Großer Beleg

Position Estimation Of A Mobile Robot Using A Single Vehicle-Mounted Camera

bearbeitet von
Tobias Pietzsch
geboren am 25. Dezember 1975 in Rodewisch

Technische Universität Dresden
Fakultät Informatik

eingereicht am 19. November 2002

Contents

Chapter 1

Introduction

To navigate a mobile robot, one of the basic problems that have to be addressed is reliable position estimation. A variety of sensors is at hand to aid in the solution of this task, such as odometry sensors, laser range finders, cameras or global positioning systems. This paper looks into how this problem can be solved using vision sensors. An algorithm is devised, that reconstructs a sequence of robot poses, more specifically a sequence of camera positions and orientations from images provided by a calibrated vehicle-mounted camera. The methods considered here are based on image correspondences alone, so there is no need for an environment model. There is an overall scale ambiguity in the reconstructed path. To update the solution to correct scale, for instance distance information from odometry sensors can be incorporated.

Another goal was to produce a working implementation of the algorithm. For this purpose the *Gandalf Vision and Numerical Library* was used, which is available under the GNU Lesser General Public License (LGPL) from `http://gandalf-library.sourceforge.net/`. Source code fragments are given at the end of the sections, illustrating how the respective concepts were realized. Several experiments have been performed using simulated data. Results are presented at the end of each chapter.

The principal approach of the algorithm is to look at two images at once, and compute an essential matrix between them. From this matrix, the relative position and orientation of the cameras that were used to take the images can be extracted. The camera pairs obtained in this way will be "stitched together" then, to form a complete sequence. The main advantage of the approach is its computational efficiency.

The reconstruction of a path of camera positions will be solved in several steps, starting from point correspondences between two images:

- Obtain the relative position and orientation of two robot poses from a pair of consecutive images taken by the robots camera. Eight point correspondences between the two images are needed, to compute an

unique solution to this problem.

- To obtain relative position and orientation between three robot poses, combine two consecutive position-orientation pairs. It will be seen, that position and orientation between two cameras can only be determined up to a scale factor. Therefore, it is necessary that point correspondences between *three* images are known to solve this part of the problem. Only one such correspondence is needed to complete the task.

- Finally, these position-orientation triples are combined to a complete path.

Since the overall scale is still undetermined, additional information must be used, to find the correct scale of the solution. How this can be done is not within the scope of this paper.

Some assumptions are made to constrain the complexity of the problem:

- **Finding point correspondences between images.** The task of identifying image points across images, that are projections of the same world point must be solved before the algorithms described in this paper can be applied. It is not considered here and it should be noted that this is by no means a trivial task. Devising a feature-tracker that accomplishs this task is subject of further work.

- **The pinhole camera model is valid.** In reality, most cameras suffer from radial distortion. It is assumed here, that radial distorion is negligible small or has been removed in a preprocessing step. The Gandalf Library provides modelling and removal of radial distortion.

- **The world is static.** The basic unit of information that is used is *two views of the same scene*. Since "two views" means *two images taken by the robots camera at different instants in time* here, the phrase "the same scene" implies that there are no moving objects in the world other than the robot itself.

- **The robot navigates on a plane.** More specifically: the camera is moving in a known plane and rotating about that planes normal. This additional restriction applies only to the algorithms described in Sections 4.3 and 4.4 and is explicitly stated again there.

This paper is organized as follows: Chapter 2 describes the underlying geometry of two views of the same scene. It introduces the camera model that is used and the concept of the essential matrix. Chapter 3 shows how relative position and orientation can be extracted once the essential matrix is obtained. Chapter 4 describes and evaluates several linear methods estimating the essential matrix from a pair of images. Chapter 5 explains

methods to combine two consecutive position-orientation pairs to a triple of cameras, determined up to a common scale factor. Chapter 6 explains how the camera triples are combined to form a complete path. Finally, experimental results are presented demonstrating the complete solution.

Chapter 2

The Geometry of Two Views

This chapter deals with the geometric properties that arise when two cameras are looking at a scene, or equivalently one camera looking at a static scene from different positions at different times. The concepts presented here are adapted from [2, 6, 1]. The notation has been taken from [2]: Symbols in boldface are used to denote vectors (like $\mathbf{x}$,$\mathbf{X}$). Symbols in Sans Serif denote matrices (line P,E). Many equations contain homogeneous entities where equality is only up to scale. Nevertheless, the = sign will be used.

This chapter starts out with the foundations by describing the camera model used and the concepts of epipolar geometry. Finally, the fundamental and essential matrix are introduced.

2.1 Camera projection matrices

A camera is a mapping from the 3D world to a 2D image. A camera model is a matrix with particular properties that represents this mapping:

$$\mathbf{x} = \mathsf{P}\mathbf{X} \tag{2.1}$$

where P is a homogeneous 3×4 matrix, referred to as the *camera projection matrix* which maps homogeneous 4-vectors $\mathbf{X}$ to homogeneous 3-vectors $\mathbf{x}$ (image points). The model used here is also known as the pinhole camera model. First an explanation of the central projection is given, which is generalized by introducing changes to the world and image coordinate systems later.

2.1.1 Central projection

As shown in Figure 2.1, the camera centre (centre of projection) is at the origin of an Euclidean coordinate system and the direction of view is parallel the Z-axis. An image of the world is projected onto the *image plane*, which is a plane $Z = f$. The parameter f is referred to as *focal length*. The line

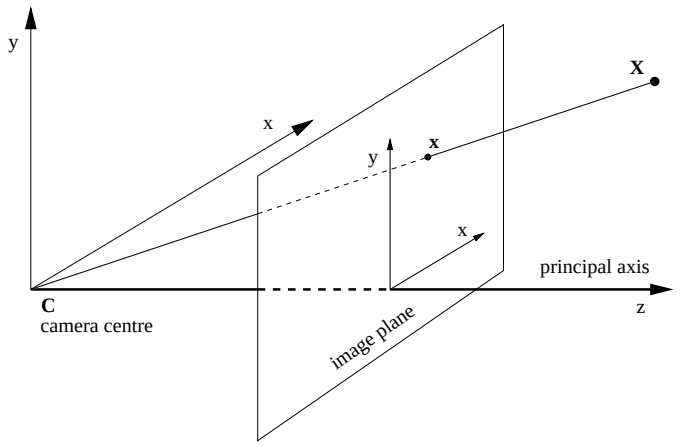

Figure 2.1: *The pinhole camera model.*

from the camera centre perpendicular to the image plane is known *principal axis*, the point where it meets the image plane is referred to as the *principal point*.

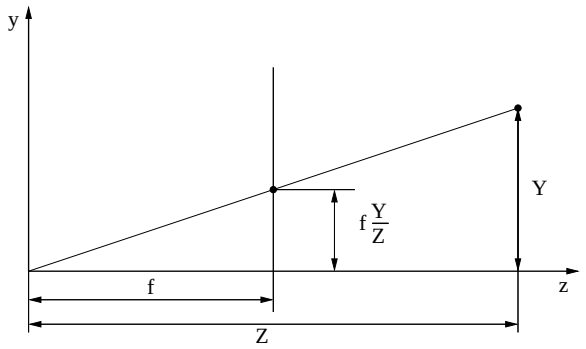

Figure 2.2: *Perspective projection.*

The image of a world point $\mathbf{X} = (X, Y, Z)^\top$ is the point $(x, y)^\top$ on the image plane, where the line joining $\mathbf{X}$ and the camera centre meets the image plane. From similar triangles, illustrated in Figure 2.2, the image of a world point is given by

$$x = f\frac{X}{Z} \tag{2.2a}$$

$$y = f\frac{Y}{Z}. \tag{2.2b}$$

This relation is nonlinear in Euclidean space, but using homogeneous coor-

dinates it may be expressed as the linear mapping

$$\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{2.3}$$

This is essentially Equation (2.1), written out.

Note that up to this point, world and image coordinates are measured in the coordinate frame of the camera. To remove this limitation, the 3×4 projection matrix is split up into

$$\mathsf{P} = \mathsf{K}_{3\times3} \, [\mathsf{I}|\mathbf{0}] \, \mathsf{T}_{4\times4}. \tag{2.4}$$

The 4×4 matrix T realizes a transformation from the world reference frame into the coordinate frame of the camera. The 3×4 matrix $[\mathsf{I}|\mathbf{0}]$ projects points measured in the camera coordinate frame into the plane $Z = 1$. Finally, the 3×3 matrix K is a transformation from camera coordinates into the image coordinate frame. The remainder of this section provides details on K and T.

2.1.2 Transformations of image coordinates

Figure 2.1 suggests that the origin of the image coordinate system is identical with the principal point, which might not be the case in practice. Instead, the image of $(X, Y, Z)^\top$ is

$$x = f\frac{X}{Z} + p_x \tag{2.5a}$$

$$y = f\frac{Y}{Z} + p_y \tag{2.5b}$$

where $(p_x, p_y)^\top$ are the (image) coordinates of the principal point. The camera projection matrix now can be represented as

$$\mathsf{P} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix}. \tag{2.6}$$

Furthermore, the scale of the world and image coordinate systems are not the same, in general. Image coordinates are measured in pixels. Since the pixels need not even be square, scale factors in both axial directions must be introduced. The camera projection matrix becomes

$$\mathsf{P} = \begin{bmatrix} f_x & & x_0 & 0 \\ & f_y & y_0 & 0 \\ & & 1 & 0 \end{bmatrix} \tag{2.7}$$

where f_x and f_y represent f measured in pixels in the x and y directions respectively, and $(x_0, y_0)^\top$ are the coordinates of the principal point, measured in pixels too.

The parameters introduced above relate to the camera itself and are independent of the position and orientation of the camera in the world coordinate frame. Therefore, they are referred to as *internal parameters* or *internal orientation*. They can be determined by camera calibration. The matrix

$$\mathsf{K} = \begin{bmatrix} f_x & & x_0 \\ & f_y & y_0 \\ & & 1 \end{bmatrix} \tag{2.8}$$

is also known as the *camera calibration matrix*.

2.1.3 Transformations of world coordinates

In practice, the coordinate frames of the camera and the world are not identical. The two systems are related by a rotation and a translation, in general. It can be safely assumed that the scales of the systems are equal; scaling is part of the internal orientation.

The representations of a point in world coordinates $\mathbf{X}$ and in camera coordinates $\mathbf{X}_{cam}$ are related by

$$\mathbf{X}_{cam} = \begin{bmatrix} \mathsf{R} & -\mathsf{R}\widetilde{\mathbf{C}} \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X} \tag{2.9}$$

where $\widetilde{\mathbf{C}}$ are the Euclidean coordinates of the camera centre in the world coordinate frame, and R is the 3×3 rotation matrix representing the orientation of the camera.

Thus, the image of a point $\mathbf{X}$ in the world coordinate frame is

$$\mathbf{x} = \mathsf{K} \left[\mathsf{R} | -\mathsf{R}\widetilde{\mathbf{C}} \right] \mathbf{X}. \tag{2.10}$$

The camera matrix becomes

$$\mathsf{P} = \mathsf{K} \left[\mathsf{R} | \mathbf{t} \right] \tag{2.11}$$

with $\mathbf{t} = -\mathsf{R}\widetilde{\mathbf{C}}$.

The parameters R and $\mathbf{t}$ are the *external parameters* or *exterior orientation* of the camera. The methods presented in this paper aim at the computation of these parameters, while the internal parameters are assumed to be known a priori.

2.2 Epipolar geometry

Epipolar geometry is the projective geometry between two views. It only depends on the internal parameters of the two cameras and their relative orientation. It is independent of the structure of the scene.

The defining relationship is the one between a point in the first image, and the corresponding *epipolar line* in the second. This relationship is projective linear and thus can be expressed by a 3×3 *fundamental matrix*. It encodes all constraints arising from two views of the same scene.

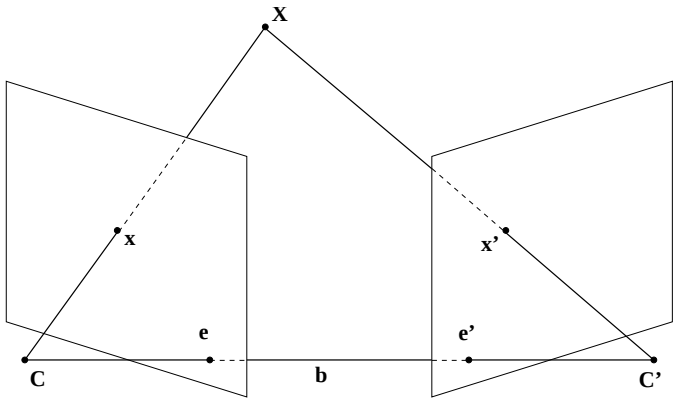

Figure 2.3: *Two cameras looking at the same scene.*

Consider two cameras looking at the same scene as shown in Figure 2.3. Entities in the second view are marked with a prime. The camera centres are $\mathbf{C}$ and $\mathbf{C}'$. The line joining the camera centres is the *baseline*, $\mathbf{b}$. The intersections $\mathbf{e}$ and $\mathbf{e}'$ of the baseline with the image planes are called *epipoles*. An epipole is the projection of one camera centre into the image plane of the other camera.

A scene point $\mathbf{X}$ is projected to the image points $\mathbf{x}$ and $\mathbf{x}'$. It can be seen in Figure 2.3, that $\mathbf{C}$, $\mathbf{C}'$, $\mathbf{x}$, $\mathbf{x}'$ and $\mathbf{X}$ are coplanar. The baseline lies in the same plane, too.

Any plane containing the baseline is an *epipolar plane*. The intersection of an epipolar plane with one of the image planes is called *epipolar line*. Clearly, all the epipolar lines in a image intersect in the epipole.

If the position of the image point $\mathbf{x}$ is known, the position of the corresponding point $\mathbf{x}'$ is constrained as follows: The position of the world point $\mathbf{X}$, which projects to $\mathbf{x}$, is constrained to the line through $\mathbf{C}$ and $\mathbf{x}$. The position of $\mathbf{x}'$, the image of $\mathbf{X}$ in the second view, is obviously constrained to the projection of this line into the second image plane (see Figure 2.4), which is the *epipolar line corresponding to* $\mathbf{x}$. In other words, the epipolar line corresponding to $\mathbf{x}$ is determined by the intersection of the second image plane and the epipolar plane through $\mathbf{x}$ (and both camera centres).

This means, that there is a mapping from image points in the first view to epipolar lines in the second. This mapping is projective linear [2, 6], and thus may be expressed in terms of a 3×3 matrix, known as the *fundamental matrix*.

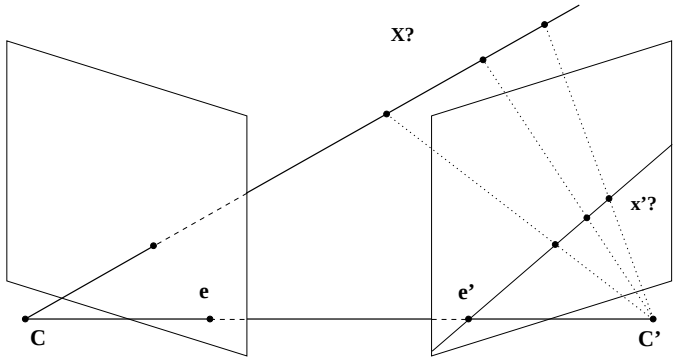

Figure 2.4: *The position of* $\mathbf{x}'$ *is constrained to lie on the epipolar line* $\mathbf{l}'$ *corresponding to* $\mathbf{x}$.

2.3 Cross products

As a short interlude, this section describes the representation of the cross product by a skew-symmetric matrix $[\mathbf{a}]_\times$, a concept that will be needed in the following material.

For a 3-vector $\mathbf{a} = (a_1, a_2, a_3)^\top$, the corresponding skew-symmetric matrix is defined as

$$[\mathbf{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}.$$

The cross product of two 3-vectors $\mathbf{a} \times \mathbf{b}$ is the vector $(a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1)^\top$. As one easily verifies, cross products are related to skew-symmetric matrices according to

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = (\mathbf{a}^\top [\mathbf{b}]_\times)^\top.$$

A particular result, required subsequently, is given without proof here:

Result 1 *For any vector* $\mathbf{t}$ *and non-singular matrix* M

$$[\mathbf{t}]_\times \mathsf{M} = \mathsf{M}^{-\top} [\mathsf{M}^{-1}\mathbf{t}]_\times.$$

For a proof, see [2, *p. 555*].

2.4 The fundamental matrix

As already stated in Section 2.2, the fundamental matrix F is a mapping from points $\mathbf{x}$ in one view to epipolar lines $\mathbf{l}'$ in the other:

$$\mathbf{l}' = \mathsf{F}\mathbf{x}, \tag{2.12}$$

and $\mathbf{x}'$, the image point corresponding to $\mathbf{x}$, lies on $\mathbf{l}'$, which can be expressed as

$$\mathbf{x}'^{\top}\mathbf{l}' = 0. \tag{2.13}$$

Combining these equations yields the defining equation for the fundamental matrix.

Result 2 *For any pair of corresponding points* $\mathbf{x} \leftrightarrow \mathbf{x}'$

$$\mathbf{x}'^{\top}\mathsf{F}\mathbf{x} = 0. \tag{2.14}$$

Epipolar geometry only depends on the internal parameters and relative orientation of the two cameras, as it has been shown in Section 2.2. The fundamental matrix will be derived now in terms of the two camera projection matrices P and P'.

Since F depends on the *relative* orientation of the cameras only, the world coordinate frame may be chosen arbitrarily. So, the coordinate system of the first camera is chosen as the reference frame, and the projection matrices become

$$\mathsf{P} = \mathsf{K}\,[\mathsf{I}|\mathbf{0}] \tag{2.15a}$$

$$\mathsf{P}' = \mathsf{K}'\,[\mathsf{R}|\mathbf{t}]\,. \tag{2.15b}$$

A world point $\mathbf{X} = (X, Y, Z, 1)^{\top}$ is projected into both image planes, to $\mathbf{x} = \mathsf{P}\mathbf{X}$ and $\mathbf{x}' = \mathsf{P}'\mathbf{X}$. From the first camera, it is obtained that

$$\mathsf{K}^{-1}\mathbf{x} = [\mathsf{I}|\mathbf{0}]\,\mathbf{X} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}. \tag{2.16}$$

From the second view:

$$\begin{aligned} \mathbf{x}' &= \mathsf{K}'\,[\mathsf{R}|\mathbf{t}]\,\mathbf{X} \\ &= \mathsf{K}'\,[\mathsf{R}|\mathbf{t}] \left[\begin{pmatrix} X \\ Y \\ Z \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right] \\ &= \mathsf{K}'\mathsf{R} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathsf{K}'\mathbf{t} \end{aligned} \tag{2.17}$$

Substituting (2.16) in (2.17) yields

$$\mathbf{x}' = \mathsf{K}'\mathsf{R}\mathsf{K}^{-1}\mathbf{x} + \mathsf{K}'\mathbf{t}. \tag{2.18}$$

This equation implies that $\mathbf{x}'$ lies on the line going through $\mathsf{K}'\mathsf{R}\mathsf{K}^{-1}\mathbf{x}$ and $\mathsf{K}'\mathbf{t}$. The collinearity of these three points may be expressed using the scalar triple product as

$$\mathbf{x}'^{\top}\Big(\big(\mathsf{K}'\mathbf{t}\big) \times \big(\mathsf{K}'\mathsf{R}\mathsf{K}^{-1}\mathbf{x}\big)\Big) = 0 \tag{2.19}$$

$$\mathbf{x}'^{\top}[\mathsf{K}'\mathbf{t}]_{\times}\mathsf{K}'\mathsf{R}\mathsf{K}^{-1}\mathbf{x} = 0. \tag{2.20}$$

Using Result 1, this transforms to

$$\mathbf{x}'^{\top}\left(\mathsf{K}'^{-\top}[\mathbf{t}]_{\times}\mathsf{R}\mathsf{K}^{-1}\right)\mathbf{x} = 0. \tag{2.21}$$

Obviously, this is the form of Result 2 and an expression for the fundamental matrix is

$$\mathsf{F} = \mathsf{K}'^{-\top}[\mathbf{t}]_{\times}\mathsf{R}\mathsf{K}^{-1}. \tag{2.22}$$

The mapping F is clearly not invertible since all $\mathbf{x}$ on an epipolar line in the first view are mapped to the same corresponding epipolar line in the second view. This implies that F is not of full rank. In general, F must have rank 2 (rank 1 corresponds to an impossible configuration, where the image planes are identical and the baseline lies in both [6]). F maps from a 2-dimensional (the first image plane) into a 1-dimensional (the pencil of epipolar lines in the second view) projective space.

It should be noted, that F is a homogeneous entity, and therefore is defined only up to a non-zero scale factor. The fundamental matrix has seven degrees of freedom: the eight independent ratios of the elements of F minus one for the rank = 2 constraint. That is, F is determined by seven corresponding pairs $\mathbf{x} \leftrightarrow \mathbf{x}'$ in general configuration.

2.5 The essential matrix

The *essential matrix* is a specialization of the fundamental matrix to the case, that the camera calibration matrices (K and K') are known. It is the equivalent of the fundamental matrix for *normalized image coordinates*. If $\mathbf{x}$ is the image of a world point $\mathbf{X}$ projected by $\mathsf{P} = \mathsf{K}\,[\mathsf{R}|\mathbf{t}]$, then $\widehat{\mathbf{x}} = \mathsf{K}^{-1}\mathbf{x}$ represents the image point in normalized coordinates. More specifically, $\widehat{\mathbf{x}}$ is the image of $\mathbf{X}$ taken by the camera $\widehat{\mathsf{P}} = [\mathsf{R}|\mathbf{t}]$, which has the identity matrix as its calibration matrix.

Corresponding to Equation (2.14), the defining equation for the essential matrix is

$$\widehat{\mathbf{x}}'^{\top}\mathsf{E}\widehat{\mathbf{x}} = 0. \tag{2.23}$$

Substituting for $\widehat{\mathbf{x}}$ and $\widehat{\mathbf{x}}'$ yields

$$\mathbf{x}'^{\top}\mathsf{K}'^{-\top}\mathsf{E}\mathsf{K}^{-1}\mathbf{x} = 0, \tag{2.24}$$

from which follows that the relationship between the essential and the fundamental matrix is

$$\mathsf{E} = \mathsf{K}'^{\top}\mathsf{F}\mathsf{K}. \tag{2.25}$$

Chosing the first camera's coordinate frame as the world reference frame, according to Equation (2.22), the essential matrix has the form

$$\mathsf{E} = [\mathbf{t}]_{\times}\mathsf{R} \tag{2.26}$$

for a pair of normalized cameras $\widehat{\mathsf{P}} = [\mathsf{I}|\mathbf{0}]$ and $\widehat{\mathsf{P}}' = [\mathsf{R}|\mathbf{t}]$.

In contrast to the fundamental matrix, the essential matrix has only five degrees of freedom: three for the rotation and two for the direction of the translation (since there is an overall scale ambiguity, translation distance is insignificant). The reduced number of degrees of freedom translates into an additional constraint.

Result 3 *A* 3×3 *matrix is an essential matrix if and only if two of its singular values are equal and the third is zero.*

For a proof, see [2, *p. 239*].

Historically, the essential matrix was introduced prior to the fundamental matrix [5]. Thus, the fundamental matrix may be seen as a generalization of the essential matrix.

Chapter 3

Extracting Cameras from the Essential Matrix

Having introduced the concepts of epipolar geometry and the essential matrix in Chapter 2, the focus is put back to the original task now.

While the robot is moving around, its camera provides different views of the world it operates in. Given two such views, the essential matrix between them can be computed. Of course, this requires to identify point correspondences between the views, which is out of the scope of this paper. Methods to compute an essential matrix from a set of point correspondences are deferred to Chapter 4.

Having estimated an essential matrix, it is possible to recover the relative position and orientation of the cameras, or equivalently position and orientation of the robot. Using Equation (2.26), the second camera's position and orientation in the coordinate frame of the first camera can be recovered. There is an overall scale ambiguity, as illustrated in Figure 3.1, so that only the *direction* of the baseline can be reconstructed, but not the exact position of the second camera.

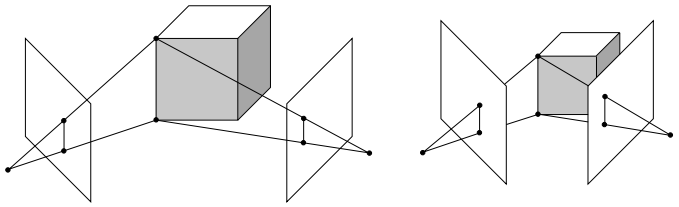

Figure 3.1: *The same images arise from two cameras standing far apart, looking at a big object and the same cameras standing close together, looking at a small object.*

Unfortunately, this is not the only ambiguity, but there are four differ-

ent solutions of baseline and orientation corresponding to a given essential matrix. Section 3.2 shows that this ambiguity can be resolved using the original point correspondences. Sections 3.1 and 3.3 introduce two different methods to factor the essential matrix according to Equation (2.26).

However, once these problems are solved, by combining several consecutive position-orientation pairs, it should then be possible to keep track of the robots position (relative to its initial pose). The issue of combining such pairs to a path will be dealt with in Chapters 5 and 6.

3.1 Factoring E using singular value decomposition

Obviously, to obtain a solution, it is necessary to factor E into the product of a skew-symmetric matrix S and a rotation matrix R. Longuet-Higgins suggested to do this using Singular Value Decomposition [5]. This approach has been adopted in [2, 10], for instance. It is implemented in the Gandalf library, too [7].

The following result shows how S and R can be obtained. It is taken from [2].

Result 4 *Suppose that the SVD of* E *is* $\mathsf{U}\,\mathrm{diag}(1,1,0)\mathsf{V}^\top$. *There are (ignoring signs) two possible factorizations* $\mathsf{E} = \mathsf{SR}$ *as follows:*

$$\mathsf{S} = \mathsf{UZU}^\top \qquad \mathsf{R} = \mathsf{UWV}^\top \quad \textit{or} \quad \mathsf{UW}^\top\mathsf{V}^\top \tag{3.1}$$

with

$$\mathsf{Z} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \textit{and} \quad \mathsf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.2}$$

That the given factorization is valid is true by inspection. For a proof that there are no other factorizations see [2, *p.240*]. Note, the scale of E can be chosen arbitrarily and in this case, is chosen such that the singular values become $(1, 1, 0)$.

Taking $[\mathbf{t}]_\times = \pm\mathsf{S}$ and combining this with the two solutions for R, there are (up to scale) four distinct solutions. The situation is illustrated in Figure 3.2. The relationship between solutions (a) and (b), respective (c) and (d), is a reversal of the sign of the baseline. Solutions (a) and (c), respective (b) and (d), are related by a 180° rotation of the second camera about the baseline.

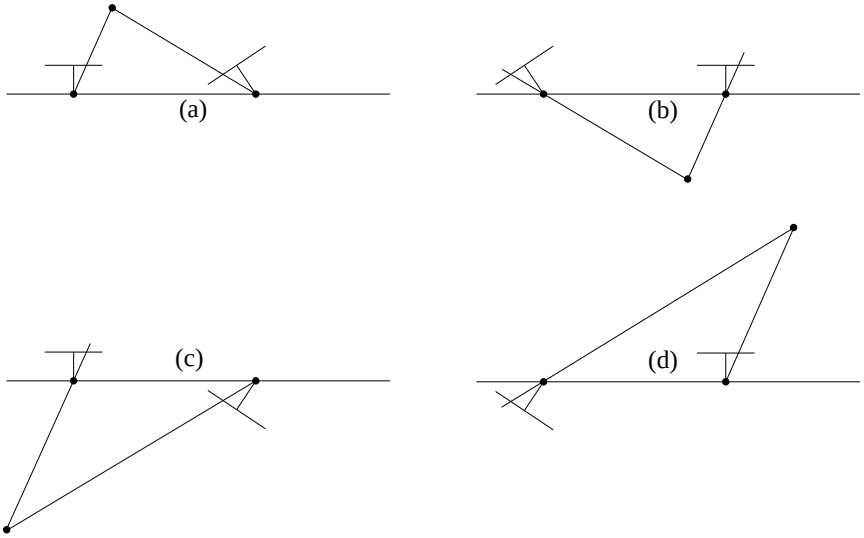

Figure 3.2: *There are four possible solutions for reconstruction from* E*. A reconstructed point is in front of both cameras in only one solution.*

3.2 Choosing the correct solution from four possible

To determine which of the four solutions is the correct one, it is necessary to go back beyond the essential matrix to the point correspondences. For each solution, it is possible to reconstruct the position of an original world point by backprojecting a corresponding pair of image points.[1] As illustrated in Figure 3.2, the reconstructed world point will lie in front of both cameras in one of the solutions only.

Theoretically, it is sufficient to test the solutions by reconstructing a single world point in this way. In the presence of measurement error though, there is a good chance that the reconstruction of a some point will be distorted so much that the wrong solution is chosen. Therefore, it is better to test with *all* the point correspondences and choose the solution that gets maximum support.

Implementation. The implementation of a function `solutionsupport` is now examined, which determines the number of point correspondences that support a particular solution.

```
int solutionsupport(Gan_Matrix34* m34P,
```

[1]Reconstruction of world points from image correspondences will be needed in Section 5.2, too, and will be explained there in detail.

```
    Gan_Vector3* av3Point0, Gan_Vector3* av3Point1,
    int count) {
...
```

It takes as parameters the second camera projection matrix `m34P` (The matrix $[\mathsf{I}|\mathbf{0}]$ is assumed to be the first camera) and a number of `count` normalized point correspondences, where the array `av3Point0` contains the points from the first image and `av3Point1` the corresponding points from the second image.

For each correspondence, the following to steps are performed:

- Reconstruct the world point $\mathbf{X}$, that projects to the given image points in both views.
- Determine, if this point is in front of both cameras. If so, increment `supportcount`.

To reconstruct $\mathbf{X}$, the function `reconstructWorldPoint()` is used which will be defined in Section 5.2.

```
Gan_Vector4 v4X;
if(!reconstructWorldPoint(m34P, &av3Point0[i],
                          &av3Point1[i], &v4X))
{
   ... //reconstruction failed, process next correspondence
}
```

To determine wether a point is in front af a camera it is first transformed into the coordinate frame of this camera. Then it is checked whether its Euclidean z-coordinate is positive.

If $\mathbf{X}$ is scaled such that its last component is positive, it is straightforward to determine wether it is in front of a camera: This is the case if the last component of $\mathbf{x} = \mathsf{P}\mathbf{X}$ is positive. For the first camera $[\mathsf{I}|\mathbf{0}]$, the last component of $\mathbf{x} = [\mathsf{I}|\mathbf{0}]\,\mathbf{X}$ is simply the third component of $\mathbf{X}$.

```
if(v4X.w < 0) {
   (void)gan_vec4_negate_i(&v4X);
}
Gan_Vector3 v3x;
(void)gan_mat34_multv4_q(m34P, &v4X, &v3x);

if((v3x.z > 0) && (v4X.z < 0)) {
   supportcount++; // count this correspondence
   ...
```

The steps above are repeated for every point correspondence, and the resulting `supportcount` is returned. The function can be called for each of the four possible P' solutions, and the correct (most likely, still subject to a measurement error) solution is the one which gets the maximum result.

3.3 Factoring E using Horn's method

An alternate method for recovering baseline and orientation from E is given by Horn in [3]. For details, the reader is referred to the original paper. The two possible solutions for the baseline $\mathbf{t}$ are computed as

$$\mathbf{t} = \pm \frac{\mathbf{e}_i \times \mathbf{e}_j}{\|\mathbf{e}_i \times \mathbf{e}_j\|} \sqrt{\frac{1}{2} \operatorname{trace}(\mathsf{E}\mathsf{E}^\top)}, \tag{3.3}$$

where $(\mathbf{e}_i \times \mathbf{e}_j)$ is the largest of the three possible cross products of any two of the columns of E. Having obtained the solutions $\mathbf{t}_+$ and $\mathbf{t}_-$ for the baseline, the corresponding solutions for orientation R_+ and R_- can be computed from

$$\|\mathbf{t}\|^2 \mathsf{R} = \operatorname{cofactors}(\mathsf{E})^\top - [\mathbf{t}]_\times \mathsf{E}. \tag{3.4}$$

In the case that the essential matrix estimate does not obey the constraints of Result 3, the solutions for R will not be real rotation matrices and should be corrected to be orthogonal.

Combining the possible baselines and orientations yields the four solutions for the second camera projection matrix.[2]

Implementation. The following function is a straightforward implementation of Horn's method. It computes the four solutions for the second camera from an essential matrix `m33E` and stores them in the array `m34P`.

```
void getRtSolutions(Gan_Matrix33* m33E, Gan_Matrix34* m34P) {
    static Gan_Matrix33 m33Orientation[2];
    static Gan_Vector3 v3Baseline[2];
    static Gan_Vector3 v3BaselineUnitNorm[2];
```

As pointed out in [3] trace($\mathsf{E}\mathsf{E}^\top$) simply is the sum of the squares of the elements of E.

```
    double TraceEET = m33E->xx * m33E->xx
       + m33E->xy * m33E->xy + m33E->xz * m33E->xz
       + m33E->yx * m33E->yx + m33E->yy * m33E->yy
       + m33E->yz * m33E->yz + m33E->zx * m33E->zx
       + m33E->zy * m33E->zy + m33E->zz * m33E->zz;
```

Now the two solutions for the baseline can be obtained using the largest cross product $(\mathbf{e}_i \times \mathbf{e}_j)$.

```
    Gan_Vector3 ecol[3];
    (void)gan_mat33_get_cols_q(m33E,
                               &ecol[0], &ecol[1], &ecol[2]);
```

[2] The combinations $[\mathsf{R}_-|\mathbf{t}_+]$ and $[\mathsf{R}_-|\mathbf{t}_+]$ correspond to the reversal of the sign of E.

```
Gan_Vector3 blsol[3];
(void)gan_vec3_cross_q(&ecol[0], &ecol[1], &blsol[0]);
(void)gan_vec3_cross_q(&ecol[1], &ecol[2], &blsol[1]);
(void)gan_vec3_cross_q(&ecol[2], &ecol[0], &blsol[2]);

double sqrlen[3];
sqrlen[0] = gan_vec3_sqrlen_q(&blsol[0]);
sqrlen[1] = gan_vec3_sqrlen_q(&blsol[1]);
sqrlen[2] = gan_vec3_sqrlen_q(&blsol[2]);
if(sqrlen[0] > sqrlen[1] && sqrlen[0] > sqrlen[2]) {
   v3Baseline[0] = blsol[0];
} else if(sqrlen[1] > sqrlen[2]) {
   v3Baseline[0] = blsol[1];
} else {
   v3Baseline[0] = blsol[2];
}

(void)gan_vec3_unit_q(&v3Baseline[0],
                      &v3BaselineUnitNorm[0]);
(void)gan_vec3_scale_q(&v3BaselineUnitNorm[0],
                       sqrt(TraceEET / 2.0),
                       &v3Baseline[0]);
(void)gan_vec3_negate_q(&v3BaselineUnitNorm[0],
                        &v3BaselineUnitNorm[1]);
(void)gan_vec3_negate_q(&v3Baseline[0], &v3Baseline[1]);
```

The two values for rotation, that correspond to the baseline solution can be obtained now. The matrix of cofactors of E is just $\text{cofactors}(\mathsf{E}) = (\mathbf{e}_2 \times \mathbf{e}_3, \mathbf{e}_3 \times \mathbf{e}_1, \mathbf{e}_1 \times \mathbf{e}_2)$.

```
Gan_Matrix33 CofactorsT;
(void)gan_mat33_set_cols_q(&CofactorsT,
              &blsol[1], &blsol[2], &blsol[0]);

Gan_Matrix33 m33B;
Gan_Matrix33 m33BE;

//for first baseline
(void)gan_mat33_cross_q(&v3Baseline[0], &m33B);
(void)gan_mat33_rmultm33_q(&m33B, m33E, &m33BE);
(void)gan_mat33_sub_q(&CofactorsT, &m33BE,
                      &m33Orientation[0]);
(void)gan_mat33_divide_i(&m33Orientation[0], TraceEET/2);
(void)gan_rot3D_matrix_adjust(&m33Orientation[0]);
```

```
    //for second baseline
    ...
```

Finally, rotations and baselines are combined to the four solutions

```
    (void)gan_mat34_set_parts_q(&m34P[0], &m33Orientation[0],
                                &v3BaselineUnitNorm[0]);
    (void)gan_mat34_set_parts_q(&m34P[1], &m33Orientation[0],
                                &v3BaselineUnitNorm[1]);
    (void)gan_mat34_set_parts_q(&m34P[2], &m33Orientation[1],
                                &v3BaselineUnitNorm[0]);
    (void)gan_mat34_set_parts_q(&m34P[3], &m33Orientation[1],
                                &v3BaselineUnitNorm[1]);
}
```

Which of these solutions is the correct one, can be determined using the function `solutionsupport()` described in the Section 3.2.

Chapter 4

Estimation of the Essential Matrix

This chapter deals with the computation of the essential matrix from point correspondences. Efficient linear methods are presented, all based on the concept of total least-squares (TLS). It should be noted, that the algorithms can be employed to compute the fundamental matrix, too. To compute the fundamental matrix, one uses the plain point correspondences, whereas to compute the essential matrix, the point correspondences are transformed to normalized coordinates beforehand.

Finally, the performance of the algorithms will be evaluated. It will be seen, that really good results can be obtained, if proper normalization is carried out.

In [2] several nonlinear iterative algorithms are presented, which may yield even better result. These are not discussed here though.

4.1 The 8-point algorithm

From Section 2.5 it is known, that the essential matrix must fulfill

$$\widehat{\mathbf{x}}'^{\top}\mathsf{E}\widehat{\mathbf{x}} = 0. \tag{4.1}$$

for any pair of matching points $\mathbf{x} \leftrightarrow \mathbf{x}'$ in two views. Writing $\widehat{\mathbf{x}} = (x, y, 1)$ and $\widehat{\mathbf{x}}' = (x', y', 1)$, a corresponding pair gives rise to one linear equation in the elements of E. Writing out Equation (4.1) yields

$$x'xe_{11} + x'ye_{12} + x'e_{13} + y'xe_{21} + y'ye_{22} + y'e_{23} + xe_{31} + ye_{32} + e_{33} = 0, \tag{4.2}$$

where e_{ij} denotes the element of E in the i-th row and j-th column. Expressing the e_{ij} as a 9-vector $\mathbf{e} = (e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}, e_{31}, e_{32}, e_{33})^{\top}$ Equation (4.2) can be written as the inner product

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1)\mathbf{e} = \mathbf{0}. \tag{4.3}$$

The coefficient vectors obtained from a set of n pairs $\mathbf{x}_n \leftrightarrow \mathbf{x}'_n$ can be stacked to form a system of linear equations

$$\mathsf{A}\mathbf{e} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{e} = 0 \qquad (4.4)$$

Obviously, from 8 point correspondences in general configuration[1] the solution vector $\mathbf{e}$ can be determined uniquely (up to scale).

If more than 8 correspondences are given and the data is not exact (because of measurement error), the rank of A may be 9 and consequently, there will be no solution apart from the trivial one $\mathbf{e} = \mathbf{0}$.

In this case a least-squares solution can be found that minimizes $\|\mathsf{A}\mathbf{e}\|$ subject to $\|\mathbf{e}\| = 1$. Since $\mathbf{e}$ is only determined up to scale, it's norm may be chosen arbitrarily. This solution is the singular vector of A corresponding to the smallest singular value, or equivalently the eigenvector of $\mathsf{A}^\top\mathsf{A}$ with the least eigenvalue.

The 8-point algorithm was introduced by Longuet-Higgins in [5].

Implementation. The Gandalf Library implements the 8-point algorithm in the functions

```
Gan_Bool gan_fundamental_matrix_fit(...)
Gan_Bool gan_essential_matrix_fit(...)
```

which, as their names suggest, compute the fundamental respective essential matrix from point correspondences.

The function `gan_fundamental_matrix_fit()` just implements the algorithm explained above. The function `gan_essential_matrix_fit()` additionally performs transformation of the point correspondences to normalized coordinates. It also reconstructs the second camera from the computed essential matrix, but it does so using SVD, so it is not recommended to use this result.

A detailed description of the methods is given in the documentation for the Gandalf Library [7, *Tutorial, Section 5.2*].

4.2 The normalized 8-point algorithm

The performance of the 8-point algorithm can be vastly improved by normalization. *Normalization*, in this context, means to apply a transformation to the input data and later remove it's effect from the computed essential respective fundamental matrix estimate. It should not be confused with normalized image coordinates, where the desired effect is to remove the influence

[1] The rows of A must be linearly independent for a unique solution to exist.

of the camera calibration matrix from the input data. In the following, the former kind of normalization is meant if the contrary is not explicitly stated.

This section proceeds with a discussion of the problems that arise from the (unnormalized) 8-point algorithm and adds a normalization step that improves the algorithm. A comparison of the results of both algorithms will be given in Section 4.6.

The 8-Point algorithm minimizes the norm $\|\mathsf{Ae}\|$. More explicitly, the norm of the residual vector $\boldsymbol{\epsilon} = \mathsf{Ae}$ is minimized. Each correspondence $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i$ contributes one element ϵ_i of the residual vector. That quantity is known as *algebraic distance*, which is defined as

$$\mathrm{d}_{\mathrm{alg}}(\mathbf{x}_1, \mathbf{x}_2)^2 = \mathbf{x}_1 \times \mathbf{x}_2. \tag{4.5}$$

For a set of correspondences, the residual vector $\boldsymbol{\epsilon}$ is the *algebraic error vector* for the whole set. One can see, that

$$\sum_i \mathrm{d}_{\mathrm{alg}}(\widehat{\mathbf{x}}'_i, \mathsf{E}\widehat{\mathbf{x}}_i)^2 = \sum_i \|\epsilon_i\|^2 = \|\boldsymbol{\epsilon}\|^2 \tag{4.6}$$

The disadvantage of the concept of algebraic distance is that the quantity that is minimized is not geometrically or statistically meaningful [2]. As a consequence, the 8-point algorithm is not invariant under a projective transformation of the input data.

Suppose, the algorithm yields E as a solution of the problem

$$\widehat{\mathbf{x}}_i'^{\top} \mathsf{E} \widehat{\mathbf{x}}_i = 0. \tag{4.7}$$

For some (invertible) projective transformations T and T', let $\widetilde{\mathbf{x}}_i = \mathsf{T}\widehat{\mathbf{x}}_i$ and $\widetilde{\mathbf{x}}'_i = \mathsf{T}'\widehat{\mathbf{x}}'_i$. Since

$$\widehat{\mathbf{x}}_i'^{\top} \mathsf{T}'^{\top} \mathsf{T}'^{-\top} \mathsf{E} \mathsf{T}^{-1} \mathsf{T} \widehat{\mathbf{x}}_i = \widehat{\mathbf{x}}_i'^{\top} \mathsf{E} \widehat{\mathbf{x}}_i = 0 \tag{4.8}$$

one would expect that the solution $\widetilde{\mathsf{E}}$ of the problem

$$\widetilde{\mathbf{x}}_i'^{\top} \widetilde{\mathsf{E}} \widetilde{\mathbf{x}}_i = 0 \tag{4.9}$$

is related to E by $\mathsf{E} = \mathsf{T}'^{\top} \widetilde{\mathsf{E}} \mathsf{T}$. However, for the 8-point algorithm this is unfortunately not true. Hartley and Zisserman [2, *p. 89*] explain this behaviour for the analogue problem (and algorithm) of estimating a 2D homography from point correspondences.

Since the property $\mathsf{E} = \mathsf{T}'^{\top} \widetilde{\mathsf{E}} \mathsf{T}$ does not hold, it is obvious that there are "good" and "bad" transformations T, T' with respect to the optimal solution. The concept of the normalized 8-point algorithm can be summarized as follows:

1. Apply "good" transformations to the input data to obtain $\widetilde{\mathbf{x}}_i = \mathsf{T}\widehat{\mathbf{x}}_i$ and $\widetilde{\mathbf{x}}'_i = \mathsf{T}'\widehat{\mathbf{x}}'_i$.
2. Compute $\widetilde{\mathsf{E}}$ as the least-squares solution to $\widetilde{\mathbf{x}}_i'^{\top}\widetilde{\mathsf{E}}\widetilde{\mathbf{x}}_i = 0$
3. Obtain the essential matrix as $\mathsf{E} = \mathsf{T}'^{\top}\widetilde{\mathsf{E}}\mathsf{T}$

Hartley and Zisserman [2] suggest to choose the normalization transformations as *"a translation and isotropic scaling of each image so that the centroid of the reference points is at the origin of the coordinates and the RMS distance of the points from the origin is equal to* $\sqrt{2}$ *"*. This approach has been implemented and tested here.

Muehlich and Mester [8] derived an optimal normalization transformation and applied TLS with fixed columns to exploit contraints in the error structure of the resulting coefficient matrix. The normalization they suggested was implemented, too, but applying plain TLS it did not show significant differences to the "Hartley"-normalization. It is therefore omitted from the experimental results section.

Implementation. A function that computes the normalization transformation suggested by Hartley and Zisserman is now examined:

```
void getHartleyNormT(Gan_SquMatrix33* ltm33T,
                     Gan_Vector3* av3Point, int count) {
```

It takes three parameters: the matrix `ltm33T` to receive the normalization transformation, an array of points `av3Point`, and the number of points in the array `count`. The transformation computed will be the upper-triangular matrix

$$\mathsf{T} = \begin{bmatrix} s & & t_x \\ & s & t_y \\ & & 1 \end{bmatrix}, \tag{4.10}$$

with s being the isotropic scaling factor and (t_x, t_y) the translation vector. Since the Gandalf Library only supports lower-triangular matrices, the transpose of T will be returned. Gandalf provides implicit transposition for matrix multiplication, this must be applied when using `ltm33T`.

The translation vector (t_x, t_y) shall move the centroid of the points to the origin. We first compute the vector (t_{x0}, t_{y0}) which moves the centroid

of the original (unscaled) pointset to the origin

$$t_{x0} = st_x = -\frac{\sum_{i=0}^{n} x_i}{n}$$
$$t_{y0} = st_y = -\frac{\sum_{i=0}^{n} y_i}{n}$$

where x_i and y_i are the Euclidean coordinates of the i-th point.

The factor s shall scale the points such that RMS distance of the points from the origin is equal to $\sqrt{2}$. This is computed as

$$s = \frac{n\sqrt{2}}{\sum_{i=0}^{n} \sqrt{(x_i + t_{x0})^2 + (y_i + t_{y0})^2}}$$

where x_i and y_i are the Euclidean coordinates of the i-th point.

```
double sum_xi = 0, sum_yi = 0;
double sum_squ = 0;
double n = (double)count;
for(int i=0; i<count; i++) {
   double xi = av3Point[i].x / av3Point[i].z;
   double yi = av3Point[i].y / av3Point[i].z;
   sum_xi += xi; sum_yi += yi;
}
double tx0 = - sum_xi / n;
double ty0 = - sum_yi / n;
for(int i=0; i<count; i++) {
   double xi = av3Point[i].x / av3Point[i].z + tx0;
   double yi = av3Point[i].y / av3Point[i].z + ty0;
   sum_squ += sqrt( xi*xi + yi*yi );
}

double s = sqrt(2) * n / sum_squ;
double tx = s * tx0;
double ty = s * ty0;
(void)gan_ltmat33_fill_q(ltm33T, s,
                                 0,  s,
                                 tx, ty, 1);
```

Note, normalization transformations should be computed from and applied to coordinates already (camera) normalized.

4.3 The 4-point algorithm

For a robot navigating in a typical office environment, some additional constraints arise: The robot is moving on a plane and rotation occurs about

the normal of this plane only. In the simplest situation the principal axis of the camera lies in a plane parallel to the x-y-plane and movement is parallel to this plane. In this case, the components of camera displacement are restricted as follows: The component y of the translation is zero,

$$\mathbf{t} = \begin{pmatrix} x \\ 0 \\ z \end{pmatrix}. \tag{4.11}$$

And the camera rotates by θ about the y-axis,

$$\mathsf{R} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}. \tag{4.12}$$

Building the resulting essential matrix, one can see that five of its elements are equal to zero.

$$\mathsf{E} = [\mathbf{t}]_\times \mathsf{R} = \begin{bmatrix} 0 & -z & 0 \\ \cos(\theta)z + \sin(\theta)x & 0 & \sin(\theta)z - \cos(\theta)x \\ 0 & x & 0 \end{bmatrix} \tag{4.13}$$

Obviously, this is very convenient since the coefficients of the zero elements drop out of Equation (4.2).

$$x'ye_{12} + y'xe_{21} + y'e_{23} + ye_{32} = 0, \tag{4.14}$$

Thus, the coefficient matrix A only has 4 columns and the solution vector $\mathbf{e} = (e_{12}, e_{21}, e_{23}, e_{32})^\top$ only has 4 elements. Consequently, instead of eight, only *four* point correspondences are needed to uniquely determine E. Note, the rank=2 constraint is automatically fulfilled because of the structure of E.

For the extraction of R and $\mathbf{t}$, the number of possible solutions drops to two. Since rotation about the baseline is impossible, R is determined uniquely.

4.4 The normalized 4-point algorithm

Despite the improvements over the 8-point algorithm, the 4-point algorithm is still associated with the same problems. To overcome these, again input data normalization is employed.

The normalization, as done in Section 4.2 (translation and isotropic scaling) results in normalization matrices of the form

$$\mathsf{T} = \begin{bmatrix} s & & t_x \\ & s & t_y \\ & & 1 \end{bmatrix}, \tag{4.15}$$

with s being the isotropic scaling factor and (t_x, t_y) the translation vector.

From the relation $\mathsf{E} = \mathsf{T}'^{\top}\widetilde{\mathsf{E}}\mathsf{T}$, it is obvious that non-zero values of t_y respective t'_y result in a "destruction" of three of the zero elements in $\widetilde{\mathsf{E}}$. This is not desirable, and therefore t_y and t'_y should be set to zero.

The following normalization is suggested: isotropic scaling so that the RMS distance of the points from the origin is equal to $\sqrt{2}$; translation in x-direction so that the average point has $x = 0$.

This results in normalization matrices of the form

$$\mathsf{T} = \begin{bmatrix} s & & t_x \\ & s & \\ & & 1 \end{bmatrix}, \tag{4.16}$$

In Section 4.6 the results of the 4-point algorithms are compared with the corrseponding results of the 8-point algorithms. The question of how much the performance of the 4-point algorithm degrades if the constraints are not perfectly met, is looked into there, too.

4.5 Generalization of the 4-point algorithms

In Section 4.3, the additional assumption was made that the principal axis is parallel to the x-z-plane. This is not necessary, though. Since two views taken from a fixed point are related by a 2D homography, it is sufficient that rotations of the camera about the x-axis and z-axis are known.

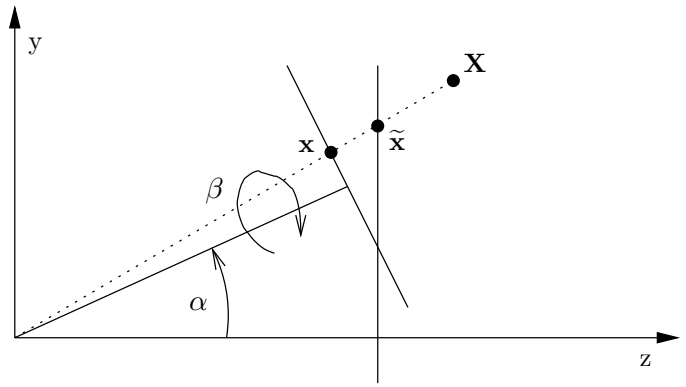

Figure 4.1: *Ideal and real image points are related by a 2D homography.*

In the situation illustrated in Figure 4.1, image points in the real image $\mathbf{x}$ (taken by the actual camera) and the ideal image $\widetilde{\mathbf{x}}$ (taken by the ideal camera, aligned with the x-z-plane) are related by

$$\widetilde{\mathbf{x}} = \mathsf{H}\mathbf{x} = \mathsf{R}_z(\beta)\mathsf{R}_x(\alpha)\mathbf{x}. \tag{4.17}$$

This is incorporated into the algorithm in the same way as image normalization:

1. Compute ideal image coordinates $\widetilde{\mathbf{x}} = \mathsf{H}\mathbf{x}$ and $\widetilde{\mathbf{x}}' = \mathsf{H}'\mathbf{x}'$.
2. Use one of the 4-point algorithms to compute an estimate for $\widetilde{\mathsf{E}}$.
3. Obtain the essential matrix as $\mathsf{E} = \mathsf{H}'^{\top}\widetilde{\mathsf{E}}\mathsf{H}$

Note, the relative orientation of the ideal cameras can be computed from $\widetilde{\mathsf{E}}$, if desired. In robot navigation this corresponds to the orientation of the robot instead of the camera.

Of course, the need to know α and β results in an additional calibration effort. The same is true for aligning the camera parallel to the x-z-plane.

4.6 Experimental results

The algorithms discussed in the previous chapters have been implemented and a number of simulations have been performed for different settings of camera motion and measurement error.

The following setup was used which is similar to the setup used in [8]: An "object" is created by choosing 25 random points inside of a cube given by world coordinates $x, y, z \in [-0.5, 0.5]$. The object is projected onto the image plane of a camera located at $(0, 0, -2)$ and looking towards the origin. The camera parameters were:

resolution	800×600 pixel
focal length	$0,6$ units $= 1000$ pixel
principal point	$x = 400$, $y = 300$ pixel

The camera was then moved to obtain a second image. The essential matrix was computed from the generated point correspondences. The position of the second camera then was reconstructed using Horn's method. Since reconstruction is only up to scale, the angular difference between the vector to the reconstructed camera and the real motion vector was taken as a measure of error.

Simulated measurement error has been introduced to the image points of the second image by adding noise vectors (x, y) drawn from a uniform distribution $x, y \in [-\epsilon/2, \epsilon/2]$ with ϵ variying between 0 and 10 pixel. The measurements in the first image remained untouched.

For a camera movement of one unit in z respective x direction, for each value of ϵ, 500 experiments were performed and the mean angular error was computed. Figures 4.2 and 4.3 compare the performance of the algorithms.

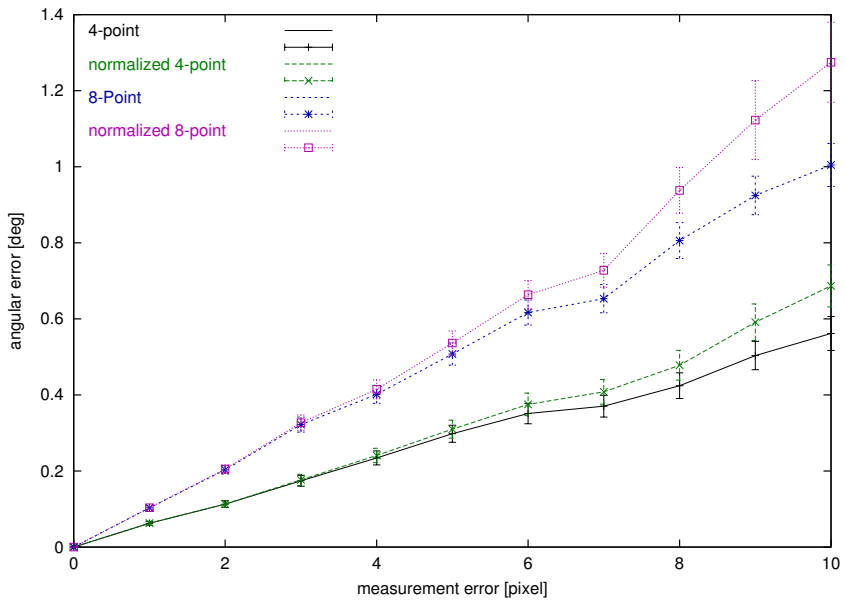

Figure 4.2: *Mean angular difference between estimated and true motion direction for motion in z direction for varying measurement error. The errorbars show the standard deviation scaled by* 0.1.

Several observations are made:

- In accordance with the results of [8], a strong bias of the standard 8-point algorithm towards the viewing direction (z) is observed. The same is true for the standard 4-point algorithm, and to some extend for the normalized versions. For the robot navigation task, this is not really a problem since movement perpendicular to the viewing direction will most likely not occur.

- The normalized versions are by far superior when movement is perpendicular to the viewing direction, where the unnormalized algorithms become intolerably inaccurate as more measurement error is introduced.

- In the case of motion in the viewing direction, the normalized algorithms slightly fall behind the unnormalized versions. It can be concluded that normalization somewhat alleviates the bias towards the viewing direction.

- The 4-point algorithms outperform their 8-point counterparts, which comes as no surprise, since more a priori knowledge is provided.

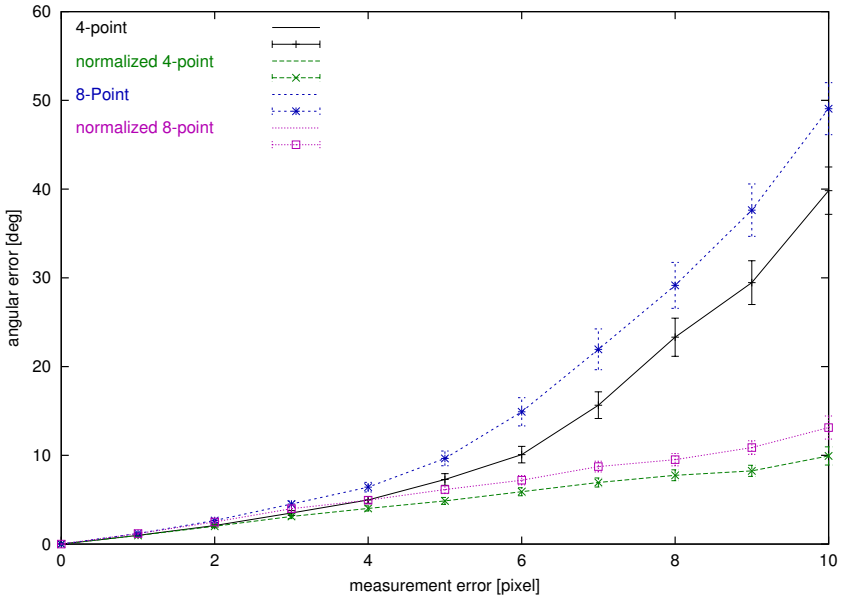

Figure 4.3: *Mean angular difference between estimated and true motion direction for motion in x direction for varying measurement error. The error-bars show the standard deviation scaled by* 0.1.

Additional experiments were performed, to determine the tolerance of the 4-point algorithms to deviation of the actual setup from the prerequisites (movement only in the x-z-plane, rotation only about the y-axis). The basic setup was left untouched, but instead of introducing error in the image measurements, the second camera was turned about the x-axis by 0 to 5 degrees (denoted as *angular error* in Figures 4.4, 4.5, 4.6). This setup is supposed to simulate a robot navigating on slightly uneven ground. For each setting 200 experiments were performed and the mean angular difference between estimated and true motion direction calculated.

The results as shown in Figures 4.4, 4.5, 4.6 are discouraging:

- For movement perpendicular to the viewing direction (Figure 4.6) the algorithm is rendered useless even by the slightest angular error.
- Even for movement towards the viewing direction, which yields the best results, the performance degradation resulting from angular error will make the 4-point algorithms lack behind the 8-point versions, unless the additional prerequisites are perfectly met.

It is probably best to use the normalized 8-point algorithm in practice. It may be considered, to fall back to one of the 4 point algorithms, in the case that there are less than eight point correspondences known.

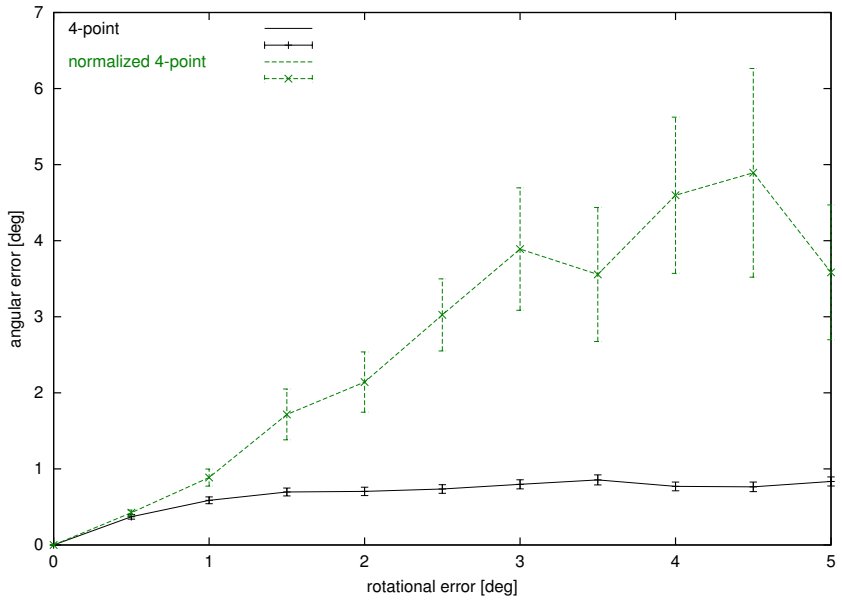

Figure 4.4: *Mean angular difference between estimated and true motion direction for motion in z direction for varying rotational error. The errorbars show the standard deviation scaled by* 0.1.

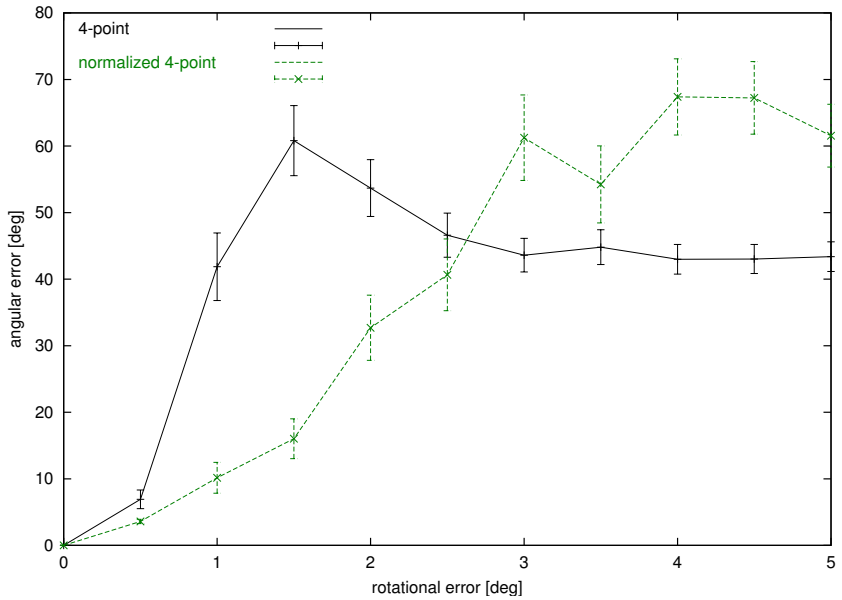

Figure 4.5: *Mean angular difference between estimated and true motion direction for motion in direction* $(1, 0, 1)$ *for varying rotational error. The errorbars show the standard deviation scaled by* 0.1.

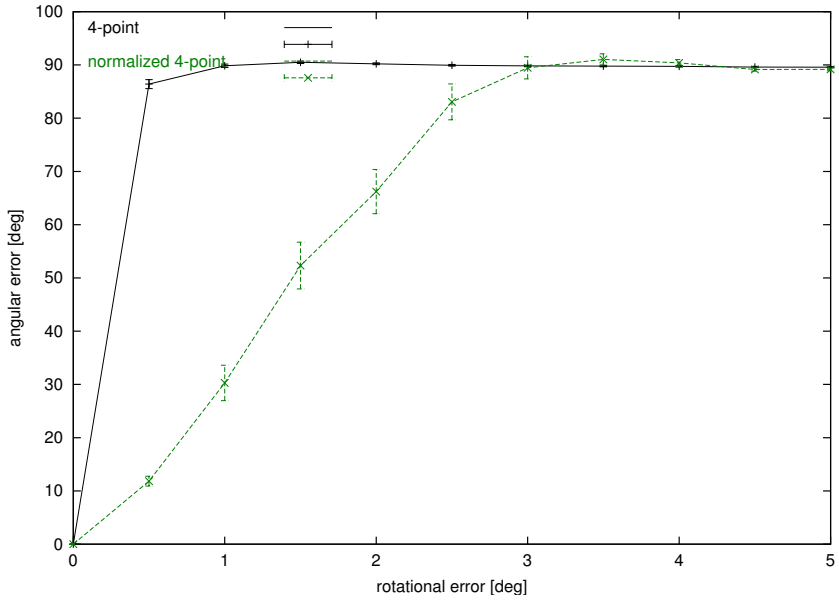

Figure 4.6: *Mean angular difference between estimated and true motion direction for motion in x direction for varying rotational error. The errorbars show the standard deviation scaled by* 0.1.

Chapter 5

Reconstructing Camera Triples

Up to this point, attention was directed to the reconstruction of camera pairs. But the final result that shall be obtained is a path i.e., a sequence of positions. This chapter describes methods to combine two consecutive position-orientation pairs to obtain relative position and orientation between three cameras. Because of the scale ambiguity of the reconstructed pairs, this is not a trivial problem. However, once the triples have been obtained, it is simple to combine them to a complete path, since two consecutive triples share one camera pair and thus the scale factor between them is obvious.

It will be seen that at least one point correspondence across all three images is needed to reconstruct a triple. At first, the problem should be solved without using such information, but it will fail for certain setups. Subsequently, an algorithm is given, that uses corresponding triples to reconstructs world points first and then calculate scale from these. Another algorithm is introduced, that does not take this detour but computes scale directly from the correspondences. Finally the performance of the latter two algorithms is compared.

The goal of all the algorithms is to compute the *scale factor* k, which is the ratio of the Euclidean distances between the camera centres in the first camera pair $(\mathsf{P}, \mathsf{P}')$ and in the second camera pair $(\mathsf{P}', \mathsf{P}'')$.

5.1 A third essential matrix

The solution to the triple reconstruction problem presented in this section is more straightforward than the other two. Unfortunately, it will not work for certain configurations. It was neither implemented nor tested.

Suppose, for three cameras P, P', P'' from the essential matrices between

P and P' respective P' and P'' the first pair was reconstructed as

$$\mathsf{P} = [\mathsf{I}|\mathbf{0}] \tag{5.1}$$
$$\mathsf{P}' = [\mathsf{R}|\mathbf{t}] \tag{5.2}$$

and the second pair as

$$\mathsf{P}' = [\mathsf{I}|\mathbf{0}] \tag{5.3}$$
$$\mathsf{P}'' = \left[\mathsf{R}'|k\mathbf{t}'\right]. \tag{5.4}$$

Since reconstruction is only up to scale and the scale may differ between the pairs, a unknown factor k has been introduced to P''. The goal of triple reconstruction is to obtain this factor k, that is the *relative scale* between the camera pairs.

Using the first cameras coordinate frame as the reference frame, the cameras become

$$\mathsf{P} = [\mathsf{I}|\mathbf{0}] \tag{5.5}$$
$$\mathsf{P}' = [\mathsf{R}|\mathbf{t}] \tag{5.6}$$
$$\mathsf{P}'' = \left[\mathsf{R}'\mathsf{R}|\mathsf{R}'\mathbf{t} + k\mathbf{t}'\right]. \tag{5.7}$$

Using Equation (2.26), the essential matrix between the first and the third view is

$$\mathsf{E}(k) = [\mathsf{R}'\mathbf{t} + k\mathbf{t}']_\times \mathsf{R}'\mathsf{R}. \tag{5.8}$$

Given a correspondence between the first and third view $\mathbf{x} \leftrightarrow \mathbf{x}''$ the factor k can be computed from

$$\widehat{\mathbf{x}}''^\top \mathsf{E}(k)\widehat{\mathbf{x}} = 0. \tag{5.9}$$

Note, that this does not require the existence of a corresponding point $\mathbf{x}'$ in the second view.

Since $\mathsf{E}(k)$ is a homogeneous entity and thus only defined up to scale, k obviously cannot be determined if $\mathsf{R}'\mathbf{t} = \mathbf{t}'$. This happens if the three camera centres are collinear.

Figure 5.1 illustrates the equivalent case when the relative position and orientation is known for the three combinations $(\mathsf{P}, \mathsf{P}')$, $(\mathsf{P}', \mathsf{P}'')$ and $(\mathsf{P}, \mathsf{P}'')$. If the camera centres are collinear, it is impossible to determine relative scale for this case, too.

It is therefore not sufficient to know $\mathsf{E}(k)$ in order to obtain a solution for relative scale. The next sections will show how k can be computed if correspondences across all three views are available.

5.2 Scale from reconstructed world points

Given a correspondence $\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}''$ and the reconstructed camera pairs, the world point $\mathbf{X}$ can be reconstructed from $\mathbf{x} \leftrightarrow \mathbf{x}'$ and the first camera

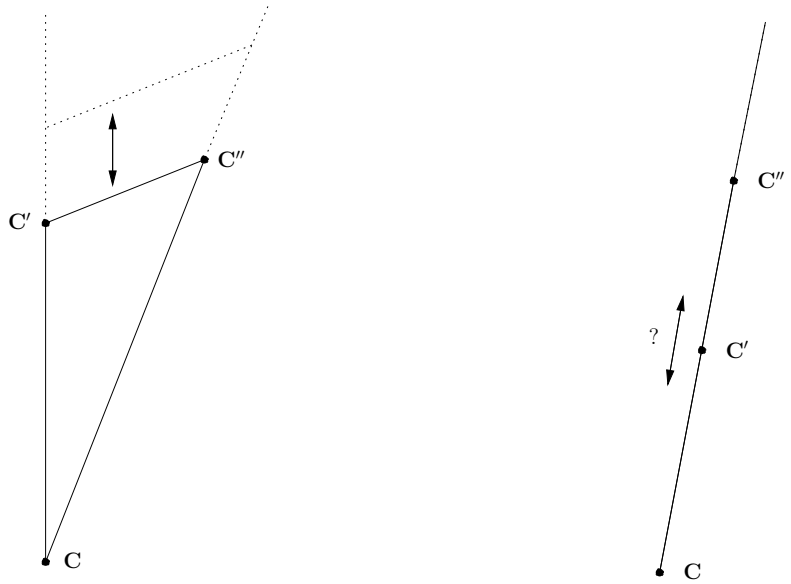

Figure 5.1: *Relative scale cannot be computed if the camera centres are collinear.*

pair or from $\mathbf{x}' \leftrightarrow \mathbf{x}''$ and the second camera pair. All the image measurements resulted from the same world point, so the reconstructed point should be the same in both cases. However, because of the undetermined scale, the reconstructed points may be different, and the scale factor k between the camera pairs can be obtained from this difference. This basic idea is illustrated in Figure 5.2 This section introduces an algorithm based on this idea. It can be summarized as follows:

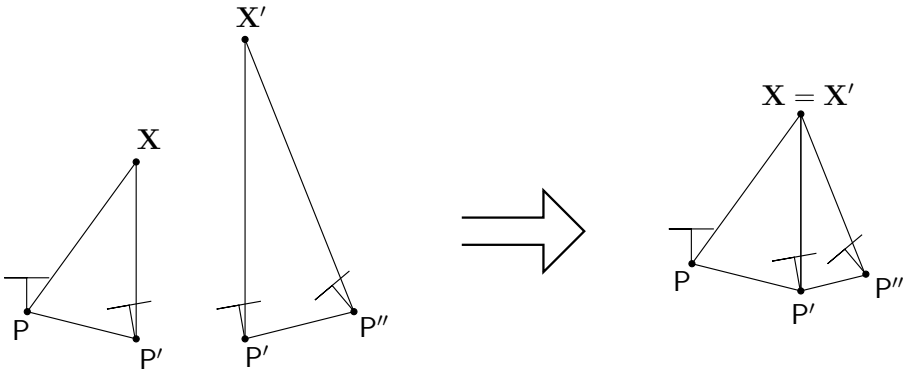

Figure 5.2: *Obtaining relative scale of a camera triple from a reconstructed world point.*

1. For each of the reconstructed pairs of cameras $(\mathsf{P}, \mathsf{P}')$, $(\mathsf{P}', \mathsf{P}'')$ choose the scale such that the distance between the camera centres equals 1.

2. Reconstruct world points $\mathbf{X}$ and $\mathbf{X}'$ by backprojecting image measurements $\mathbf{x} \leftrightarrow \mathbf{x}'$ and $\mathbf{x}' \leftrightarrow \mathbf{x}''$ using the second camera frame as the reference frame.

3. Since $\mathbf{X}$ and $\mathbf{X}'$ should represent the same world point, the scale may be computed as

$$k = \frac{\left\|\widetilde{\mathbf{C}}'' - \widetilde{\mathbf{C}}'\right\|}{\left\|\widetilde{\mathbf{C}}' - \widetilde{\mathbf{C}}\right\|} = \frac{\left\|\widetilde{\mathbf{X}}'\right\|}{\left\|\widetilde{\mathbf{X}}\right\|} \tag{5.10}$$

   where the tilde superscript indicates the use of Euclidean coordinates.

This computation process can be repeated for each corresponding triple $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i \leftrightarrow \mathbf{x}''_i$, and the scaling factor can be obtained as the mean value of the k_i.

This section proceeds with the introduction of a simple linear triangulation method for reconstructing a world point from image measurements. It then looks into the uncertainty of reconstruction, which will be incorporated into the algorithm as weighting factors for the k_i. Finally, an implementation of the method is given, computing scale as a weighted average of the k_i.

5.2.1 Linear triangulation method

Theoretically, the world point $\mathbf{X}$, which is projected to $\mathbf{x}$ and $\mathbf{x}'$ by two cameras P and P' can be computed as the intersection of the backprojected rays through $\mathbf{C}$ and $\mathbf{x}$ respective $\mathbf{C}'$ and $\mathbf{x}'$. But in the presence of measurement error in $\mathbf{x}$ and $\mathbf{x}'$, these rays will be skew in general, and consequently there will be no point $\mathbf{X}$ that satisfies $\mathbf{x} = \mathsf{P}\mathbf{X}$ and $\mathbf{x}' = \mathsf{P}'\mathbf{X}$.

However, an estimate for $\mathbf{X}$ can be computed. The method used here is similar to the 8-point algorithm. One obtains a least-squares solution to a set of equations $\mathsf{A}\mathbf{X} = \mathbf{0}$, subject to $\|\mathbf{X}\| = 1$. The following explanation of how A is formed is taken from [2].

In each image, we have a measurement $\mathbf{x} = \mathsf{P}\mathbf{X}$, $\mathbf{x}' = \mathsf{P}'\mathbf{X}$, and these equations can be combined into a form $\mathsf{A}\mathbf{X} = \mathbf{0}$, which is an equation linear in $\mathbf{X}$. First, the homogeneous scale factor is eliminated by a cross product to obtain three equations for each image point, of which two are linearly independent. For example, for the first image, $\mathbf{x} \times (\mathsf{P}\mathbf{X}) = \mathbf{0}$ and writing this out gives

$$\begin{aligned} x\left(\mathbf{p}^{3\top}\mathbf{X}\right) - \left(\mathbf{p}^{1\top}\mathbf{X}\right) &= 0 \\ y\left(\mathbf{p}^{3\top}\mathbf{X}\right) - \left(\mathbf{p}^{2\top}\mathbf{X}\right) &= 0 \\ x\left(\mathbf{p}^{2\top}\mathbf{X}\right) - y\left(\mathbf{p}^{1\top}\mathbf{X}\right) &= 0 \end{aligned}$$

where $\mathbf{p}^{i\top}$ are the rows of P. These equations are *linear* in the components of $\mathbf{X}$.

An equation of the form $\mathsf{A}\mathbf{X} = \mathbf{0}$ can be composed with

$$\mathsf{A} = \begin{bmatrix} x\mathbf{p}^{3\top} - \mathbf{p}^{1\top} \\ y\mathbf{p}^{3\top} - \mathbf{p}^{2\top} \\ x'\mathbf{p}'^{3\top} - \mathbf{p}'^{1\top} \\ y'\mathbf{p}'^{3\top} - \mathbf{p}'^{2\top} \end{bmatrix}$$

where two equations have been included from each image, giving a total of four equations in four unknowns. This is a redundant set of equations, since the solution is determined only up to scale.

The solution can be obtained as the singular vector of A corresponding to the smallest singular value.

The method presented here has two drawbacks. First, equivalently to what was described in Section 4.2, the quantity that is minimized is not geometrically or statistially meaningful. Second, for certain configurations, A might not be of full rank. In this case, $\mathsf{A}\mathbf{X} = \mathbf{0}$ has no unique solution. For instance, consider the case of pure translation in z-direction. The camera

matrices will be

$$\mathsf{P} = [\mathsf{I}|\mathbf{0}] \qquad \mathsf{P}' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}.$$

For a world point lying on the x-z-plane, $y = y' = 0$. Since $\mathbf{p}^{2\top} = \mathbf{p}'^{2\top}$, the second and fourth row of A are linearily dependent and A is not of full rank.

The following implementation takes the latter problem into account by considering results invalid, that were obtained from a configuration, where A is numerically very close to having rank < 4. If the difference between the two smallest singular values is below a certain border, the reconstruction is considered invalid.

Implementation. The implementation of a function is now examined, which estimates the position of a world point.

```
bool reconstructWorldPoint(Gan_Matrix34* m34P,
      Gan_Vector3* v3Point0, Gan_Vector3* v3Point1,
      Gan_Vector4* v4X) {
   ...
```

It takes as parameters the second camera projection matrix `m34P` (the matrix $[\mathsf{I}|\mathbf{0}]$ is assumed to be the first camera) and a point correspondence, where `v3Point0` contains a point from the first image and `v3Point1` the corresponding point from the second image, both in normalized coordinates. The return value indicates whether the reconstruction was successful. As indicated above, cases where the difference between the two smallest singular values is below a certain border are considered invalid. The border was chosen as 0.01. The reconstructed world point is stored in `v4X`.

First the $\mathbf{p}^{i\top}$ and $\mathbf{p}'^{i\top}$ vectors are initialized from the camera projection matrices.

```
   //rows of [I|0]
   Gan_Vector4 p1_0, p2_0, p3_0;
   Gan_Matrix34 m34I0;
   (void)gan_mat34_fill_q(&m34I0,
                          1, 0, 0, 0,
                          0, 1, 0, 0,
                          0, 0, 1, 0);
   gan_mat34_get_rows_q(&m34I0, &p1_0, &p2_0, &p3_0);

   //rows of m34P
   Gan_Vector4 p1_1, p2_1, p3_1;
   gan_mat34_get_rows_q(m34P, &p1_1, &p2_1, &p3_1);
```

Then the matrix A is built:

```
Gan_Matrix44 m44A;
Gan_Vector4 v4temp;
(void)gan_vec4_scale_q(&p3_0, v3Point0->x, &v4temp);
(void)gan_vec4_sub_i1(&v4temp, &p1_0);
m44A.xx = v4temp.x;
m44A.xy = v4temp.y;
m44A.xz = v4temp.z;
m44A.xw = v4temp.w;

(void)gan_vec4_scale_q(&p3_0, v3Point0->y, &v4temp);
(void)gan_vec4_sub_i1(&v4temp, &p2_0);
m44A.yx = v4temp.x;
m44A.yy = v4temp.y;
m44A.yz = v4temp.z;
m44A.yw = v4temp.w;

(void)gan_vec4_scale_q(&p3_1, v3Point1->x, &v4temp);
(void)gan_vec4_sub_i1(&v4temp, &p1_1);
m44A.zx = v4temp.x;
m44A.zy = v4temp.y;
m44A.zz = v4temp.z;
m44A.zw = v4temp.w;

(void)gan_vec4_scale_q(&p3_1, v3Point1->y, &v4temp);
(void)gan_vec4_sub_i1(&v4temp, &p2_1);
m44A.wx = v4temp.x;
m44A.wy = v4temp.y;
m44A.wz = v4temp.z;
m44A.ww = v4temp.w;
```

and the solution $\mathbf{X}$ is obtained as the last column of V from the singular value decomposition $\mathrm{svd}(\mathsf{A}) = \mathsf{UWV}^\top$.

```
Gan_Matrix44 m44U;
Gan_Vector4 v4W;
Gan_Matrix44 m44VT;
(void)gan_mat44_svd(&m44A, &m44U, &v4W, &m44VT);

//solution is the last column of V (last row of VT)
(void)gan_vec4_fill_q(v4X,
                m44VT.wx, m44VT.wy, m44VT.wz, m44VT.ww);
```

Now, the difference between the two smallest singular values is checked. If it is below 0.01, the reconstruction is considered invalid, and the function returns `false`.

```
   return !(v4W.z - v4W.w < 0.01);
}
```

Having solved the problem of reconstructing a world point from image measurements, the remaining parts of the algorithm to compute the scale of a camera triple should be implemented now.

5.2.2 Obtaining scale

Using the function `reconstructWorldPoint`, thge scale reconstruction can be implemented in a straightforward way. The following source code shows how to obtain the scale factor k_i from a given corresponding triple $\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}''$ and the cameras $\mathsf{P} = [\mathsf{I}|\mathbf{0}]$, $\mathsf{P}' = [\mathsf{R}|\mathbf{t}]$, measured in the first cameras coordinate frame, and $\mathsf{P}'' = [\mathsf{R}'|\mathbf{t}']$, measured in the second cameras coordinate frame.

Implementation. The cameras P' and P'' are given by

```
Gan_Matrix34* m34P1;
Gan_Matrix34* m34P2;
```

The cameras must be normalized such that $\|\widetilde{\mathsf{C}}'\| = \|\widetilde{\mathsf{C}}''\| = 1$. The function `getRtSolutions` for extracting cameras from the essential matrix (described in Section 3.3 on pages 19 ff.) yields projection matrices that are normalized in this way.

The points $\mathbf{x}_i$, $\mathbf{x}'_i$, $\mathbf{x}''_i$ that constitute the correspondences are stored (in normalized coordinates) in the arrays

```
Gan_Vector3* av3Point0;
Gan_Vector3* av3Point1;
Gan_Vector3* av3Point2;
```

For the `i`th correspondencing triple, the scale factor k_i is obtained as follows. First, the world point $\mathbf{X}_i$ is reconstructed for each of the camera pairs. If the reconstruction is invalid or one of the reconstructed points is at infinity, nothing can be said about k_i for this point correspondence.

```
Gan_Vector4 v4X1temp;
Gan_Vector4 v4X2;
if( !reconstructWorldPoint(m34P1, &av3Point0[i],
                           &av3Point1[i], &v4X1temp) ||
    !reconstructWorldPoint(m34P2, &av3Point1[i],
                           &av3Point2[i], &v4X2) ||
    (v4X1temp.w == 0) ||
    (v4X2.w == 0))
{
   ... //skip this point correspondence
}
```

Since the reconstructed point `v4X1temp` is in the first cameras coordinates frame, it has to be transformed to the second cameras coordinate frame, which is used as the reference frame. The transformation matrix is obtained by augmenting the second camera projection matrix `m34P1` by a fourth row $(0, 0, 0, 1)$.

```
Gan_Matrix44 m44Cam1;
Gan_Matrix33 m33R;
Gan_Vector3 v3t;
Gan_Vector3 v3zero;
(void)gan_mat34_get_m33l_q(m34P1, &m33R);
(void)gan_mat34_get_v3r_q(m34P1, &v3t);
(void)gan_vec3_zero_q(&v3zero);
(void)gan_mat44_set_parts_q(&m44Cam1, &m33R, &v3t, &v3zero, 1);

Gan_Vector4 v4X1;
(void)gan_mat44_multv4_q(&m44Cam1, &v4X1temp, &v4X1);
```

Finally, the scale factor is obtained as the quotient of the euclidean norms of the reconstructed world points.

```
double len1 = sqrt((v4X1.x/v4X1.w)*(v4X1.x/v4X1.w) +
              (v4X1.y/v4X1.w)*(v4X1.y/v4X1.w) +
              (v4X1.z/v4X1.w)*(v4X1.z/v4X1.w));
double len2 = sqrt((v4X2.x/v4X2.w)*(v4X2.x/v4X2.w) +
              (v4X2.y/v4X2.w)*(v4X2.y/v4X2.w) +
              (v4X2.z/v4X2.w)*(v4X2.z/v4X2.w));
double scale = len1 / len2;
```

The final solution for the scale factor k is obtained as the mean of k_i. Section 5.5 will show that the algorithm is reasonable accurate if measurement error is small. The next section describes how the algorithm can be enhanced by considering the uncertainty in the reconstruction of world points to determine the significance of the respective k_i.

5.3 The uncertainty of reconstruction

This section looks into the question of how measurement error in the point correspondences translates to uncertainty in the reconstruction of world points. Of course, as measurement error increases, the error in the reconstruction will also increase. Additionally, as illustrated in Figure 5.3, reconstructed world points are less precisely localized as the backprojected rays become more parallel.

While the measurement error for individual point correspondences is not known, the angle between the backprojected rays can be computed easily.

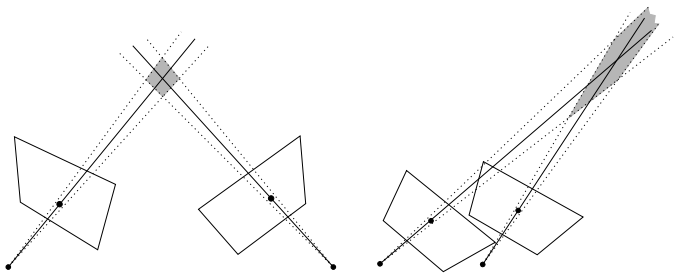

Figure 5.3: *Reconstruction uncertainty increases as backprojected rays become more parallel.*

This observation should be incorporated into the scale-reconstruction algorithm. Scale factors k_i should be considered less significant if the angles α_i respective α_i' between the backprojected rays through $\mathbf{x}_i$ and $\mathbf{x}_i'$ respective $\mathbf{x}_i'$ and $\mathbf{x}_i''$ are small.

Weighting factors $w_i(\alpha_i, \alpha_i')$ are introduced, which depend on the values of α_i and α_i'. The solution k is obtained as the weighted average of the k_i

$$k = \frac{\sum_i w_i k_i}{\sum_i k_i}. \tag{5.11}$$

The following weighting function is suggested:

$$w_i = \tan\frac{\alpha}{2} \cdot \tan\frac{\alpha'}{2} \qquad \text{with } 0 \leq \alpha, \alpha' \leq \frac{\pi}{2}. \tag{5.12}$$

It is defined intuitively and no attempt is made to determine whether it is optimal.

Implementation. The following code is an implementation of the w_i computation. The cameras P' and P'' are given by

```
Gan_Matrix34* m34P1;
Gan_Matrix34* m34P2;
```

The cameras must be normalized such that $\|\widetilde{\mathsf{C}}'\| = \|\widetilde{\mathsf{C}}''\| = 1$. The points $\mathbf{x}_i$, $\mathbf{x}_i'$, $\mathbf{x}_i''$ that constitute the correspondences are stored (in normalized coordinates) in the arrays

```
Gan_Vector3* av3Point0;
Gan_Vector3* av3Point1;
Gan_Vector3* av3Point2;
```

For the `i`th correspondencing triple, the weight w_i is obtained as follows. Before the angles between the backprojection rays can be computed, the

rays have to be transformed into a common coordinate system. For the first pair of cameras, $\mathbf{x}$ is transformed into the coordinate system of the second camera.

```
Gan_Vector3 rayrot;
Gan_Matrix33 m33R;
(void)gan_mat34_get_m33l_q(m34P1, &m33R);
(void)gan_mat33_multv3_q(&m33R, &av3Point0[i], &rayrot);
double n_x = sqrt(gan_vec3_sqrlen_q(&rayrot));
double n_y = sqrt(gan_vec3_sqrlen_q(&av3Point1[i]));
```

Now the angle between the rays is computed in the interval $[0, \pi/2]$.

```
double rayangle1 = acos( gan_vec3_dot_q(&rayrot,&av3Point1[i])
                         / (n_x * n_y));
```

Similarly, for the second pair of cameras

```
(void)gan_mat34_get_m33l_q(m34P2, &m33R);
(void)gan_mat33_multv3_q(&m33R, &av3Point1[i], &rayrot);
n_x = sqrt(gan_vec3_sqrlen_q(&rayrot));
n_y = sqrt(gan_vec3_sqrlen_q(&av3Point2[i]));
double rayangle2 = acos( gan_vec3_dot_q(&rayrot,&av3Point2[i])
                         / (n_x * n_y));
```

And finally

```
double weight = tan(rayangle1/2.0) * tan(rayangle2/2.0);
```

The Experimental results in Section 5.5 show that the use of this weight function significantly improves the results of the scale-reconstruction algorithms. The same weighting function can also be used with the alternative approach to scale reconstruction which is introduced in the next section.

5.4 Scale from backprojected rays

This chapter proposes another algorithm to reconstruct the scale between two camera pairs. Again, corresponding triples and backprojected rays from camera centres through image measurements will be used. Contrary to the previous algorithm the world points will not be reconstructed and no minimization problem has to be solved. Instead, scale will be computed from the backprojected rays directly. The "scale from reconstructed world points" algorithm will therefore also be called *indirect algorithm*. The algorithm described in this section will also be called *direct algorithm*.

Again, the second cameras coordinate frame is used as the reference frame and the camera projection matrices P and P' are normalized such that $\|\widetilde{\mathsf{C}}'\| = \|\widetilde{\mathsf{C}}''\| = 1$.

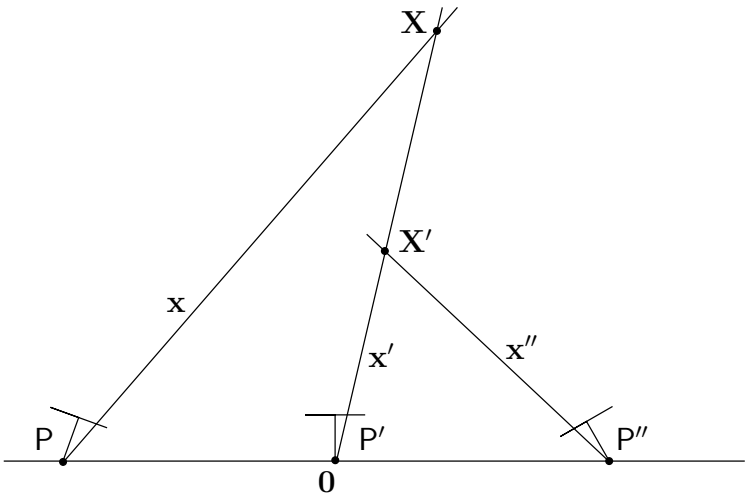

Figure 5.4: *Scale of two camera pairs, chosen such that the baseline lengths are one unit.*

Figure 5.4 shows the resulting configuration. $\mathbf{x}$, $\mathbf{x}'$ and $\mathbf{x}''$ are the backprojected rays through the respective image measurements. $\mathbf{X}$ and $\mathbf{X}'$ are the reconstructions of the world point at the intersections of $\mathbf{x}'$ with $\mathbf{x}$ and $\mathbf{x}''$. Since scale is arbitrarily fixed for each of the camera pairs, $\mathbf{X}$ and $\mathbf{X}'$ will be at different positions in general.

Now, scale is chosen differently, such that $\mathbf{X}$ and $\mathbf{X}'$ are located at the same point and $\|\widetilde{\mathbf{X}}\| = 1$ in Euclidean space. Let $\widehat{\mathsf{P}}$ and $\widehat{\mathsf{P}}''$ be the cameras P and P'' with the baselines scaled such that these conditions are satisfied. This is illustrated in Figure 5.5. Obviously, the relative scale is between the cameras is chosen correctly in this cas:

$$k = \frac{\left\|\widehat{\mathbf{C}}''\right\|}{\left\|\widehat{\mathbf{C}}\right\|}. \tag{5.13}$$

The tilde superscript denoting Euclidean representation has been left from $\widehat{\mathbf{C}}$ and $\widehat{\mathbf{C}}''$, which nevertheless denote the Euclidean coordinates of the camera centres of $\widehat{\mathsf{P}}$ and $\widehat{\mathsf{P}}''$.

Of course, up to this point this is essentially the same thing that was done in the indirect algorithm. Finding $\widehat{\mathsf{P}}$ or $\widehat{\mathsf{P}}''$ means estimating the "intersection" of two most likely skew lines. But as indicated above, the real positions of neither $\widehat{\mathsf{P}}$ and $\widehat{\mathsf{P}}''$ nor $\mathbf{X}$ shall be computed, but only k.

As illustrated in Figure 5.6, a plane $\mathbf{\Pi}$ is defined, which contains the origin (the second cameras centre) and has $\mathbf{x}$ as its normal. The backprojected rays from the camera centres $\widetilde{\mathbf{C}}$ and $\widehat{\mathbf{C}}$ intersect the plane at the points $\mathbf{p}$ and $\mathbf{q}$. Since $\widetilde{\mathbf{C}}$, $\widehat{\mathbf{C}}$ lie on the baseline and $\mathbf{p}$, $\mathbf{q}$ lie on parallel lines which

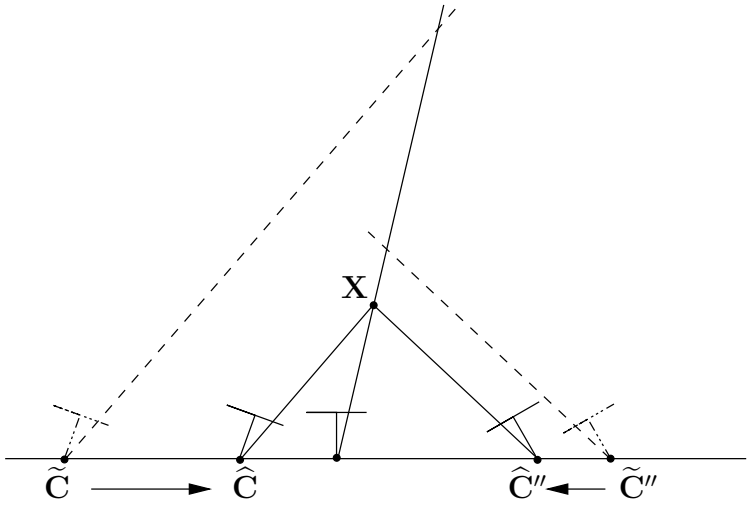

Figure 5.5: *Scale of two camera pairs, chosen such that both reconstructions of a world point are the same.*

intersect the baseline, obviously these four points are coplanar. Thus, from similar triangles, it is obtained:

$$\frac{\left\|\widetilde{\mathbf{C}}\right\|}{\left\|\widehat{\mathbf{C}}\right\|} = \frac{\|\widetilde{\mathbf{p}}\|}{\|\widetilde{\mathbf{q}}\|}. \tag{5.14}$$

The point $\mathbf{q}$ can equivalently be determined by perpendicularly projecting $\mathbf{X}$ onto $\mathbf{\Pi}$.

A plane $\mathbf{\Pi}'$ is defined similarly with $\mathbf{x}''$ as its normal. The points $\mathbf{p}'$ and $\mathbf{q}'$ are the intersection points of $\mathbf{\Pi}'$ and the backprojected rays from $\widetilde{\mathbf{C}}''$ and $\widehat{\mathbf{C}}''$.

Using Equations (5.13) and (5.14), the scale factor k is determined as

$$k = \frac{\left\|\widehat{\mathbf{C}}''\right\|}{\left\|\widehat{\mathbf{C}}\right\|} = \frac{\left\|\widetilde{\mathbf{C}}\right\| \|\widetilde{\mathbf{p}}'\| \|\widetilde{\mathbf{q}}\|}{\left\|\widetilde{\mathbf{C}}''\right\| \|\widetilde{\mathbf{p}}\| \|\widetilde{\mathbf{q}}'\|} = \frac{\|\widetilde{\mathbf{p}}'\| \|\widetilde{\mathbf{q}}\|}{\|\widetilde{\mathbf{p}}\| \|\widetilde{\mathbf{q}}'\|} \tag{5.15}$$

The resulting algorithm (for one corresponding triple $\mathbf{x}_i \leftrightarrow \mathbf{x}'_i \leftrightarrow \mathbf{x}''_i$) is now briefly summarized:

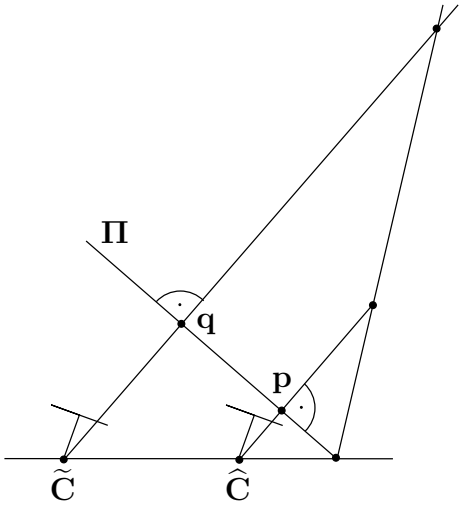

Figure 5.6: *Similar triangles using the plane* $\mathbf{\Pi}$.

1. Transform P, P'', $\mathbf{x}_i$ and $\mathbf{x}_i''$ into the coordinate system of P' and choose scale such that $\|\widetilde{\mathsf{C}}\| = \|\widetilde{\mathsf{C}}''\| = 1$.
2. Obtain the planes $\mathbf{\Pi}$ and $\mathbf{\Pi}'$, which both contain the origin and have the normal vectors $\mathbf{x}_i$ respective $\mathbf{x}_i''$.
3. Obtain $\mathbf{X}$ as a point lying on $\mathbf{x}'$ and satisfying $\|\widetilde{\mathbf{X}}\| = 1$.
4. Obtain the perpendicular projections $\mathbf{p}$ and $\mathbf{q}$ of $\mathbf{C}$ and $\mathbf{X}$ onto $\mathbf{\Pi}$. Obtain the perpendicular projections $\mathbf{p}'$ and $\mathbf{q}'$ of $\mathbf{C}''$ and $\mathbf{X}$ onto $\mathbf{\Pi}'$.
5. Compute the scale factor as

$$k_i = \frac{\|\widetilde{\mathbf{p}}'\| \, \|\widetilde{\mathbf{q}}\|}{\|\widetilde{\mathbf{p}}\| \, \|\widetilde{\mathbf{q}}'\|}. \tag{5.16}$$

The algorithm will fail for world points at infinity, where the rays backprojected through the image measurements are parallel. In this case, $\|\widetilde{\mathbf{q}}\| = \|\widetilde{\mathbf{q}}'\| = 0$ and k_i is undefined. The following implementation simply omits such cases.

Implementation. The auxiliary function `pointOnPlane` is needed:

```
void pointOnPlane(Gan_Vector3* p,
                  Gan_Vector3* n,
```

```
                    Gan_Vector3* result) {
   Gan_Vector3 t;
   (void)gan_vec3_scale_q(n,-gan_vec3_dot_q(n,p),&t);
   (void)gan_vec3_add_q(p,&t,result);
}
```

This function computes the closest point to `p` on the plane containing the origin and having the normal vector `n`. The result is stored in the third parameter `result`. All parameters are Euclidean 3-vectors, `p` and `n` must have unit norm.

The implementation of the algorithm exactly follows the five steps given above. The input consists of a corresponding triple $\mathbf{x} \leftrightarrow \mathbf{x}' \leftrightarrow \mathbf{x}''$ and the cameras $\mathsf{P} = [\mathsf{I}|\mathbf{0}]$, $\mathsf{P}' = [\mathsf{R}|\mathbf{t}]$, measured in the first cameras coordinate frame, and $\mathsf{P}'' = [\mathsf{R}'|\mathbf{t}']$, measured in the second cameras coordinate frame.

The cameras P' and P'' are given by

```
Gan_Matrix34* m34P1;
Gan_Matrix34* m34P2;
```

and must be normalized such that $\|\widetilde{\mathsf{C}}'\| = \|\widetilde{\mathsf{C}}''\| = 1$. The function `getRt-Solutions` for extracting cameras from the essential matrix (described in Section 3.3 on page 19 ff.) computes projection matrices that are normalized in this way.

The points $\mathbf{x}_i$, $\mathbf{x}'_i$, $\mathbf{x}''_i$ constituting the correspondences are stored (in normalized coordinates) in the arrays

```
Gan_Vector3* av3Point0;
Gan_Vector3* av3Point1;
Gan_Vector3* av3Point2;
```

For the `i`th correspondencing triple, the scale factor k_i is obtained as follows. First, all necessary values are transformed into the coordinate system of the second camera. The second camera centre is at the origin of course. Measured in the coordinate system of the first camera, the second camera is $\mathsf{P}' = [\mathsf{R}|\mathbf{t}]$, with $\mathbf{t} = -\mathsf{R}\widetilde{\mathbf{C}}'$. The first camera centre transformed into the second view is $\widetilde{\mathbf{C}} = -\mathsf{R}\widetilde{\mathbf{C}}'$, too. So $\widetilde{\mathbf{C}}$ is simply the last column of `m34P1`.

```
Gan_Vector3 c0;
(void)gan_mat34_get_v3r_q(m34P1, &c0);
```

Measured in the coordinate system of the second camera, the third camera is $\mathsf{P}'' = [\mathsf{R}'|\mathbf{t}']$, with $\mathbf{t}' = -\mathsf{R}'\widetilde{\mathbf{C}}''$. So, the third camera centre is obtained as $\widetilde{\mathbf{C}}'' = -\mathsf{R}'^{\top}\mathbf{t}'$.

```
Gan_Vector3 c2;
Gan_Vector3 t2;
Gan_Matrix33 m33R2;
```

```
(void)gan_mat34_get_v3r_q(m34P2, &t2);
(void)gan_mat34_get_m33l_q(m34P2, &m33R2);
(void)gan_mat33T_multv3_q(&m33R2, &t2, &c2);
(void)gan_vec3_negate_i(&c2);
```

The directions $\mathbf{x}_i$, $\mathbf{x}_i'$ and $\mathbf{x}_i''$ are needed in the reference frame, too. For convenience, they are scaled to unit norm.

```
//direction x in second frame (unit norm)
Gan_Vector3 x0n;
Gan_Matrix33 m33R1;
(void)gan_mat34_get_m33l_q(m34P1, &m33R1);
(void)gan_mat33_multv3_q(&m33R1, &av3Point0[i], &x0n);
(void)gan_vec3_unit_i(&x0n);

//direction x' in second frame (unit norm)
Gan_Vector3 x1n = av3Point1[i];
(void)gan_vec3_unit_i(&x1n);

//direction x'' in second frame (unit norm)
Gan_Vector3 x2n;
(void)gan_mat33T_multv3_q(&m33R2, &av3Point2[i], &x2n);
(void)gan_vec3_unit_i(&x2n);
```

The planes $\mathbf{\Pi}$ and $\mathbf{\Pi}'$ are implicitly given by their normals `x0n` and `x2n`. The Euclidean coordinates of the point $\mathbf{X}$ (lying on $\mathbf{x}'$ and satisfying $\|\widetilde{\mathbf{X}}\| = 1$) are `x1n`.

So, $\widetilde{\mathbf{p}}$, $\widetilde{\mathbf{q}}$, $\widetilde{\mathbf{p}}'$ and $\widetilde{\mathbf{q}}'$ can be computed using the function `pointOnPlane` that was defined above.

```
Gan_Vector3 p0, p2, q0, q2;
pointOnPlane(&c0, &x0n, &p0);
pointOnPlane(&c2, &x2n, &p2);
pointOnPlane(&x1n, &x0n, &q0);
pointOnPlane(&x1n, &x2n, &q2);
```

Next, it is checked whether the situation is close to the case $\|\widetilde{\mathbf{q}}\| = \|\widetilde{\mathbf{q}}'\| = 0$, where k_i is undefined.

```
if(p0_sqrlen < 0.001 || p2_sqrlen < 0.001) {
   ... //skip this point correspondence
}
```

Finally, from the norms of $\widetilde{\mathbf{p}}$, $\widetilde{\mathbf{q}}$, $\widetilde{\mathbf{p}}'$ and $\widetilde{\mathbf{q}}'$, the scale factor k_i can be obtained.

```
double scale = sqrt( (q2_sqrlen * p0_sqrlen)
                     / (q0_sqrlen * p2_sqrlen));
```

The final solution for the scale factor k can be obtained as the mean of k_i. Of course, what has been said about reconstruction uncertainty in Section 5.3 applies to the direct algorithm, too. The weighting function suggested there can be applied successfully here, too, as the experimental results in the next section will show.

5.5 Experimental results

This section will compare the results of the algorithms introduced in this chapter. For the sake of brevity, the "scale from reconstructed world points" algorithm will be called *indirect algorithm*, and the "scale from backprojected rays" algorithm will be called *direct algorithm*. Both algorithms have been implemented with and without making use of the weighting factors.

A number of simulations have been performed using the same basic setup as in Section 4.6. A camera located at $(0, 0, -2)$ is looking at an "object" of 25 random points located at the origin. Camera position is changed to obtain a second and a third image.

Simulated measurement error has been introduced to the image points of the first and third image by adding noise vectors (x, y) drawn from a uniform distribution $x, y \in [-\epsilon/2, \epsilon/2]$ with ϵ variying between 0 and 10 pixel. The measurements in the second image remained untouched.

The camera projection matrices for all three cameras were exactly known (up to scale). They had not to be reconstructed beforehand.

The scale factor k was computed from the generated point correspondences and compared to the real scale factor. Error was computed as

$$E = \left| \frac{k_{\text{computed}}}{k_{\text{real}}} - 1 \right|$$

Two series of experiments have been performed. For the first, the camera was moved by 0.5 units in x-direction two times. For the second, the camera was moved by 0.5 units in z-direction two times. Both times, for each value of ϵ 500 experiments were performed. The results are shown in Figures 5.7, 5.8 and 5.9

There are several observations:

- All algorithms perform equally well for motion perpendicular to the viewing direction (Figure 5.7). This is because the displacement between image measurements, and equivalently the angles between the backprojected rays, is bigger in this case than for motion in the viewing direction.

- For motion in the viewing direction (Figures 5.8 and 5.9) the situation is different. The unweighted indirect algorithm lacks behind the others. This is caused by single "outlier" correspondences, where for one

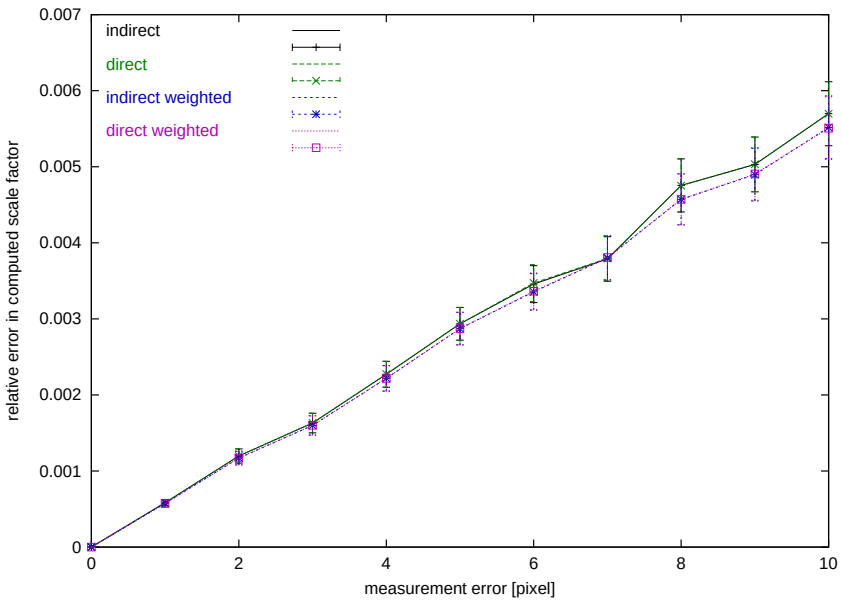

Figure 5.7: *Error in reconstructed scale for motion in x direction and varying measurement error. The errorbars show the standard deviation scaled by* 0.1.

of the camera pairs the reconstructed points very far from the camera. To some extend the same is true for the unweighted direct algorithm.

- The direct algorithms perform better than the indirect ones. A likely explanation for this is the Euclidean nature of the direct approach, which is since geometrically more meaningful.
- For motion in the viewing direction, the stabilizing effect of the weighting function is clearly visible. The weighted versions of both, the direct and the indirect algorithm perform considerably better.
- Ultimately, the weighted direct algorithm yields the best results.

As a conclusion from the experiments, it is suggested that the weighted direct algorithm is used in practice.

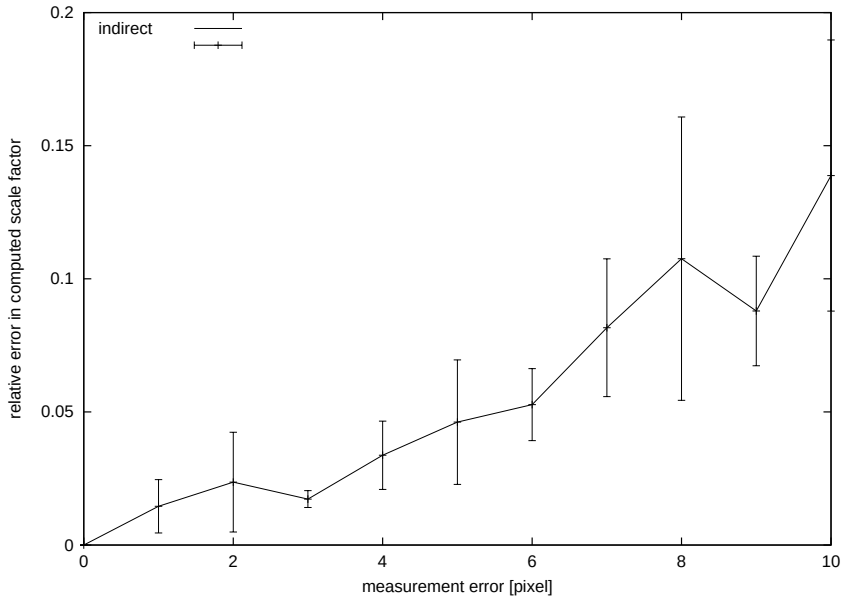

Figure 5.8: *Error in reconstructed scale for motion in z direction and varying measurement error. The errorbars show the standard deviation scaled by* 0.1.

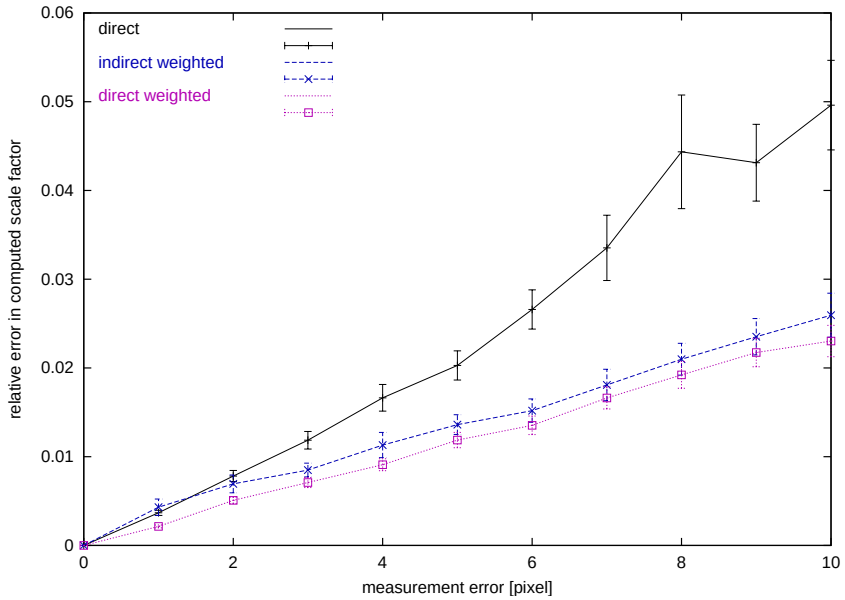

Figure 5.9: *Error in reconstructed scale for motion in z direction and varying measurement error. The errorbars show the standard deviation scaled by* 0.1.

Chapter 6

Reconstructing a Camera Path

Having obtained the relative scale of the triples of consecutive cameras, the remaining task is to put these together to a sequence of camera positions and orientations. In fact, this is rather sraightforward. Some experiments will be performed to see from an exemplary setup what overall results can be achieved from the resulting algorithm.

6.1 Combining camera triples

A slightly different notation will be used in this section. Instead of marking different cameras with primes, cameras will be denoted as P_n, where P_0 is the first camera, P_1 the second and so on. Every P_n is measured in the coordinate frame of the camera P_{n-1} (except for P_0 which is chosen as $[\mathsf{I}|\mathbf{0}]$ per definition). The camera P_n transformed to the coordinate frame of the first camera is labeled P_n^0.

What can be obtained by the algorithms mentioned before is a series of cameras $\mathsf{P}_0 \ldots \mathsf{P}_n$ and the scale factors k_n for the triples $(\mathsf{P}_{n-2}, \mathsf{P}_{n-1}, \mathsf{P}_n)$. The cameras $\mathsf{P}_n = [\mathsf{R}_n|\mathbf{t}_n]$ are normalized such that $\|\mathbf{t}_n\| = 1$.

The reconstructed cameras consist of a perspective projection matrix and a 4×4 transformation matrix

$$\mathsf{P}_n = [\mathsf{I}|\mathbf{0}] \begin{bmatrix} \mathsf{R}_n & \mathbf{t}_n \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{6.1}$$

where the latter matrix realizes an Euclidean transformation of world points from the (reference) coordinate frame of P_{n-1} into the coordinate frame of P_n.

To transform a world point $\mathbf{X}$ from the coordinate frame of the *first* camera into the frame of P_n, it has to be transformed into the second camera

frame, then into the third and so on, up to P_n:

$$\mathbf{X}_{cam_n} = \begin{bmatrix} \mathsf{R}_n & k_n^0 \mathbf{t}_n \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathsf{R}_{n-1} & k_{n-1}^0 \mathbf{t}_{n-1} \\ \mathbf{0}^\top & 1 \end{bmatrix} \cdots \begin{bmatrix} \mathsf{R}_1 & k_1^0 \mathbf{t}_1 \\ \mathbf{0}^\top & 1 \end{bmatrix} \mathbf{X}. \tag{6.2}$$

The values k_n^0 that are used here, denote the relative scale of the camera pairs $(\mathsf{P}_{n-1}, \mathsf{P}_n)$ and $(\mathsf{P}_0, \mathsf{P}_1)$, that is the ratio of the euclidean distances between the camera centres $\|\widetilde{\mathbf{C}}_1 - \widetilde{\mathbf{C}}_0\|$ and $\|\widetilde{\mathbf{C}}_n - \widetilde{\mathbf{C}}_{n-1}\|$.

They can be computed from the k_n as

$$k_1^0 = 1 \tag{6.3}$$

$$k_n^0 = k_n k_{n-1}^0 \quad \text{for } n \geq 2. \tag{6.4}$$

By writing out Equation (6.2) and left-multiplying by the perspective projection matrix $[\mathsf{I}|\mathbf{0}]$, the cameras $\mathsf{P}_n^0 = \left[\mathsf{R}_n^0 | \mathbf{t}_n^0\right]$ are obtained, having

$$\mathsf{R}_1^0 = \mathsf{R}_1 \tag{6.5}$$

$$\mathbf{t}_1^0 = \mathbf{t}_1 \tag{6.6}$$

$$\mathsf{R}_n^0 = \mathsf{R}_n \mathsf{R}_{n-1}^0 \quad \text{for } n \geq 2 \tag{6.7}$$

$$\mathbf{t}_n^0 = \mathsf{R}_n \mathbf{t}_{n-1}^0 + k_n^0 \mathbf{t}_n \quad \text{for } n \geq 2 \tag{6.8}$$

Since the P_n^0 are all in one reference frame (the first cameras coordinate system), the camera centres can be obtained as[1]

$$\widetilde{\mathsf{C}}_n = -\mathsf{R}_n^{0\top} \mathbf{t}_n^0. \tag{6.9}$$

The orientation of the cameras is implicitly given by R_n^0.

Implementation. The values for R_n^0 and $\mathbf{t}_n^0$ are stored in `m33R0` and `v3t0`. These are initialized with I and $\mathbf{0}$, and will be updated in each subsequent step.

```
Gan_Matrix33 m33R0;
(void)gan_mat33_ident_q(&m33R0);
Gan_Vector3 v3t0;
(void)gan_vec3_zero_q(&v3t0);
```

In the same way, k_n^0 is represented by `k0`.

```
double k0 = 1.0;
```

Each time a new camera projection matrix `Gan_Matrix34* m34P` has been reconstructed, these variables are updated. The scale factor k computed for `m34P` and the previous two cameras is needed, too. It is supplied in `double ki`. Of course, for the second camera (the first one reconstructed), `k0` must be set to 1.

First, the R and $\mathbf{t}$ parts are extracted from `m34P`.

[1] using $\mathbf{t} = -\mathsf{R}\widetilde{\mathbf{C}}$.

```
Gan_Matrix33 m33Ri;
Gan_Vector3 v3ti;
(void)gan_mat34_get_m33l_q(m34P, &m33Ri);
(void)gan_mat34_get_v3r_q(m34P, &v3ti);
```

Then `m33R0`, `v3t0` and `k0` are updated using Equations (6.7), (6.8) and (6.4).

```
k0 = k0 * ki;
Gan_Matrix33 m33R0temp;
(void)gan_mat33_rmultm33_q(&m33Ri, &m33R0, &m33R0temp);
m33R0 = m33R0temp;
Gan_Vector3 v3t0temp;
(void)gan_mat33_multv3_q(&m33Ri, &v3t0, &v3t0temp);
(void)gan_vec3_scale_q(&v3ti, k0, &v3t0);
(void)gan_vec3_add_i2(&v3t0temp, &v3t0);
```

After each step, the coordinates of the centre of the (currently) last camera can be computed.

```
Gan_Vector3 v3C;
(void)gan_mat33T_multv3_q(&m33R0, &v3t0, &v3C);
(void)gan_vec3_negate_i(&v3C);
```

Of course there is still an *overall* scale ambiguity in the reconstructed path. This can not be resolved without using additional information. Such information might be odometry data or knowledge of the true distance of certain points in the world.

6.2 Experimental results

This section is somewhat different from the other *experimental results* sections in this paper. It does not evaluate different algorithms against each other. It does not try to arrive at a "final truth" but merely shows for "some configuration" what can be achieved when the different steps discussed before are put together.

The following setup is used. A robot moves on the floor of a rectangular room. The room's size is $1000 \times 500 \times 300$ units. The walls, floor and ceiling are covered by a grid that has a width of 25 units. The grid points are used for generating image measurements (and correspondences).

The robot is driving a semicircle, as shown in Figure 6.1. The same camera is used as in Sections 4.6 and 5.5. It is mounted on top of the robot, 20 units above the ground. The viewing direction is the motion direction. A new image is available everytime the robot has moved by 10 units, resulting in a total of 42 images. Simulated measurement error is

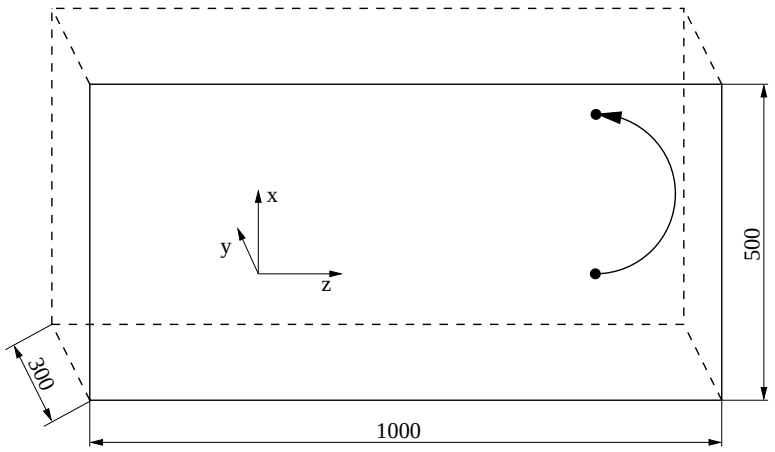

Figure 6.1: *Setup for the experiments.*

introduced to the image points by adding noise vectors (x, y) drawn from a uniform distribution $x, y \in [-\epsilon/2, \epsilon/2]$.

The following algorithms are used to reconstruct the camera path:

- The essential matrices between consecutive images are estimated from point correspondences using the normalized 4-point algorithm. (Section 4.4)
- Cameras are reconstructed from the essential matrices using Horn's method. (Section 3.3)
- Relative scale between consecutive camera pairs is obtained from the reconstructed cameras and point correspondences using the weighted direct (scale from backprojected rays) algorithm. (Section 5.4)
- From the reconstructed cameras and the relative scale factors, a complete sequence of camera positions is computed. (Section 6.1)

Figures 6.2, 6.3 and 6.4 show the real path and the reconstructed paths for simulated measurement error varying between 1 and 3 pixels. The reconstructed paths have been isometrically scaled, such that the distance between the first reconstructed camera pair equals the real distance between the first camera pair. The reconstructed paths have also been translated, such that the starting positions match.

Although the setup is not really comparable to a realworld situation, two observation are made.

- The errors in the computed relative scale between camera pairs add up fast, even for small errors in the image measurements. The computed

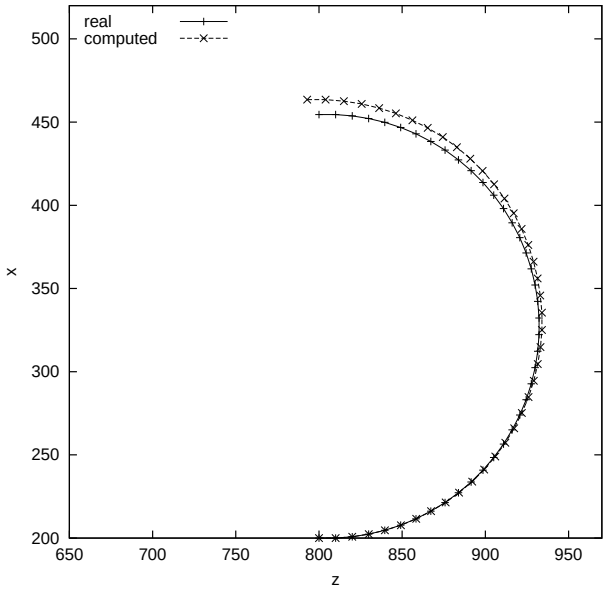

Figure 6.2: *Real and reconstructed path for simulated measurement error* $\epsilon = 1$ *pixel.*

final positions quickly diverges from the real one as measurement error increases.

- In contrast to that, the relative position and orientation of the cameras seems to be quite stable.

The first observation suggests, that it will not be sufficient to use additional information to determine overall scale. Instead, such information should also be used to correct scale for indiviual segments of the computed path, to prevent the propagation of scale errors. How to perform this exactly, is a subject that still has to be looked into.

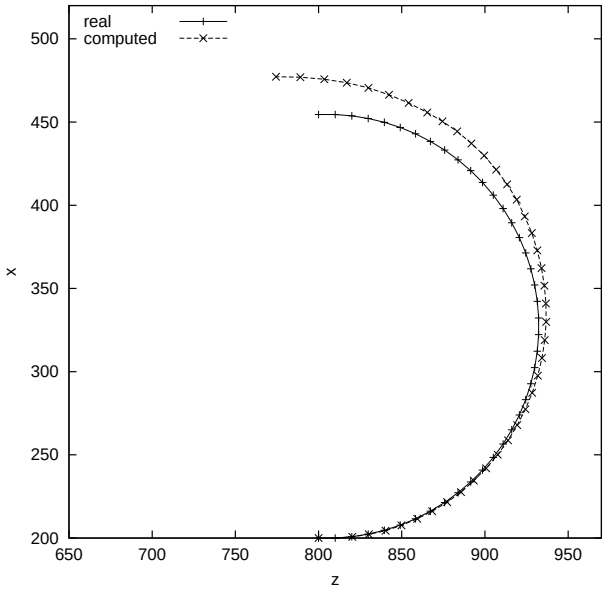

Figure 6.3: *Real and reconstructed path for simulated measurement error* $\epsilon = 2$ *pixel.*

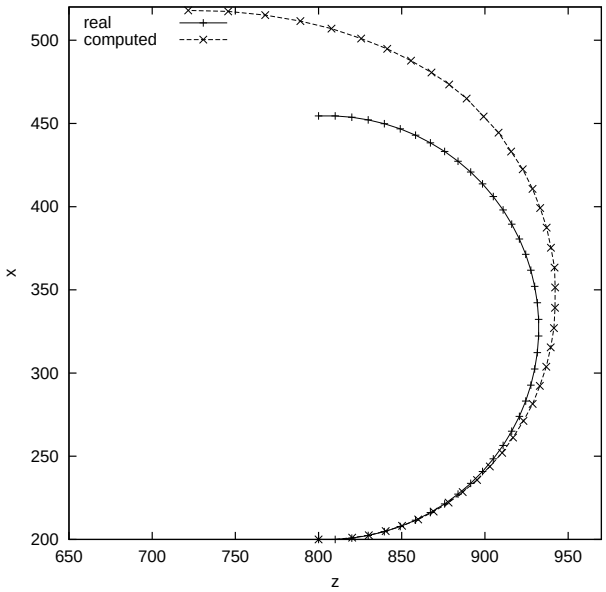

Figure 6.4: *Real and reconstructed path for simulated measurement error* $\epsilon = 3$ *pixel.*

Chapter 7

Conclusion

An algorithm has been devised and implemented which reconstructs a sequence of camera positions and orientations, unique up to an overall scale factor. The techniques that have been used are not the most sophisticated available, but they are simple and fast. The most expensive step of the algorithm is the estimation of a solution for the essential matrix, where singular value decomposition is used. The computational complexity of computing the SVD of an $m \times n$ matrix is $O(mn^2)$. The number of columns is fixed in the given context, so the complexity of the algorithm described here is $O(n^2)$ for every step, given n point correspondences. Several experiments have been conducted using simulated data, to evaluate the quality of the algorithm and its various steps.

For the 8-point variations of the algorithm one big advantage over the information that can be obtained from odometry sensor or laser range finders is gained: Full spatial orientation is computed. This means, the information that can be gained is not restricted to position on a ground plane. This is especially advantageously for uneven terrain, where up- or down- movement may occur.

The question whether and where the approach fits into the context of position tracking of a mobile robot is left unanswered for now, since there are still many issues that have to be investigated. First and foremost, the gap between raw images and point correspondences has to be bridged. The algorithm may be evaluated in a real world setup then.

The results obtained in Section 6.2 suggest that in order to be applicable in practice the approach has to be integrated with other sensory information, e.g., odometry sensors and laser range finders. How exactly this can be done is subject of further work.

Finally, more advanced methods may be implemented to achieve higher-quality results. One drawback of the current approach is, that only two images are used at a time. There are methods for estimating cameras and scene structure from more than two images, like bundle adjustment [2]. It

is inconvenient, too, that the approach requires the camera to be calibrated. There are auto-calibration methods to calibrate the camera "on the fly" [2, 1].

However, the next step will be to implement a feature-tracker and the more advanced issues should be deferred, until "real" experimental results have been obtained.

Bibliography

[1] Andrea Fusiello. Uncalibrated euclidean reconstruction: A review. *Image and Vision Computing*, 18:555–563, May 2000.

[2] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge Univerity Press, 2000.

[3] Berthold K. P. Horn. Recovering baseline and orientation from essential matrix, 1990. http://www.ai.mit.edu/people/bkph/publications.html.

[4] Berthold K. P. Horn. Projective geometry considered harmful, 1999. http://www.ai.mit.edu/people/bkph/publications.html.

[5] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, pages 133–135, September 1981.

[6] Q.-T. Luong and O. D. Faugeras. The fundamental matrix: Theory, algorithms, and stability analysis. *International Journal of Computer Vision*, 17:43–76, 1996.

[7] Philip McLauchlan et al. Gandalf: The fast computer vision and numerical library. http://gandalf-library.sourceforge.net/.

[8] Matthias Muehlich and Rudolf Mester. The role of total least squares in motion analysis. In *Proceedings of the European Conference on Computer Vision 1998*, pages 305–321. Springer, 1998.

[9] William H. Press et al. *Numerical Recipes in C, second edition*. Cambridge Univerity Press, 1992.

[10] Miroslav Trajkovic. Rigid motion estimation from point correspondences. http://www.dai.ed.ac.uk/CVonline/.