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
Overview

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

A short characterisation 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 7. Afterwards, the methods to perform 3D measurements are described in detail.

Unless specified otherwise, the HDevelop example programs can be found in the subdirectory 3d_machine_vision of the directory %HALCONROOT%\examples\application_guide.
## Contents


2. **Basics** 7
   2.1 3D Transformations and Poses .................................................. 7
   2.2 Camera Model and Parameters .................................................... 18

3. **3D Machine Vision in a Specified Plane With a Single Camera** 22
   3.1 Calibrating the Camera ................................................................. 24
   3.2 Transforming Image into World Coordinates and Vice Versa ............... 35
   3.3 Rectifying Images ........................................................................... 39
   3.4 Inspection of Non-Planar Objects .................................................... 47

4. **Calibrated Mosaicking** 50
   4.1 Setup ............................................................................................... 50
   4.2 Calibration ........................................................................................ 50
   4.3 Merging the Individual Images into One Larger Image ....................... 53

5. **Uncalibrated Mosaicking** 60
   5.1 Rules for Taking Images for a Mosaic Image .................................... 63
   5.2 Definition of Overlapping Image Pairs .............................................. 64
   5.3 Detection of Characteristic Points .................................................... 68
   5.4 Matching of Characteristic Points .................................................... 69
   5.5 Generation of the Mosaic Image ....................................................... 71

6. **Pose Estimation of Known 3D Objects With a Single Camera** 72

7. **3D Machine Vision With a Binocular Stereo System** 76
   7.1 The Principle of Stereo Vision .......................................................... 78
   7.2 The Setup of a Stereo Camera System ............................................. 80
   7.3 Calibrating the Stereo Camera System ............................................. 82
   7.4 Obtaining 3D Information from Images ............................................ 88

8. **Rectification of Arbitrary Distortions** 97
   8.1 Basic Principle ................................................................................... 99
   8.2 Rules for Taking Images of the Rectification Grid ............................... 100
8.3 Machine Vision on Ruled Surfaces .............................................. 102
8.4 Using Self-Defined Rectification Grids ............................................ 105

9 HDevelop Procedures Used in this Application Note .......................... 110
9.1 disp_coordinate_system_3d ............................................................ 110
9.2 gen_hom_mat3d_from_three_points .............................................. 111
9.3 interactive_pose_assessment ......................................................... 111
9.4 parameters_image_to_world_plane_centered .................................. 112
9.5 parameters_image_to_world_plane_entire ...................................... 113
9.6 tilt_correction ............................................................................. 114
9.7 visualize_results_of_find_marks_and_pose .................................... 115
**1 Can You Really Perform 3D Machine 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 application note will introduce you to the world of 3D machine 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. 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 22). 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 39). 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 50) or highly automated for arbitrary and even varying image configurations (section 5 on page 60).

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 72). This is, e.g., necessary for pick-and-place applications.

If you need to determine the 3D shape of arbitrary objects, you can use HALCON’s binocular stereo vision functionality (section 7 on page 76). 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 inspection becomes possible.

Thus, we can answer the question that was posed at the beginning: Yes, you can perform 3D machine 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.

Note that so far no calibration of line scan cameras is possible.

We propose to read section 2.2 on page 18 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. For details on 3D transformations and poses, please refer to section 2.1 on page 7.
Figure 1: How to find the appropriate section of this application note.
2 Basics

2.1 3D Transformations and Poses

Before we start explaining how to perform machine vision in world coordinates 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:

$$
p^c = \begin{pmatrix} x^c_p \\ y^c_p \\ z^c_p \end{pmatrix}, \quad p^w = \begin{pmatrix} x^w_p \\ y^w_p \\ z^w_p \end{pmatrix}
$$

Figure 2 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 2: Coordinates of a point in two different coordinate systems.](image)
2.1.2 Translation

Translation of Points

In figure 3, our example point has been translated along the x-axis of the camera coordinate system.

The coordinates of the resulting point $P_2$ can be calculated by adding two vectors, the coordinate vector $p_1$ of the point and the translation vector $t$:

$$p_2 = p_1 + t = \begin{pmatrix} x_{p_1} + x_t \\ y_{p_1} + y_t \\ z_{p_1} + z_t \end{pmatrix}$$  \hspace{1cm} (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 4, 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 $c_1$ ($o_{c_2}^{c_1}$), is identical to the translation vector:

$$t^{c_1} = o_{c_2}^{c_1}$$  \hspace{1cm} (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 4: The coordinate system $c_1$, together with the point $Q_1$, is translated.
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., \( q_{1}^{c_1} = q_{2}^{c_2} \).

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

\[
q_{2}^{c_1} = q_{2}^{c_2} + \mathbf{t}_{c_1} = q_{2}^{c_2} + \mathbf{t}^{c_1}
\]  

(3)

In fact, figure 4 visualizes this equation: \( q_{2}^{c_1} \), 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 \( c_2 \) (\( q_{2}^{c_2} \)).

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 \( c_1 \).

**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 5a, the point $p_1$ is rotated by $-90^\circ$ around the z-axis of the camera coordinate system.

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

$$ p_3 = R_z(\gamma) \cdot p_1 = \begin{bmatrix} \cos \gamma & -\sin \gamma & 0 \\ \sin \gamma & \cos \gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{p_1} \\ y_{p_1} \\ z_{p_1} \end{bmatrix} = \begin{bmatrix} \cos \gamma \cdot x_{p_1} - \sin \gamma \cdot y_{p_1} \\ \sin \gamma \cdot x_{p_1} + \cos \gamma \cdot y_{p_1} \\ z_{p_1} \end{bmatrix} $$(4)

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

$$ R_y(\beta) = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \quad R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha \\ 0 & \sin \alpha & \cos \alpha \end{bmatrix} $$ (5)

Chain of Rotations

In figure 5b, 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:

$$ p_4 = R_y(\beta) \cdot p_3 = R_y(\beta) \cdot R_z(\gamma) \cdot p_1 $$ (6)

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 coordinate systems. This orientation changes when the coordinate system is rotated. For example, in figure 6a 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.

\[
\begin{align*}
\text{Coordinate system 1} & \quad \mathbf{R}_y (90^\circ) & \quad \text{Coordinate system 3} & \quad \mathbf{R}_z (-90^\circ) & \quad \text{Coordinate system 4} \\
(x^{c1}, y^{c1}, z^{c1}) & & (x^{c3}, y^{c3}, z^{c3}) & & (x^{c4}, y^{c4}, z^{c4})
\end{align*}
\]

Figure 6: 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 8), 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:

\[
\mathbf{R} = \begin{bmatrix}
x^{c_1}_{c_3} & y^{c_1}_{c_3} & z^{c_1}_{c_3}
\end{bmatrix}
\]

For example, the axis vectors of the coordinate system $c_3$ in figure 6a can be determined from the corresponding rotation matrix $\mathbf{R}_y(90^\circ)$ as shown in the following equation; you can easily check the result in the figure.

\[
\mathbf{R}_y(90^\circ) = \begin{bmatrix}
\cos(90^\circ) & 0 & \sin(90^\circ) \\
0 & 1 & 0 \\
-\sin(90^\circ) & 0 & \cos(90^\circ)
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 1 \\
0 & 1 & 0 \\
-1 & 0 & 0
\end{bmatrix}
\]

\[
\Rightarrow x^{c_1}_{c_3} = \begin{pmatrix}
0 \\
0 \\
-1
\end{pmatrix} \quad y^{c_1}_{c_3} = \begin{pmatrix}
0 \\
1 \\
0
\end{pmatrix} \quad z^{c_1}_{c_3} = \begin{pmatrix}
1 \\
0 \\
0
\end{pmatrix}
\]
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 \( c_3 \):

\[
q^{c_1} = c_1 \cdot R_{c_3} \cdot q^{c_3}
\]  

(8)

This is depicted in figure 6 also for a chain of rotations, which corresponds to the following equation:

\[
q^{c_1} = c_1 \cdot R_{c_3} \cdot R_{c_4} \cdot q^{c_4} = R_y(\beta) \cdot R_z(\gamma) \cdot q^{c_1} = c_1 \cdot R_{c_4} \cdot q^{c_4}
\]  

(9)

In Which Sequence and Around Which Axes are Rotations Performed?

If you compare the chains of rotations in figure 5 and figure 6 and the corresponding equations 6 and 9, you will note that two different sequences of rotations are described by the same chain of rotation matrices: In figure 5, the point was rotated first around the \( z \)-axis and then around the \( y \)-axis, whereas in figure 6 the coordinate system is rotated first around the \( y \)-axis and then around the \( z \)-axis. Yet, both are described by the chain \( R_y(\beta) \cdot R_z(\gamma) \)!

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 5 it is read from the right to left, and in figure 6 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.

<table>
<thead>
<tr>
<th>Performing a chain of rotations: ( R_y(90^\circ) \ast R_z(-90^\circ) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) reading from left to right = rotating around “new” axes</td>
</tr>
<tr>
<td>( c^1 ) ( R_y(90^\circ) ) ( c^3 ) ( R_z(-90^\circ) )</td>
</tr>
<tr>
<td>b) reading from right to left = rotating around “old” axes</td>
</tr>
<tr>
<td>( c^1 ) ( R_z(-90^\circ) ) ( c^3 ) ( R_y(90^\circ) )</td>
</tr>
</tbody>
</table>

Figure 7: 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 6b. 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 5, 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 7 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 8, the translation and rotation of the point from figures 3 and 5 are combined. Such a transformation is described as follows:

$$p_5 = R \cdot p_1 + t$$  \hspace{1cm} (10)

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

$$p_6 = R_a \cdot (R_b \cdot p_1 + t_b) + t_a = R_a \cdot R_b \cdot p_1 + R_a \cdot t_b + t_a$$  \hspace{1cm} (11)
Figure 8: Combining the translation from figure 3 on page 8 and the rotation of figure 5 on page 10 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 $H$ contains both the rotation matrix and the translation vector. For example, the rigid transformation from equation 10 can be rewritten as follows:

$$
\begin{pmatrix}
  p_5 \\
  1
\end{pmatrix} =
\begin{bmatrix}
  R & t \\
  0 & 1
\end{bmatrix}
\begin{pmatrix}
  p_1 \\
  1
\end{pmatrix} = \begin{pmatrix} R \cdot p_1 + t \end{pmatrix} = H \cdot p_1
$$

(12)

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:

$$
H_1 \cdot H_2 =
\begin{bmatrix}
  R_a & t_a \\
  0 & 1
\end{bmatrix}
\begin{bmatrix}
  R_b & t_b \\
  0 & 1
\end{bmatrix} = \begin{bmatrix} R_a \cdot R_b & R_a \cdot t_b + t_a \end{bmatrix}
$$

(13)

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:

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

(14)

If the rotation is composed of multiple rotations around axes as in figure 8, the individual rotations can...
also be written as homogeneous transformation matrices:

\[
H = \begin{bmatrix}
R_y(\beta) \cdot R_z(\gamma) & t \\
0 & 0 & 0 & 1
\end{bmatrix} = 
\begin{bmatrix}
1 & 0 & 0 & t \\
0 & 1 & 1 & t \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \cdot 
\begin{bmatrix}
R_y(\beta) & 0 \\
0 & 0 & 0 & 1 \\
R_z(\gamma) & 0
\end{bmatrix}
\]

Reading this chain from right to left, you can follow the transformation of the point in figure 8: 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 \(c_1 H_{c_5}\) 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.

\[
c_1 H_{c_5} = 
\begin{bmatrix}
x_{c_5}^{c_1} & y_{c_5}^{c_1} & z_{c_5}^{c_1} & o_{c_5}^{c_1}
0 & 0 & 0 & 1
\end{bmatrix}
\tag{15}
\]

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:

\[
\begin{pmatrix}
P_5^{c_1} \\
1
\end{pmatrix} = c_1 H_{c_5} \cdot 
\begin{pmatrix}
P_5^{c_5} \\
1
\end{pmatrix}
\tag{16}
\]

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

\[
P_5^{c_1} = c_1 H_{c_5} \cdot P_5^{c_5}
\tag{17}
\]

### 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.

As we already anticipated at the beginning of section 2.1 on page 7, 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.

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: \((\text{TransX}, \text{TransY}, \text{TransZ}, \text{Rot1}, \text{Rot2}, \text{Rot3})\). 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.

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:

\[
R_{gba} = R_x(\text{Rot1}) \cdot R_y(\text{Rot2}) \cdot R_z(\text{Rot3})
\]  

(18)

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

\[
R_{abg} = R_z(\text{Rot3}) \cdot R_y(\text{Rot2}) \cdot R_x(\text{Rot1})
\]  

(19)

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

\[
H = \begin{bmatrix}
R_y(\beta) \cdot R_z(\gamma) & t \\
0 & 0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
\cos \beta \cdot \cos \gamma & -\cos \beta \cdot \sin \gamma & \sin \beta & x_t \\
\sin \gamma & \cos \gamma & 0 & y_t \\
-\sin \beta \cdot \cos \gamma & \sin \beta \cdot \sin \gamma & \cos \beta & z_t \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
The corresponding pose with the rotation order 'gba' is much easier to read:

\[
(\text{TransX} = x_t, \text{TransY} = y_t, \text{TransZ} = z_t, \text{Rot1} = 0, \text{Rot2} = 90^\circ, \text{Rot3} = 90^\circ)
\]

If you look closely at figure 6 on page 11, you can see that the rotation can also be described by the sequence \( R_z(90^\circ) \cdot R_x(90^\circ) \). Thus, the transformation can also be described by the following pose with the rotation order 'abg':

\[
(\text{TransX} = x_t, \text{TransY} = y_t, \text{TransZ} = z_t, \text{Rot1} = 90^\circ, \text{Rot2} = 0, \text{Rot3} = 90^\circ)
\]

**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 exterior camera parameters as described in section 3.1.3 on page 27.

Figure 9 shows how to proceed for a rather simple example. The task is to determine the pose of the world coordinate system from figure 2 on page 7 relative to the camera coordinate system.

In such a case, we recommend to build up the rigid transformation from individual translations and rotations from left to right. Thus, in figure 9 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\(^\circ\), 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.

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.

Figure 10 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.

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 11. 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.

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 \( ^cH_w \). Therefore, the camera coordinates \( p^c = (x^c, y^c, z^c)^T \) of point \( P \) can be calculated from its world coordinates \( p^w = (x^w, y^w, z^w)^T \) simply by

\[
p^c = ^cH_w \cdot p^w \quad (20)
\]

The six parameters of this transformation (the three translations \( t_x, t_y, \) and \( t_z \) and the three rotations \( \alpha, \beta, \) and \( \gamma \)) are called the exterior 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.

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

\[
\begin{pmatrix}
u \\
v \\
\end{pmatrix} = \frac{f}{z^c} \begin{pmatrix}
x^c \\
 y^c \\
\end{pmatrix} \quad (21)
\]
Figure 10: Perspective projection by a pinhole camera.
For the telecentric camera model, the projection is a parallel projection, which is given by

\[
\begin{pmatrix}
u \\
v
\end{pmatrix} = \begin{pmatrix}
x^c \\
y^c
\end{pmatrix}
\]  \hspace{1cm} (22)

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.

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 \( P \).
2.2 Camera Model and Parameters

through the optical center, indicated by the dotted line in figure 12. Lens distortions cause the point $P'$ to lie at a different position.

$$P'$$

CCD chip

Optical center

Figure 12: 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. For most lenses, the distortion can be approximated sufficiently well by a radial distortion, which is given by

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \frac{2}{1 + \sqrt{1 - 4\kappa (u^2 + v^2)}} \begin{pmatrix} u \\ v \end{pmatrix}$$

(23)

The parameter $\kappa$ models the magnitude of the radial distortions. If $\kappa$ is negative, the distortion is barrel-shaped, while for positive $\kappa$ it is pincushion-shaped (see figure 13). This model for the lens distortions has the great advantage that the distortion correction can be calculated analytically by

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{1 + \kappa (\tilde{u}^2 + \tilde{v}^2)} \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}$$

(24)

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):

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} \frac{\tilde{u}}{s_y} + c_y \\ \frac{\tilde{v}}{s_x} + c_x \end{pmatrix}$$

(25)

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 radial 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 \((f, \kappa, s_x, s_y, c_x, c_y)\) of the pinhole camera and the five parameters \((\kappa, s_x, s_y, c_x, c_y)\) of the telecentric camera are called the \textit{interior camera parameters} because they determine the projection from 3D to 2D performed by the camera.

In HALCON, 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 the focal length is set to zero, the telecentric camera model is used.

With this, we can see that camera calibration is the process of determining the interior camera parameters \((f, \kappa, s_x, s_y, c_x, c_y)\) and the exterior camera parameters \((t_x, t_y, t_z, \alpha, \beta, \gamma)\).

### 3D Machine 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 \textit{stereo} approach; see \textbf{section 7} on page 76.

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 \textit{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 14 for an example), where in one image the calibration plate has been placed directly on the measurement plane.

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 interior 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 39).
3.1 Calibrating the Camera

As you have seen in the above example, in HALCON the calibration is determined simply by using the operator \texttt{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 15). See section 3.1.6 on page 34 on how to obtain this calibration plate. You can place it almost anywhere in front of the camera to determine the camera parameters.

Figure 15: 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 interior 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 exterior camera parameters, which are the pose of the measurement plane.

Note that the determination of the interior and of the exterior camera parameters can be separated. For this, the operator `camera_calibration` must be called twice. First, for the determination of the interior camera parameters only. Then, for the determination of the exterior camera parameters with the interior 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.

```plaintext
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 \text{MarkThresh} and maximum gray value 255 is applied with the intention to find the calibration plate. Therefore, \text{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 \text{MinDiamMarks} are eliminated to reduce the impact of noise. The number of marks is read from the calibration plate description file \text{CalTabDescrFile}.

\text{find_marks_and_pose} extracts the calibration marks and precisely determines their image coordinates. Therefore, in the input image \text{Image} an edge detector is applied to the region \text{CalTabRegion}, which can be found by the operator \text{find_caltab}. The edge detector can be controlled via the parameter \text{Alpha}. Larger values for \text{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 \text{StartThresh}. If the search for the closed contours or the successive pose estimate (see section 3.1.3) fails, this threshold value is successively decreased by \text{DeltaThresh} down to a minimum value of \text{MinThresh}.

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

The image coordinates of the calibration marks are determined by applying \text{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 \text{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 at least in two images.

• Use at least 10 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 over exposed.

• The diameter of a circle should be at least 10 pixels.

• The calibration plate should be completely inside the image.

• The images should contain as little noise as possible.

If you take into account these few rules for the acquisition of the calibration images, you can expect all HALCON operators used for the calibration process to work properly.

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 are used, the more accurate results will be achieved.

### 3.1.3 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.

The initial values for the interior 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 \([f, \kappa, s_x, s_y, c_x, c_y, NumColumns, NumRows]\), i.e., in addition to the interior camera parameters, the width (number of columns) and height (number of rows) of the image must be given. See section 2.2 on page 18 for a description of the interior camera parameters.
In the following, some hints for the determination of the initial values for the interior camera parameters are given:

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

$k$:
Use 0.0 as initial value. The calibrated value normally lies between -1000.0 and -50000.0 [1/m²] depending on the used lens.

$s_x$:
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:

<table>
<thead>
<tr>
<th></th>
<th>Full image (640*480)</th>
<th>Subsampling (320*240)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3”-Chip</td>
<td>0.0000055 m</td>
<td>0.00000110 m</td>
</tr>
<tr>
<td>1/2”-Chip</td>
<td>0.0000086 m</td>
<td>0.00000172 m</td>
</tr>
<tr>
<td>2/3”-Chip</td>
<td>0.0000110 m</td>
<td>0.00000220 m</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th></th>
<th>Full image (640*480)</th>
<th>Subsampling (320*240)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3”-Chip</td>
<td>0.0000055 m</td>
<td>0.00000110 m</td>
</tr>
<tr>
<td>1/2”-Chip</td>
<td>0.0000086 m</td>
<td>0.00000172 m</td>
</tr>
<tr>
<td>2/3”-Chip</td>
<td>0.0000110 m</td>
<td>0.00000220 m</td>
</tr>
</tbody>
</table>

$c_x$ 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:

<table>
<thead>
<tr>
<th></th>
<th>Full image (640*480)</th>
<th>Subsampling (320*240)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_x$</td>
<td>320.0</td>
<td>160.0</td>
</tr>
<tr>
<td>$c_y$</td>
<td>240.0</td>
<td>120.0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Full image (640*480)</th>
<th>Subsampling (320*240)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Width</td>
<td>640</td>
<td>320</td>
</tr>
<tr>
<td>Image Height</td>
<td>480</td>
<td>240</td>
</tr>
</tbody>
</table>
3.1.4 Determining the Interior Camera Parameters

The initial values for the exterior 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 exterior 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 exterior parameters yourself.

3.1.4 Determining the Interior 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.

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 exterior orientation. A typical sequence of images used for calibration is displayed in figure 16.

![Figure 16: A sequence of calibration images.](image)

If \( l \) images of the calibration plate are taken, the parameters to optimize are the interior parameters and \( l \) sets of the exterior 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 25 and section 3.1.3 on page 27 for a description on how to obtain these input values.

If the parameter EstimateParams is set to 'all', the interior parameters for the used camera are determined as well as the exterior parameters for each image. If the parameter is set to 'pose', only the exterior 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.

The computed average errors (Errors) give an impression of the accuracy of the calibration. The error values (deviations in x- and y-coordinates) are given in pixels.

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

### 3.1.5 Determining the Exterior Camera Parameters

The exterior camera parameters describe the relation between the measurement plane and the camera, i.e., only if the exterior 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 exterior 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 exterior 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 exterior 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 exterior camera parameters can be determined independently from the interior 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 exterior 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 interior parameters.
Since the pose of the calibration plate is determined by the operator \texttt{camera_calibration}, you can just pick the respective pose from the output parameter \texttt{NFinalPose}. In this way, interior and exterior parameters are determined in one single calibration step. The following code fragment from the program \texttt{hdevelop\camera_calibration\_multi\_image.dev} is an example for this easy way of determining the exterior parameters. Here, the pose of the calibration plate in the eleventh calibration image is determined. Please note that each pose consists of seven values.

\begin{verbatim}
NumImage := 11
Pose := NFinalPose[(NumImage-1)*7:(NumImage-1)*7+6]
\end{verbatim}

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 \texttt{set\_origin\_pose} can be used. The corresponding HALCON code is:

\begin{verbatim}
set_origin_pose (Pose, 0, 0, 0.00075, Pose)
\end{verbatim}

In general, the calibration plate can be oriented arbitrarily within the WCS (see \textit{figure 17}). 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.

\begin{verbatim}
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)
\end{verbatim}

If the advantages of using the HALCON calibration plate should be combined with the flexibility given by the separation of the interior and exterior camera parameters the \textbf{second} method for the determination of the exterior camera parameters can be used.

At first, only the interior parameters are determined as described in \textit{section 3.1.4} on page 29. This can be done, e.g., prior to the deployment of the camera.

This is shown in the example program \texttt{hdevelop\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 interior 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.
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 interior 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 exterior 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 exterior 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 interior camera parameter unchanged. Alternatively, EstimateParams can be set to the tuple ['alpha', 'beta', 'gamma', 'transx', 'transy', 'transz'], which also means that the six exterior parameters will be estimated. Again, the pose must be corrected for the thickness of the calibration plate as described above.

The program hdevelop\camera_calibration_exterior.dev shows how to determine the exterior camera parameters from a calibration plate that is positioned on the object’s surface.

First, the interior 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 interior parameters.

If it is necessary to perform the measurements within a given world coordinate system, the third case for the determination of the exterior 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 exterior 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 exterior parameters is given in the following program. First, the world coordinates of three points are set:

```plaintext
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 exterior camera parameters:

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

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

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

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

```plaintext
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 interior parameters.

### 3.1.6 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 ±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 × 0.65 mm², 10 × 10 mm², 200 × 200 mm²). Thus, you can choose the one that is optimal for your application. As a rule of thumb, the width of the calibration plate should be approximately one third of the image width. For example, if the image shows an area of 100 mm × 70 mm, the 30 × 30 mm² calibration plate would be appropriate. 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 25).

For test purposes, you can create a calibration plate yourself with the operator `create_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 `create_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.

### 3.1.7 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 3.1.4 on page 29).

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.

### 3.2 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 39).

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 center of the camera through the given point in the image plane, must be intersected with the measurement plane (see figure 18).

Figure 18: Intersecting the line of sight with the measurement plane.
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 \ldots_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 \ldots_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 \texttt{image_points_to_world_plane}. In the following code example, the row and column coordinates of pitch lines are transformed into world coordinates.

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

As input, the operator requires the interior and exterior 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 \texttt{Scale} (see also the description of the operator \texttt{image_to_world_plane} in section 3.3.1 on page 39). 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 26).

\[
\text{Scale} = \frac{\text{unit of resulting world coordinates}}{\text{unit of world coordinates of calibration target}}
\]  

(26)

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 \texttt{Scale} to 'm' (corresponding to the value 1.0, which could be set alternatively for the parameter \texttt{Scale}), 'cm' (0.01), 'mm' (0.001), 'microns' (1e-6), or '\( \mu \)m' (again 1e-6). Then, if the parameter \texttt{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 \( \mu \)m and the resulting coordinates have to be given in millimeters, the parameter \texttt{Scale} must be set to:

\[
\text{Scale} = \frac{\text{mm}}{\text{\( \mu \)m}} = \frac{1 \cdot 10^{-3} \text{m}}{1 \cdot 10^{-6} \text{m}} = 1000
\]  

(27)

### 3.2.3 World Coordinates for Contours

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

```plaintext
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 $CCS_{H\,WCS}$ 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 $CCS_{H\,WCS}$ 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 Radial Distortions Only

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

The procedure is to specify the original interior camera parameters and those of a virtual camera that does not produce radial distortions, i.e., with $\kappa = 0$. 
The easiest way to obtain the interior camera parameters of the virtual camera would be to simply set $\kappa$ to zero. This can be done directly by changing the respective value of the interior camera parameters.

\[
\text{CamParVirtualFixed} := \text{CamParOriginal} \\
\text{CamParVirtualFixed}[1] := 0
\]

Alternatively, the operator \texttt{change_radial_distortion_cam_par} can be used with the parameter \texttt{Mode} set to 'fixed' and the parameter \texttt{Kappa} set to 0.

\[
\text{change_radial_distortion_cam_par ('fixed', CamParOriginal, 0, CamParVirtualFixed)}
\]

Then, for the rectification of the segmentation results, the HALCON operator \texttt{change_radial_distortion_contours_xld} can be used, which requires as input parameters the original and the virtual interior camera parameters.

\[
\text{change_radial_distortion_contours_xld (Edges, EdgesRectifiedFixed, CamParOriginal, CamParVirtualFixed)}
\]

This changes the visible part of the scene (see figure 19b). To obtain virtual camera parameters such that the whole image content lies within the visible part of the scene, the parameter \texttt{Mode} of the operator \texttt{change_radial_distortion_cam_par} must be set to 'fullsize' (see figure 19c). Again, to eliminate the radial distortions, the parameter \texttt{Kappa} must be set to 0.

\[
\text{change_radial_distortion_cam_par ('fullsize', CamParOriginal, 0, CamParVirtualFullsize)}
\]

If the radial distortions are eliminated in the image itself using the rectification procedure described in section 3.3.2 on page 45, the mode 'fullsize' may lead to undefined pixels in the rectified image. The mode 'adaptive' (see figure 19d) slightly reduces the visible part of the scene to prevent such undefined pixels.

\[
\text{change_radial_distortion_cam_par ('adaptive', CamParOriginal, 0, CamParVirtualAdaptive)}
\]

### 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 radial and perspective distortions must be eliminated before the OCR can be applied.

#### 3.3.1 Transforming Images into the WCS

The operator \texttt{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 radial and no perspective distortions. It corresponds to an image captured by a camera that produces no radial distortions and that looks perpendicularly to the measurement plane.
Figure 19: Eliminating radial distortions: The original image overlaid with (a) edges extracted from the original image; (b) edges rectified by setting $\kappa$ to zero; (c) edges rectified with mode ‘fullsize’; (d) edges rectified with mode ‘adaptive’.

```plaintext
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`.

```plaintext
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 \( \text{Scale} \) (see also the description of the operator \texttt{image_points_to_world_plane} in section 3.2.2 on page 37). 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 28).

\[
\text{Scale} = \frac{\text{pixel size of rectified image}}{\text{unit of world coordinates of calibration target}}
\]  

(28)

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 \( \text{Scale} \) to \( 'm' \) (corresponding to the value 1.0, which could be set alternatively for the parameter \( \text{Scale} \), \( 'cm' \) (0.01), \( 'mm' \) (0.001), \( 'microns' \) (1e-6), or \( '\mu m' \) (again, 1e-6). Then, if the parameter \( \text{Scale} \) is set to, e.g., \( '\mu m' \), one pixel of the rectified image has a size that corresponds to an area of \( 1 \mu m \times 1 \mu m \) in the world. The parameter \( \text{Scale} \) 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 \( \text{Interpolation} \) specifies whether bilinear interpolation (\( '\text{bilinear}' \)) should be applied between the pixels in the input image or whether the gray value of the nearest neighboring pixel (\( '\text{none}' \)) should be used.

The rectified image \( \text{ImageWorld} \) 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 exterior camera parameters \( \text{CamPose} \) the position of the rectified image \( \text{ImageWorld} \) can be translated by applying the operator \texttt{set_origin_pose} to the exterior camera parameters. Arbitrary transformations can be applied to the exterior camera parameters based on homogeneous transformation matrices. See below for examples of how the exterior camera parameters can be set.

In figure 20, 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 20, 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 21, 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 21, 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 \texttt{gen_image_to_world_plane_map} in combination with \texttt{map_image} is much more efficient than the operator \texttt{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 \texttt{gen_image_to_world_plane_map}. Then, this map is used by the operator \texttt{map_image} to transform the image very efficiently.

The following example from hdevelop\transform_image_into_wcs.dev shows how to perform the transformation of images into the world coordinate system using the operators...
Figure 20: Projection of the image into the measurement plane.

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.

First, the size of the rectified image is defined.
Figure 21: Projection of the image into the measurement plane with part of the rectified image lying outside the image area.

\[
\text{WidthMappedImage} := 652 \\
\text{HeightMappedImage} := 494
\]
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.

\[
\text{Dist}_{\text{ICS}} := 1
\]

\[
\text{image\_points\_to\_world\_plane}\left(\text{CamParam, Pose, CenterRow, CenterCol, 1, CenterX, CenterY}\right)
\]

\[
\text{image\_points\_to\_world\_plane}\left(\text{CamParam, Pose, CenterRow+Dist}_{\text{ICS}}, \text{CenterCol, 1, BelowCenterX, BelowCenterY}\right)
\]

\[
\text{image\_points\_to\_world\_plane}\left(\text{CamParam, Pose, CenterRow, CenterCol+Dist}_{\text{ICS}}, 1, \text{RightOfCenterX, RightOfCenterY}\right)
\]

\[
\text{distance\_pp}\left(\text{CenterY, CenterX, BelowCenterY, BelowCenterX, Dist}_{\text{WCS}\_\text{Vertical}}\right)
\]

\[
\text{distance\_pp}\left(\text{CenterY, CenterX, RightOfCenterY, RightOfCenterX, Dist}_{\text{WCS}\_\text{Horizontal}}\right)
\]

\[
\text{Scale}_{\text{Vertical}} := \frac{\text{Dist}_{\text{WCS}\_\text{Vertical}}}{\text{Dist}_{\text{ICS}}}
\]

\[
\text{Scale}_{\text{Horizontal}} := \frac{\text{Dist}_{\text{WCS}\_\text{Horizontal}}}{\text{Dist}_{\text{ICS}}}
\]

\[
\text{Scale\_For\_Centered\_Image} := \frac{\left(\text{Scale}_{\text{Vertical}}+\text{Scale}_{\text{Horizontal}}\right)}{2.0}
\]

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

\[
\text{DX} := \text{CenterX}-\text{Scale\_For\_Centered\_Image}\times\text{Width\_Mapped\_Image}/2.0
\]

\[
\text{DY} := \text{CenterY}-\text{Scale\_For\_Centered\_Image}\times\text{Height\_Mapped\_Image}/2.0
\]

\[
\text{DZ} := 0
\]

\[
\text{set\_origin\_pose}\left(\text{Pose, DX, DY, DZ, Pose\_For\_Centered\_Image}\right)
\]

These calculations are implemented in the HDevelop procedure

\[
\text{procedure parameters\_image\_to\_world\_plane\_centered}\left(\text{: : CamParam, Pose, CenterRow, CenterCol, Width\_Mapped\_Image, Height\_Mapped\_Image, Scale\_For\_Centered\_Image, Pose\_For\_Centered\_Image}\right)
\]

which is part of the example program hdevelop\transform\_image\_into\_wcs.dev (see section 9.4 on page 112).

Finally, the image can be transformed.

\[
\text{gen\_image\_to\_world\_plane\_map}\left(\text{Map, CamParam, Pose\_For\_Centered\_Image, Width\_Original\_Image, Height\_Original\_Image, Width\_Mapped\_Image, Height\_Mapped\_Image, Scale\_For\_Centered\_Image, ’bilinear’}\right)
\]

\[
\text{map\_image}\left(\text{Image, Map, Image\_Mapped}\right)
\]

The second part of the example program hdevelop\transform\_image\_into\_wcs.dev shows how to set the parameters \textbf{Scale} and \textbf{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.
Then, the extent of the image in world coordinates is determined.

\[
\text{smallest_rectangle1_xld}\ (\text{ImageBorderWCS}, \text{MinY}, \text{MinX}, \text{MaxY}, \text{MaxX})
\]

\[
\text{ExtentX} := \text{MaxX} - \text{MinX}
\]

\[
\text{ExtentY} := \text{MaxY} - \text{MinY}
\]

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

\[
\text{ScaleX} := \frac{\text{ExtentX}}{\text{WidthMappedImage}}
\]

\[
\text{ScaleY} := \frac{\text{ExtentY}}{\text{HeightMappedImage}}
\]

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

\[
\text{ScaleForEntireImage} := \max([\text{ScaleX}, \text{ScaleY}])
\]

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

\[
\text{set_origin_pose}\ (\text{Pose}, \text{MinX}, \text{MinY}, 0, \text{PoseForEntireImage})
\]

These calculations are implemented in the HDevelop procedure

\[
\text{procedure}\ \text{parameters_image_to_world_plane_entire}\ (\text{Image} : \text{CamParam}, \text{Pose}, \text{WidthMappedImage}, \text{HeightMappedImage}, \text{ScaleForEntireImage}, \text{PoseForEntireImage})
\]

which is part of the example program hdevelop\transform\_image\_into\_wcs.dev (see section 9.5 on page 113).

If the object is not planar the projection map that is needed by the operator \text{map\_image} may be determined by the operator \text{gen\_grid\_rectification\_map}, which is described in section 8.3 on page 102.

If only the radial distortions should be eliminated the projection map can be determined by the operator \text{gen\_radial\_distortion\_map}, which is described in the following section.

### 3.3.2 Compensate for Radial Distortions Only

The principle of the compensation for radial distortions has already be described in section 3.2.6 on page 38.

If only one image must be rectified the operator \text{change\_radial\_distortion\_image} can be used. It is used analogously to the operator \text{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 \text{Region}. 
Again, the interior parameters of the virtual camera (with $\kappa = 0$) can be determined by setting only $\kappa$ to zero (see figure 22b) or by using the operator `change_radial_distortion_cam_par` with the parameter `Mode` set to 'fixed' (equivalent to setting $\kappa$ to zero; see figure 22b), 'adaptive' (see figure 22c), or 'fullsize' (see figure 22d).

Figure 22: Eliminating radial distortions: (a) The original image; (b) the image rectified by setting $\kappa$ to zero; (c) the image rectified with mode 'fullsize'; (d) the image rectified with mode 'adaptive'.

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 39. If a ROI is to be specified, it must be rectified separately (see section 3.2.4 on page 38).
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 21 on page 18) 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 23 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 measurement plane is indicated by $\Delta z$, the distance of the projection center from the measurement plane by $z$, and the displacement by $\Delta r$. The point $N$ indicates the perpendicular projection of the projection center ($PC$) onto the measurement plane.

![Figure 23: Displacement $\Delta r$ caused by a deviation of the object surface from the measurement plane.](image)

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 $\ldots_{\text{to_world_plane}}$ 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 $\Delta z$ of the object’s surface from the measurement plane...
plane, the displacement $\Delta r$ can be calculated by:

$$\Delta r = \Delta z \cdot \frac{r}{z}$$  \hspace{1cm} (29)

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

If you need to determine $\Delta r$ 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 $\Delta r$. 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 $WCSH_{CCS}$ needed for the above mentioned transformation, first, generate the homogeneous transformation matrix $CCS_{WCS}$ 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 (`hdevelop\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 $\Delta z$, the following conclusions can be drawn for $\Delta r$:

- $\Delta r$ increases with increasing $r$.
- If the measurement plane is more or less perpendicular to the optical axis, $\Delta r$ increases towards the image borders.
- At the point $N$, $\Delta r$ is always equal to zero.
• $\Delta r$ 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:

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

(30)

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 30 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.

$$\text{RelPose} := [0, 3.2, 0, -14, 0, 0, 0]$$

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

```plaintext
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.

```plaintext
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.

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

If the object is too 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 76.
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 exterior 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 39). 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 22, 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 machine vision with a binocular stereo system, please refer to section 7 on page 76.

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 60. 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 24. 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.

Typically, the following steps must be carried out:

1. Determination of the interior camera parameters for each camera separately.
2. Determination of the exterior 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 25 for an example). The overlapping area can be even smaller than depicted in figure 25, since the overlap is only necessary to ensure that there are no gaps in the resulting combined image.

4.2 Calibration

The calibration of the images can be broken down into two separate steps.
The first step is to determine the interior camera parameters for each of the cameras in use. This can be done for each camera independently, as described in section 3.1.4 on page 29.

The second step is to determine the exterior 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 26, which consists of as many calibration plates as the number of cameras that are used.

For the determination of the exterior 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 exterior camera parameters for each camera. See section 3.1.5 on page 30 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 26. The respective calibration images for the determination of the exterior camera parameters are shown in figure 27. Note that the relative position of the calibration plates with respect to each other must be known precisely.

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.
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 28.

The operators `gen_image_to_world_plane_map` and `map_image` are described in section 3.3.1 on page 39. 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.
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.

\[
ULRow := \text{HeightImage1} \times \text{BorderInPercent} / 100.0 \\
ULCol := \text{WidthImage1} \times \text{BorderInPercent} / 100.0
\]

Then, this point must be transformed into world coordinates.

\[
\text{image_points_to_world_plane (CamParam1, Pose1, ULRow, ULCol, 'm', ULX, ULY)}
\]

The resulting coordinates can be used directly, together with the shift that compensates the thickness of
the calibration plate (see section 3.1.5 on page 30) 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 29).

![Figure 29: Definition of the upper left corner of the first rectified image.](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-\text{BorderInPercent})/100.0
```

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

```
image_points_to_world_plane (CamParam1, Pose1, LowerRow, ULC01, '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 \( c_{p1}H_{ur1} \) (see figure 30), 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 \( c_{p1}H_{cp2} \) describes how the world coordinate system defined by the calibration plate in the first image is transformed into the world coordinate system defined by the calibration plate in the second image (figure 31). 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 32):
Figure 30: Definition of the upper right corner of the first rectified image.

Figure 31: Transformation between the two world coordinate systems, each defined by the respective calibration plate.

\[ c_{p2}^2 H_{ul2} = c_{p2}^2 H_{cp1} \cdot c_{p1}^1 H_{ur1} \]  (31)
This can be implemented in HDevelop as follows:

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

Figure 32: Definition of the upper left corner of the second rectified image.

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)
```

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 33.
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 34, into the two rectified images that are shown in figure 35.

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

\[ \text{Concat} := [\text{RectifiedImage1}, \text{RectifiedImage2}] \]

The resulting mosaic image is displayed in figure 36.
An example for such an application is given in figure 37. 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 50. 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 (hdevelop\mosaicking.dev) generates the mosaic image displayed in figure 43 on page 67. First, the images are read from file and collected in one tuple.

```
Images := []
for J := 1 to 10 by 1
  read_image (Image, ImgPath+ImgName+J"02")
  Images := [Images,Image]
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]
```

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]
  ImageF := Images[F]
  ImageT := Images[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.

\(^1\) 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.
Figure 37: A first example for image mosaicking.
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 50.

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). If the whole object shows little texture, the overlapping areas should be chosen larger.
- 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 38.

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 39).

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 40).

The radial distortions of the images are not compensated by the mosaicking process. Therefore, if radial 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 44 on page 67). To eliminate this effect, the radial distortions can be compensated before starting the mosaicking process (see section 3.3.2 on page 45).

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 69).
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 41** 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:

```
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 41**, 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):
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 42, 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 42: A configuration of ten overlapping images.](image)

Figure 43 shows such an image sequence of ten images of a BGA and the resulting mosaic image. Figure 44 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 radial distortions have been eliminated, respectively. The position of the cut-out is indicated in figure 43 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 → 6:

```
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 45).

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 46).

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 44, now the neighboring images fit better.
5.2 Definition of Overlapping Image Pairs

Figure 43: Ten overlapping images and the resulting (rigid) mosaic image.

<table>
<thead>
<tr>
<th>Unfavorable Configuration</th>
<th>With Radial Distortions</th>
<th>Radial Distortions Eliminated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good Configuration</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 44: Seam between image 5 and image 10 for various configurations.

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

```
1 → 2 → 3 → 4 → 5
```

```
6 → 7 → 8 → 9 → 10
```

Figure 45: Unfavorable configuration of 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.

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`
- `points_sojka`
- `saddle_points_sub_pix`

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

In figure 47, 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 four 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`. 

![Figure 46: Good configuration of image pairs.](image-url)
The operator `points_sojka` is specialized in the extraction of corner points. It is the fastest out of the four operators presented in this section.

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_sojka`, `saddle_points_sub_pix`).

### 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.

```python
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 48a, 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 48b).

![Figure 48: Separate images (a) and mosaic image (b) of a BGA.](image)

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]
```

### 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`.

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

With HALCON, it is possible to determine the pose of known 3D objects with a single camera. This is, e.g., necessary if you want to pick up objects that may be placed in an arbitrary position and orientation.

The basic idea is that the pose of the object, i.e., the exterior 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 `hdev/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 interior camera parameters and, if the pose of the object is to be determined relative to a given world coordinate system, the exterior camera parameters must be determined. See section 3.1 on page 24 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 49). This can be achieved, e.g., by using the operator `set_origin_pose`:

```c
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 7):

```c
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 $^wH_o$, which corresponds to the pose of the world coordinate system, world coordinates can be transformed into camera coordinates.

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 Application Note on Shape-Based Matching).

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 hdevelop\pose_of_known_3d_object.dev.

Now, the operator \texttt{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 \texttt{camera_calibration} via the parameters \texttt{NX}, \texttt{NY}, and \texttt{NZ} (3D object coordinates) as well as \texttt{NRow} and \texttt{NCol} (2D image coordinates). The known interior camera parameters are given via the parameter \texttt{StartCamParam} and an initial pose of the object with respect to the camera must be specified by the parameter \texttt{NStartPose}. The parameter \texttt{EstimateParams} must be set to \textquote{pose}'.

\begin{verbatim}
camera_calibration (ControlX, ControlY, ControlZ, RowCenter, ColCenter, 
CamParam, StartPose, \textquote{pose'}, _, PoseOfObject, Errors)
\end{verbatim}

You can determine the initial pose of the object from one image where the object is in a typical po-
sition and orientation. Place the HALCON calibration plate on the object and apply the operators \texttt{find_caltab} and \texttt{find_marks_and_pose} (see section 3.1.1 on page 25). 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 50), it is easy to determine the transformation matrices for the transformation of object coordinates into world coordinates and vice versa:

\begin{align}
^wH_o &= \ ^wH_c \cdot \ ^cH_o \\
&= (\ ^cH_w)^{-1} \cdot \ ^cH_o
\end{align}

where \(^wH_o\) is the homogeneous transformation matrix for the transformation of object coordinates into
world coordinates and \(^cH_w\) and \(^cH_o\) 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.

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

\begin{align}
^oH_w &= (^wH_o)^{-1}
\end{align}

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

\begin{verbatim}
pose_to_hom_mat3d (PoseOfWCS, camHwcs)
pose_to_hom_mat3d (PoseOfObject, camHobj)
\end{verbatim}

Then, the transformation matrix for the transformation of object coordinates into world coordinates is
derived.
Camera with optical center

Camera coordinate system \((x^c, y^c, z^c)\)

Object

Object coordinate system \((x^o, y^o, z^o)\)

World coordinate system \((x^w, y^w, z^w)\)

Figure 50: Pose of the object coordinate system and transformation between object coordinates and world coordinates.

```
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 `hdevelop\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 51). For the visualization of the coordinate systems, the HDevelop procedure

```plaintext
procedure disp_coordinate_system_3d (: : WindowHandle, CamPar,
        HomMat_WCS_to_CCS, NameCS: )
```

was used (see section 9.1 on page 110).

![Object coordinates and world coordinates for the four corners of the rectangular hole of the metal part.](image)

Figure 51: Object coordinates and world coordinates for the four corners of the rectangular hole of the metal part.

### 7 3D Machine 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 52 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
Figure 52: Top: Stereo camera system; Center: Stereo image pair; Bottom: Height map.
• 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.

The example programs used in this section are

- hdevelop\stereo_calibration_with_consistency_check.dev
- hdevelop\height_above_reference_plane_from_stereo.dev
- hdevelop\3d_information_for_selected_points.dev

### 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 interior parameters as shown in figure 53. 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

\[
\begin{align*}
    u_1 &= f \frac{x^c}{z^c} \\
    u_2 &= f \frac{x^c - b}{z^c}
\end{align*}
\]

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* $d$:

\[
d = (u_1 - u_2) = f \frac{b}{z^c}
\]

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
7.1 The Principle of Stereo Vision

Figure 53: Vertical section of a binocular stereo camera system.

$$z^c = \frac{f \cdot b}{d} \tag{39}$$

Note that the interior 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:

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 24.
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. 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 54).

It is very important to ensure that neither the interior 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 $\Delta z$, 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

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

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 $\Delta d$ 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 $1 \mu m$ for a camera with $7.4 \mu m$ pixel size.

In figure 55, 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 $\Delta d$ to be $1 \mu m$.

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 24). 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 24. 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 56.

The rules for taking the calibration images for the single camera calibration (see section 3.1.2 on page 26) apply accordingly.

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.

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
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 25. The only difference is that the operators must be applied to both stereo images separately.
If the rotation around the optical axis differs heavily between the two cameras the resulting stereo image pair may look like the one shown in figure 57. Here, it is advisable to check the consistency of the initial poses that are estimated by the operator `find_marks_and_pose`. This check is only necessary during the calibration phase. It can be carried out easily based on an asymmetrical feature of the calibration plate, e.g., a small black dot that indicates its upper left corner.

To do this check interactively, the estimated initial poses must be visualized. This can be done, e.g., by transforming the coordinate axes of the calibration plate coordinate system into the image while using the respective estimated initial pose. With this, each image pair where the asymmetrical feature does not lie in the same quadrant of the visualized coordinate system of the calibration plate can be rejected.
In the example program `hdevelop\stereo_calibration_with_consistency_check.dev`, a set of heavily distorted stereo image pairs is used for the calibration process. For each image, the calibration plate coordinate system, estimated by the operator `find_marks_and_pose`, is visualized (figure 57). The visualization is implemented within the example program as the HDevelop procedure

```plaintext
procedure visualize_results_of_find_marks_and_pose (Image: : WindowHandle,
    RCoord, CCoord, Pose,
    CamPar: )
```

which can be inserted into your HDevelop program, where you can adapt it to your special needs (see section 9.7 on page 115).

```plaintext
find_marks_and_pose (ImageL, CaltabL, 'caltab_30mm.descr',
    StartCamParL, StartThresh, DeltaThresh, MinThresh,
    Alpha, MinContLength, MaxDiamMarks, RCoordL,
    CCoordL, StartPoseL)
visualize_results_of_find_marks_and_pose (ImageL, WindowHandle1,
    RCoordL, CCoordL, StartPoseL,
    StartCamParL)
find_marks_and_pose (ImageR, CaltabR, 'caltab_30mm.descr',
    StartCamParR, StartThresh, DeltaThresh, MinThresh,
    Alpha, MinContLength, MaxDiamMarks, RCoordR,
    CCoordR, StartPoseR)
visualize_results_of_find_marks_and_pose (ImageR, WindowHandle2,
    RCoordR, CCoordR, StartPoseR,
    StartCamParR)
```

Now, the user can assess the consistency of the two poses, one for the left image and one for the right image, interactively. This can be done using the procedure

```plaintext
procedure interactive_pose_assessment (: : WindowHandle: IsConsistent)
```

Only the results from the accepted image pairs are accumulated into respective tuples for further use by the operator `binocular_calibration`.

```plaintext
if (IsConsistent)
    RowsL := [RowsL,RCoordL]
    ColsL := [ColsL,CCoordL]
    StartPosesL := [StartPosesL,StartPoseL]
    RowsR := [RowsR,RCoordR]
    ColsR := [ColsR,CCoordR]
    StartPosesR := [StartPosesR,StartPoseR]
endif
```

The use of results from image pairs for which the estimated initial poses are not consistent would lead to erroneous calibration results.

The initial values for the interior camera parameters can be determined as explained in section 3.1.3 on page 27.
7.3.3 Determining the Camera Parameters

The actual calibration of the stereo camera system is carried out with the operator \texttt{binocular_cal}.

\begin{verbatim}
\end{verbatim}

The only differences to the operator \texttt{camera_cal}, described in section 3.1.4 on page 29 are that the operator \texttt{binocular_cal} 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 interior camera parameters of that camera again (\texttt{binocular_cal} with \texttt{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 (\texttt{binocular_cal} with \texttt{EstimateParams} set to 'rel_pos').

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 58 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 59.

![Figure 58: Original stereo images.](image)
The rectification itself is carried out using the operators `gen_binocular_rectification_map` and `map_image`.

\[
\text{gen_binocular_rectification_map} (\text{MapL}, \text{MapR}, \text{CamParamL}, \text{CamParamR}, \text{cLPcR}, 1, '\text{geometric}', '\text{bilinear}', \text{RectCamParL}, \text{RectCamParR}, \text{CamPoseRectL}, \text{CamPoseRectR}, \text{RectLPosRectR})
\]

The operator `gen_binocular_rectification_map` requires the interior 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 used 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 60. 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**.

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

If the calibration was erroneous, e.g., because of inconsistent pose estimations (see section 7.3.2 on page 82), 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 61. This incorrect rec-
tification result has been achieved by accepting all image pairs used in the example program hde-
velop\stereo_calibration_with_consistency_check.dev, even those for which the poses have
not been estimated consistently.

![Incorrectly rectified stereo images.](image)

**Figure 61: Incorrectly rectified stereo images.**

### 7.4 Obtaining 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`.

If metrical information is required, the operator `binocular_distance` can be used to extract the dis-
tance of the object surface from the stereo camera system (see section 7.4.3 on page 93 for the definition
of the distance).

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 Rules for Taking Stereo Image Pairs

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 62).

![Rectified stereo images and matching result](image)

Figure 62: Rectified stereo images and matching result in a poorly textured area (regions where the matching process failed are displayed white).

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 63).

The following rules for the acquisition of the stereo image pairs should be considered:
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.
- Place the object such that repetitive patterns are not aligned with the rows of the rectified images.

### 7.4.2 Determining Disparities

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 39 on page 79).

Therefore, in case it is only necessary to know whether there are locally high objects it suffices to derive the disparities. This is done by using the operator `binocular_disparity`.

```markdown
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 7.4.3 on page 93), 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 ($N_{\text{image1}}^{\text{col}}$ and $N_{\text{image2}}^{\text{col}}$) and of the point $F$, which is the farthest away ($F_{\text{image1}}^{\text{col}}$ and $F_{\text{image2}}^{\text{col}}$), each in both rectified images.

Now, the values for the definition of the disparity range can be calculated as follows:

\[
\begin{align*}
\text{MinDisparity} & = N_{\text{image2}}^{\text{col}} - N_{\text{image1}}^{\text{col}} \\
\text{MaxDisparity} & = F_{\text{image2}}^{\text{col}} - F_{\text{image1}}^{\text{col}}
\end{align*}
\]

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 ≤ score ≤ 255) and 'ssd' (0 ≤ score ≤ 65025) return lower matching scores for better matches. In contrast, the method 'ncc' (-1 ≤ score ≤ 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 64, a rectified stereo image pair is displayed, from which the disparity and score images, displayed in figure 65 were derived.

![Figure 64: Rectified stereo images.](image)

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 65, 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 Distances

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. It can be determined by the operator `binocular_distance`, which is used analogously to the operator `binocular_disparity` described in the previous section.

```
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 66 shows the distance image and the respective score image for the rectified stereo pair of figure 64 on page 92. Because the distance is calculated directly from the disparities and from the camera parameters, the distance image looks similar to the disparity image (figure 65). What is more, the score images are identical, since the underlying matching process is identical.

It can be seen from figure 66 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 67).

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.
Figure 66: Distance image (left) and score image (right).

Figure 67: Distances of the object surface from the $x$-$y$-plane of the coordinate system of the first rectified camera.

These calculations are carried out in the procedure...
procedure tilt_correction (DistanceImage, RegionDefiningReferencePlane: 
    DistanceImageCorrected: : )

which is part of the example program hdevelop\height_above_reference_plane_from_stereo.dev (section 9.6 on page 114). 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 68.

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 68 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 68: 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).

### 7.4.4 Determining Distances or 3D Coordinates for Selected Points

If only the distances or the 3D coordinates of selected points should be determined, the operators disparity_to_distance, disparity_to_point_3d, or intersect_lines_of_sight can be used.

The operator disparity_to_distance 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.

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.

To calculate the \(x\), \(y\), and \(z\) coordinates of points, two different operators are available: The operator \texttt{disparity\_to\_point\_3d} derives the 3D coordinates from image coordinates and the respective disparities, whereas the operator \texttt{intersect\_lines\_of\_sight} uses the image coordinates from the two stereo images to determine the 3D position of points.

The operator \texttt{disparity\_to\_point\_3d} requires the camera parameters of the two rectified cameras as well as the image coordinates and the disparities of the selected points.

\[
\texttt{disparity\_to\_point\_3d (RectCamParL, RectCamParR, RectLPosRectR, RL, CL, DisparityOfSelectedPoints, X\_CCS\_FromDisparity, Y\_CCS\_FromDisparity, Z\_CCS\_FromDisparity)}
\]

The \(x\), \(y\), and \(z\) coordinates are returned in the coordinate system of the first rectified camera.

The operator \texttt{intersect\_lines\_of\_sight} determines 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.

\[
\texttt{intersect\_lines\_of\_sight (RectCamParL, RectCamParR, RectLPosRectR, RL, CL, RR, CR, X\_CCS\_FromIntersect, Y\_CCS\_FromIntersect, Z\_CCS\_FromIntersect, Dist)}
\]

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 \texttt{find\_caltab}, \texttt{find\_marks\_and\_pose}, and \texttt{camera\_calibration}.

\[
\texttt{find\_caltab (ImageRectifiedL, CaltabL, \textquote{caltab\_30mm.descr}, SizeGauss, MarkThresh, MinDiamMarks)}
\]

\[
\texttt{find\_marks\_and\_pose (ImageRectifiedL, CaltabL, \textquote{caltab\_30mm.descr}, RectCamParL, StartThresh, DeltaThresh, MinThresh, Alpha, MinContLength, MaxDiamMarks, RCoordL, CCoordL, StartPoseL)}
\]

\[
\texttt{camera\_calibration (X, Y, Z, RCoordL, CCoordL, RectCamParL, StartPoseL, \textquote{pose}, _, PoseOfCalibrationPlate, Errors)}
\]

The resulting pose can be converted into a homogeneous transformation matrix.
If necessary, the transformation matrix can be modified with the operators \texttt{hom\_mat3d\_rotate\_local}, \texttt{hom\_mat3d\_translate\_local}, and \texttt{hom\_mat3d\_scale\_local}.

The homogeneous transformation matrix must be inverted in order to represent the transformation from the (rectified) camera coordinate system into the WCS.

Finally, the 3D coordinates can be transformed using the operator \texttt{affine\_trans\_point\_3d}.

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

which is part of the example program \texttt{hdevelop\_3d\_information\_for\_selected\_points.dev}.

The resulting homogeneous transformation matrix can be used as input for the operator \texttt{affine\_trans\_point\_3d}, as shown above.

### 8 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 the radial lens distortions as well as by non-radial lens distortions, a non-flat object surface, or by any other reason. In the first two cases, i.e., if the object surface is flat and the camera shows only radial distortions, the rectification can be carried out very precisely as described in section 3.3.1 on page 39. 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 (\texttt{hdevelop\_grid\_rectification\_ruled\_surface.dev}) shows how the grid rectification can be used to rectify the image of a cylindrically shaped object (\textit{figure 69}). In the rectified image (\textit{figure 69b}), the bar code could be read correctly, which was not possible in the original image (\textit{figure 69a}).
First, this pattern, which is called rectification grid, must be created with the operator `create_rectification_grid`.

```python
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 70a.

Now, the object must be wrapped with the rectification grid and an image of the wrapped object must be
8.1 Basic Principle

taken (figure 70b).

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:

```plaintext
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 69b). This mapping is performed by the operator `map_image`.

```plaintext
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 8.3 on page 102, 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.

8.1 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 71).

In the distorted image, the black and white fields do not appear as squares (figure 72a) because of the non-planar object surface, the perspective distortions, and the lens distortions.

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 72b). These corners must be connected along the borders of the black and white fields with the operator `connect_grid_points` (figure 72c). Finally, the connecting lines must be combined into meshes (figure 72d) 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 8.4 on page 105 for an example).

The mapping is determined such that the distorted rectification grid will be mapped into its original undistorted geometry (figure 73). 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 8.2 for details).
8.2 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.
- 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 74a), i.e., whose grid size is too large, leads to errors in the rectified image (figure 74b).

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 75a), the rectified image (figure 75b) will contain distortions because of the bilinear interpolation within the meshes.
8.2 Rules for Taking Images of the Rectification Grid

Figure 72: 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 73: Mapping of the distorted rectification grid (a) into the undistorted rectification grid (b).
8.3 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 (hdevelop\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 8.2 on page 100. Figure 76a shows an image of a cylindrical object and figure 76b shows the same object wrapped by the rectification grid.

Then, the rectification grid is searched in this image with the operator `find_rectification_grid`.

\[
\text{find_rectification_grid (Image, GridRegion, MinContrast, Radius)}
\]

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 77a).

\[
\text{reduce_domain (Image, GridRegion, ImageReduced)}
\]

Figure 76: Cylindrical object: a) Without and b) with rectification grid.

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 77b).

\[
\text{saddle_points_sub_pix (ImageReduced, 'facet', SigmaSaddlePoints, Threshold, Row, Col)}
\]

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 77c). This is done with the operator `connect_grid_points`.
Figure 77: 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.

\texttt{connect\_grid\_points (ImageReduced, ConnectingLines, Row, Col, SigmaConnectGridPoints, MaxDist)}

Again, the parameter \texttt{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 \([\text{sigma\_min, sigma\_max, sigma\_step}]\) is passed instead of only one value, the operator \texttt{connect\_grid\_points} tests every sigma within the given range from \texttt{sigma\_min} to \texttt{sigma\_max} with a step size of \texttt{sigma\_step} and chooses the sigma that causes the largest number of connecting lines. The same happens when a tuple of only two values \texttt{sigma\_min} and \texttt{sigma\_max} is passed. However, in this case a fixed step size of 0.05 is used. The parameter \texttt{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 78 shows the connecting lines if the parameter \texttt{MaxDist} has been selected inappropriately: In figure 78a, \texttt{MaxDist} has been selected to small, whereas in figure 78b, it has been selected too large.

Figure 78: Connecting lines: Parameter \texttt{MaxDist} selected a) too small and b) too large.

Then, the rectification map is determined from the distorted grid with the operator \texttt{gen\_grid\_rectification\_map}.
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 × 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 79). Note that the objects must appear at exactly the same position in the distorted images.

![Rectification Grids](image)

**Figure 79: Rectified images: a) Rectification grid; b) object.**

### 8.4 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 80a shows the reference image and figure 80b 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 (hdevelop\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
8.4 Using Self-Defined Rectification Grids

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.

```plaintext
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.

```plaintext
MinValue := 0
MaxValue := HeightTest+GridSpacing
sample_corresponding_values (RowTest, RowRef-0.5, MinValue, MaxValue,
 GridSpacing, RefGridRowValues)
```

The interpolation is performed within the procedure

```plaintext
procedure sample_corresponding_values (: : Values, CorrespondingValues,
 MinValue, MaxValue,
 InterpolationInterval:
 SampledCorrespondingValues)
```

which is part of the example program hdevelop\grid_rectification_arbitrary_distortion.dev.

Now, the distorted grid is generated row by row.

```plaintext
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\(^2\).

```plaintext
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.

\(^2\)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.
cell, which must be defined by the parameter \textbf{GridSpacing}. Possibly, the size of the mapped reference image must be adapted to the size of the test image.

\begin{verbatim}
map_image (ImageRef, Map, ImageMapped)
crop_part (ImageMapped, ImagePart, 0, 0, WidthTest, HeightTest)
\end{verbatim}

Finally, the test image can be subtracted from the mapped reference image.

\begin{verbatim}
sub_image (ImagePart, ImageTest, ImageSub, 1, 128)
\end{verbatim}

\textbf{Figure 81} shows the resulting difference image. In this case, missing parts appear dark while the smudges appear bright.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figures/difference.png}
\caption{Difference image: a) The entire image overlaid with a rectangle that indicates the position of the cut-out. b) A cut-out.}
\label{fig:diff}
\end{figure}

The differences between the test image and the reference image can now be extracted easily from the difference image with the operator \texttt{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 \texttt{dyn_threshold}.

\textbf{Figure 82} shows the differences in a cut-out of the reference image (\texttt{figure 82a}) and of the test image (\texttt{figure 82b}). The markers near the left border of \texttt{figure 82b} indicate the vertical position of the components 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.

Figure 82: 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.
9 HDevelop Procedures Used in this Application Note

9.1 disp_coordinate_system_3d

procedure disp_coordinate_system_3d (: : WindowHandle, CamPar,
                                     HomMat_WCS_to_CCS, NameCS: )
  dev_set_window (WindowHandle)
  ArrowLength := 0.02
  ArrowX_WCS := [0, ArrowLength, 0, 0]
  ArrowY_WCS := [0, 0, ArrowLength, 0]
  ArrowZ_WCS := [0, 0, 0, ArrowLength]
  affine_trans_point_3d (HomMat_WCS_to_CCS, ArrowX_WCS, ArrowY_WCS,
                           ArrowZ_WCS, ArrowX_CCS, ArrowY_CCS, ArrowZ_CCS)
  project_3d_point (ArrowX_CCS, ArrowY_CCS, ArrowZ_CCS, CamPar, ArrowRow,
                    ArrowColumn)
  disp_arrow (WindowHandle, ArrowRow[0], ArrowColumn[0], ArrowRow[1],
               ArrowColumn[1], 1)
  disp_arrow (WindowHandle, ArrowRow[0], ArrowColumn[0], ArrowRow[2],
               ArrowColumn[2], 1)
  disp_arrow (WindowHandle, ArrowRow[0], ArrowColumn[0], ArrowRow[3],
               ArrowColumn[3], 1)
  set_tposition (WindowHandle, ArrowRow[0], ArrowColumn[0])
  write_string (WindowHandle, NameCS)
  set_tposition (WindowHandle, ArrowRow[1], ArrowColumn[1])
  write_string (WindowHandle, 'x')
  set_tposition (WindowHandle, ArrowRow[2], ArrowColumn[2])
  write_string (WindowHandle, 'y')
  set_tposition (WindowHandle, ArrowRow[3], ArrowColumn[3])
  write_string (WindowHandle, 'z')
  return ()
end
9.2 gen_hom_mat3d_from_three_points

```plaintext
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 ()
end
```

This procedure uses the procedure

```plaintext
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 ()
end
```

9.3 interactive_pose_assessment

```plaintext
procedure interactive_pose_assessment (: : WindowHandle: IsConsistent)
    set_tposition (WindowHandle, 24, 12)
    dev_set_color ('green')
    write_string (WindowHandle,
                  'Has the pose been estimated consistently?')
    set_tposition (WindowHandle, 44, 30)
    write_string (WindowHandle, 'If yes -> left mouse button')
    set_tposition (WindowHandle, 64, 30)
    write_string (WindowHandle, 'If no -> right mouse button')
    get_mbutton (WindowHandle, _, _, Button)
    if (Button=1)
        IsConsistent := true
    else
        IsConsistent := false
    endif
    return ()
end
```
9.4 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 ()
end
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 ()
end
9.6 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 () 
end
9.7 visualize_results_of_find_marks_and_pose

procedure visualize_results_of_find_marks_and_pose (Image: : WindowHandle,
                             RCoord, CCoord, Pose,
                             CamPar: )

  dev_set_window (WindowHandle)
  dev_display (Image)
  dev_set_color ('red')
  disp_cross (WindowHandle, RCoord, CCoord, 6, 0)
  ArrowLength := 0.02
  ArrowX_CPCS := [0,ArrowLength,0]
  ArrowY_CPCS := [0,0,ArrowLength]
  ArrowZ_CPCS := [0,0,0]
  pose_to_hom_mat3d (Pose, HomMat_CPCS_CCS)
  affine_trans_point_3d (HomMat_CPCS_CCS, ArrowX_CPCS, ArrowY_CPCS,
                        ArrowZ_CPCS, ArrowX_CCS, ArrowY_CCS, ArrowZ_CCS)
  project_3d_point (ArrowX_CCS, ArrowY_CCS, ArrowZ_CCS, CamPar, ArrowRow,
                   ArrowColumn)
  dev_set_color ('green')
  disp_arrow (WindowHandle, ArrowRow[0], ArrowColumn[0], ArrowRow[1],
              ArrowColumn[1], 1)
  disp_arrow (WindowHandle, ArrowRow[0], ArrowColumn[0], ArrowRow[2],
              ArrowColumn[2], 1)
  set_tposition (WindowHandle, ArrowRow[1], ArrowColumn[1])
  write_string (WindowHandle, 'x')
  set_tposition (WindowHandle, ArrowRow[2], ArrowColumn[2])
  write_string (WindowHandle, 'y')
  dev_set_color ('white')
  return ()
end