion

After the projection into the image plane, the lens distortion causes the coordinates $(u, v)^T$ to be modified. If no lens distortions were present, the projected point P' would lie on a straight line from P through the optical center, indicated by the dotted line in figure 13. Lens distortions cause the point P' to lie at a different position.

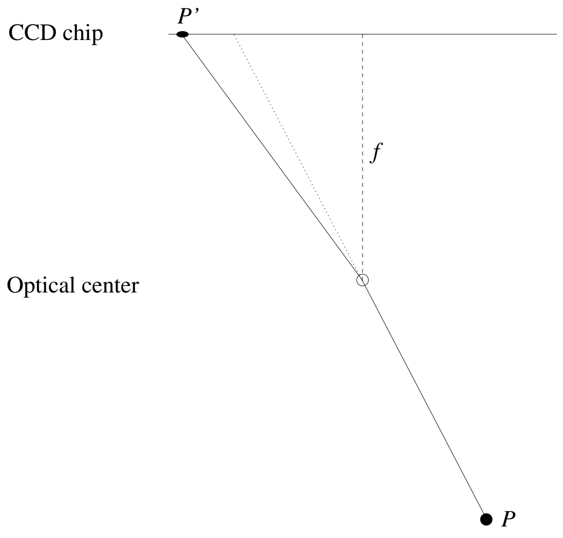


Figure 13: Schematic illustration of the effect of the lens distortion.

The lens distortion is a transformation that can be modeled in the image plane alone, i.e., 3D information is unnecessary. In HALCON, the distortions can be modeled either by the division model or by the polynomial model.

The division model uses one parameter (κ) to model the radial distortions. The following equations transform the distorted image plane coordinates into undistorted image plane coordinates if the division model is used:

$$u = \frac{\tilde{u}}{1 + \kappa(\tilde{u}^2 + \tilde{v}^2)} \quad \text{and} \quad v = \frac{\tilde{v}}{1 + \kappa(\tilde{u}^2 + \tilde{v}^2)} \tag{24}$$

These equations can be inverted analytically, which leads to the following equations that transform undistorted coordinates into distorted coordinates if the division model is used:

$$\tilde{u} = \frac{2u}{1+\sqrt{1-4\kappa(u^2+v^2)}} \quad \text{and} \quad \tilde{v} = \frac{2v}{1+\sqrt{1-4\kappa(u^2+v^2)}} \tag{25}$$

The parameter κ models the magnitude of the radial distortions. If κ is negative, the distortion is barrel-shaped, while for positive κ it is pincushion-shaped (see figure 14).

The polynomial model uses three parameters (K_1, K_2, K_3) to model the radial distortions, and two parameters (P_1, P_2) to model the decentering distortions. The following equations transform the distorted image plane coordinates into undistorted image plane coordinates if the polynomial model is used:

$$u = \tilde{u} + \tilde{u}(K_1 r^2 + K_2 r^4 + K_3 r^6) + 2P_1\tilde{u}\tilde{v} + P_2(r^2 + 2\tilde{u}^2) \tag{26}$$

$$v = \tilde{v} + \tilde{v}(K_1 r^2 + K_2 r^4 + K_3 r^6) + P_1(r^2 + 2\tilde{v}^2) + 2P_2\tilde{u}\tilde{v} \tag{27}$$

with $r = \sqrt{\tilde{u}^2 + \tilde{v}^2}$. These equations cannot be inverted analytically. Therefore, distorted image plane coordinates must be calculated from undistorted image plane coordinates numerically.

The parameters K_1, K_2, and K_3 model the magnitude of the radial distortions and the parameters P_1 and P_2 model the magnitude of the decentering distortions. Some examples for the kind of distortions that can be modeled with the polynomial model are shown in figure 15.

Transformation into pixel coordinates

Finally, the point $(\tilde{u}, \tilde{v})^T$ is transformed from the image plane coordinate system into the image coordinate system (the pixel coordinate system):

$$\begin{pmatrix} r \\ c \end{pmatrix} = \begin{pmatrix} \frac{\tilde{v}}{S_y} + C_y \\ \frac{\tilde{u}}{S_x} + C_x \end{pmatrix} \tag{28}$$

Here, S_x and S_y are scaling factors. For pinhole cameras, they represent the horizontal and vertical distance of the sensors elements on the CCD chip of the camera. For cameras with telecentric lenses, they represent the size of a pixel in world coordinates (not taking into account the lens distortions). The point $(C_x, C_y)^T$ is the principal point of the image. For pinhole cameras, this is the perpendicular projection of the optical center onto the image plane, i.e., the point in the image from which a ray through the optical center is perpendicular to the image plane. It also defines the center of the radial distortions. For telecentric cameras, no optical center exists. Therefore, the principal point is solely defined by the radial distortions.

The six parameters $(f, \kappa, S_x, S_y, C_x, C_y)$ or the ten parameters $(f, K_1, K_2, K_3, P_1, P_2, S_x, S_y, C_x, C_y)$ of the pinhole camera and the five parameters $(\kappa, S_x, S_y, C_x, C_y)$ or the nine parameters $(K_1, K_2, K_3, P_1, P_2, S_x, S_y, C_x, C_y)$ of the telecentric camera are called the *internal camera parameters* because they determine the projection from 3D to 2D performed by the camera.

Note that, in HALCON, the internal camera parameters always contain the focal length and, in addition to the above described parameters, the width (*NumColumns*) and the height (*NumRows*) of the image. The differentiation among the two camera models (pinhole and telecentric) is done based on the value of the focal length. If it has a positive value, a pinhole camera with the given focal length is assumed. If

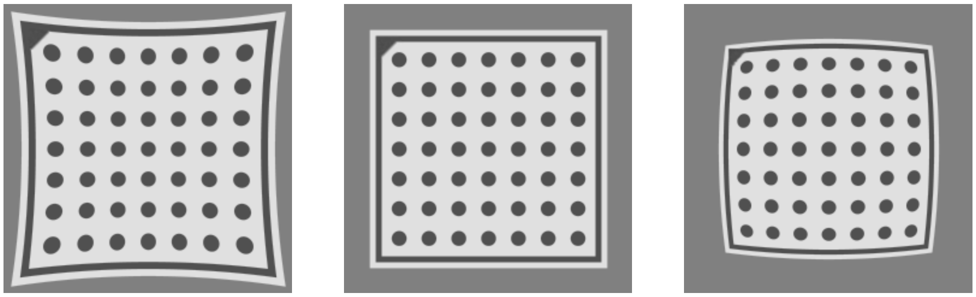


Figure 14: Effect of radial distortions modeled with the division model with $\kappa > 0$ (left), $\kappa = 0$ (middle), and $\kappa < 0$ (right).

the focal length is set to zero, the telecentric camera model is used. The differentiation among the two models for the lens distortion (division model and polynomial model) is done based on the number of values of the internal camera parameters. If eight values are passed, the division model is used. If twelve values are passed, the polynomial model is used.

With this, we can see that camera calibration is the process of determining the internal camera parameters $(f, \kappa, S_x, S_y, C_x, C_y)$ and the external camera parameters $(t_x, t_y, t_z, \alpha, \beta, \gamma)$.

2.2.2 Line scan cameras

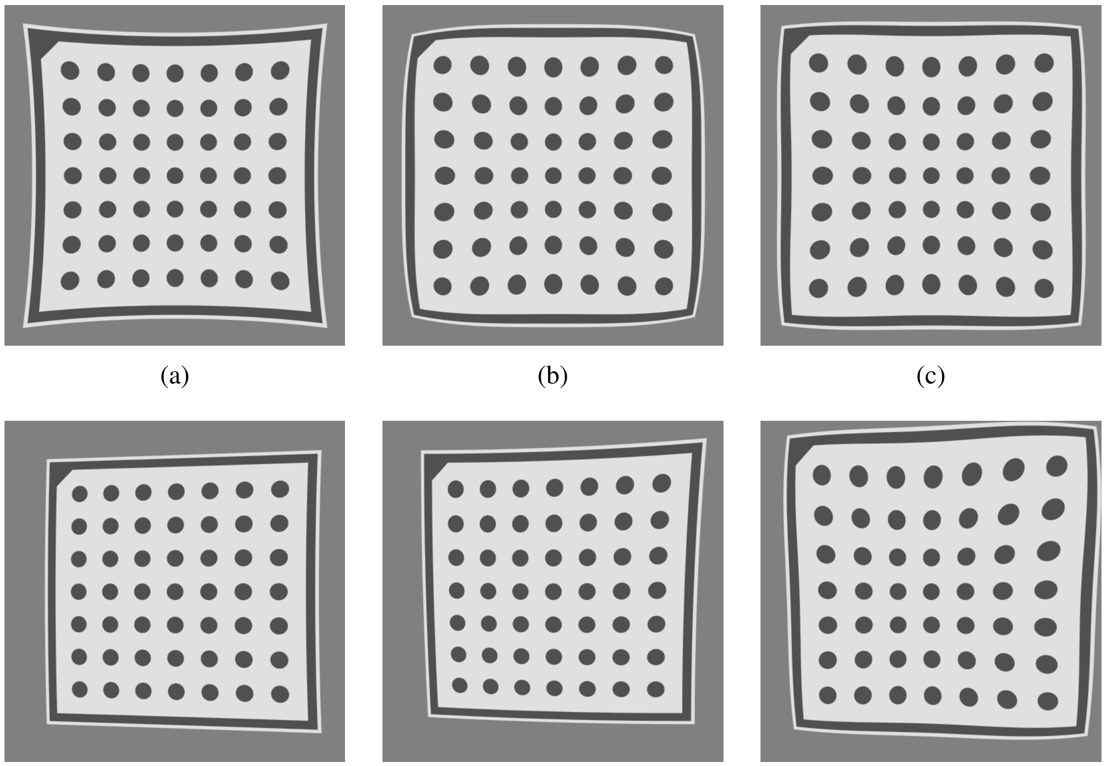
A line scan camera has only a one-dimensional line of sensor elements, i.e., to acquire an image, the camera must move relative to the object (see figure 16 on page 27). This means that the camera moves over a fixed object, the object travels in front of a fixed camera, or camera and object are both moving.

The relative motion between the camera and the object is modeled in HALCON as part of the internal camera parameters. In HALCON, the following assumptions for this motion are made:

1. the camera moves — relative to the object — with constant velocity along a straight line
2. the orientation of the camera is constant with respect to the object
3. the motion is equal for all images

The motion is described by the motion vector $V = (V_x, V_y, V_z)^T$, which must be given in [meters/scanline] in the camera coordinate system. The motion vector describes the motion of the camera, i.e., it assumes a fixed object. In fact, this is equivalent to the assumption of a fixed camera with the object traveling along $-V$.

The camera coordinate system of line scan cameras is defined as follows (see figure 17 on page 28): The origin of the coordinate system is the center of projection. The z-axis is identical to the optical axis and it is directed so that the visible points have positive z coordinates. The y-axis is perpendicular to the sensor line and to the z-axis. It is directed so that the motion vector has a positive y-component, i.e., if a fixed object is assumed, the y-axis points in the direction in which the camera is moving. The x-axis is perpendicular to the y- and z-axis, so that the x-, y-, and z-axis form a right-handed coordinate system.



	K_1	K_2	K_3	P_1	P_2
(a)	< 0	0	0	0	0
(b)	< 0	> 0	0	0	0
(c)	< 0	> 0	< 0	0	0
(d)	< 0	0	0	< 0	0
(e)	< 0	0	0	< 0	> 0
(f)	< 0	> 0	< 0	< 0	> 0

Figure 15: Effect of distortions modeled with the polynomial model with different values for the parameters K_1, K_2, K_3, P_1, and P_2.

Similarly to area scan cameras, the projection of a point given in world coordinates into the image is modeled in two steps: First, the point is transformed into the camera coordinate system. Then, it is projected into the image.

As the camera moves over the object during the image acquisition, also the camera coordinate system moves relative to the object, i.e., each image line has been imaged from a different position. This means that there would be an individual pose for each image line. To make things easier, in HALCON all transformations from world coordinates into camera coordinates and vice versa are based on the pose of the first image line only. The motion V is taken into account during the projection of the point $\mathbf{p}^c$ into the image.

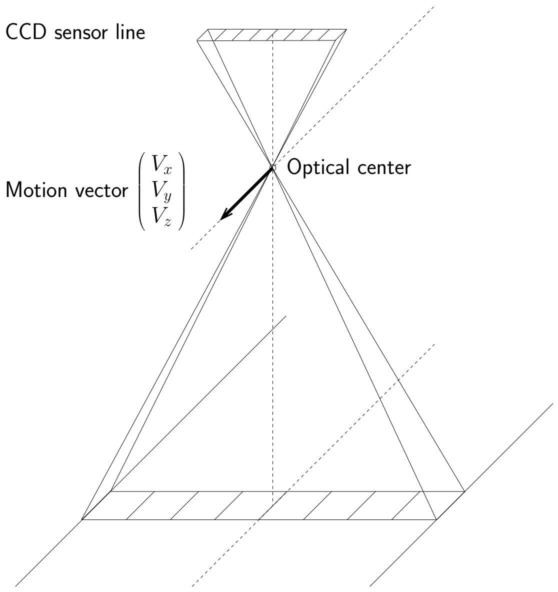

CCD sensor line
Optical center
Motion vector $\begin{pmatrix} V_x \\ V_y \\ V_z \end{pmatrix}$

Figure 16: Principle of line scan image acquisition.

The transformation from the WCS to the CCS of the first image line is a rigid transformation, which can be expressed by a pose or, equivalently, by the homogeneous transformation matrix ${}^{c}\mathbf{H}_{w}$. Therefore, the camera coordinates $\mathbf{p}^c = (\mathbf{x}^c, \mathbf{y}^c, \mathbf{z}^c)^T$ of point P can be calculated from its world coordinates $\mathbf{p}^w = (\mathbf{x}^w, \mathbf{y}^w, \mathbf{z}^w)^T$ simply by

$$\mathbf{p}^c = {}^{c}\mathbf{H}_{w} \cdot \mathbf{p}^w \tag{29}$$

The six parameters of this transformation (the three translations t_x, t_y, and t_z and the three rotations α, β, and γ) are called the *external camera parameters* because they determine the position of the camera with respect to the world. In HALCON, they are stored as a pose, i.e, together with a code that describes the order of translation and rotations.

For line scan cameras, the projection of the point $\mathbf{p}^c$ that is given in the camera coordinate system of the first image line into a (sub-)pixel [r,c] in the image is defined as follows:

Assuming

$$\mathbf{p}^c = \begin{pmatrix} x \\ y \\ z \end{pmatrix},$$

the following set of equations must be solved for m, $\tilde{u}$, and t:

$$\begin{aligned} m \cdot D \cdot \tilde{u} &= x - t \cdot V_x \\ -m \cdot D \cdot p_v &= y - t \cdot V_y \\ m \cdot f &= z - t \cdot V_z \end{aligned}$$

with

$$D = \frac{1}{1 + \kappa(\tilde{u}^2 + {p_v}^2)}$$

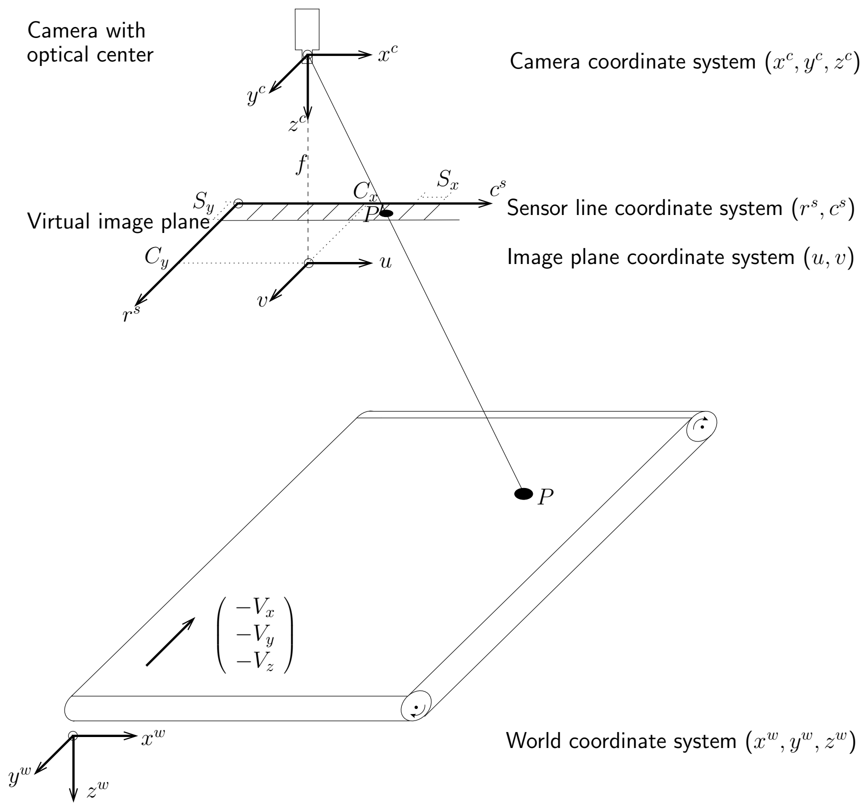

Camera with optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
f
S_x
c^s
S_y
C_x
P
Sensor line coordinate system (r^s, c^s)
Virtual image plane
C_y
u
Image plane coordinate system (u, v)
r^s
v
P
$\begin{pmatrix} -V_x \\ -V_y \\ -V_z \end{pmatrix}$
x^w
y^w
z^w
World coordinate system (x^w, y^w, z^w)

Figure 17: Coordinate systems in regard to a line scan camera.

$$p_v = S_y \cdot C_y$$

This already includes the compensation for radial distortions.

Finally, the point is transformed into the image coordinate system, i.e., the pixel coordinate system:

$$c = \frac{\tilde{u}}{S_x} + C_x \quad \text{and} \quad r = t$$

S_x and S_y are scaling factors. S_x represents the distance of the sensor elements on the CCD line, S_y is the extent of the sensor elements in y-direction. The point $(C_x, C_y)^T$ is the principal point. Note that in contrast to area scan images, $(C_x, C_y)^T$ does not define the position of the principal point in image coordinates. It rather describes the relative position of the principal point with respect to the sensor line.

The nine parameters $(f, \kappa, S_x, S_y, C_x, C_y, V_x, V_y, V_z)$ of the pinhole line scan camera are called the *internal camera parameters* because they determine the projection from 3D to 2D performed by the camera.

As for area scan cameras, the calibration of a line scan camera is the process of determining the internal camera parameters $(f, \kappa, S_x, S_y, C_x, C_y, V_x, V_y, V_z)$ and the external camera parameters $(t_x, t_y, t_z, \alpha, \beta, \gamma)$ of the first image line.

3 3D Vision in a Specified Plane With a Single Camera

In HALCON it is easy to obtain undistorted measurements in world coordinates from images. In general, this can only be done if two or more images of the same object are taken at the same time with cameras at different spatial positions. This is the so-called *stereo* approach; see section 7 on page 100.

In industrial inspection, we often have only one camera available and time constraints do not allow us to use the expensive process of finding corresponding points in the stereo images (the so-called *stereo matching* process).

Nevertheless, it is possible to obtain measurements in world coordinates for objects acquired through telecentric lenses and objects that lie in a known plane, e.g., on an assembly line, for pinhole cameras. Both of these tasks can be solved by intersecting an optical ray (also called line of sight) with a plane.

With this, it is possible to measure objects that lie in a plane, even when the plane is tilted with respect to the optical axis. The only prerequisite is that the camera has been calibrated. In HALCON, the calibration process is very easy as can be seen in the following first example, which introduces the operators that are necessary for the calibration process.

The easiest way to perform the calibration is to use the HALCON standard calibration plates. You just need to take a few images of the calibration plate (see figure 18 for an example), where in one image the calibration plate has been placed directly on the measurement plane.

Note that the calibration plate has an asymmetrical pattern such that the coordinate system can be uniquely determined. Older calibration plates do not have this pattern but you can easily add it by yourself (see appendix A on page 166).

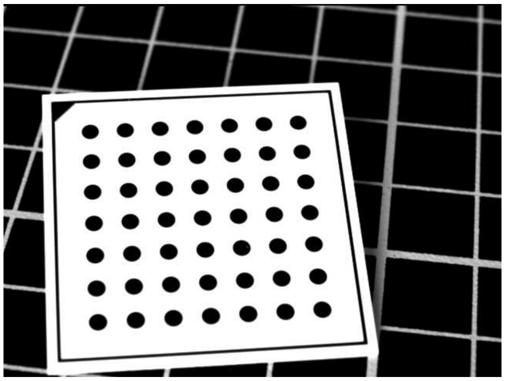


Figure 18: The HALCON calibration plate.

After reading in the calibration images, the operators `find_caltab` and `find_marks_and_pose` can be used to detect the calibration plate and to determine the exact positions of the (dark) calibration targets on it. Additionally, some approximate values are determined, which are necessary for the further processing.

```
find_caltab (Image, Caltab, CaltabName, SizeGauss, MarkThresh, MinDiamMarks)
find_marks_and_pose (Image, Caltab, CaltabName, StartCamPar, StartThresh, \
                     DeltaThresh, MinThresh, Alpha, MinContLength, \
                     MaxDiamMarks, RCoord, CCoord, StartPose)
```

After collecting the positions of the calibration targets and the approximate values for all the calibration images, the operator `camera_calibration` can be called. It determines the internal camera parameters as well as the pose of the calibration plate in each of the calibration images.

```
camera_calibration (X, Y, Z, NRow, NCol, StartCamPar, NStartPose, 'all', \
                    CamParam, NFinalPose, Errors)
```

Now, you can pick the pose of the calibration plate from the image, where the calibration plate has been placed on the measurement plane.

Based on this pose, it is easy to transform image coordinates into world coordinates. For example, to transform point coordinates, the operator `image_points_to_world_plane` can be used.

```
image_points_to_world_plane (CamParam, Pose, Row, Col, 'mm', X1, Y1)
```

Alternatively, the image can be transformed into the world coordinate system by using the operator `image_to_world_plane` (see section 3.3.1 on page 54).

```
image_to_world_plane (Image, ImageMapped, CamParam, PoseForCenteredImage, \
                      WidthMappedImage, HeightMappedImage, \
                      ScaleForCenteredImage, 'bilinear')
```

In the following sections, we will describe the calibration process as well as the transformation between the image and the world coordinates in detail.

3.1 3D Camera Calibration

In the following, the process of 3D camera calibration is described in detail. Note that from HALCON 9.0 on, you can easily calibrate your camera with the help of HDevelop's Calibration Assistant (see the HDevelop User's Guide, section 6.2 on page 164 for more details).

In HALCON, area scan cameras as well as line scan cameras can be calibrated. In both cases, the same operators are used. The differentiation between the two kinds of cameras and between the two distortion models, which are available for area scan cameras, is done based on the number of internal camera parameters as follows:

Area scan camera:
- Division model for lens distortions:
  8 values: *[*f, κ, S_x, S_y, C_x, C_y, *NumColumns, NumRows]*
- Polynomial model for lens distortions:
  12 values: *[*f, K_1, K_2, K_3, P_1, P_2, S_x, S_y, C_x, C_y, *NumColumns, NumRows]*

Line scan camera:
- Division model for lens distortions:
  11 values: *[*f, κ, S_x, S_y, C_x, C_y, *NumColumns, NumRows,* V_x, V_y, V_z*]*

See section 2.2 on page 19 for the description of the underlying camera models.

As you have seen above, in HALCON the calibration is determined simply by using the operator `camera_calibration`. Its input can be grouped into two categories:

1. Corresponding points, given in world coordinates as well as in image coordinates
2. Initial values for the camera parameters.

The first category of input parameters requires the location of a sufficiently large number of 3D points in world coordinates and the correspondence between the world points and their projections in the image.

To define the 3D points in world coordinates, usually objects or marks that are easy to extract, e.g., circles or linear grids, are placed into known locations. If the location of a camera must be known with respect to a given coordinate system, e.g., with respect to the building plan of, say, a factory building, then each mark location must be measured very carefully within this coordinate system. Fortunately, in most cases it is sufficient to know the position of a reference object with respect to the camera to be able to measure the object precisely, since the absolute position of the object in world coordinates is unimportant. Therefore, we propose to use a HALCON calibration plate (figure 19). See section 3.1.7 on page 46 on how to obtain this calibration plate. You can place it almost anywhere in front of the camera to determine the camera parameters.

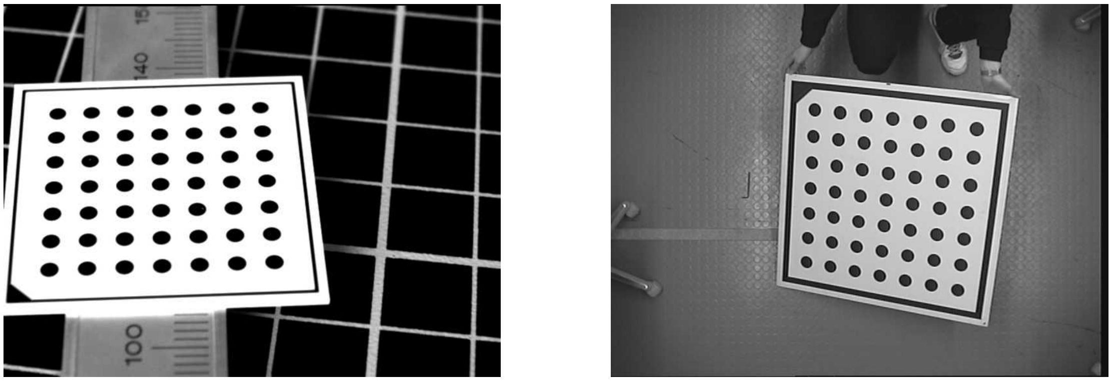


Figure 19: Examples of calibration plates used by HALCON .

The determination of the correspondence of the known world points and their projections in the image is in general a hard problem. The HALCON calibration plate is constructed such that this correspondence can be determined automatically.

Also the second category of input parameters, the starting values, can be determined automatically if the HALCON calibration plate is used.

The results of the operator `camera_calibration` are the internal camera parameters and the pose of the calibration plate in each of the images from which the corresponding points were determined. If the calibration plate was placed directly on the measurement plane its pose can be used to easily derive the external camera parameters, which are the pose of the measurement plane.

Note that the determination of the internal and of the external camera parameters can be separated. For this, the operator camera_calibration must be called twice. First, for the determination of the internal camera parameters only. Then, for the determination of the external camera parameters with the internal camera parameters remaining unchanged. This may be useful in cases where the measurements should be carried out in several planes when using a single camera.

In the following, the calibration process is described in detail, especially the determination of the necessary input values. Additionally, some hints are given on how to obtain precise results.

3.1.1 Camera Calibration Input I: Corresponding Points

The first category of input parameters for the operator camera_calibration comprises corresponding points, i.e., points, for which the world coordinates as well as the image coordinates of their projections into the image are given.

If the HALCON calibration plate is used, the world coordinates of the calibration marks can be read from the calibration plate description file using the operator caltab_points. It returns the coordinates stored in the tuples X, Y, and Z. The length of these tuples depends on the number of calibration marks. Assume we have a calibration plate with m calibration marks. Then, X, Y, and Z are of length m.

```
caltab_points (CaltabName, X, Y, Z)
```

As mentioned above, it is necessary to extract the marks of the calibration plate and to know the correspondence between the marks extracted from the image and the respective 3D points. If the HALCON calibration plate is used, this can be achieved by using the operator find_caltab to find the inner part of the calibration plate and find_marks_and_pose to locate the centers of the circles and to determine the correspondence.

```
for I := 1 to NumImages by 1
  read_image (Image, ImgPath+'calib_'+I$'02d')
  find_caltab (Image, Caltab, CaltabName, SizeGauss, MarkThresh, \
               MinDiamMarks)
  find_marks_and_pose (Image, Caltab, CaltabName, StartCamPar, StartThresh, \
                       DeltaThresh, MinThresh, Alpha, MinContLength, \
                       MaxDiamMarks, RCoord, CCoord, StartPose)
  NStartPose := [NStartPose,StartPose]
  NRow := [NRow,RCoord]
  NCol := [NCol,CCoord]
endfor
```

find_caltab searches for the calibration plate based on the knowledge that it appears bright with dark calibration marks on it. SizeGauss determines the size of the Gauss filter that is used to smooth the input image. A larger value leads to a stronger smoothing, which might be necessary if the image is very noisy. After smoothing the image, a thresholding operator with minimum gray value MarkThresh and maximum gray value 255 is applied with the intention to find the calibration plate. Therefore, MarkThresh should be set to a gray value that is lower than that of the white parts of the calibration plate, but preferably higher than that of any other large bright region in the image. Among the regions resulting from the threshold operation, the most convex region with an almost correct number of holes (corresponding to the dark marks of the calibration plate) is selected. Holes with a diameter smaller than MinDiamMarks are eliminated to reduce the impact of noise. The number of marks is read from the calibration plate description file CalTabDescrFile.

find_marks_and_pose extracts the calibration marks and precisely determines their image coordinates. Therefore, in the input image Image an edge detector is applied to the region CalTabRegion, which can be found by the operator find_caltab. The edge detector can be controlled via the parameter Alpha. Larger values for Alpha lead to a higher sensitivity of the edge detector with respect to small details, but also to less robustness to noise.

In the edge image, closed contours are extracted. For the detection of the contours a threshold operator is applied to the amplitude of the edges. All points with a high amplitude (i.e., borders of marks) are selected. First, the threshold value is set to StartThresh. If the search for the closed contours or the successive pose estimate (see section 3.1.4 on page 35) fails, this threshold value is successively decreased by DeltaThresh down to a minimum value of MinThresh.

The number of closed contours must correspond to the number of calibration marks as described in the calibration plate description file CalTabDescrFile and the contours must have an elliptical shape. Contours shorter than MinContLength are discarded, just as contours enclosing regions with a diameter larger than MaxDiamMarks (e.g., the border of the calibration plate).

The image coordinates of the calibration marks are determined by applying find_marks_and_pose for each image separately. They must be concatenated such that all row coordinates are together in one tuple and all column coordinates are in a second tuple.

The length of these tuples depends on the number of calibration marks and on the number of calibration images. Assume we have a calibration plate with m calibration marks and l calibration images. Then, the tuples containing all the image coordinates of the calibration marks have a length of $m \cdot l$, because they contain the coordinates of the m calibration marks extracted from each of the l images. The order of the values is “image by image”, i.e., the first m values correspond to the coordinates of the m calibration marks extracted from the first image, namely in the order in which they appear in the parameters X, Y, and Z, which are returned by the operator caltab_points. The next m values correspond to the marks extracted from the second image, etc.

Note that the order of all the parameter values must be followed strictly. Therefore, it is very important that each calibration mark is extracted in each image.

3.1.2 Rules for Taking Calibration Images

If you want to achieve accurate results, please follow the rules given in this section:

- Use a clean calibration plate.
- Cover the whole field of view with multiple images, i.e, place the calibration plate in all areas of the field of view at least once.
- Vary the orientations of the calibration plate. This includes rotations around the x- and y-axis of the calibration plate, such that the perspective distortions of the calibration pattern are clearly visible. Without some tilted calibration plates the focal length can not be calculated properly (a tilt angle of approximately 45 degrees is recommended).

- Use at least 10 – 15 images.
- Use an illumination where the background is darker than the calibration plate.
- The bright parts of the calibration plate should have a gray value of at least 100.
- The contrast between the bright and the dark parts of the calibration plate should be more than 100 gray values.
- Use an illumination where the calibration plate is homogeneous.
- The images should not be overexposed (the gray values of the bright parts of the image should be strictly below 255).
- The diameter of a circle should be at least 10 pixels and the size of the calibration plate should be at least a quarter of the image.
- The calibration plate should be completely visible inside the image.
- The images should contain as little noise as possible.
- The images should be sharply focused, i.e., transitions between objects should be clearly delimited.

Note that a good calibration result can be obtained only for a homogeneous distribution of calibration marks within the field of view of the camera. You can imagine the part of the 3D space that corresponds to the field of view as a calibration volume like shown in figure 20. There, two poses of calibration plates and the positions of their calibration marks, when seen from different views, are illustrated. You can see, e.g., in the view from side 1, that large parts are not covered by marks. To get a homogeneous distribution of the marks and thus enable a good calibration result, you have to place the calibration plates in your other images so that the empty parts of the calibration volume are minimized for all views. Be aware that when having very small calibration plates (compared to the field of view), this means that it may be necessary to use significantly more than the recommended 10 – 15 calibration images.

If only one image is used for the calibration process or if the orientations of the calibration plate do not vary over the different calibration images it is not possible to determine both the focal length and the pose of the camera correctly; only the ratio between the focal length and the distance between calibration plate and camera can be determined in this case. Nevertheless, it is possible to measure world coordinates in the plane of the calibration plate but it is not possible to adapt the camera parameters in order to measure in another plane, e.g., the plane onto which the calibration plate was placed.

The accuracy of the resulting world coordinates depends — apart of the measurement accuracy in the image — very much on the number of images used for the calibration process. The more images (with significantly different calibration plate poses) are used, the more accurate results will be achieved.

3.1.3 Which Distortion Model to Use

For area scan cameras, two distortion models can be used: The division model and the polynomial model. The division model uses one parameter to model the radial distortions while the polynomial model uses five parameters to model radial and decentering distortions (see section 2.2.1 on page 20).

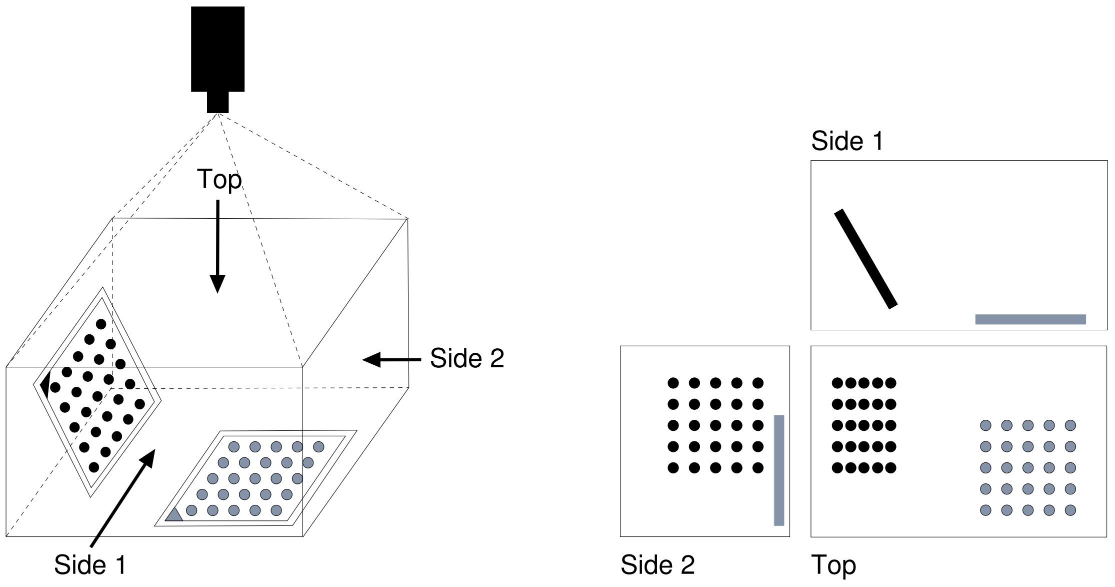


Figure 20: Investigation of the calibration volume: (left) calibration volume with two calibration plate poses and (right) the corresponding distribution of calibration marks when seen from different views. For a good calibration result, the areas without calibration marks (which are especially large in the view from side 1) have to be minimized by a cautious selection of the further calibration plate poses.

The advantages of the division model are that the distortions can be applied faster, especially the inverse distortions, i.e., if world coordinates are projected into the image plane. Furthermore, if only few calibration images are used or if the field of view is not covered sufficiently, the division model typically yields more stable results than the polynomial model. The main advantage of the polynomial model is that it can model the distortions more accurately because it uses higher order terms to model the radial distortions and because it also models the decentering distortions. Note that the polynomial model cannot be inverted analytically. Therefore, the inverse distortions must be calculated iteratively, which is slower than the calculation of the inverse distortions with the (analytically invertible) division model.

Typically, the division model should be used for the calibration. If the accuracy of the calibration is not high enough, the polynomial model can be used. But note that the calibration sequence used for the polynomial model must provide a complete coverage of the area in which measurements will later be performed. The distortions may be modeled inaccurately outside of the area that was covered by the calibration plate. This holds for the image border as well as for areas inside the field of view that were not covered by the calibration plate.

3.1.4 Camera Calibration Input II: Initial Values

The second category of input parameters of the operator `camera_calibration` comprises initial values for the camera parameters.

As the camera calibration is a difficult non-linear optimization problem, good initial values are required for the parameters.

Internal camera parameters

The initial values for the internal camera parameters can be determined from the specifications of the CCD sensor and the lens. They must be given as a tuple of the form

- *[f, κ, S_x, S_y, C_x, C_y, NumColumns, NumRows]*
  for area scan cameras using the division model,
- *[f, K_1, K_2, K_3, P_1, P_2, S_x, S_y, C_x, C_y, NumColumns, NumRows]*
  for area scan cameras using the polynomial model, and
- *[f, κ, S_x, S_y, C_x, C_y, NumColumns, NumRows, V_x, V_y, V_z]*
  for line scan cameras using the division model, respectively.

Note that in addition to the internal camera parameters, the width (*NumColumns*) and height (*NumRows*) of the image must be given. See section 2.2 on page 19 for a description of the internal camera parameters.

```
StartCamPar := [0.016,0,0.0000074,0.0000074,326,247,652,494]
```

In the following, some hints for the determination of the initial values for the internal camera parameters of an **area scan camera** are given:

Focus f: The initial value is the nominal focal length of the used lens, e.g., 0.016 m.

κ: Use 0.0 as initial value (omitted if the polynomial model is used).

Or:

K_1, K_2, K_3, P_1, P_2: Use the initial value 0.0 for each of the five coefficients (omitted if the division model is used).

S_x: For pinhole cameras, the initial value for the horizontal distance between two neighboring CCD cells depends on the dimension of the used CCD chip of the camera (see technical specifications of the camera). Generally, common CCD chips are either 1/3”-Chips (e.g., SONY XC-73, SONY XC-777), 1/2”-Chips (e.g., SONY XC-999, Panasonic WV-CD50), or 2/3”-Chips (e.g., SONY DXC-151, SONY XC-77). Notice: The value of S_x increases if the image is sub-sampled! Appropriate initial values are:

	Full image (640*480)	Subsampling (320*240)
1/3"-Chip	0.0000055 m	0.0000110 m
1/2"-Chip	0.0000086 m	0.0000172 m
2/3"-Chip	0.0000110 m	0.0000220 m

The value for S_x is calibrated, since the video signal of a CCD camera normally is not sampled pixel-synchronously.

S_y: Since most off-the-shelf cameras have square pixels, the same values for S_y are valid as for S_x. In contrast to S_x the value for S_y will *not* be calibrated for pinhole cameras because the video signal of a CCD camera normally is sampled line-synchronously. Thus, the initial value is equal to the final value. Appropriate initial values are:

	Full image (640*480)	Subsampling (320*240)
1/3"-Chip	0.0000055 m	0.0000110 m
1/2"-Chip	0.0000086 m	0.0000172 m
2/3"-Chip	0.0000110 m	0.0000220 m

C_x and C_y: Initial values for the coordinates of the principal point are the coordinates of the image center, i.e., the half image width and the half image height. Notice: The values of C_x and C_y decrease if the image is subsampled! Appropriate initial values are, for example:

	Full image (640*480)	Subsampling (320*240)
C_x	320.0	160.0
C_y	240.0	120.0

ImageWidth and ImageHeight: These two parameters are set by the the used frame grabber and therefore are not calibrated. Appropriate initial values are, for example:

	Full image (640*480)	Subsampling (320*240)
ImageWidth	640	320
ImageHeight	480	240

In the following, some hints for the determination of the initial values for the internal camera parameters of a **line scan camera** are given:

Focus f: The initial value is the nominal focal length of the the used lens, e.g., 0.035 m.

κ: Use 0.0 as initial value.

S_x: The initial value for the horizontal distance between two neighboring sensor elements can be taken from the technical specifications of the camera. Typical initial values are $7{\cdot}10^{-6}$ m, $10{\cdot}10^{-6}$ m, and $14{\cdot}10^{-6}$ m. Notice: The value of S_x increases if the image is subsampled!

S_y: The initial value for the size of a cell in the direction perpendicular to the sensor line can also be taken from the technical specifications of the camera. Typical initial values are $7{\cdot}10^{-6}$ m, $10{\cdot}10^{-6}$ m, and $14{\cdot}10^{-6}$ m. Notice: The value of S_y increases if the image is subsampled! In contrast to S_x, the value for S_y will NOT be calibrated for line scan cameras because it cannot be determined separately from the parameter C_y.

C_x: The initial value for the x-coordinate of the principal point is half the image width. Notice: The values of C_x decreases if the image is subsampled! Appropriate initial values are:

```
Image width:  1024  2048  4096  8192
Cx:            512  1024  2048  4096
```

C_y: Normally, the initial value for the y-coordinate of the principal point can be set to 0.

ImageWidth and ImageHeight: These two parameters are determined by the used frame grabber and therefore are not calibrated.

V_x, V_y, V_z: The initial values for the x-, y-, and z-component of the motion vector depend on the image acquisition setup. Assuming a fixed camera that looks perpendicularly onto a conveyor belt, such that the y-axis of the camera coordinate system is anti-parallel to the moving direction of the conveyor belt (see figure 21 on page 39), the initial values are $V_x = V_z = 0$. The initial value for V_y can then be determined, e.g., from a line scan image of an object with known size (e.g., calibration plate or ruler):

$$V_y = L[m]/L[row]$$

with:

$$L[m] = \text{Length of the object in object coordinates [meter]}$$
$$L[row] = \text{Length of the object in image coordinates [rows]}$$

If, compared to the above setup, the camera is rotated 30 degrees around its optical axis, i.e., around the z-axis of the camera coordinate system (figure 22 on page 40), the above determined initial values must be changed as follows:

$$\begin{aligned} V_{zx} &= sin(30°) \cdot V_y \\ V_{zy} &= cos(30°) \cdot V_y \\ V_{zz} &= V_z = 0 \end{aligned}$$

If, compared to the first setup, the camera is rotated -20 degrees around the x-axis of the camera coordinate system (figure 23 on page 41), the following initial values result:

$$\begin{aligned} V_{xx} &= V_x = 0 \\ V_{xy} &= cos(-20°) \cdot V_y \\ V_{xz} &= sin(-20°) \cdot V_y \end{aligned}$$

The quality of the initial values for V_x, V_y, and V_z are crucial for the success of the whole calibration. If they are not accurate enough, the calibration may fail.

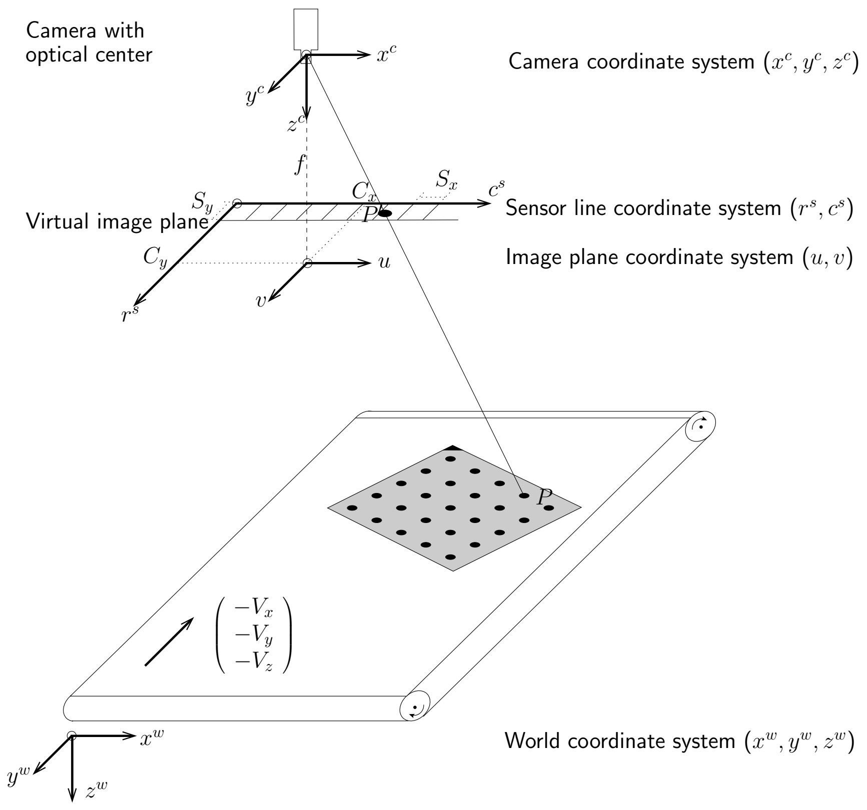

Camera with optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
f
S_y
C_x
S_x
c^s
P'
Virtual image plane
Sensor line coordinate system (r^s, c^s)
C_y
u
v
r^s
Image plane coordinate system (u, v)
P
$\begin{pmatrix} -V_x \\ -V_y \\ -V_z \end{pmatrix}$
x^w
y^w
z^w
World coordinate system (x^w, y^w, z^w)

Figure 21: Line scan camera looking perpendicularly onto a conveyor belt.

External camera parameters

The initial values for the external parameters are in general harder to obtain. For the HALCON calibration plate, good starting values are computed by the operator find_marks_and_pose based on the geometry and size of the projected calibration marks. Again, these values are determined for each calibration image separately. They must be concatenated into one tuple. Assume we have l calibration images. Then, the length of this tuple is $l \cdot 7$ (l times the 6 external camera parameters together with the code for the pose type). The first 7 values correspond to the camera pose of the first image, the next 7 values to the pose of the second image, etc.

If you use another calibration object the operator find_marks_and_pose cannot be used. In this case, you must determine the initial values for the external parameters yourself.

3.1.5 Determining the Internal Camera Parameters

Given some initial values for the camera parameters, the known 3D locations of the calibration marks can be projected into the CCS. Then, the camera parameters can be determined such that the distance of the projections of the calibration marks and the mark locations extracted from the imagery is minimized.

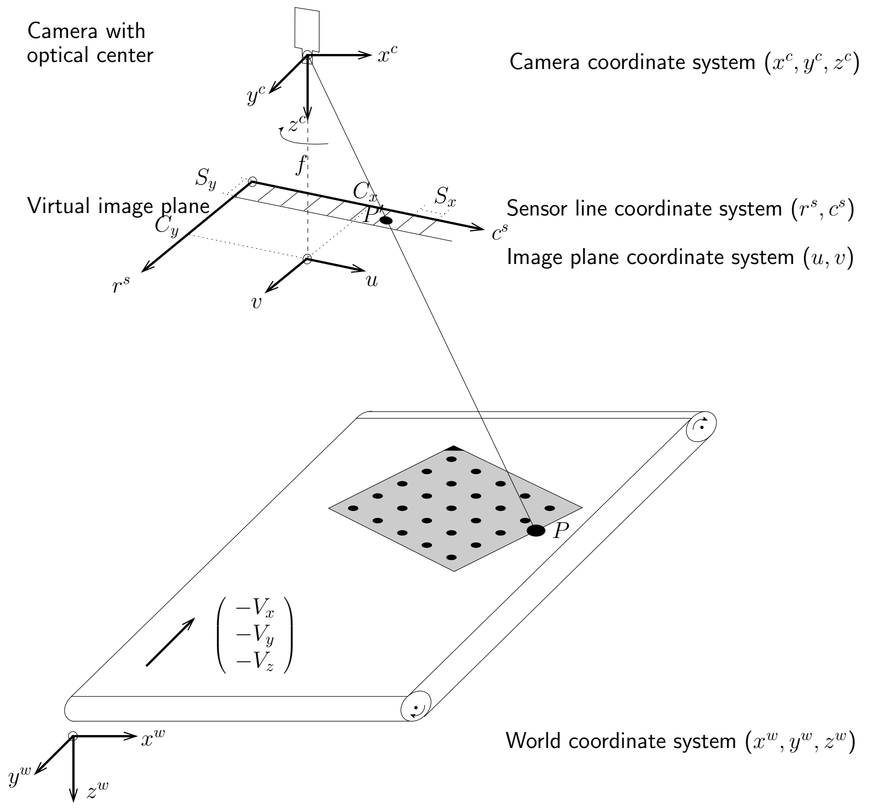

Camera with optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
f
S_y
C_x
S_x
Virtual image plane
C_y
P'
Sensor line coordinate system (r^s, c^s)
c^s
Image plane coordinate system (u, v)
r^s
u
v
P
$\begin{pmatrix} -V_x \\ -V_y \\ -V_z \end{pmatrix}$
x^w
y^w
z^w
World coordinate system (x^w, y^w, z^w)

Figure 22: Line scan camera rotated around the optical axis.

This minimization process will return fairly accurate values for the camera parameters. However, to obtain the camera parameters with the highest accuracy, it is essential that more than one image of the calibration plate is taken, where the plate is placed and rotated differently in each image so as to use all degrees of freedom of the external orientation. A typical sequence of images used for calibration is displayed in figure 24.

If l images of the calibration plate are taken, the parameters to optimize are the internal parameters and l sets of the external parameters. Now, the aim of the optimization is to determine all these parameters such that in each of the l images the distance of the extracted mark locations and the projections of the respective 3D locations is minimal. In HALCON, this is exactly what the operator camera_calibration does.

```
camera_calibration (X, Y, Z, NRow, NCol, StartCamPar, NStartPose, 'all', \
                    CamParam, NFinalPose, Errors)
```

The operator camera_calibration needs the coordinates of the corresponding points in the world coordinate system and the pixel coordinate system as well as some initial values for the camera parameters. See section 3.1.1 on page 32 and section 3.1.4 on page 35 for a description on how to obtain these input

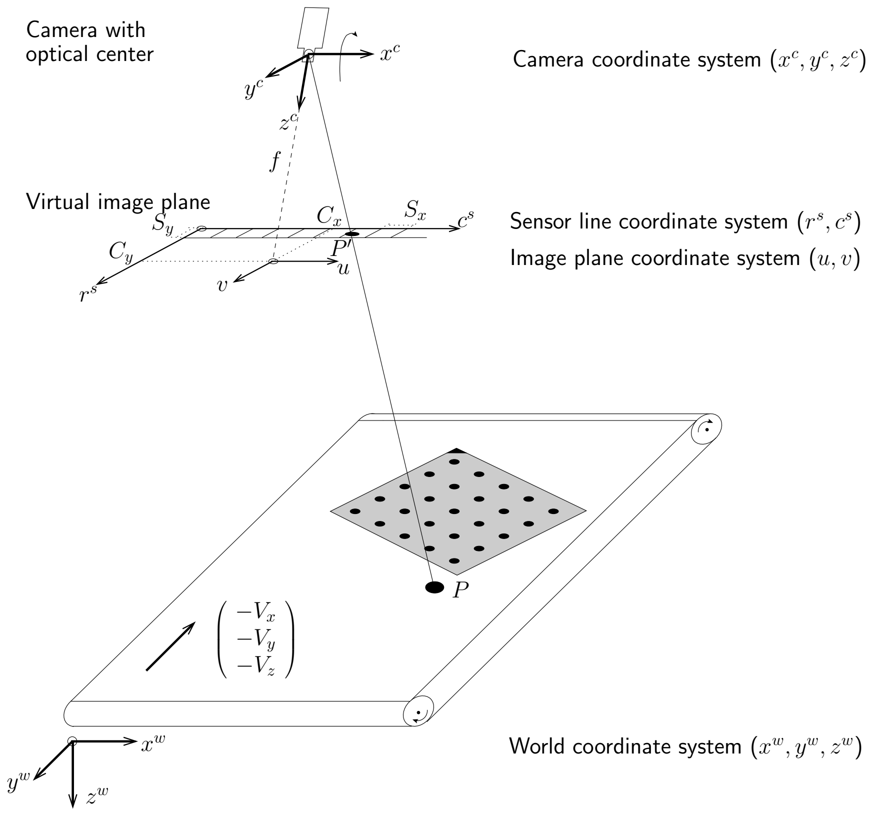

Camera with optical center
x^c
y^c
z^c
f
Camera coordinate system (x^c, y^c, z^c)
Virtual image plane
S_y
C_x
S_x
c^s
C_y
P'
u
v
r^s
Sensor line coordinate system (r^s, c^s)
Image plane coordinate system (u, v)
P
$\begin{pmatrix} -V_x \\ -V_y \\ -V_z \end{pmatrix}$
x^w
y^w
z^w
World coordinate system (x^w, y^w, z^w)

Figure 23: Line scan camera rotated around the x-axis.

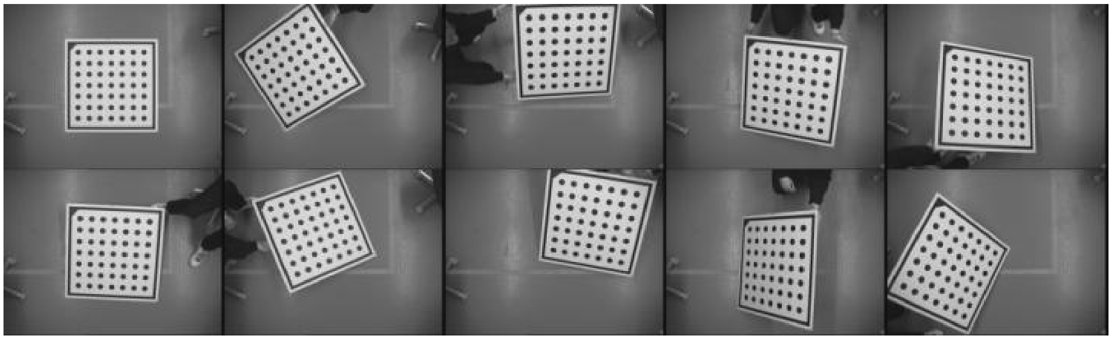


Figure 24: A sequence of calibration images..

values.

If the parameter EstimateParams is set to *'all'*, the internal parameters for the used camera are determined as well as the external parameters for each image. If the parameter is set to *'pose'*, only the external parameters are determined. To determine just selected parameters, EstimateParams can be set to a list that contains the respective parameter names (*['alpha','beta','gamma','transx','transy','transz','focus','kappa', 'cx','cy','sx','sy']*). It is also possible to prevent the determination of certain parameters by adding their names with the prefix ~ to the list, e.g., if EstimateParams is set to *['all', '~focus']*, all parameters but the focal length are determined.

Note that if the polynomial model is used to model the lens distortions, it is not possible to specify the values K_1, K_2, K_3, P_1, and P_2, individually. They can only be specified in the following groups:

'poly':	K_1, K_2, K_3, P_1, and P_2
'poly_rad_2':	K_1
'poly_rad_4':	K_1 and K_2
'poly_rad_6':	K_1, K_2, and K_3
'poly_tan_2':	P_1 and P_2

The computed average error (Errors) reflects the accuracy of the calibration. The error value (root mean square error of the position) is measured in pixels.

Up to now, the external parameters are not necessarily related to the measurement plane.

3.1.6 Determining the External Camera Parameters

The external camera parameters describe the relation between the measurement plane and the camera, i.e., only if the external parameters are known it is possible to transform coordinates from the CCS into the coordinate system of the measurement plane and vice versa. In HALCON, the measurement plane is defined as the plane $z = 0$ of the world coordinate system (WCS). The external camera parameters can be determined in different ways:

1. Use the pose of the calibration plate present in one of the calibration images. In this case, it is not necessary to call the operator camera_calibration a second time.

2. Obtain an additional calibration image where the calibration plate has been placed directly on the measurement plane. Apply find_caltab and find_marks_and_pose to extract the calibration marks. Then, use the operator camera_calibration to determine only the external camera parameters.

3. Determine the correspondences between 3D world points and their projections in the image by yourself. Again, use the operator camera_calibration to determine the external camera parameters.

If it is only necessary to measure accurately the dimensions of an object, regardless of the absolute position of the object in a given coordinate system, one of the first two cases can be used.

The latter two cases have the advantage that the external camera parameters can be determined independently from the internal camera parameters. This is more flexible and might be useful if the measurements should be done in several planes from one camera only or if it is not possible to calibrate the camera in situ.

In the following, these different cases are described in more detail.

The **first** case is the easiest way of determining the external parameters. The calibration plate must be placed directly on the measurement plane, e.g., the assembly line, in one of the (many) images used for the determination of the internal parameters.

Since the pose of the calibration plate is determined by the operator `camera_calibration`, you can just pick the respective pose from the output parameter `NFinalPose`. In this way, internal and external parameters are determined in one single calibration step. The following code fragment from the program `examples\solution_guide\3d_machine_vision\camera_calibration_multi_image.dev` is an example for this easy way of determining the external parameters. Here, the pose of the calibration plate in the eleventh calibration image is determined. Please note that each pose consists of seven values.

```
NumImage := 11
Pose := NFinalPose[(NumImage-1)*7:(NumImage-1)*7+6]
```

The resulting pose would be the true pose of the measurement plane if the calibration plate were infinitely thin. Because real calibration plates have a thickness $d > 0$, the pose of the calibration plate is shifted by an amount $-d$ perpendicular to the measurement plane, i.e., along the z axis of the WCS. To correct this, we need to shift the pose by d along the z axis of the WCS. To perform this shift, the operator `set_origin_pose` can be used. The corresponding HALCON code is:

```
set_origin_pose (Pose, 0, 0, 0.00075, Pose)
```

In general, the calibration plate can be oriented arbitrarily within the WCS (see figure 25). In this case, to derive the pose of the measurement plane from the pose of the calibration plate a rigid transformation is necessary. In the following example, the pose of the calibration plate is adapted by a translation along the y axis followed by a rotation around the x axis.

```
pose_to_hom_mat3d (FinalPose, HomMat3D)
hom_mat3d_translate_local (HomMat3D, 0, 3.2, 0, HomMat3DTranslate)
hom_mat3d_rotate_local (HomMat3DTranslate, rad(-14), 'x', HomMat3DAdapted)
hom_mat3d_to_pose (HomMat3DAdapted, PoseAdapted)
```

If the advantages of using the HALCON calibration plate should be combined with the flexibility given by the separation of the internal and external camera parameters the **second** method for the determination of the external camera parameters can be used.

At first, only the internal parameters are determined as described in section 3.1.5 on page 39. This can be done, e.g., prior to the deployment of the camera.

This is shown in the example program `examples\solution_guide\3d_machine_vision\camera_calibration_interior.dev`, which is similar to the example program given above, except that no image is used in which the calibration plate is positioned on the object and that the calculated internal camera parameters are written to a file.

Note that we do not use any of the images to derive the pose of the measurement plane.

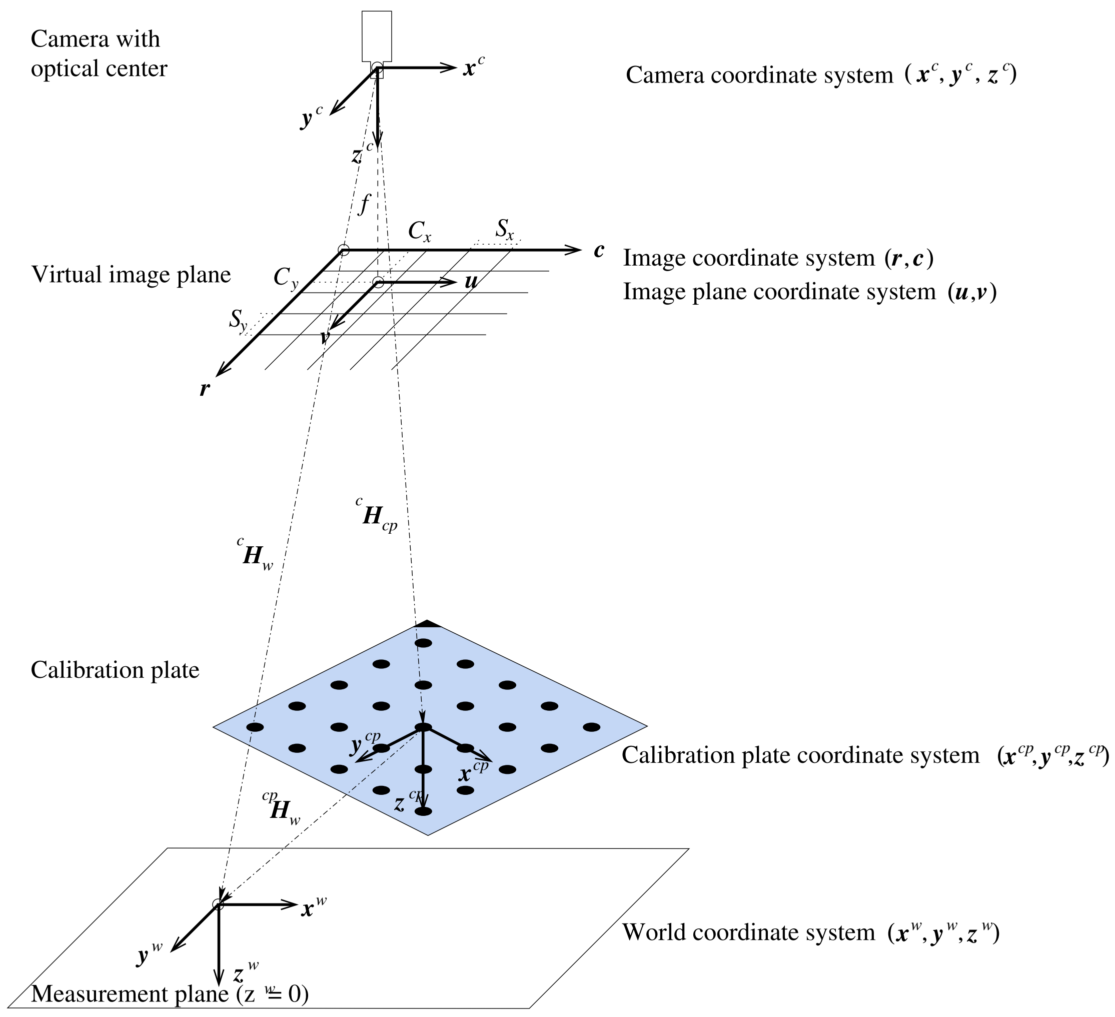

Camera with optical center
Camera coordinate system (x^c, y^c, z^c)
x^c
y^c
z^c
f
C_x
S_x
C_y
S_y
Virtual image plane
Image coordinate system (r, c)
Image plane coordinate system (u, v)
c
u
v
r
${}^cH_{cp}$
${}^cH_w$
Calibration plate
Calibration plate coordinate system (x^{cp}, y^{cp}, z^{cp})
y^{cp}
x^{cp}
z^{cp}
${}^{cp}H_w$
x^w
y^w
z^w
World coordinate system (x^w, y^w, z^w)
Measurement plane $(z^w = 0)$

Figure 25: Relation between calibration plate and measurement plane.

```
for I := 1 to NumImages by 1
  read_image (Image, ImgPath+'calib_'+I$'02d')
  find_caltab (Image, Caltab, CaltabName, SizeGauss, MarkThresh, \
               MinDiamMarks)
  find_marks_and_pose (Image, Caltab, CaltabName, StartCamPar, StartThresh, \
                       DeltaThresh, MinThresh, Alpha, MinContLength, \
                       MaxDiamMarks, RCoord, CCoord, StartPose)
  NStartPose := [NStartPose,StartPose]
  NRow := [NRow,RCoord]
  NCol := [NCol,CCoord]
endfor
camera_calibration (X, Y, Z, NRow, NCol, StartCamPar, NStartPose, 'all', \
                    CamParam, NFinalPose, Errors)
```

Then, the internal camera parameters can be written to a file:

```
write_cam_par (CamParam, 'camera_parameters.dat')
```

Then, after installing the camera at its usage site, the external parameters can be determined. The only thing to be done is to take an image where the calibration plate is placed directly on the measurement plane, from which the external parameters can be determined.

Again the operators `find_caltab` and `find_marks_and_pose` can be used to extract the calibration marks. Then, the operator `camera_calibration` with the parameter `EstimateParams` set to *'pose'* determines just the pose of the calibration plate and leaves the internal camera parameter unchanged. Alternatively, `EstimateParams` can be set to the tuple *['alpha', 'beta', 'gamma', 'transx', 'transy', 'transz']*, which also means that the six external parameters will be estimated. Again, the pose must be corrected for the thickness of the calibration plate as described above.

The program `examples\solution_guide\3d_machine_vision\camera_calibration_exterior.dev` shows how to determine the external camera parameters from a calibration plate that is positioned on the object's surface.

First, the internal camera parameters, the image, where the calibration plate was placed directly on the measurement plane, and the world coordinates of the calibration marks are read from file:

```
read_cam_par ('camera_parameters.dat', CamParam)
read_image (Image, ImgPath+'calib_11')
caltab_points (CaltabName, X, Y, Z)
```

Then, the calibration marks are extracted.:

```
find_caltab (Image, Caltab, CaltabName, SizeGauss, MarkThresh, MinDiamMarks)
find_marks_and_pose (Image, Caltab, CaltabName, CamParam, StartThresh, \
                     DeltaThresh, MinThresh, Alpha, MinContLength, \
                     MaxDiamMarks, RCoord, CCoord, \
                     InitialPoseForCalibrationPlate)
```

Now, the actual calibration can be carried out:

```
camera_calibration (X, Y, Z, RCoord, CCoord, CamParam, \
                    InitialPoseForCalibrationPlate, 'pose', \
                    CamParamUnchanged, FinalPoseFromCalibrationPlate, \
                    Errors)
```

Finally, to take the thickness of the calibration plate into account, the z value of the origin given by the camera pose must be translated by the thickness of the calibration plate:

```
set_origin_pose (FinalPoseFromCalibrationPlate, 0, 0, 0.00075, \
                 FinalPoseFromCalibrationPlate)
```

Note that it is very important to fix the focus of your camera if you want to separate the calibration process into two steps as described in this section, because changing the focus is equivalent to changing the focal length, which is part of the internal parameters.

If it is necessary to perform the measurements within a given world coordinate system, the **third** case for the determination of the external camera parameters can be used. Here, you need to know the 3D world coordinates of at least three points that do not lie on a straight line. Then, you must determine the corresponding image coordinates of the projections of these points. Now, the operator `camera_calibration` with the parameter `EstimateParams` set to *'pose'* can be used for the determination of the external camera parameters.

Note that in this case, no calibration plate needs to be placed on the measurement plane. This means also that the operator `find_marks_and_pose` cannot be used to extract the calibration marks. Therefore, you must generate the input parameter tuples `NX`, `NY`, and `NZ` as well as `NRow` and `NCol` yourself. Also the initial values for the pose must be set appropriately because in this case they are not determined automatically since the operator `find_marks_and_pose` is not used.

An example for this possibility of determining the external parameters is given in the following program. First, the world coordinates of three points are set:

```
X := [0,50,100]
Y := [5,0,5]
Z := [0,0,0]
```

Then, the image coordinates of the projections of these points in the image are determined. In this example, they are simply set to some approximate values. In reality, they should be determined with subpixel accuracy since they define the external camera parameters:

```
RCoord := [414,227,85]
CCoord := [119,318,550]
```

Now, the starting value for the pose must be set appropriately:

```
create_pose (-50, 25, 400, 0, 0, -30, 'Rp+T', 'gba', 'point', InitialPose)
```

Finally, the actual determination of the external camera parameters can be carried out:

```
camera_calibration (X, Y, Z, RCoord, CCoord, CamParam, InitialPose, 'pose', \
                    CamParamUnchanged, FinalPose, Errors)
```

Also in this case, it is very important to fix the focus of your camera because changing the focus is equivalent to changing the focal length, which is part of the internal parameters.

3.1.7 How to Obtain a Suitable Calibration Plate

The simplest method to determine the camera parameters of a CCD camera is to use the HALCON calibration plate. In this case, the whole process of finding the calibration plate, extracting the calibration marks, and determining the correspondences between the extracted calibration marks and the respective 3D world coordinates can be carried out automatically. Even more important, these calibration plates are highly accurate, up to $\pm$ 150 nm (nanometers), which is a prerequisite for high accuracy applications.

Therefore, we recommend to obtain such a calibration plate from the local distributor from which you purchased HALCON.

The calibration plates are available in different materials (ceramics for front light and glass for back light applications) and sizes (e.g., $0.65 \times 0.65\,\text{mm}^2$, $10 \times 10\,\text{mm}^2$, $200 \times 200\,\text{mm}^2$). Thus, you can choose the one that is optimal for your application. A calibration plate should cover at very least the ninth part of the image area. To improve the calibration quality, we recommend, however, a plate that covers at a least quarter of the image area. Detailed information about the available materials, sizes, and the accuracy can be obtained from your distributor.

Each calibration plate comes with a description file. Place this file in the subdirectory `calib` of the folder where you installed HALCON, then you can use its file name directly in the operator `caltab_points` (see section 3.1.1 on page 32).

For test purposes, you can create a calibration plate yourself with the operator `gen_caltab`. Print the resulting PostScript file and mount it on a planar and rigid surface, e.g., an aluminum plate or a solid cardboard. If you do not mount the printout on a planar and rigid surface, you will not get meaningful results by HALCON's camera calibration as the operator `gen_caltab` assumes that the calibration marks lie within a plane. Such self-made calibration plates should only be used for test purposes as you will not achieve the high accuracy that can be obtained with an original HALCON calibration plate. Note that the printing process is typically not accurate enough to create calibration plates smaller than 3 cm.

3.1.8 Using Your Own Calibration Object

With HALCON, you are not restricted to using a planar calibration object like the HALCON calibration plate. The operator `camera_calibration` is designed such that the input tuples `NX`, `NY`, and `NZ` for the world coordinates of the calibration marks and `NRow` and `NCol` for the image coordinates of the locations of the calibration marks within the images can contain any 3D/2D correspondences (compare section 3.1.5 on page 39).

Thus, it is not important how the required 3D model marks and the corresponding extracted 2D marks are determined. You can use a 3D calibration object or even arbitrary characteristic points (natural landmarks). The only requirement is that the 3D world position of the model points is known with high accuracy.

However, if you use your own calibration object, you cannot use the operators `find_caltab` and `find_marks_and_pose` anymore. Instead, you must determine the 2D locations of the model points and the correspondence to the respective 3D points as well as the initial value for the poses yourself.

3.1.9 Special information for the calibration of line scan cameras

In general, the procedure for the calibration of line scan cameras is identical to the one for the calibration of area scan cameras.

However, line scan imaging suffers from a high degree of parameter correlation. For example, any small rotation of the linear array around the x-axis of the camera coordinate system can be compensated by changing the y-component of the translation vector of the respective pose. Even the focal length is correlated with the scale factor S_x and with the z-component of the translation vector of the pose, i.e., with the distance of the object from the camera.

The consequences of these correlations for the calibration of line scan cameras are that some parameters cannot be determined with high absolute accuracy. Nevertheless, the set of parameters is determined consistently, what means that the world coordinates can be measured with high accuracy.

Another consequence of the parameter correlations is that the calibration may fail in some cases where the start values for the internal camera parameters are not accurate enough. If this happens, try the following approach: In many cases, the start values for the motion vector are the most difficult to set. To achieve better start values for the parameters V_x, V_y, and V_z, reduce the number of parameters to be estimated such that the camera calibration succeeds. Try first to estimate the parameters V_x, V_y, V_z, α, β, γ, t_x, t_y, and t_z by setting `EstimateParams` to *['vx', 'vy', 'vz', 'alpha', 'beta', 'gamma', 'transx', 'transy', 'transz']* and if this does not work, try to set `EstimateParams` to *['vx', 'vy', 'vz', 'transx', 'transy', 'transz']*. Then, determine the whole set of parameters using the above determined values for V_x, V_y, and V_z as start values.

If this still does not work, repeat the determination of the start poses with the operator `find_marks_and_pose` using the above determined values for the internal camera parameters as start values. Then retry to calibrate the camera with the operator `camera_calibration`.

If none of the above works, try to determine better start values directly from the camera setup. If possible, change the setup such that it is easier to determine appropriate start values, e.g., mount the camera such that it looks approximately perpendicularly onto the conveyor belt (see figure 21 on page 39).

3.2 Transforming Image into World Coordinates and Vice Versa

In this section, you learn how to obtain world coordinates from images based on the calibration data. On the one hand, it is possible to process the images as usual and then to transform the extraction results into the world coordinate system. In many cases, this will be the most efficient way of obtaining world coordinates. On the other hand, some applications may require that the segmentation itself must be carried out in images that are already transformed into the world coordinate system (see section 3.3 on page 54).

In general, the segmentation process reduces the amount of data that needs to be processed. Therefore, rectifying the segmentation results is faster than rectifying the underlying image. What is more, it is often better to perform the segmentation process directly on the original images because smoothing or aliasing effects may occur in the rectified image, which could disturb the segmentation and may lead to inaccurate results. These arguments suggest to rectify the segmentation results instead of the images.

In the following, first some general remarks on the underlying principle of the transformation of image coordinates into world coordinates are given. Then, it is described how to transform points, contours, and regions into the world coordinate system. Finally, we show that it is possible to transform world coordinates into image coordinates as well, e.g., in order to visualize information given in the world coordinate system.

3.2.1 The Main Principle

Given the image coordinates of one point, the goal is to determine the world coordinates of the corresponding point in the measurement plane. For this, the line of sight, i.e., a straight line from the optical

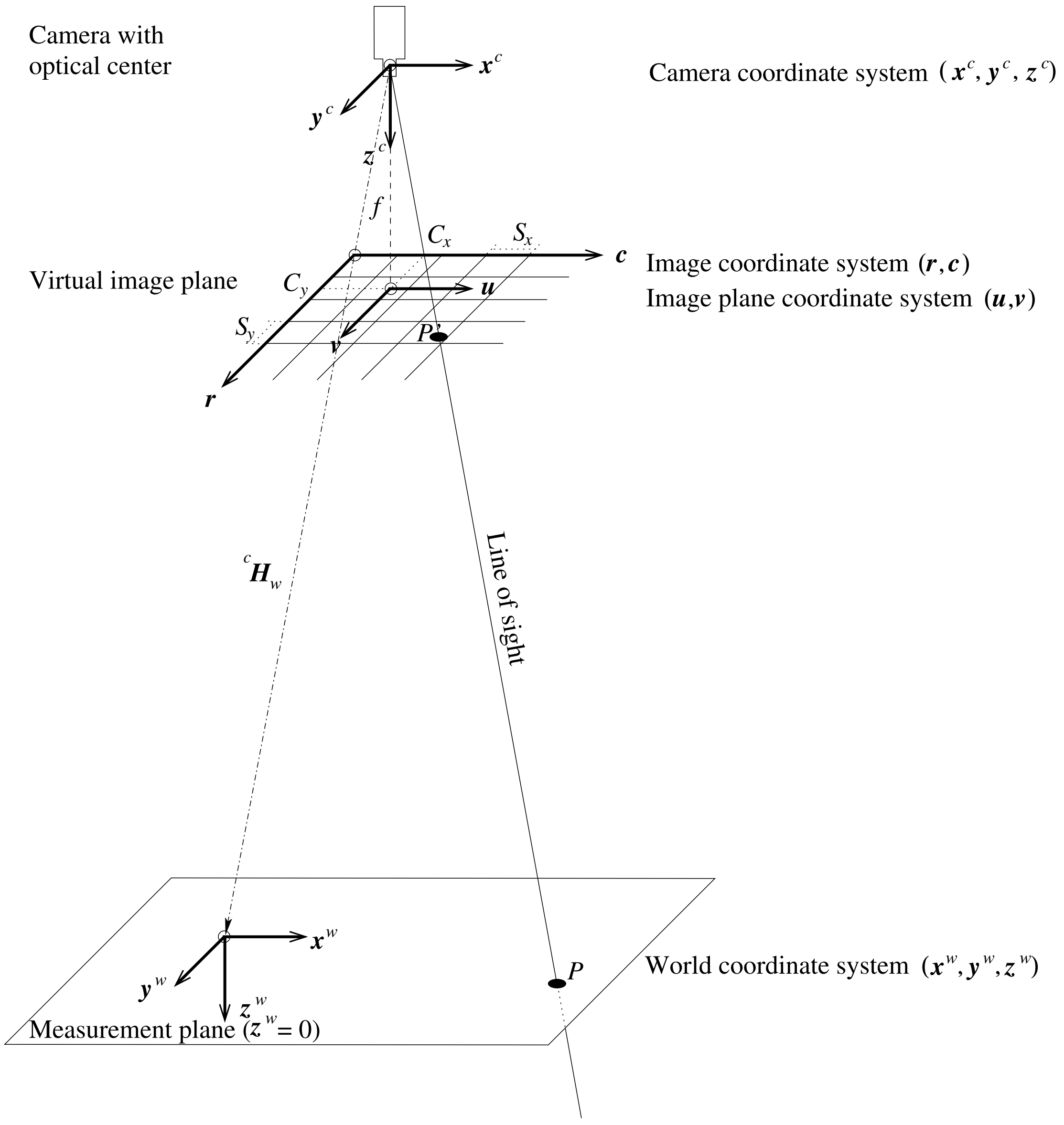

Camera with
optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
f
C_x
S_x
c
Image coordinate system (r, c)
Image plane coordinate system (u, v)
Virtual image plane
C_y
u
S_y
v
P'
r
${}^cH_w$
Line of sight
x^w
y^w
z^w
P
World coordinate system (x^w, y^w, z^w)
Measurement plane $(z^w = 0)$

Figure 26: Intersecting the line of sight with the measurement plane..

center of the camera through the given point in the image plane, must be intersected with the measurement plane (see figure 26).

The calibration data is necessary to transform the image coordinates into camera coordinates and finally into world coordinates.

All these calculations are performed by the operators of the family `..._to_world_plane`.

Again, please remember that in HALCON the measurement plane is defined as the plane $z = 0$ with respect to the world coordinate system. This means that all points returned by the operators of the family `..._to_world_plane` have a z-coordinated equal to zero, i.e., they lie in the plane $z = 0$ of the world coordinate system.

3.2.2 World Coordinates for Points

The world coordinates of an image point (r, c) can be determined using the operator `image_points_to_world_plane`. In the following code example, the row and column coordinates of pitch lines are transformed into world coordinates.

```
image_points_to_world_plane (CamParam, FinalPose, RowPitchLine, \
                              ColPitchLine, 1, X1, Y1)
```

As input, the operator requires the internal and external camera parameters as well as the row and column coordinates of the point(s) to be transformed.

Additionally, the unit in which the resulting world coordinates are to be given is specified by the parameter `Scale` (see also the description of the operator `image_to_world_plane` in section 3.3.1 on page 54). This parameter is the ratio between the unit in which the resulting world coordinates are to be given and the unit in which the world coordinates of the calibration target are given (equation 30).

$$\texttt{Scale} = \frac{\text{unit of resulting world coordinates}}{\text{unit of world coordinates of calibration target}} \tag{30}$$

In many cases the coordinates of the calibration target are given in meters. In this case, it is possible to set the unit of the resulting coordinates directly by setting the parameter `Scale` to *'m'* (corresponding to the value *1.0*, which could be set alternatively for the parameter `Scale`), *'cm'* (*0.01*), *'mm'* *(0.001)*, *'microns'* (*1e-6*), or *'µm'* (again, *1e-6*). Then, if the parameter `Scale` is set to, e.g., *'m'*, the resulting coordinates are given in meters. If, e.g., the coordinates of the calibration target are given in µm and the resulting coordinates have to be given in millimeters, the parameter `Scale` must be set to:

$$\texttt{Scale} = \frac{mm}{\mu m} = \frac{1 \cdot 10^{-3}\, m}{1 \cdot 10^{-6}\, m} = 1000 \tag{31}$$

3.2.3 World Coordinates for Contours

If you want to convert an XLD object containing pixel coordinates into world coordinates, the operator `contour_to_world_plane_xld` can be used. Its parameters are similar to those of the operator `image_points_to_world_plane`, as can be seen from the following example program:

```
lines_gauss (ImageReduced, Lines, 1, 3, 8, 'dark', 'true', 'true', 'true')
contour_to_world_plane_xld (Lines, ContoursTrans, CamParam, PoseAdapted, 1)
```

3.2.4 World Coordinates for Regions

In HALCON, regions cannot be transformed directly into the world coordinate system. Instead, you must first convert them into XLD contours using the operator `gen_contour_region_xld`, then apply the transformation to these XLD contours as described in the previous section.

If the regions have holes and if these holes would influence your further calculations, set the parameter `Mode` of the operator `gen_contour_region_xld` to *'border_holes'*. Then, in addition to the outer border of the input region the operator `gen_contour_region_xld` returns the contours of all holes.

3.2.5 Transforming World Coordinates into Image Coordinates

In this section, the transformation between image coordinates and world coordinates is performed in the opposite direction, i.e., from world coordinates to image coordinates. This is useful if you want to visualize information given in world coordinates or it may be helpful for the definition of meaningful regions of interest (ROI).

First, the world coordinates must be transformed into the camera coordinate system. For this, the homogeneous transformation matrix ${}^{CCS}\mathbf{H}_{WCS}$ is needed, which can easily be derived from the pose of the measurement plane with respect to the camera by the operator `pose_to_hom_mat3d`. The transformation itself can be carried out using the operator `affine_trans_point_3d`. Then, the 3D coordinates, now given in the camera coordinate system, can be projected into the image plane with the operator `project_3d_point`. An example program is given in the following:

First, the world coordinates of four points defining a rectangle in the WCS are defined.

```
ROI_X_WCS := [-2,-2,112,112]
ROI_Y_WCS := [0,0.5,0.5,0]
ROI_Z_WCS := [0,0,0,0]
```

Then, the transformation matrix ${}^{CCS}\mathbf{H}_{WCS}$ is derived from the respective pose.

```
pose_to_hom_mat3d (FinalPose, CCS_HomMat_WCS)
```

Finally, the world points are transformed into the image coordinate system.

```
affine_trans_point_3d (CCS_HomMat_WCS, ROI_X_WCS, ROI_Y_WCS, ROI_Z_WCS, \
                       CCS_RectangleX, CCS_RectangleY, CCS_RectangleZ)
project_3d_point (CCS_RectangleX, CCS_RectangleY, CCS_RectangleZ, \
                  CamParamUnchanged, RectangleRow, RectangleCol)
```

3.2.6 Compensate for Lens Distortions Only

All operators discussed above automatically compensate for lens distortions. In some cases, you might want to compensate for lens distortions only without transforming results or images into world coordinates.

Note that in the following, only the compensation for radial distortions using the division model is described. The compensation for radial and decentering distortions using the polynomial model is done analogously by replacing $\kappa = 0$ with $K_1 = K_2 = K_3 = P_1 = P_2 = 0$.

The procedure is to specify the original internal camera parameters and those of a virtual camera that does not produce lens distortions, i.e., with $\kappa = 0$.

The easiest way to obtain the internal camera parameters of the virtual camera would be to simply set κ to zero. This can be done directly by changing the respective value of the internal camera parameters.

```
CamParVirtualFixed := CamParOriginal
CamParVirtualFixed[1] := 0
```

Alternatively, the operator `change_radial_distortion_cam_par` can be used with the parameter `Mode` set to *'fixed'* and the parameter `Kappa` set to *0*.

```
change_radial_distortion_cam_par ('fixed', CamParOriginal, 0, \
                                  CamParVirtualFixed)
```

Then, for the rectification of the segmentation results, the HALCON operator `change_radial_distortion_contours_xld` can be used, which requires as input parameters the original and the virtual internal camera parameters. If you want to change the lens distortion of image coordinates (`Row`, `Col`), you can alternatively use `change_radial_distortion_points`.

```
change_radial_distortion_contours_xld (Edges, EdgesRectifiedFixed, \
                                       CamParOriginal, CamParVirtualFixed)
```

The rectification of the segmentation results changes the visible part of the scene (see figure 27b). To obtain virtual camera parameters such that the whole image content lies within the visible part of the scene, the parameter `Mode` of the operator `change_radial_distortion_cam_par` must be set to *'fullsize'* (see figure 27c). Again, to eliminate the lens distortions, the parameter `Kappa` must be set to *0*, or all coefficients of the polynomial model must be set to zero, respectively.

```
change_radial_distortion_cam_par ('fullsize', CamParOriginal, 0, \
                                  CamParVirtualFullsize)
```

If the lens distortions are eliminated in the image itself using the rectification procedure described in section 3.3.2 on page 60, the mode *'fullsize'* may lead to undefined pixels in the rectified image. The mode *'adaptive'* (see figure 27d) slightly reduces the visible part of the scene to prevent such undefined pixels.

```
change_radial_distortion_cam_par ('adaptive', CamParOriginal, 0, \
                                  CamParVirtualAdaptive)
```

The mode *'preserve_resolution'* (see figure 27e) works similar to the mode *'fullsize'* but prevents undefined pixels by additionally increasing the size of the modified image so that the image resolution does not decrease in any part of the image.

```
change_radial_distortion_cam_par ('preserve_resolution', CamParOriginal, 0, \
                                  CamParVirtualPreservedResolution)
```

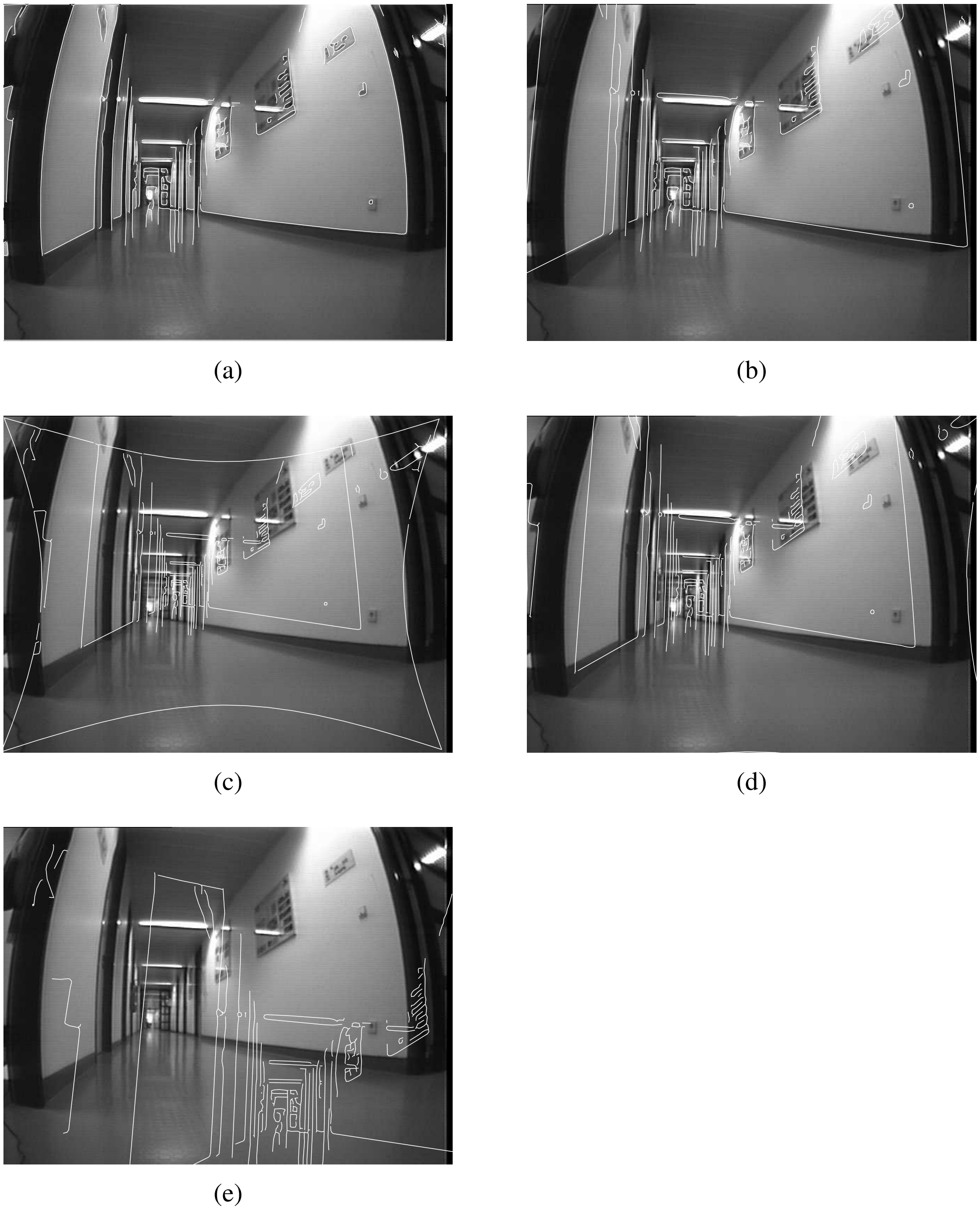


Figure 27: Eliminating radial distortions: The original image overlaid with (a) edges extracted from the original image; (b) edges rectified by setting κ to zero; (c) edges rectified with mode *'fullsize'*; (d) edges rectified with mode *'adaptive'*; (e) edges rectified with mode *'preserved_resolution'*.

Note that this compensation for lens distortions is not possible for line scan images because of the acquisition geometry of line scan cameras. To eliminate radial distortions from segmentation results of line scan images, the segmentation results must be transformed into the WCS (see section 3.2.2 on page 50, section 3.2.3 on page 50, and section 3.2.4 on page 51).

3.3 Rectifying Images

For applications like blob analysis or OCR, it may be necessary to have undistorted images. Imagine that an OCR has been trained based on undistorted image data. Then, it will not be able to recognize characters in heavily distorted images. In such a case, the image data must be rectified, i.e., the lens and perspective distortions must be eliminated before the OCR can be applied.

3.3.1 Transforming Images into the WCS

The operator `image_to_world_plane` rectifies an image by transforming it into the measurement plane, i.e., the plane $z = 0$ of the WCS. The rectified image shows no lens and no perspective distortions. It corresponds to an image captured by a camera that produces no lens distortions and that looks perpendicularly to the measurement plane.

```
image_to_world_plane (Image, ImageMapped, CamParam, PoseForCenteredImage, \
                      WidthMappedImage, HeightMappedImage, \
                      ScaleForCenteredImage, 'bilinear')
```

If more than one image must be rectified, a projection map can be determined with the operator `gen_image_to_world_plane_map`, which is used analogously to the operator `image_to_world_plane`, followed by the actual transformation of the images, which is carried out by the operator `map_image`.

```
gen_image_to_world_plane_map (Map, CamParam, PoseForCenteredImage, \
                              WidthOriginalImage, HeightOriginalImage, \
                              WidthMappedImage, HeightMappedImage, \
                              ScaleForCenteredImage, 'bilinear')
map_image (Image, Map, ImageMapped)
```

The size of the rectified image can be chosen with the parameters `Width` and `Height` for the operator `image_to_world_plane` and with the parameters `WidthMapped` and `HeightMapped` for the operator `gen_image_to_world_plane_map`. The size of the rectified image must be given in pixels.

The pixel size of the rectified image is specified by the parameter `Scale` (see also the description of the operator `image_points_to_world_plane` in section 3.2.2 on page 50). This parameter is the ratio between the pixel size of the rectified image and the unit in which the world coordinates of the calibration target are given (equation 32).

$$\texttt{Scale} = \frac{\text{pixel size of rectified image}}{\text{unit of world coordinates of calibration target}} \tag{32}$$

In many cases the coordinates of the calibration targets are given in meters. In this case, it is possible to set the pixel size directly by setting the parameter `Scale` to *'m'* (corresponding to the value *1.0*, which could be set alternatively for the parameter Scale), *'cm'* (*0.01*), *'mm' (0.001)*, *'microns'* (*1e-6*), or *'µm'* (again, *1e-6*). Then, if the parameter Scale is set to, e.g., 'µm', one pixel of the rectified image has a size that corresponds to an area of 1 µm × 1 µm in the world. The parameter Scale should be chosen such that in the center of the area of interest the pixel size of the input image and of the rectified image is similar. Large scale differences would lead to aliasing or smoothing effects. See below for examples of how the scale can be determined.

The parameter Interpolation specifies whether bilinear interpolation (*'bilinear'*) should be applied between the pixels in the input image or whether the gray value of the nearest neighboring pixel (*'none'*) should be used.

The rectified image ImageWorld is positioned such that its upper left corner is located exactly at the origin of the WCS and that its column axis is parallel to the x-axis of the WCS. Since the WCS is defined by the external camera parameters CamPose the position of the rectified image ImageWorld can be translated by applying the operator set_origin_pose to the external camera parameters. Arbitrary transformations can be applied to the external camera parameters based on homogeneous transformation matrices. See below for examples of how the external camera parameters can be set.

In figure 28, the WCS has been defined such that the upper left corner of the rectified image corresponds to the upper left corner of the input image. To illustrate this, in figure 28, the full domain of the rectified image, transformed into the virtual image plane of the input image, is displayed. As can be seen, the upper left corner of the input image and of the projection of the rectified image are identical.

Note that it is also possible to define the WCS such that the rectified image does not lie or lies only partly within the imaged area. The domain of the rectified image is set such that it contains only those pixels that lie within the imaged area, i.e., for which gray value information is available. In figure 29, the WCS has been defined such that the upper part of the rectified image lies outside the imaged area. To illustrate this, the part of the rectified image for which no gray value information is available is displayed dark gray. Also in figure 29, the full domain of the rectified image, transformed into the virtual image plane of the input image, is displayed. It can be seen that for the upper part of the rectified image no image information is available.

If several images must be rectified using the same camera parameters the operator gen_image_to_world_plane_map in combination with map_image is much more efficient than the operator image_to_world_plane because the transformation must be determined only once. In this case, a projection map that describes the transformation between the image plane and the world plane is generated first by the operator gen_image_to_world_plane_map. Then, this map is used by the operator map_image to transform the image very efficiently.

The following example from examples\solution_guide\3d_machine_vision\transform_image_into_wcs.dev shows how to perform the transformation of images into the world coordinate system using the operators gen_image_to_world_plane_map together with map_image as well as the operator image_to_world_plane.

In the first part of the example program the parameters Scale and CamPose are set such that a given point appears in the center of the rectified image and that in the surroundings of this point the scale of the rectified image is similar to the scale of the original image.

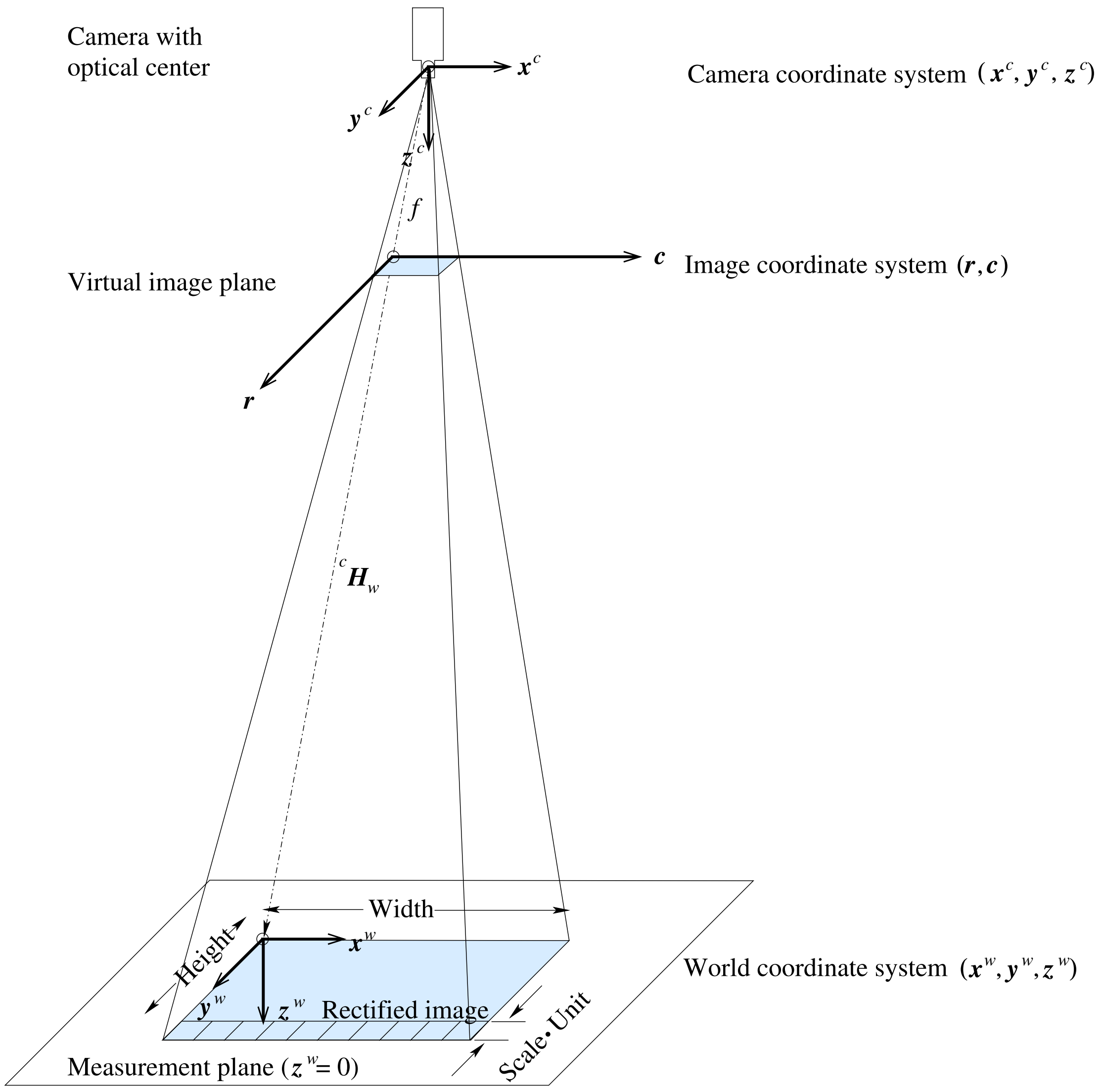

Camera with optical center
x^c
Camera coordinate system (x^c, y^c, z^c)
y^c
z^c
f
c
Image coordinate system (r, c)
Virtual image plane
r
${}^cH_w$
Width
Height
x^w
World coordinate system (x^w, y^w, z^w)
y^w
z^w
Rectified image
Scale• Unit
Measurement plane $(z^w = 0)$

Figure 28: Projection of the image into the measurement plane.

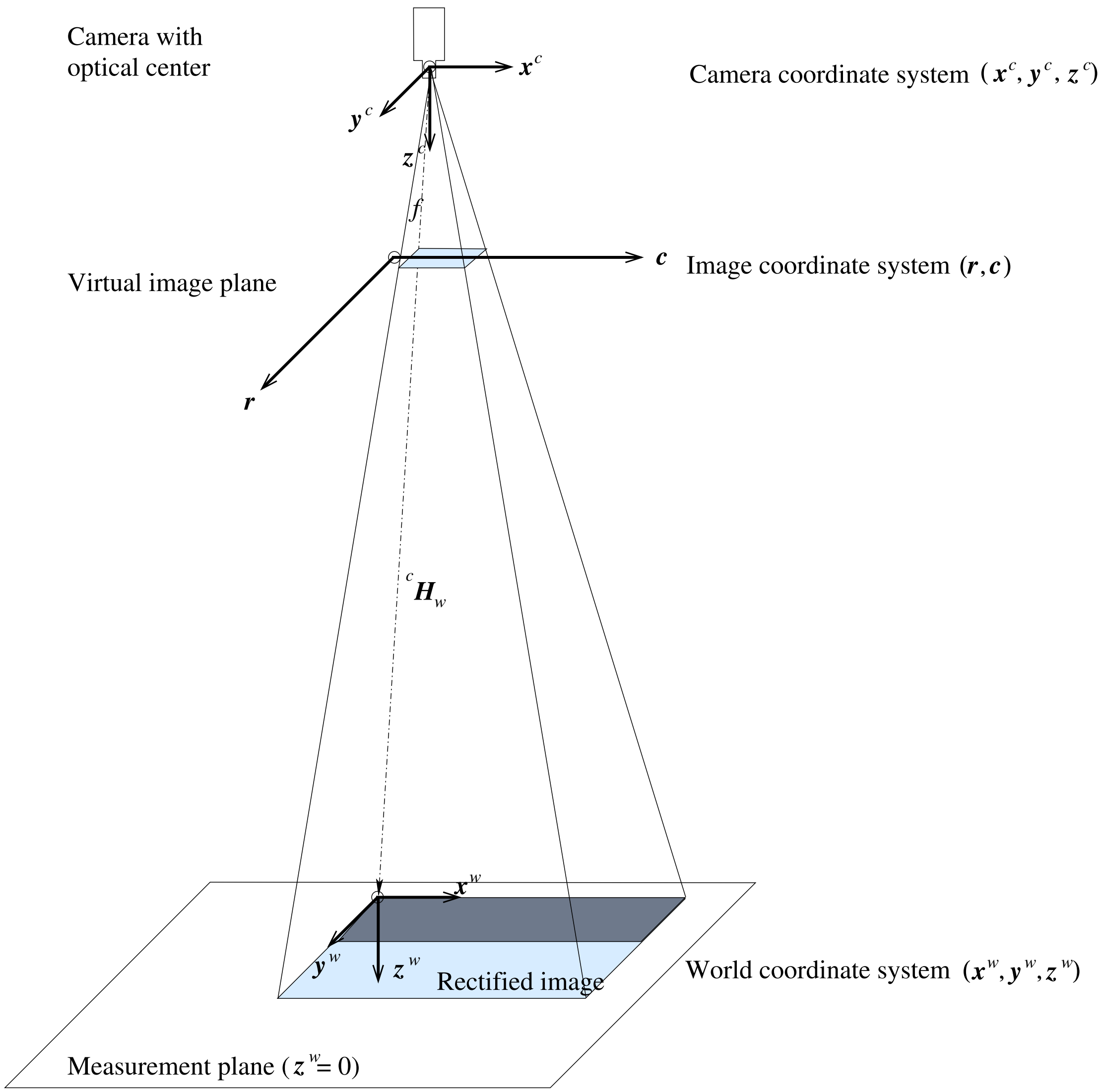

Camera with optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
f
c
Image coordinate system (r, c)
Virtual image plane
r
${}^cH_w$
x^w
y^w
z^w
Rectified image
World coordinate system (x^w, y^w, z^w)
Measurement plane ($z^w = 0$)

Figure 29: Projection of the image into the measurement plane with part of the rectified image lying outside the image area.

First, the size of the rectified image is defined.

```
WidthMappedImage := 652
HeightMappedImage := 494
```

Then, the scale is determined based on the ratio of the distance between points in the WCS and of the respective distance in the ICS.

```
Dist_ICS := 1
image_points_to_world_plane (CamParam, Pose, CenterRow, CenterCol, 1, \
                             CenterX, CenterY)
image_points_to_world_plane (CamParam, Pose, CenterRow+Dist_ICS, CenterCol, \
                             1, BelowCenterX, BelowCenterY)
image_points_to_world_plane (CamParam, Pose, CenterRow, CenterCol+Dist_ICS, \
                             1, RightOfCenterX, RightOfCenterY)
distance_pp (CenterY, CenterX, BelowCenterY, BelowCenterX, \
             Dist_WCS_Vertical)
distance_pp (CenterY, CenterX, RightOfCenterY, RightOfCenterX, \
             Dist_WCS_Horizontal)
ScaleVertical := Dist_WCS_Vertical/Dist_ICS
ScaleHorizontal := Dist_WCS_Horizontal/Dist_ICS
ScaleForCenteredImage := (ScaleVertical+ScaleHorizontal)/2.0
```

Now, the pose of the measurement plane is modified such that a given point will be displayed in the center of the rectified image.

```
DX := CenterX-ScaleForCenteredImage*WidthMappedImage/2.0
DY := CenterY-ScaleForCenteredImage*HeightMappedImage/2.0
DZ := 0
set_origin_pose (Pose, DX, DY, DZ, PoseForCenteredImage)
```

These calculations are implemented in the HDevelop procedure

```
procedure parameters_image_to_world_plane_centered (: : CamParam, Pose,
                                                       CenterRow, CenterCol,
                                                       WidthMappedImage,
                                                       HeightMappedImage:
                                                       ScaleForCenteredImage,
                                                       PoseForCenteredImage)
```

which is part of the example program `examples\solution_guide\3d_machine_vision\transform_image_into_wcs.dev` (see appendix B.2 on page 168).

Finally, the image can be transformed.

```
gen_image_to_world_plane_map (Map, CamParam, PoseForCenteredImage, \
                              WidthOriginalImage, HeightOriginalImage, \
                              WidthMappedImage, HeightMappedImage, \
                              ScaleForCenteredImage, 'bilinear')
map_image (Image, Map, ImageMapped)
```

The second part of the example program `examples\solution_guide\3d_machine_vision\transform_image_into_wcs.dev` shows how to set the parameters `Scale` and `CamPose` such that the entire image is visible in the rectified image.

First, the image coordinates of the border of the original image are transformed into world coordinates.

```
full_domain (Image, ImageFull)
get_domain (ImageFull, Domain)
gen_contour_region_xld (Domain, ImageBorder, 'border')
contour_to_world_plane_xld (ImageBorder, ImageBorderWCS, CamParam, Pose, 1)
```

Then, the extent of the image in world coordinates is determined.

```
smallest_rectangle1_xld (ImageBorderWCS, MinY, MinX, MaxY, MaxX)
ExtentX := MaxX-MinX
ExtentY := MaxY-MinY
```

The scale is the ratio of the extent of the image in world coordinates and of the size of the rectified image.

```
ScaleX := ExtentX/WidthMappedImage
ScaleY := ExtentY/HeightMappedImage
```

Now, the maximum value must be selected as the final scale.

```
ScaleForEntireImage := max([ScaleX,ScaleY])
```

Finally, the origin of the pose must be translated appropriately.

```
set_origin_pose (Pose, MinX, MinY, 0, PoseForEntireImage)
```

These calculations are implemented in the HDevelop procedure

```
procedure parameters_image_to_world_plane_entire (Image: : CamParam, Pose,
                                                 WidthMappedImage,
                                                 HeightMappedImage:
                                                 ScaleForEntireImage,
                                                 PoseForEntireImage)
```

which is part of the example program `examples\solution_guide\3d_machine_vision\transform_image_into_wcs.dev` (see appendix B.3 on page 169).

If the object is not planar the projection map that is needed by the operator `map_image` may be determined by the operator `gen_grid_rectification_map`, which is described in section 9.3 on page 142.

If only the lens distortions should be eliminated the projection map can be determined by the operator `gen_radial_distortion_map`, which is described in the following section.

3.3.2 Compensate for Lens Distortions Only

The principle of the compensation for lens distortions has already be described in section 3.2.6 on page 51.

If only one image must be rectified the operator change_radial_distortion_image can be used. It is used analogously to the operator change_radial_distortion_contours_xld described in section 3.2.6, with the only exception that a region of interest (ROI) can be defined with the parameter Region.

```
change_radial_distortion_image (GrayImage, ROI, ImageRectifiedAdaptive, \
                                CamParOriginal, CamParVirtualAdaptive)
```

Again, the internal parameters of the virtual camera that does not show lens distortions can be determined by setting κ to zero for the division model or K_1, K_2, K_3, P_1, and P_2 to zero for the polynomial model (see figure 30b). Alternatively, the internal parameters of the virtual camera can be obtained by using the operator change_radial_distortion_cam_par with the parameter Mode set to *'fixed'* (equivalent to setting κ or the coefficients of the polynomial model to zero; see figure 30b), *'adaptive'* (see figure 30c), *'fullsize'* (see figure 30d), or *'preserve_resolution'* (see figure 30e).

If more than one image must be rectified, a projection map can be determined with the operator gen_radial_distortion_map, which is used analogously to the operator change_radial_distortion_image, followed by the actual transformation of the images, which is carried out by the operator map_image, described in section 3.3.1 on page 54. If a ROI is to be specified, it must be rectified separately (see section 3.2.4 on page 51).

```
gen_radial_distortion_map (MapFixed, CamParOriginal, CamParVirtualFixed, \
                           'bilinear')
map_image (GrayImage, MapFixed, ImageRectifiedFixed)
```

Note that this compensation for lens distortions is not possible for line scan images because of the acquisition geometry of line scan cameras. To eliminate radial distortions from line scan images, the images must be transformed into the WCS (see section 3.3.1 on page 54).

3.4 Inspection of Non-Planar Objects

Note that the measurements described so far will only be accurate if the object to be measured is planar, i.e., if it has a flat surface. If this is not the case the perspective projection of the pinhole camera (see equation 22 on page 22) will make the parts of the object that lie closer to the camera appear bigger than the parts that lie farther away. In addition, the respective world coordinates are displaced systematically. If you want to measure the top side of objects with a flat surface that have a significant thickness that is equal for all objects it is best to place the calibration plate onto one of these objects during calibration. With this, you can make sure that the optical rays are intersected with the correct plane.

The displacement that results from deviations of the object surface from the measurement plane can be estimated very easily. Figure 31 shows a vertical section of a typical measurement configuration. The measurement plane is drawn as a thick line, the object surface as a dotted line. Note that the object surface does not correspond to the measurement plane in this case. The deviation of the object surface from the

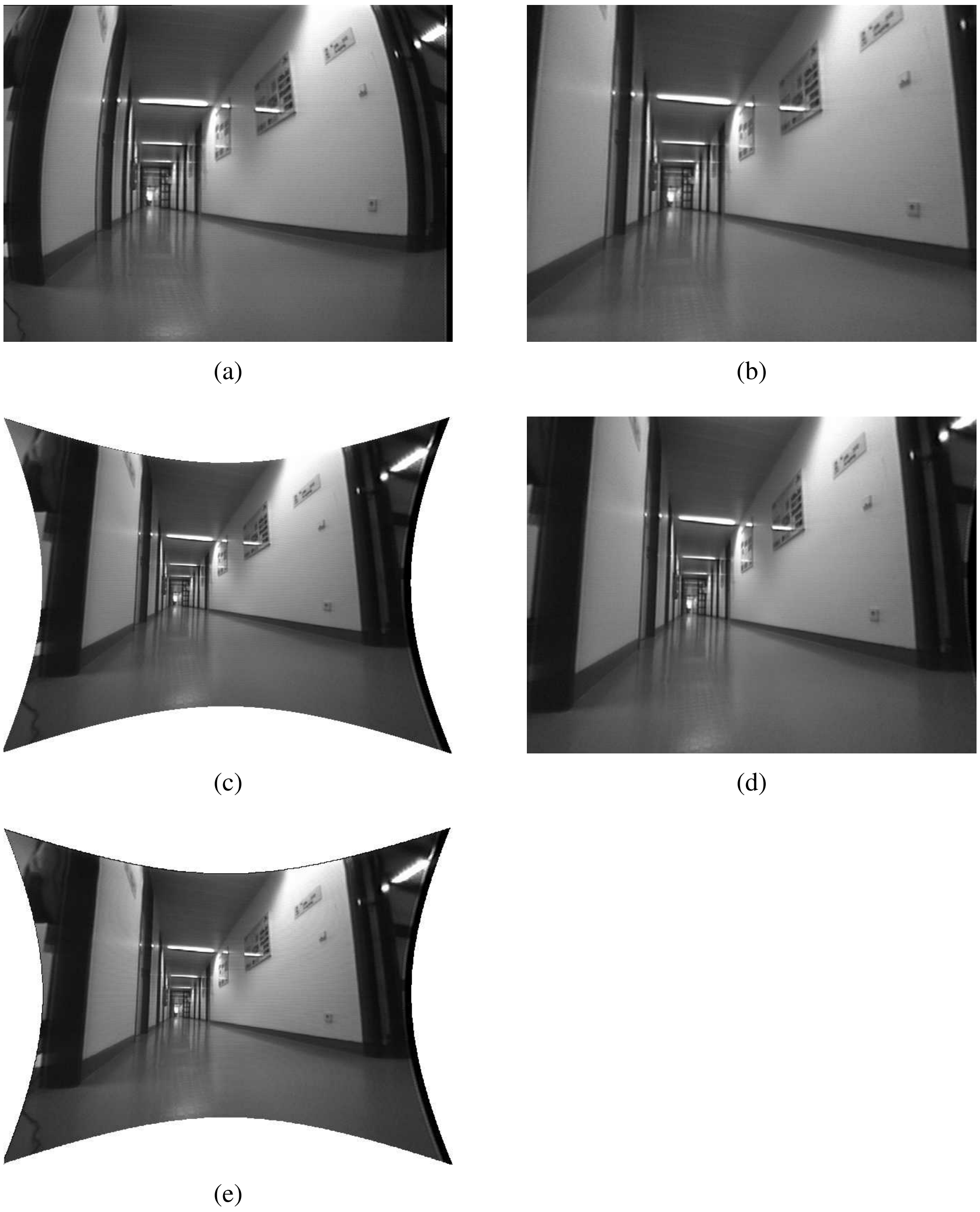


Figure 30: Eliminating radial distortions: (a) The original image; (b) the image rectified by setting κ to zero; (c) the image rectified with mode *'fullsize'*; (d) the image rectified with mode *'adaptive'*; (e) the image rectified with mode *'preserved_resolution'*.

measurement plane is indicated by Δz, the distance of the projection center from the measurement plane by z, and the displacement by Δr. The point N indicates the perpendicular projection of the projection center (PC) onto the measurement plane.

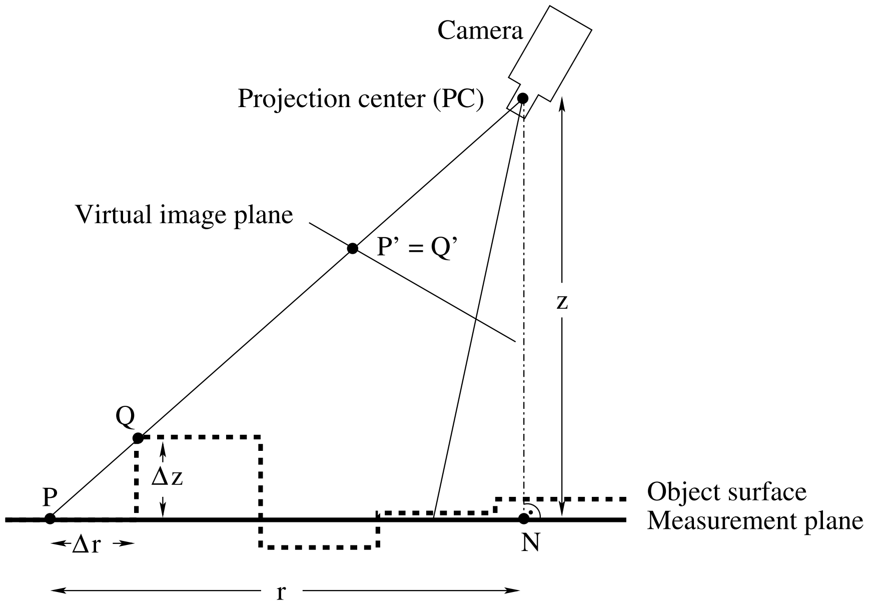


Figure 31: Displacement Δr caused by a deviation of the object surface from the measurement plane.

For the determination of the world coordinates of point Q, which lies on the object surface, the optical ray from the projection center of the camera through Q', which is the projection of Q into the image plane, is intersected with the measurement plane. For this reason, the operators of the family `..._to_world_plane` do not return the world coordinates of Q, but the world coordinates of point P, which is the perspective projection of point Q' onto the measurement plane.

If we know the distance r from P to N, the distance z, which is the shortest distance from the projection center to the measurement plane, and the deviation Δz of the object's surface from the measurement plane, the displacement Δr can be calculated by:

$$\Delta r = \Delta z \cdot \frac{r}{z} \tag{33}$$

Often, it will be sufficient to have just a rough estimate for the value of Δr. Then, the values r, z, and Δz can be approximately determined directly from the measurement setup.

If you need to determine Δr more precisely, you first have to calibrate the camera. Then you have to select a point Q' in the image for which you want to know the displacement Δr. The transformation of Q' into the WCS using the operator `image_points_to_world_plane` yields the world coordinates of point P. Now, you need to derive the world coordinates of the point N. An easy way to do this is to transform the camera coordinates of the projection center PC, which are $(0,0,0)^T$, into the world coordinate system, using the operator `affine_trans_point_3d`. To derive the homogeneous transformation matrix ${}^{WCS}\mathbf{H}_{CCS}$ needed for the above mentioned transformation, first, generate the homogeneous transformation matrix ${}^{CCS}\mathbf{H}_{WCS}$ from the pose of the measurement plane via the operator `pose_to_hom_mat3d` and then, invert the resulting homogeneous transformation matrix (`hom_mat3d_invert`). Because N is the perpendicular projection of PC onto the measurement plane, its x and y world coordinates are equal to the respective world coordinates of PC and its z coordinate is equal to zero. Now, r and z can be derived as follows: r is the distance from P to N, which can be calculated by the operator `distance_pp`; z is simply the z coordinate of PC, given in the WCS.

The following HALCON program (`examples\solution_guide\3d_machine_vision\height_displacement.dev`) shows how to implement this approach. First, the camera parameters are read from file.

```
read_cam_par ('camera_parameters.dat', CamParam)
read_pose ('pose_from_three_points.dat', Pose)
```

Then, the deviation of the object surface from the measurement plane is set.

```
DeltaZ := 2
```

Finally, the displacement is calculated, according to the method described above.

```
get_mbutton (WindowHandle, RowQ, ColumnQ, _)
image_points_to_world_plane (CamParam, Pose, RowQ, ColumnQ, 1, WCS_PX, \
                             WCS_PY)
pose_to_hom_mat3d (Pose, CCS_HomMat_WCS)
hom_mat3d_invert (CCS_HomMat_WCS, WCS_HomMat_CCS)
affine_trans_point_3d (WCS_HomMat_CCS, 0, 0, 0, WCS_PCX, WCS_PCY, WCS_PCZ)
distance_pp (WCS_PX, WCS_PY, WCS_PCX, WCS_PCY, r)
z := fabs(WCS_PCZ)
DeltaR := DeltaZ*r/z
```

Assuming a constant Δz, the following conclusions can be drawn for Δr:

- Δr increases with increasing r.
- If the measurement plane is more or less perpendicular to the optical axis, Δr increases towards the image borders.
- At the point N, Δr is always equal to zero.
- Δr increases the more the measurement plane is tilted with respect to the optical axis.

The maximum acceptable deviation of the object's surface from the measurement plane, given a maximum value for the resulting displacement, can be derived by the following formula:

$$\Delta z = \Delta r \cdot \frac{z}{r} \qquad (34)$$

The values for r and z can be determined as described above.

If you want to inspect an object that has a surface that consists of several parallel planes you can first use equation 34 to evaluate if the measurement errors stemming from the displacements are acceptable within your project or not. If the displacements are too large, you can calibrate the camera such that the measurement plane corresponds to, e.g., the uppermost plane of the object. Now, you can derive a pose for each plane, which is parallel to the uppermost plane simply by applying the operator set_origin_pose. This approach is also useful if objects of different thickness may appear on the assembly line. If it is possible to classify these objects into classes corresponding to their thickness, you can select the appropriate pose for each object. Thus, it is possible to derive accurate world coordinates for each object.

Note that if the plane in which the object lies is severely tilted with respect to the optical axis, and if the object has a significant thickness, the camera will likely see some parts of the object that you do not want to measure. For example, if you want to measure the top side of a cube and the plane is tilted, you will see the side walls of the cube as well, and therefore might measure the wrong dimensions. Therefore, it is usually best to align the camera so that its optical axis is perpendicular to the plane in which the objects are measured. If the objects do not have significant thickness, you can measure them accurately even if the plane is tilted.

What is more, it is even possible to derive world coordinates for an object's surface that consists of several non-parallel planes if the relation between the individual planes is known. In this case, you may define the relative pose of the tilted plane with respect to an already known measurement plane.

```
RelPose := [0,3.2,0,-14,0,0,0]
```

Then, you can transform the known pose of the measurement plane into the pose of the tilted plane.

```
pose_to_hom_mat3d (FinalPose, HomMat3D)
pose_to_hom_mat3d (RelPose, HomMat3DRel)
hom_mat3d_compose (HomMat3D, HomMat3DRel, HomMat3DAdapted)
hom_mat3d_to_pose (HomMat3DAdapted, PoseAdapted)
```

Alternatively, you can use the operators of the family hom_mat3d_..._local to adapt the pose.

```
hom_mat3d_translate_local (HomMat3D, 0, 3.2, 0, HomMat3DTranslate)
hom_mat3d_rotate_local (HomMat3DTranslate, rad(-14), 'x', HomMat3DAdapted)
hom_mat3d_to_pose (HomMat3DAdapted, PoseAdapted)
```

Now, you can obtain world coordinates for points lying on the tilted plane, as well.

```
contour_to_world_plane_xld (Lines, ContoursTrans, CamParam, PoseAdapted, 1)
```

If the object is to complex to be approximated by planes, or if the relations between the planes are not known, it is not possible to perform precise measurements in world coordinates using the methods described in this section. In this case, it is necessary to use two cameras and to apply the HALCON stereo operators described in section 7 on page 100.

4 Calibrated Mosaicking

Some objects are too large to be covered by one single image. Multiple images that cover different parts of the object must be taken in such cases. You can measure precisely across the different images if the cameras are calibrated and their external parameters are known with respect to one common world coordinate system.

It is even possible to merge the individual images into one larger image that covers the whole object. This is done by rectifying the individual images with respect to the same measurement plane (see section 3.3.1 on page 54). In the resulting image, you can measure directly in world coordinates.

Note that the 3D coordinates of objects are derived based on the same principle as described in section 3 on page 29, i.e., a measurement plane that coincides with the object surface must be defined. Although two or more cameras are used, this is no stereo approach. For more information on 3D vision with a binocular stereo system, please refer to section 7 on page 100.

If the resulting image is not intended to serve for high-precision measurements in world coordinates, you can generate it using the mosaicking approach described in section 5 on page 76. With this approach, it is not necessary to calibrate the cameras.

A setup for generating a high-precision mosaic image from two cameras is shown in figure 32. The cameras are mounted such that the resulting pair of images has a small overlap. The cameras are first calibrated and then the images are merged together into one larger image. All further explanations within this section refer to such a two-camera setup.

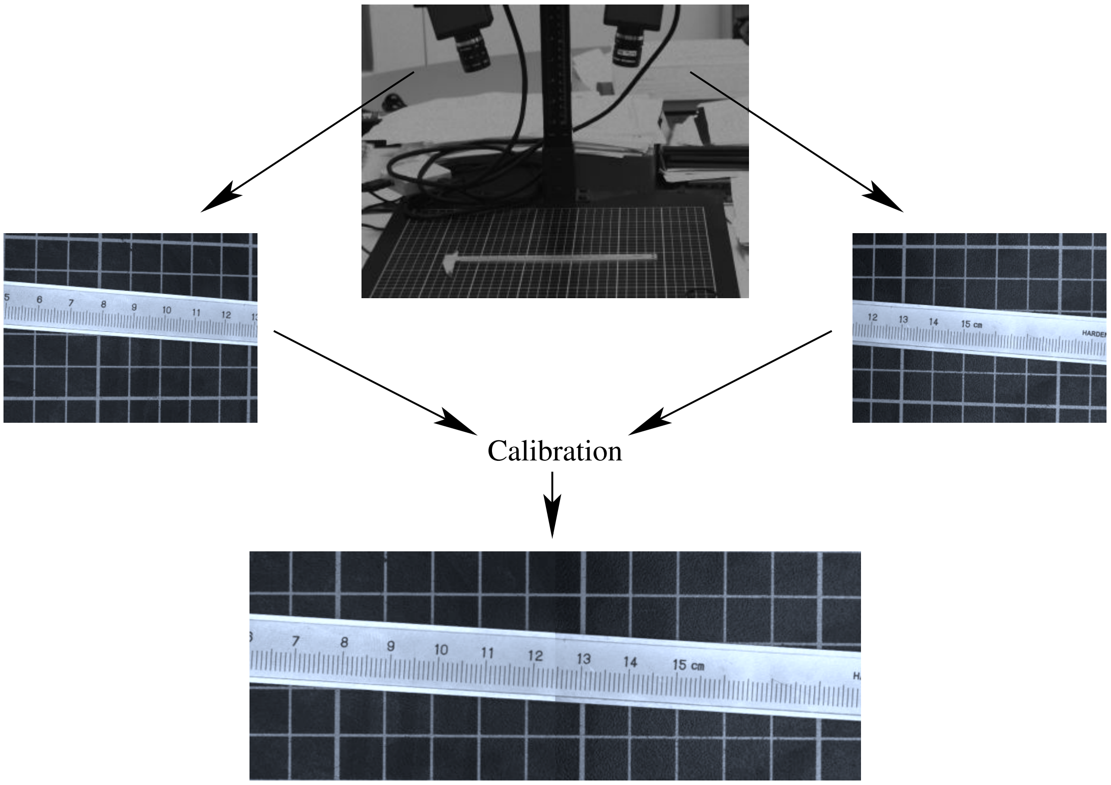


Figure 32: Two-camera setup.

Typically, the following steps must be carried out:

1. Determination of the internal camera parameters for each camera separately.
2. Determination of the external camera parameters, using *one* calibration object, to facilitate that the relation between the cameras can be determined.
3. Merge the images into one larger image that covers the whole object.

4.1 Setup

Two or more cameras must be mounted on a *stable* platform such that each image covers a part of the whole scene. The cameras can have an arbitrary orientation, i.e., it is not necessary that they are looking parallel or perpendicular onto the object surface.

To setup focus, illumination, and overlap appropriately, use a big reference object that covers all fields of view. To permit that the images are merged into one larger image, they must have some overlap (see figure 33 for an example). The overlapping area can be even smaller than depicted in figure 33, since the overlap is only necessary to ensure that there are no gaps in the resulting combined image.

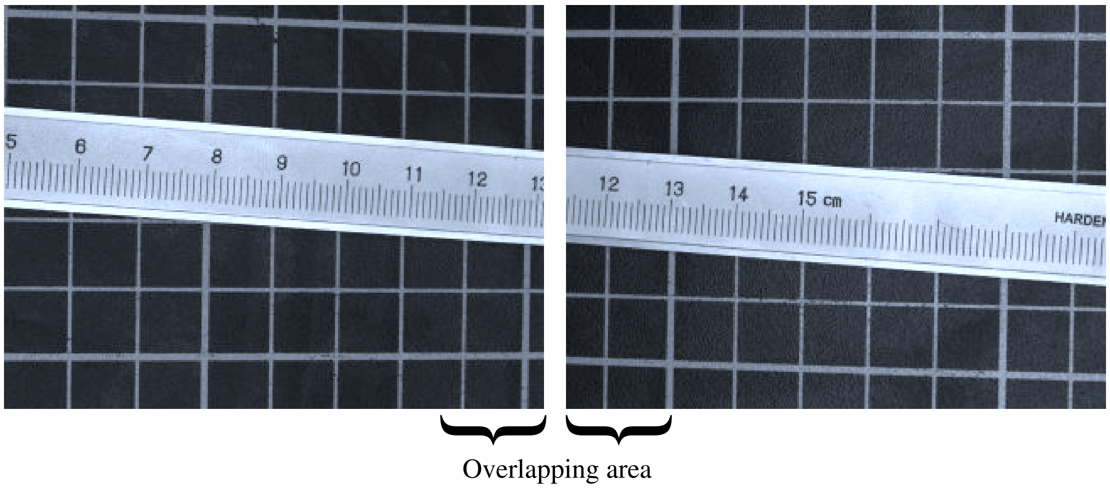


Figure 33: Overlapping images.

4.2 Calibration

The calibration of the images can be broken down into two separate steps.

The first step is to determine the internal camera parameters for each of the cameras in use. This can be done for each camera independently, as described in section 3.1.5 on page 39.

The second step is to determine the external camera parameters for all cameras. Because the final coordinates should refer to *one* world coordinate system for all images, a big calibration object that appears in all images has to be used. We propose to use a calibration object like the one displayed in figure 34, which consists of as many calibration plates as the number of cameras that are used.

For the determination of the external camera parameters, it is sufficient to use one calibration image from each camera only. Note that the calibration object must not be moved in between the acquisition of the individual images. Ideally, the images are acquired simultaneously.

In each image, at least three points of the calibration object, i.e., points for which the world coordinates are known, have to be measured in the images. Based on these point correspondences, the operator `camera_calibration` can determine the external camera parameters for each camera. See section 3.1.6 on page 42 for details.

The calibration is easy if standard HALCON calibration plates mounted on some kind of carrier plate are used such that in each image one calibration plate is completely visible. An example for such a calibration object for a two-camera setup is given in figure 34. The respective calibration images for the determination of the external camera parameters are shown in figure 35. Note that the relative position of the calibration plates with respect to each other must be known precisely. This can be done with the pose estimation described in section 6 on page 90.

Note also that only the relative position of the calibration marks among each other shows the high accuracy stated in section 3.1.7 on page 46 but not the borders of the calibration plate. The rows of calibration marks may be slanted with respect to the border of the calibration plate and even the distance of the calibration marks from the border of the calibration plate may vary. Therefore, aligning the calibration plates along their boundaries may result in a shift in x- *and* y-direction with respect to the coordinate system of the calibration plate in its initial position.

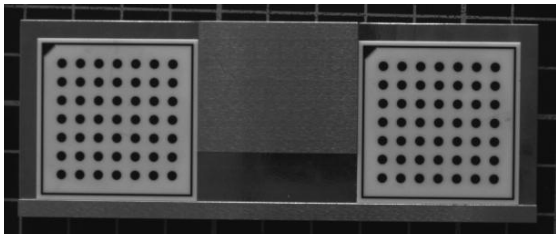


Figure 34: Calibration object for two-camera setup.

The world coordinates of the calibration marks of each calibration plate can be read from the respective calibration plate description file.

```
caltab_points (CaltabName, X, Y, Z)
```

If an arbitrary calibration object is used, these coordinates must be determined precisely in advance. For HALCON calibration plates, the image coordinates of the calibration marks as well as the initial values for the poses can be determined easily with the operators `find_caltab` and `find_marks_and_pose`.

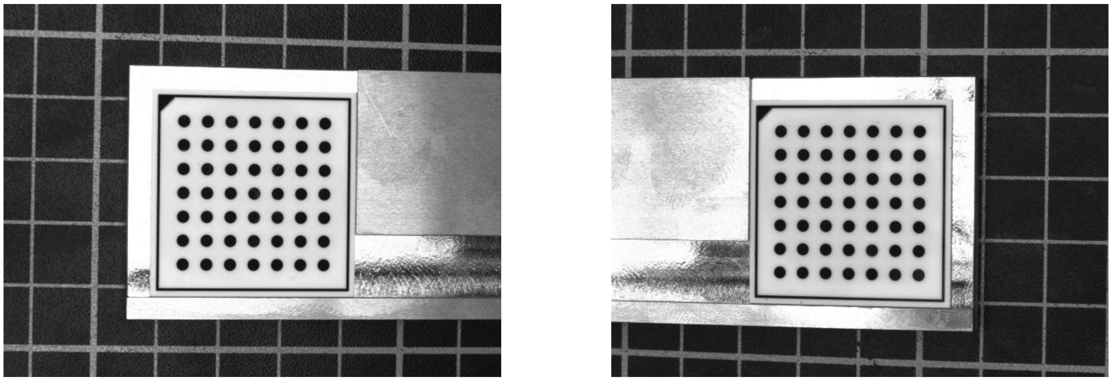


Figure 35: Calibration images for two-camera setup.

```
min_max_gray (Image1, Image1, 3, Min, _, _)
find_caltab (Image1, Caltab, CaltabName, 3, min([Min+40,200]), 5)
find_marks_and_pose (Image1, Image1, CaltabName, CamParam1, 128, 10, 18, \
                     0.7, 15, 100, RCoord1, CCoord1, StartPose1)
min_max_gray (Image2, Image2, 3, Min, _, _)
find_caltab (Image2, Caltab, CaltabName, 3, min([Min+40,200]), 5)
find_marks_and_pose (Image2, Image2, CaltabName, CamParam2, 128, 10, 18, \
                     0.7, 15, 100, RCoord2, CCoord2, StartPose2)
```

The following code fragments shows the calibration of the two-camera setup. It assumes that the internal camera parameters of both cameras are already known.

```
camera_calibration (X, Y, Z, RCoord1, CCoord1, CamParam1, StartPose1, \
                    'pose', _, Pose1, Errors1)
camera_calibration (X, Y, Z, RCoord2, CCoord2, CamParam2, StartPose2, \
                    'pose', _, Pose2, Errors2)
```

4.3 Merging the Individual Images into One Larger Image

At first, the individual images must be rectified, i.e, transformed so that they exactly fit together. This can be achieved by using the operators gen_image_to_world_plane_map and map_image. Then, the mosaic image can be generated by the operator tile_images, which tiles multiple images into one larger image. These steps are visualized in figure 36.

The operators gen_image_to_world_plane_map and map_image are described in section 3.3.1 on page 54. In the following, we will only discuss the problem of defining the appropriate image detail, i.e., the position of the upper left corner and the size of the rectified images. Again, the description is based on the two-camera setup.

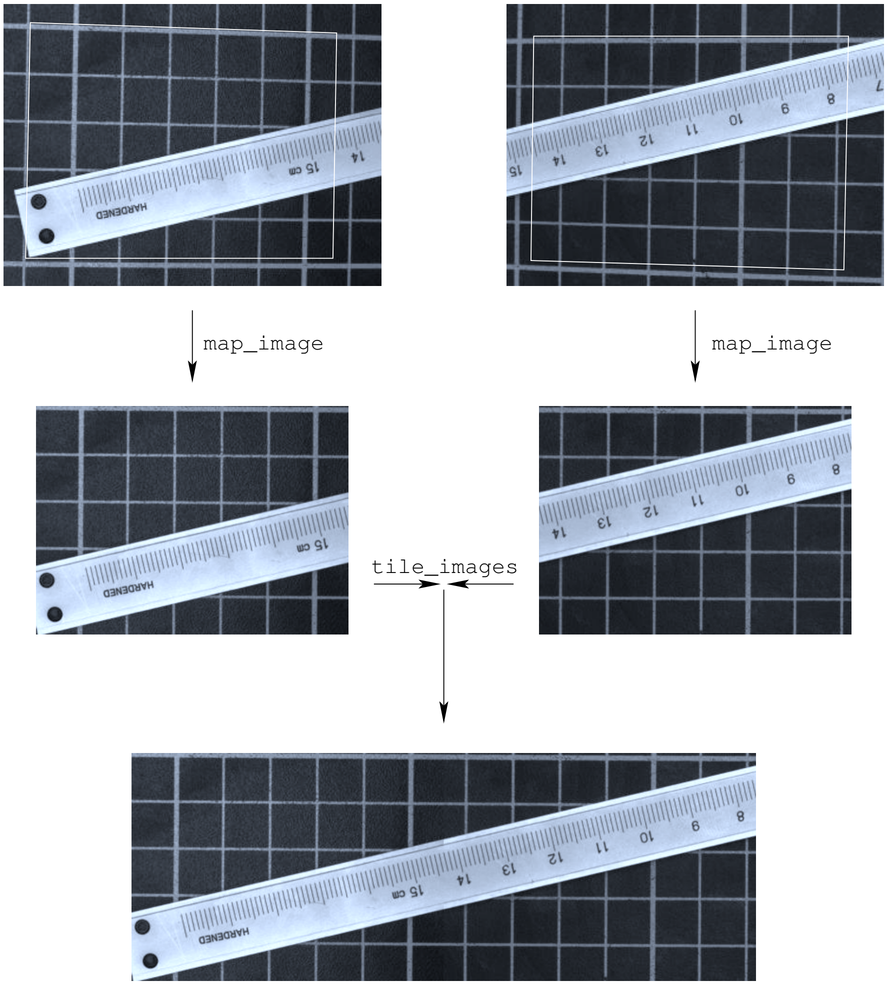


Figure 36: Image rectification and tiling.

4.3.1 Definition of the Rectification of the First Image

For the first (here: left) image, the determination of the necessary shift of the pose is straightforward. You can define the upper left corner of the rectified image in image coordinates, e.g., interactively or, as in the example program, based on a preselected border width.

```
ULRow := HeightImage1*BorderInPercent/100.0
ULCol := WidthImage1*BorderInPercent/100.0
```

Then, this point must be transformed into world coordinates.

```
image_points_to_world_plane (CamParam1, Pose1, ULRow, ULCol, 'm', ULX, ULY)
```

The resulting coordinates can be used directly, together with the shift that compensates the thickness of the calibration plate (see section 3.1.6 on page 42) to modify the origin of the world coordinate system in the left image.

```
set_origin_pose (Pose1, ULX, ULY, DiffHeight, PoseNewOrigin1)
```

This means that we shift the origin of the world coordinate system from the center of the calibration plate to the position that defines the upper left corner of the rectified image (figure 37).

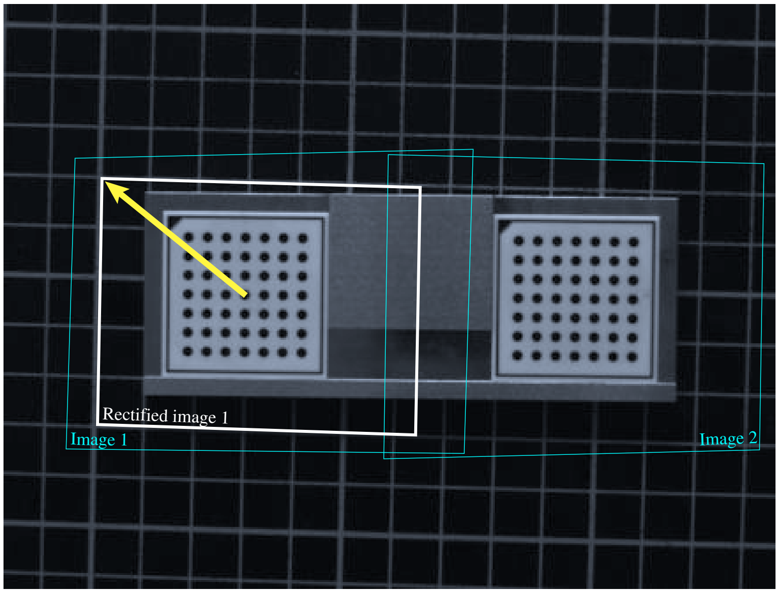


Figure 37: Definition of the upper left corner of the first rectified image.

The size of the rectified image, i.e., its width and height, can be determined from points originally defined in image coordinates, too. In addition, the desired pixel size of the rectified image must be specified.

```
PixelSize := 0.0001
```

For the determination of the height of the rectified image we need to define a point that lies near the lower border of the first image.

```
LowerRow := HeightImage1*(100-BorderInPercent)/100.0
```

Again, this point must be transformed into the world coordinate system.

```
image_points_to_world_plane (CamParam1, Pose1, LowerRow, ULCol, 'm', _, \
                             LowerY)
```

The height can be determined as the vertical distance between the upper left point and the point near the lower image border, expressed in pixels of the rectified image.

```
HeightRect := int((LowerY-ULY)/PixelSize)
```

Analogously, the width can be determined from a point that lies in the overlapping area of the two images, i.e., near the right border of the first image.

```
RightCol := WidthImage1*(100-OverlapInPercent/2.0)/100.0
image_points_to_world_plane (CamParam1, Pose1, ULRow, RightCol, 'm', RightX, \
                             _)
WidthRect := int((RightX-ULX)/PixelSize)
```

Note that the above described definitions of the image points, from which the upper left corner and the size of the rectified image are derived, assume that the x- and y-axes of the world coordinate system are approximately aligned to the column- and row-axes of the first image. This can be achieved by placing the calibration plate in the first image approximately aligned with the image borders. Otherwise, the distances between the above mentioned points make no sense and the upper left corner and the size of the rectified image must be determined in a manner that is adapted for the configuration at hand.

With the shifted pose and the size of the rectified image, the rectification map for the first image can be derived.

```
gen_image_to_world_plane_map (MapSingle1, CamParam1, PoseNewOrigin1, Width, \
                              Height, WidthRect, HeightRect, PixelSize, \
                              'bilinear')
```

4.3.2 Definition of the Rectification of the Second Image

The second image must be rectified such that it fits exactly to the right of the first rectified image. This means that the upper left corner of the second rectified image must be identical with the upper right corner of the first rectified image. Therefore, we need to know the coordinates of the upper right corner of the first rectified image in the coordinate system that is defined by the calibration plate in the second image.

First, we express the upper right corner of the first rectified image in the world coordinate system that is defined by the calibration plate in the first image. It can be determined by a transformation from the origin into the upper left corner of the first rectified image (a translation in the example program) followed by a translation along the upper border of the first rectified image. Together with the shift that compensates the thickness of the calibration plate, this transformation is represented by the homogeneous transformation matrix ${}^{cp1}\mathbf{H}_{ur1}$ (see figure 38), which can be defined in HDevelop by:

```
hom_mat3d_translate_local (HomMat3DIdentity, ULX+PixelSize*WidthRect, ULY, \
                           DiffHeight, cp1Hur1)
```

Then, we need the transformation between the two calibration plates of the calibration object. The homogeneous transformation matrix ${}^{cp1}\mathbf{H}_{cp2}$ describes how the world coordinate system defined by

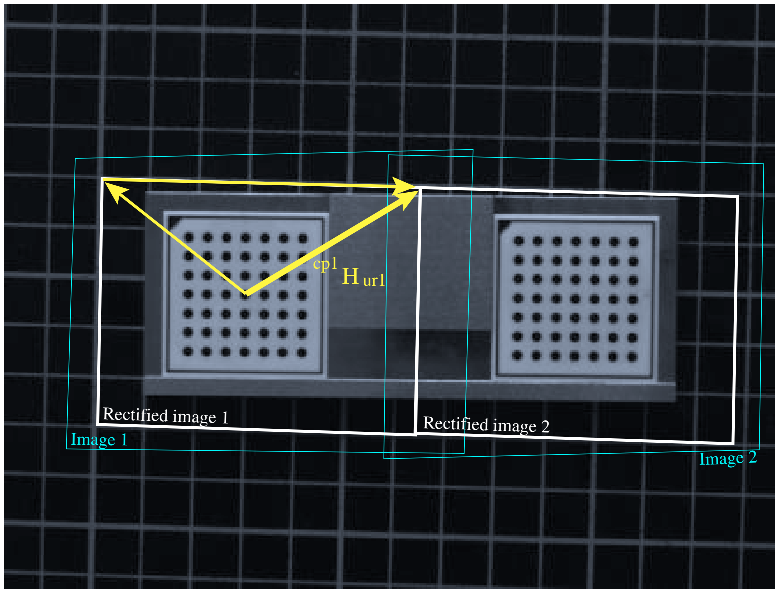


Figure 38: Definition of the upper right corner of the first rectified image.

the calibration plate in the first image is transformed into the world coordinate system defined by the calibration plate in the second image (figure 39). This transformation must be known beforehand from a precise measurement of the calibration object.

From these two transformations, it is easy to derive the transformation that transforms the world coordinate system of the second image such that its origin lies in the upper left corner of the second rectified image. For this, the two transformations have to be combined appropriately (see figure 40):

$$\begin{aligned} {}^{cp2}\mathbf{H}_{ul2} &= {}^{cp2}\mathbf{H}_{cp1} \cdot {}^{cp1}\mathbf{H}_{ur1} && (35) \\ &= \left({}^{cp1}\mathbf{H}_{cp2}\right)^{-1} \cdot {}^{cp1}\mathbf{H}_{ur1} && (36) \end{aligned}$$

This can be implemented in HDevelop as follows:

```
hom_mat3d_invert (cp1Hcp2, cp2Hcp1)
hom_mat3d_compose (cp2Hcp1, cp1Hur1, cp2Hul2)
```

With this, the pose of the calibration plate in the second image can be modified such that the origin of the world coordinate system lies in the upper left corner of the second rectified image:

```
pose_to_hom_mat3d (Pose2, cam2Hcp2)
hom_mat3d_compose (cam2Hcp2, cp2Hul2, cam2Hul2)
hom_mat3d_to_pose (cam2Hul2, PoseNewOrigin2)
```

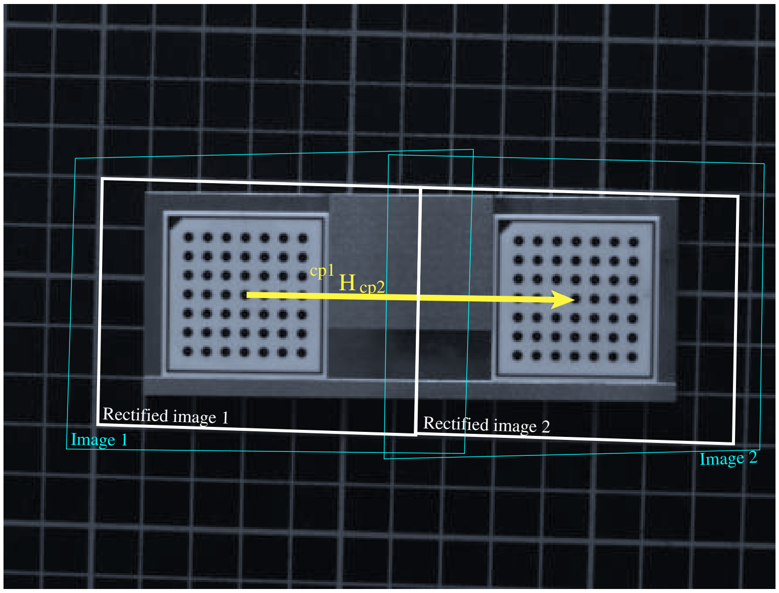


Figure 39: Transformation between the two world coordinate systems, each defined by the respective calibration plate.

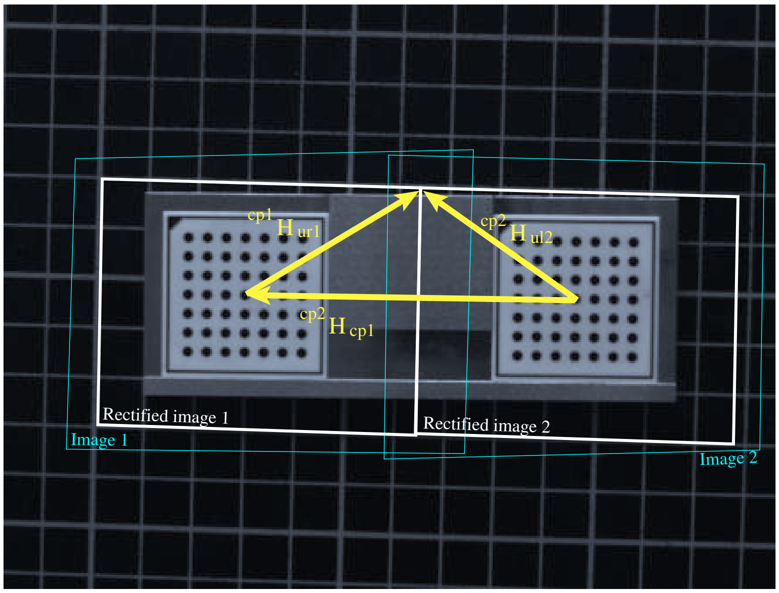


Figure 40: Definition of the upper left corner of the second rectified image.

With the resulting new pose and the size of the rectified image, which can be the same as for the first rectified image, the rectification map for the second image can be derived.

```
gen_image_to_world_plane_map (MapSingle2, CamParam2, PoseNewOrigin2, Width, \
                              Height, WidthRect, HeightRect, PixelSize, \
                              'bilinear')
```

4.3.3 Rectification of the Images

Once the rectification maps are created, every image pair from the two-camera setup can be rectified and tiled very efficiently. The resulting mosaic image consists of the two rectified images and covers a part as indicated in figure 41.

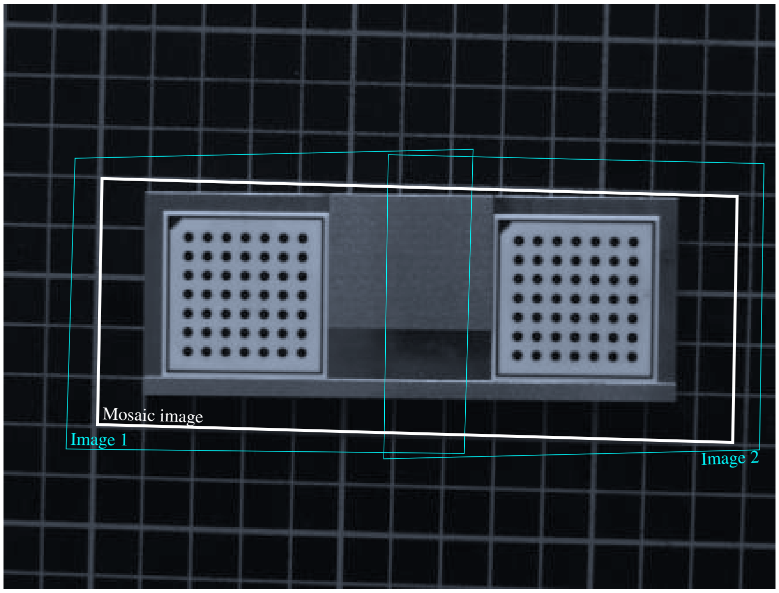


Figure 41: The position of the final mosaic image.

The rectification is carried out by the operator map_image.

```
map_image (Image1, MapSingle1, RectifiedImage1)
map_image (Image2, MapSingle2, RectifiedImage2)
```

This transforms the two images displayed in figure 42, into the two rectified images that are shown in figure 43.

As a preparation for the tiling, the rectified images must be concatenated into one tuple, which then contains both images.

```
concat_obj (RectifiedImage1, RectifiedImage2, Concat)
```

Then the two images can be tiled.

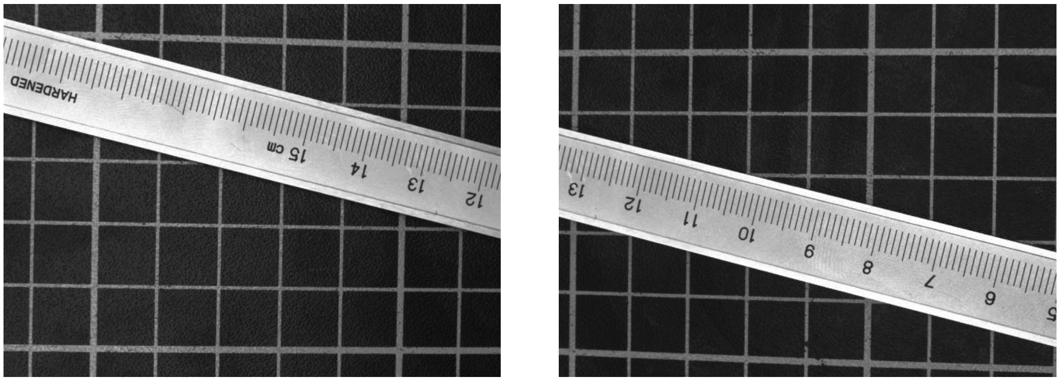


Figure 42: Two test images acquired with the two-camera setup.

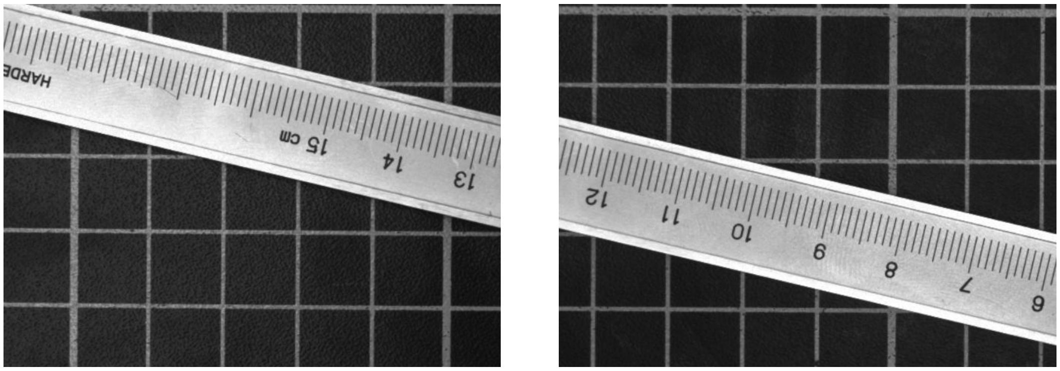


Figure 43: Rectified images.

```
tile_images (Concat, Combined, 2, 'vertical')
```

The resulting mosaic image is displayed in figure 44.

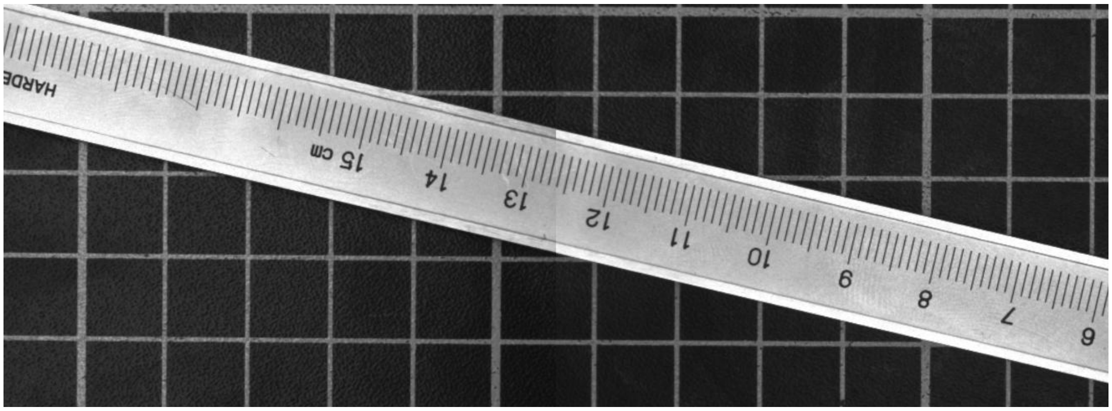


Figure 44: Mosaic image.

5 Uncalibrated Mosaicking

If you need an image of a large object, but the field of view of the camera does not allow to cover the entire object with the desired resolution, you can use image mosaicking to generate a large image of the entire object from a sequence of overlapping images of parts of the object.

An example for such an application is given in figure 45. On the left side, six separate images are displayed stacked upon each other. On the right side, the mosaic image generated from the six separate images is shown. Note that the folds visible in the image do not result from the mosaicking. They are due to some degradations on the PCB, which can be seen already in the separate images.

The mosaicking approach described in this section is designed for applications where it is not necessary to achieve the high-precision mosaic images as described in section 4 on page 64. The advantages compared to this approach are that no camera calibration is necessary and that the individual images can be arranged automatically.

The following example program (`examples\solution_guide\3d_machine_vision\mosaicking.dev`) generates the mosaic image displayed in figure 51 on page 82. First, the images are read from file and collected in one tuple.

```
gen_empty_obj (Images)
for J := 1 to 10 by 1
  read_image (Image, ImgPath+ImgName+J$'02')
  concat_obj (Images, Image, Images)
endfor
```

Then, the image pairs must be defined, i.e., which image should be mapped to which image.

```
  From := [1,2,3,4,6,7,8,9,3]
  To := [2,3,4,5,7,8,9,10,8]
```

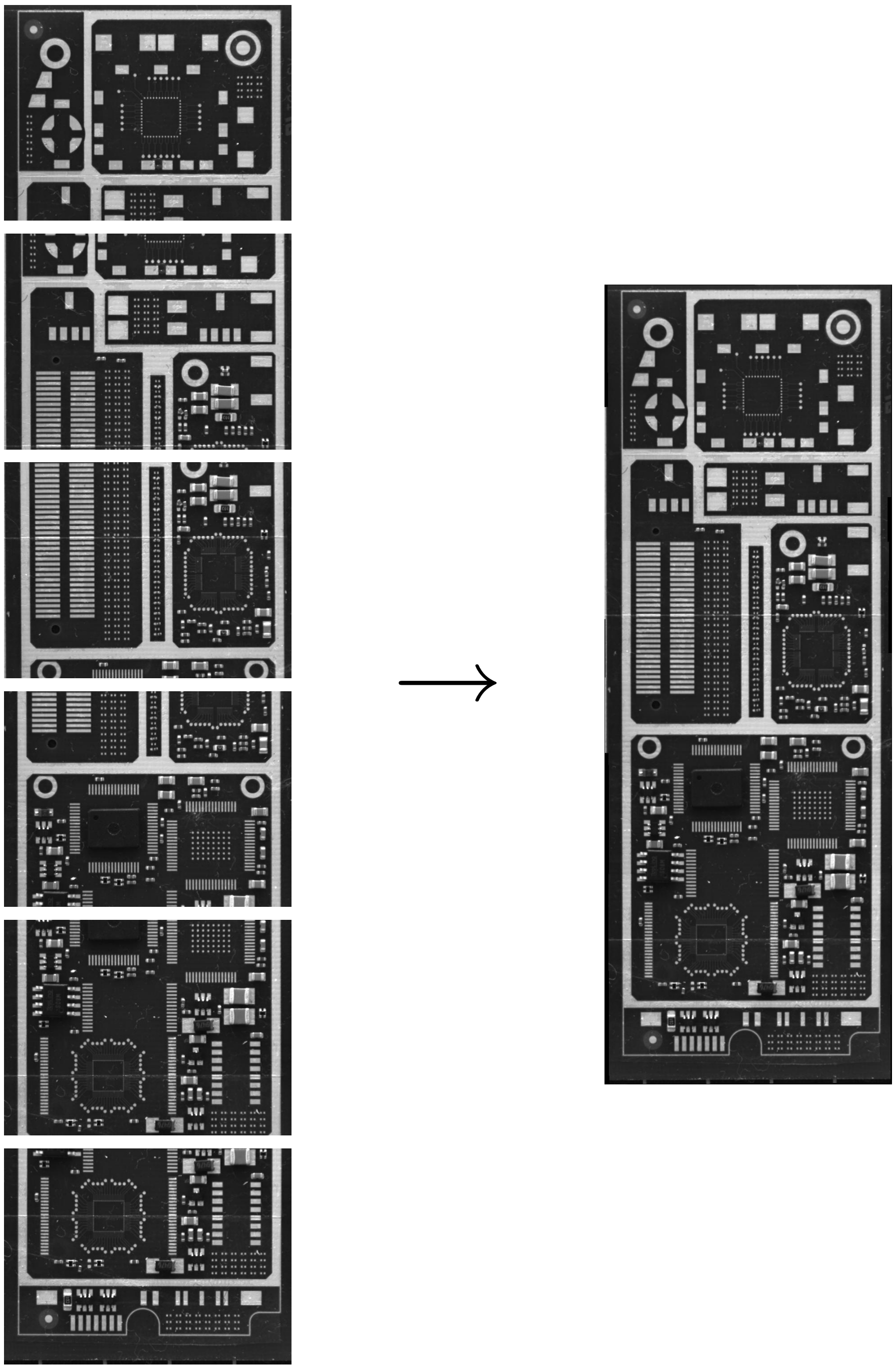


Figure 45: A first example for image mosaicking.

Now, characteristic points must be extracted from the images, which are then used for the matching between the image pairs. The resulting projective transformation matrices[1] must be accumulated.

```
Num := |From|
ProjMatrices := []
for J := 0 to Num-1 by 1
  F := From[J]
  T := To[J]
  select_obj (Images, ImageF, F)
  select_obj (Images, ImageT, T)
  points_harris (ImageF, SigmaGrad, SigmaSmooth, Alpha, Threshold, RowFAll, \
                 ColFAll)
  points_harris (ImageT, SigmaGrad, SigmaSmooth, Alpha, Threshold, RowTAll, \
                 ColTAll)
  proj_match_points_ransac (ImageF, ImageT, RowFAll, ColFAll, RowTAll, \
                            ColTAll, 'sad', MaskSize, RowMove, ColMove, \
                            RowTolerance, ColTolerance, Rotation, \
                            MatchThreshold, 'gold_standard', \
                            DistanceThreshold, RandSeed, ProjMatrix, \
                            Points1, Points2)
  ProjMatrices := [ProjMatrices,ProjMatrix]
endfor
```

Finally, the image mosaic can be generated.

```
gen_projective_mosaic (Images, MosaicImage, StartImage, From, To, \
                       ProjMatrices, StackingOrder, 'false', \
                       MosaicMatrices2D)
```

Note that image mosaicking is a tool for a quick and easy generation of large images from several overlapping images. For this task, it is not necessary to calibrate the camera. If you need a high-precision image mosaic, you should use the method described in section 4 on page 64.

In the following sections, the individual steps for the generation of a mosaic image are described.

5.1 Rules for Taking Images for a Mosaic Image

The following rules for the acquisition of the separate images should be considered:

- The images must overlap each other.
- The overlapping area of the images must be textured in order to allow the automatic matching process to identify identical points in the images. The lack of texture in some overlapping areas may be overcome by an appropriate definition of the image pairs (see section 5.2 on page 80). If the whole object shows little texture, the overlapping areas should be chosen larger.

[1] A projective transformation matrix describes a perspective projection. It consists of 3×3 values. If the last row contains the values [0,0,1], it corresponds to a homogeneous transformation matrix of HALCON and therefore describes an affine transformation.

- Overlapping images must have approximately the same scale. In general, the scale differences should not exceed 5-10 %.
- The images should be radiometrically similar, at least in the overlapping areas, as no radiometric adaption of the images is carried out. Otherwise, i.e., if the brightness differs heavily between neighboring images, the seams between them will be clearly visible as can be seen in figure 46.

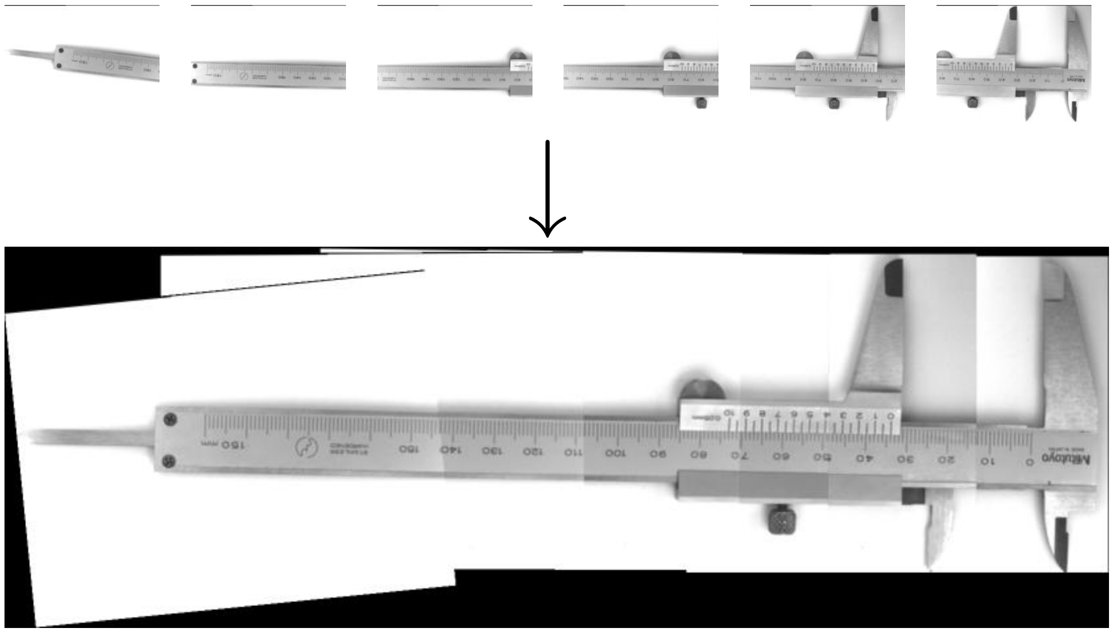


Figure 46: A second example for image mosaicking.

The images are mapped onto a common image plane using a projective transformation. Therefore, to generate a geometrically accurate image mosaic from images of non-flat objects, the separate images must be acquired from approximately the same point of view, i.e., the camera can only be rotated around its optical center (see figure 47).

When dealing with flat objects, it is possible to acquire the images from arbitrary positions and with arbitrary orientations if the scale difference between the overlapping images is not too large (figure 48).

The lens distortions of the images are not compensated by the mosaicking process. Therefore, if lens distortions are present in the images, they cannot be mosaicked with high accuracy, i.e., small distortions at the seams between neighboring images cannot be prevented (see figure 52 on page 83). To eliminate this effect, the lens distortions can be compensated before starting the mosaicking process (see section 3.3.2 on page 60).

If processing time is an issue, it is advisable to acquire the images in the same orientation, i.e., neither the camera nor the object should be rotated around the optical axis, too much. Then, the rotation range can be restricted for the matching process (see section 5.4 on page 85).

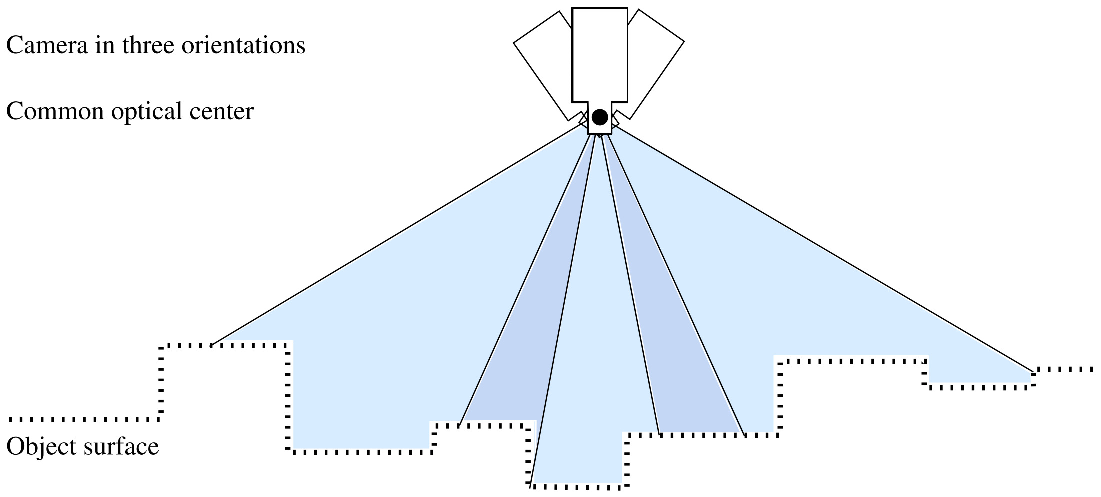


Figure 47: Image acquisition for non-flat objects.

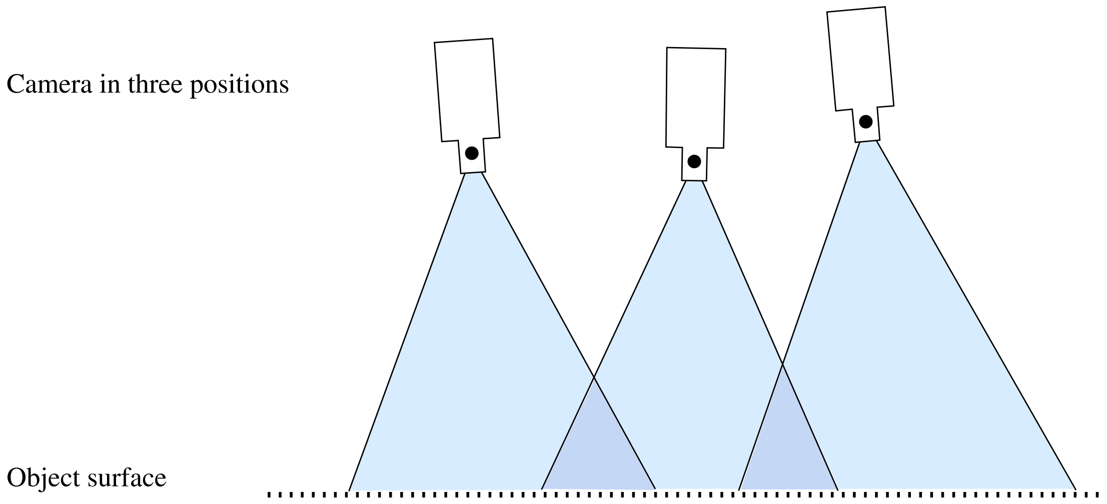


Figure 48: Image acquisition for flat objects.

5.2 Definition of Overlapping Image Pairs

As shown in the introductory example, it is necessary to define the overlapping image pairs between which the transformation is to be determined. The successive matching process will be carried out for these image pairs only.

Figure 49 shows two configurations of separate images. For configuration (a), the definition of the image pairs is simply (1,2) and (2,3), which can be defined in HDevelop as:

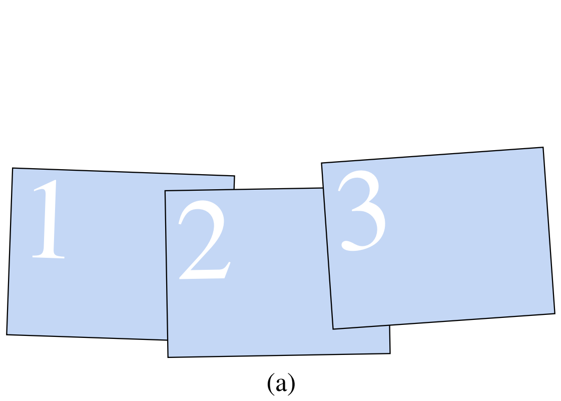


Figure 49: Two configurations of overlapping images.

```
From := [1,2]
To := [2,3]
```

In any case, it is important to ensure that each image must be “connected” to all the other images. For example, for configuration (b) of figure 49, it is not possible to define the image pairs as (1,2) and (3,4), only, because images 1 and 2 would not be connected to images 3 and 4. In this case, it would, e.g., be possible to define the three image pairs (1,2), (1,3), and (2,4):

```
From := [1,1,2]
To := [2,3,4]
```

Assuming there is no texture in the overlapping area of image two and four, the matching could be carried out between images three and four instead:

```
From := [1,1,3]
To := [2,3,4]
```

If a larger number of separate images are mosaicked, or, e.g., an image configuration similar to the one displayed in figure 50, where there are elongated rows of overlapping images, it is important to thoroughly arrange the image pair configuration. Otherwise it is possible that some images do not fit together precisely. This happens since the transformations between the images cannot be determined with perfect accuracy because of very small errors in the point coordinates due to noise. These errors are propagated from one image to the other.

Figure 51 shows such an image sequence of ten images of a BGA and the resulting mosaic image. Figure 52 shows a cut-out of that mosaic image. It depicts the seam between image 5 and image 10 for two image pair configurations, using the original images and the images where the lens distortions have been eliminated, respectively. The position of the cut-out is indicated in figure 51 by a rectangle.

First, the matching has been carried out in the two image rows separately and the two rows are connected via image pair $1 \longrightarrow 6$:

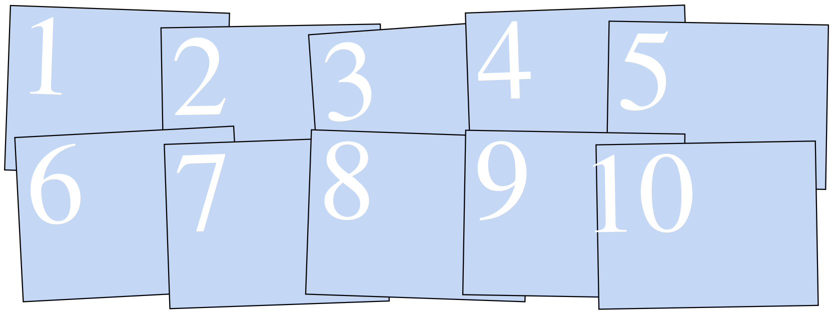


Figure 50: A configuration of ten overlapping images.

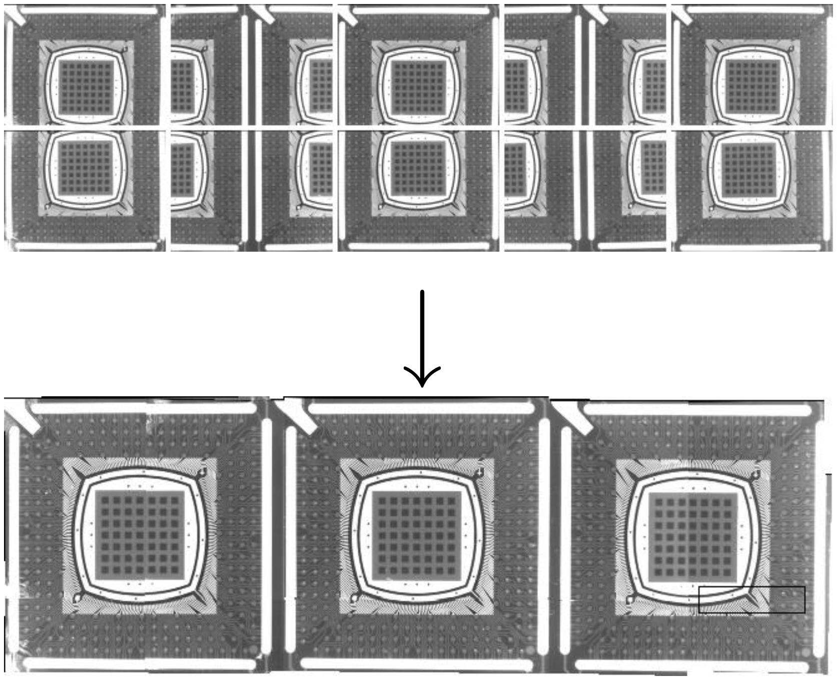


Figure 51: Ten overlapping images and the resulting (rigid) mosaic image.

```
From := [1,2,3,4,6,7,8,9,1]
To :=  [2,3,4,5,7,8,9,10,6]
```

In this configuration the two neighboring images 5 and 10 are connected along a relatively long path (figure 53).

To improve the geometrical accuracy of the image mosaic, the connections between the two image rows could be established by the image pair (3,8), as visualized in (figure 54)).

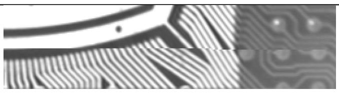
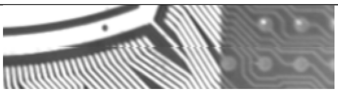
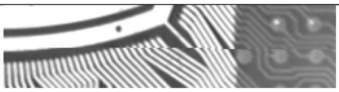
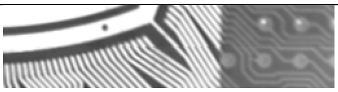
	with lens distortions	lens distortions eliminated
unfavorable configuration		
good configuration		

Figure 52: Seam between image 5 and image 10 for various configurations.

$$\begin{tikzcd}
\boxed{1} \arrow[r] \arrow[d] & \boxed{2} \arrow[r] & \boxed{3} \arrow[r] & \boxed{4} \arrow[r] & \boxed{5} \\
\boxed{6} \arrow[r] & \boxed{7} \arrow[r] & \boxed{8} \arrow[r] & \boxed{9} \arrow[r] & \boxed{10}
\end{tikzcd}$$

Figure 53: Unfavorable configuration of image pairs.

This can be achieved by defining the image pairs as follows.

```
From := [1,2,3,4,6,7,8,9,3]
To := [2,3,4,5,7,8,9,10,8]
```

As can be seen in figure 52, now the neighboring images fit better.

Recapitulating, there are three basic rules for the arrangement of the image pairs:

Take care that

1. each image is connected to all the other images.
2. the path along which neighboring images are connected is not too long.
3. the overlapping areas of image pairs are large enough and contain enough texture to ensure a proper matching.

In principle, it is also possible to define more image pairs than required (number of images minus one). However, then it cannot be controlled which pairs are actually used. Therefore, we do not recommend this approach.

$$\begin{tikzcd}
\boxed{1} & \boxed{2} \arrow[l] & \boxed{3} \arrow[l] \arrow[r] \arrow[d] & \boxed{4} \arrow[r] & \boxed{5} \\
\boxed{6} & \boxed{7} \arrow[l] & \boxed{8} \arrow[l] \arrow[r] & \boxed{9} \arrow[r] & \boxed{10}
\end{tikzcd}$$

Figure 54: Good configuration of image pairs.

5.3 Detection of Characteristic Points

HALCON provides you with various operators for the extraction of characteristic points (interest points). The most important of these operators are

- points_foerstner
- points_harris and points_harris_binomial
- points_lepetit
- points_sojka
- saddle_points_sub_pix

All of these operators can determine the coordinates of interest points with subpixel accuracy.

In figure 55, a test image together with typical results of these interest operators is displayed.

The operator points_foerstner classifies the interest points into two categories: junction-like features and area-like features. The results are very reproducible even in images taken from a different point of view. Therefore, it is very well suited for the extraction of points for the subsequent matching. It is very accurate but computationally the most expensive operator out of the interest operators presented in this section.

The results of the operator points_harris are very reproducible, too. Admittedly, the points extracted by the operator points_harris are sometimes not meaningful to a human, e.g., they often lie slightly beside a corner or an eye-catching image structure. Nevertheless, it is faster than the operator points_foerstner. The operator points_harris_binomial detects points of interest using the binomial approximation of the points_harris operator. It is therefore faster than points_harris.

The operator points_lepetit extracts points of interest like corners or blob-like structures from the image. This operator can especially be used for very fast interest point extraction. It is the fastest out of the six operators presented in this section.

The operator points_sojka is specialized in the extraction of corner points.

The operator saddle_points_sub_pix is designed especially for the extraction of saddle points, i.e., points whose image intensity is minimal along one direction and maximal along a different direction.

The number of interest points influence the execution time and the result of the subsequent matching process. The more interest points are used, the longer the matching takes. If too few points are used the probability of an erroneous result increases.

In most cases, the default parameters of the interest operators need not be changed. Only if too many or too few interest points are found adaptations of the parameters might be necessary. For a description of the parameters, please refer to the respective pages of the reference manual (points_foerstner, points_harris, points_harris_binomial, points_lepetit, points_sojka, saddle_points_sub_pix).

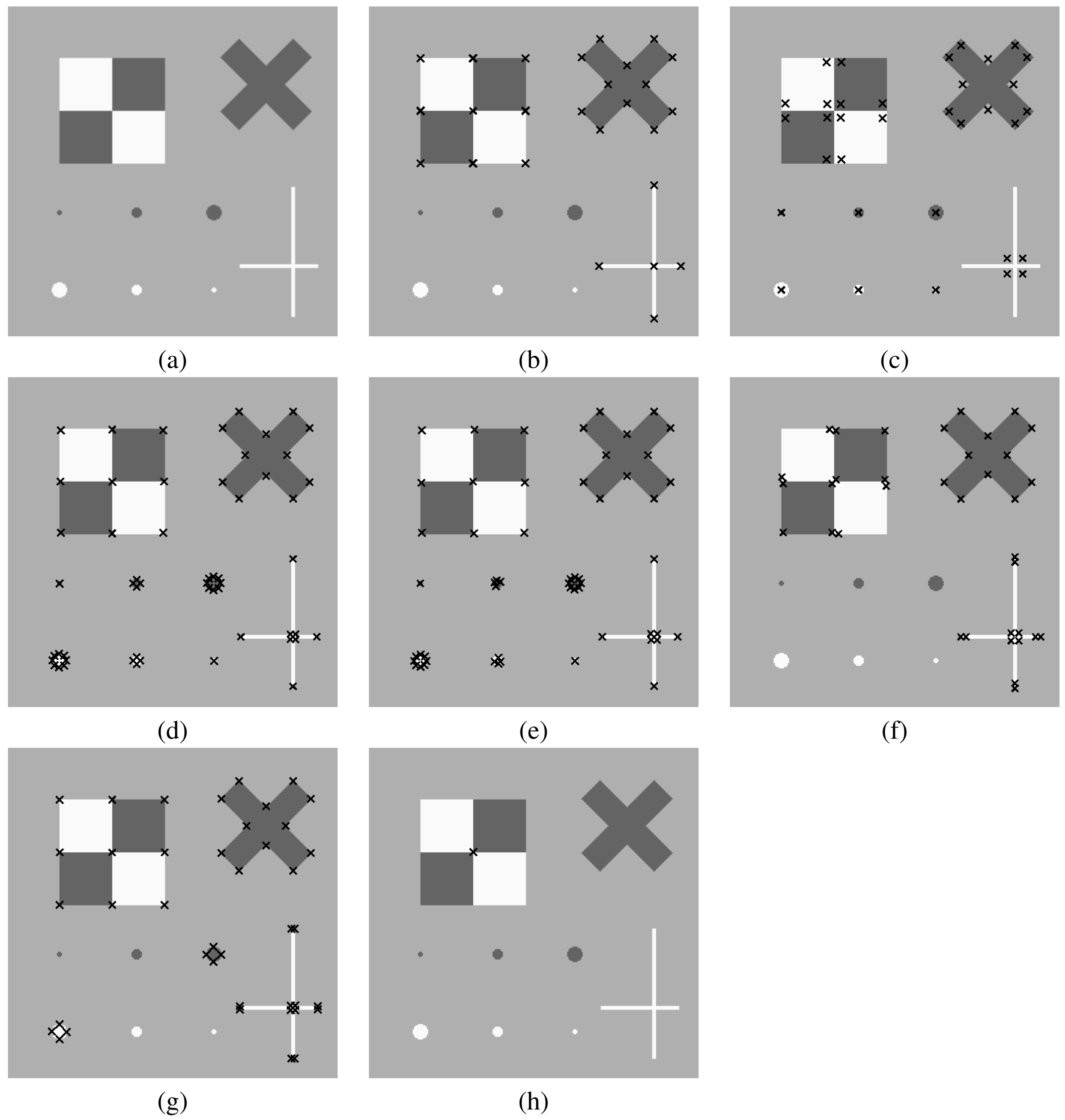


Figure 55: Comparison of typical results of interest operators. a) Test image; b) Förstner, junctions; c) Förstner, area; d) Harris; e) Harris, binomial f) Lepetit; g) Sojka; h) Saddle points.

5.4 Matching of Characteristic Points in Overlapping Areas and Determination of the Transformation between the Images

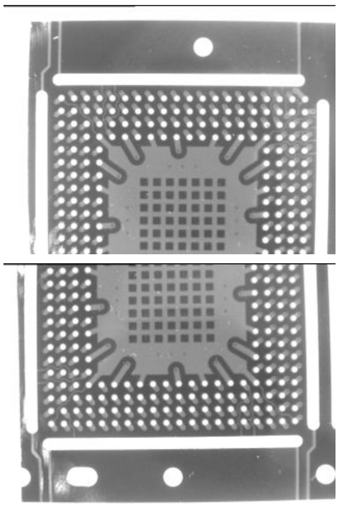
The most demanding task during the generation of an image mosaic is the matching process. The operator `proj_match_points_ransac` is able to perform the matching even if the two images are shifted and rotated arbitrarily.

```
proj_match_points_ransac (ImageF, ImageT, RowFAll, ColFAll, RowTAll, \
                          ColTAll, 'sad', MaskSize, RowMove, ColMove, \
                          RowTolerance, ColTolerance, Rotation, \
                          MatchThreshold, 'gold_standard', \
                          DistanceThreshold, RandSeed, ProjMatrix, Points1, \
                          Points2)
```

The only requirement is that the images should have approximately the same scale. If information about shift and rotation is available it can be used to restrict the search space, which speeds up the matching process and makes it more robust.

In case the matching fails, ensure that there are enough characteristic points and that the search space and the maximum rotation are defined appropriately.

If the images that should be mosaicked contain repetitive patterns, like the two images of a BGA shown in figure 56a, it may happen that the matching does not work correctly. In the resulting erroneous mosaic image, the separate images may not fit together or may be heavily distorted. To achieve a correct matching result for such images, it is important to provide initial values for the shift between the images with the parameters `RowMove` and `ColMove`. In addition, the search space should be restricted to an area that contains only one instance of the repetitive pattern, i.e., the values of the parameters `RowTolerance` and `ColTolerance` should be chosen smaller than the distance between the instances of the repetitive pattern. With this, it is possible to obtain proper mosaic images, even for objects like BGAs (see figure 56b).


(a)

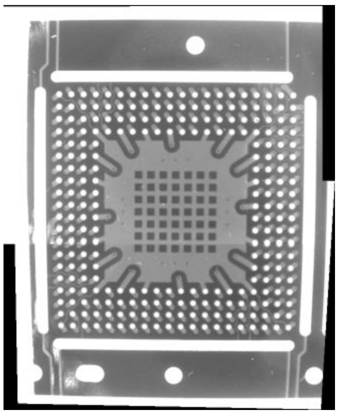

(b)

Figure 56: Separate images (a) and mosaic image (b) of a BGA.

For a detailed description of the other parameters, please refer to the reference manual (`proj_match_points_ransac`).

The results of the operator `proj_match_points_ransac` are the projective transformation matrix and the two tuples Points1 and Points2 that contain the indices of the matched input points from the two images.

The projective transformation matrices resulting from the matching between the image pairs must be accumulated.

```
ProjMatrices := [ProjMatrices,ProjMatrix]
```

Alternatively, if it is known that the mapping between the images is a rigid 2D transformation, the operator proj_match_points_ransac can be used to determine the point correspondences only, since it returns the indices of the corresponding points in the tuples Points1 and Points2. With this, the corresponding point coordinates can be selected.

```
RowF := subset(RowFAll,Points1)
ColF := subset(ColFAll,Points1)
RowT := subset(RowTAll,Points2)
ColT := subset(ColTAll,Points2)
```

Then, the rigid transformation between the image pair can be determined with the operator vector_to_rigid. Note that we have to add *0.5* to the coordinates to make the extracted pixel positions fit the coordinate system that is used by the operator gen_projective_mosaic.

```
vector_to_rigid (RowF+0.5, ColF+0.5, RowT+0.5, ColT+0.5, HomMat2D)
```

Because gen_projective_mosaic expects a 3×3 transformation matrix, but vector_to_rigid returns a 2×3 matrix, we have to add the last row *[0,0,1]* to the transformation matrix before we can accumulate it.

```
ProjMatrix := [HomMat2D,0,0,1]
ProjMatrices := [ProjMatrices,ProjMatrix]
```

Furthermore, the operator proj_match_points_ransac_guided is available. Like proj_match_points_ransac, it can be used to calculate the projective transformation matrix between two images by finding correspondences between points. But in contrast to proj_match_points_ransac, it is based on a known approximation of the projective transformation matrix. Thus, it can be used, for example, to speed up the matching of very large images by implementing an image-pyramid-based projective matching algorithm. The HDevelop example program examples\hdevelop\Tools\2D-Transformations\mosaicking_pyramid.dev shows how to implement the image-pyramid-based approach and compares the runtime for different numbers of pyramid levels.

5.5 Generation of the Mosaic Image

Once the transformations between the image pairs are known the mosaic image can be generated with the operator gen_projective_mosaic.

```
gen_projective_mosaic (Images, MosaicImage, StartImage, From, To, \
                       ProjMatrices, StackingOrder, 'false', \
                       MosaicMatrices2D)
```

It requires the images to be given in a tuple. All images are projected into the image plane of a so-called start image. The start image can be defined by its position in the image tuple (starting with *1*) with the parameter StartImage.

Additionally, the image pairs must be specified together with the corresponding transformation matrices.

The order in which the images are added to the mosaic image can be specified with the parameter StackingOrder. The first index in this array will end up at the bottom of the image stack while the last one will be on top. If *'default'* is given instead of an array of integers, the canonical order (the order in which the images are given) will be used.

If the domains of the images should be transformed as well, the parameter TransformRegion must be set to *'true'*.

The output parameter MosaicMatrices2D contains the projective 3×3 transformation matrices for the mapping of the separate images into the mosaic image. These matrices can, e.g., be used to transform features extracted from the separate images into the mosaic image by using the operators projective_trans_pixel, projective_trans_region, projective_trans_contour_xld, or projective_trans_image.

5.6 Bundle Adjusted Mosaicking

It is also possible to generate the mosaic based on the matching results of all overlapping image pairs. The transformation matrices between the images are then determined together within one bundle adjustment. For this, the operators bundle_adjust_mosaic and gen_bundle_adjusted_mosaic are used.

The main advantage of the bundle adjusted mosaicking compared with the mosaicking based on single image pairs is that the bundle adjustment determines the geometry of the mosaic as robustly as possible. Typically, this leads to more accurate results. Another advantage is that there is no need to figure out a good pair configuration, you simply pass the matching results of all overlapping image pairs. What is more, it is possible to define the class of transformations that is used for the transformation between the individual images. A disadvantage of the bundle adjusted mosaicking is that it takes more time to perform the matching between all overlapping image pairs instead of just using a subset. Furthermore, if the matching between two images was erroneous, sometimes the respective image pair is difficult to find in the set of all image pairs.

With this, it is obvious that the bundle adjustment is worthwhile if there are multiple overlaps between the images, i.e., if there are more than $n - 1$ overlapping image pairs, with n being the number of images. Another reason for using the bundle adjusted mosaicking is the possibility to define the class of transformations.

The example program examples\solution_guide\3d_machine_vision\bundle_adjusted_mosaicking.dev shows how to generate the bundle adjusted mosaic from the ten images of the BGA displayed in figure 51 on page 82. The design of the program is very similar to that of the example program examples\solution_guide\3d_machine_vision\mosaicking.dev, which is described in the introduction of section 5 on page 76. The main differences are that

- the matching is carried out between all overlapping images,
- in addition to the projective transformation matrices also the coordinates of the corresponding points must be accumulated, and
- the operator gen_projective_mosaic is replaced with the operators bundle_adjust_mosaic and gen_bundle_adjusted_mosaic.

First, the matching is carried out between all overlapping image pairs, which can be defined as follows:

```
From := [1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5]
To := [6,7,2,6,7,8,3,7,8,9,4,8,9,10,5,9,10]
```

In addition to the accumulation of the projective transformation matrices, as described in section 5.4 on page 85, also the coordinates of the corresponding points as well as the number of corresponding points must be accumulated.

```
Rows1 := [Rows1,subset(RowFAll,Points1)]
Cols1 := [Cols1,subset(ColFAll,Points1)]
Rows2 := [Rows2,subset(RowTAll,Points2)]
Cols2 := [Cols2,subset(ColTAll,Points2)]
NumCorrespondences := [NumCorrespondences,|Points1|]
```

This data is needed by the operator bundle_adjust_mosaic, which determines the bundle adjusted transformation matrices.

```
bundle_adjust_mosaic (10, StartImage, From, To, ProjMatrices, Rows1, Cols1, \
                      Rows2, Cols2, NumCorrespondences, Transformation, \
                      MosaicMatrices2D, Rows, Cols, Error)
```

The parameter Transformation defines the class of transformations that is used for the transformation between the individual images. Possible values are *'projective'*, *'affine'*, *'similarity'*, and *'rigid'*. Thus, if you know, e.g., that the camera looks perpendicular onto a planar object and that the camera movement between the images is restricted to rotations and translations in the object plane, you can choose the transformation class *'rigid'*. If translations may also occur in the direction perpendicular to the object plane, you must use *'similarity'* because this transformation class allows scale differences between the images. If the camera looks tilted onto the object, the transformation class *'projective'* must be used, which can be approximated by the transformation class *'affine'*. Figure 57 shows cut-outs of the resulting mosaic images. They depict the seam between image 5 and image 10. The mosaic images have been created using the images where the lens distortions have been eliminated. The position of the cut-out within the whole mosaic image is indicated by the rectangle in figure 51 on page 82.

Finally, with the transformation matrices MosaicMatrices2D, which are determined by the operator bundle_adjust_mosaic, the mosaic can be generated with the operator gen_bundle_adjusted_mosaic.

```
gen_bundle_adjusted_mosaic (Images, MosaicImage, MosaicMatrices2D, \
                            StackingOrder, TransformRegion, TransMat2D)
```

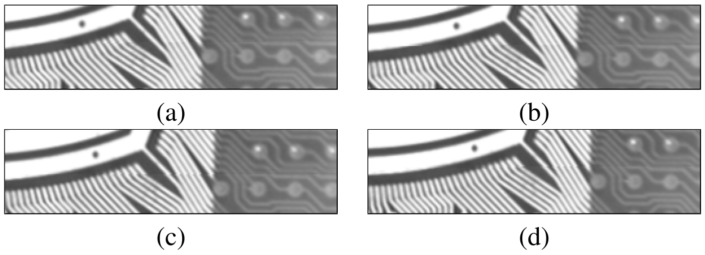


Figure 57: Seam between image 5 and image 10 for different classes of transformations: (a) projective, (b) affine, (c) similarity, and (d) rigid.

5.7 Spherical Mosaicking

The methods described in the previous sections arranged the images on a plane. As the name suggests, using spherical mosaicking you can arrange them on a sphere instead. Note that this method can only be used if the camera is only rotated around its optical center or zoomed. If the camera movement includes a translation or if the rotation is not performed exactly around the optical center, the resulting mosaic image will not be accurate and can therefore not be used for high-accuracy applications.

To create a spherical mosaic, you first perform the matching as described in the previous sections to determine the projective transformation matrices between the individual images. This information is the input for the operator `stationary_camera_self_calibration`, which determines the internal camera parameters of the camera and the rotation matrices for each image. Based on this information, the operator `gen_spherical_mosaic` then creates the mosaic image. Please have a look at the HDevelop example program `hdevelop\Tools\Calibration\stationary_camera_self_calibration.dev` for more information about how to use these operators.

As an alternative, you can map the images on the six sides of a cube using `gen_cube_map_mosaic`. Cube maps are especially useful in computer graphics.

6 Pose Estimation of Known 3D Objects With a Single Camera

Estimating the 3D pose of an object is an important task in many application areas, e.g., during completeness checks or for 3D alignment in robot vision applications (see section 8.4.1 on page 131). HALCON provides multiple methods to determine the pose of known 3D objects with a single camera. The most powerful one is 3D matching (see the Solution Guide I, chapter 10 on page 125), which performs a full object recognition, i.e., it not only estimates the pose but first locates the object in the image.

However, for many applications 3D matching is "too powerful", in particular if object recognition is not necessary because you can locate the object with simpler means, or if you do not have a CAD model of the object. For these applications, HALCON offers different methods, which are optimized for different object shapes:

- **arbitrarily shaped objects**: section 6.1

- **3D circles**: section 6.2 on page 100
- **3D rectangles**: section 6.3 on page 100

An example application for pose estimation in a robot vision system is described in section 8.4.3 on page 133.

6.1 Pose Estimation for General 3D Objects

For arbitrarily shaped 3D objects, different approaches are available.

- First we describe how to generally determine the pose of a known 3D object by applying a camera calibration, by manually extracting corresponding points for the object of interest, and by applying several transformations (section 6.1.1).
- Then, two types of template matching are introduced that can be used for a 3D pose estimation. Both can be applied to planar objects that are arbitrarily positioned in the 3D space.
  - The calibrated perspective, deformable matching determines the 3D pose of a planar object that was defined by a template object by using automatically derived contours of the object (section 6.1.2 on page 95).
  - The calibrated descriptor-based matching determines the 3D pose of a planar object that was defined by a template object by using automatically derived distinctive points of the object, which are called interest points (section 6.1.3 on page 98).

6.1.1 Pose Estimation Using Manually Extracted Correspondences

Here, the basic idea is that the pose of the object, i.e., the external camera parameters with respect to the object, can be determined by a call of the operator `camera_calibration`.

The individual steps are illustrated based on the example program `examples\solution_guide\3d_machine_vision\pose_of_known_3d_object.dev`, which determines the pose of a metal part with respect to a given world coordinate system.

First, the camera must be calibrated, i.e., the internal camera parameters and, if the pose of the object is to be determined relative to a given world coordinate system, the external camera parameters must be determined. See section 3.1 on page 30 for a detailed description of the calibration process. The world coordinate system can either be identical to the calibration plate coordinate system belonging to the calibration plate from one of the calibration images, or it can be modified such that it fits to some given reference coordinate system (figure 58). This can be achieved, e.g., by using the operator `set_origin_pose`

```
set_origin_pose (PoseOfWCS, -0.0568, 0.0372, 0, PoseOfWCS)
```

or if other transformations than translations are necessary, via homogeneous transformation matrices (section 2.1 on page 8).

```
pose_to_hom_mat3d (PoseOfWCS, camHwcs)
hom_mat3d_rotate_local (camHwcs, rad(180), 'x', camHwcs)
hom_mat3d_to_pose (camHwcs, PoseOfWCS)
```

With the homogeneous transformation matrix ${}^{c}\mathbf{H}_{w}$, which corresponds to the pose of the world coordinate system, world coordinates can be transformed into camera coordinates.

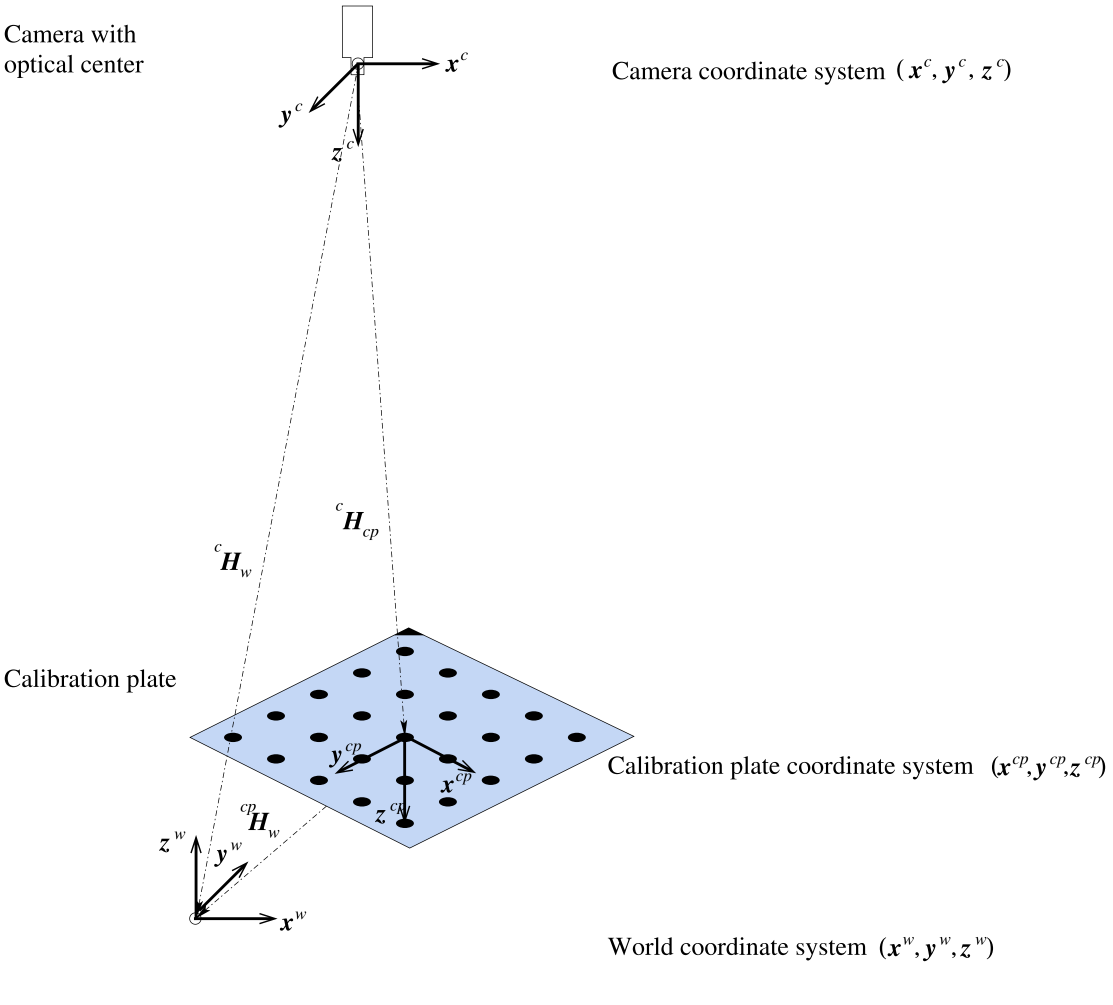

Camera with optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
${}^{c}H_{cp}$
${}^{c}H_{w}$
Calibration plate
y^{cp}
x^{cp}
z^{cp}
Calibration plate coordinate system (x^{cp}, y^{cp}, z^{cp})
${}^{cp}H_{w}$
z^w
y^w
x^w
World coordinate system (x^w, y^w, z^w)

Figure 58: Determination of the pose of the world coordinate system.

Then, the pose of the object can be determined from at least three points (control points) for which both the 3D object coordinates and the 2D image coordinates are known.

The 3D coordinates of the control points need to be determined only once. They must be given in a coordinate system that is attached to the object. You should choose points that can be extracted easily and accurately from the images. The 3D coordinates of the control points are then stored in three tuples, one for the x coordinates, one for the y coordinates, and the last one for the z coordinates.

In each image from which the pose of the object should be determined, the control points must be extracted. This task depends heavily on the object and on the possible poses of the object. If it is known that the object will not be tilted with respect to the camera the detection can, e.g., be carried out by shape-based matching (for a detailed description of shape-based matching, please refer to the Solution Guide II-B).

Once the image coordinates of the control points are determined, they must be stored in two tuples that contain the row and the column coordinates, respectively. Note that the 2D image coordinates of the control points must be stored in the same order as the 3D coordinates.

In the example program, the centers of the three holes of the metal part are used as control points. Their image coordinates are determined with the HDevelop procedure

```
procedure determine_control_points (Image: : : RowCenter, ColCenter)
```

which is part of the example program `examples\solution_guide\3d_machine_vision\pose_of_known_3d_object.dev`.

Now, the operator `camera_calibration` can be used to determine the pose of the object. For this, the 3D object coordinates and the 2D image coordinates of the control points must be passed to the operator `camera_calibration` via the parameters `NX`, `NY`, and `NZ` (3D object coordinates) as well as `NRow` and `NCol` (2D image coordinates). The known internal camera parameters are given via the parameter `StartCamParam` and an initial pose of the object with respect to the camera must be specified by the parameter `NStartPose`. The parameter `EstimateParams` must be set to *'pose'*.

```
camera_calibration (ControlX, ControlY, ControlZ, RowCenter, ColCenter, \
                    CamParam, StartPose, 'pose', _, PoseOfObject, Errors)
```

You can determine the initial pose of the object from one image where the object is in a typical position and orientation. Place the HALCON calibration plate on the object and apply the operators `find_caltab` and `find_marks_and_pose` (see section 3.1.1 on page 32). The resulting estimation for the pose can be used as initial pose, even for images where the object appears slightly shifted or rotated. In the example program, the (initial) pose of the calibration plate that was used for the definition of the WCS is used as initial pose of the object.

If both the pose of the world coordinate system and the pose of the object coordinate system are known with respect to the camera coordinate system (see figure 59), it is easy to determine the transformation matrices for the transformation of object coordinates into world coordinates and vice versa:

$$\begin{aligned} {}^{w}\mathbf{H}_{o} &= {}^{w}\mathbf{H}_{c} \cdot {}^{c}\mathbf{H}_{o} && (37)\\ &= ({}^{c}\mathbf{H}_{w})^{-1} \cdot {}^{c}\mathbf{H}_{o} && (38)\end{aligned}$$

where ${}^{w}\mathbf{H}_{o}$ is the homogeneous transformation matrix for the transformation of object coordinates into world coordinates and ${}^{c}\mathbf{H}_{w}$ and ${}^{c}\mathbf{H}_{o}$ are the homogeneous transformation matrices corresponding to the pose of the world coordinate system and the pose of the object coordinate system, respectively, each with respect to the camera coordinate system.

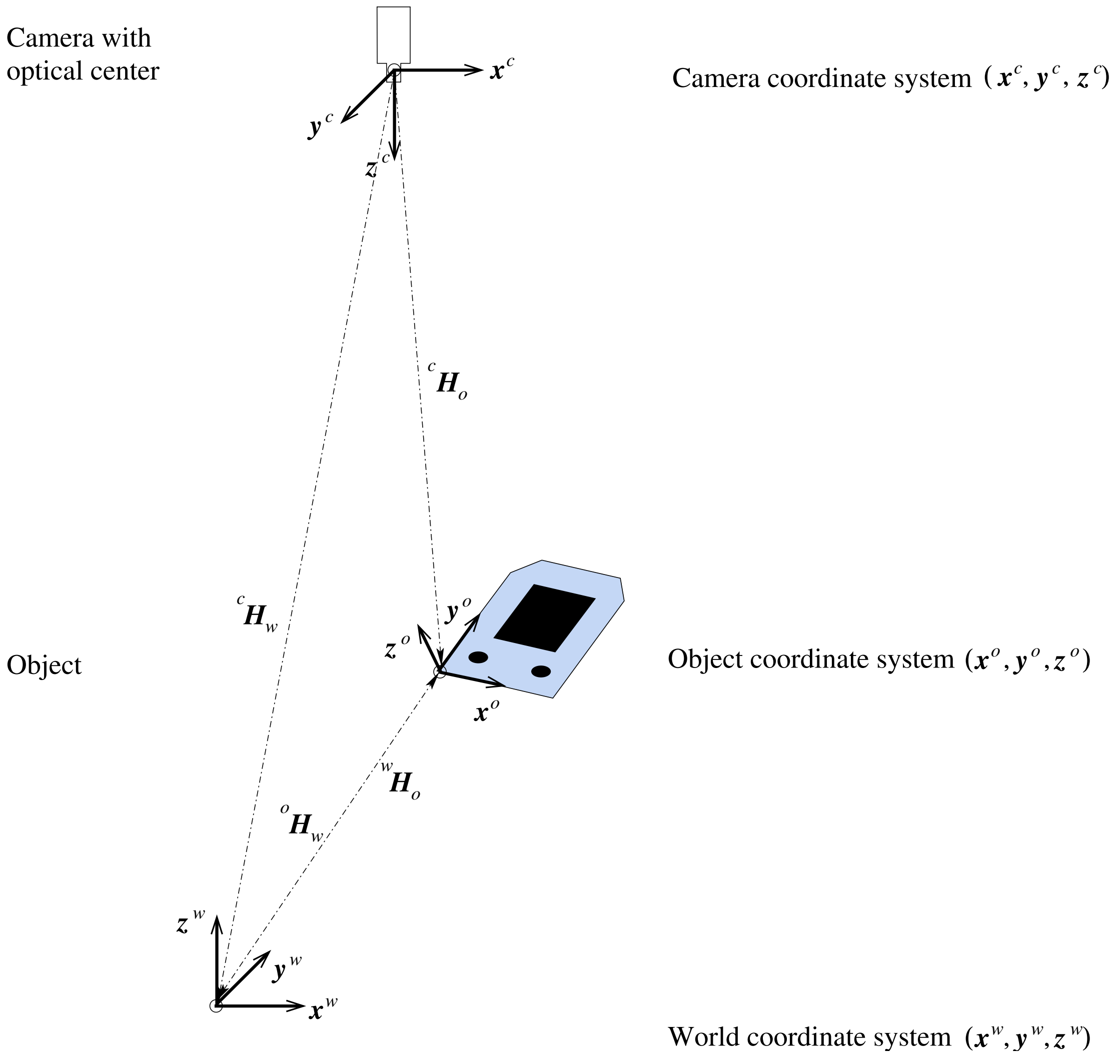

Camera with
optical center
x^c
y^c
z^c
Camera coordinate system (x^c, y^c, z^c)
${}^cH_o$
${}^cH_w$
y^o
z^o
x^o
Object
Object coordinate system (x^o, y^o, z^o)
${}^wH_o$
${}^oH_w$
z^w
y^w
x^w
World coordinate system (x^w, y^w, z^w)

Figure 59: Pose of the object coordinate system and transformation between object coordinates and world coordinates.

The transformation matrix for the transformation of world coordinates into object coordinates can be derived by:

$$ {}^{o}\mathbf{H}_{w} \quad = \quad ({}^{w}\mathbf{H}_{o})^{-1} \tag{39} $$

The calculations described above can be implemented in HDevelop as follows. First, the homogeneous transformation matrices are derived from the respective poses.

```
pose_to_hom_mat3d (PoseOfWCS, camHwcs)
pose_to_hom_mat3d (PoseOfObject, camHobj)
```

Then, the transformation matrix for the transformation of object coordinates into world coordinates is derived.

```
hom_mat3d_invert (camHwcs, wcsHcam)
hom_mat3d_compose (wcsHcam, camHobj, wcsHobj)
```

Now, known object coordinates can be transformed into world coordinates with:

```
affine_trans_point_3d (wcsHobj, CornersXObj, CornersYObj, CornersZObj, \
                       CornersXWCS, CornersYWCS, CornersZWCS)
```

In the example program examples\solution_guide\3d_machine_vision\pose_of_known_3d_object.dev, the world coordinates of the four corners of the rectangular hole of the metal part are determined from their respective object coordinates. The object coordinate system and the world coordinate system are visualized as well as the respective coordinates for the four points (see figure 60).

The following approaches also need a calibrated camera for the estimation of the 3D pose, but it is not necessary to manually determine corresponding points for the object.

6.1.2 Pose Estimation using Calibrated Perspective, Deformable Matching

The perspective, deformable matching finds and locates objects that are similar to a template model in an image. This kind of template matching uses the contours of the object in the images and is independent of the perspective view on the object, i.e., perspective deformations are considered when searching the model in unknown images. The perspective, deformable matching can be applied either for a calibrated camera, then the 3D pose of the object is returned, or for an uncalibrated camera, then only the 2D projective transformation matrix (homography) is returned. Here, the focus is on the calibrated matching. For the uncalibrated matching, we refer to the Solution Guide I, chapter 9 on page 95.

The main idea is to image a template object, derive a metric description of it, store this metric model in a handle, and then locate the model independently of perspective deformations in unknown images. The main proceeding of the matching and many of the parameters that can be adjusted are similar to those used for shape-based matching (see Solution Guide II-B). In particular, a model that consists of contours is created from a template image using `create_planar_calib_deformable_model` or read from file with `read_deformable_model`. Suggestions for suitable parameter settings can be queried with `determine_deformable_model_params` before creating the model. After creating the model, additional parameters can be set via `set_deformable_model_param`. The operator `find_planar_calib_deformable_model` is then used to determine the 3D poses of perspectively deformed representations of the model in other images. Finally, the model is cleared from memory using `clear_deformable_model`.

In contrast to a shape-based matching, for the calibrated perspective, deformable matching a reference pose for the template model and the internal camera parameters have to be specified so that the 3D

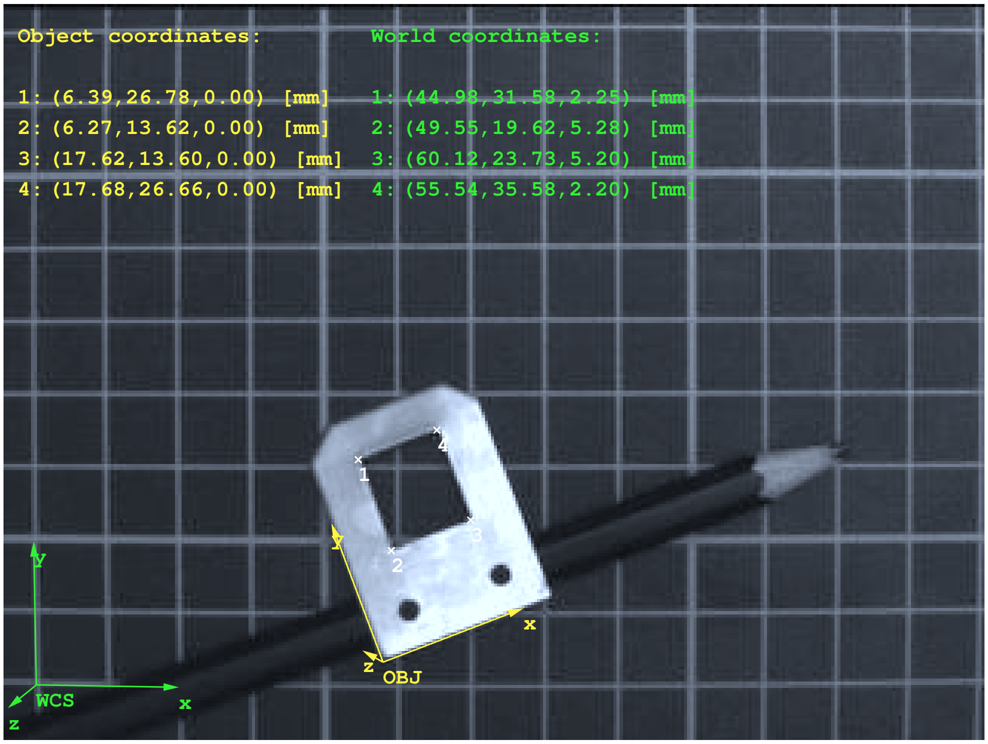

Object coordinates:
1: (6.39,26.78,0.00) [mm]
2: (6.27,13.62,0.00) [mm]
3: (17.62,13.60,0.00) [mm]
4: (17.68,26.66,0.00) [mm]
World coordinates:
1: (44.98,31.58,2.25) [mm]
2: (49.55,19.62,5.28) [mm]
3: (60.12,23.73,5.20) [mm]
4: (55.54,35.58,2.20) [mm]
1
2
3
4
y
x
z
OBJ
y
x
z
WCS

Figure 60: Object coordinates and world coordinates for the four corners of the rectangular hole of the metal part.

poses of the found instances in unknown images can be returned. The reference pose can be obtained by different means. You can, e.g., manually measure the extents of the model. This approach is rather intricate and often inaccurate. A more convenient approach to measure the pose of the template object is explained in the previous section, i.e., by placing a calibration plate onto the object plane and then calling `find_caltab` and `find_marks_and_pose` (then additionally the internal camera parameters are obtained). Other approaches for determining the pose of the object plane comprise binocular stereo vision (section 7 on page 100) or 3D laser triangulation, e.g., using sheet of light (section 10 on page 149).

When creating the template model with `create_planar_calib_deformable_model`, the reference pose is automatically modified by an offset so that its origin corresponds to the projection of the center of gravity of a rectified version of the template image onto the reference plane and the axes of the obtained model coordinate system are parallel to those of the initial reference pose (see figure 61). Note that you can additionally apply an offset to the origin of the thus obtained model pose using `set_deformable_model_origin`. This offset is set in image coordinates for the internally rectified template image. To determine the neccessary number of pixels for the row and column direction it is convenient to query the rectified contour of the object from the template model with `get_deformable_model_contours`. After setting a manual offset, the obtained image point is automatically projected to the object plane as origin of the modified pose and the pose is adapted accordingly for all following operations. Setting a new origin may be suitable if the origin should be placed on a specific part of the object, e.g., a corner of the object or a drill hole.

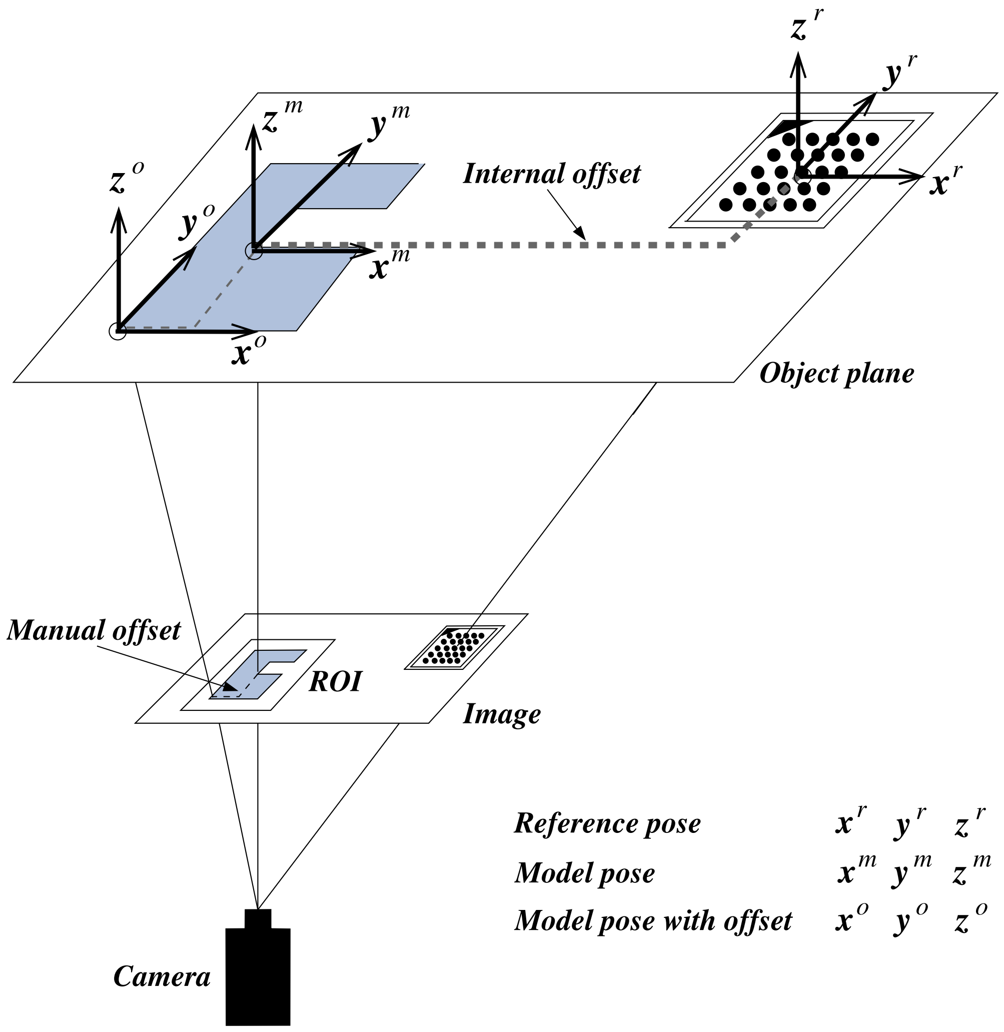

z^r
y^r
x^r
z^m
y^m
x^m
z^o
y^o
x^o
Internal offset
Object plane
Manual offset
ROI
Image
Reference pose x^r y^r z^r
Model pose x^m y^m z^m
Model pose with offset x^o y^o z^o
Camera

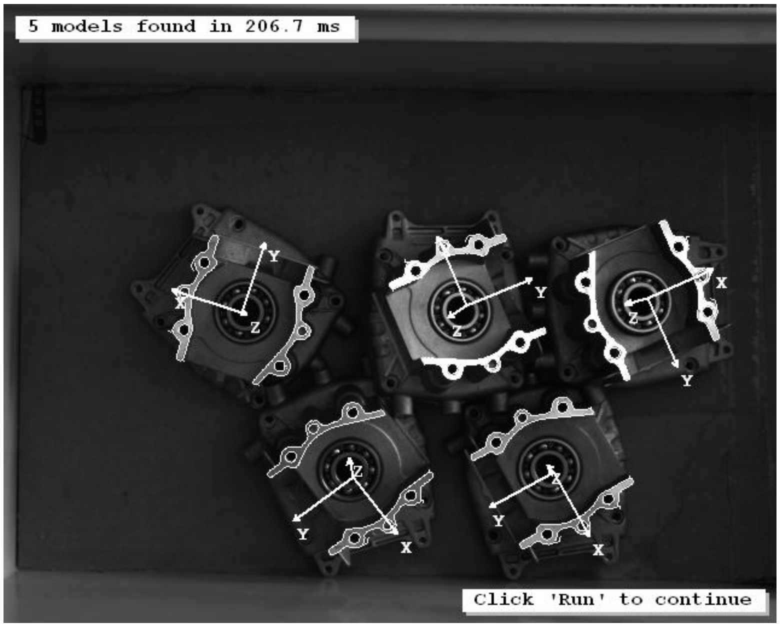
Figure 61: Internal and manual offset: the offset from the reference pose to the model pose is calculated automatically from the reference pose, the internal camera parameters, and the template image. An additional offset can be applied manually and is defined in image coordinates.

The perspective, deformable matching is suitable for all planar objects or object parts that are clearly distinguishable by their contours. Compared to the 3D matching (see the Solution Guide I, chapter 10 on page 125), there is no need to pregenerate different views of an object. Thus, it is significantly faster. Hence, if you search for planar perspectively deformed objects, we recommend the perspective, deformable matching.

The HDevelop examples `\examples\hdevelop\Applications\3D-Vision\locate_engine_parts.dev` and `\examples\hdevelop\Applications\3D-Vision\`

locate_car_door.dev show in detail how to apply the approach.

Figure 62 shows the poses of engine parts obtained by the first example.


5 models found in 206.7 ms

Click 'Run' to continue

Figure 62: 3D poses of engine parts obtained by a calibrated perspective, deformable matching.

6.1.3 Pose Estimation using Calibrated Descriptor-Based Matching

Similar to the perspective, deformable matching, the descriptor-based matching finds and locates objects that are similar to a template model in an image. Again, the matching can be applied either for a calibrated camera, then the 3D pose of an object is returned, or for an uncalibrated camera, then only the 2D projective transformation matrix (homography) is returned. Here, the focus is on the calibrated matching. For the uncalibrated case, we refer to the Solution Guide I, chapter 9 on page 95. Note that the reference pose needed for the template model can be obtained similar to the perspective, deformable matching.

The essential difference between the descriptor-based and the perspective, deformable matching is that the descriptor-based matching is not based on contours but uses so-called interest points to describe the template and to find the model in the image. These are first extracted by a detector and then are described by a descriptor. When creating the template with create_calib_descriptor_model, a detector has to be selected and parameters for the descriptor can be set.

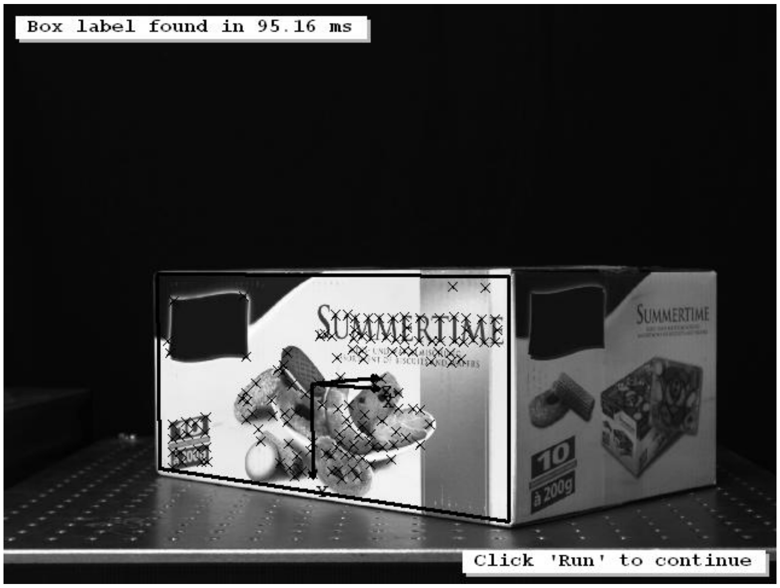
Available detectors are 'lepetit', 'harris', and 'harris_binomial', which correspond to the HALCON point operators points_lepetit, points_harris, and points_harris_binomial. We recommend to apply tests with the selected point operator before creating the template model. When visualizing the returned points using gen_cross_contour_xld, about 50 to 450 points should be uniformly distributed within the region of interest. If you have found the appropriate parameter setting for the selected point operator, you can set these parameters also for the template model in `create_calib_descriptor_model`.

The currently implemented descriptor uses randomized ferns to classify the extracted points, i.e., to build characteristic descriptions of the gray value neighborhood for the interest points. A set of parameters for the descriptor can be adjusted also when creating the model.

To locate the same interest points in unknown images of the same or a similar object, the operator `find_calib_descriptor_model` is called. Finally, the model is cleared from memory using `clear_descriptor_model`.

Note that the calibrated descriptor-based matching is suitable mainly to determine the 3D pose of planar objects with characteristic texture and distinctive object points. For low-textured objects with rounded edges you should select one of the other pose estimation approaches. Further, the descriptor-based matching is less accurate than the perspective, deformable matching. But on the other hand, it is significantly faster if a large search space, e.g., caused by a large scale range, is used.

The HDevelop examples `\examples\hdevelop\Applications\Packaging\locate_cookie_box.dev` and `\examples\hdevelop\Applications\Packaging\locate_cookie_box_multiple_models.dev` show in detail how to apply the approach. Figure 63 shows the 3D pose of a cookie box obtained by the first example. A comparison between the calibrated descriptor-based matching and pose estimation methods that use manually extracted correspondences is provided by the HDevelop example `\examples\hdevelop\Matching\Descriptor-based\pose_from_point_correspondences.dev`.


Box label found in 95.16 ms

Click 'Run' to continue

Figure 63: 3D pose of a cookie box label obtained by a calibrated descriptor-based matching.

6.2 Pose Estimation for 3D Circles

HALCON offers an alternative approach to estimate the pose of 3D circles, which can be applied with less effort than the general approach. It is based on the known geometrical behavior of perspectively distorted circles. In particular, 3D circles are usually represented as ellipses in the image. Using the extracted 2D ellipse of a 3D circle together with the internal camera parameters and the known radius of the circle, the two possible 3D poses of the circle (having the same position but opposite orientations) can be obtained easily using the operator `get_circle_pose`. The HDevelop example `\examples\hdevelop\Applications\Calibration\3d_position_of_circles.dev` shows in detail how to apply the approach.

6.3 Pose Estimation for 3D Rectangles

Additionally to the pose estimation for 3D circles, also the poses of 3D rectangles can be estimated with an approach that can be applied with less effort than the general approach. It is based on the known geometrical behavior of perspectively distorted rectangles. In particular, a contour is segmented into four line segments and their intersections are considered as the corners of a quadrangular contour. Using the extracted 2D quadrangle of the 3D rectangle together with the internal camera parameters and the known size of the rectangle, the four (or in case of a square eight) possible 3D poses of the rectangle can be obtained easily using the operator `get_rectangle_pose`. The HDevelop examples `\examples\hdevelop\Applications\Calibration\get_rectangle_pose_barcode.dev` and `\examples\hdevelop\Applications\Calibration\get_rectangle_pose_work_sheet.dev` show in detail how to apply the approach.

7 3D Vision With a Binocular Stereo System

With a binocular stereo system, it is possible to derive 3D information of the surface of arbitrarily shaped objects. Figure 64 shows an image of a stereo camera system, the resulting stereo image pair, and the height map that has been derived from the images.

The most important requirements for acquiring a pair of stereo images are that the images

- show the same object,
- at the same time, but
- taken from different positions.

The images must be calibrated and rectified. Thereafter, the 3D information can be determined either in form of disparities or as the distance of the object surface from the stereo camera system. The 3D information is available as images that encode the disparities or the distances, respectively.

Additionally, it is possible to directly determine 3D coordinates for points of the object's surface.

Applications for a binocular stereo system comprise, but are not limited to, completeness checks, inspection of ball grid arrays, etc.

If not stated otherwise, the example programs used in this section are

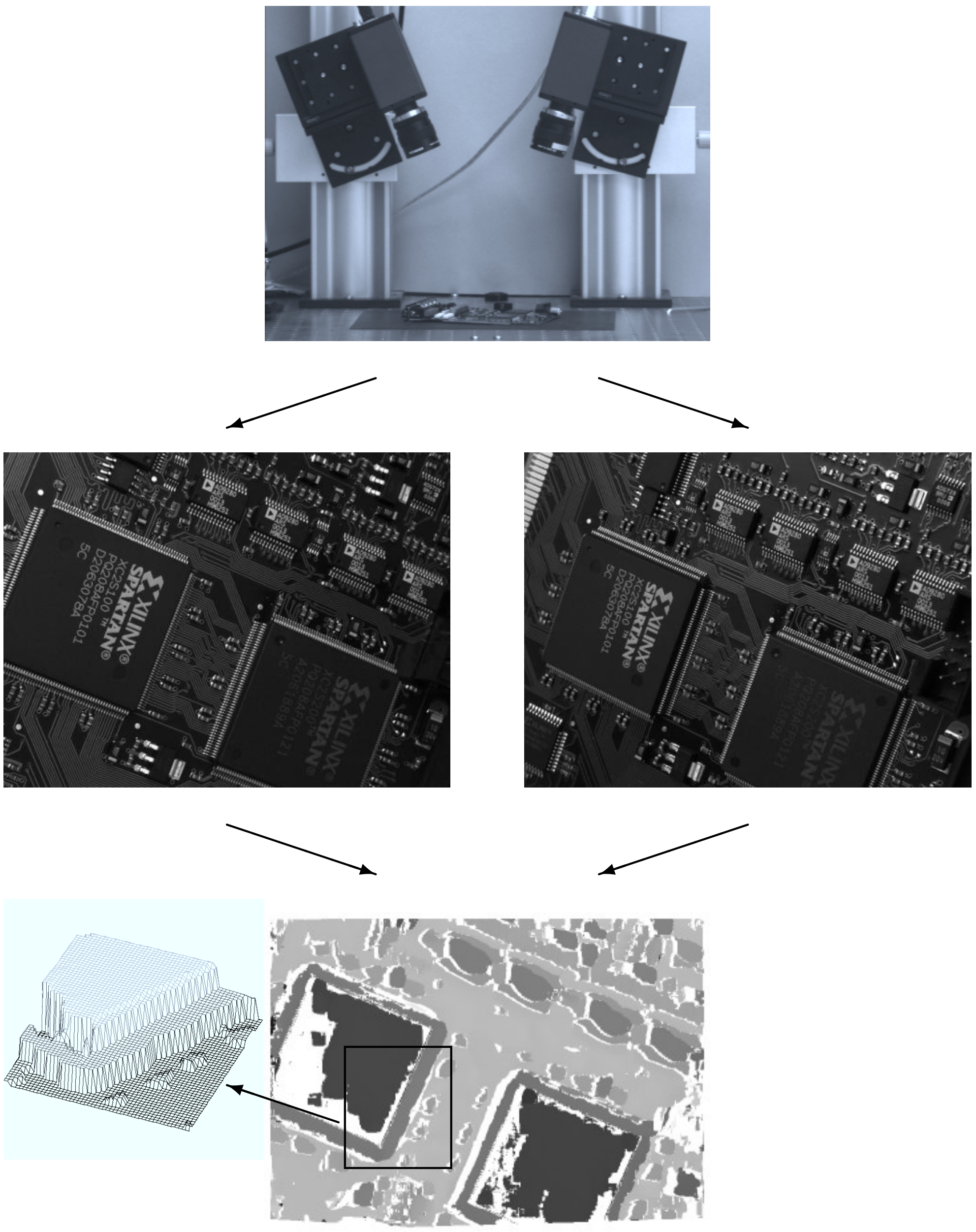


Figure 64: Top: Stereo camera system; Center: Stereo image pair; Bottom: Height map.

- `stereo_calibration.dev`,
- `height_above_reference_plane_from_stereo.dev`, and
- `3d_information_for_selected_points.dev`.

They can be found in the directory `examples\solution_guide\3d_machine_vision`.

As an alternative to fully calibrated stereo, HALCON offers the so-called uncalibrated stereo vision. Here, the relation between the cameras is determined from the scene itself, i.e., without needing a special calibration object. Please refer to section 7.5 on page 123 for more information about this method.

7.1 The Principle of Stereo Vision

The basic principle of binocular stereo vision is very easy to understand. Assume the simplified configuration of two parallel looking 1D cameras with identical internal parameters as shown in figure 65. Furthermore, the basis, i.e., the straight line connecting the two optical centers of the two cameras, is assumed to be coincident with the x-axis of the first camera.

Then, the image plane coordinates of the projections of the point $P(x^c, z^c)$ into the two images can be expressed by

$$u_1 = f\frac{x^c}{z^c} \quad (40)$$

$$u_2 = f\frac{x^c - b}{z^c} \quad (41)$$

where f is the focal length and b the length of the basis.

The pair of image points that results from the projection of one object point into the two images is often referred to as *conjugate points* or *homologous points*.

The difference between the two image locations of the *conjugate points* is called the *disparity* d:

$$d = (u_1 - u_2) = \frac{f \cdot b}{z^c} \quad (42)$$

Given the camera parameters and the image coordinates of two conjugate points, the z^c coordinate of the corresponding object point P, i.e., its distance from the stereo camera system, can be computed by

$$z^c = \frac{f \cdot b}{d} \quad (43)$$

Note that the internal camera parameters of both cameras and the relative pose of the second camera in relation to the first camera are necessary to determine the distance of P from the stereo camera system.

Thus, the tasks to be solved for stereo vision are:

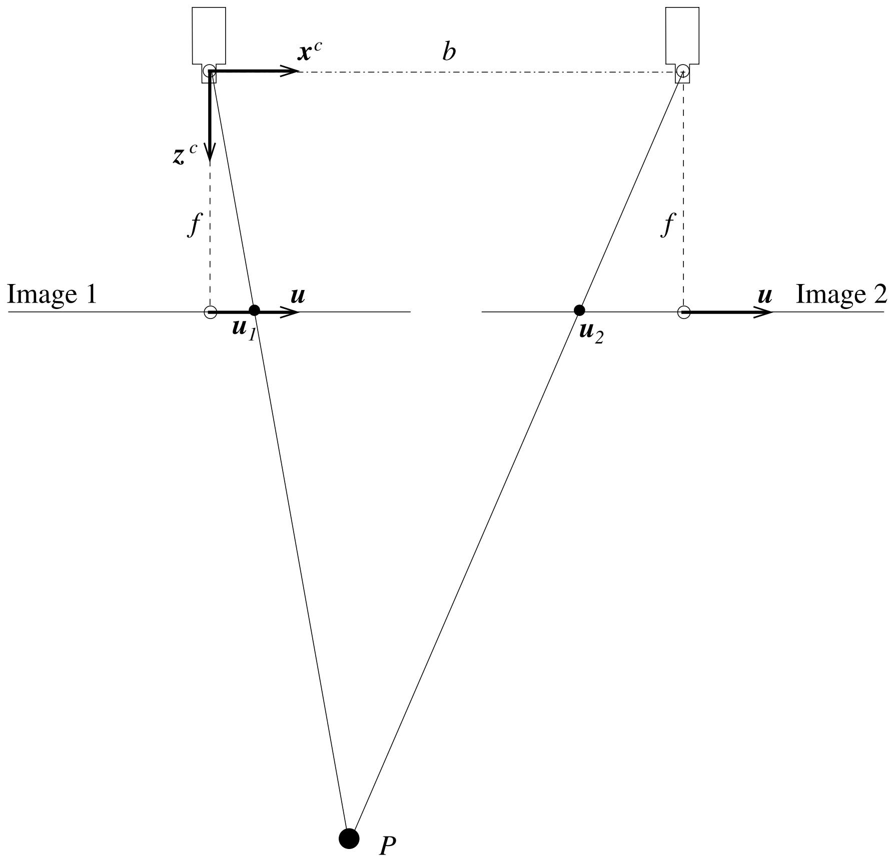


Figure 65: Vertical section of a binocular stereo camera system.

1. Determination of the camera parameters
2. Determination of conjugate points

The first task is achieved by the calibration of the binocular stereo camera system, which is described in section 7.3. This calibration is quite similar to the calibration of a single camera, described in section 3.1 on page 30.

The second task is the so-called stereo matching process, which in HALCON is just a call of the operator `binocular_disparity` (or `binocular_distance`, respectively) for the correlation-based stereo and a call of the operator `binocular_disparity_mg` (or `binocular_distance_mg`, respectively) for multigrid stereo. These operators are described in section 7.4, together with the operators doing all the necessary calculations to obtain world coordinates from the stereo images.

7.2 The Setup of a Stereo Camera System

The stereo camera system consists of two cameras looking at the same object from different positions (see figure 66).

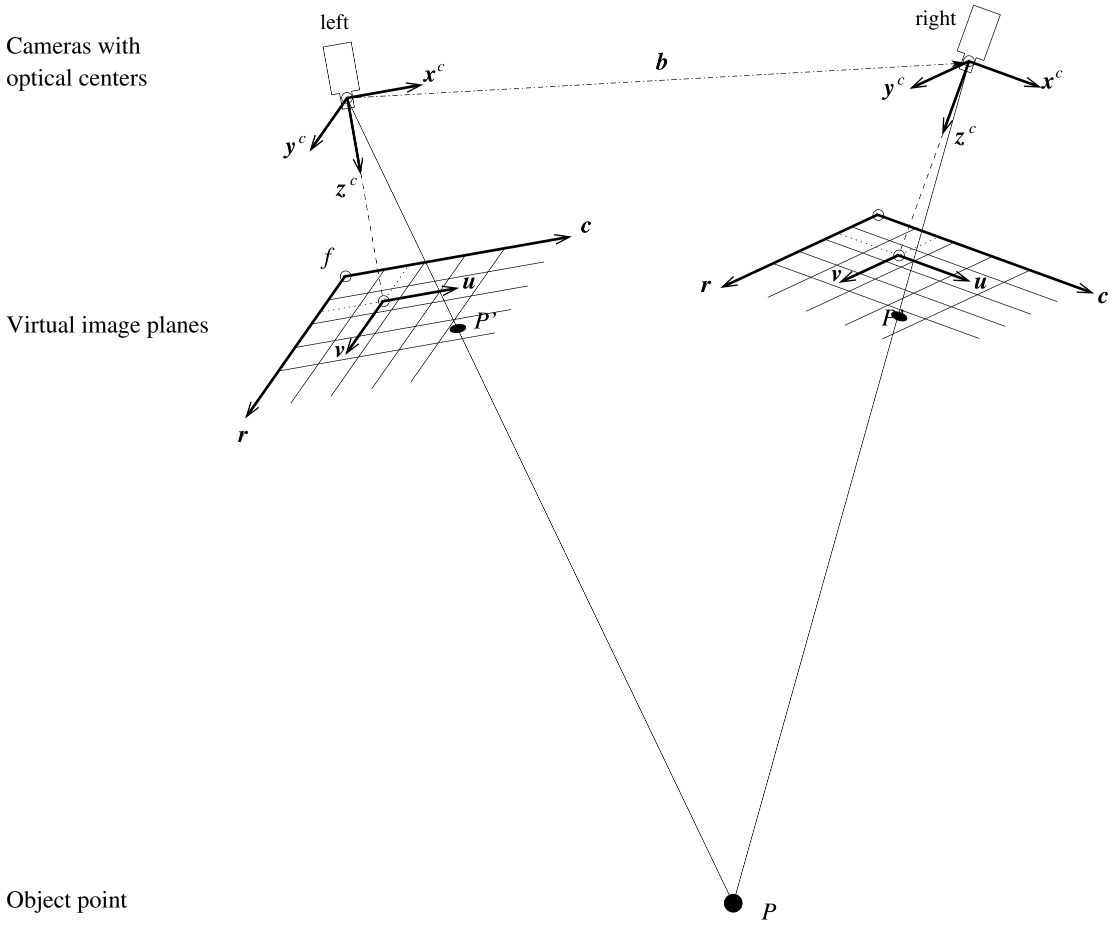


Figure 66: Stereo camera system.

It is very important to ensure that neither the internal camera parameters (e.g., the focal length) nor the relative pose (e.g., the distance between the two cameras) of the two cameras changes during the calibration process or between the calibration process and the ensuing application of the calibrated stereo camera system. Therefore, it is advisable to mount the two cameras on a *stable* platform.

The manner in which the cameras are placed influences the accuracy of the results that is achievable with the stereo camera system.

The distance resolution Δz, i.e., the accuracy with which the distance z of the object surface from the stereo camera system can be determined, can be expressed by

$$\Delta z = \frac{z^2}{f \cdot b} \cdot \Delta d \tag{44}$$

To achieve a high distance resolution, the setup should be chosen such that the length b of the basis as well as the focal length f are large, and that the stereo camera system is placed as close as possible to the object. In addition, the distance resolution depends directly on the accuracy Δd with which the disparities can be determined. Typically, the disparities can be determined with an accuracy of $1/5$ up to $1/10$ pixel, which corresponds to approximately 1 μm for a camera with 7.4 μm pixel size.

In figure 67, the distance resolutions that are achievable in the ideal case are plotted as a function of the distance for four different configurations of focal lengths and base lines, assuming Δd to be 1 μm.

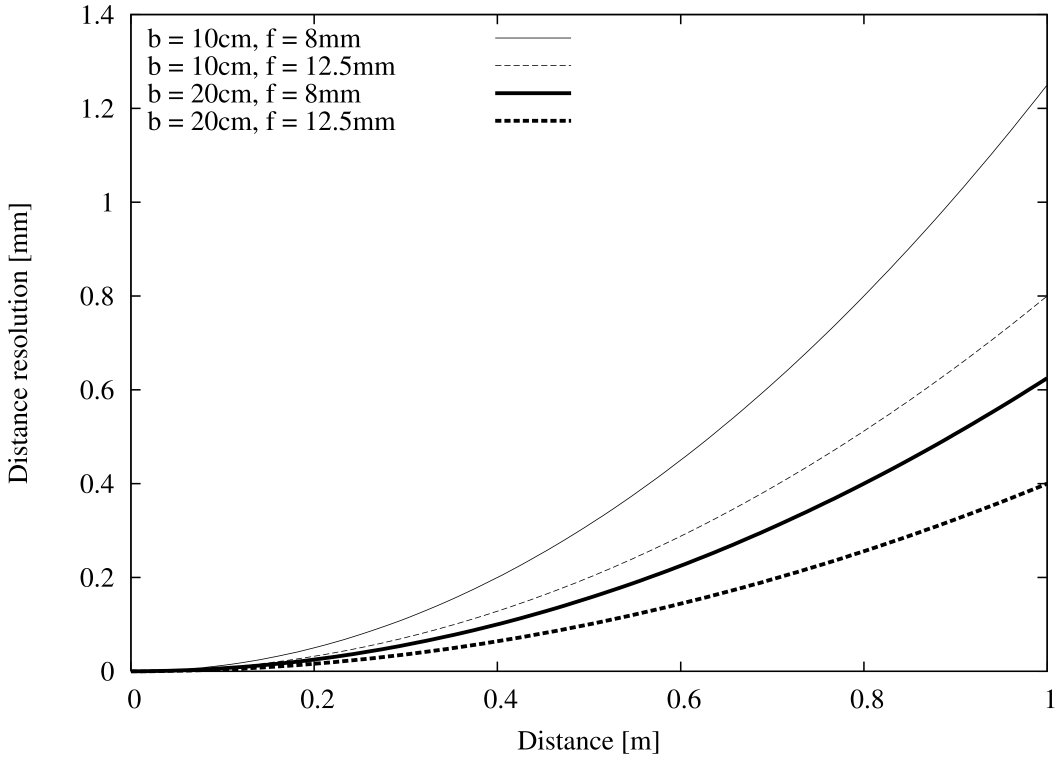


Figure 67: Distance resolution plotted over the distance.

Note that if the ratio between b and z is very large, problems during the stereo matching process may occur, because the two images of the stereo pair differ too much. The maximum reasonable ratio b/z depends on the surface characteristics of the object. In general, objects with little height differences can be imaged with a higher ratio b/z, whereas objects with larger height differences should be imaged with a smaller ratio b/z.

In any case, to ensure a stable calibration the overlapping area of the two stereo images should be as large as possible and the cameras should be approximately aligned, i.e., the rotation around the optical axis should not differ too much between the two cameras.

7.3 Calibrating the Stereo Camera System

As mentioned above, the calibration of the binocular stereo camera system is very similar to the calibration of a single camera (section 3.1 on page 30). The major differences are that the calibration plate must be placed such that it lies completely within both images of each stereo image pair and that the calibration of both images is carried out simultaneously within the operator `binocular_calibration`. Finally, the stereo images must be rectified in order to facilitate all further calculations.

In this section only a brief description of the calibration process is given. More details can be found in section 3.1 on page 30. Only the stereo-specific parts of the calibration process are described in depth.

7.3.1 Rules for Taking Calibration Images

For the calibration of the stereo camera system, multiple stereo images of the calibration plate are necessary, where the calibration plate must be completely visible in both images of each stereo image pair. A typical sequence of stereo calibration image pairs is displayed in figure 68.

The following rules help you to acquire suitable stereo image pairs. Below, you find background information to understand the rules.

- The rules for taking the calibration images for the single camera calibration (see section 3.1.2 on page 33) apply accordingly.
- Do not change the camera setup between the acquisition of the calibration images and the acquisition of the stereo images of the object to be investigated.
- Ensure a proper illumination of the object, avoid reflections.
- If the object shows no texture, consider to project texture onto it or use multigrid stereo (see section 7.4.1 on page 113).
- Place the object such that repetitive patterns are not aligned with the rows of the rectified images.

Overlapping area

In general, the overlapping area of the two stereo images is smaller than the field of view of each individual camera. Therefore, it is not possible to place the calibration plate in all areas of the field of view of each camera. Nevertheless, it is very important to place the calibration plate in the multiple images such that the whole overlapping area is covered as well as possible.

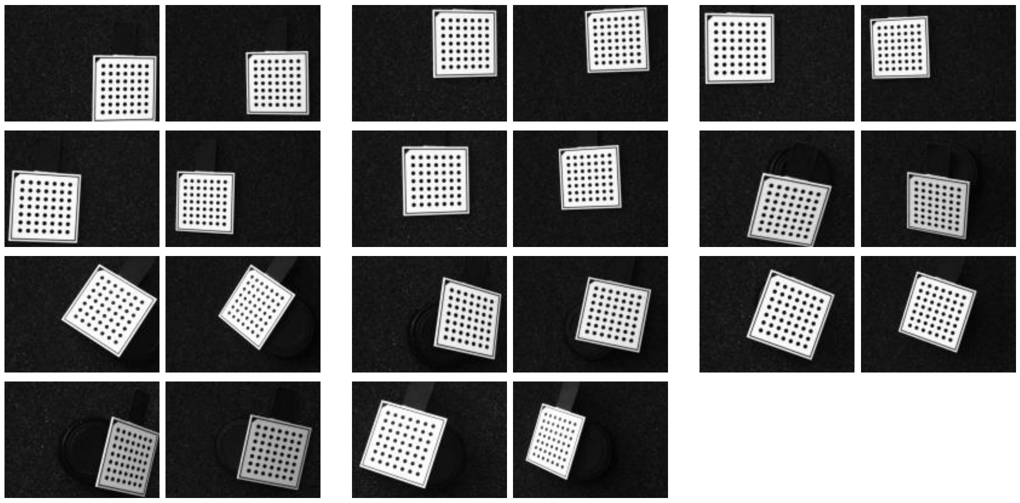


Figure 68: A sequence of eleven stereo calibration image pairs.

Image texture

The 3D coordinates of each object point are derived by intersecting the lines of sight of the respective conjugate image points. The conjugate points are determined by an automatic matching process. This matching process has some properties that should be accounted for during the image acquisition.

For each point of the first image, the conjugate point in the second image must be determined. This point matching relies on the availability of texture. The conjugate points cannot be determined correctly in areas without sufficient texture (figure 69).

This applies in particular for the correlation-based stereo. The multigrid stereo, however, can interpolate values in areas without texture. But note that even then, a certain amount of texture must be available to enable the interpolation. In section 7.4.1 on page 113 the differences between both stereo matching approaches are described in more detail.

Repetitive patterns

If the images contain repetitive patterns, the matching process may be confused, since in this case many points look alike. In order to make the matching process fast and reliable, the stereo images are rectified such that pairs of conjugate points always have identical row coordinates in the rectified images, i.e., that the search space in the second rectified image is reduced to a line. With this, repetitive patterns can disturb the matching process only if they are parallel to the rows of the rectified images (figure 70).

7.3.2 Camera Calibration Input

As in the case of the single camera calibration, the input parameters for the camera calibration can be grouped into two categories:

1. Corresponding points, given in world coordinates as well as in image coordinates of both images

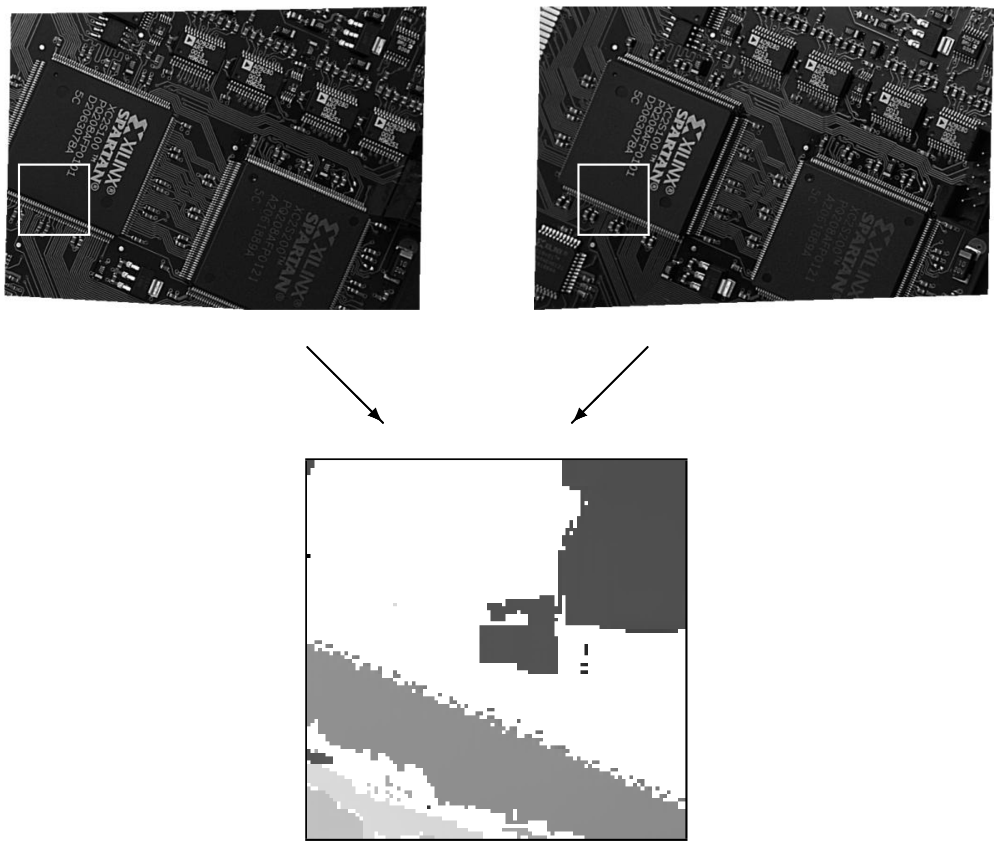


Figure 69: Rectified stereo images and matching result of the correlation-based stereo in a poorly textured area (regions where the matching process failed are displayed white).

2. Initial values for the camera parameters of both cameras

The corresponding points as well as the initial values for the camera parameters are determined similar to the single camera calibration by the use of the operators `find_caltab` and `find_marks_and_pose`. Both operators are described in detail in section 3.1.1 on page 32. The only difference is that the operators must be applied to both stereo images separately.

The initial values for the internal camera parameters can be determined as explained in section 3.1.4 on page 35.

7.3.3 Determining the Camera Parameters

The actual calibration of the stereo camera system is carried out with the operator `binocular_calibration`.

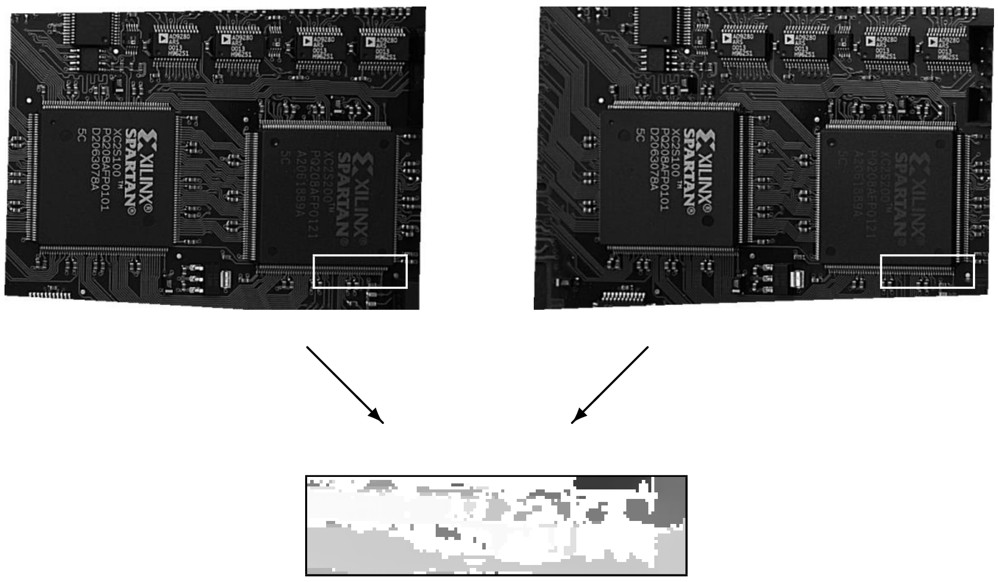


Figure 70: Rectified stereo images with repetitive patterns aligned to the image rows and cutout of the matching result (regions where the matching process failed are displayed white).

```
binocular_calibration (X, Y, Z, RowsL, ColsL, RowsR, ColsR, StartCamParL, \
                       StartCamParR, StartPosesL, StartPosesR, 'all', \
                       CamParamL, CamParamR, NFinalPoseL, NFinalPoseR, \
                       cLPcR, Errors)
```

The only differences to the operator `camera_calibration`, described in section 3.1.5 on page 39 are that the operator `binocular_calibration` needs the image coordinates of the calibration marks from *both* images of each stereo pair, that it needs the initial values for the camera parameters of *both* cameras, and that it also returns the relative pose of the second camera in relation to the first camera.

Note that it is assumed that the parameters of the *first* image stem from the *left* image and the parameters of the *second* image stem from the *right* image, whereas the notations 'left' and 'right' refer to the line of sight of the two cameras. If the images are used in the reverse order, they will appear upside down after the rectification (see section 7.3.4 for an explanation of the rectification step).

Once the stereo camera system is calibrated, it should be left unchanged. If, however, the focus of one camera was modified, it is necessary to determine the internal camera parameters of that camera again (`binocular_calibration` with `EstimateParams` set to *'cam_par1'* or *'cam_par2'*, respectively). In case one camera has been shifted or rotated with respect to the other camera, the relative pose of the second camera in relation to the first camera must be determined again (`binocular_calibration` with `EstimateParams` set to *'pose_rel'*). Further, if the point of view of the camera allows only stereo images with marginal overlaps or the acquisition of an insufficient number of calibration images, `binocular_calibration` can be applied also to already calibrated cameras (with `EstimateParams` set to *'pose_rel'*).

7.3.4 Rectifying the Stereo Images

After the rectification of stereo images, conjugate points lie on the same row in both rectified images. In figure 71 the original images of a stereo pair are shown, where the two cameras are rotated heavily with respect to each other. The corresponding rectified images are displayed in figure 72.

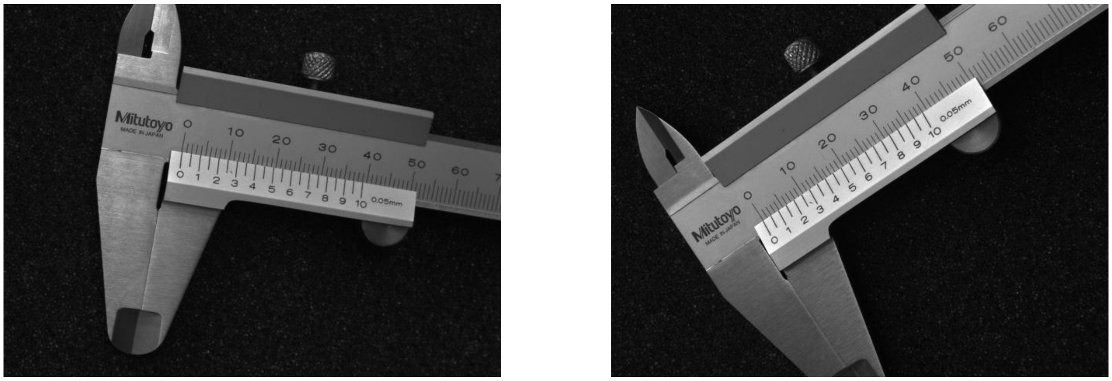


Figure 71: Original stereo images.

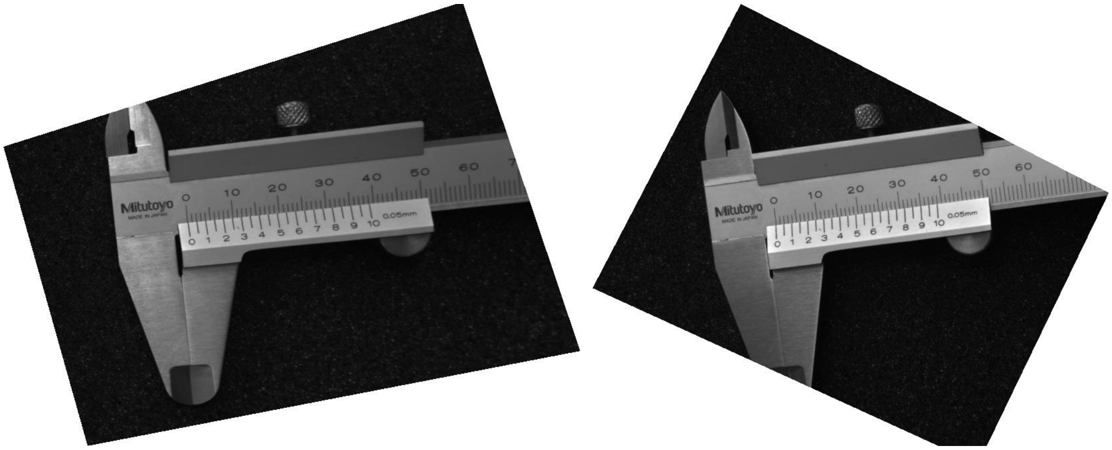


Figure 72: Rectified stereo images.

The rectification itself is carried out using the operators gen_binocular_rectification_map and map_image.

```
gen_binocular_rectification_map (MapL, MapR, CamParamL, CamParamR, cLPcR, 1, \
                                 'geometric', 'bilinear', RectCamParL, \
                                 RectCamParR, CamPoseRectL, CamPoseRectR, \
                                 RectLPosRectR)
```

The operator gen_binocular_rectification_map requires the internal camera parameters of both cameras and the relative pose of the second camera in relation to the first camera. This data is returned by the operator binocular_calibration.

The parameter SubSampling can be used to change the size and resolution of the rectified images with respect to the original images. A value of *1* indicates that the rectified images will have the same size as the original images. Larger values lead to smaller images with a resolution reduced by the given factor, smaller values lead to larger images.

Reducing the image size has the effect that the following stereo matching process runs faster, but also that less details are visible in the result. In general, it is proposed not to use values below *0.5* or above *2*. Otherwise, smoothing or aliasing effects occur, which may disturb the matching process.

The rectification process can be described as projecting the original images onto a common rectified image plane. The method to define this plane can be selected by the parameter Method. So far, only the method *'geometric'* can be selected, in which the orientation of the common rectified image plane is defined by the cross product of the base line and the line of intersection of the two original image planes.

The rectified images can be thought of as being acquired by a virtual stereo camera system, called rectified stereo camera system, as displayed in figure 73. The optical centers of the rectified cameras are the same as for the real cameras, but the rectified cameras are rotated such that they are looking parallel and that their x-axes are collinear. In addition, both rectified cameras have the same focal length. Therefore, the two image planes coincide. Note that the principal point of the rectified images, which is the origin of the image plane coordinate system, may lie outside the image.

The parameter Interpolation specifies whether bilinear interpolation (*'bilinear'*) should be applied between the pixels of the input images or whether the gray value of the nearest pixel (*'none'*) should be used. Bilinear interpolation yields smoother rectified images, whereas the use of the nearest neighbor is faster.

The operator returns the rectification maps and the camera parameters of the virtual, rectified cameras.

Finally, the operator map_image can be applied to both stereo images using the respective rectification map generated by the operator gen_binocular_rectification_map.

```
map_image (ImageL, MapL, ImageRectifiedL)
map_image (ImageR, MapR, ImageRectifiedR)
```

If the calibration was erroneous, the rectification will produce wrong results. This can be checked very easily by comparing the row coordinates of conjugate points selected from the two rectified images. If the row coordinates of conjugate points are different within the two rectified images, they are not correctly rectified. In this case, you should check the calibration process carefully.

An incorrectly rectified image pair may look like the one displayed in figure 74.

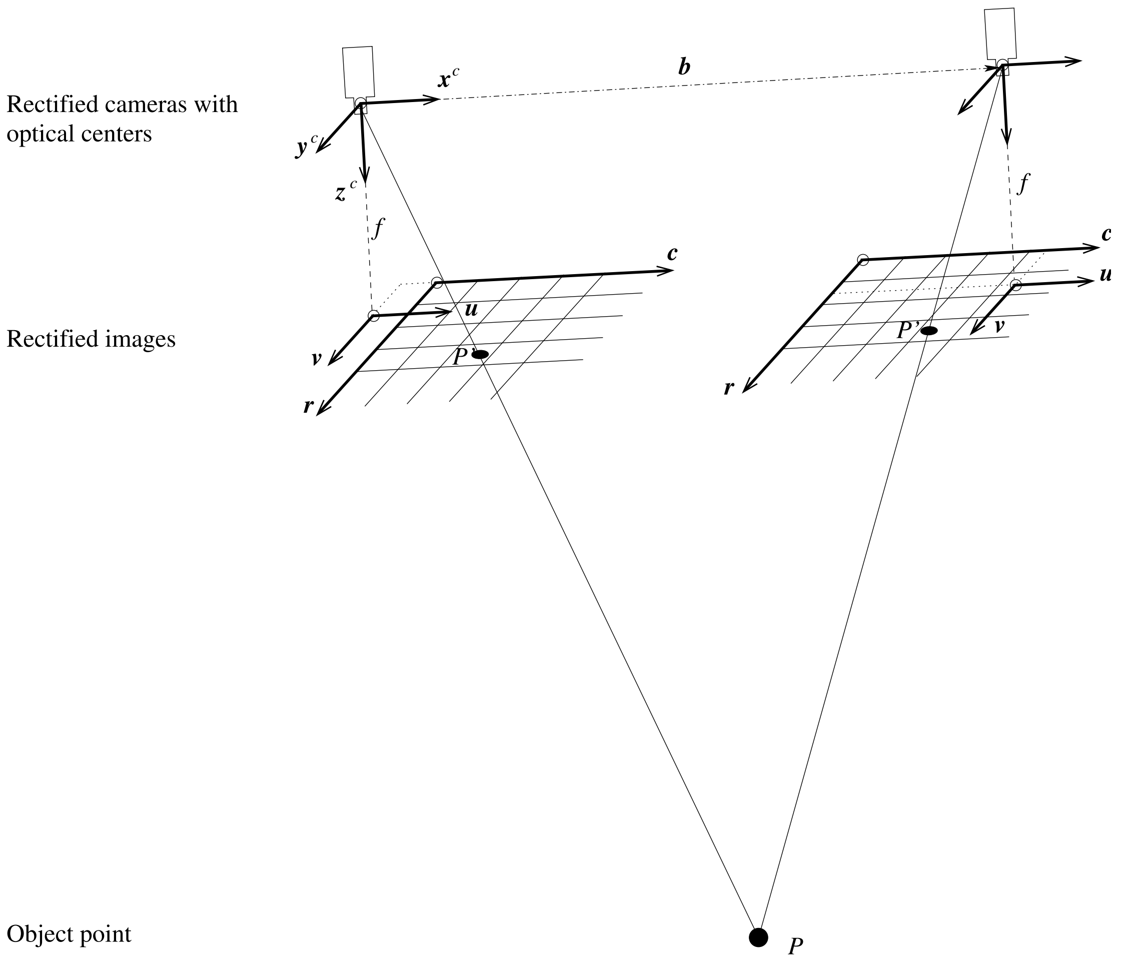


Figure 73: Rectified stereo camera system.

7.4 Reconstructing 3D Information from Images

There are many possibilities to derive 3D information from rectified stereo images. If only non-metrical information about the surface of an object is needed, it may suffice to determine the disparities within the overlapping area of the stereo image pair by using the operator `binocular_disparity` for correlation-based stereo (section 7.4.2 on page 114) or `binocular_disparity_mg` for multigrid stereo (section 7.4.3 on page 116). The differences between both stereo matching approaches are described in more detail in section 7.4.1.

If metrical information is required, the operator `binocular_distance` (section 7.4.4 on page 118) or `binocular_distance_mg` (section 7.4.5 on page 120), respectively, can be used to extract the distance of the object surface from the stereo camera system (see section 7.4.4 on page 118 for the definition of the distance).

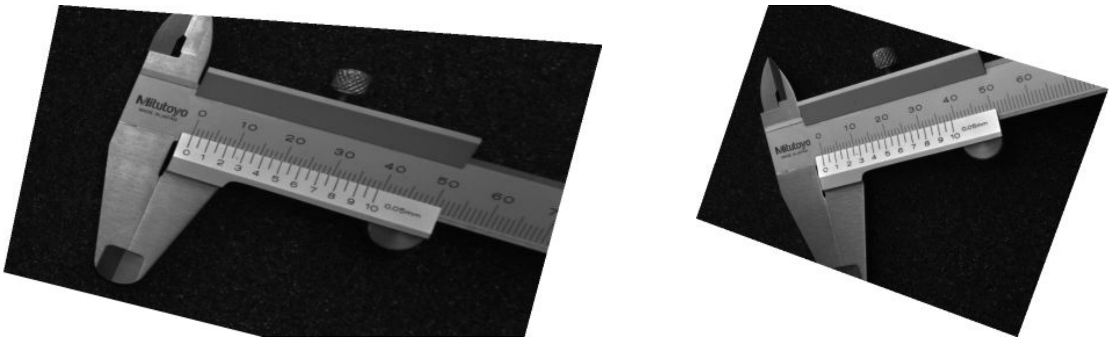


Figure 74: Incorrectly rectified stereo images.

Having a disparity image of a rectified binocular stereo system, you can additionally derive the corresponding x, y, and z coordinates using disparity_image_to_xyz.

To derive metrical information for selected points only, the operators disparity_to_distance or disparity_to_point_3d can be used. The first of these two operators calculates the distance z of points from the stereo camera system based on their disparity. The second operator calculates the x, y, and z coordinates from the row and column position of a point in the first rectified image and from its disparity.

Alternatively, the operator intersect_lines_of_sight can be used to calculate the x, y, and z coordinates of selected points. It does not require to determine the disparities in advance. Only the image coordinates of the conjugate points need to be given, together with the camera parameters. This operator can also handle image coordinates of the original stereo images. Thus, the rectification can be omitted. Admittedly, the conjugate points must be determined by yourself.

Note that all operators, which deal with disparities or distances require all inputs to be based on the rectified images. This holds for the image coordinates as well as for the camera parameters.

7.4.1 Correlation-Based Versus Multigrid Stereo

Two approaches for stereo matching are available, the traditionally used correlation-based stereo matching and the multigrid stereo matching. The correlation-based stereo matching uses correlation techniques to find corresponding points and thus to determine the disparities or distances for the observed image points. The disparities or distances are calulated with the operators binocular_disparity or binocular_distance, respectively. The correlation-based stereo is characterized by the following advantages and disadvantages:

Most important advantages of correlation-based stereo:

- fast,
- can be automatically parallelized on multi-core or multi-processor hardware, and
- is invariant against gray value changes.

Most important disadvantage of correlation-based stereo:

- works good only for significantly textured areas. Areas without enough texture can not be reconstructed.

The multigrid stereo matching uses a variational approach based on multigrid methods. This approach returns disparity and distance values also for image parts that contain no texture (as long as these parts are surrounded by significant structures between which an interpolation of values is possible). The disparities or distances are calculated with the operators `binocular_disparity_mg` or `binocular_distance_mg`, respectively. The multigrid stereo is characterized by the following advantages and disadvantages:

Most important advantages of multigrid stereo:

- interpolates 3D information for areas without texture based on the surrounding areas,
- in particular for edges, the accuracy is higher as for correlation-based stereo, and
- the resolution is higher than for correlation-based stereo, i.e., smaller objects can be reconstructed.

Most important disadvantages of multigrid stereo:

- only partially invariant against gray value changes and
- cannot be parallelized automatically.

7.4.2 Determining Disparities Using Correlation-Based Stereo

Disparities are an indicator for the distance z of object points from the stereo camera system, since points with equal disparities also have equal distances z (equation 43 on page 102).

Therefore, in case it is only necessary to know whether there are locally high objects it suffices to derive the disparities. For correlation-based stereo, this is done by using the operator `binocular_disparity`.

```
binocular_disparity (ImageRectifiedL, ImageRectifiedR, DisparityImage, \
                     ScoreImageDisparity, 'ncc', MaskWidth, MaskHeight, \
                     TextureThresh, MinDisparity, MaxDisparity, NumLevels, \
                     ScoreThresh, 'left_right_check', 'interpolation')
```

The operator requires the two *rectified* images as input. The disparities are only derived for those conjugate points that lie within the respective image domain in both images. With this, it is possible to speed up the calculation of the disparities if the image domain of at least one of the two rectified images is reduced to a region of interest, e.g., by using the operator `reduce_domain`.

Several parameters can be used to control the behavior of the matching process that is performed by the operator `binocular_disparity` to determine the conjugate points:

With the parameter `Method`, the matching function is selected. The methods *'sad'* (summed absolute differences) and *'ssd'* (summed squared differences) compare the gray values of the pixels within a matching window directly, whereas the method *'ncc'* (normalized cross correlation) compensates for the mean gray value and its variance within the matching window. Therefore, if the two images differ in brightness and contrast, the method *'ncc'* should be preferred. However, since the internal computations are less complex for the methods *'sad'* and *'ssd'*, they are faster than the method *'ncc'*.

The width and height of the matching window can be set independently with the parameters `MaskWidth` and `MaskHeight`. The values should be odd numbers. Otherwise they will be increased by one. A larger matching window will lead to a smoother disparity image, but may result in the loss of small details. In contrary, the results of a smaller matching window tend to be noisy but they show more spatial details.

Because the matching process relies on the availability of texture, low-textured areas can be excluded from the matching process. The parameter `TextureThresh` defines the minimum allowed variance within the matching window. For areas where the texture is too low no disparities will be determined.

The parameters `MinDisparity` and `MaxDisparity` define the minimum and maximum disparity values. They are used to restrict the search space for the matching process. If the specified disparity range does not contain the actual range of the disparities, the conjugate points cannot be found correctly; therefore, the disparities will be incomplete and erroneous. On the other hand, if the disparity range is specified too large, the matching process will be slower and the probability of mismatches rises.

Therefore, it is important to set the parameters `MinDisparity` and `MaxDisparity` carefully. There are several possibilities to determine the appropriate values:

- If you know the minimum and maximum distance of the object from the stereo camera system (section 7.4.4 on page 118), you can use the operator `distance_to_disparity` to determine the respective disparity values.
- You can also determine these values directly from the rectified images. For this, you should display the two rectified images and measure the approximate column coordinates of the point N, which is nearest to the stereo camera system (N_{col}^{image1} and N_{col}^{image2}) and of the point F, which is the farthest away (F_{col}^{image1} and F_{col}^{image2}), each in both rectified images.
  Now, the values for the definition of the disparity range can be calculated as follows:

$$MinDisparity = N_{col}^{image2} - N_{col}^{image1} \tag{45}$$

$$MaxDisparity = F_{col}^{image2} - F_{col}^{image1} \tag{46}$$

The operator `binocular_disparity` uses image pyramids to improve the matching speed. The disparity range specified by the parameters `MinDisparity` and `MaxDisparity` is only used on the uppermost pyramid level, indicated by the parameter `NumLevels`. Based on the matching results on that level, the disparity range for the matching on the next lower pyramid levels is adapted automatically.

The benefits with respect to the execution time are greatest if the objects have different regions between which the appropriate disparity range varies strongly. However, take care that the value for `NumLevels` is not set too large, as otherwise the matching process may fail because of lack of texture on the uppermost pyramid level.

The parameter `ScoreThresh` specifies which matching scores are acceptable. Points for which the matching score is not acceptable are excluded from the results, i.e., the resulting disparity image has a reduced domain that comprises only the accepted points.

Note that the value for `ScoreThresh` must be set according to the matching function selected via `Method`. The two methods *'sad'* ($0 \leq$ score ≤ 255) and *'ssd'* ($0 \leq$ score ≤ 65025) return lower matching scores for better matches. In contrast, the method *'ncc'* (-1 $\leq$ score $\leq$ 1) returns higher values for better matches, where a score of zero indicates that the two matching windows are totally different and a score of minus one denotes that the second matching window is exactly inverse to the first matching window.

The parameter `Filter` can be used to activate a downstream filter by which the reliability of the resulting disparities is increased. Currently, it is possible to select the method *'left_right_check'*, which verifies the matching results based on a second matching in the reverse direction. Only if both matching results correspond to each other, the resulting conjugate points are accepted. In some cases, this may lead to gaps in the disparity image, even in well textured areas, as this verification is very strict. If you do not want to verify the matching results based on the *'left_right_check'*, set the parameter `Filter` to *'none'*.

The subpixel refinement of the disparities is switched on by setting the parameter `SubDisparity` to *'interpolation'*. It is switched off by setting the parameter to *'none'*.

The results of the operator `binocular_disparity` are the two images `Disparity` and `Score`, which contain the disparities and the matching score, respectively. In figure 75, a rectified stereo image pair is displayed, from which the disparity and score images, displayed in figure 76 were derived.

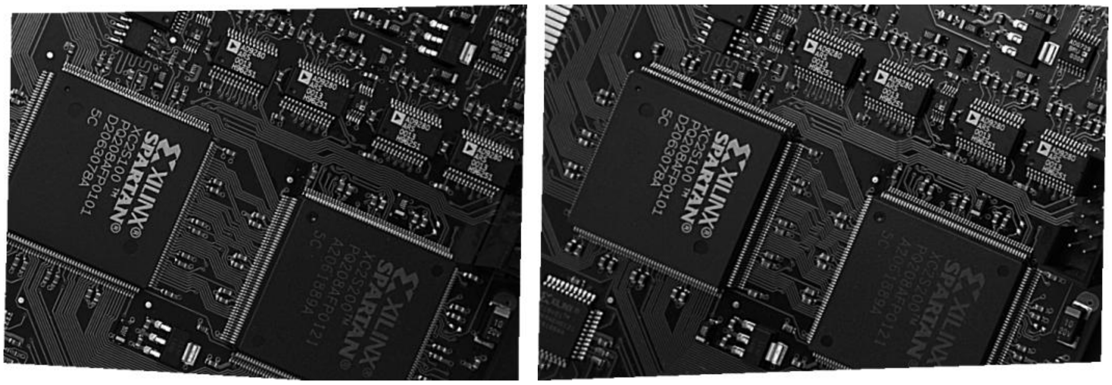


Figure 75: Rectified stereo images.

Both resulting images refer to the image geometry of the first rectified image, i.e., the disparity for the point (r,c) of the first rectified image is the gray value at the position (r,c) of the disparity image. The disparity image can, e.g., be used to extract the components of the board, which would be more difficult in the original images, i.e., without the use of 3D information.

In figure 76, areas where the matching did not succeed, i.e., undefined regions of the images, are displayed white in the disparity image and black in the score image.

7.4.3 Determining Disparities Using Multigrid Stereo

For multigrid stereo, the disparities can be derived with `binocular_disparity_mg`. Similar to the correlation-based approach, the two rectified images are used as input and the disparity image as well as a score image are returned. But as the disparities are obtained by a different algorithm, the parameters that control the behavior of the multigrid stereo matching process are completely different. In particular, the multigrid-specific control parameters are `GrayConstancy`, `GradientConstancy`, `Smoothness`,

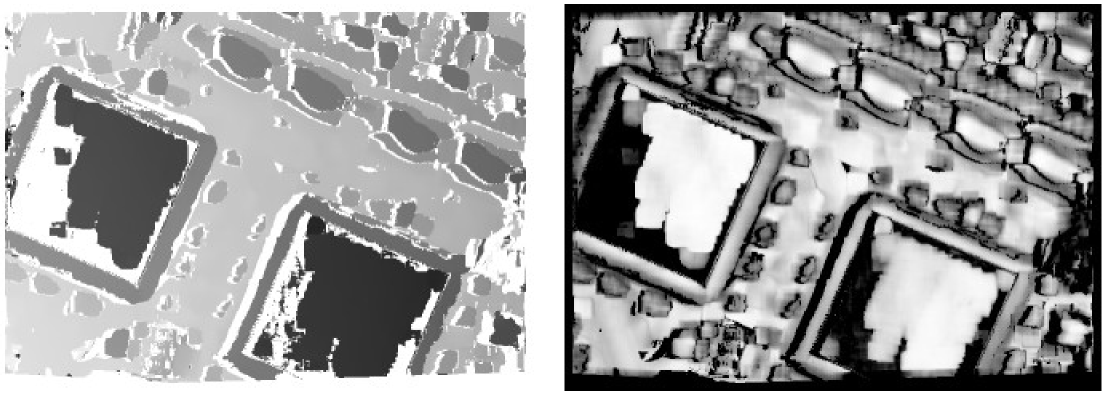


Figure 76: Disparity image (left) and score image (right).

InitialGuess, CalculateScore, MGParamName, and MGParamValue. They are explained in detail in the Reference Manual entry for binocular_disparity_mg. The significant differences between correlation-based and multigrid stereo are listed in section 7.4.1 on page 113.

The HDevelop example program examples\hdevelop\Tools\Stereo\binocular_disparity_mg.dev shows how to determine the disparities of the components on a PCB using binocular_disparity_mg with different levels of accuracy.

There, first the two stereo images are rectified.

```
read_image (ImageL, 'stereo/board/board_l_01')
read_image (ImageR, 'stereo/board/board_r_01')
CamParamL := [0.0130507774353,-665.817817207,1.4803417027e-5,1.48e-5, \
              155.89225769,126.70664978,320,240]
CamParamR := [0.0131776504517,-731.860636733,1.47997569293e-5,1.48e-5, \
              162.98210144,119.301040649,320,240]
RelPose := [0.153573364258,-0.00373362231255,0.0447351264954, \
            0.174289124775,319.843388114,359.894955219,0]
gen_binocular_rectification_map (MapL, MapR, CamParamL, CamParamR, RelPose, \
                                 1, 'geometric', 'bilinear', RectCamParL, \
                                 RectCamParR, CamPoseRectL, CamPoseRectR, \
                                 RectLPosRectR)
map_image (ImageL, MapL, ImageRectifiedL)
map_image (ImageR, MapR, ImageRectifiedR)
```

Then, binocular_disparity_mg is called four times, each time with a different parameter for the parameter MGParamValue, which sets the accuracy that should be achieved. Note that an increasing accuracy also leads to an increasing runtime.

```
DefaultParameters := ['fast','fast_accurate','accurate','very_accurate']
for I := 0 to |DefaultParameters|-1 by 1
  Parameters := DefaultParameters[I]
  binocular_disparity_mg (ImageRectifiedL, ImageRectifiedR, Disparity, \
                          Score, 1, 10, 5, 0, 'false', \
                          'default_parameters', Parameters)
endfor
```

7.4.4 Determining Distances Using Correlation-Based Stereo

The distance of an object point from the stereo camera system is defined as its distance from the x-y-plane of the coordinate system of the first rectified camera. For correlation-based stereo, it can be determined by the operator `binocular_distance`, which is used analogously to the operator `binocular_disparity` described in section 7.4.2 on page 114.

```
binocular_distance (ImageRectifiedL, ImageRectifiedR, DistanceImage, \
                    ScoreImageDistance, RectCamParL, RectCamParR, \
                    RectLPosRectR, 'ncc', MaskWidth, MaskHeight, \
                    TextureThresh, MinDisparity, MaxDisparity, NumLevels, \
                    ScoreThresh, 'left_right_check', 'interpolation')
```

The three additional parameters, namely the camera parameters of the rectified cameras as well as the relative pose of the second rectified camera in relation to the first rectified camera can be taken directly from the output of the operator `gen_binocular_rectification_map`.

Figure 77 shows the distance image and the respective score image for the rectified stereo pair of figure 75 on page 116. Because the distance is calculated directly from the disparities and from the camera parameters, the distance image looks similar to the disparity image (figure 76). What is more, the score images are identical, since the underlying matching process is identical.

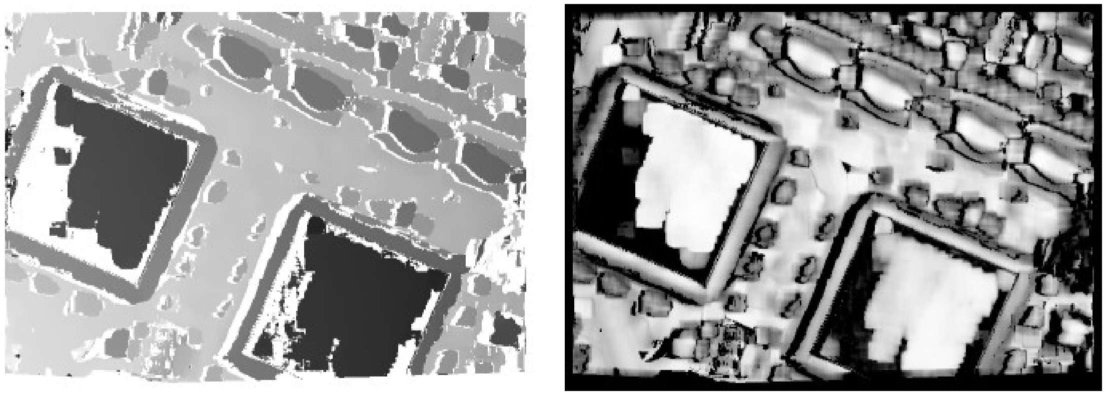


Figure 77: Distance image (left) and score image (right).

It can be seen from figure 77 that the distance of the board changes continuously from left to right. The reason is that, in general, the x-y-plane of the coordinate system of the first rectified camera will be tilted with respect to the object surface (see figure 78).

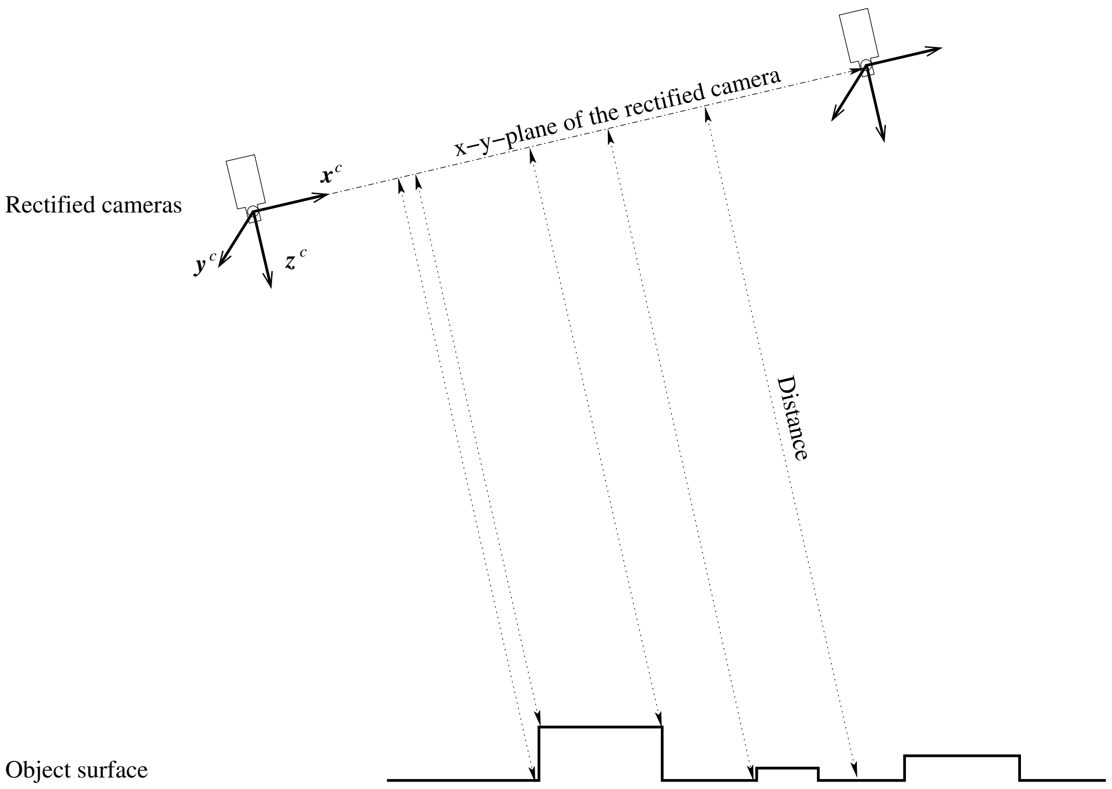


Figure 78: Distances of the object surface from the x-y-plane of the coordinate system of the first rectified camera.

If it is necessary that one reference plane of the object surface has a constant distance value of, e.g., zero, the tilt can be compensated easily: First, at least three points that lie on the reference plane must be defined. These points are used to determine the orientation of the (tilted) reference plane in the distance image. Therefore, they should be selected such that they enclose the region of interest in the distance image. Then, a distance image of the (tilted) reference plane can be simulated and subtracted from the distance image of the object. Finally, the distance values themselves must be adapted by scaling them with the cosine of the angle between the tilted and the corrected reference plane.

These calculations are carried out in the procedure

```
procedure tilt_correction (DistanceImage, RegionDefiningReferencePlane:
                           DistanceImageCorrected:
                           :
                           )
```

which is part of the example program `examples\solution_guide\3d_machine_vision\`

height_above_reference_plane_from_stereo.dev (appendix B.4 on page 170). In principle, this procedure can also be used to correct the disparity image, but note that you must not use the corrected disparity values as input to any operators that derive metric information.

If the reference plane is the ground plane of the object, an inversion of the distance image generates an image that encodes the heights above the ground plane. Such an image is displayed on the left hand side in figure 79.

Objects of different height above or below the ground plane can be segmented easily using a simple threshold with the minimal and maximal values given directly in units of the world coordinate system, e.g., meters. The image on the right hand side of figure 79 shows the results of such a segmentation, which can be carried out based on the corrected distance image or the image of the heights above the ground plane.

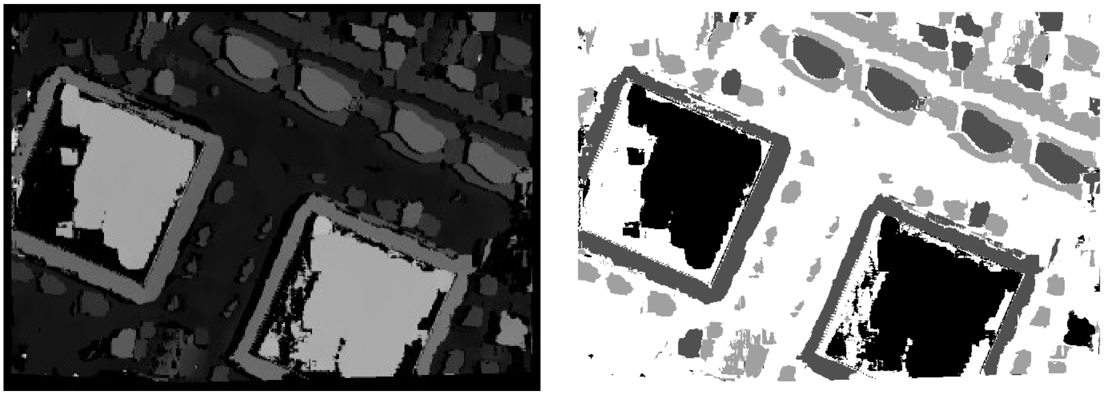


Figure 79: Left: Height above the reference plane; Right: Segmentation of high objects (white: 0-0.4 mm, light gray: 0.4-1.5 mm, dark gray: 1.5-2.5 mm, black: 2.5-5 mm).

7.4.5 Determining Distances Using Multigrid Stereo

For multigrid stereo, the distance of an object point from the stereo camera system can be determined by the operator binocular_distance_mg, which is used analogously to the operator binocular_disparity_mg described in section 7.4.3 on page 116, but with the three additional parameters described also for binocular_distance. These are the camera parameters of the rectified cameras as well as the relative pose of the second rectified camera in relation to the first rectified camera, which can be taken directly from the output of the operators binocular_calibration and gen_binocular_rectification_map.

The HDevelop example program examples\hdevelop\Tools\Stereo\binocular_distance_mg.dev shows how to determine the depths of the components on a PCB with high accuracy.

```
binocular_distance_mg (ImageRectifiedL, ImageRectifiedR, Distance, Score, \
                       RectCamParL, RectCamParR, RectLPosRectR, 1, 10, 5, \
                       0, 'false', 'default_parameters', 'accurate')
```

7.4.6 Determining Distances for Selected Points from the Disparity Image

If only the distances of selected points should be determined, the operator `disparity_to_distance` can be used. It simply transforms given disparity values into the respective distance values. For example, if you want to know the minimum and maximum distance of the object from the stereo camera system you can determine the minimum and maximum disparity values from the disparity image and transform them into distances.

```
min_max_gray (CaltabL, Disparity, 0, MinDisparity, MaxDisparity, _)
disparity_to_distance (RectCamParL, RectCamParR, RectLPosRectR, \
                       [MinDisparity,MaxDisparity], MinMaxDistance)
```

This transformation is constant for the entire rectified image, i.e., all points having the same disparity have the same distance from the x-y-plane of the coordinate system of the first rectified camera. Therefore, besides the camera parameters of the rectified cameras, only the disparity values need to be given.

7.4.7 Determining 3D Coordinates from the Disparity Image

If the x, y, and z coordinates of points have to be calculated, different operators are available that derive the corresponding information from the disparity image:

`disparity_to_point_3d` computes the 3D coordinates for specified image coordinates and returns them in three tuples:

```
disparity_to_point_3d (RectCamParL, RectCamParR, RectLPosRectR, RL, CL, \
                       DisparityOfSelectedPoints, X_CCS_FromDisparity, \
                       Y_CCS_FromDisparity, Z_CCS_FromDisparity)
```

`disparity_image_to_xyz` computes 3D coordinates for the complete image and returns them in three images (see the HDevelop example program `examples\hdevelop\Tools\Stereo\disparity_image_to_xyz.dev`):

```
disparity_image_to_xyz (DisparityImage, X, Y, Z, RectCamParL, RectCamParR, \
                        RectLPosRectR)
```

The operators require the camera parameters of the two rectified cameras and the relative pose of the cameras.

The x, y, and z coordinates are returned in the coordinate system of the first rectified camera.

7.4.8 Determining 3D Coordinates for Selected Points from Point Correspondences

If only the 3D coordinates of selected points should be determined, you can alternatively use the operator `intersect_lines_of_sight` to determine the x, y, and z coordinates of points from the image coordinates of the respective conjugate points. Note that you must determine the image coordinates of the conjugate points yourself.

```
intersect_lines_of_sight (RectCamParL, RectCamParR, RectLPosRectR, RL, CL, \
                          RR, CR, X_CCS_FromIntersect, Y_CCS_FromIntersect, \
                          Z_CCS_FromIntersect, Dist)
```

The x, y, and z coordinates are returned in the coordinate system of the first (rectified) camera.

The operator can also handle image coordinates of the original stereo images. Thus, the rectification can be omitted. In this case, the camera parameters of the original stereo cameras have to be given instead of the parameters of the rectified cameras.

It is possible to transform the x, y, and z coordinates determined by the latter two operators from the coordinate system of the first (rectified) camera into a given coordinate system WCS, e.g., a coordinate system with respect to the building plan of, say, a factory building. For this, a homogeneous transformation matrix, which describes the transformation between the two coordinate systems is needed.

This homogeneous transformation matrix can be determined in various ways. The easiest way is to take an image of a HALCON calibration plate with the first camera only. If the 3D coordinates refer to the *rectified* camera coordinate system, the image must be rectified as well. Then, the pose of the calibration plate in relation to the first (rectified) camera can be determined using the operators find_caltab, find_marks_and_pose, and camera_calibration.

```
find_caltab (ImageRectifiedL, CaltabL, 'caltab_30mm.descr', SizeGauss, \
             MarkThresh, MinDiamMarks)
find_marks_and_pose (ImageRectifiedL, CaltabL, 'caltab_30mm.descr', \
                     RectCamParL, StartThresh, DeltaThresh, MinThresh, \
                     Alpha, MinContLength, MaxDiamMarks, RCoordL, CCoordL, \
                     StartPoseL)
camera_calibration (X, Y, Z, RCoordL, CCoordL, RectCamParL, StartPoseL, \
                    'pose', _, PoseOfCalibrationPlate, Errors)
```

The resulting pose can be converted into a homogeneous transformation matrix.

```
pose_to_hom_mat3d (PoseOfCalibrationPlate, HomMat3d_WCS_to_RectCCS)
```

If necessary, the transformation matrix can be modified with the operators hom_mat3d_rotate_local, hom_mat3d_translate_local, and hom_mat3d_scale_local.

```
hom_mat3d_translate_local (HomMat3d_WCS_to_RectCCS, 0.01125, -0.01125, 0, \
                           HomMat3DTranslate)
hom_mat3d_rotate_local (HomMat3DTranslate, rad(180), 'y', \
                        HomMat3d_WCS_to_RectCCS)
```

The homogeneous transformation matrix must be inverted in order to represent the transformation from the (rectified) camera coordinate system into the WCS.

```
hom_mat3d_invert (HomMat3d_WCS_to_RectCCS, HomMat3d_RectCCS_to_WCS)
```

Finally, the 3D coordinates can be transformed using the operator affine_trans_point_3d.

```
affine_trans_point_3d (HomMat3d_RectCCS_to_WCS, X_CCS_FromIntersect, \
                        Y_CCS_FromIntersect, Z_CCS_FromIntersect, X_WCS, \
                        Y_WCS, Z_WCS)
```

The homogeneous transformation matrix can also be determined from three specific points. If the origin of the WCS, a point on its x-axis, and a third point that lies in the x-y-plane, e.g., directly on the y-axis, are given, the transformation matrix can be determined using the procedure

```
procedure gen_hom_mat3d_from_three_points (: : Origin, PointOnXAxis,
                                                PointInXYPlane: HomMat3d)
```

which is part of the example program `examples\solution_guide\3d_machine_vision\3d_information_for_selected_points.dev`.

The resulting homogeneous transformation matrix can be used as input for the operator `affine_trans_point_3d`, as shown above.

7.5 Uncalibrated Stereo Vision

Similar to uncalibrated mosaicking (see section 5 on page 76), HALCON also provides an "uncalibrated" version of stereo vision, which derives information about the cameras from the scene itself by matching characteristic points. The main advantage of this method is that you need no special calibration object. The main disadvantage of this method is that, without a precisely known calibration object, the accuracy of the results is highly dependent on the content of the scene, i.e., the accuracy of the results degrades if the scene does not contain enough 3D information or if the extracted characteristic points in the two images do not precisely correspond to the same world points, e.g., due to occlusion.

In fact, HALCON provides two versions of uncalibrated stereo: **Without any knowledge about the cameras and about the scene**, you can rectify the stereo images and perform a segmentation similar to the method described in section 7.4.4 on page 118. For this, you first use the operator `match_fundamental_matrix_ransac`, which determines the so-called *fundamental matrix*. This matrix models the cameras and their relation; but in contrast to the model described in section 7.1 on page 102, internal and external parameters are not available separately, thus no metric information can be derived.

The fundamental matrix is then passed on to the operator `gen_binocular_proj_rectification`, which is the "uncalibrated" version of the operator `gen_binocular_rectification_map`. With the output of this operator, i.e., the rectification maps, you can then proceed to rectify the images as described in section 7.3.4 on page 110. Because the relative pose of the cameras is not known, you cannot generate the distance image and segment it as described in section 7.4.4 on page 118. The HDevelop example program `hdevelop\Applications\Stereo\board_segmentation_uncalib.dev` shows an alternative approach that can be used if the reference plane appears dominant in the images, i.e., if many correspondences are found on it.

Because no calibration object is needed, this method can be used to perform stereo vision with a single camera. Note, however, that the method assumes that there are no lens distortions in the images. Therefore, the accuracy of the results degrades if the lens has significant distortions.

If the internal parameters of the camera are known, you can determine the relative pose between the cameras using the operator `match_rel_pose_ransac` and then use all the stereo methods described for the fully calibrated case. There is, however, a limitation: The determined pose is relative in a second sense, because it can be determined only up to a scale factor. The reason for this is that without any knowledge about the scene, the algorithm cannot know whether the points in the scene are further away or whether the cameras are further apart because the effect in the image is the same in both cases. If you have additional information about the scene, you can solve this ambiguity and determine the "real" relative pose. This method is shown in the HDevelop example program `hdevelop\Tools\Stereo\uncalib_stereo_boxes.dev`.

8 Robot Vision

A typical application area for 3D vision is robot vision, i.e., whenever robots are equipped with cameras that supply information about the parts to manufacture. Such systems are also called "hand-eye systems" because the robotic "hand" is guided by mechanical "eyes".

In order to use the information extracted by the camera, it must be transformed into the coordinate system of the robot. Thus, besides calibrating the camera(s) you must also calibrate the hand-eye system, i.e., determine the transformation between camera and robot coordinates. The following sections explain how to perform this hand-eye calibration with HALCON.

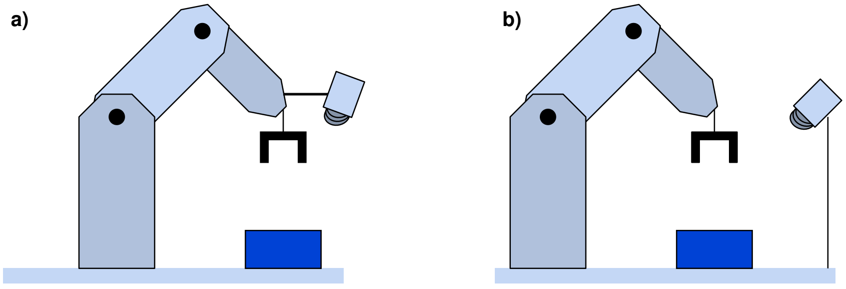


Figure 80: Robot vision scenarios: (a) moving camera, (b) stationary camera.

! Please note that in order to use HALCON's hand-eye calibration, **the camera must observe the workspace of the robot**. Figure 80 depicts the two possible scenarios: The camera can either be mounted on the robot and is moved to different positions by it, or it can be stationary. If the camera does not observe the workspace of the robot, e.g., if the camera observes parts on a conveyor belt, which are then handled by a robot further down the line, you must determine the relative pose of robot and camera with different means.

The calibration result can be used for different tasks. Typically, the results of machine vision, e.g., the position of a part, are to be transformed from camera into robot coordinates to create the appropriate robot commands, e.g., to grasp the part. Section 8.4 on page 131 describes such an application. Another possible application for hand-eye systems with a moving camera is to transform the information extracted from different camera poses into a common coordinate system.

Note that HALCON's hand-eye calibration is not restricted to systems with a "hand", i.e., a manipulator. You can also use it to calibrate cameras mounted on a pan-tilt head or surveillance cameras that rotate

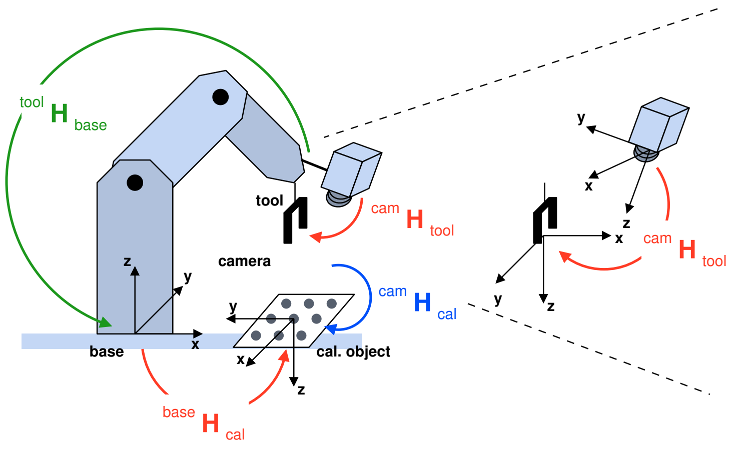


Figure 81: Chain of transformations for a moving camera system.

to observe a large area. Both systems correspond to a camera mounted on a robot; the calibration then allows you to accumulate visual information from different camera poses.

Further note that, although in the following only systems with a single camera are described, you can of course also use a stereo camera system. In this case, you typically calibrate only the relation of the robot to one of the cameras, because the relation between the cameras is determined by the stereo calibration (see section 7.3.3 on page 108).

8.1 The Principle of Hand-Eye Calibration

Like the camera calibration (see section 3.1 on page 30), the hand-eye calibration is based on providing multiple images of a known calibration object. But in contrast to the camera calibration, here the calibration object is not moved manually but by the robot, which moves either the calibration object in front of a stationary camera or the camera over a stationary calibration object. The pose, i.e., the position and orientation, of the robot in each calibration image **must be known with high accuracy!** !

This results in a chain of coordinate transformations (see figure 81 and figure 82). In this chain, two transformations (poses) are known: the robot pose ${}^{base}\mathbf{H}_{tool}$ and the pose of the calibration object in camera coordinates ${}^{cam}\mathbf{H}_{cal}$, which is determined from the calibration images before starting the hand-eye calibration. The hand-eye calibration then estimates the other two poses, i.e., the relation between the robot and the camera and between the robot and the calibration object, respectively.

Note that the chain consists of different poses depending on the used scenario. For a *moving camera*, the pose of the robot tool in camera coordinates and the pose of the calibration object in robot base coordinates are determined (see figure 81):

$$ {}^{cam}\mathbf{H}_{cal} = {}^{cam}\mathbf{H}_{tool} \cdot {}^{tool}\mathbf{H}_{base} \cdot {}^{base}\mathbf{H}_{cal} \tag{47}$$

Please beware that in this chain the inverse robot pose, i.e., the pose of the robot base in tool coordinates, is used.

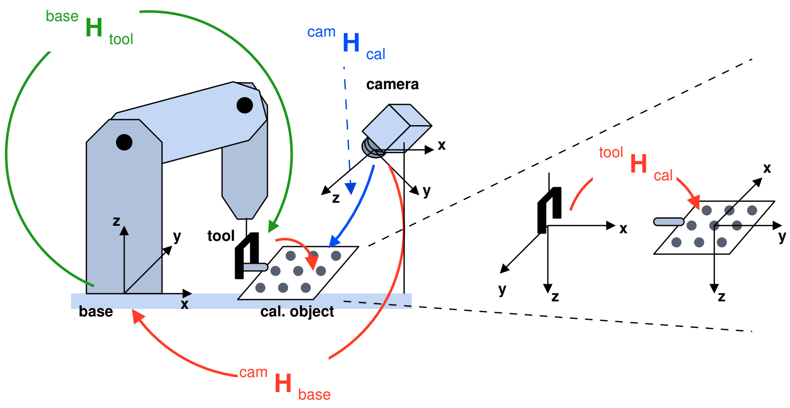


Figure 82: Chain of transformations for a stationary camera system.

For a *stationary camera*, the pose of the robot base in camera coordinates and of the calibration object in robot tool coordinates are determined (see figure 82):

$$^{cam}\mathbf{H}_{cal} = {}^{cam}\mathbf{H}_{base} \cdot {}^{base}\mathbf{H}_{tool} \cdot {}^{tool}\mathbf{H}_{cal} \tag{48}$$

The hand-eye calibration is performed with a single call to the operator `hand_eye_calibration`:

```
hand_eye_calibration (NX, NY, NZ, NRow, NCol, NumPoints, RobotPoses, \
                      CamParam, 'nonlinear', 'error_pose', CameraPose, \
                      CalibrationPose, PoseError)
```

Let's have a brief look at the parameters; the referenced sections contain more detailed information:

- `X`, `Y`, `Z`, `Row`, `Col`, `NumPoints` (see section 8.2.1)

  !

  As for the camera calibration, you must supply 3D model points and their corresponding image points. Note, however, that for the hand-eye calibration the **3D point coordinates must be supplied for each image**, together with the number of 3D points visible in each image. This requirement might seem tedious if you use the standard calibration plate, because then the same points are visible in each image, but it offers more flexibility for users of other calibration objects.

- `RobotPoses` (see section 8.2.2 on page 128)

  This parameter contains the poses of the robot in each calibration image.

- `CamParam`

  In this parameter you pass the internal camera parameters. In the HDevelop example programs `examples\solution_guide\3d_machine_vision\handeye_movingcam_calibration.dev` and `examples\solution_guide\3d_machine_vision\handeye_stationarycam_calibration.dev`, the camera is calibrated using the calibration images acquired for the hand-eye calibration. For detailed information about obtaining the internal camera parameters please refer to section 3.1.5 on page 39.

- `Method`, `QualityType`

These parameters allow you to choose between different methods for calibration and result assessment.

- CameraPose, CalibrationPose, Quality

  These are the calibration results and their quality assessment. How to use them in robot vision applications is described in section 8.4 on page 131.

Besides the coordinate systems described above, two others may be of interest in a robot vision application: First, sometimes results must be transformed into a reference (world) coordinate system. You can define such a coordinate system easily based on a calibration image. Secondly, especially if the robot system uses different tools (grippers), it might be useful to place the tool coordinate system used in the calibration at the mounting point of the tools and introduce additional coordinate systems at the gripper (tool center point). The example application in section 8.4 on page 131 shows how to handle both cases.

8.2 Determining Suitable Input Parameters

Below, we show how to determine values for the input parameters of hand_eye_calibration. The code examples stem from the HDevelop programs examples\solution_guide\3d_machine_vision\handeye_movingcam_calibration.dev and examples\solution_guide\3d_machine_vision\handeye_stationarycam_calibration.dev, which perform the calibration of hand-eye systems with a moving and stationary camera, respectively. The programs stop after processing each calibration image; press Run to continue.

8.2.1 Corresponding World and Image Points

As for the camera calibration, you must supply 3D model points and their corresponding image points (compare section 3.1.1 on page 32). First, we create empty tuples to accumulate data from all calibration images:

```
NRow := []
NCol := []
NX := []
NY := []
NZ := []
NumPoints := []
```

When using the standard calibration plate, the 3D model points, i.e., the 3D coordinates of the calibration marks, can be read from the description file:

```
caltab_points (CalTabFile, X, Y, Z)
```

In each calibration image, we then locate the calibration plate and extract the image coordinates of the calibration marks. Please note that for the hand-eye calibration **we strongly recommend to use the asymmetric calibration plate** introduced with HALCON 7.1 (see section 3.1.7 on page 46). If even only in a single calibration image the pose of the old, symmetric calibration plate is estimated wrongly because it is rotated by more than 90 degrees, the calibration will fail!

```
for i := 0 to NumImages-1 by 1
  read_image (Image, ImageNameStart+i$'02d')
  find_caltab (Image, Caltab, CalTabFile, SizeGauss, MarkThresh, \
               MinDiamMarks)
  find_marks_and_pose (Image, Caltab, CalTabFile, StartCamParam, \
                       StartThresh, DeltaThresh, MinThresh, Alpha, \
                       MinContLength, MaxDiamMarks, RCoord, CCoord, \
                       StartPose)
```

Finally, the corresponding coordinates are accumulated in the tuples:

```
  NRow := [NRow,RCoord]
  NCol := [NCol,CCoord]
  NX := [NX,X]
  NY := [NY,Y]
  NZ := [NZ,Z]
  NumPoints := [NumPoints,49]
endfor
```

Note that the 3D coordinates and number of marks per image are accumulated even if they don't change between images. As already explained, the possibility to use different model points in each image is not necessary when using the standard calibration plate, but can be very useful if you use your own calibration object, especially if it is a three-dimensional one.

8.2.2 Robot Poses

! For each of the calibration images, you must specify the corresponding pose of the robot. Note that **the accuracy of the poses is critical to obtain an accurate hand-eye calibration**. There are two ways to "feed" the poses into HALCON: In many cases, you will simply read them from the robot control unit and then enter them into your HALCON program manually. For this, you can use the HDevelop example program `examples\solution_guide\3d_machine_vision\handeye_create_robot_poses.dev`, which lets you input the poses in a text window and writes them into files.

As an alternative, if the robot has a serial interface, you can also send them via this connection to your HALCON program (see the section "System ▷ Serial" in the Reference Manual for information about communicating via the serial interface).

In both cases, you then convert the data into HALCON 3D poses using the operator `create_pose`. As described in section 2.1.5 on page 17 (and in the Reference Manual entry for `create_pose`), you can specify a pose in more than one way, because the orientation can be described by different sequences of rotations. Therefore, you must first check which sequence is used by your robot system. In many cases, it will correspond to

$$\mathbf{R}_{abg} = \mathbf{R}_z(\texttt{RotZ}) \cdot \mathbf{R}_y(\texttt{RotY}) \cdot \mathbf{R}_x(\texttt{RotX}) \qquad (49)$$

If this is the case, select the value *'abg'* for the parameter `OrderOfRotation` of `create_pose`. For the inverse order, select *'gba'*.

If your robot system uses yet another sequence, you cannot use create_pose but must create a corresponding homogeneous transformation matrix and convert it into a pose using hom_mat3d_to_pose. If, e.g., your robot system uses the following sequence of rotations where the rotations are performed around the z-axis, then around the y-axis, and finally again around the z-axis

$$\mathbf{R}_{zyz} = \mathbf{R}_z(Rl) \cdot \mathbf{R}_y(Rm) \cdot \mathbf{R}_z(Rr) \qquad (50)$$

the pose can be created with the following code:

```
hom_mat3d_identity (HomMat3DIdentity)
hom_mat3d_translate (HomMat3DIdentity, Tx, Ty, Tz, HomMat3DTranslate)
hom_mat3d_rotate_local (HomMat3DTranslate, rad(Rl), 'z', HomMat3DT_Rl)
hom_mat3d_rotate_local (HomMat3DT_Rl, rad(Rm), 'y', HomMat3DT_Rl_Rm)
hom_mat3d_rotate_local (HomMat3DT_Rl_Rm, rad(Rr), 'z', HomMat3D)
hom_mat3d_to_pose (HomMat3D, Pose)
```

Note that the rotation operators expect angles to be given in radians, whereas create_pose expects them in degrees!

The example program examples\solution_guide\3d_machine_vision\handeye_create_robot_poses.dev allows you to enter poses of the three types described above. If your robot system uses yet another sequence of rotations, you can easily extend the program by modifying (or copying and adapting) the code for ZYZ poses.

The HDevelop example programs examples\solution_guide\3d_machine_vision\handeye_movingcam_calibration.dev and examples\solution_guide\3d_machine_vision\handeye_stationarycam_calibration.dev read the robot pose files in the loop of processing the calibration images. Before this, an empty tuple is created to accumulate the poses:

```
RobotPoses := []
```

For each calibration image, the pose of the robot is read from file using read_pose and accumulated in the tuple RobotPoses:

```
read_pose (DataNameStart+'robot_pose_'+i$'02d'+'.dat', TmpRobotPose)
RobotPoses := [RobotPoses,TmpRobotPose]
```

If you are using a hand-eye system with a moving camera, you must invert the pose (compare the chain of transformations in figure 81 on page 125):

```
hom_mat3d_invert (base_H_tool, tool_H_base)
hom_mat3d_to_pose (tool_H_base, RobotPoseInverse)
RobotPoses := [RobotPoses,RobotPoseInverse]
```

8.3 Performing the Calibration

Similar to the camera calibration, the main effort lies in collecting the input data. The calibration itself is performed with a single operator call:

```
hand_eye_calibration (NX, NY, NZ, NRow, NCol, NumPoints, RobotPoses, \
                      CamParam, 'nonlinear', 'error_pose', CameraPose, \
                      CalibrationPose, PoseError)
```

Typically, you then save the calibrated poses in files so that your robot vision application can read them at a later time. The following code does so for a system with a moving camera:

```
write_pose (CameraPose, DataNameStart+'final_pose_cam_tool.dat')
write_pose (CalibrationPose, DataNameStart+'final_pose_base_calplate.dat')
```

Of course, you should check whether the calibration was successful by looking at the output parameter `Quality`, which is a measure for the accuracy of the pose parameters. With the parameter `QualityType`, you can specify the type of quality measures (note that not all types are possible for all calibration methods selected with the parameter `Method`):

`'error_pose'` returns the pose error of the complete chain of transformations in form of a tuple with four elements:

- the root-mean-square error of the translational part
- the root-mean-square error of the rotational part
- the maximum translational error
- the maximum rotational error

`'standard_deviation'` returns the standard deviations of the two poses in a tuple with 12 elements The first six elements refer to the camera pose and the others to the pose of the calibration points.

`'covariance'` returns the full 12x12 covariance matrix of both poses.

In the example, the pose error values are returned and displayed.

```
disp_message (WindowHandle, \
              'Quality of the results:  root mean square    maximum', \
              'window', 40, -1, 'black', 'true')
disp_message (WindowHandle, \
              '  translation part:       ' + PoseError[0] + '      ' + PoseError[2], \
              'window', 60, -1, 'black', 'true')
disp_message (WindowHandle, \
              '  rotation part:          ' + PoseError[1] + '         ' + PoseError[3], \
              'window', 80, -1, 'black', 'true')
```

The example programs then visualize the calibrated poses by displaying the coordinate system of the calibration plate in each calibration image. For this, they compute the pose of the calibration plate in camera coordinates based on the calibrated poses. For a moving camera system, this corresponds to the following code (compare equation 47 on page 125):

```
* CalplatePose = cam_H_calplate = cam_H_tool * tool_H_base *  \
* base_H_calplate
pose_to_hom_mat3d (CalibrationPose, base_H_calplate)
pose_to_hom_mat3d (CameraPose, cam_H_tool)
pose_to_hom_mat3d (RobotPoseInverse, tool_H_base)
hom_mat3d_compose (cam_H_tool, tool_H_base, cam_H_base)
hom_mat3d_compose (cam_H_base, base_H_calplate, cam_H_calplate)
hom_mat3d_to_pose (cam_H_calplate, CalplatePose)
```

This code is encapsulated in a procedure, which is called in a loop over all images:

```
for i := 0 to NumImages-1 by 1
  read_image (Image, ImageNameStart+i$'02d')
  TmpRobotPoseInverse := RobotPoses[i*7:i*7+6]
  calc_calplate_pose_movingcam (CalibrationPose, CameraPose, \
                                TmpRobotPoseInverse, TmpCalplatePose)
  display_calplate_coordinate_system (CalTabFile, TmpCalplatePose, CamParam, \
                                      WindowHandle)
endfor
```

The corresponding procedure for a stationary camera system is listed in appendix B.9 on page 172.

8.4 Using the Calibration Data

Typically, you use the result of the hand-eye calibration to **transform the results of machine vision from camera coordinates into robot coordinates** (${}^{cam}\mathbf{H}_{obj} \rightarrow {}^{base}\mathbf{H}_{obj}$) to generate the appropriate robot commands, e.g., to grasp an object whose position has been determined in an image as in the application described in section 8.4.3 on page 133. For a stationary camera, this transformation corresponds to the following equation (compare figure 82 on page 126):

$$ {}^{base}\mathbf{H}_{obj} = {}^{base}\mathbf{H}_{cam} \cdot {}^{cam}\mathbf{H}_{obj} = (\texttt{CameraPose})^{-1} \cdot {}^{cam}\mathbf{H}_{obj} \qquad (51) $$

For a moving camera system, the equation also contains the pose of the robot when acquiring the image of the object ${}^{base}\mathbf{H}_{tool}$(acq. pos.) (compare figure 81 on page 125):

$$ \begin{aligned} {}^{base}\mathbf{H}_{obj} &= {}^{base}\mathbf{H}_{tool}(\text{acq. pos.}) \cdot {}^{tool}\mathbf{H}_{cam} \cdot {}^{cam}\mathbf{H}_{obj} \\ &= {}^{base}\mathbf{H}_{tool}(\text{acq. pos.}) \cdot (\texttt{CameraPose})^{-1} \cdot {}^{cam}\mathbf{H}_{obj} \qquad (52) \end{aligned} $$

8.4.1 Using the Hand-Eye Calibration for Grasping (3D Alignment)

Grasping an object corresponds to a very simple equation that says “move the robot gripper to the pose of the object” (“grasping pose”). This is also called 3D alignment.

Note that if the tool coordinate system used during hand-eye calibration is not placed at the gripper (tool center point), the equation also contains the transformation between tool and gripper coordinate system.

This transformation cannot be calibrated with the hand-eye calibration but must be measured or taken from the CAD model or technical drawing of the gripper.

$$\begin{array}{lrcl} \text{tool} = \text{gripper:} & {}^{base}\mathbf{H}_{tool}(\text{gr. pos.}) & = & {}^{base}\mathbf{H}_{obj} \\ \text{tool} \neq \text{gripper:} & {}^{base}\mathbf{H}_{tool}(\text{gr. pos.}) & \cdot & {}^{tool}\mathbf{H}_{gripper} = {}^{base}\mathbf{H}_{obj} \\ & {}^{base}\mathbf{H}_{tool}(\text{gr. pos.}) & = & {}^{base}\mathbf{H}_{obj} \cdot ({}^{tool}\mathbf{H}_{gripper})^{-1} \end{array} \tag{53}$$

If we replace ${}^{base}\mathbf{H}_{obj}$ according to equation 51 on page 131 and equation 52 on page 131, we get the "grasping equation" for a **stationary camera**

$${}^{base}\mathbf{H}_{tool}(\text{gr. pos.}) = (\texttt{CameraPose})^{-1} \cdot {}^{cam}\mathbf{H}_{obj} \left[\cdot ({}^{tool}\mathbf{H}_{gripper})^{-1} \right] \tag{54}$$

and for a **moving camera**

$${}^{base}\mathbf{H}_{tool}(\text{gr. pos.}) = {}^{base}\mathbf{H}_{tool}(\text{acq. pos.}) \cdot (\texttt{CameraPose})^{-1} \cdot {}^{cam}\mathbf{H}_{obj} \left[\cdot ({}^{tool}\mathbf{H}_{gripper})^{-1} \right] \tag{55}$$

The notation $\left[\cdot ({}^{tool}\mathbf{H}_{gripper})^{-1} \right]$ indicates that this part is only necessary if the tool coordinate system is not identical with the gripper coordinate system.

8.4.2 How to Get the 3D Pose of the Object

The 3D pose of the object in camera coordinates (${}^{cam}\mathbf{H}_{obj}$) can stem from different sources:

- With a **binocular stereo system**, you can determine the 3D pose of unknown objects directly (see section 7 on page 100).
- For single camera systems, HALCON provides multiple methods. The most powerful one is **3D Matching** (see the Solution Guide I, chapter 10 on page 125), which performs a full object recognition, i.e., it not only estimates the pose but first locates the object in the image. If only one, planar side of the object is visible, fast alternatives are the calibrated perspective, deformable matching (section 6.1.2 on page 95) and the calibrated descriptor-based matching (section 6.1.3 on page 98).
- If a full object recognition is not necessary, you can use **pose estimation** to determine the 3D pose of known objects (see section 8.4.3 for an example application and section 6 on page 90 for more details on pose estimation).
- Finally, you can determine the 3D coordinates of unknown objects if **object points lie in a known plane** (see section 8.4.3 for an example application and section 3.2 on page 48 for more details on determining 3D points in a known plane).

Please note that if you want to use the 3D pose for grasping the object, the extracted pose, in particular the orientation, must be identical to the pose of the gripper at the grasping position.

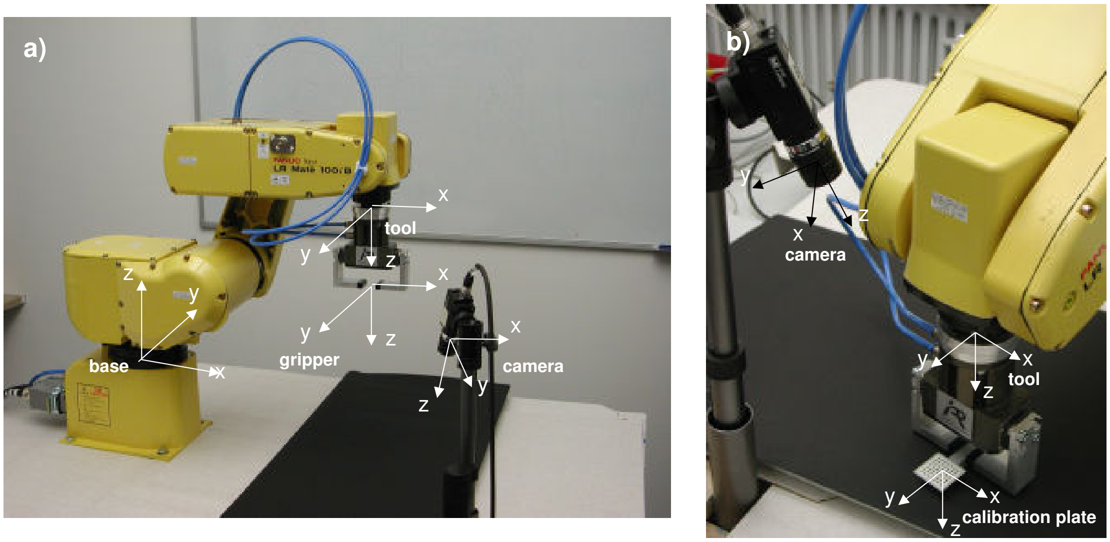


Figure 83: Example hand-eye system with a stationary camera: coordinate systems (a) of robot and camera, (b) with calibration plate.

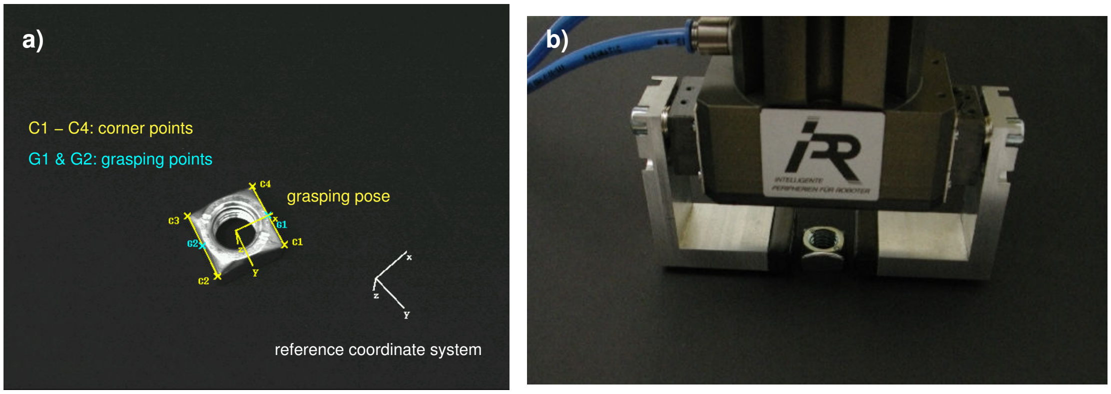


Figure 84: (a) Determining the 3D pose for grasping a nut; (b) robot at grasping pose.

8.4.3 Example Application with a Stationary Camera: Grasping a Nut

This section describes an example application realized with the hand-eye system depicted in figure 83. The task is to localize a nut and determine a suitable grasping pose for the robot (see figure 84). The HDevelop example program `examples\solution_guide\3d_machine_vision\handeye_stationarycam_grasp_nut.dev` performs the machine vision part and transforms the resulting pose into robot coordinates using the calibration data determined with `examples\solution_guide\3d_machine_vision\handeye_stationarycam_calibration.dev` as described in the previous sections. As you will see, using the calibration data is the shortest part of the program, its main part is devoted to machine vision.

Step 1: Read calibration data

First, the calibration data is read from files; for later computations, the poses are converted into homogeneous transformation matrices:

```
read_cam_par (DataNameStart+'final_campar.dat', CamParam)
read_pose (DataNameStart+'final_pose_cam_base.dat', PoseCamBase)
pose_to_hom_mat3d (PoseCamBase, cam_H_base)
read_pose (DataNameStart+'final_pose_tool_calplate.dat', PoseToolCalplate)
pose_to_hom_mat3d (PoseToolCalplate, tool_H_calplate)
```

In the used hand-eye system, the tool coordinate system used in the calibration process is located at the mounting point of the tool; therefore, an additional coordinate system is needed between the fingers of the gripper (see figure 83a on page 133). Its pose in tool coordinates is also read from file:

```
read_pose (DataNameStart+'pose_tool_gripper.dat', PoseToolGripper)
pose_to_hom_mat3d (PoseToolGripper, tool_H_gripper)
```

Step 2: Define reference coordinate system

Now, a reference coordinate system is defined based on one of the calibration images. In this image, the calibration plate has been placed into the plane on top of the nut. This allows to determine the 3D coordinates of extracted image points on the nut with a single camera and without knowing the dimensions of the nut. The code for defining the reference coordinate system is contained in a procedure, which is listed in appendix B.10 on page 173:

```
define_reference_coord_system (ImageNameStart+'calib3cm_00', CamParam, \
                               CalplateFile, WindowHandle, PoseRef)
pose_to_hom_mat3d (PoseRef, cam_H_ref)
```

Step 3: Extract grasping points on the nut

The following code extracts grasping points on two opposite sides of the nut. The nut is found with simple blob analysis; its boundary is converted into XLD contours:

```
threshold (Image, BrightRegion, 60, 255)
connection (BrightRegion, BrightRegions)
select_shape (BrightRegions, Nut, 'area', 'and', 500, 99999)
fill_up (Nut, NutFilled)
gen_contour_region_xld (NutFilled, NutContours, 'border')
```

The contours are then processed to find long, parallel straight line segments; their corners are accumulated in tuples:

```
segment_contours_xld (NutContours, LineSegments, 'lines', 5, 4, 2)
fit_line_contour_xld (LineSegments, 'tukey', -1, 0, 5, 2, RowBegin, \
                      ColBegin, RowEnd, ColEnd, Nr, Nc, Dist)
gen_empty_obj (Lines)
for i := 0 to |RowBegin| -1 by 1
  gen_contour_polygon_xld (Contour, [RowBegin[i],RowEnd[i]], [ColBegin[i], \
                          ColEnd[i]])
  concat_obj (Lines, Contour, Lines)
endfor
gen_polygons_xld (Lines, Polygon, 'ramer', 2)
gen_parallels_xld (Polygon, ParallelLines, 50, 100, rad(10), 'true')
get_parallels_xld (ParallelLines, Row1, Col1, Length1, Phi1, Row2, Col2, \
                   Length2, Phi2)
CornersRow := [Row1[0], Row1[1], Row2[0], Row2[1]]
CornersCol := [Col1[0], Col1[1], Col2[0], Col2[1]]
```

Step 4: Determine the grasping pose in camera coordinates

The grasping pose is calculated in 3D coordinates. For this, the 3D coordinates of the corner points in the reference coordinate system are determined using the operator image_points_to_world_plane. The origin of the grasping pose lies in the middle of the corners:

```
image_points_to_world_plane (CamParam, PoseRef, CornersRow, CornersCol, 'm', \
                             CornersX_ref, CornersY_ref)
CenterPointX_ref := sum(CornersX_ref)*0.25
CenterPointY_ref := sum(CornersY_ref)*0.25
```

The grasping pose is oriented almost like the reference coordinate system, only rotated around the z-axis so that it is identical to the gripper coordinate system, i.e., so that the gripper “fingers” are parallel to the sides of the nut. To calculate the rotation angle, first the grasping points in the middle of the sides are determined. Their angle can directly be used as the rotation angle:

```
GraspPointsX_ref := [(CornersX_ref[0]+CornersX_ref[1])*0.5, \
                      (CornersX_ref[2]+CornersX_ref[3])*0.5]
GraspPointsY_ref := [(CornersY_ref[0]+CornersY_ref[1])*0.5, \
                      (CornersY_ref[2]+CornersY_ref[3])*0.5]
GraspPhiZ_ref := atan((GraspPointsY_ref[1]-GraspPointsY_ref[0])/  \
                 (GraspPointsX_ref[1]-GraspPointsX_ref[0]))
```

With the origin and rotation angle, the grasping pose is first determined in the reference coordinate system and then transformed into camera coordinates:

```
hom_mat3d_identity (HomMat3DIdentity)
hom_mat3d_rotate (HomMat3DIdentity, GraspPhiZ_ref, 'z', 0, 0, 0, \
                  HomMat3D_RZ_Phi)
hom_mat3d_translate (HomMat3D_RZ_Phi, CenterPointX_ref, CenterPointY_ref, 0, \
                     ref_H_grasp)
hom_mat3d_compose (cam_H_ref, ref_H_grasp, cam_H_grasp)
```

Alternatively, the example also shows how to calculate the grasping pose using pose estimation (see section 6 on page 90 for a detailed description). This method can be used when points on the object are known. In the example, we specify the 3D coordinates of the corners of the nut:

```
NX := [0.009, -0.009, -0.009, 0.009]
NY := [0.009, 0.009, -0.009, -0.009]
```

The grasping pose is then calculated by simply calling the operator `camera_calibration`, using the reference coordinate system as the start pose. Before, however, the image coordinates of the corners must be sorted such that the first one lies close to the x-axis of the reference coordinate system. Otherwise, the orientation of the reference coordinate system would differ too much from the grasping pose and the pose estimation would fail.

```
sort_corner_points (CornersRow, CornersCol, WindowHandle, NRow, NCol)
camera_calibration (NX, NY, NZ, NRow, NCol, CamParam, PoseRef, 'pose', \
                    CamParam, PoseCamNut, Errors)
disp_3d_coord_system (WindowHandle, CamParam, PoseCamGripper, 0.01)
```

The result of both methods is displayed in figure 84a on page 133.

Step 5: Transform the grasping pose in robot coordinates

Now comes the moment to use the results of the hand-eye calibration: The grasping pose is transformed into robot coordinates with the formula shown in equation 54 on page 132:

```
hom_mat3d_invert (cam_H_base, base_H_cam)
hom_mat3d_compose (base_H_cam, cam_H_grasp, base_H_grasp)
```

As already mentioned, the tool coordinate system used in the calibration process is placed at the mounting point of the tool, not between the fingers of the gripper. Thus, the pose of the tool in gripper coordinates must be added to the chain of transformations to obtain the pose of the tool in base coordinates:

```
hom_mat3d_invert (tool_H_gripper, gripper_H_tool)
hom_mat3d_compose (base_H_grasp, gripper_H_tool, base_H_tool)
```

Step 6: Transform pose type

Finally, the pose is converted into the type used by the robot controller:

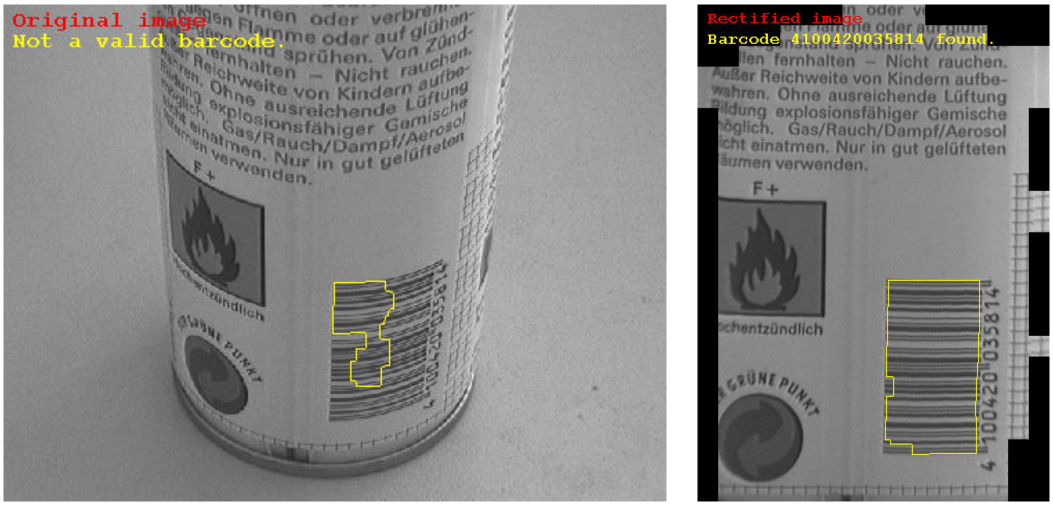
```
hom_mat3d_to_pose (base_H_tool, PoseRobotGrasp)
convert_pose_type (PoseRobotGrasp, 'Rp+T', 'abg', 'point', \
                   PoseRobotGrasp_ZYX)
```

Figure 84b on page 133 shows the robot at the grasping pose.

9 Rectification of Arbitrary Distortions

For many applications like OCR or bar code reading, distorted images must be rectified prior to the extraction of information. The distortions may be caused by the perspective projection and by the radial lens distortions as well as by the decentering lens distortions, a non-flat object surface, or by any other reason. In the first three cases, i.e., if the object surface is flat and the camera shows only radial or decentering distortions, the rectification can be carried out very precisely as described in section 3.3.1 on page 54. For the remaining cases, a piecewise bilinear rectification can be carried out. In HALCON, this kind of rectification is called grid rectification.

The following example (`examples\solution_guide\3d_machine_vision\grid_rectification_ruled_surface.dev`) shows how the grid rectification can be used to rectify the image of a cylindrically shaped object (figure 85). In the rectified image (figure 85b), the bar code could be read correctly, which was not possible in the original image (figure 85a). Note that since HALCON 8.0, the bar code reader can manage also the bar code in the original image, but for even more distorted bar codes nevertheless a rectification can be necessary.



a) b)

Figure 85: Cylindrical object: a) Original image; b) rectified image.

The main idea of the grid rectification is that the mapping for the rectification is determined from an image of the object, where the object is covered by a known pattern.

First, this pattern, which is called rectification grid, must be created with the operator `create_rectification_grid`.

```
create_rectification_grid (WidthOfGrid, NumSquares, 'rectification_grid.ps')
```

The resulting PostScript file must be printed. An example for such a rectification grid is shown in figure 86a.

Now, the object must be wrapped with the *rectification grid* and an image of the wrapped object must be taken (figure 86b).

From this image, the mapping that describes the transformation from the distorted image into the rectified image can be derived. For this, first, the rectification grid must be extracted. Then, the rectification map is derived from the distorted grid. This can be achieved by the following lines of code:

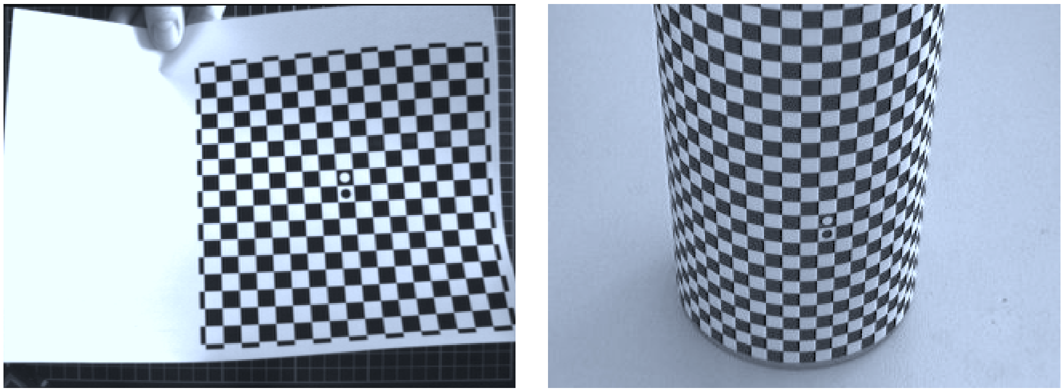


Figure 86: a) Example of a rectification grid. b) Cylindrical object wrapped with the rectification grid.

```
find_rectification_grid (Image, GridRegion, MinContrast, Radius)
reduce_domain (Image, GridRegion, ImageReduced)
saddle_points_sub_pix (ImageReduced, 'facet', SigmaSaddlePoints, Threshold, \
                       Row, Col)
connect_grid_points (ImageReduced, ConnectingLines, Row, Col, \
                     SigmaConnectGridPoints, MaxDist)
gen_grid_rectification_map (ImageReduced, ConnectingLines, Map, Meshes, \
                            GridSpacing, 0, Row, Col)
```

Using the derived map, any image that shows the same distortions can be rectified such that the parts that were covered by the rectification grid appear undistorted in the rectified image (figure 85b). This mapping is performed by the operator `map_image`.

```
map_image (ImageReduced, Map, ImageMapped)
```

In the following section, the basic principle of the grid rectification is described. Then, some hints for taking images of the rectification grid are given. In section 9.3 on page 142, the use of the involved HALCON operators is described in more detail based on the above example application. Finally, it is described briefly how to use self-defined grids for the generation of rectification maps.

9.1 Basic Principle

The basic principle of the grid rectification is that a mapping from the distorted image into the rectified image is determined from a distorted image of the rectification grid whose undistorted geometry is well known: The black and white fields of the printed rectification grid are squares (figure 87).

In the distorted image, the black and white fields do not appear as squares (figure 88a) because of the non-planar object surface, the perspective distortions, and the lens distortions.

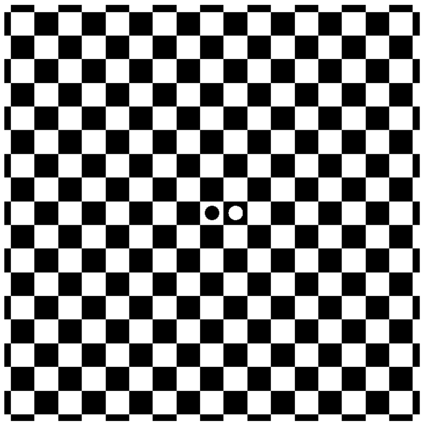


Figure 87: Rectification grid.

To determine the mapping for the rectification of the distorted image, the distorted rectification grid must be extracted. For this, first, the corners of the black and white fields must be extracted with the operator saddle_points_sub_pix (figure 88b). These corners must be connected along the borders of the black and white fields with the operator connect_grid_points (figure 88c). Finally, the connecting lines must be combined into meshes (figure 88d) with the operator gen_grid_rectification_map, which also determines the mapping for the rectification of the distorted image.

If you want to use a self-defined grid, the grid points must be defined by yourself. Then, the operator gen_arbitrary_distortion_map can be used to determine the mapping (see section 9.4 on page 144 for an example).

The mapping is determined such that the distorted rectification grid will be mapped into its original undistorted geometry (figure 89). With this mapping, any image that shows the same distortions can be rectified easily with the operator map_image. Note that within the meshes a bilinear interpolation is carried out. Therefore, it is important to use a rectification grid with an appropriate grid size (see section 9.2 for details).

9.2 Rules for Taking Images of the Rectification Grid

If you want to achieve accurate results, please follow the rules given in this section:

- The image must not be overexposed or underexposed: otherwise, the extraction of the corners of the black and white fields of the rectification grid may fail.
- The contrast between the bright and the dark fields of the rectification grid should be as high as possible.
- Ensure that the rectification grid is homogeneously illuminated.
- The images should contain as little noise as possible.

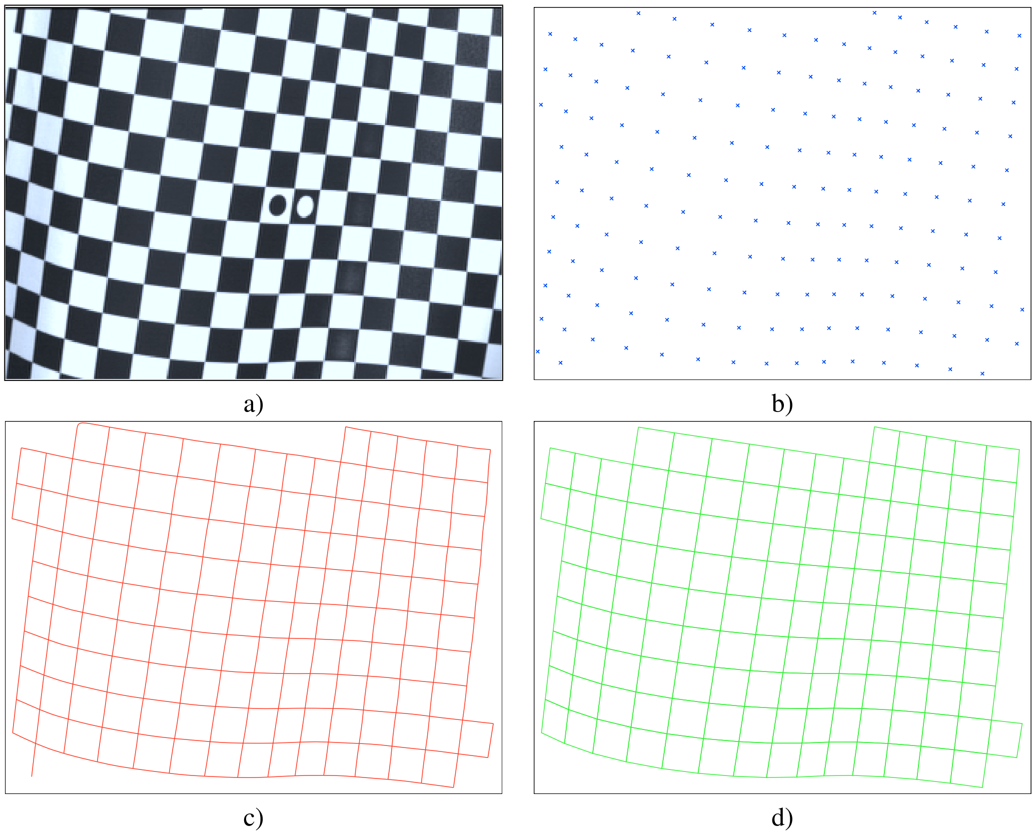


Figure 88: Distorted rectification grid: a) Image; b) extracted corners of the black and white fields; c) lines that connect the corners; d) extracted rectification grid.

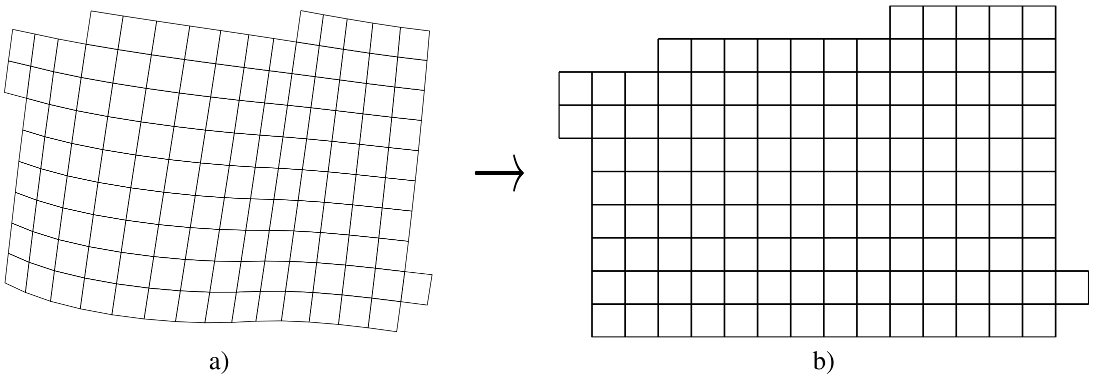


Figure 89: Mapping of the distorted rectification grid (a) into the undistorted rectification grid (b).

* The border length of the black and white fields should be at least 10 pixels.

In addition to these few rules for the taking of the images of the rectification grid, it is very important to use a rectification grid with an appropriate grid size because the mapping is determined such that within the meshes of the rectification grid a bilinear interpolation is applied. Because of this, non-linear distortions within the meshes cannot be eliminated.

The use of a rectification grid that is too coarse (figure 90a), i.e., whose grid size is too large, leads to errors in the rectified image (figure 90b).

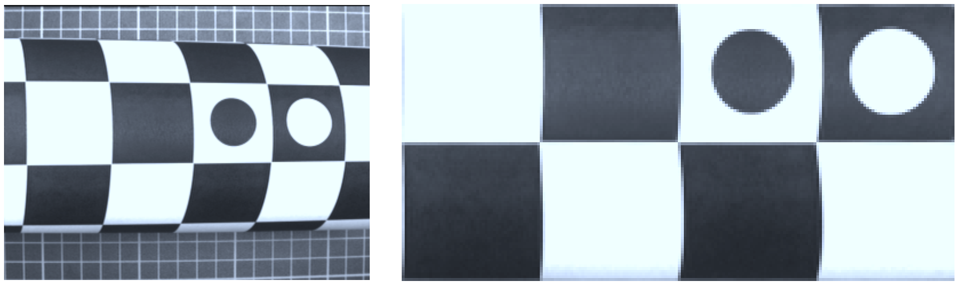


Figure 90: Cylindrical object covered with a very coarse rectification grid: a) Distorted image; b) rectified image.

If it is necessary to fold the rectification grid, it should be folded along the borders of the black and white fields. Otherwise, i.e., if the fold crosses these fields (figure 91a), the rectified image (figure 91b) will contain distortions because of the bilinear interpolation within the meshes.

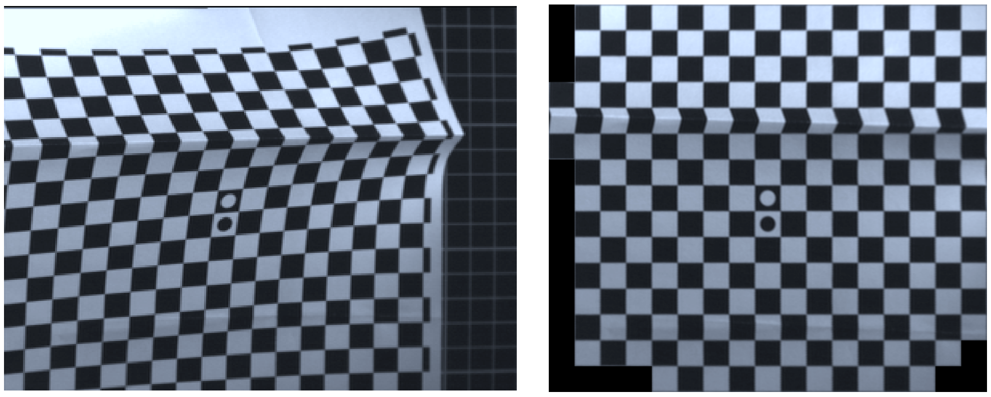


Figure 91: Rectification grid folded across the borders of the black and white fields: a) Distorted image; b) rectified image.

9.3 Machine Vision on Ruled Surfaces

In this section, the rectification of images of ruled surfaces is described in detail. Again, the example of the cylindrically shaped object (`examples\solution_guide\3d_machine_vision\grid_rectification_ruled_surface.dev`) is used to explain the involved operators.

First, the operator `create_rectification_grid` is used to create a suitable rectification grid.

```
create_rectification_grid (WidthOfGrid, NumSquares, 'rectification_grid.ps')
```

The parameter `WidthOfGrid` defines the effectively usable size of the rectification grid in meters and the parameter `NumSquares` sets the number of squares (black and white fields) per row. The rectification grid is written to the PostScript file that is specified by the parameter `GridFile`.

To determine the mapping, an image of the rectification grid, wrapped around the object, must be taken as described in section 9.2 on page 139. Figure 92a shows an image of a cylindrical object and figure 92b shows the same object wrapped by the rectification grid.

Then, the rectification grid is searched in this image with the operator `find_rectification_grid`.

```
find_rectification_grid (Image, GridRegion, MinContrast, Radius)
```

The operator `find_rectification_grid` extracts image areas with a contrast of at least `MinContrast` and fills up the holes in these areas. Note that in this case, contrast is defined as the gray value difference of neighboring pixels in a slightly smoothed copy of the image (Gaussian smoothing with $\sigma = 1.0$). Therefore, the value for the parameter `MinContrast` must be set significantly lower than the gray value difference between the black and white fields of the rectification grid. Small areas of high contrast are then eliminated by an opening with the radius `Radius`. The resulting region is used to restrict the search space for the following steps with the operator `reduce_domain` (see figure 93a).

```
reduce_domain (Image, GridRegion, ImageReduced)
```

The corners of the black and white fields appear as saddle points in the image. They can be extracted with the operator `saddle_points_sub_pix` (see figure 93b).

```
saddle_points_sub_pix (ImageReduced, 'facet', SigmaSaddlePoints, Threshold, \
                       Row, Col)
```

The parameter `Sigma` controls the amount of Gaussian smoothing that is carried out before the actual extraction of the saddle points. Which point is accepted as a saddle point is based on the value of the parameter `Threshold`. If `Threshold` is set to higher values, fewer but more distinct saddle points are returned than if `Threshold` is set to lower values. The filter method that is used for the extraction of the saddle points can be selected by the parameter `Filter`. It can be set to *'facet'* or *'gauss'*. The method *'facet'* is slightly faster. The method *'gauss'* is slightly more accurate but tends to be more sensitive to noise.

To generate a representation of the distorted rectification grid, the extracted saddle points must be connected along the borders of the black and white fields (figure 93c). This is done with the operator `connect_grid_points`.

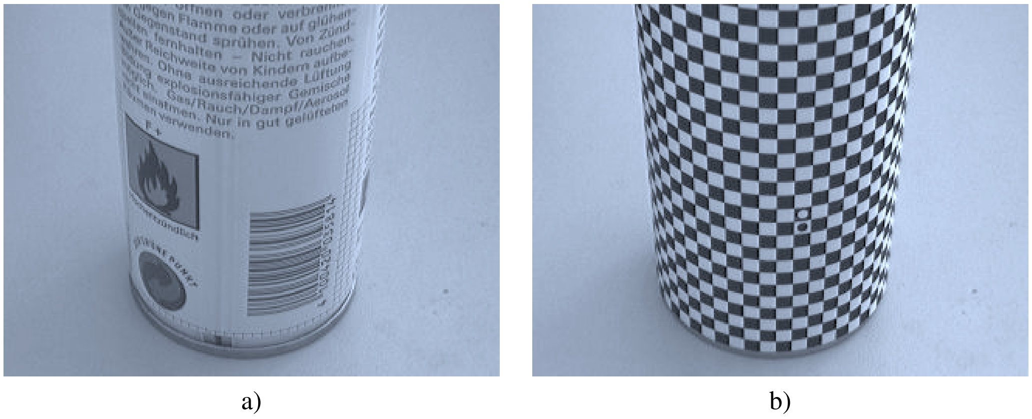


Figure 92: Cylindrical object: a) Without and b) with rectification grid.

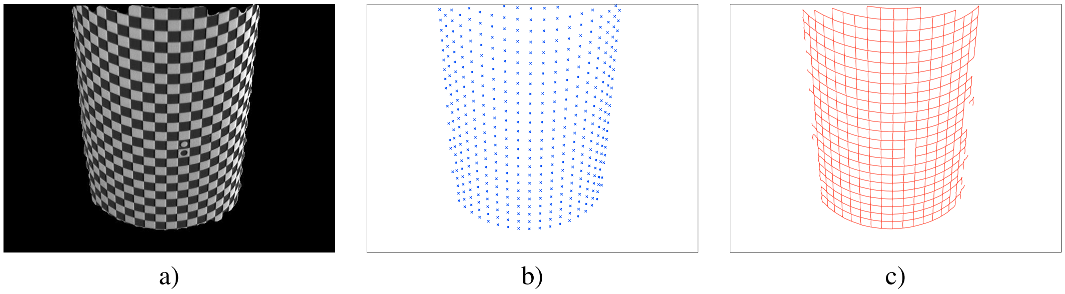


Figure 93: Distorted rectification grid: a) Image reduced to the extracted area of the rectification grid; b) extracted corners of the black and white fields; c) lines that connect the corners.

```
connect_grid_points (ImageReduced, ConnectingLines, Row, Col, \
                     SigmaConnectGridPoints, MaxDist)
```

Again, the parameter Sigma controls the amount of Gaussian smoothing that is carried out before the extraction of the borders of the black and white fields. When a tuple of three values [*sigma_min*, *sigma_max*, *sigma_step*] is passed instead of only one value, the operator connect_grid_points tests every sigma within the given range from *sigma_min* to *sigma_max* with a step size of *sigma_step* and chooses the sigma that causes the largest number of connecting lines. The same happens when a tuple of only two values *sigma_min* and *sigma_max* is passed. However, in this case a fixed step size of 0.05 is used. The parameter MaxDist defines the maximum distance with which an edge may be linked to the respectively closest saddle point. This helps to overcome the problem that edge detectors typically return inaccurate results in the proximity of edge junctions. Figure 94 shows the connecting lines if the parameter MaxDist has been selected inappropriately: In figure 94a, MaxDist has been selected to small, whereas in figure 94b, it has been selected too large.

Then, the rectification map is determined from the distorted grid with the operator gen_grid_rectification_map.

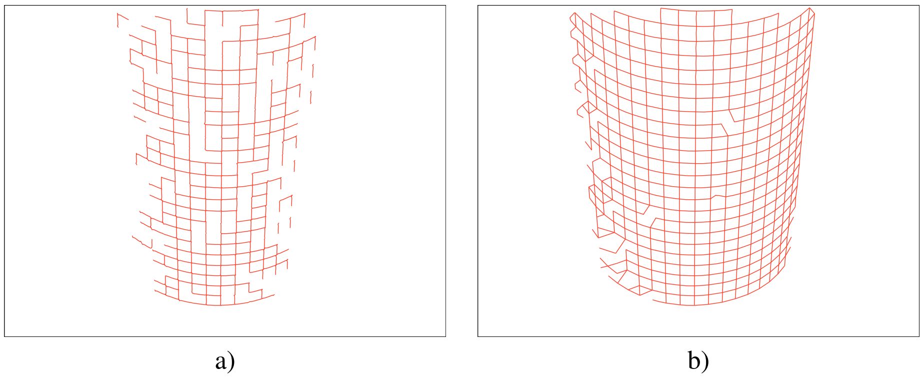


Figure 94: Connecting lines: Parameter MaxDist selected a) too small and b) too large.

```
gen_grid_rectification_map (ImageReduced, ConnectingLines, Map, Meshes, \
                            GridSpacing, 0, Row, Col)
```

The parameter GridSpacing defines the size of the grid meshes in the rectified image. Each of the black and white fields is projected onto a square of GridSpacing $\times$ GridSpacing pixels. The parameter Rotation controls the orientation of the rectified image. The rectified image can be rotated by *0*, *90*, *180*, or *270* degrees, or it is rotated such that the black circular mark is left of the white circular mark if Rotation is set to *'auto'*.

Using the derived rectification map, any image that shows the same distortions can be rectified very fast with the operator map_image (see figure 95). Note that the objects must appears at exactly the same position in the distorted images.

```
map_image (ImageReduced, Map, ImageMapped)
```

9.4 Using Self-Defined Rectification Grids

Up to now, we have used the predefined rectification grid together with the appropriate operators for its segmentation. In this section, an alternative to this approach is presented. You can arbitrarily define the rectification grid by yourself, but note that in this case you must also carry out the segmentation by yourself.

This example shows how the grid rectification can be used to generate arbitrary distortion maps based on self-defined grids.

The example application is a print inspection. It is assumed that some parts are missing and that smudges are present. In addition, lines may be vertically shifted, e.g., due to an inaccurate paper transport, i.e., distortions in the vertical direction of the printed document may be present. These distortions should not result in a rejection of the tested document. Therefore, it is not possible to simply compute the difference image between a reference image and the image that must be checked.

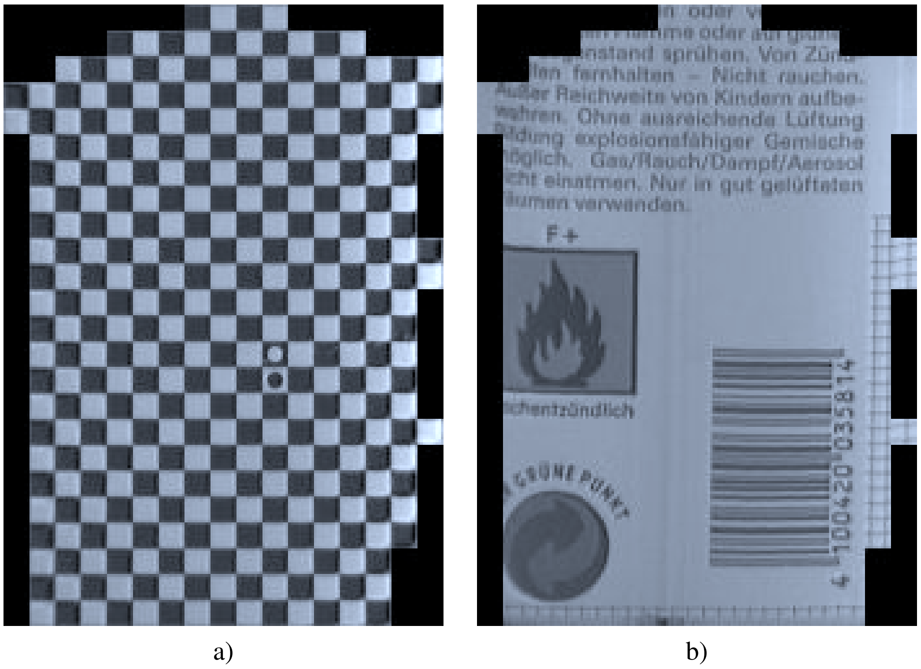


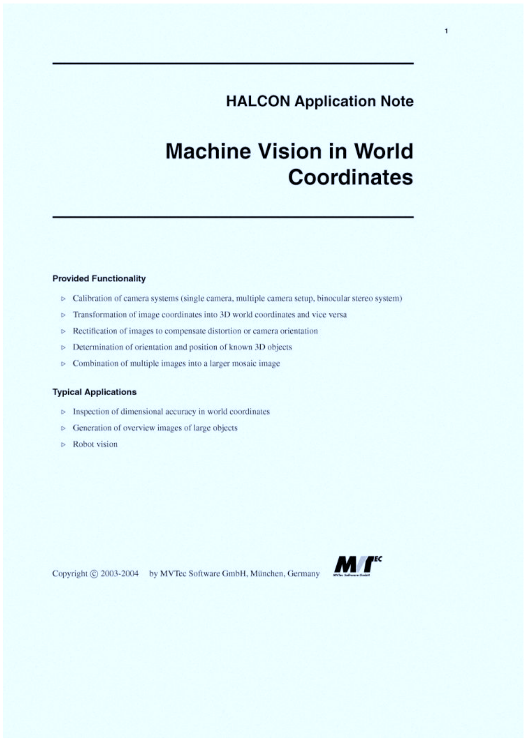
Figure 95: Rectified images: a) Rectification grid; b) object.

Figure 96a shows the reference image and figure 96b the test image that must be checked.

In a first step, the displacements between the lines in the reference document and the test document are determined. With this, the rectification grid is defined. The resulting rectification map is applied to the reference image to transform it into the geometry of the test image. Now, the difference image of the mapped reference image and the test image can be computed.

The example program (`examples\solution_guide\3d_machine_vision\grid_rectification_arbitrary_distortion.dev`) uses the component-based matching to determine corresponding points in the reference image and the test image. First, the component model is generated with the operator `create_component_model`. Then, the corresponding points are searched in the test image with the operator `find_component_model`.

Based on the corresponding points of the reference and the test image (`RowRef`, `ColRef`, `RowTest`, and `ColTest`), the coordinates of the grid points of the distorted grid are determined. In this example, the row and column coordinates can be determined independently from each other because only the row coordinates are distorted. Note that the upper left grid point of the undistorted grid is assumed to have the coordinates *(-0.5, -0.5)*. This means that the corresponding grid point of the distorted grid will be mapped to the point *(-0.5, -0.5)*. Because there are only vertical distortions in this example, the column coordinates of the distorted grid are equidistant, starting at the value *-0.5*.



1

HALCON Application Note

Machine Vision in World Coordinates

Provided Functionality

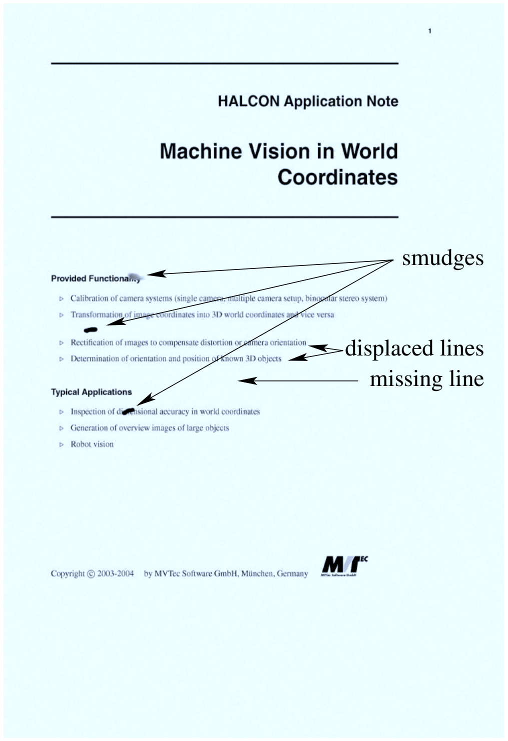
- ▷ Calibration of camera systems (single camera, multiple camera setup, binocular stereo system)
- ▷ Transformation of image coordinates into 3D world coordinates and vice versa
- ▷ Rectification of images to compensate distortion or camera orientation
- ▷ Determination of orientation and position of known 3D objects
- ▷ Combination of multiple images into a larger mosaic image

Typical Applications

- ▷ Inspection of dimensional accuracy in world coordinates
- ▷ Generation of overview images of large objects
- ▷ Robot vision

Copyright © 2003-2004 by MVTec Software GmbH, München, Germany

a)



1

HALCON Application Note

Machine Vision in World Coordinates

Provided Functionality

- ▷ Calibration of camera systems (single camera, multiple camera setup, binocular stereo system)
- ▷ Transformation of image coordinates into 3D world coordinates and vice versa
- ▷ Rectification of images to compensate distortion or camera orientation
- ▷ Determination of orientation and position of known 3D objects

Typical Applications

- ▷ Inspection of dimensional accuracy in world coordinates
- ▷ Generation of overview images of large objects
- ▷ Robot vision

Copyright © 2003-2004 by MVTec Software GmbH, München, Germany

b)

Figure 96: Images of one page of a document: a) Reference image; b) test image that must be checked.

```
GridSpacing := 10
ColShift := mean(ColTest-ColRef)
RefGridColValues := []
for HelpCol := -0.5 to WidthTest+GridSpacing by GridSpacing
  RefGridColValues := [RefGridColValues, HelpCol+ColShift]
endfor
```

The row coordinates of the distorted grid are determined by a linear interpolation between the above determined pairs of corresponding row coordinates.

```
MinValue := 0
MaxValue := HeightTest+GridSpacing
sample_corresponding_values (RowTest, RowRef-0.5, MinValue, MaxValue, \
                             GridSpacing, RefGridRowValues)
```

The interpolation is performed within the procedure

```
procedure sample_corresponding_values (: : Values, CorrespondingValues,
                                       MinValue, MaxValue,
                                       InterpolationInterval:
                                       SampledCorrespondingValues)
```

which is part of the example program `examples\solution_guide\3d_machine_vision\grid_rectification_arbitrary_distortion.dev`.

Now, the distorted grid is generated row by row.

```
RefGridRow := []
RefGridCol := []
Ones := gen_tuple_const(|RefGridColValues|, 1)
for r := 0 to |RefGridRowValues|-1 by 1
  RefGridRow := [RefGridRow, RefGridRowValues[r]*Ones]
  RefGridCol := [RefGridCol, RefGridColValues]
endfor
```

The operator `gen_arbitrary_distortion_map` uses this distorted grid to derive the rectification map that maps the reference image into the geometry of the test image[2].

```
gen_arbitrary_distortion_map (Map, GridSpacing, RefGridRow, RefGridCol, \
                              |RefGridColValues|, WidthRef, HeightRef)
```

With this rectification map, the reference image can be transformed into the geometry of the test image. Note that the size of the mapped image depends on the number of grid cells and on the size of one grid cell, which must be defined by the parameter `GridSpacing`. Possibly, the size of the mapped reference image must be adapted to the size of the test image.

```
map_image (ImageRef, Map, ImageMapped)
crop_part (ImageMapped, ImagePart, 0, 0, WidthTest, HeightTest)
```

Finally, the test image can be subtracted from the mapped reference image.

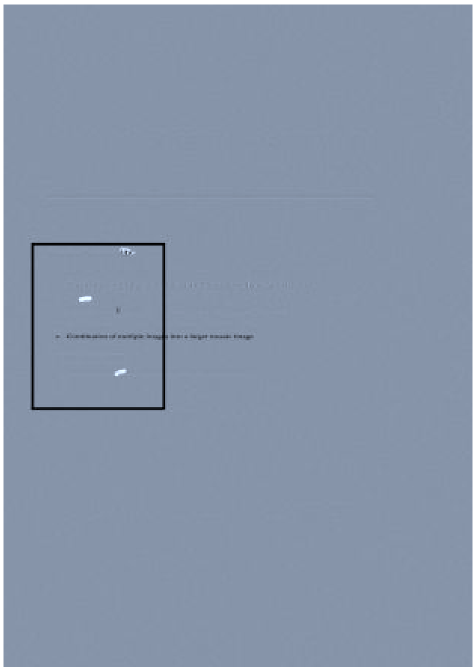
```
sub_image (ImagePart, ImageTest, ImageSub, 1, 128)
```

Figure 97 shows the resulting difference image. In this case, missing parts appear dark while the smudges appear bright.

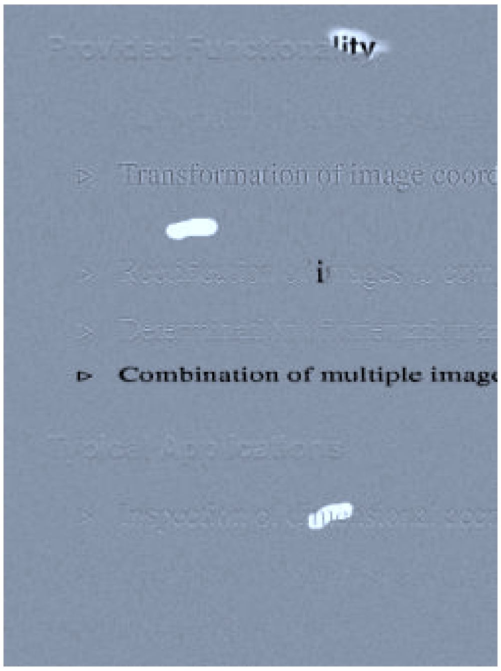
The differences between the test image and the reference image can now be extracted easily from the difference image with the operator `threshold`. If the difference image is not needed, e.g., for visualization purposes, the differences can be derived directly from the test image and the reference image with the operator `dyn_threshold`.

Figure 98 shows the differences in a cut-out of the reference image (figure 98a) and of the test image (figure 98b). The markers near the left border of figure 98b indicate the vertical position of the components

[2]In this case, the reference image is mapped into the geometry of the test image to facilitate the marking of the differences in the test image. Obviously, the rectification grid can also be defined such that the test image is mapped into the geometry of the reference image.



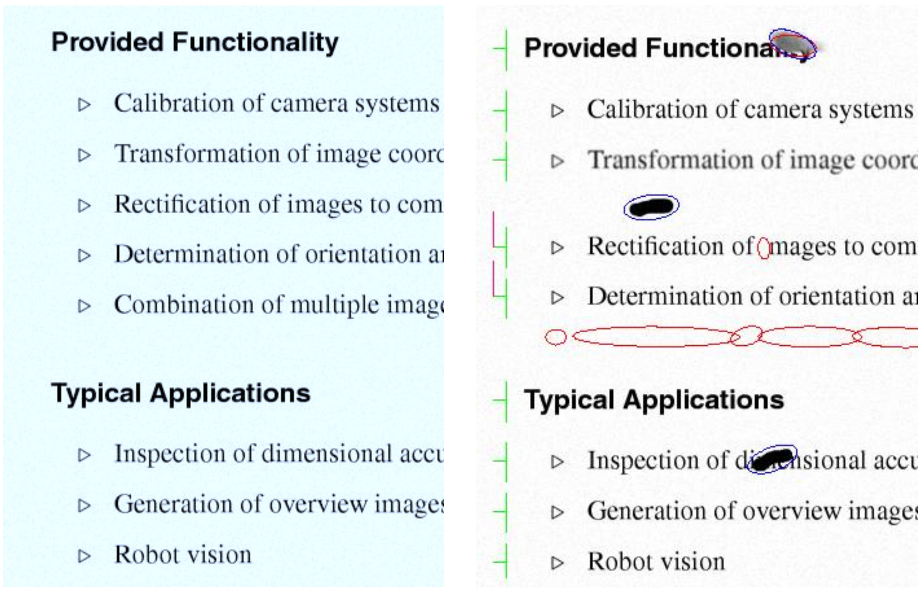
a)



b)

Figure 97: Difference image: a) The entire image overlaid with a rectangle that indicates the position of the cut-out. b) A cut-out.

that were used for the determination of the corresponding points. Vertical shifts of the components with respect to the reference image are indicated by a vertical line of the respective length that is attached to the respective marker. All other differences that could be detected between the test image and the reference image are encircled.


Provided Functionality
▷ Calibration of camera systems
▷ Transformation of image coorc
▷ Rectification of images to com
▷ Determination of orientation ai
▷ Combination of multiple imag
Typical Applications
▷ Inspection of dimensional accu
▷ Generation of overview images
▷ Robot vision
Provided Functiona
▷ Calibration of camera systems
▷ Transformation of image coorc
▷ Rectification of mages to com
▷ Determination of orientation ai
Typical Applications
▷ Inspection of d sional accu
▷ Generation of overview images
▷ Robot vision

a) b)

Figure 98: Cut-out of the reference and the checked test image with the differences marked in the test image: a) Reference image; b) checked test image.

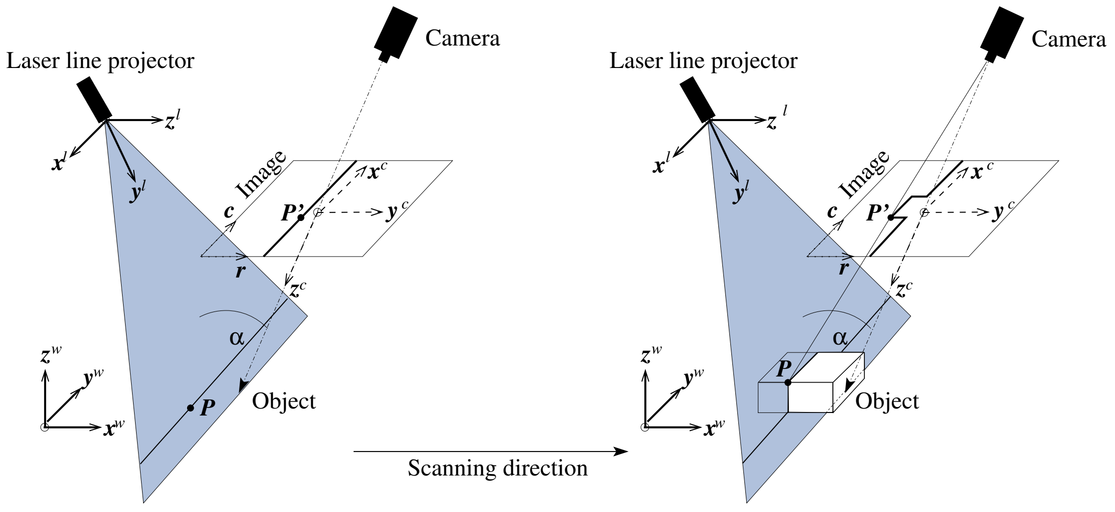
10 Laser Triangulation with Sheet of Light

Laser triangulation can be used to reconstruct the surface of a 3D object by approximating it via a set of height profiles. HALCON provides operators for a special type of laser triangulation that is called sheet-of-light technique.

10.1 The Principle of Sheet of Light

The basic idea of the sheet-of-light technique is to project a thin luminous straight line, e.g., generated by a laser line projector, onto the surface of the object that is to be reconstructed and then image the projected line with a camera. As shown in figure 99 the projection of the laser line builds a plane that is called light plane or sheet of light. The optical axis of the camera and the light plane form an angle α, which is called angle of triangulation. The points of intersection between the laser line and the camera view depend on the height of the object. Thus, if the object onto which the laser line is projected differs in height, the line is not imaged as a straight line but represents a profile of the object. Using this profile, we can obtain the height differences of the object. To reconstruct the whole surface of an object, i.e., to get many height profiles, the object must be moved relative to the measurement system, i.e., the unit built by the laser line projector and the camera.

The sheet-of-light technique can be applied either to a calibrated measurement setup or to the uncalibrated setup. If the setup is calibrated, the measurement returns the disparities and the x, y, and z coordinates of the points that build the profiles in the world coordinate system (WCS, see figure 99).



Light plane pose	x^l y^l z^l
Camera coordinate system	x^c y^c z^c
World coordinate system	x^w y^w z^w
Image coordinate system	r (row) c (column)
Angle of triangulation	α

Figure 99: Basic principle of sheet of light (light plane marked in gray).

The disparities are returned in form of a disparity image, i.e., the disparities of each measured profile are stored in one row of the disparity image (see figure 100 on page 151 and note that the camera must be oriented such that the profiles are roughly parallel to the rows of the image). The x, y, and z coordinates are also not explicitly returned as values but are expressed as values of pixels within images. That is, three images are returned, one for the x coordinates (X), one for the y coordinates (Y), and one for the z coordinates (Z). If the setup is uncalibrated, only the disparity image and a score that describes how reliable the measurement result is can be returned by the measurement.

Note that the disparity image for sheet of light has not exactly the same meaning as the disparity image described for stereo matching in section 7.1 on page 102 and section 7.4.2 on page 114. For stereo, the disparity describes the difference between the row coordinates of the left and right stereo images, whereas for sheet of light, the disparity is built by the subpixel row values at which the profile was detected.

10.2 The Measurement Setup

The hardware needed for a sheet-of-light measurement consists of a projector that is able to project a thin luminous line, a camera, a positioning system, and the object to measure. In the following, we assume the projector to be a laser line projector and the positioning system to be linear (e.g., a conveyor belt), as these are very common in laser triangulation applications.

The relation between the projector, the camera, and the linear positioning system must not be changed,

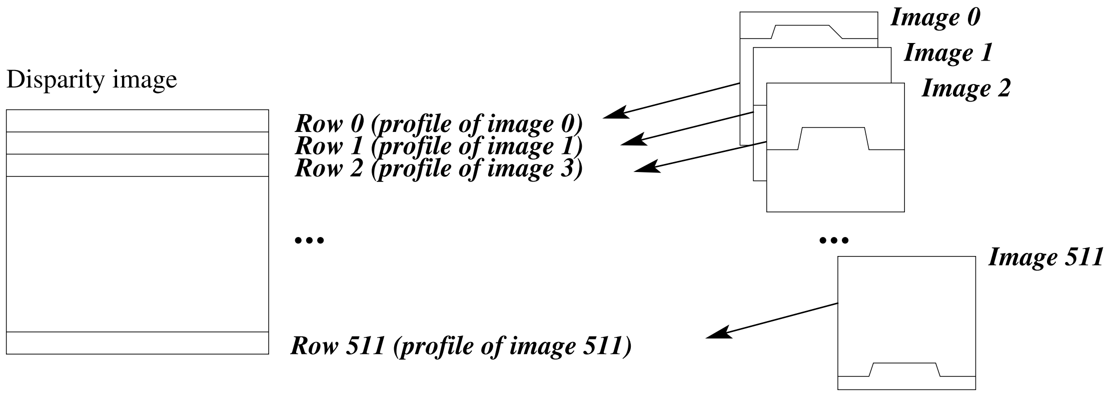

Disparity image
Image 0
Image 1
Image 2
Row 0 (profile of image 0)
Row 1 (profile of image 1)
Row 2 (profile of image 3)
...
...
Image 511
Row 511 (profile of image 511)

Figure 100: Disparity image: the disparity obtained from each profile (or image, respectively) is stored in a row of the disparity image.

whereas the position of the object that is transported by the positioning system changes in relation to the projector-camera unit. Since the profile images are processed column by column, the profiles must be oriented roughly horizontal, i.e., roughly parallel to the rows of the image.

The relation between the laser line projector, the camera, and the object to measure can be described by different measurement setups. Figure 101 shows the three apparent configurations for the three components. In the first case, the camera view is orthogonal to the object and the light plane is tilted. The second case shows a tilted camera view and an orthogonal light plane. For the third case, both the camera view and the laser line are tilted.

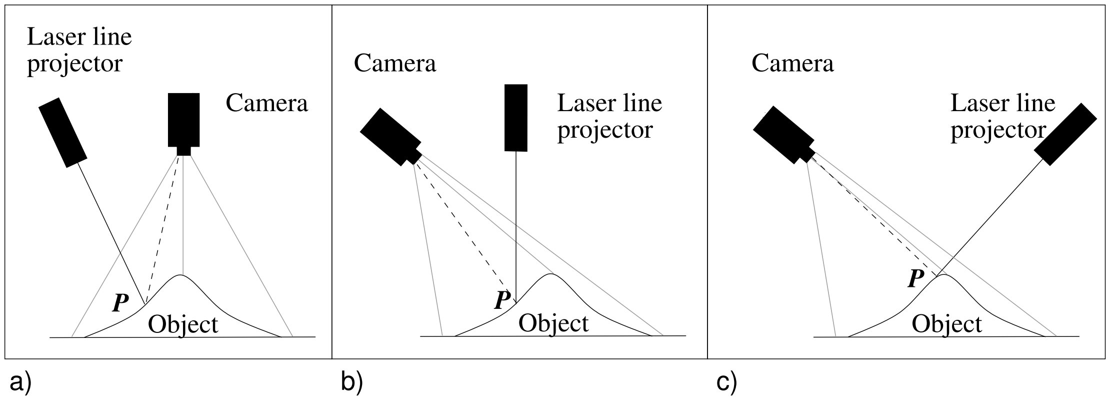


Figure 101: Basic configurations possible for a sheet-of-light measurement setup.

Figure 102 exemplarily shows a measurement setup as it is used for the examples that will be discussed in the following sections.

Which measurement configuration to use depends on the goal of the measurement and the geometry of the object. The configuration in figure 101 (a), e.g., is especially suitable if an orthogonal projection

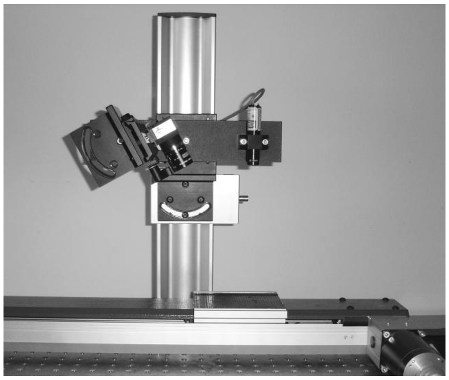


Figure 102: Examplary setup for a sheet-of-light measurement consisting of a camera, a laser line projector, and a positioning system.

of the object in the image is needed for any reason. Then, a cuboid is imaged as a rectangle. For all configurations in which the camera is not placed orthogonal, it would be imaged as a trapezoid because of the perspective deformations (see figure 103).

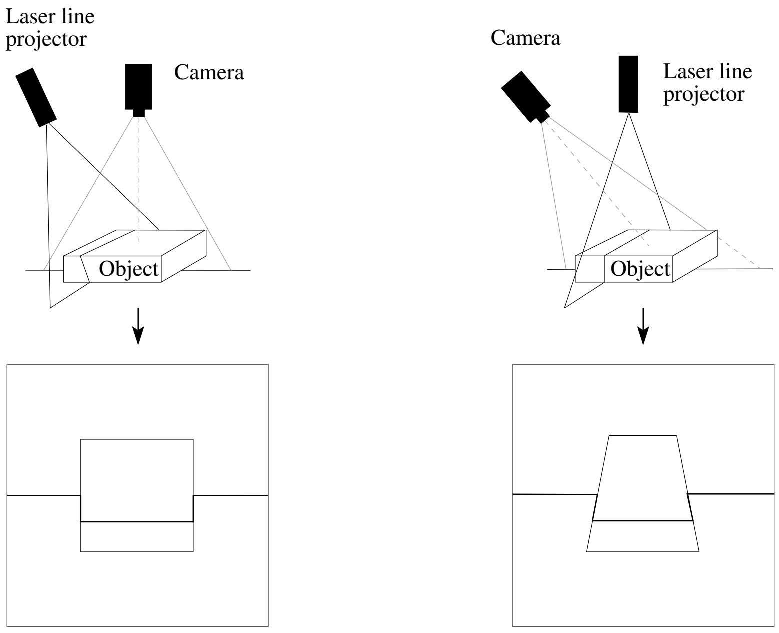


Figure 103: With the camera being orthogonal to the object, an orthogonal projection of the object is possible: (left) orthogonal camera view, (right) perspective view.

The most important criterion for the selection of the measurement setup is the geometry of the object. The setup should be selected such that the amount of shadowing effects and occlusions is minimized. Occlusions occur if an object point is illuminated by the laser line but is not visible in the image, because other parts of the object lie between the line of sight of the camera and the object point (see figure 104, top). Shadowing effects occur if an object point is visible in the image but is not illuminated by the laser line, because other parts of the object lie between the laser projection and the imaged object point (see figure 104, bottom).

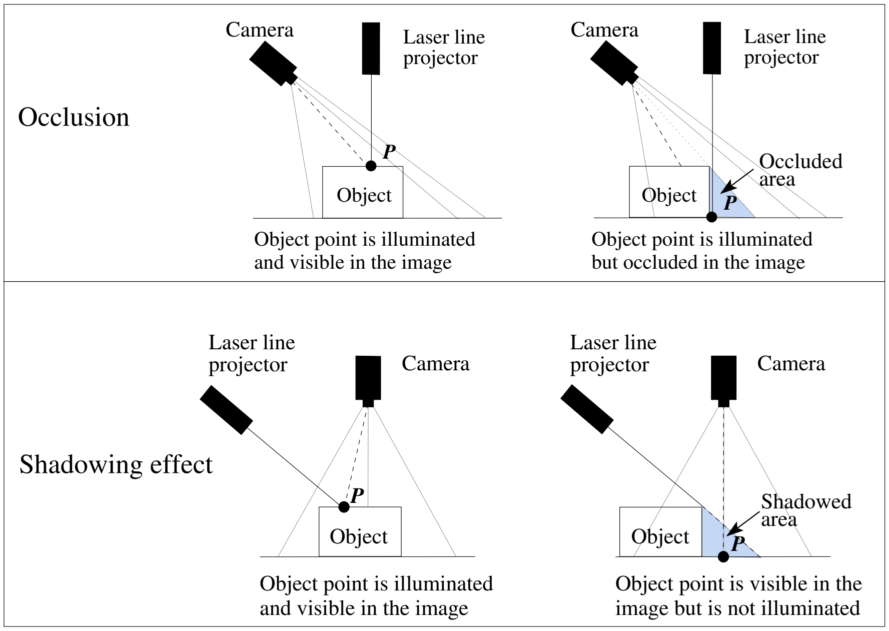

Occlusion
Camera
Laser line projector
P
Object
Object point is illuminated and visible in the image
Camera
Laser line projector
Occluded area
Object
P
Object point is illuminated but occluded in the image
Shadowing effect
Laser line projector
Camera
P
Object
Object point is illuminated and visible in the image
Laser line projector
Camera
Shadowed area
Object
P
Object point is visible in the image but is not illuminated

Figure 104: Problems that have to be considered before selecting the measurement setup: (top) occlusions and (bottom) shadowing effects.

For all three setup configurations, the angle of triangulation, i.e., the angle between the light plane and the optical axis of the camera, should be in a range of 30° to 60° to get a good measurement accuracy. If the angle is smaller, the accuracy decreases. If the angle is larger, the accuracy increases, but more problems because of occlusions and shadowing effects are to be expected. Thus, you have to find a trade-off between the accuracy and the completeness of the measurement.

10.3 Calibrating the Sheet-of-Light Setup

This section describes how to calibrate the sheet-of-light measurement setup. If the uncalibrated case is sufficient for your application, you can skip this section and proceed with section 10.4 on page 158.

The calibration of the sheet-of-light setup is applied to get the internal and external camera parameters, the orientation of the light plane in the WCS, and the relative movement of the object between two successive measurements. The calibration consists of the following steps:

1. Calibrate the camera.

2. Determine the orientation of the light plane with respect to the WCS.

3. Calibrate the movement of the object relative to the measurement setup.

The camera is calibrated by a standard camera calibration as described in section 3.1 on page 30. As result, the camera calibration returns the internal camera parameters and the pose of the WCS relative to the camera coordinate system (external camera parameters).

To determine the light plane and its pose, we need at least three corresponding points (see figure 105), in particular two points in the plane of the WCS with `'z=0'` (`P1`, `P2`) and one point that differs significantly in z direction (`P3`). Thus, you place the calibration object, e.g., the standard HALCON calibration plate, once or twice so that it lies in the plane of the WCS with `'z=0'`, and once so that a higher position can be viewed, i.e., the plate is either translated in z direction or it is placed in a tilted position. For each position of the calibration plate, you take two images, one showing the calibration plate and one showing the laser line. Note that you have to adapt the lighting in between to get one image with a clearly represented calibration plate and one image that shows a well-defined laser line. The translated or tilted position of the calibration plate should be selected so that the plane that is built by the points `P1`, `P2`, and `P3` becomes as large as possible. The height difference should be at least as big as the height differences expected for the objects to measure.

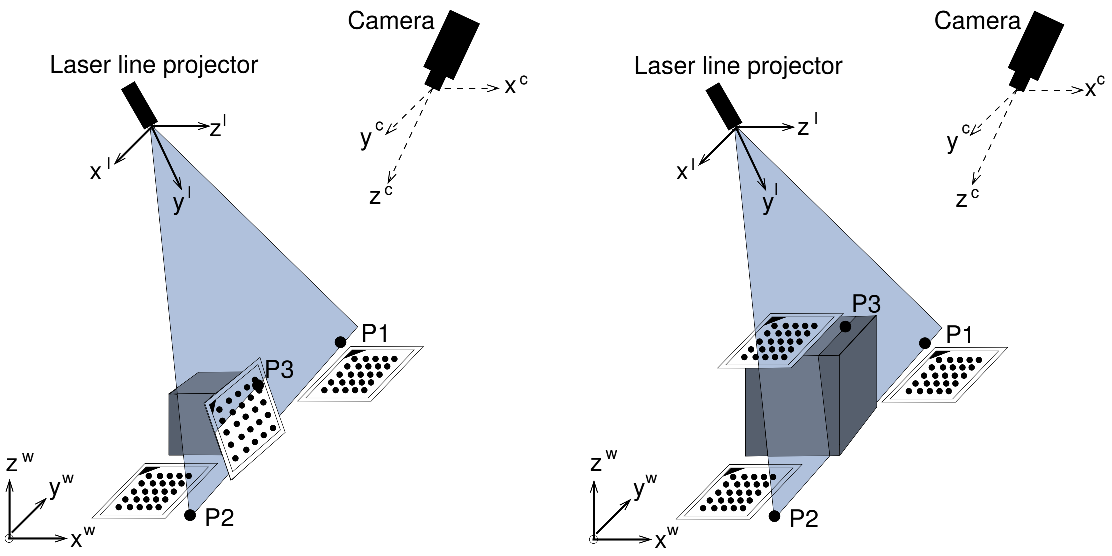


Figure 105: Position of the third point (P3) obtained by (left) tilted calibration plate or (right) translated calibration plate.

Note that the laser line has to be projected onto the same plane in which the calibration plate is placed. But if possible, it should not be directly projected onto the calibration plate but only near to it. This is because the standard HALCON calibration plate is made of a ceramic that shows a strong volume scattering. This leads to a broadened profile (see figure 106), which results in a poor accuracy. If you use a calibration object that consists of a material with different reflection properties, this might be no problem.

Note further that the three corresponding points described above represent the minimum number of points needed to obtain a plane. To enhance the precision of the calibration, redundancy is needed; thus, we

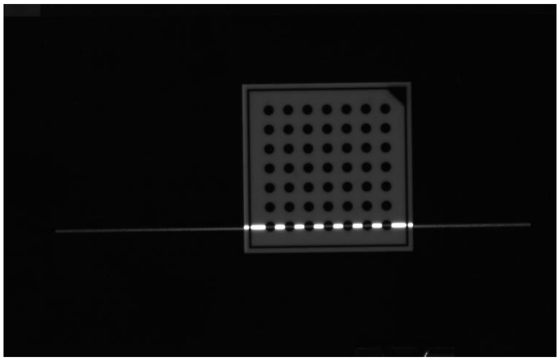


Figure 106: The white parts of a HALCON calibration plate show a very broad laser line because of volume scattering.

recommended to measure more than three corresponding points. Then, the light plane is approximated by fitting a plane into the obtained point cloud.

In the final step, the pose describing the movement of the object must be calibrated using two images containing a calibration plate that was moved by the positioning system by a known number of movement steps.

Summarized, we have to acquire:

- a set of images for the camera calibration,
- at least two images that clearly show the laser line in different planes and which correspond to images that were used for the calibration, and
- at least two images that show the calibration plate in the plane of the linear positioning system. Between the acquisition of the first and the second image, the calibration plate must be moved by the positioning system by a known number of movement steps.

The HDevelop example program `examples\hdevelop\Applications\3D-Vision\calibrate_sheet_of_light.dev` shows in detail how to calibrate a sheet-of-light measurement setup.

For the first step, i.e., the camera calibration, initial values for the internal camera parameters and for the thickness of the calibration plate are set.

```
StartParameters := [0.0125, 0.0, 0.0, 0.0, 0.0, 0.0, 0.000006, 0.000006, \
                    376.0, 120.0, 752, 240]
CalTabDescription := 'caltab_30mm.descr'
CalTabThickness := .00063
```

Then, the calibration images are read. These should fulfill the requirements that are described for a camera calibration in section 3.1.2 on page 33. Now, for each image, the calibration plates are searched and their marks and poses are derived.

```
NumCalibImages := 20
for Index := 1 to NumCalibImages by 1
  read_image (Image, 'sheet_of_light/connection_rod_calib_'+Index$'.2')
  find_caltab (Image, TmpObj_PlateRegion, CalTabDescription, 3, 105, 5)
  find_marks_and_pose (Image, TmpObj_PlateRegion, CalTabDescription, \
                       StartParameters, 128, 6, 10, 0.5, 15, 100, \
                       TmpCtrl_MarkRows, TmpCtrl_MarkColumns, \
                       TmpCtrl_EstimatedPose)
  TmpCtrl_AllMarkRows := [TmpCtrl_AllMarkRows, TmpCtrl_MarkRows]
  TmpCtrl_AllMarkColumns := [TmpCtrl_AllMarkColumns, TmpCtrl_MarkColumns]
  TmpCtrl_StartPoses := [TmpCtrl_StartPoses, TmpCtrl_EstimatedPose]
endfor
```

With the obtained data, the actual camera calibration is performed and the internal camera parameters (`CameraParameters`) as well as a tuple containing the external camera parameters (camera poses) for all calibration images (`TmpCtrl_FinalPoses`) are returned. The internal camera parameters and the camera pose for one of the calibration images are the first two variables that we need for the sheet-of-light measurement that is described in the next section. Note that by selecting the camera pose of one of the calibration images you define the origin of the WCS used for the measurement.

```
caltab_points (CalTabDescription, TmpCtrl_X, TmpCtrl_Y, TmpCtrl_Z)
camera_calibration (TmpCtrl_X, TmpCtrl_Y, TmpCtrl_Z, TmpCtrl_AllMarkRows, \
                    TmpCtrl_AllMarkColumns, StartParameters, \
                    TmpCtrl_StartPoses, 'all', CameraParameters, \
                    TmpCtrl_FinalPoses, TmpCtrl_Errors)
```

For the second step, i.e., the orientation of the light plane in relation to the WCS, the poses of two of the calibration images are needed. The images show the calibration plates in different heights. The pose of one image is used to define the WCS and the pose of the other image is used to define a temporary coordinate system (TCS). For both images, the origins of the poses are shifted with `set_origin_pose` to consider the thickness of the calibration plate.

```
Index := 19
set_origin_pose (TmpCtrl_FinalPoses[(Index-1)*7:(Index-1)*7+6], 0.0, 0.0, \
                 CalTabThickness, CameraPose)
Index := 20
set_origin_pose (TmpCtrl_FinalPoses[(Index-1)*7:(Index-1)*7+6], 0.0, 0.0, \
                 CalTabThickness, TmpCameraPose)
```

For each of the two calibration images a corresponding laser line image was acquired. There, the laser line is clearly projected onto the same plane that contained the calibration plate in the calibration image. With the two laser line images and the poses obtained from the two corresponding calibration images the procedure `compute_3d_coordinates_of_light_line` calculates the 3D coordinates of the points that build the laser lines. The obtained point cloud consists of the points of the light plane in the plane of the WCS with `'z=0'` (see `P1` and `P2` in figure 105 on page 154) and the points of the light plane in the plane of the TCS with `'z=0'` (see `P3` in figure 105 on page 154).

```
read_image (ProfileImage1, \
            'sheet_of_light/connection_rod_lightline_019.png')
compute_3d_coordinates_of_light_line (ProfileImage1, MinThreshold, \
                                      CameraParameters, [], CameraPose, \
                                      X19, Y19, Z19)
read_image (ProfileImage2, \
            'sheet_of_light/connection_rod_lightline_020.png')
compute_3d_coordinates_of_light_line (ProfileImage2, MinThreshold, \
                                      CameraParameters, TmpCameraPose, \
                                      CameraPose, X20, Y20, Z20)
```

Now, the procedure `fit_3d_plane_xyz` fits a plane into the point cloud. This plane is the light plane, for which the pose is needed as the third variable for the calibrated sheet-of-light measurement. This pose (`LightPlanePose`) is calculated from the plane using the procedure `get_light_plane_pose`.

```
procedure fit_3d_plane_xyz (: : X, Y, Z: Ox, Oy, Oz, Nx, Ny, Nz, MeanResidual)
get_light_plane_pose (Ox, Oy, Oz, Nx, Ny, Nz, LightPlanePose)
```

In the third step, i.e., the calibration of the movement of the object in relation to the measurement setup, the calibration plate is moved in discrete steps by the linear positioning system that will be used also for the following measurement. To calibrate the movement of the linear positioning system, two images with different movement states are needed. To enhance the accuracy, we do not use images of two succeeding movement steps but use images with a known number of movement steps between them. Here, the number of movement steps between both images is `19`.

```
read_image (CaltabImagePos1, 'sheet_of_light/caltab_at_position_1.png')
read_image (CaltabImagePos20, 'sheet_of_light/caltab_at_position_2.png')
StepNumber := 19
```

Now, for both images the poses of the calibration plates are derived.

```
find_caltab (CaltabImagePos1, CaltabPos1, CalTabDescription, 3, 105, 15)
find_marks_and_pose (CaltabImagePos1, CaltabPos1, CalTabDescription, \
                     CameraParameters, 128, 10, 20, 0.8, 15, 100, \
                     RCoordPos1, CCoordPos1, StartCameraPosePos1)
camera_calibration (TmpCtrl_X, TmpCtrl_Y, TmpCtrl_Z, RCoordPos1, CCoordPos1, \
                    CameraParameters, StartCameraPosePos1, 'pose', \
                    CamParam, CameraPosePos1, ErrorsPos1)
find_caltab (CaltabImagePos20, CaltabPos20, CalTabDescription, 3, 105, 15)
find_marks_and_pose (CaltabImagePos20, CaltabPos20, CalTabDescription, \
                     CameraParameters, 128, 10, 20, 0.8, 15, 100, \
                     RCoordPos20, CCoordPos20, StartCameraPosePos20)
camera_calibration (TmpCtrl_X, TmpCtrl_Y, TmpCtrl_Z, RCoordPos20, \
                    CCoordPos20, CameraParameters, StartCameraPosePos20, \
                    'pose', CamParam, CameraPosePos20, ErrorsPos20)
```

Then, the pose that describes the transformation between these two poses, i.e., the transformation needed for `19` movement steps, is calculated (`MovementPoseNSteps`).

```
pose_to_hom_mat3d (CameraPosePos1, HomMat3DPos1ToCamera)
pose_to_hom_mat3d (CameraPosePos20, HomMat3DPos20ToCamera)
pose_to_hom_mat3d (CameraPose, HomMat3DWorldToCamera)
hom_mat3d_invert (HomMat3DWorldToCamera, HomMat3DCameraToWorld)
hom_mat3d_compose (HomMat3DCameraToWorld, HomMat3DPos1ToCamera, \
                   HomMat3DPos1ToWorld)
hom_mat3d_compose (HomMat3DCameraToWorld, HomMat3DPos20ToCamera, \
                   HomMat3DPos20ToWorld)
hom_mat3d_invert (HomMat3DPos20ToWorld, HomMat3DWorldToPos20)
hom_mat3d_compose (HomMat3DWorldToPos20, HomMat3DPos1ToWorld, \
                   HomMat3DPos1ToPos20)
hom_mat3d_invert (HomMat3DPos1ToPos20, HomMat3DMovementNSteps)
hom_mat3d_to_pose (HomMat3DMovementNSteps, MovementPoseNSteps)
```

To get the pose for a single movement step (`MovementPose`), the elements of `MovementPoseNSteps` that describe a translation are divided by the number of steps. A rotation is not assumed and therefore all rotational elements are set to `0`. `MovementPose`, together with the internal and external camera parameters and the pose of the light plane can now be used to apply a calibrated sheet-of-light measurement.

```
MovementPose := [MovementPoseNSteps[0]/StepNumber, \
                 MovementPoseNSteps[1]/StepNumber, \
                 MovementPoseNSteps[2]/StepNumber,0,0,0,0]
```

For details about the proceedings inside the stated procedures, we refer to the example.

10.4 Performing the Measurement

A sheet-of-light measurement is applied to get height information for the object to measure. This height information is presented either by a disparity image in which each row contains the disparities of one measured profile of the object (see figure 100 on page 151) or by the images `X`, `Y`, and `Z`, which express the x, y, and z coordinates of the measured profiles as values of pixels within images. The images `X`, `Y`, and `Z` can be obtained only for a calibrated measurement setup, whereas the disparity image can be obtained also for the uncalibrated case. A sheet-of-light measurement consists of the following basic steps:

1. Calibrate the measurement setup (if a calibrated measurement is needed) as described in the previous section.
2. Create a sheet-of-light model with `create_sheet_of_light_model` and set additional parameters with successive calls to `set_sheet_of_light_param`.
3. Acquire images for each profile to measure, e.g., using `grab_image_async`.
4. Measure the profile of each image with `measure_profile_sheet_of_light`.
5. Get the results of the measurement with successive calls to `get_sheet_of_light_result`.

6. If only the uncalibrated case was applied and a disparity image was obtained, but the x, y, and z coordinates are still needed, you can subsequently apply a calibration. Then, you have to calibrate the measurement setup like described in the previous section and add the obtained camera parameters to the model with set_sheet_of_light_param. Afterwards you call the operator apply_sheet_of_light_calibration with the disparity image and the adapted sheet-of-light model. The resulting images that contain the coordinates X, Y, and Z are queried from the model with get_sheet_of_light_result. How to subsequently apply a sheet of light calibration to a disparity image is shown in the HDevelop example program examples\hdevelop\Applications\3D-Vision\calibrate_sheet_of_light.dev.

7. Clear the model from memory with clear_sheet_of_light_model.

Optionally, you can query all parameters that you have already set for a specific model or that were set by default using get_sheet_of_light_param. To query all parameters that can be set for a sheet-of-light model you call query_sheet_of_light_params.

10.4.1 Calibrated Sheet-of-Light Measurement

How to apply a calibrated sheet-of-light measurement is shown in the HDevelop example program examples\hdevelop\Applications\3D-Vision\reconstruct_connection_rod_calib.dev, which measures the object shown in figure 107.

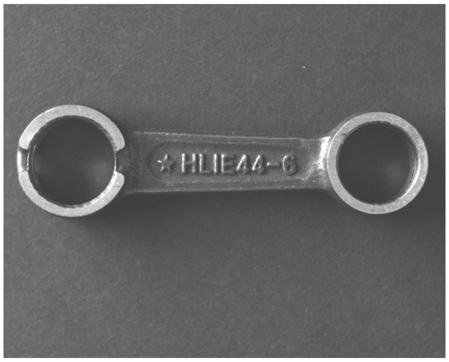


Figure 107: Object to measure.

The first step is to assign the information obtained by the calibration of the sheet-of-light measurement setup (see previous section) to a set of variables.

```
CamParam := [0.0126389, 658.417, -2.43408e+07, 1.09591e+12, -0.0995053, \
              0.0286728, 6.00046e-06, 6e-06, 388.079, 120.651, 752, 240]
CamPose := [-0.00180691,-6.9929e-005,0.299885,0.590058,0.572089,180.027,0]
LightplanePose := [0.00271412,-0.0055037,0.00844186,67.0192,359.723, \
                   0.65525,0]
MovementPose := [-5.84916e-007,0.000120099,2.50926e-6,0,0,0,0]
```

Then, a sheet-of-light model is created for a rectangular region of interest using `create_sheet_of_light_model`. The ROI should be selected as large as necessary but as small as possible. That is, it should approximately be some pixels larger than the width of the object in width and the maximal expected displacement of the laser line caused by the height of the object, i.e., the largest expected disparity, in height.

Now, some parameters are set with `set_sheet_of_light_param`. As a calibrated measurement is applied, the parameter `'calibration'` is set to `'xyz'`. For an uncalibrated measurement, it would be `'none'`, which is the default. Further, the variables with the calibration information are passed as values to the corresponding parameters for the internal camera parameters (`'camera_parameter'`), the external camera parameters (`'camera_pose'`), the pose of the light plane (`'lightplane_pose'`), and the movement of the object relative to the measurement setup (`'movement_pose'`).

```
gen_rectangle1 (ProfileRegion, 120, 75, 195, 710)
create_sheet_of_light_model (ProfileRegion, ['min_gray','num_profiles', \
                             'ambiguity_solving'], [70,290,'first'], \
                              SheetOfLightModelID)
set_sheet_of_light_param (SheetOfLightModelID, 'calibration', 'xyz')
set_sheet_of_light_param (SheetOfLightModelID, 'scale', 'mm')
set_sheet_of_light_param (SheetOfLightModelID, 'camera_parameter', CamParam)
set_sheet_of_light_param (SheetOfLightModelID, 'camera_pose', CamPose)
set_sheet_of_light_param (SheetOfLightModelID, 'lightplane_pose', \
                          LightplanePose)
set_sheet_of_light_param (SheetOfLightModelID, 'movement_pose', \
                          MovementPose)
```

Then, for each profile to measure an image is acquired (see, e.g., figure 108) to apply the actual measurement. Here, the images for each movement step are read from file with `read_image`. In practice, you will most probably grab the images directly from your image acquisition device using `grab_image_async` (see Solution Guide II-A for details about image acquisition). For each image, the profile within the rectangular region of interest is measured with `measure_profile_sheet_of_light`, i.e., the disparities for the profile are determined and stored in the sheet-of-light model.

```
for Index := 1 to 290 by 1
  read_image (ProfileImage, 'sheet_of_light/connection_rod_'+Index$'.3')
  measure_profile_sheet_of_light (ProfileImage, SheetOfLightModelID, [])
endfor
```

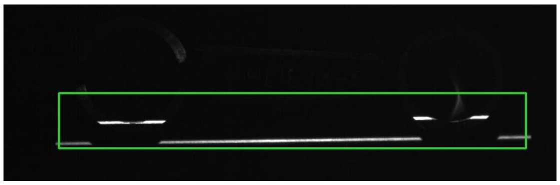


Figure 108: Measure profile inside a rectangular ROI.

The default for the number of profiles to measure is `512`. You can change it with the parameter `'num_profiles'` within `create_sheet_of_light_model` or `set_sheet_of_light_param`. If you measure more than the specified number of profiles, the value of 'num_profiles' is automatically adapted in the model. Nevertheless, this adaptation requires additional runtime. Thus, we recommend to set 'num_profiles' to a suitable value before starting the measurement. Note that the number of measured profiles defines the number of rows and the width of the ROI used for the measurement defines the number of columns for the result images (i.e., the disparity image, the X, Y, and Z images, and the score image).

After all measurements were performed, the results of the sheet-of-light measurement are queried with calls to get_sheet_of_light_result. Here, we query the disparity image (ResultName set to 'disparity') as well as the images X, Y, and Z (ResultName set to 'x', 'y', and 'z', respectively). The images X, Y, and Z are shown in figure 109.

The interpretation of the gray values of the disparity image and the images X, Y, and Z is as follows: black parts are outside of the domain of the resulting image, i.e., they indicate parts for which no 3D information could be reconstructed. For the pixels inside the domain of the image bright parts show low object parts and dark parts show higher object parts. Note that in this example the images are not visualized by their default gray values but are converted additionally by a look-up table so that the images are colored. This is done because the human eye can separate much more colors than gray values. Thus, details can be better distinguished during a visual inspection.

```
get_sheet_of_light_result (Disparity, SheetOfLightModelID, 'disparity')
get_sheet_of_light_result (X, SheetOfLightModelID, 'x')
get_sheet_of_light_result (Y, SheetOfLightModelID, 'y')
get_sheet_of_light_result (Z, SheetOfLightModelID, 'z')
```

At the end of the measurement, the model is cleared from memory.

```
clear_sheet_of_light_model (SheetOfLightModelID)
```

10.4.2 Uncalibrated Sheet-of-Light Measurement

The uncalibrated sheet-of-light measurement is shown in the HDevelop example program examples\hdevelop\Applications\3D-Vision\reconstruct_connection_rod_uncalib.dev. Here, no calibration results are needed, so we simply create the model for the specified region of interest and set the few needed parameters directly within create_sheet_of_light_model.

```
gen_rectangle1 (ProfileRegion, 120, 75, 195, 710)
create_sheet_of_light_model (ProfileRegion, ['min_gray','num_profiles', \
                             'ambiguity_solving','score_type'], [70,290, \
                             'first','width'], SheetOfLightModelID)
```

The actual measurement is applied by the same process used for the calibrated measurement.

```
for Index := 1 to 290 by 1
  read_image (ProfileImage, 'sheet_of_light/connection_rod_'+Index$'.3')
  measure_profile_sheet_of_light (ProfileImage, SheetOfLightModelID, [])
endfor
```

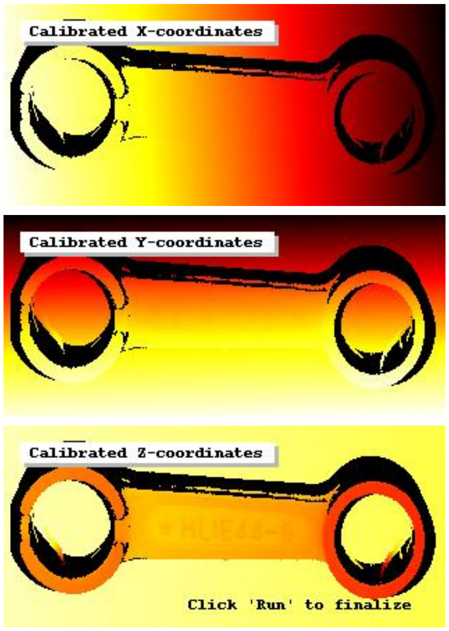


Figure 109: Result of calibrated sheet-of-light measurement: images representing the (from top to bottom) x, y, and z coordinates of the object.

As result, we can only query the disparity image (see figure 110) and the score (see section 10.5 on page 163) of the measurement (`ResultName` set to `'score'`).

```
get_sheet_of_light_result (Disparity, SheetOfLightModelID, 'disparity')
get_sheet_of_light_result (Score, SheetOfLightModelID, 'score')
```

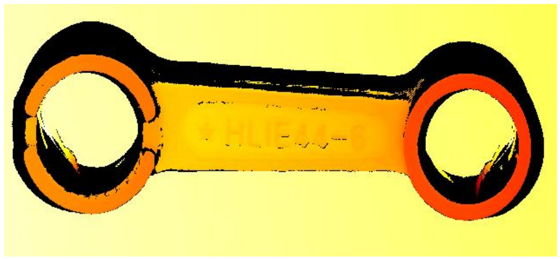


Figure 110: Result of uncalibrated sheet-of-light measurement: disparity image.

At the end of the program, the model is cleared from memory.

```
clear_sheet_of_light_model (SheetOfLightModelID)
```

10.5 Using the Score Image

Caused by the specific characteristics of a laser line projector and the general principle of triangulation the results of a sheet-of-light measurement, i.e., the disparities or the calibrated coordinates, sometimes show disturbing artifacts. The score image can be used to detect and partially remove artifacts.

There are two types of artifacts. The first type is caused by the geometry of the surface that is to be reconstructed. As illustrated in figure 111, compared to flat and smooth surfaces (e.g., the object in figure 110 on page 162), curved surfaces with a small radius of curvature and surfaces with a significant slope lead to a broadening of the light line. Furthermore, the light distribution within the profile might be no longer symmetric, which leads to a reduced measurement accuracy.

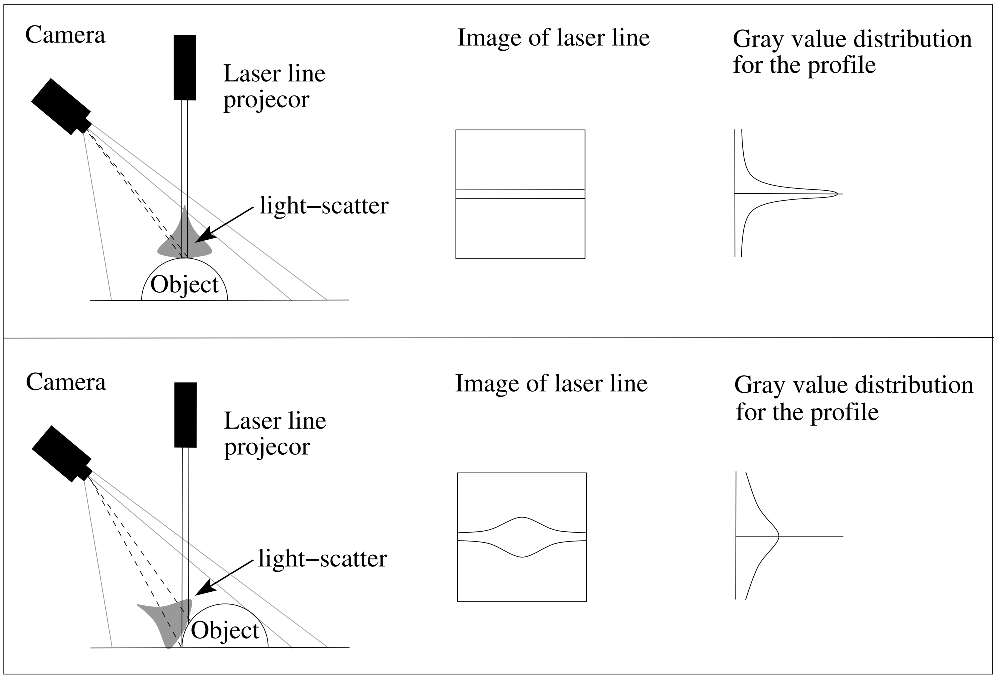


Figure 111: Curved surfaces with a small radius of curvature and surfaces with a significant slope lead to a broadening of the light line and thus to a low score: (top) small influence of curvature, (bottom) large influence of curvature.

By using the width of the profile stored in the score image (for each pixel of the disparity, the score value is set to the number of pixels used to determine the disparity value) it is possible to detect artifacts and to reject the corresponding disparities or the corresponding calibrated coordinates. Figure 112 shows the score image obtained by the uncalibrated sheet-of-light measurement performed in the HDevelop example program `examples\hdevelop\Applications\3D-Vision\reconstruct_connection_rod_uncalib.dev`. The gray values inside the score image indicate the widths of the laser line in each pixel. Thus, artifacts, in this case parts with a significantly broadened laser line, can be recognised easily by their brightness.

In the example, the artifacts are rejected from the disparity image by applying a threshold to the score

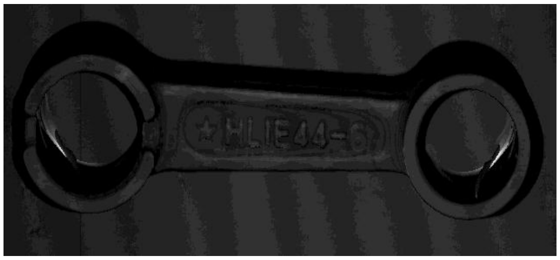


Figure 112: Result of uncalibrated sheet-of-light measurement: score image (`score_type` set to `'width'`, i.e., bright parts indicate artifacts).

image (pixels with a value larger than `7.5` are rejected) and reducing the disparity image to the obtained region.

```
threshold (Score, ScoreRegion, 1.5, 7.5)
reduce_domain (Disparity, ScoreRegion, DisparityReduced)
```

The second type of artifacts is caused by the interaction of the coherent laser light with the surface of the object. Laser light produces disturbing interference patterns when it is projected on a rough textured surface. Those interference patterns are called speckle and can be considered as a non-additive noise, which means that this noise can not be reduced by averaging during the image acquisition. In this case, the only way to increase the measurement accuracy is to use a higher aperture for the image acquisition or a low-speckle line projector. Note that enlarging the aperture for the image acquisition device will also reduce the depth of field which might be an undesired side effect. If your application requires high accuracy, we strongly recommend to use low-speckle projection devices. Note that speckle in most cases is the limiting factor for the accuracy of laser triangulation systems.

10.6 3D Cameras for Sheet of Light

The proceeding described in the previous sections works for any standard 2D camera that is suitable for machine vision. An alternative is to use specific 3D cameras for which the sheet-of-light measurement is applied inside the camera. These cameras are more expensive than standard 2D cameras, but the sheet-of-light measurement becomes significantly faster because of the reduced CPU load. Using one of these cameras, you simply access the camera with HALCON and basically leave the measurement to the camera.

Generally, we distinguish between cameras with an inbuilt laser, i.e., the camera and the laser are integrated in a single unit, and cameras for which the laser is mounted separately.

If the camera and the laser are integrated in a single unit, the measurement setup is restricted to a fixed angle of triangulation and should be oriented in a defined way. For example, the SICK Ruler camera should be oriented so that the laser is perpendicular to the linear positioning system. Because of the preset measurement setup, the camera and the orientation of the light plane with respect to the world coordinate system are already calibrated. Thus, no further processing with HALCON is needed to obtain calibrated height profiles.

If the camera and the laser are mounted separately, any configuration of the measurement setup is possible (see section 10.2 on page 150), but by default the result of the measurement is uncalibrated. If the result of the measurement is needed in world coordinates, you can either query the uncalibrated data from the camera and subsequently apply a calibration with HALCON as described in section 10.4 on page 158, or, before performing the actual measurement, you apply a calibration that is provided specifically for the selected camera. For the SICK Ranger cameras, e.g., the camera-specific calibration needs the software provided with the camera (the SICK Coordinator tool) and a specific calibration object that has to be purchased separately.

Note that in contrast to the proceeding described in the previous sections, the movement of the linear positioning system is mostly assumed to be known, because the measurement of each profile is triggered by a signal coming from an encoder on the linear positioning system. That is, when working with an encoder and if the thus obtained accuracy is sufficient, it is not necessary to calibrate the distance between two profiles.

Appendix

A The HALCON Calibration Plate

Figure 113a shows a HALCON calibration plate. Note that it has an asymmetric pattern in the upper left corner. This pattern ensures that the pose of the calibration plate can be determined uniquely.

Old calibration plates do not have this pattern (see figure 113b). This may lead to problems if, e.g., a stereo camera or hand-eye system must be calibrated because the poses must be determined uniquely for this. To overcome this problem, you can make an asymmetric calibration plate out of your old calibration plate by marking one corner. Pay attention that the asymmetric pattern is not too close to the circular calibration marks because otherwise it could have an influence on the geometric accuracy of the calibration result.

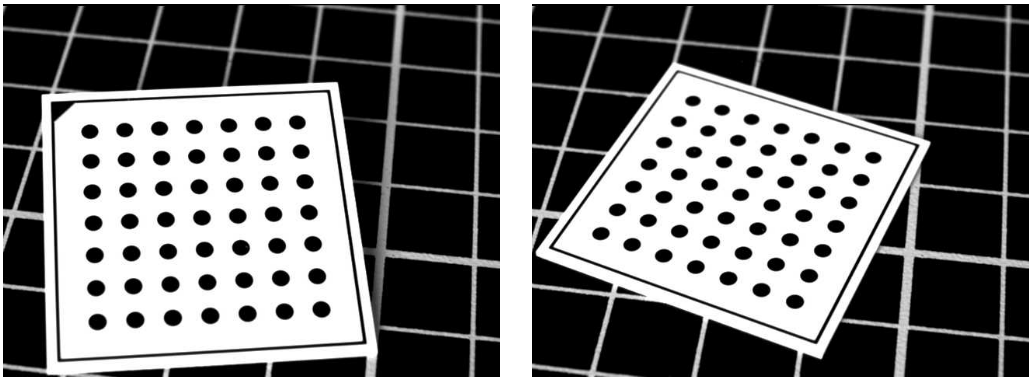

(a) (b)

Figure 113: (a) The HALCON calibration plate with the asymmetric pattern in the upper left corner; (b) an old calibration plate that does not have the asymmetric pattern.

There are two different types of calibration plate description files, which typically lie in the subdirectory `calib` of the folder where you installed HALCON: The standard description files for calibration plates that have the asymmetric pattern and the old description files for calibration plates without the asymmetric pattern. The old description files are indicated with the suffix *old* (**o**rientation-**l**ess **d**escription).

The behavior of the operator `find_marks_and_pose` depends on the combination of the used calibration plate and the specified calibration plate description file:

Calibration plate	Description file	Behavior of `find_marks_and_pose`
asymmetric	asymmetric	The pose will be determined uniquely.
asymmetric	old	The pose will be determined such that the x-axis points to the right and the y-axis points downwards.
old	asymmetric	The operator `find_marks_and_pose` returns an error because it cannot find the asymmetric pattern.
old	old	The pose will be determined such that the x-axis points to the right and the y-axis points downwards.

B HDevelop Procedures Used in this Solution Guide

B.1 gen_hom_mat3d_from_three_points

```
procedure gen_hom_mat3d_from_three_points (: : Origin, PointOnXAxis,
                                           PointInXYPlane: HomMat3d)
XAxis := [PointOnXAxis[0]-Origin[0],PointOnXAxis[1]-Origin[1], \
          PointOnXAxis[2]-Origin[2]]
XAxisNorm := XAxis/sqrt(sum(XAxis*XAxis))
VectorInXYPlane := [PointInXYPlane[0]-Origin[0], \
                    PointInXYPlane[1]-Origin[1], \
                    PointInXYPlane[2]-Origin[2]]
cross_product (XAxisNorm, VectorInXYPlane, ZAxis)
ZAxisNorm := ZAxis/sqrt(sum(ZAxis*ZAxis))
cross_product (ZAxisNorm, XAxisNorm, YAxisNorm)
HomMat3d_WCS_to_RectCCS := [XAxisNorm[0],YAxisNorm[0],ZAxisNorm[0], \
                            Origin[0],XAxisNorm[1],YAxisNorm[1], \
                            ZAxisNorm[1],Origin[1],XAxisNorm[2], \
                            YAxisNorm[2],ZAxisNorm[2],Origin[2]]
hom_mat3d_invert (HomMat3d_WCS_to_RectCCS, HomMat3d)
return ()
```

This procedure uses the procedure

```
procedure cross_product (: : V1, V2: CrossProduct)
CrossProduct := [V1[1]*V2[2]-V1[2]*V2[1],V1[2]*V2[0]-V1[0]*V2[2], \
                 V1[0]*V2[1]-V1[1]*V2[0]]
return ()
```

B.2 parameters_image_to_world_plane_centered

```
procedure parameters_image_to_world_plane_centered (: : CamParam, Pose,
                                                    CenterRow, CenterCol,
                                                    WidthMappedImage,
                                                    HeightMappedImage:
                                                    ScaleForCenteredImage,
                                                    PoseForCenteredImage)
* Determine the scale for the mapping
* (here, the scale is determined such that in the
*   surroundings of the given point  the image scale of the
*   mapped image is similar to the image scale of the original image)
Dist_ICS := 1
image_points_to_world_plane (CamParam, Pose, CenterRow, CenterCol, 1, \
                             CenterX, CenterY)
image_points_to_world_plane (CamParam, Pose, CenterRow+Dist_ICS, CenterCol, \
                             1, BelowCenterX, BelowCenterY)
image_points_to_world_plane (CamParam, Pose, CenterRow, CenterCol+Dist_ICS, \
                             1, RightOfCenterX, RightOfCenterY)
distance_pp (CenterY, CenterX, BelowCenterY, BelowCenterX, \
             Dist_WCS_Vertical)
distance_pp (CenterY, CenterX, RightOfCenterY, RightOfCenterX, \
             Dist_WCS_Horizontal)
ScaleVertical := Dist_WCS_Vertical/Dist_ICS
ScaleHorizontal := Dist_WCS_Horizontal/Dist_ICS
ScaleForCenteredImage := (ScaleVertical+ScaleHorizontal)/2.0
* Determine the parameters for set_origin_pose such
* that the point given via get_mbutton will be in the center of the
* mapped image
DX := CenterX-ScaleForCenteredImage*WidthMappedImage/2.0
DY := CenterY-ScaleForCenteredImage*HeightMappedImage/2.0
DZ := 0
set_origin_pose (Pose, DX, DY, DZ, PoseForCenteredImage)
return ()
```

B.3 parameters_image_to_world_plane_entire

```
procedure parameters_image_to_world_plane_entire (Image: : CamParam, Pose,
                                                  WidthMappedImage,
                                                  HeightMappedImage:
                                                  ScaleForEntireImage,
                                                  PoseForEntireImage)
* Transform the image border into the WCS (scale = 1)
full_domain (Image, ImageFull)
get_domain (ImageFull, Domain)
gen_contour_region_xld (Domain, ImageBorder, 'border')
contour_to_world_plane_xld (ImageBorder, ImageBorderWCS, CamParam, Pose, 1)
smallest_rectangle1_xld (ImageBorderWCS, MinY, MinX, MaxY, MaxX)
* Determine the scale of the mapping
ExtentX := MaxX-MinX
ExtentY := MaxY-MinY
ScaleX := ExtentX/WidthMappedImage
ScaleY := ExtentY/HeightMappedImage
ScaleForEntireImage := max([ScaleX,ScaleY])
* Shift the pose by the minimum X and Y coordinates
set_origin_pose (Pose, MinX, MinY, 0, PoseForEntireImage)
return ()
```

B.4 tilt_correction

```
procedure tilt_correction (DistanceImage, RegionDefiningReferencePlane:
                           DistanceImageCorrected:
                           :
                           )
* Reduce the given region, which defines the reference plane
* to the domain of the distance image
get_domain (DistanceImage, Domain)
intersection (RegionDefiningReferencePlane, Domain, \
              RegionDefiningReferencePlane)
* Determine the parameters of the reference plane
moments_gray_plane (RegionDefiningReferencePlane, DistanceImage, MRow, MCol, \
                    Alpha, Beta, Mean)
* Generate a distance image of the reference plane
get_image_pointer1 (DistanceImage, _, Type, Width, Height)
area_center (RegionDefiningReferencePlane, _, Row, Column)
gen_image_surface_first_order (ReferencePlaneDistance, Type, Alpha, Beta, \
                               Mean, Row, Column, Width, Height)
* Subtract the distance image of the reference plane
* from the distance image of the object
sub_image (DistanceImage, ReferencePlaneDistance, DistanceImageWithoutTilt, \
           1, 0)
* Determine the scale factor for the reduction of the distance values
CosGamma := 1.0/sqrt(Alpha*Alpha+Beta*Beta+1)
* Reduce the distance values
scale_image (DistanceImageWithoutTilt, DistanceImageCorrected, CosGamma, 0)
return ()
```

B.5 visualize_results_of_find_marks_and_pose

```
procedure visualize_results_of_find_marks_and_pose (Image: : WindowHandle,
                                                    RCoord, CCoord,
                                                    Pose, CamPar,
                                                    CalTabFile: )
dev_set_window (WindowHandle)
dev_display (Image)
dev_set_color ('yellow')
gen_cross_contour_xld (Cross, RCoord, CCoord, 6, 0)
dev_display (Cross)
display_calplate_coordinate_system (CalTabFile, Pose, CamPar, WindowHandle)
return ()
```

B.6 display_calplate_coordinate_system

```
procedure display_calplate_coordinate_system (: : CalTabFile, Pose, CamPar,
                                              WindowHandle: )
caltab_points (CalTabFile, X, Y, Z)
* arrow should point to farthest marks
ArrowLength := abs(X[0])
display_3d_coordinate_system (Pose, CamPar, ArrowLength, WindowHandle, \
                              'blue')
return ()
```

B.7 display_3d_coordinate_system

```
procedure display_3d_coordinate_system (: : Pose, CamPar, ArrowLength,
                                        WindowHandle, Color: )
pose_to_hom_mat3d (Pose, HomMat3D)
* store coordinates of the arrows in tuples
* sequence: origin, x-axis, y-axis, z-axis
ArrowsXCoords := [0,ArrowLength,0,0]
ArrowsYCoords := [0,0,ArrowLength,0]
ArrowsZCoords := [0,0,0,ArrowLength]
* transform arrow points into camera coordinates
affine_trans_point_3d (HomMat3D, ArrowsXCoords, ArrowsYCoords, \
                       ArrowsZCoords, ArrowsXCoords_cam, ArrowsYCoords_cam, \
                       ArrowsZCoords_cam)
* get the image coordinates
project_3d_point (ArrowsXCoords_cam, ArrowsYCoords_cam, ArrowsZCoords_cam, \
                  CamPar, ArrowsRows, ArrowsCols)
* display the coordinate system
dev_set_color (Color)
gen_contour_polygon_xld (XAxis, [ArrowsRows[0], ArrowsRows[1]], \
                         [ArrowsCols[0], ArrowsCols[1]])
dev_display (XAxis)
set_tposition (WindowHandle, ArrowsRows[1], ArrowsCols[1])
write_string (WindowHandle, 'x')
gen_contour_polygon_xld (YAxis, [ArrowsRows[0], ArrowsRows[2]], \
                         [ArrowsCols[0], ArrowsCols[2]])
dev_display (YAxis)
set_tposition (WindowHandle, ArrowsRows[2], ArrowsCols[2])
write_string (WindowHandle, 'y')
gen_contour_polygon_xld (ZAxis, [ArrowsRows[0], ArrowsRows[3]], \
                         [ArrowsCols[0], ArrowsCols[3]])
dev_display (ZAxis)
set_tposition (WindowHandle, ArrowsRows[3], ArrowsCols[3])
write_string (WindowHandle, 'z')
return ()
```

B.8 calc_calplate_pose_movingcam

```
procedure calc_calplate_pose_movingcam (: : CalibrationPose, CameraPose,
                                        RobotPoseInverse: CalplatePose)
* CalplatePose = cam_H_calplate = cam_H_tool * tool_H_base *  \
* base_H_calplate
pose_to_hom_mat3d (CalibrationPose, base_H_calplate)
pose_to_hom_mat3d (CameraPose, cam_H_tool)
pose_to_hom_mat3d (RobotPoseInverse, tool_H_base)
hom_mat3d_compose (cam_H_tool, tool_H_base, cam_H_base)
hom_mat3d_compose (cam_H_base, base_H_calplate, cam_H_calplate)
hom_mat3d_to_pose (cam_H_calplate, CalplatePose)
return ()
```

B.9 calc_calplate_pose_stationarycam

```
procedure calc_calplate_pose_stationarycam (: : CalibrationPose, CameraPose,
                                            RobotPose: CalplatePose)
* CalplatePose = cam_H_calplate = cam_H_base * base_H_tool *  \
* tool_H_calplate
pose_to_hom_mat3d (CalibrationPose, tool_H_calplate)
pose_to_hom_mat3d (CameraPose, cam_H_base)
pose_to_hom_mat3d (RobotPose, base_H_tool)
hom_mat3d_compose (cam_H_base, base_H_tool, cam_H_tool)
hom_mat3d_compose (cam_H_tool, tool_H_calplate, cam_H_calplate)
hom_mat3d_to_pose (cam_H_calplate, CalplatePose)
return ()
```

B.10 define_reference_coord_system

```
procedure define_reference_coord_system (: : ImageName, CamParam,
                                         CalplateFile, WindowHandle:
                                         PoseCamRef)
read_image (RefImage, ImageName)
dev_display (RefImage)
caltab_points (CalplateFile, X, Y, Z)
* parameter settings for find_caltab and find_marks_and_pose
SizeGauss := 3
MarkThresh := 100
MinDiamMarks := 5
StartThresh := 128
DeltaThresh := 3
MinThresh := 18
Alpha := 0.5
MinContLength := 15
MaxDiamMarks := 100
find_caltab (RefImage, Caltab, CalplateFile, SizeGauss, MarkThresh, \
             MinDiamMarks)
find_marks_and_pose (RefImage, Caltab, CalplateFile, CamParam, StartThresh, \
                     DeltaThresh, MinThresh, Alpha, MinContLength, \
                     MaxDiamMarks, RCoord, CCoord, StartPose)
camera_calibration (X, Y, Z, RCoord, CCoord, CamParam, StartPose, 'pose', \
                    CamParam, PoseCamRef, Errors)
disp_3d_coord_system (WindowHandle, CamParam, PoseCamRef, 0.01)
return ()
```

Index