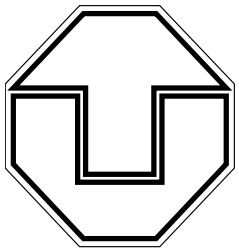
Diplomarbeit

Application of a monocular camera as a motion sensor for mobile robots

Tobias Pietzsch
geboren am 25. Dezember 1975 in Rodewisch



Fakultät Informatik
Technische Universität Dresden

eingereicht am 1. März 2004

Betreuender Hochschullehrer: Prof. Dr. Steffen Hölldobler
Betreuer: Dr. Axel Großmann

Contents

Chapter 1

Introduction

Reliable navigation is an important ability for mobile robots, crucial to the successful achievement of most higher-level goals. Mobile robot localization and environment mapping have been important research topics for the past years. Now, the state of the art is referred to as Simultaneous Localisation And Mapping (SLAM), where an environment map is build and the robot is localized therein, simultaneously. Traditionally, most approaches use precise distance sensors, such as laser range finders. Often, due to the properties of the sensors localization is carried out in 2D maps, restricting applicability to planar structured environments, such as offices and corridors.

Vision sensors are capable of providing much more information, and can be seen as the best sensing modality for mobile robots. The data provided by a single camera is essentially 2D as well, but using stereo cameras or monocular image sequences, full spatial information can be reconstructed. Reconstruction from images is a thoroughly investigated topic in computer vision. The major obstacle for extensive use of vision sensors is computational complexity. But with the increase in computing power we experienced in the past few years, full real-time spatial information from cameras seems to be within our grasp in the near future. Our goal is to explore potential uses for vision sensors in the context of robot localization and mapping. The focus of this thesis is on the application of a camera as a position sensor. On the other hand, large parts of the presented work are prerequisites for the reconstruction of scene structure as well, e.g. feature tracking and outlier detection.

In previous work [32] we developed an efficient method for estimating camera/robot motion from a sequence of monocular images using classical structure from motion methods. As a starting point we used known 2D feature positions and correspondence between them across images, i.e., we assumed that some level of abstraction from raw images had already been achieved.

A major goal of the present work is to supply the missing preprocessing

stages that will allow us to start from raw image data. Specifically, we need to identify 2D features and track them through a sequence of images. Our motion estimation method assumes that all the feature points correspond to fixed 3D entities in a rigid scene, and thus that the image motion of features is due purely to the motion of the camera. Therefore, before automatically detected features can be used as input data for position estimation, we need to pass them through an outlier detection stage where features that do not conform to our assumptions are rejected.

Another goal is to evaluate our position estimation approach in real-world experiments. As a platform for experimentation we use a Pioneer 2 mobile robot which is equipped with a CCD camera. We compare our results with those of a commercial Bundle Adjustment solution.

All stages of the algorithm have been implemented, integrated and tested. Features are detected and tracked in real-time by an on-board laptop. The resulting data is passed through an outlier detection stage where incorrectly tracked features are discarded. The path of the robot is reconstructed from the remaining inlier data. The result is a system which computes the trajectory of the robot from a sequence of camera images. The method is suitable for real-time application. However, some open problems remain. Presently, we cannot handle pure rotations. Errors in the reconstructed path accumulate over time. Although we can improve our previous method [32] in this respect, we still cannot compete with the accuracy of odometry.

Structure from motion (SFM) methods have a long history in computer vision, and recently have been applied to localization as well. They can be roughly categorized into two classes. Firstly, there are recursive or Kalman filter-based methods, where images of a sequence are used to update a dynamical model of 3D structure and motion. Recently, Chiuso et.al. [11] presented a causal structure from motion approach, working in real-time. Davison [12] applied a similar approach to camera localization. He uses a small set of distinguished features that serve as long-term landmarks, i.e., can be relocated after being out of view for some time.

Secondly, there are approaches that simultaneously process several images and use the inherent constraints in multiple view geometry to estimate both, 3D structure and camera locations. In contrast to recursive methods, no explicit model of camera motion is used, i.e just several images of a scene from different perspectives are needed, not a video sequence. Multiple view relations have been a very active field of research, dating back at least to the introduction of the 8-point algorithm by Longuet-Higgins [29]. Hartley and Zisserman [21] give an excellent overview of the subject.

This thesis is organized as follows. Chapter 2 introduces the basics of the projective geometric framework and two-view relations. We also discuss preliminary work on camera calibration and the integration of our algorithms with the existing robotics system architecture. Chapter 3 explains how

point features are extracted and tracked between images. We present our implementation of the Kanade-Lucas-Tomasi feature tracker. The problem of outliers, i.e., features that are incorrectly tracked, and how they can be detected is discussed in Chapter 4. At this point, we will have achieved all the necessary preprocessing steps and our motion estimation method [32] can be readily applied. An extension to the method is introduced in Chapter 5, where we try to exploit additional information available from redundant two-view motion estimates. Chapter 6 presents results on simulated and real-world experiments. Chapter 7 is the conclusion.

Chapter 2

Foundations and Preliminary Work

2.1 Notation

Throughout this document, the following notational conventions are used.

- Scalars are denoted by italic symbols, e.g., a, b, c.
- Matrices are denoted by Sans Serif symbols, e.g., E, F. Usually, matrix dimensions are clear from the context, but occasionally $row \times columns$ subscripts are used for clarification, e.g., $\mathsf{M}_{3\times 4}$.
- Vectors are denoted by boldface symbols, e.g., $\mathbf{x}$, $\mathbf{X}$. Mostly, geometric relations are presented in a projective framework and vectors are homogeneous entities. Whenever a vector represents Euclidean coordinates, this is indicated by a tilde superscript. Lower-case symbols represent vectors in two-dimensional space, e.g., $\mathbf{x} = (x, y, w)^\top \in \mathbb{P}^2$ or $\widetilde{\mathbf{x}} = (x, y)^\top \in \mathbb{R}^2$. Upper-case symbols represent vectors in three dimensions, e.g., $\mathbf{X} = (X, Y, Z, W)^\top \in \mathbb{P}^3$ or $\widetilde{\mathbf{X}} = (X, Y, Z)^\top \in \mathbb{R}^3$.

Equality for homogeneous entities is only up to scale,[1] so the $=$ symbol is used in the sense of "equal up to scale" in the projective context.

Matrix representation of cross products. Taking the cross product of two vectors can be represented as multiplication of a vector and a skew-symmetric matrix.

[1] For instance, the homogeneous points $(sX, sY, sZ, s)^\top$ with $s \neq 0$ all represent the same point $(X, Y, Z)^\top$ in Euclidean space.

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

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

It is easily verified that

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

for all $a, b \in \mathbb{R}^3$.

Quaternion representation of rotations. Sometimes, unit quaternions will be used to represent rotations in three-dimensional space, rather than the more conventional 3×3 rotation matrices.

Unit quaternions are represented by 4-vectors $\mathbf{q} = (w, x, y, z)^\top$, subject to the constraint $\|\mathbf{q}\| = 1$.

Every rotation can be interpreted as a right handed rotation of $-2\pi < \theta < 2\pi$ radians about some axis. If $\widetilde{\mathbf{x}}$ is a unit vector representing the axis, the associated quaternion rotation is

$$\mathbf{q} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2})\widetilde{\mathbf{x}} \end{bmatrix}$$

Appendix A explains in more detail the properties of quaternions and their relationship to rotation matrices.

2.2 Camera model and calibration

To make effective use of image measurements, we need a model of the image formation process. Once the model parameters are known, the world and camera coordinate frames can be related. The image projection of a given scene point can be calculated and, given an image point, a ray can be determined on which the corresponding scene point must lie.

In this section, we shortly describe the camera model we use and how its parameters are obtained. First, the basic pinhole camera model is introduced which can be conveniently realised as a linear mapping from $\mathbb{P}^3$ to $\mathbb{P}^2$. Unfortunately, real cameras deviate from this simple model. Lens systems, assuming the function of the pinhole, introduce nonlinear distortions to the imaging process. Once lens distortion has been removed from the image measurements, it is valid to use the linear model. We will describe how lens distortion is modeled and, finally, give some details on the camera calibration procedure.

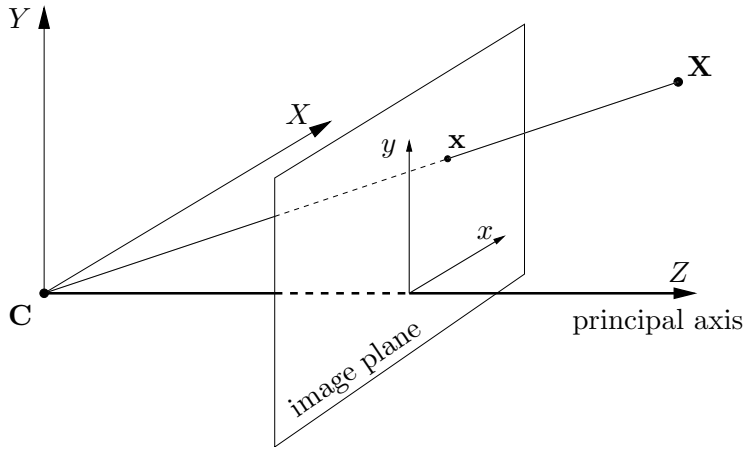


Figure 2.1: *The pinhole camera model.*

2.2.1 The pinhole camera model

The pinhole model describes image formation as a central projection of points in space onto a plane. As illustrated in Figure 2.1, the image $\mathbf{x}$ of a 3D point $\mathbf{X}$ is the intersection of the ray from the camera centre $\mathbf{C}$ to $\mathbf{X}$ with the image plane. The axes of the camera-centered world coordinate system are labeled X, Y, Z. The axes of the image system are x, y. The line perpendicular to the image plane passing through the camera center, i.e., the camera Z-axis, is referred to as the *principal axis*. It pierces the image plane at the *principal point*. The perpendicular distance from the camera centre to the image plane is referred to as the *focal length*, f.

Actually, the situation depicted in Figure 2.1, where camera and world coordinate system are the same, is somewhat idealized. If furthermore the scales of the image and world systems match and the focal length is $f = 1$, the projection of a world point onto the image plane then simply is

$$x = \frac{X}{Z}$$
$$y = \frac{Y}{Z},$$

where (x, y) and and (X, Y, Z) are the Euclidean coordinates of the image and world point, respectively. Using homogeneous coordinates $\mathbf{X} = (X, Y, Z, W)^\top$, $\mathbf{x} = (x, y, w)^\top$, this can be expressed conveniently as a linear system of equations

$$\mathbf{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X}.$$

This basic 3D to 2D projection matrix $\mathsf{P} = [\mathsf{I}|\mathbf{0}]$ is multiplied now by a 4×4 matrix T to the right to transform scene points from the global world coordinate frame to camera-centered coordinates, and by a 3×3 matrix K to the left for certain manipulations of the image coordinate frame, such as altering pixel size and so on.

$$\mathsf{P} = \mathsf{K}[\mathsf{I}|\mathbf{0}]\mathsf{T}$$

More specifically, the transformation matrix T is

$$\mathsf{T} = \begin{bmatrix} \mathsf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix},$$

where R and $\mathbf{t}$ denote relative orientation and translation, respectively, between the world and camera coordinate frames. The parameters of T are referred to as the *external parameters* or *exterior orientation* of the camera.

The matrix K comprises the *internal parameters* or *interior orientation*. K is also known as the *camera calibration matrix*.

$$\mathsf{K} = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

The parameters f_x, f_y denote the focal length in terms of pixel dimensions in the x and y direction, respectively. The origin of the image coordinate system may be shifted. This is taken into account by x_0 and y_0, the image coordinates of the principal point. Finally, s is the *skew parameter*, which is zero for most normal cameras. It indicates the skewness of the image x and y axes, i.e., whether the axes deviate from orthogonality.

2.2.2 Lens distortion

For real cameras, the pinhole model is not perfectly valid since light rays pass through a lens instead of a pinhole. Nonlinearities occur which are inherent to the physics of refraction by lenses.[2] Figure 2.2 shows an example image in which the effects of nonlinear distortion are clearly visible. The camera model should incorporate these nonlinearities. The model we use is known as the "Plumb Bob" model[3] and was first introduced by Brown [10] in 1966. Besides the elements of the calibration matrix K the model includes two parameters for radial-symmetric distortion, A_1 and A_2, and two parameters for tangential distortion, B_1 and B_2.

[2] Details on the optics of image formation and the nonlinear effects that arise can be found in [31], for instance.

[3] In a perspective projection, the image of a straight line will be a straight line if no lens distorion is present. Deviations from straightness can be related to lens distortion. The distortion parameters can be determined from the images of straight lines. As "straight lines in the world" Brown initially used fine white thread streched by plumb bobs. Hence the name.

(a) (b)

Figure 2.2: *(a) An image from a simple CCD camera. Radial "barrel" distortion is clearly visible. (b) The same image with radial distortion removed. Now straight lines in the world appear as straight lines in the image, e.g., the door frame on the left or the edge of the blackboard on the right.*

Projection of scene points is modeled as follows. First, the scene point is transformed into the camera coordinate system and its normalized projection (i.e., the calibration matrix is the identity matrix) is obtained as

$$\mathbf{x}_n = [\mathsf{I}|0]\mathsf{T}\mathbf{X}.$$

Let $\widetilde{\mathbf{x}}_n = (x, y)^\top$ be the Euclidean coordinates of $\mathbf{x}_n$. The distorted point $\widetilde{\mathbf{x}}_d$ is then obtained as

$$\widetilde{\mathbf{x}}_d = (1 + A_1 r^2 + A_2 r^4)\widetilde{\mathbf{x}}_n + \begin{bmatrix} 2B_1 xy + B_2(r^2 + 2x^2) \\ B_1(r^2 + 2y^2) + 2B_2 xy \end{bmatrix}, \qquad (2.1)$$

where $r = \sqrt{x^2 + y^2}$ is the distance of $\widetilde{\mathbf{x}}_n$ from the principal point. Finally, the distorted point is transformed to pixel coordinates by the calibration matrix,

$$\mathbf{x} = \mathsf{K}\mathbf{x}_d.$$

The computation can also be performed in the reverse direction. If we know the internal parameters, that is, the matrix K and the distortion parameters A_1, A_2, B_1, B_2, we can find the normalized projection of any given image point $\mathbf{x}$. The distorted normalized point $\mathbf{x}_d$ can be obtained as $\mathbf{x}_d = \mathsf{K}^{-1}\mathbf{x}$. To find $\mathbf{x}_n$, the distortion has to be removed from $\mathbf{x}_d$. This is achieved by using Equation (2.1) to iteratively improve an initial guess of $\mathbf{x}_n$, for instance $\mathbf{x}_n = \mathbf{x}_d$.

2.2.3 Camera calibration

To estimate the internal parameters of the robot camera, we use Matlab along with the Camera Calibration Toolbox by Jean-Yves Bouguet [8]. For

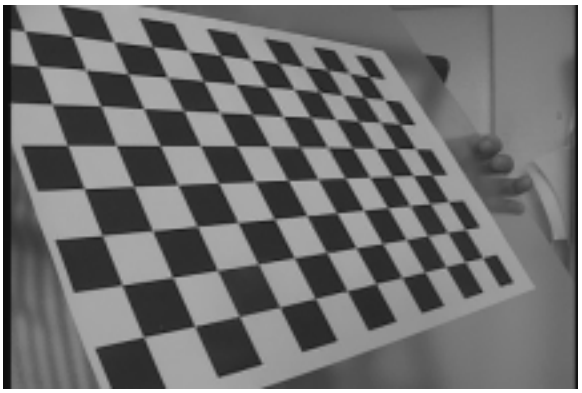


Figure 2.3: *Sample image of the planar calibration pattern.*

the calibration procedure several images of a planar checkboard pattern from different perspectives are needed, as the one shown in Figure 2.3. The size of the squares in the pattern must be known to the program. With a little user interaction, the program extracts the corners of the squares from the images. The corner coordinates are then used to compute the internal orientation of the camera in a nonlinear minimization procedure. The following parameters are simultaneously estimated:

f_x, f_y	focal length in terms of pixel width and height.
x_0, x_0	pixel coordinates of the principal point.
A_1, A_2	parameters of radial-symmetric distortion.
B_1, B_2	parameters of radial-asymmetric and tangential distortion.

References to papers on the details of the model and the minimization procedure used can be found on the program's website [8].

2.3 Epipolar geometry

Epipolar geometry is the intrinsic projective geometry between two views of a rigid scene. The relation between the two views is independent of the scene structure. It only depends on the internal parameters of the cameras and their relative pose.

The entities involved in the epipolar relations will be briefly introduced now. The situation is illustrated in Figure 2.4. A scene point $\mathbf{X}$ is projected onto the image planes of the two cameras respectively. The corresponding image point $\mathbf{x}$ in the first view is located at the intersection of the ray from the first camera centre $\mathbf{C}$ to $\mathbf{X}$ with the image plane. Likewise, where $\mathbf{x}'$ is the projection of $\mathbf{X}$ onto the second cameras image plane.

The line joining the first and second camera centres $\mathbf{C}$ and $\mathbf{C}'$ is known as the *baseline*, $\mathbf{b}$. The points where the baseline intersects the image planes

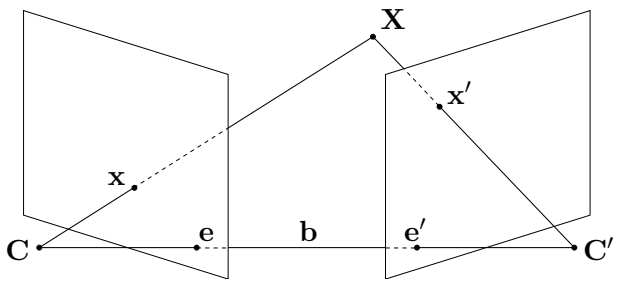


Figure 2.4: *Entities in two view geometry, where the ones marked with a prime refer to the second view.*

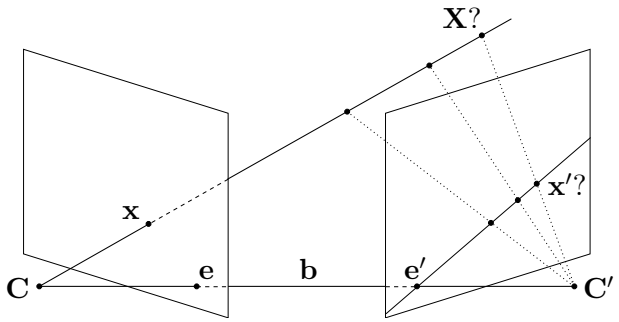


Figure 2.5: *Two view geometry. The epipolar constraint.*

are referred to as *epipoles*. They are denoted $\mathbf{e}$ and $\mathbf{e}$‘. The epipoles can also be thought of as being the respective projections of the centre of one camera onto the image plane of the other. Finally, as can be seen from Figure 2.4, all the entities $\mathbf{X}$, $\mathbf{C}$, $\mathbf{C}'$, $\mathbf{x}$, $\mathbf{x}'$, $\mathbf{e}$, $\mathbf{e}'$, $\mathbf{b}$ are coplanar. Such a plane is referred to as an *epipolar plane*. Clearly, regardless of the positions of scene point $\mathbf{X}$ and its projections, every epipolar plane will always contain the baseline (an thus the epipoles, the camera centres).

Now, without knowledge of scene structure, i.e., $\mathbf{X}$ is unknown, one constraint on the images arises: From the location of the projection $\mathbf{x}$ on the first image plane, we can infer that the corresponding scene point must lie somewhere along the ray backprojected from $\mathbf{C}$ through $\mathbf{x}$ (see Figure 2.5). When all those possible scene points are projected onto the second image plane, we see that the possible projections $\mathbf{x}'$ are to lie on a line. This line is referred to as the *epipolar line corresponding to* $\mathbf{x}$. Of course, reversing the roles of the views, $\mathbf{x}$ is constraint to lie on the epipolar line defined by $\mathbf{x}'$.

2.3.1 The fundamental matrix

The mapping from points in one view to epipolar lines in the second can conveniently be represented by a 3×3 matrix, the fundamental matrix F. The epipolar line corresponding to $\mathbf{x}$ is

$$\mathbf{l}' = \mathsf{F}\mathbf{x}.$$

Since $\mathbf{x}'$ is constrained to lie on $\mathbf{l}'$

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

This leads to a very concise expression for the epipolar constraint: For any pair of corresponding points $\mathbf{x} \leftrightarrow \mathbf{x}'$

$$\mathbf{x}'^{\top}\mathsf{F}\mathbf{x} = 0. \tag{2.2}$$

The matrix F encodes all the relations between two views. Given the parameters of the two cameras, F is computed straightforwardly. It is also possible to compute the fundamental matrix "the other way around". Seven corresponding pairs $\mathbf{x} \leftrightarrow \mathbf{x}'$ in general configuration determine a matrix F, such that (2.2) is fulfilled on all of them.[4]

2.3.2 The essential matrix

When the cameras are calibrated, the effect of internal parameters can be removed from the image points. This results in *normalized image coordinates*, $\widehat{\mathbf{x}}$ denoting what the location of the projection of $\mathbf{X}$ would be, were it taken by a perfect pinhole camera (whose calibration matrix is the 3×3 identity matrix).

The normalized equivalent of the fundamental matrix is the *essential matrix* E with the following normalized epipolar constraint: For any pair of corresponding points $\widehat{\mathbf{x}} \leftrightarrow \widehat{\mathbf{x}}'$

$$\widehat{\mathbf{x}}'^{\top}\mathsf{E}\widehat{\mathbf{x}} = 0. \tag{2.3}$$

Unlike the fundamental matrix, the essential matrix depends on the external parameters of the cameras only. The world reference frame can be attached to the first camera, such that $\widehat{\mathsf{P}} = [\mathsf{I}|\mathbf{0}]$. If the corresponding second camera is $\widehat{\mathsf{P}}' = [\mathsf{R}|\mathbf{t}]$, the essential matrix is

$$\mathsf{E} = [\mathbf{t}]_{\times}\mathsf{R}. \tag{2.4}$$

[4] The fundamental matrix has seven degrees of freedom. It is only defined up to scale, so there are eight independent ratios of its elements. Additionally, it is required $rank(\mathsf{F}) = 2$ since F is a mapping from 2-Dimesional to a 1-Dimensional space. This leaves 7 degrees of freedom, F is determined by 7 point correspondences.

Five corresponding pairs $\widehat{\mathbf{x}} \leftrightarrow \widehat{\mathbf{x}}'$ in general configuration determine a matrix E such that (2.3) is fulfilled on all of them.[5] It is possible to decompose E to obtain the relative orientation R, $\mathbf{t}$ of the cameras. In contrast, from a fundamental matrix it is only possible to reconstruct the cameras up to a projectivity.

2.4 Sequential alignment of image triples

This section revises the pose estimation method described in our previous work [32]. This method was used as a starting point for the current work - later chapters of this paper will describe how it was extended and improved. The method is based on the estimation of two-view epipolar relations and camera pose recovery from these. It is assumed, that the world in which the robot (camera) moves is static. Thus, epipolar geometry is a valid model, although the different views of the scene are taken at different times. From a sequence of images of the scene, the method tries to recover the path that the camera traversed.

The algorithm can be decomposed into three parts:

1. From corresponding points in two consecutive images, compute an estimate of the essential matrix E. The essential matrix is then decomposed to obtain the relative pose between the two cameras.

2. Combine two consecutive pairs of relatively oriented images to form an oriented image triple. The problem here is that two-view orientation can only be determined up to scale. The scaling factors for two pairs of views may vary. It is necessary to use corresponding points in three views to compute the proportion of the scaling factors.

3. Once the triples have been obtained, it is straightforward to connect them to a complete sequence of poses. Since consecutive triples overlap in one image pair, the proportions of the scaling factors can be computed easily.

The remainder of this section shortly explains each of these steps. See [32] for a detailed explanation.

2.4.1 Relative orientation of two views

To find the relative orientation of a pair of cameras, the essential matrix E relating the views must be estimated. This is done using the normalized

[5] The essential matrix has five degrees of freedom, three for the rotation and two for the direction of translation (since there is an overall scale ambiguity, amount of translation is insignificant). In addition to the constraints on F, it is required that two of the singular values of E are equal and the third one is 0.

8-point algorithm [29]. The essential matrix can be decomposed then, to obtain the relative orientation of the cameras. We use the decomposition proposed by Horn [23].

Recalling the definition of the normalized epipolar constraint (2.3), the essential matrix must fulfill

$$\widehat{\mathbf{x}}'^{\top}\mathsf{E}\widehat{\mathbf{x}} = 0. \tag{2.5}$$

for any pair of corresponding points. Using $\widehat{\mathbf{x}} = (x, y, 1)$ and $\widehat{\mathbf{x}}' = (x', y', 1)$ this is written out as

$$x'xe_{11} + x'ye_{12} + x'e_{13} + y'xe_{21} + y'ye_{22} + y'e_{23} + xe_{31} + ye_{32} + e_{33} = 0, \tag{2.6}$$

a linear equation in the elements of E, where e_{ij} denotes the i-th row, j-th column entry of E. Using a set of n correspondences $\widehat{\mathbf{x}}_i \leftrightarrow \widehat{\mathbf{x}}'_i$, a system of linear equations is formed

$$\mathsf{A}\mathbf{e} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{e} = 0, \tag{2.7}$$

where the 9-vector $\mathbf{e} = (e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}, e_{31}, e_{32}, e_{33})^{\top}$ comprises the elements of E. For 8 or more correspondences a least squares solution for E can be found by minimizing $\|\mathsf{A}\mathbf{e}\|$ subject to $\|\mathbf{e}\| = 1$. This is the 8-point algorithm as introduced by Longuet-Higgins [29].

In its original form, the algorithm is very suspective to noise since the quantity that is minimized is not geometrically or statistically meaningful. Hartley [22] shows, that the algorithm may be significantly improved by preconditioning the point correspondences. Several heuristics for the conditioning transformation were proposed. We use the one suggested by Hartley and Zisserman [21]. The resulting method, called the *normalized 8-point algorithm*[6] is briefly summarized:

[6] The term "normalized" refers to the preconditioning in this context, not to the transformation to normalized coordinates and ideal pinhole cameras.

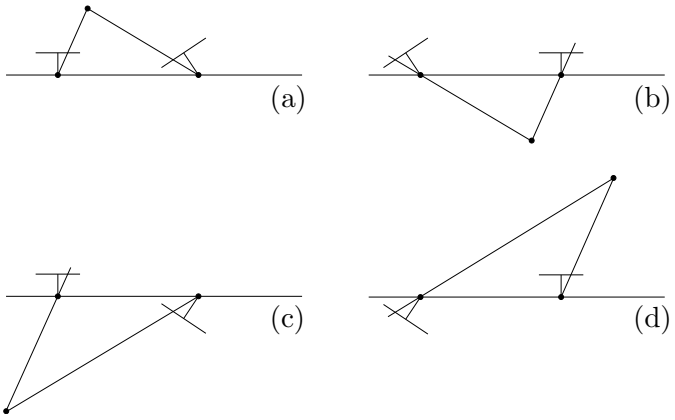


Figure 2.6: *Four possible configurations corresponding to a single essential matrix. A triangulated scene point will lie in front of both cameras in one configuration only.*

1. **Normalization:** Transform the image coordinates according to $\breve{\mathbf{x}}_i = \mathsf{T}\widehat{\mathbf{x}}_i$ and $\breve{\mathbf{x}}'_i = \mathsf{T}'\widehat{\mathbf{x}}'_i$.

   T and T' are normalization transformations, performing 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 $\sqrt{2}$.

2. Find the essential matrix $\breve{\mathsf{E}}$ from the correspondences $\breve{\mathbf{x}}_i \leftrightarrow \breve{\mathbf{x}}'_i$ as a least squares solution to Equation (2.7).

3. **Denormalization:** Obtain the essential matrix E corresponding to the unconditioned points as $\mathsf{E} = \mathsf{T}'^{\top}\breve{\mathsf{E}}\mathsf{T}$.

Once the essential matrix is computed, it can be decomposed to yield the relative orientation of the two views, i.e., the rotation R and the direction of translation $\mathbf{t}$. We achieve this using the method proposed by Horn [23].

For every essential matrix, there are four possible R, $\mathbf{t}$ explanations. The correct one can be determined by triangulating a world point $\mathbf{X}_i$ using its projections $\mathbf{x}_i$, $\mathbf{x}'_i$ and the cameras $\mathsf{P} = [\mathsf{I}|\mathbf{0}]$, $\mathsf{P}' = [\mathsf{R}|\mathbf{t}]$. In only one of the four solutions will the reconstructed scene point lie in front of both cameras (illustrated in Figure 2.6).

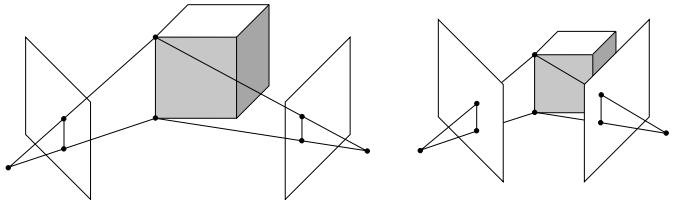


Figure 2.7: *Scale ambiguity of two-view geometry. The same images arise from two cameras far-apart viewing a big scene and two cameras being closer to each other looking at a small object.*

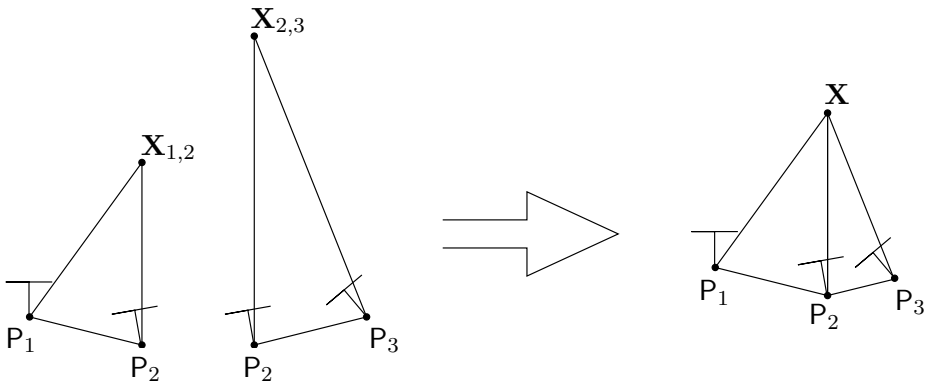


Figure 2.8: *Relative scale of a camera triple may be obtained by the distance of a triangulated scene point.*

2.4.2 Relative orientation of an image triple

To form a camera triple from two consecutive oriented two-view pairs, the relative scale factor between the pairs must be determined. This is because there is an ambiguity of scale in the reconstructed two-view orientation. The problem is intrinsic to multiple view geometry and is illustrated in Figure 2.7. The overall scale of the world (containing both, the scene and the cameras) may be varied, and the images arising will be the same. That is why we can only recover the relative translation direction from epipolar constraints, but not the amount of translation. However, since we know that both image pairs depict the same scene, we can infer the ratio of the translation amounts from the fact that the scene must be equally sized in both pairs. Since consecutive camera pairs $(\mathsf{P}_1, \mathsf{P}_2)$, $(\mathsf{P}_2, \mathsf{P}_3)$ overlap in one camera P_2, it is equivalently required that the respective triangulations of a scene point $\mathbf{X}$ must be at equal distances from P_2.

This results in the simple algorithm illustrated in Figure 2.8. We want

to obtain the ratio k of the Euclidean distances between the camera centres such that $\|\widetilde{\mathbf{C}}_3 - \widetilde{\mathbf{C}}_2\| = k\|\widetilde{\mathbf{C}}_2 - \widetilde{\mathbf{C}}_1\|$.

1. For each of the oriented pairs of cameras $(\mathsf{P}_1, \mathsf{P}_2)$, $(\mathsf{P}_2, \mathsf{P}_3)$ fix the scale factor such that the distance between the respective camera centres is 1.
2. In the fixed frames compute triangulations $\widetilde{\mathbf{X}}_{1,2}$, $\widetilde{\mathbf{X}}_{2,3}$ of scene point $\mathbf{X}$ using image points $\mathbf{x}_1 \leftrightarrow \mathbf{x}_2$, $\mathbf{x}_3 \leftrightarrow \mathbf{x}_3$ respectively.
3. Compute the scale ratio k as

$$k = \frac{\|\widetilde{\mathbf{X}}_{2,3} - \widetilde{\mathbf{C}}_2\|}{\|\widetilde{\mathbf{X}}_{1,2} - \widetilde{\mathbf{C}}_2\|}.$$

Since there is measurement error attached to the locations of the image points, k_i is estimated for every triple of corresponding points. The final k is obtained as the average of the k_i.

Furthermore, some point triples are better suited for triangulation than others. Triangulated scene points are less precisely localized as the backprojected rays become more parallel. Thus, weighting factors w_i are introduced, making k_i less significant if the angles $\alpha^i_{1,2}$, respective $\alpha^i_{2,3}$, between backprojected rays through $\mathbf{x}^i_1$ and $\mathbf{x}^i_2$, respective $\mathbf{x}^i_2$ and $\mathbf{x}^i_3$, are small. The solution for k is then obtained as a weighted average of the individual k_i. In practice, we have found that weighting factors

$$w_i = \tan\frac{\alpha^i_{1,2}}{2} \cdot \tan\frac{\alpha^i_{2,3}}{2} \qquad \text{with } 0 \leq \alpha^i_{1,2}, \alpha^i_{2,3} \leq \frac{\pi}{2}.$$

yield good results.

Actually, a variation of this method introduced in [32] is used. It omits explicitly triangulating scenepoints and yields slightly improved results. For the sake of brevity, it is not described here.

2.4.3 Concatenating image triples

The remaining task is to form a completely oriented sequence of cameras from oriented triples. Consecutive triples overlap in one camera pair. Due to this fact, it is straightforward to concatenate them into a path.

From the two-view stage camera matrices $\mathsf{P}_n = [\mathsf{R}_n|\mathbf{t}_n]$ were obtained, where camera P_n is oriented relative to P_{n-1}, i.e., supposing $\mathsf{P}_{n-1} = [\mathsf{I}|\mathbf{0}]$. The cameras are normalized such that $\|vct_n\| = 1$. The first camera is fixed

as $\mathsf{P}_0 = [\mathsf{I}|\mathbf{0}]$. Furthermore, from the three-view stage scale ratios k_n are obtained, where k_n refers to the scale ratio of triple $(\mathsf{P}_{n-2}, \mathsf{P}_{n-1}, \mathsf{P}_n)$.

First, we calculate the factors l_n denoting the scale ratio between the camera pairs $(\mathsf{P}_{n-1}, \mathsf{P}_n)$ and $(\mathsf{P}_0, \mathsf{P}_1)$. These are obtained as

$$\begin{aligned} l_1 &= 1 \\ l_n &= k_n l_{n-1} \quad \text{for } n \geq 2. \end{aligned}$$

Then, we can obtain the sequence of cameras $\mathsf{P}_n^0 = [\mathsf{R}_n^0|\mathbf{t}_n^0]$, where orientation is in the frame attached to the first camera P_0, as

$$\begin{aligned} \mathsf{R}_0^0 &= \mathsf{I} \\ \mathbf{t}_0^0 &= \mathbf{0} \\ \mathsf{R}_1^0 &= \mathsf{R}_1 \\ \mathbf{t}_1^0 &= \mathbf{t}_1 \\ \mathsf{R}_n^0 &= \mathsf{R}_n \mathsf{R}_{n-1}^0 \quad \text{for } n \geq 2 \\ \mathbf{t}_n^0 &= \mathsf{R}_n \mathbf{t}_{n-1}^0 + l_n \mathbf{t}_n \quad \text{for } n \geq 2 \end{aligned}$$

There is still an overall scale ambiguity for the resulting camera sequence. Scale can be fixed by additional knowledge about metric distances, either between known points in the scene or between individual camera positions.

A major advantage of the method is speed: Complexity of adding a new view to a sequence is $O(n^2)$ given n corresponding points between the previous view and the new one.

A drawback lies in error propagation and accumulation over time, as the exemplary results in Figure 2.9 show. Like with odometry sensors, rotational and translational error is accumulated. Furthermore, a drift in relative scale occurs.

2.5 Bundle Adjustment

We have evaluated our method by comparing it with the results of the ORIENT software [25]. ORIENT is a commercial implementation of bundle adjustment, developed at the TU Wien.

Bundle adjustment is a photogrammetric technique for reconstructing 3D information from (image) measurements. The name refers to the "bundles" of light rays leaving each 3D feature and converging on each camera centre, which are "adjusted" optimally with respect to both feature and camera positions. Bundle adjustment is optimal in the sense of providing a maximum likelihood estimate of both 3D structure and viewing parameters. In contrast to the method described in this paper, *all* available data is used at once, i.e., all feature measurements from all images, resulting in a large sparse geometric parameter estimation problem.

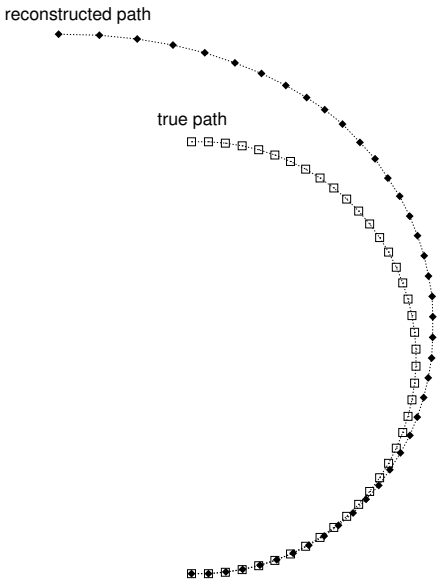


Figure 2.9: *Experimental results about "sequential alignment of image triples" were obtained in a simulation. Original and reconstructed camera paths are shown from a birds-eye perspective (camera movement was parallel to the ground plane). Reconstruction is from projections of the simulated scene to a* 800×600 *pixel image plane. Artifical measurement noise was added to the individual image points, drawn from a uniform distribution of* 3×3 *pixels.*

In principle, bundle adjustment finds the solution to the following problem: Given measurements $\mathbf{x}_j^i$, i.e., measurement of feature j in image i, we want to estimate camera matrices $\widehat{\mathsf{P}}^i$ and 3D features $\widehat{\mathbf{X}}_j$ which project to image features $\widehat{\mathbf{x}}_j^i = \widehat{\mathsf{P}}^i\widehat{\mathbf{X}}_j$ such that the sum of squared distances between measurements $\mathbf{x}_j^i$ and reprojections $\widehat{\mathbf{x}}_j^i$ is minimized. Not only point features can be incorporated but all kinds of measurements. The solution is found by nonlinear optimization. A good estimate is needed to initialize the computation. This initial estimate is then iteratively refined. In each iteration step, a linear approximation of the error function centered on the current estimate is made. Depending on the minimization method, first or second order derivatives are needed to compute an improved estimate. Obtaining the derivatives is computaionally expensive since it involves decomposition and inversion of large matrices. However, by exploiting the sparseness of the problem, complexity can be significantly reduced.

Bundle adjustment methods have a long history in photogrammetry[7] and are constantly gaining followers in the computer vision community, too. Triggs et al. [39] give a survey of bundle adjustment theory and methods aimed at computer vision researchers. A good recent photogrammetric textbook is [31].

2.6 System architecture

The control architecture of the robot comprises several modules. Some tasks are executed on the robot itself, other components are offboard on an external PC. The modules talk to each other via interprocess communication over wireless LAN.

Figure 2.10 shows the setup that we used for experimentation. The task of the modules on the lefthand side is to make the robot execute some predefined movements. The modules on the righthand side try to estimate the robots pose from sensor readings using the methods described later in this paper.

The robot, a Pioneer 2 by ActivMedia[8], is controlled by a microcontroller that runs a special embedded operating system called P2OS. The Saphira client running on the host computer issues motion commands to P2OS and receives sonar and odometry sensor readings.

For our experiments, the robot was required to travel along a predefined path, which should then be recovered from the images aquired by the robots camera as it moves. The *path planner* is provided a 2-dimensional map of the environment by the *map module* and a sequence of waypoints therein.

[7] The basic photogrammetric bundle method was developed for the U. S. Airforce by D. C. Brown in 1957 - 1959.

[8] ActivMedia Robotics, http://www.activmedia.com

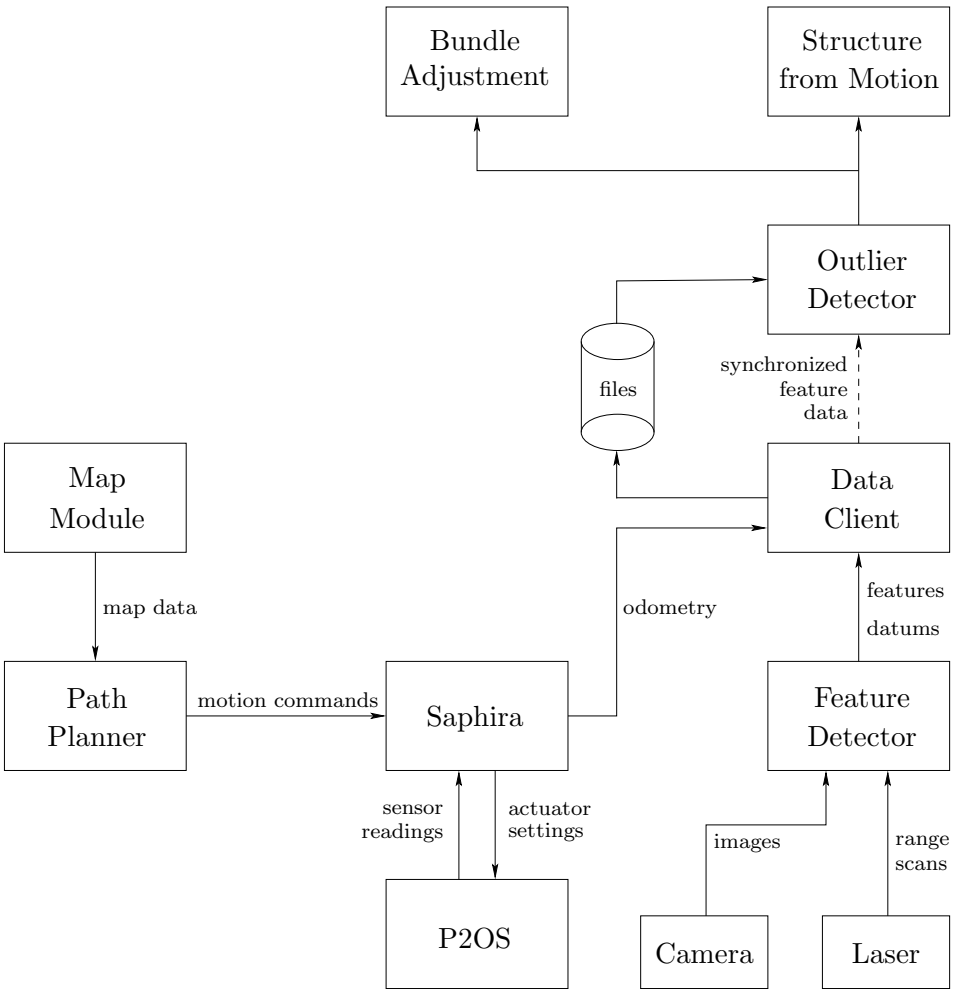


Figure 2.10: *Control architecture of the robot for visual navigation experiments.*

The path planner issues direct motion commands [3] to the Saphira client in order to travel from one waypoint to the next.

Besides odometry and sonar sensors, the robot is equipped with a colour CCD camera and a SICK laser range finder. The camera images and the laser range scans are processed by the *feature detector* module which is running on a laptop mounted on the robot. 2D feature points are automatically detected and tracked between the camera images. Furthermore, there are some special landmarks placed at fixed locations in the environment. They are specially marked, so that they can be easily detected in the images by colour-segmentation. The feature detection module is able to find those landmarks both in the images and the laser range scans yielding metric measurements of their positions, so-called datum definitions, which can later be used to fix the overall scale of the reconstructed camera and 3D feature positions.

The *data client* collects feature and datum positions from the feature detection module and synchronizes them with odometry readings from the saphira client. The output of the data client are feature locations for individual frames, each attached with a timestamp and a corresponding odometry estimate. For our experiments the collected data is stored into files and processed offline later. This is done for repeatability and in order to speed up testing cycles. Later, this step might be ommited and the data directly provided to the later stages of position estimation.

The feature data is then passed through the *outlier detector* module where mismatched features are removed from the data. Finally, the *Structure from Motion* and *Bundle adjustment* modules respectively compute position estimates from the data. SFM refers to the methods described in this work. The bundle adjustment module comprises a commercial photogrammetric software [25] whose results are used as a reference for evaluating our method.

Chapter 3

From Images to Point Features

Feature selection and tracking is a fundamental problem in computer vision research. Most structure-from-motion methods [21, 7, 43] simplify the real world by representing it as a set of points, perhaps line segments. A lot of information available from the original images is discarded which of course may introduce ambiguities and lead to less accurate solutions. On the other hand, the complexity of the reconstruction procedure is greatly reduced.

To accomplish good results with point-based methods, it is crucial to solve the correspondence problem, that is, identify features that correspond to points in the physical world and match them across images.

Promising feature windows can be selected based on a measure of texturedness or corderedness. Examples of corner detection or interest point operators are Harris[20], Förstner [16] and Kitchen and Rosenfeld [26]. However, corners and textured regions in the image do not necessarily correspond to fixed points in the world. For example, a textured region might actually correspond to a highlight on a glossy surface or an apparent intersection of two lines that are skew in space.

Once features have been identified, correspondence across images must be established. When feature displacement is small, for instance between frames of a video sequence (aquired and sampled at a sufficiently high time frequency), this may be achieved by tracking features, searching a small region around their previous location.

Our method of choice is the Kanade-Lucas-Tomasi (KLT) Tracker [33]. It is widely used (for instance [13, 11, 24, 2, 1]) and seemed to be suited well for our setup. Stan Birchfield has made available an implementation [5] which we used as a starting point of our own implementation.

This chapter proceeds as follows: The basic ideas behind the KLT tracker are explained in Section 3.1. Section 3.2 deals with some practical issues like image pyramids and smoothing, and briefly reviews how they are solved

in Stan Birchfield's implementation. Section 3.3 explains how the original code was modified to improve time performance. Finally, in Section 3.4 we presents results of our implementation on real images and compare the performance of our implementation with Stan Birchfield's implementation.

3.1 Kanade-Lucas-Tomasi feature tracker

The origins of the Kanade-Lucas-Tomasi Tracker go back to the work of Lucas and Kanade [30]. Though the most readily accessible description is the paper by Shi and Tomasi [33], an earlier paper by Tomasi and Kanade [35] already contained a fully developed version. They introduced a way to select features that is explicitly based on the tracking equation. Their intention is to select those features that make the tracker work best. They also proposed using an affine model of image motion to monitor feature dissimilarity between the first and the current frame. Subsequently, Tommasini et al. [36] proposed a scheme for selecting a dissimilarity threshold for feature rejection automatically.

Due to the long history of usage, the method it is referred to by various combinations of its authors names, including Kanade-Lucas-Tomasi [6], Lucas-Kanade [9], Shi-Tomasi-Kanade [36] and Tomasi-Kanade Tracker [4].

The remainder of this section consists of the derivation of the tracking equation and a description of the feature selection method. The notation has been adapted from [35].

3.1.1 The KLT tracking equation

Most of the time, it is impossible to determine where a single pixel went in the subsequent frame, based only on local information. Because of this, small *windows* of pixels are used as features. The goal of tracking is to determine the *displacement* $\mathbf{d}$ of a feature window from one frame to the next.

The displacement is chosen as to minimze the *dissimilarity* between two feature windows, one in image I and one in image J:

$$\epsilon = \int\int_W [J(\mathbf{x}+\mathbf{d}) - I(\mathbf{x})]^2 w(\mathbf{x}) d\mathbf{x} \tag{3.1}$$

where W is the given feature window, $\mathbf{x} = [x, y]^\top$ are coordinates in the image and $\mathbf{d} = [d_x, d_y]^\top$ is the displacement. The weighting function $w(\mathbf{x})$ is usually set to the constant 1.

We try find the displacement $\mathbf{d}$ that minimizes the dissimilarity. To do this, we differentiate Equation (3.1) with respect to $\mathbf{d}$ and set the result to zero.

$$\frac{\partial \epsilon}{\partial \mathbf{d}} = 2\int\int_W [J(\mathbf{x}+\mathbf{d}) - I(\mathbf{x})] \frac{\partial J(\mathbf{x}+\mathbf{d})}{\partial \mathbf{d}} w(\mathbf{x}) d\mathbf{x} = 0 \tag{3.2}$$

Using the Taylor series expansion of J about $\mathbf{x}$, truncated to the linear term, we obtain

$$J(\mathbf{x}+\mathbf{d}) \approx J(\mathbf{x}) + d_x \frac{\partial J(\mathbf{x})}{\partial x} + d_y \frac{\partial J(\mathbf{x})}{\partial y}.$$

Plugging this into Equation (3.2) yields

$$\frac{\partial \epsilon}{\partial \mathbf{d}} = 2 \int \int_W [J(\mathbf{x}) - I(\mathbf{x}) + \mathbf{g}(\mathbf{x})^\top \mathbf{d}] \mathbf{g}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} = 0,$$

where

$$\mathbf{g} = \begin{bmatrix} \frac{\partial}{\partial x} J \\ \frac{\partial}{\partial y} J \end{bmatrix}.$$

Rearranging terms yields a linear 2×2 system

$$\mathsf{Z}\mathbf{d} = \mathbf{e}, \tag{3.3}$$

where Z is the 2×2 matrix

$$\mathsf{Z} = \int \int_W \mathbf{g}(\mathbf{x}) \mathbf{g}(\mathbf{x})^\top w(\mathbf{x}) d\mathbf{x} \tag{3.4}$$

and $\mathbf{e}$ is the 2-vector

$$\mathbf{e} = \int \int_W [I(\mathbf{x}) - J(\mathbf{x})] \mathbf{g}(\mathbf{x}) w(\mathbf{x}) d\mathbf{x} \tag{3.5}$$

Equation (3.3) is only approximately satisfied, because of the linearization of Equation (3.2). However, the correct displacement can be found by iterating on Equation (3.3) in a Newton-Raphson style minimization.

Our implementation uses a slightly different equation based on the derivation in [6]. By replacing $[J(\mathbf{x}+\mathbf{d}) - I(\mathbf{x})]$ with $[J(\mathbf{x}+\frac{\mathbf{d}}{2}) - I(\mathbf{x}-\frac{\mathbf{d}}{2})]$, the computation is made symmetric with respect to both images. The original version was shown here because in this form it is also applied to feature selection (discussed in the following section).

3.1.2 Selecting good features

An arbitrary feature window does not necessarily contain complete motion information. For a horizontal intensity edge only the vertical motion component can be determined, for instance. Even worse, a feature window inside a homogeneous region contains no motion information at all. This is a fundamental problem, regardless of the selected method of tracking.

In order to avoid this, it is necessary to choose feature windows carefully. Several "interest operators" have been proposed based on intuitive ideas of what good features should look like. Shi and Tomasi [33] propose a more principled criterion that is optimal by construction: "A good feature is one that can be tracked well."

In order to track a feature it is necessary that Equation (3.3) can be solved reliably. A solution involves inverting the matrix Z. Note, that Z can be computed from a single image. To obtain a good solution, both eigenvalues of Z must be large. Two small eigenvalues correspond to a roughly constant intensity within the window. A large and a small eigenvalue correspond to a unidirectional pattern. Two large eigenvalues represent reliably trackable patterns, for instance corners or salt-and-pepper textures.

In practice, there is a bound on the maximum eigenvalue since the intensity change between pixels cannot become arbitrarily large. Thus, the matrix Z is usually well-conditioned if the smaller eigenvalue is sufficiently large to be above the image noise level. If λ_1 and λ_2 are the eigenvalues of Z, a window is chosen as a feature candidate if

$$\min(\lambda_1, \lambda_2) > threshold, \tag{3.6}$$

where *threshold* is predefined. In general, the greater $\min(\lambda_1, \lambda_2)$, the better a window is suited. The actual features are chosen from among the candidates. For instance, the highest ranking n candidates are chosen [5], or windows whose rank is a local maximum [9].

3.2 Implementation issues

In the implementation of the tracker, besides putting the tracking and selecting equations to use, some practical issues arise, such as image pyramids, smoothing and interpolation for subpixel accuracy. This section proceeds with a description of these problems and their solutions in Stan Birchfield's implementation. Finally, some further ideas are listed which are subject to further implementations.

3.2.1 Pyramidal implementation

When choosing the feature window size, there is a trade-off between accuracy and robustness. The accuracy component relates to the local sub-pixel accuracy attached to tracking. Displacement of individual points in the integration window may vary, especially at occluding boundaries. the one hand, a small window size is preferable in order not to "smooth out" too much detail. A smaller size makes it less likely, that displacements vary within the window too much. The robustness component, on the other hand, relates to sensitivity of tracking with respect to size of image motion, lighting changes, and so on. For the tracking equation to work, we need to have the displacement smaller than the integration window size. Therefore, to ensure robustness, especially when dealing with large image motion, the window should be chosen as large as possible.

A solution to this dilemma is to use a pyramidal representation of the image. The lowest level of the pyramid is the original image, while higher

levels contain smaller recursively subsampled versions. Features are tracked at the highest level of the pyramid, first, to arrive at an approximate solution. The displacement is then promoted down to the next level where tracking is continued to improve on the estimate. This is done in a recursive fashion until the lowest level is reached. This effectively enables us to deal with large displacements on top of the pyramid, while maintaining sub-pixel accuracy at the bottom.

The resolution of the topmost pyramid level should be selected according to the window size and the maximum expected displacement. The number of pyramid levels may be chosen empirically. In general, two pyramid levels should suffice. The subsampling factor may be computed accordingly. For instance, by default Birchfield's implementation uses 7×7 feature windows and two pyramid levels subsampled at every fourth pixel. This yields good results for displacements up to ca. 15 pixels.

3.2.2 Image smoothing

Image smoothing is a so called low-pass filtering operation – low pass in the sense of passing low spatial frequencies and rejecting high spatial frequencies. One of the most popular operations of this type is the Gaussian blur filter, which has the advantage of being circularly symmetric, so that edges and lines are treated similarly in all directions.

Smoothing occurs in the implementation for two purposes. First, it is obviously useful for anti-aliasing when building the image pyramid. Applying a Gaussian blur operator before subsampling ensures that the subsampled pixels accurately represent "their regions" in the higher resolution image. Second, smoothing is useful as a preprocessing step when dealing with noisy images. Being usually determined by random processes during image aquisition, noise mostly occurs at high spatial and temporal frequencies. Blurring the images beforehand smooths out high frequency noise which might be mistaken for valid features.

Birchfield uses Gaussian filters for both purposes. The standard deviation for the Gaussians default to $\sigma = 0.1 \cdot \mathit{imagesize}$ for the preprocessing step and $\sigma = 0.9 \cdot \mathit{subsampling}$ for building the image pyramid.

3.2.3 Subpixel computation

It is desirable to keep all computations at a subpixel accuracy level. For instance, evaluating the tracking Equation (3.3) will not yield integer values for the displacement $\mathbf{d}$, in general. It is therefore necessary to be able to compute intensity values for locations between pixels. Birchfield implemented bilinear interpolation to solve this. Bouguet [9] suggested to use this, too.

The principle of bilinear interpolation is illustrated in Figure 3.1. To

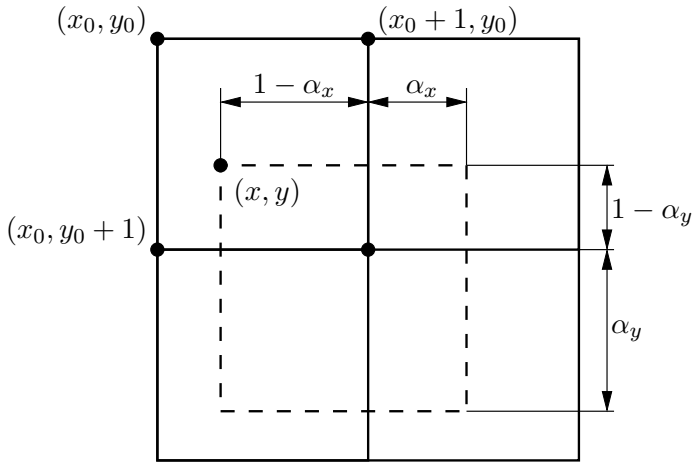


Figure 3.1: *Bilinear interpolation.*

obtain the image value at location $\mathbf{x} = [x, y]^\top$, x and y are decomposed into their respective integer parts x_0, y_0 and remainders α_x, α_y.

$$x = x_0 + \alpha_x$$
$$y = y_0 + \alpha_y$$

The image value is computed then as a weighted average of the four pixels that intersect the "virtual" pixel located at $\mathbf{x}$:

$$\begin{aligned} I(x, y) =& (1 - \alpha_x)(1 - \alpha_y) I(x_0, y_0) + \alpha_x (1 - \alpha_y) I(x_0 + 1, y_0) + \\ & (1 - \alpha_x) \alpha_y I(x_0, y_0 + 1) + \alpha_x \alpha_y I(x_0 + 1, y_0 + 1). \end{aligned}$$

Note, that the weighting factors represent the sizes of the overlapping areas between the virtual pixel and the four contributing pixels.

3.2.4 Feature selection

To select features from among the candidates, Birchfield uses the following approach. The eigenvalues of Z are evaluated at every pixel of the input image, and pixels are then sorted by $\min(\lambda_1, \lambda_2)$ in descending order. Starting with the highest-ranking pixel, each pixel is added to the set of features until a user-defined number of features has been selected, or the criterion $\min(\lambda_1, \lambda_2) > \mathit{threshold}$ is no longer met. For every new candidate it is checked whether it lies within a threshold distance to any of the features already selected. If so, the candidate is rejected. The threshold distance defaults to 10 pixels. The threshold on the eigenvalues defaults to 1.

3.2.5 Unimplemented features

Finally we want to briefly list ideas, that are as yet not contained in the implementation, but might be considered to improve the tracker.

- **Tracking features close to the image boundary.** Features are declared lost if they have a portion of their integration window outside the image. Because with every pyramid level the effective window size increases, there might be a rather large "forbidden band" at the image borders. In order to prevent this, Bouguet [9] proposed to perform the summations in the expressions for Z and $\mathbf{e}$ (see Section 3.1.1) only over the valid portion of the neighborhood.

- **Monitoring feature quality.** Shi and Tomasi [33] proposed to evaluate the consistency of features between non-consecutive frames using an affine model of dissimilarity. A dissimilarity threshold for feature rejection can be computed automatically [36].

3.3 Optimizing for speed

The feature tracker shall be employed on the robot, processing images as they arrive, in real-time. It is crucial to have an implementation as efficient as possible for two reasons: Firstly, potential feature displacement increases with the time passing between subsequent frames. There is a threshold on the displacement that the tracker can deal with. (Since feature tracking is an iterative procedure starting from the previous feature location as an initial guess, tracking will fail if the new feature location is outside of the feature window size.) Thus, the higher the frame rate at which the tracker operates, the better it is expected to work. Secondly, in the final position estimation framework feature tracking is only a preprocessing step. We will want to leave as much as possible processing capability to other tasks.

Birchfield's code is designed to be readable, with educational purposes in mind. We have found, that there is plenty room for optimization without sacrificing accuracy. This section gives a short summary of the changes that we made to increase processing speed.

3.3.1 General approach

The original code is designed to be readable and flexible, rather than fast. A lot of redundant computation can be removed when the straight-forward manner of putting concepts to work is abandoned. For instance, the summation of values over feature windows was originally implemented as a brute force summation over the neighborhood. When selecting feature candidates, this has to be done for every single pixel of the image. Using 7×7 windows,

this means every pixel has to be read and processed 49 times. When a shifting summation window is used instead, every pixel has to be touched only twice, one time when it enters the window and another time when it leaves.

We have remedied such situations wherever appropriate. In particular, we have applied a code generation approach. The parameters of the tracker are set at compile time and the actual tracking code is generated from templates. Thus, it possible to employ smoothing and gradient filters that are especially tailored to the parameters and more efficient than generic ones. In the following, we comment on some particular improvements that were made. Finally, we present some results on real images to quantify the gain in performance.

3.3.2 Efficient blur and gradient filters

A large part of the processing time is spent with the application of Gaussian blur and Laplacian of Gaussian (LoG) filters in order to create the pyramidal representation of the intensity and gradient images. Actually, in the original implementation these operations comprise about 70 percent of processing time. By using efficient implementations, processing time can be dramatically reduced.

Efficient Gaussian blur. Two-dimensional Gaussian blur operations are rather costly when carried out in a brute force way, especially where large kernels are involved. We employed an efficient implementation using the SKIPSM (Separated Kernel Image Processing using finite State Machines) paradigm [40]. To significantly reduce the execution time of the operation two properties of Gaussian blurs are put to use:

- a 2-Dimensional Gaussian blur operation is separable into independent row and column operations,
- large kernels can be decomposed into sequential application of small kernels.

Application of the first property alone reduces processing time from $O(nw^2)$ to $O(2nw)$, where n is the number of pixels and w the kernel width. Birchfield's code already put this to use.

Usually, the coefficients of the kernels are based on Pascal's triangle (the binomial coefficients – these approach the Gaussian curve more and more closely as the number of values increases). A property of Pascal's triangle is that each entry can be computed as the sum of the two entries diagonally above it. This reflects in the second property stated above: convolving to times with $\begin{vmatrix}1 & 1\end{vmatrix}$ (second line of Pascal's triangle) is equivalent to convolving with $\begin{vmatrix}1 & 2 & 1\end{vmatrix}$ (third line). Figure 3.2 shows how this is used to efficiently implement a Gaussian kernel of width 5. In contrast to the brute

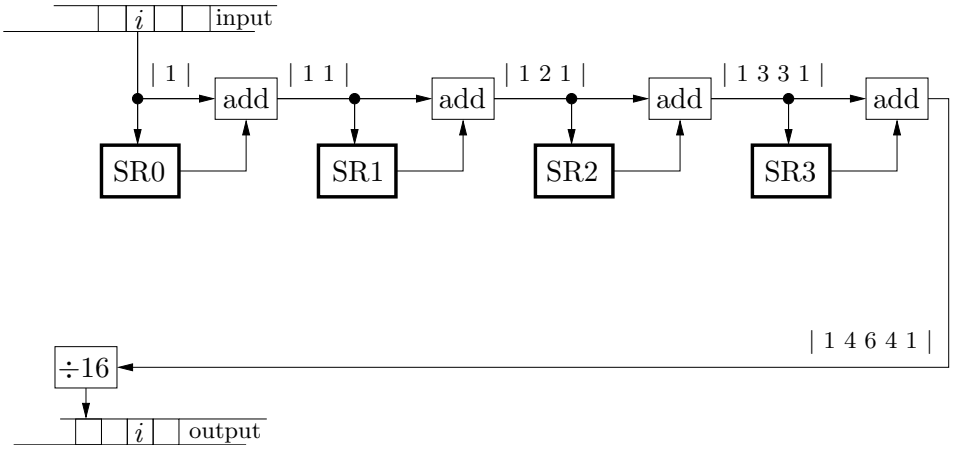


Figure 3.2: *SKIPSM implementation of a Gaussian blur operator. The "SRi" components are simple registers, each capable of storing a single integer value. The "add" and "÷16" components perform the obvious operations.*

force implementation, each pixel is read only once and no multiplication operations are needed at all. The same number of additions are required for both approaches.

Gradient computation. The horizontal and vertical gradient images are computed by convolution with Laplacian of Gaussian kernels. Those are separable into row and comlumn operations, as well. In this case, depending on the gradient direction, one of the 1-dimensional operations will be a Gaussian and the other a LoG operator. For the Gaussian operation the SKIPSM implementation is used, too.

Image pyramids. Blurring is also used before creating subsampled images for higher pyramid levels. With default parameters, the original implementation employs 21×21 Gaussian operators which is *very* costly, even using the SKIPSM implementation.

Those were replaced by simple averaging operations (11×11 by default). In comparison to the Gaussian, the resulting blur is not circularly symmetric, and also not as "smooth". This leads to slightly reduced quality of the subsampled images. Tracking accuracy may suffer at higher pyramid levels. Intuitively, this should not pose a problem since the final result is refined at the lowest level (which is of high quality).

Applying horizontal operations only. It turns out that applying horizontal kernels is faster than applying vertical kernels. The exact reason

for this is beyond our knowledge, possibly it is due to caching being more efficient on sequentially ordered data.

The best results were obtained by applying horizontal operations only and "transposing" the output in the process (reversing row and column indices). As an example, Figure 3.3 shows the operations necessary to compute the coefficients for feature selection. Image pyramids are built in a similar fashion.

3.3.3 Feature selection

Two modifications were made to the feature selection procedure. The first is concerned with the "lazy evaluation" of the eigenvalue conditions on the matrix Z – many unpromising feature candidates may be discarded early in the evaluation procedure. The second modification deals with the sorting of the candidate list. The sorting procedure can be prematurely terminated when the desired number of features has been found.

When checking for potential feature candidates, the following equation must be evaluated at every pixel:

$$\frac{g_{xx} + g_{yy} - \sqrt{(g_{xx} - gyy)^2 + 4g_{xy}^2}}{2} > ev \tag{3.7}$$

with

$$g_{xx} = \int\int_W (\frac{\partial I}{\partial x})^2$$
$$g_{yy} = \int\int_W (\frac{\partial I}{\partial y})^2$$
$$g_{xy} = \int\int_W \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$$

The left-hand side of (3.7) represents the smaller eigenvalue of the matrix Z (Equation (3.4)) on which ev is a threshold.

For Equation (3.7) to be fulfilled, the relaxed constraints

$$g_{xx} > ev$$

and

$$g_{yy} > ev$$

must hold, too. Being computationally cheaper operations, these conditions are evaluated first – failing pixels are discarded immediately.

The second modification concerns the sorting of the candidate list. There may be much more feature candidates than are actually needed. To avoid unnecessary sorting of lower-ranking pixels, an incremental version of quicksort is used which iterates on the partially sorted sublists only as long as there are more features required.

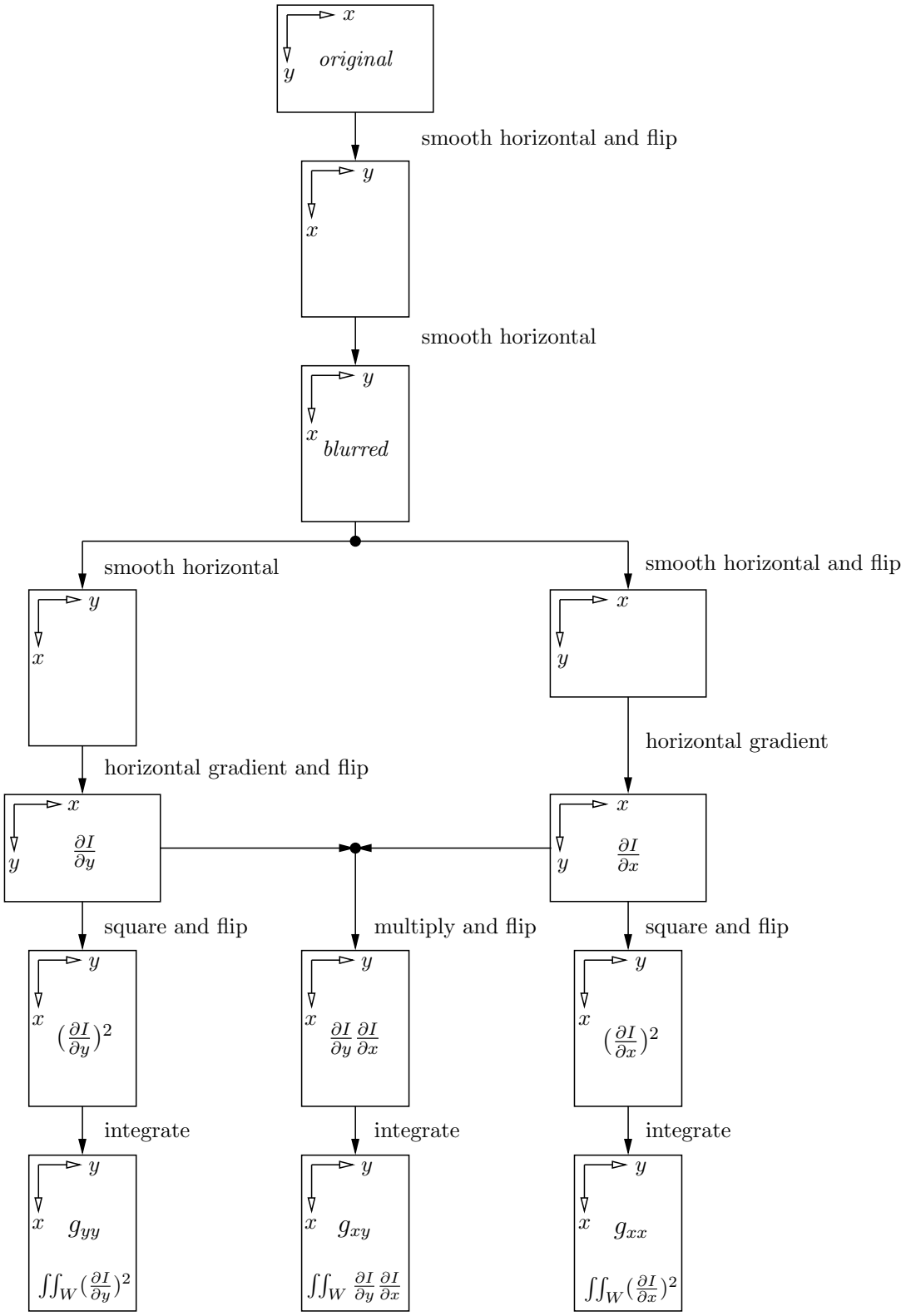


Figure 3.3: *Efficiently calculating entries of* Z *for feature selection.*

3.4 Experimental results

We have tested our implementation on real images. A goal of the experiments was to evaluate the benefits of our optimizations. We evaluated results of the original and optimized version on identical images. Performance was measured for

- selecting 400 features in a 640×480 image and
- tracking those 400 features in the next image and replacing lost features.

The test images are shown in Figures 3.4 and 3.5. The found respective tracked features are overlaid.

It is difficult to compare the implementations in terms of tracking quality. Visually judged, the result were identical. Comparison of the execution times indicates that our implementation is significantly more efficient than the original one. The following table shows the execution times of the implementations on an AMD Athlon XP 1800+ running at 1.6 GHz.

	original	**optimized**
selecting 400 features	610 ms	72 ms
tracking 400 features, replacing lost ones	855 ms	166 ms

The optimized version is more than 5 times faster than the original one, which is a satisfying result.

Figure 3.4: *A* 640×480 *image aquired by the robot's CCD camera. The locations of the features that were found are indicated by filled white rectangles.*

Figure 3.5: *The features that were selected in Figure 3.4 were tracked in the next image of the sequence. Filled white rectangles are placed at the current feature positions. The white lines indicate from which locations in the previous image the features were tracked.*

Chapter 4

Dealing With Outliers

We want to use the point correspondences established by the feature tracker to estimate the position of the robot. An intermediate step is to compute the relative orientation between pairs of images. This is achieved by estimating the essential matrix E using the normalized 8-Point algorithm. In the application of the 8-Point algorithm it is implicitly assumed, that the only source of error is in the measurement of the point positions. This error is commonly modeled as a Gaussian distribution. However, in practice there is also the problem of false feature correspondences, in the following referred to as *outliers*.

The focus of this chapter is on *random sampling* methods, a group of robust estimators that has proven efficient for handling putative correspondence data [21, 38, 37, 43]. More general accounts of various robust techniques in computer vision can be found in [34, 42]. Another possibility of robustification is monitoring feature quality in the KLT tracker [33]. However, situations like the one in Figure 4.2*(a)* will not be detected since feature qualitity is defined in a two-dimensional context. Nevertheless, feature monitoring can be a useful preprocessing step, reducing the number of outliers before the methods described in this chapter are applied.

In Section 4.1 we describe common sources of error which cause outliers in the results of feature tracking. We discuss the notion of robustness. Section 4.2 introduces the RAndom SAmpling Consensus (RANSAC) algorithm, a general robust estimation method, which can be used to detect outliers. Section 4.3 deals with alternative random sampling methods, basically variations of RANSAC. In particular we describe the M-estimator SAmpling Consensus (MSAC) algorithm, which is the method we chose to implement, as described in Section 4.4.

4.1 Outliers and Robustness

Figure 4.1 shows some erroneous features that were tracked by the Kanade-Lucas-Tomasi tracker. These features are not correctly modeled by the Gaussian error model for several reasons. Firstly, there are features which are perfectly valid in the two-dimensional image but do not correspond to fixed positions in the three-dimensional scene. Figure 4.2(a) shows a magnification of detail *a* from Figure 4.1. The respective image region is shown for the previous (left) and the current image (right). The features shown are located at an occlusion boundary. One part of the feature is formed by the box in the foreground and the other part by an object on the wall in the background. The apparent "corner" is not fixed to a particular location in space.

Secondly, the featuretracker is not perfect, occasionally there will be mismatched points when the tracker drifts away from its original target to a "similar-looking" feature. Figure 4.2(b) shows such a situation. In this case, two textures on the box with similar appearance are confused. Image motion is big enough to cause the feature tracker to converge to the wrong local minimum.

Finally, we implicitly assume a rigid scene where the only moving object is the camera. Features corresponding to moving objects in the field of view, although correctly tracked, will severely disturb the estimation process.

Such features which are not correctly represented by the motion or error model are labeled *outliers*. The goal of robust estimation is to determine a set of *inliers* from partially corrupted data, feature coordinates in our case. The essential matrix can then be estimated using only the inliers.

A common measure to define robustness more formally is the *breakdown point*, which is the minimum fraction of outlying data that can cause an estimate to diverge arbitrarily far from the true solution. The breakdown point of least squares methods, such as the 8-Point method for instance, is 0 – one bad point can move the least squares fit arbitrarily far away from the correct solution. Thus, outlier removal is essential for the task we want to accomplish. The theoretical maximum breakdown point is 0.5, since when half of the data are corrupted they can be arranged as to represent an alternate inlier set. In practice however, when the corrupted data are not maliciously arranged to break the computation, robust estimation methods are able to deal with significantly more than 50% outliers.

4.2 The RANdom SAmple Consensus

The RANdom SAmple Consencus (RANSAC) algorithm [15] is a general robust estimator, i.e., a method to fit a model to outlier-contaminated data. It is applicable to a wide variety of estimation tasks. It can be categorized as

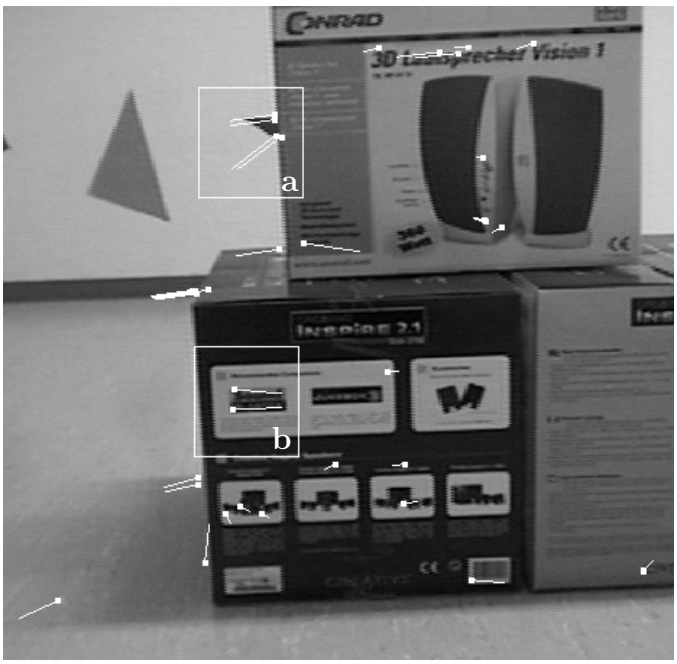
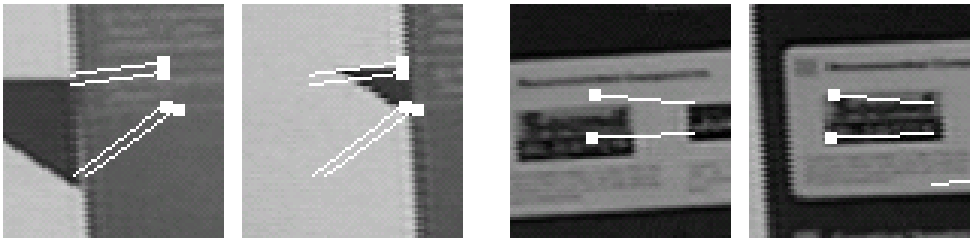


Figure 4.1: *Some tracked 2D features do not correspond to actual 3D points in the scene. Filled white rectangles are placed at the current feature positions. The white lines indicate from which locations in the previous image the features were tracked.*



(a) (b)

Figure 4.2: *Magnified details a and b from Figure 4.1, for the last and the current image respective.*

a hypothesis and verify algorithm. In computer vision, RANSAC has been used successfully for the estimation of homographies, fundamental matrices, and trifocal tensors. Following the explanations in [21], the basic idea of RANSAC will be illustrated now using a simple example.

4.2.1 The basic idea

Suppose a set of 2-dimensional points is measured, and we want to estimate a straight line, $y = ax + b$, that fits the points best. There is a measurement error in the coordinates, following a Gaussian probability distribution. Furthermore, some of the points are outliers, they do not belong to the line. The task is to classify the points into inliers and outliers. Then, the line can be estimated from the inliers (by minimizing the sum of squared perpendicular distances to the line).

RANSAC proceeds as follows:

1. *Randomly select a minimal sample.* In our case, two points are sufficient to define a line. So, two points are randomly selected and a line is fit through them.

2. *Measure the support for the sample.* For every data point the distance $d_\perp$ to the hypothesis is calculated. The points for which this distance is below a threshold T form the *consensus set*. The support for the sample is the cardinality of the consensus set, e.g., the number of points which agree with the hypothesis.

   The distance function should be the one which will be minimized when computing the final estimate from the inliers, in our case the squared perpendicular distance to the line. The threshold is predefined, according to the expected measurement noise.

This process is repeated a number of times, and the hypothesis with maximum support is chosen. Points which support it are labeled as inliers, points which don't are labeled as outliers.

Intuitively, it is expected that a sample containing outliers will not be supported by many points. Furthermore, for samples that comprise only inliers, the closer they approximate the final fit, the more likely it is that they gain much support. For instance, the sample comprising *(c, d)*, in the lefthand side of Figure 4.3, will get less support than the sample *(a, b)*.

4.2.2 Choosing a threshold on the distance

The threshold T on the distance function is usually chosen empirically. Sometimes this is the only option, because the parameters of the error distribution are unknown.

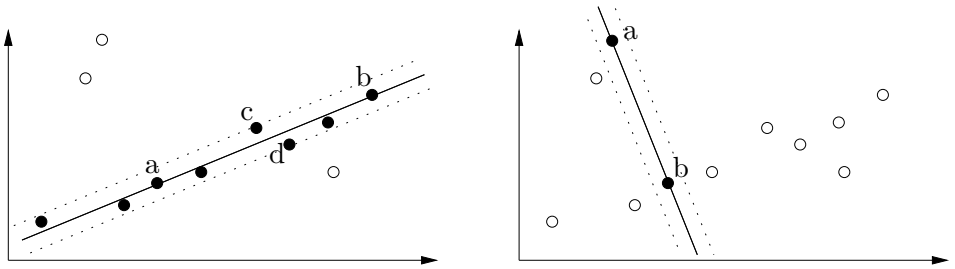


Figure 4.3: *Two hypothesis for the line estimation example. The points labeled a and b respectively comprise the sample. Filled points support the sample. The dotted lines represent the distance threshold. The hypothesis on the lefthand side would be prefered because it has more support.*

However, Hartley and Zisserman [21] argue, that for Gaussian measurement error with zero mean, the squared perpendicular distance to the solution follows a χ^2 distribution. When the standard deviation σ of measurement error is known, a threshold T may be computed, such that inliers will be correctly classified with probability α (with respect to the true model parameters). For details see [21, *p. 102f*].

4.2.3 Determining how many samples to take

An important question is: How many samples should be tried to obtain a good solution? Of course, to make sure the best possible hypothesis is found, the space of samples can be searched exhaustively. However, this is often computationally infeasible. Instead, one can choose a number of samples N such that with probability p at least one of the samples comprises solely inliers.

Suppose w is the probability of selecting an inlier. The probability of selecting a sample of s points, such that all of them are inliers, is approximately w^s, given that there are much more than s data points. So, the probability that a sample contains at least one outlier is $(1 - w^s)$, and the probability that each of N samples contains at least one outlier is $(1 - w^s)^N = 1 - p$. Thus, to ensure that with probability p at least one sample of inliers is selected,

$$N = \frac{\log(1-p)}{\log(1-w^s)} \tag{4.1}$$

samples have to be taken at least. Note, that the total number of points has no influence on the number of samples to take.

Returning to the line fit example: If there are 50% outliers and 17 two-point samples are taken, there is a 99% probability that one of the samples comprises two inliers (no matter how much data points there are). However,

since even two inliers can make for a bad hypothesis, as pointed out in Section 4.2.1, it is never a bad idea to take more samples.

Often, the fraction of inliers w will be unknown. In such cases, the necessary number of samples, N, can be adjusted adaptively. N is computed starting from a pessimistic guess, 20% inliers for instance. When a sample is found which is supported by more than 20% of the data, it can be inferred that the fraction of inliers is at least that big, and N is updated accordingly. This is repeated until N drops below the number of samples that were actually taken.

4.2.4 Application to finding outliers in tracked features

From the material presented above, some key concepts of the RANSAC algorithm can be identified: A *model* is fit to outlier-contaminated *data*. This is done by computing hypothesis for the models parameters from *minimal samples* of the data, and measuring support by evaluating the *distance* of the data from the model. This section describes the instantiation of these concepts when RANSAC is employed for detecting outliers in the results of the feature tracker.

The data. The data that should be classified into inliers and outliers are the point correspondences computed by the feature tracker.

The model. From the inliers, we want to estimate the relative orientation of cameras, using two frames at a time. No information other than point correspondences will be used. It is assumed that the scene is static, so features attached to moving objects should be rejected as outliers. Thus, the model that is to be fit is the *epipolar geometry* between two frames. The fundamental matrix or essential matrix for the uncalibrated respective calibrated case represent the model parameters.

The sample size. The size of a sample depends on the algorithm used to compute the model parameters.[1] To estimate epipolar geometry, a minimum of 7 point correspondences are needed. If calibrated cameras are used, even 5 points suffice. Unfortunately, in both cases, there need not be a unique solution. For 7 correspondences there may be one or three real solutions. For 5 points as much as 10 real solutions may occur. Each of these solutions has to be checked for support separately. An option is to use the 8-Point algorithm, which yields a unique solution. On the other hand a larger sample size implies that more samples have to be tried (see Equation (4.1)).

[1] The algorithm is also labeled *search engine* in this context.

The distance function. Epipolar geometry constrains corresponding points to lie on their respective epipolar lines. In terms of homogeneous coordinates and the fundamental matrix

$$\mathbf{x}'^{\top}\mathbf{l}' = \mathbf{x}'^{\top}\mathsf{F}\mathbf{x} = 0$$

and

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

The distance function $d_{\perp}$ should measure, how closely a corresponding pair of points satisfies the epipolar geometry.

As a common measure we can use the *Symmetric epipolar distance*:

$$d_{\perp}(\mathbf{x}, \mathbf{x}', \mathsf{F}) = d(\mathbf{x}', \mathsf{F}\mathbf{x})^2 + d(\mathbf{x}, \mathsf{F}^{\top}\mathbf{x}')^2,$$

where $d()$ denotes the perpendicular distance from a point to a line. Symmetric epipolar distance is also known as the *residual error*.

Another common measure is the *geometric error*:

$$d_{\perp}(\mathbf{x}, \mathbf{x}', \mathsf{F}) = d(\mathbf{x}, \hat{\mathbf{x}})^2 + d(\mathbf{x}', \hat{\mathbf{x}}')^2,$$

where $d()$ denotes the Euclidean distance between two points and $\hat{\mathbf{x}}, \hat{\mathbf{x}}'$ is the closest pair of points such that $\hat{\mathbf{x}}'^{\top}\mathsf{F}\hat{\mathbf{x}} = 0$. The *Sampson distance* is a first-order approximation to geometric error. Both measures are more accurate and more expensive to compute than the residual error. A detailed description can be found in [21].

4.3 Alternative random sampling methods

Torr and Zisserman [38] propose two variants of the RANSAC approach, both of which yield improvements of the accuracy of the "winner" hypothesis. They point out that with RANSAC the robust estimate can be very poor if the distance threshold T is set too high. In effect, RANSAC finds the minimum of the cost function

$$C = \sum_i \rho(d_{\perp i}^2)$$

where $\rho()$ is

$$\rho(d_{\perp}^2) = \begin{cases} 0 & \text{if } d_{\perp}^2 < T^2 \\ \text{constant} & \text{if } d_{\perp}^2 \geq T^2 \end{cases},$$

which means inliers score nothing and outliers score a constant penalty. This means that several hypothesis can score the same cost C, although they are

in fact not equally "good". If T is choosen sufficiently large then all data will be inliers and all hypothesis will have the same support.

The first modification, MSAC (M-estimator sample consensus), improves this situation by redefining $\rho()$ as

$$\rho(d_\perp^2) = \begin{cases} d_\perp^2 & \text{if } d_\perp^2 < T^2 \\ T^2 & \text{if } d_\perp^2 \geq T^2 \end{cases}.$$

Now, inliers are scored on how well they fit the hypothesis, while outliers still score a constant penalty.

The second proposed algorithm is labeled MLESAC (maximum likelihood consensus). Here, the error (or distance) for both inliers and outliers is modeled as a mixture of a Gaussian and a uniform distribution:

$$P(d_\perp) = \left(\gamma \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{d_\perp^2}{2\sigma^2}\right) + (1-\gamma)\frac{1}{v}\right)$$

where γ is the mixing parameter, v is a constant describing the uniform outlier distribution, σ is the standart deviation of the error for the Gaussian inlier distribution. For every sample the mixing parameter is estimated using Expectation-Maximization, and hypothesises are scored by their negative log likelihood.

Torr and Zisserman evaluated MSAC and MLESAC for fundamental matrix estimation. Their experiments show that both methods provide a 5% – 10% improvement in accuracy over RANSAC. MLESAC gives slightly better results while being a little more computationally expensive.

Feng and Hung [14] proposed a further improvement on MLESAC. They estimate not only the mixing parameter γ but also σ of the inlier distribution in an iterative algorithm.

The LMS (Least median of squares) method scores hypothesises by the median of the distances to the data. Zhang [43] applies LMS to the estimation of epipolar relations. The advantage is that no threshold needs to be defined, thus no prior knowledge of the variance of the measurement error is necessary. On the other hand, LMS will fail if there are more than 50% outliers since the median distance is then to an outlier.

Zhang [43] also proposes a bucketing technique to make sampling more efficient. He points out that it is a waste of time to evaluate samples whose points are very close to each other, since hypothesis estimation from such samples is highly unstable. To avoid such samples the input region is evenly divided into a number of buckets. A sample of s points is taken by first randomly selecting s mutually different buckets, and then ramdomly choosing one point in each selected buckets. The probability of selecting a particular bucket is linked to the number of points it contains, to ensure that each input point has the same probability to be selected.

4.4 Implementation

The MSAC algorithm was chosen to robustify the results of the KLT feature tracker. This choice is due to the method being easily implementable and yielding acceptable results, both in terms of accuracy and computational efficiency. Symmetric epipolar distance is used as a distance measure. The threshold distance was experimentally chosen to be $T = 1.0$ pixels. The normalized 8-Point algorithm is used as a search engine. Consequently, sample size is 8 point correspondences. The normalized 7-Point algorithm, as described in [21, *p. 264f*], has been implemented, too. However, experimental comparison of both algorithms as search engines yielded better results for the 8-Point method.

Epipolar geometry estimates are more reliable, when the baseline between the frames is large. Most of the time, camera movement between consectutive images is quite small. [2] Thus, to effectively enlarge the baseline features are tracked for several images and MSAC is only applied between "key" frames which are selected using the following heuristic:

- Features have been tracked for at least $N = 4$ input frames since the last key frame, *and*
- at least $F = 10$ features have moved for at least $D = 10$ pixels since the last key frame.

The required number of samples is determined adaptively using Equation (4.1). The starting guess is that there are 50% outliers in the feature tracker data. The number of samples, N, is determined such that a inliers-only sample is selected with 99% probability. Actually, 5 times N samples are taken, to increase the probability that a sample is selected which not only comprises inliers but yields a good hypothesis, too. The inliers found by MSAC are then used to obtain the final estimate for the essential matrix and recover the orientation of consecutive cameras.

Figure 4.4 shows some results obtained for real images. In this example, the feature tracker yielded 214 correspondences. 78 of those were found to be outliers. 855 samples were taken.

[2] This is necessary to ensure good trackability of features. Thus, increasing the time between input images is not an option.

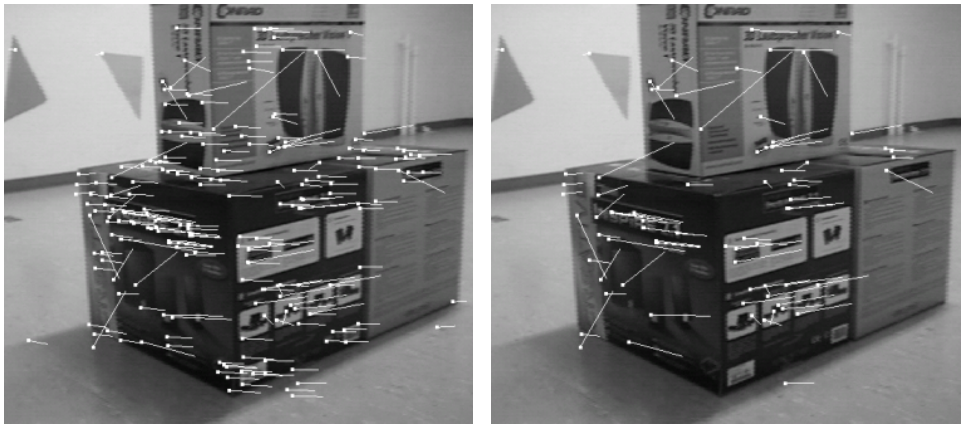


Figure 4.4: *Results of robust estimation. Left: Point correspondences delivered by the KLT feature tracker. Right: Correspondences labeled as outliers by the MSAC algorithm.*

Chapter 5

Using Redundant Information

In Chapters 3 and 4, we have shown how to extract features from images, establish correspondence across images between those, and remove outliers to the model of epipolar geometry. The remaining inliers form the input to our "Sequential alignment of image triples" algorithm, which was reviewed in Section 2.4. The result of the algorithm is a sequence of camera positions and orientations. The reconstruction is based on two-view geometry estimates which will be inaccurate due to measurement noise in the feature locations. Constraints from non-consecutive views are not used explicitly, thus allowing rotational, translational and scale drift error to accumulate over time. This chapter describes how we tried to improve this situation.

Most features are tracked across more than two views. A straightforward way to use this additional information is to estimate epipolar geometry between non-consecutive frames. This yields a redundant set of orientation estimates, e.g., we do not only have estimates for relative orientation between views $(1 \leftrightarrow 2)$, $(2 \leftrightarrow 3)$ and so on but also between views $(1 \leftrightarrow 3)$, $(1 \leftrightarrow 4)$, etc.

These redundant estimates will not be consistent with each other, and the problem is to find a "best fit" set of orientations to this overdetermined system. We have partially adopted Govindu's "consistency models" approach [19, 18] to solve this problem.

This chapter starts with a description of the basic idea of the method and its application to fitting rotations and translations to inconsistent estimates. We then discuss open problems with the approach to translation fitting and propose a solution.

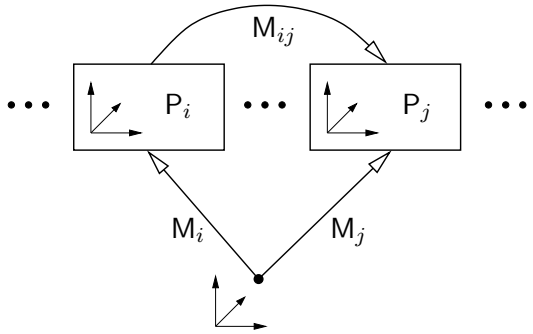


Figure 5.1: *The notion of consistency. Composition of motions* M_i *and* M_{ij} *must be equal to motion* M_j.

5.1 Linear fitting of motion estimates

For a set of N views, there are $\frac{N(N-1)}{2}$ pairs of views, and consequently there could be that many estimates of relative two-view motions. Let M_{ij} denote the motion between views i and j, i.e., the transformation that takes a scene point from the coordinate frame of view i to the coordinate frame of view j.

Given noisy measurements of some of the M_{ij}, we want to estimate the $(N-1)$ independent relative motions between the N views. Equivalently, we can compute the motions M_i, i.e., the transformations from an arbitrarily fixed reference system to the coordinate system of view i. For a consistent system of such motions, it is required that

$$\mathsf{M}_{ij}\mathsf{M}_i = \mathsf{M}_j. \tag{5.1}$$

Moving from the reference frame to view i and then from view i to j should yield the same result as moving from the reference frame to j directly.

Given an inconsistent set of estimates $\widehat{\mathsf{M}}_{ij}$ for the two-view motions, there is no solution for the M_i such that $\widehat{\mathsf{M}}_{ij}\mathsf{M}_i = \mathsf{M}_j$. Instead, we try to find a solution that minimizes the distance between the $\mathsf{M}_{ij} = \mathsf{M}_j\mathsf{M}_i - 1$ and the measurements M_{ij}. Using Equation (5.1), we can formulate an overdetermined system of equations

$$\mathsf{M}_{ij}\mathsf{M}_i - \mathsf{M}_j = 0, \quad \forall i \neq j. \tag{5.2}$$

This can be rewritten as

$$\begin{bmatrix} \dots & \widehat{\mathsf{M}}_{ij} & \dots & -\mathsf{I} & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ \mathsf{M}_i \\ \vdots \\ \mathsf{M}_j \\ \vdots \end{bmatrix} = \mathbf{0},$$

from which a linear least squares solution for the M_i can be obtained.

The intuition is, that by estimating the motions M_i that are most consistent with the measurements, the errors in individual estimates $\widehat{\mathsf{M}}_{ij}$ are "averaged" out. Of course, it is important to choose a suitable representation for the motions, in order to make the "distance" between motions $(\mathsf{M}_i - \mathsf{M}_j)$ a meaningful quantity.

Note, that finding consistent motion estimates can be separated into two subproblems: estimating a set of consistent rotations and estimating a set of consistent translations. The consistency relationship for rotations is

$$\mathsf{R}_{ij}\mathsf{R}_i = \mathsf{R}_j.$$

The consistency relationship for translations is

$$\mathbf{T}_{ij} + \mathbf{T}_i = \mathbf{T}_j.$$

Both relationships must hold and can be solved for independently.

5.2 Estimating rotations

Using rotation matrices, Equations (5.2) would be of the form

$$\widehat{\mathsf{R}}_{ij}\mathsf{R}_i - \mathsf{R}_j = \mathbf{0}.$$

However, a rotation has only 3 degrees of freedom whereas a 3×3 matrix has 9 degrees of freedom. Consequently, there are a number of constraints on the elements of a rotation matrix. In the formulation above, these constraints are not enforced. Thus, the solutions R_i to the above system most likely will not be rotation matrices.

We have to choose another representation for rotations. In [19] it was shown that a formulation using quaternions is optimal in the sense that its linear least squares solution is the Maximum Likelihood estimate given Gaussian distributed rotation error in the $\widehat{\mathsf{R}}_{ij}$.

Representing the rotation estimates as quaternions $\widehat{\mathbf{q}}_{ij} = (w, x, y, z)^\top$ the system of equations becomes

$$\widehat{\mathbf{q}}_{ij}\mathbf{q}_i - \mathbf{q}_j = \mathbf{0},$$

where $\widehat{\mathbf{q}}_{ij}\mathbf{q}_i$ is a quaternion multiplication.
Defining matrices $\widehat{\mathsf{Q}}_{ij}$ corresponding to $\widehat{\mathbf{q}}_{ij} = (w, x, y, z)^\top$ as

$$\widehat{\mathsf{Q}}_{ij} = \begin{bmatrix} w & -x & -y & -z \\ x & w & -z & y \\ y & z & w & -x \\ z & -y & x & w \end{bmatrix}$$

the equations can be reformulated using ordinary matrix multiplication

$$\widehat{\mathsf{Q}}_{ij}\mathbf{q}_i - \mathbf{q}_j = \mathbf{0}.$$

The consistent solution $\mathbf{q}_i$ are obtained by computing a least squares solution to

$$\begin{bmatrix} \dots & \widehat{\mathsf{Q}}_{ij} & \dots & -\mathsf{I} & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ \mathsf{q}_i \\ \vdots \\ \mathsf{q}_j \\ \vdots \end{bmatrix} = \mathbf{0},$$

and then enforcing $\|\mathbf{q}_i\| = 1$ on the individual $\mathbf{q}_i$. For details on quaternion notation see Appendix A.

5.3 Estimating translations

The original paper by Govindu [19] proposes the following approach for estimating consistent translations, which, unfortunately, is not adequate for our purposes for reasons that will be explained later.

As noted earlier, the consistency equations for translations will be of the form $\mathbf{T}_{ij} + \mathbf{T}_i = \mathbf{T}_j$.[1] However, the orientation between two views can only be reconstructed up to a scale factor, i.e., only the directions $\mathbf{t}_{ij}$ of the translations are known. Thus, we arrive at consistency equations of the form $\mathbf{t}_{ij} = \lambda_{ij}(\mathbf{T}_i - \mathbf{T}_j)$ where the λ_{ij} are unknown scale factors.

However, instead of the equality above, we can utilise the fact that $\mathbf{t}_{ij}$ and $(\mathbf{T}_i - \mathbf{T}_j)$ have to be parallel which can be formulated as

$$[\mathbf{t}_{ij}]_\times(\mathbf{T}_i - \mathbf{T}_j) = \mathbf{0}. \tag{5.3}$$

Solving Equation (5.3) will recover the translations (not only the directions) up to a common scale factor then. As Govindu [19] points out, the solution will not be optimal since the equations of the system are unequally

[1] Before this equations can be employed, the translations have to be transformed into a common coordinate system by rotating them according to the previously estimated consistent rotations.

weighted. He proposes a method to scale the individual equations correctly in an iterative procedure.

The problem with the approach above is that it does not work when the centres of all cameras are collinear. All the $\mathbf{t}_{ij}$ will be equal then and obviously Equations (5.3) will be satisfied for any choice of $\mathbf{T}_i = s_i \mathbf{t}_{ij}$, where s_i are arbitrary scaling factors.[2] This is an inherent problem of this configuration. As we pointed out in [32], it is impossible to estimate relative scale in this case *without* using corresponding features across three images explicitly.

Since for our application this configuration is quite likely to occur, i.e., the robot will often travel in a straight line for longer periods of time, this method cannot be applied.

5.4 Re-estimating epipolar geometry with fixed rotation

We have found that we should the approach for estimating consistent translations, because extended sequences of collinear camera positions are likely in the application we aim at. However, the consistenct rotation estimation perfectly works in this situation. It can be applied to estimate a consistent set of rotations from multiple inconsistent two-frame rotation estimates. Furthermore, since the rotation between consecutive views is determined more accurately now, we should be able to recover a better translation estimate as well. This is achieved by using the computed rotation as an additional constraint for translation estimation. Finally, we may expect an improvement in the relative scale between consecutive two-view pairs as well. Scale is determined by the triangulation of scene points. Better estimates for realtive orientation will lead to improved accuracy of triangulation and thus, to more accurate scale recovery.

We adopt the following approach to exploit the redundant rotation information:

1. Estimate the epipolar geometry between two-view pairs using the normalized 8-point algorithm.

2. As described in Section 5.2, estimate a consistent set of rotations using quaternion representation.

3. Keeping this rotations fixed, estimate the translation directions between consecutive views.

[2] Our experiments have shown, that also for configurations that are close to collinearity, the estimation is unstable. We did not implement the iterative reweighting procedure, perhaps this would improve the results. However, inevitably first estimates occur where all the s_i but one are 0. This would result in infinite weights, forbidding unmodified implementation of the reweighting procedure.

4. Using the consistent rotation and the re-estimated translation parameters, find the relative scale between consecutive two-view pairs and concatenate the resulting triples to a complete path (as described in Section 2.4).

Although the estimation of translation and scale does not directly involve redundant information, we may still expect that more accurate rotation estimates lead to improved results in those steps.

We will now give some details on the translation reestimation step. A linear least squares approach, similar to the 8-point algorithm is used. Using the normalized epipolar constraint (Equation 2.3) and the definition of the essential matrix (Equation 2.4), for each pair of corresponding (normalized) points $\widehat{\mathbf{x}} \leftrightarrow \widehat{\mathbf{x}}'$ the constraint

$$\widehat{\mathbf{x}}'^{\top}[\mathbf{t}]_{\times}\mathsf{R}\widehat{\mathbf{x}} = 0$$

must hold. Since we want to fix the known rotation R, we can apply it to $\widehat{\mathbf{x}}$ beforehand, yielding

$$\widehat{\mathbf{x}}'^{\top}[\mathbf{t}]_{\times}\widehat{\mathbf{y}} = 0,$$

where $\widehat{\mathbf{y}} = \mathsf{R}\widehat{\mathbf{x}}$.

Using $\widehat{\mathbf{x}} = (a, b, 1)$ and $\widehat{\mathbf{y}}' = (a', b', 1)$ this is written out as

$$(b' - b)t_1 + (a - a')t_2 + (a'b - b'a)t_3 = 0, \tag{5.4}$$

a linear equation in the elements of $\mathbf{t}$.

Using a set of n correspondences $\widehat{\mathbf{x}}_i \leftrightarrow \widehat{\mathbf{y}}'_i$ a system of linear equations is formed

$$\mathsf{A}\mathbf{t} = \begin{bmatrix} b'_1 - b_1 & a_1 - a'_1 & a'_1 b_1 - b'_1 a_1 \\ \vdots & \vdots & \vdots \\ b'_n - b_n & a_n - a'_n & a'_n b_n - b'_n a_n \end{bmatrix} \mathbf{t} = 0, \tag{5.5}$$

A least squares solution for $\mathbf{t}$ can be found by minimizing $\|\mathsf{A}\mathbf{t}\|$ subject to $\|\mathbf{t}\| = 1$.

In the 8-point algorithm, the solution is overparameterized. We estimate the 8 independent ratios of the elements of a 3×3 matrix which has in fact only 7 (for the fundamental matrix) respective 5 (for the essential matrix) degrees of freedom. However, our translation reestimation method is not overparameterized. The translation direction has 2 degrees of freedom, and we estimate 2 independent ratios of the elements of a 3-vector.

Chapter 6

Experimental Results

We have performed a number of experiments to evaluate our method both with simulated data and real images. In the first part of the experimental evaluation, we have used simulation to explore whether the use of redundant information, as explained in Chapter 5, can improve the "sequential alignment" algorithm (Section 2.4, [32]). In this setting, the feature tracker and outlier detection stages were left. Image measurements and correspondence were generated from a simulated scene.

In the second part of the experimental evaluation, the algorithms were applied on the mobile robot. Thereby, features were extracted and tracked in real-time, by an on-board laptop while the robot followed a predefined path. Subsequently, we tried to recover the traversed path from the collected feature data.

Finally, we discuss the experimental results and present a comparison of "sequential alignment of image triples" and Bundle Adjustment [27], where similar experiments were performed but corresponding points were determined manually.

6.1 Simulation

In a series of simulations, we evaluated the "linear consistency" method described in Chapter 5 against the "sequential alignment of image triples" algorithm (Section 2.4). We wanted to examine whether using redundant two-view orientation estimates can improve the reconstruction accuracy of our previous method. We will first describe the evaluated methods and the experimental setup that was used. Then, we present results on the reconstruction accuracy with respect to rotation, translation and scale drift errors. Analysis of the results leads to an interesting conclusion: Apparently, the translation reestimation step always enhances the solution even if the rotation estimates can not be improved beforehand. This finding is confirmed in a further experiment.

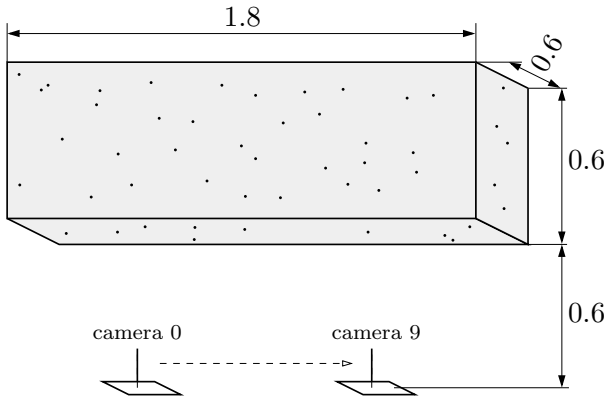


Figure 6.1: *First scenario: Camera movement is along the X axis, i.e., perpendicular to the viewing direction.*

6.1.1 Setup

The following algorithms have been compared:

1. *"triples"*. This is the "sequential alignment of image triples" algorithm described in Section 2.4. It does not make use of redundant information. The rotation and translation direction parameters of consecutive camera poses are estimated from image pairs using the normalized 8-point algorithm. The relative scale of two-view pairs is determined using point correspondences across three images to obtain image triples, which in turn are concatenated to a path.

2. *"consistency"*. This is the algorithm described in Section 5.4. Using the normalized 8-point algorithm, rotation estimates between all image pairs in a short subsequence are computed. For the experiments the length of the subsequence was chosen to be 5 frames. Then, a consistent set of rotations is computed from the redundant estimates. Keeping the rotation parameters fixed, the translation directions for consecutive frames are re-estimated from point correspondences. The final steps are the same as in the "triples" algorithm: Image triples are obtained using correspondences across three views. Triples are concatenated to a path.

Both algorithms yield a path of camera poses which are then compared to the true poses. The input to the algorithms is a set of simulated image measurements and correspondence generated as follows.

Two different scenarios are used. Figures 6.1 and 6.2 illustrate the setups. In both cases, the scenes consisted of 400 point features randomly distributed inside a cuboid. The points are projected onto the image plane of a camera with the following internal parameters:

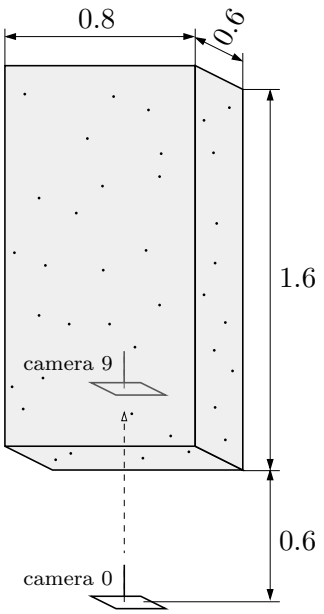


Figure 6.2: *Second scenario: Camera movement is along the Z axis, i.e., parallel to the viewing direction.*

image width	0.8 units = 800 pixel
image height	0.6 units = 600 pixel
focal length	0.6 units
principal point	$x = 0.4$, $y = 0.3$ units

Along with correspondence between image points, this forms the input to the compared algorithms. In both cases 10 camera positions are used. The distance between consecutive cameras is 0.1 units. In the first scenario, the camera is moved along the X axis, i.e., perpendicular to the viewing direction. In the second scenario, the camera is moved along the Z axis, i.e., in the viewing direction.

To simulate measurement error the projected image points are perturbed by adding noise vectors (x, y) drawn from a uniform distribution $x, y \in [-\epsilon/2, \epsilon/2]$. The error level ϵ is varied between 0 and 10 pixels.

6.1.2 Results

The following criteria are used to evaluate the results:

Rotation error. The angle between reconstructed and true camera rotation in degrees, averaged over the reconstructed reconstructed path.

Translation error. Since only the direction of translation can be recovered, we use the angle between reconstructed and true translation

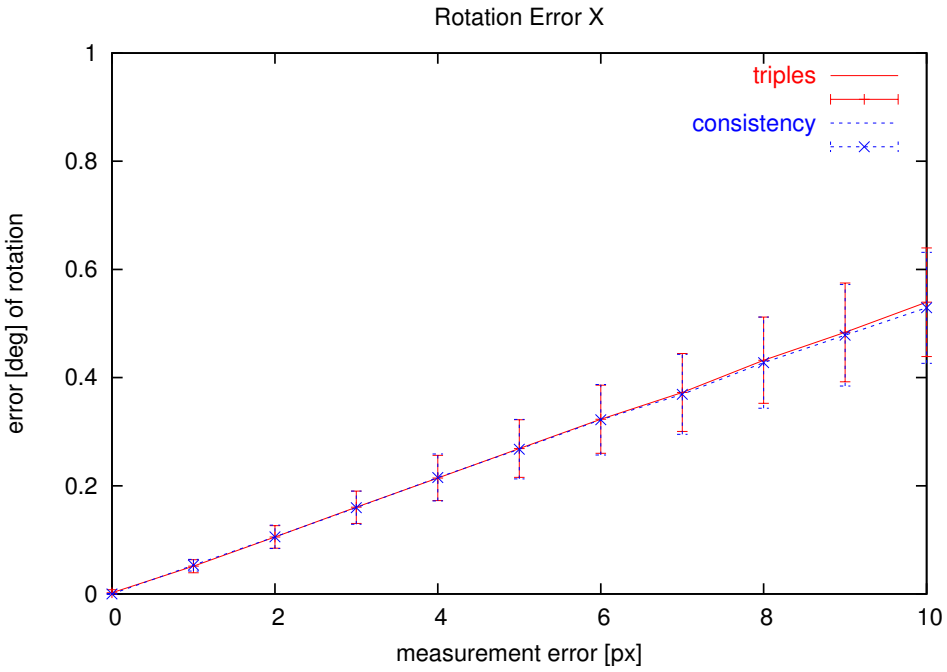


Figure 6.3: *Average rotation error for motion perpendicular to the viewing direction (first scenario). Mean and standard deviation for 1000 runs.*

direction in degrees, averaged over the reconstructed path.

Scale drift. There are 10 cameras in a straight line, equally spaced. Assuming that the distance between the first two cameras is 1 unit, the length of the path should be 9 units. The scale drift value denotes by how many percents the length of the reconstructed path deviates from this.

For each scenario and value ϵ, 1000 experiments were performed. The results are shown in Figures 6.3 through 6.8.

For movement in the viewing direction, rotation reconstruction is improved by the "linear consistency" algorithm as expected. To get the best results from the consistent rotation fitting procedure, we would want rotational errors to be Gaussian distributed. Obviously, this is not the case for the second scenario: For movement perpendicular to the viewing direction, rotation improvement is very small respective not visible at all.

However, surprisingly the error of the translation estimates is significantly improved in both scenarios. Other than we expected, this seems not only due to the improvement in rotation accuracy. In fact, the mere translation reestimation procedure seems beneficial. To confirm this, we augmented the "triples" method by the translation reestimation step, and evaluated it, too. The results are included in Figures 6.5 through 6.8, labeled as "triples

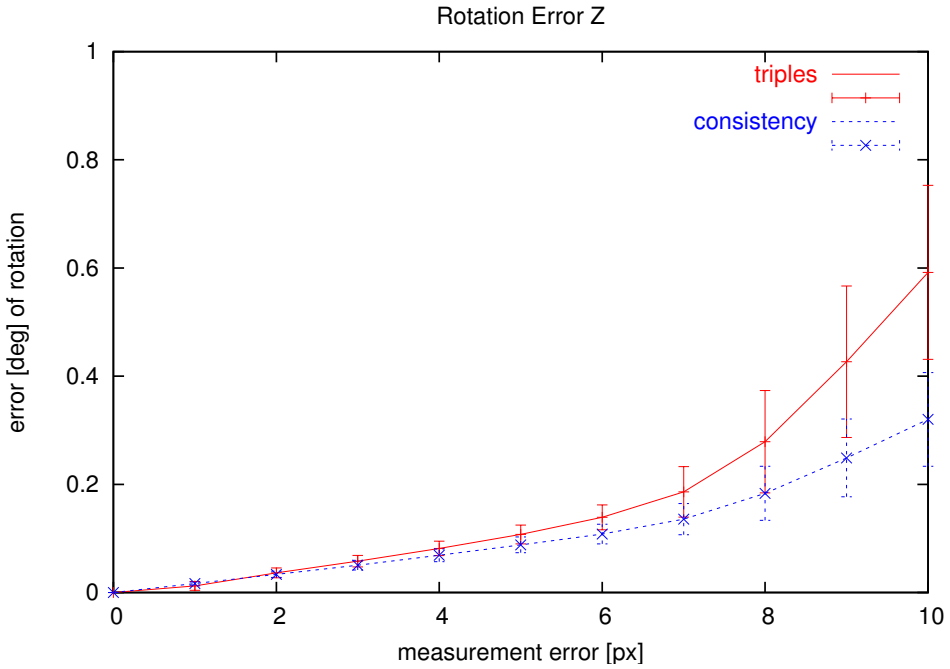


Figure 6.4: *Average rotation error for motion parallel to the viewing direction (second scenario). Mean and standard deviation for 1000 runs.*

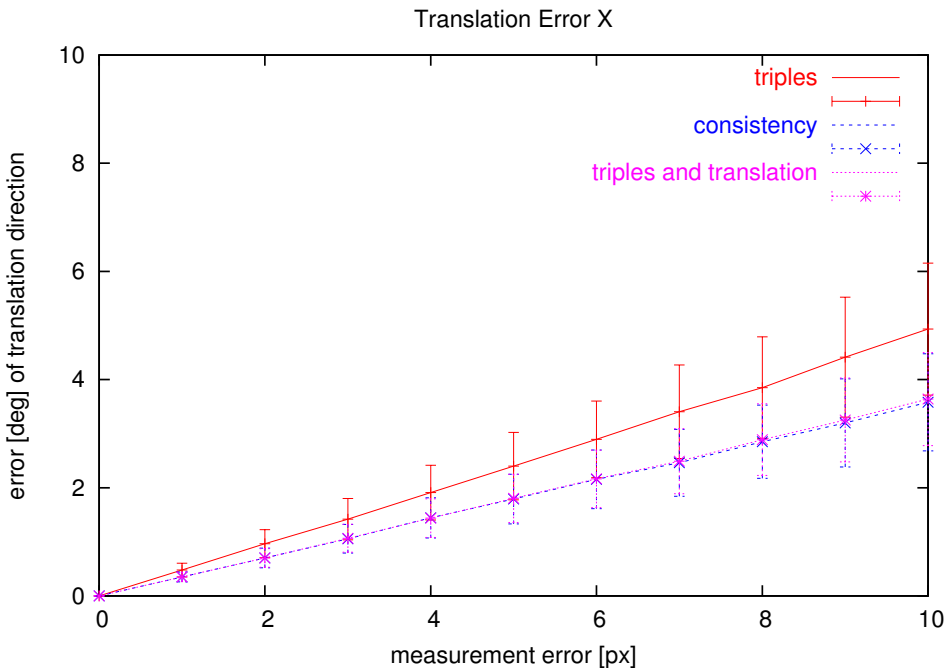


Figure 6.5: *Average translation error for motion perpendicular to the viewing direction (first scenario). Mean and standard deviation for 1000 runs.*

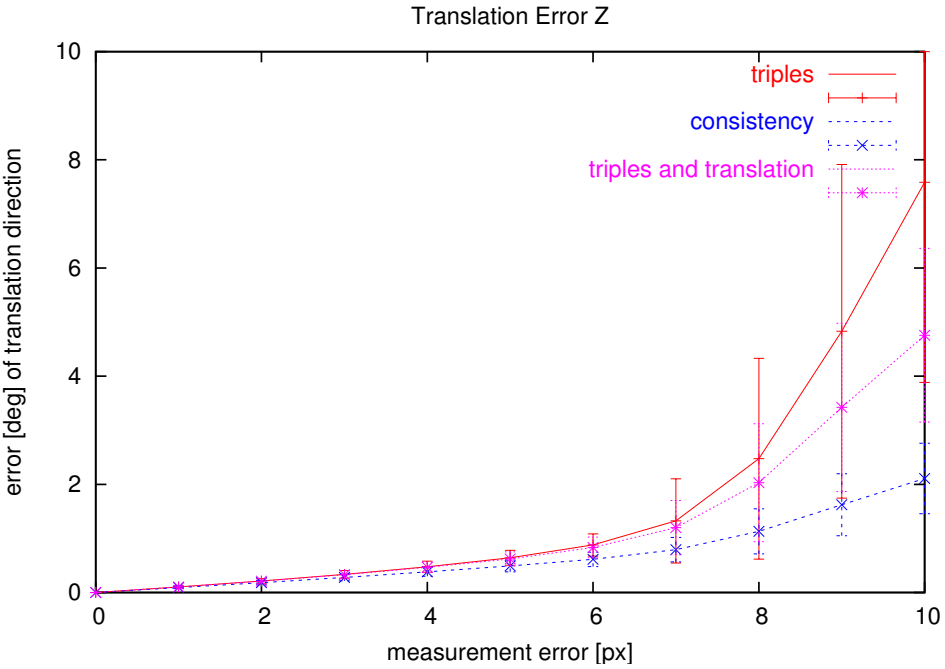


Figure 6.6: *Average translation error for motion parallel to the viewing direction (second scenario). Mean and standard deviation for 1000 runs.*

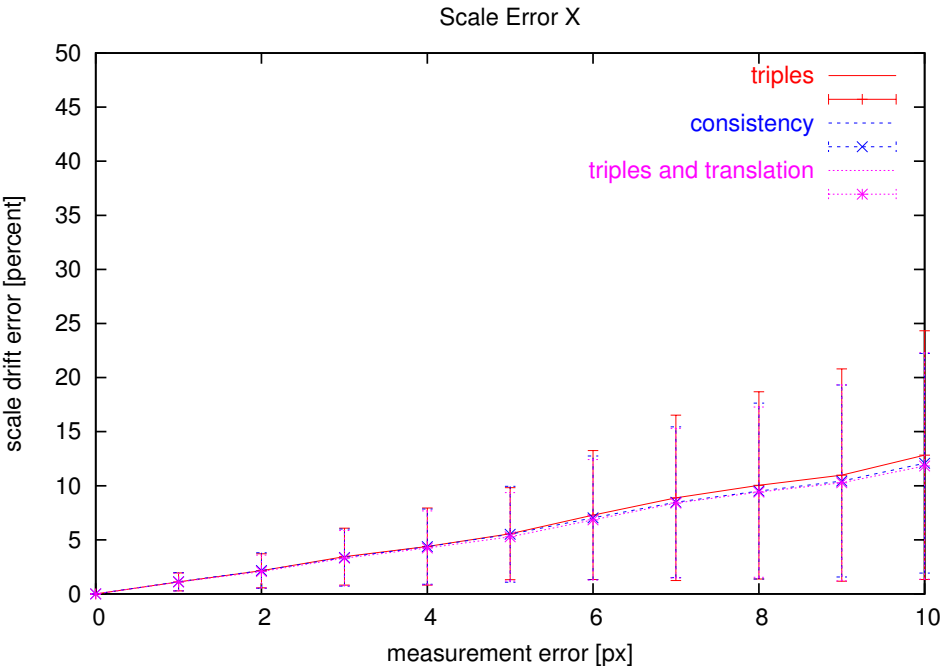


Figure 6.7: *Scale drift for motion perpendicular to the viewing direction (first scenario). Mean and standard deviation for 1000 runs.*

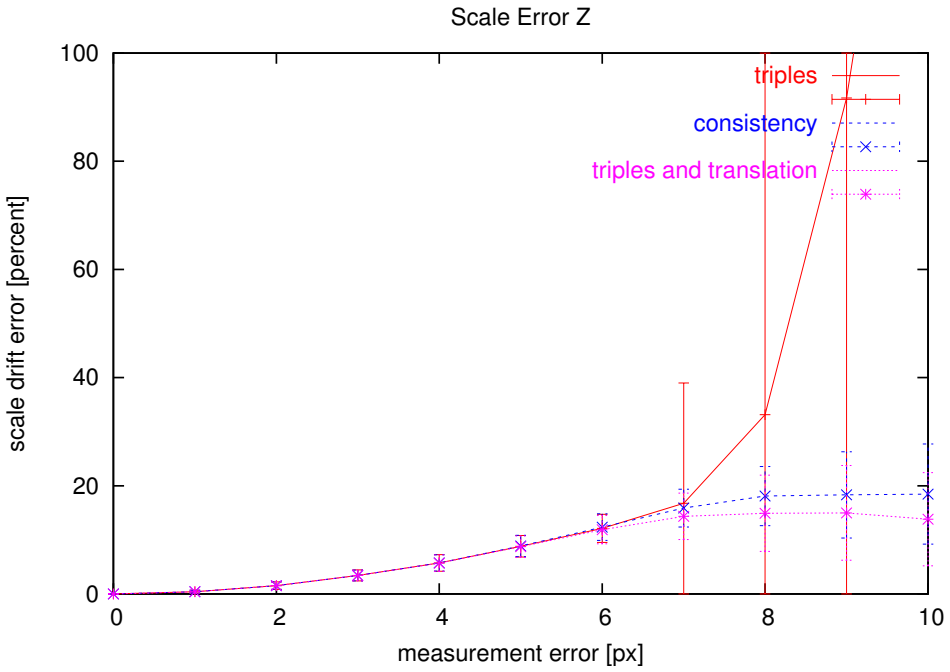


Figure 6.8: *Scale drift for motion parallel to the viewing direction (second scenario). Mean and standard deviation for 1000 runs.*

and translation". Indeed, we find that translation reestimation improves the results even without using rotation redundancy.

Figures 6.7 and 6.8 show the evaluation for scale drift error. First, we see that scale drift is less severe for motion perpendicular to the vieweing direction. This is not surprising since scale estimation is done via triangulation of scenepoints and the geometry for triangulation is better for the first scenario, i.e., the angles between the image point reprojection rays are larger.

Both, "consistency" and "triples and translation" succeed in decreasing scale drift, although not as hefty as one might have hoped. For high levels of noise, we observe that "triples and translation" actually performs a little better than "consistency". This is possibly due to the fact that the relative motion obtained by "consistency" is closer to reality, but the two-frame estimates of "triples and translation" better match the (erroneous) image measurements.

6.2 Robot vision experiments

The mobile robot used in the experiments was a Pioneer 2 equipped with a Sony EVI-D30 camera. Feature data was collected using the feature tracker while the robot followed a predefined path in the environment. Two different

Figure 6.9: *Some images acquired by the robot camera during the "navigation" experiment. Since natural visual features in the scene are sparse, several cardboard boxes have been placed on the floor.*

scenarios have been set up having two different potential applications of visual sensors in mind. The first is referred to as "navigation scenario". Being given a target location, the robot at first steers toward the target and follows a fairly straight line afterwards. Thereby, the camera is looking straight ahead. This is a common situation in many robotic applications. Secondly, a "mapping scenario" was set up where the robot circled around an object situated on the flooe. In this case, the aim is to obtain a 3D model of the object and add it to the robots representation of the environment.

We will first describe the setup in detail and discuss some problems and limitations that had to be addressed in the practical performance of the experiments. Then, we present results of our method. We find that reasonable results can be obtained, but some unsolved problems still remain.

6.2.1 Setup

For the experiments, the robot is required to collect feature data while it is moving along a predefined path. In the "navigation" experiment, the robot drives down a corridor. Some images taken by the robot camera during execution are shown in Figure 6.9. It can be seen that the corridor has been decorated with some cardboard boxes. This is meant to provide the naked white-walled corridor with some additional visual cues. Figure 6.10 depicts the trajectory for this experiment.

Sample images from the "mapping" experiment are shown in Figure 6.11. The trajectory is shown in Figure 6.12. Thereby, the camera keeps looking at the object, the robot is going around. Camera motion is approximately perpendicular to the viewing direction.

The robot control architecture described in Section 2.6 is employed to make the robot execute the pre-planned motion. Particularly, control points are defined in the map, each annotated with a target, the camera should look at. Based on this, the robot controller generates motion commands that

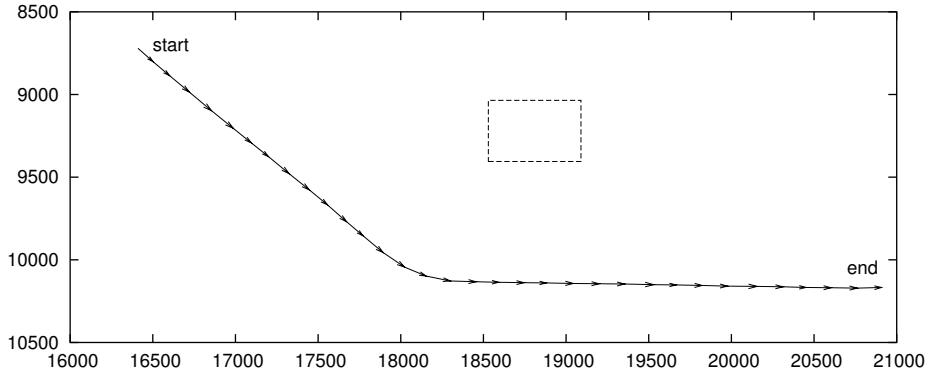


Figure 6.10: *The trajectory of the robot for the "navigation" experiment in birds-eye view. The camera is oriented in the heading direction of the robot.*

Figure 6.11: *Some images acquired by the robot camera during the "mapping" experiment. The robot is circling around an object.*

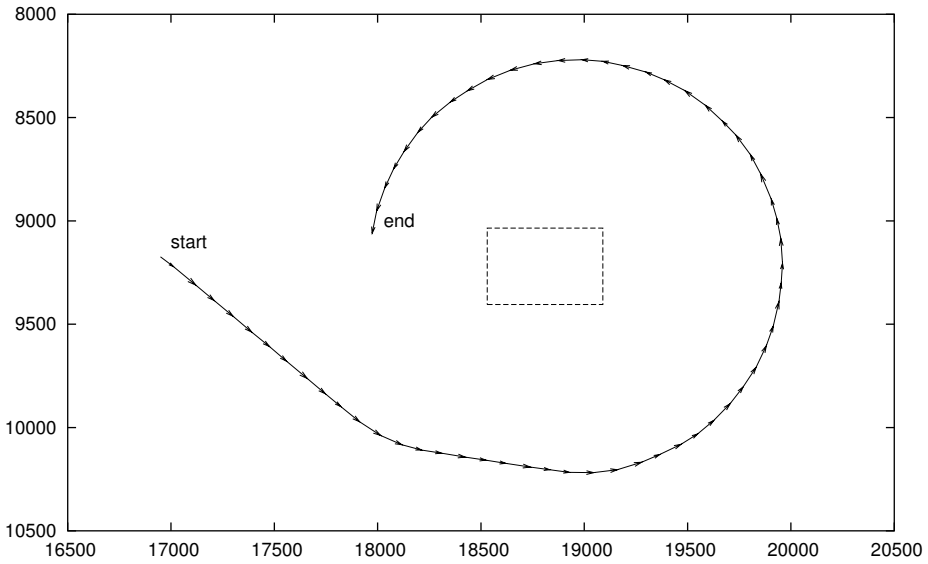


Figure 6.12: *The trajectory of the robot for the "mapping" experiment in birds-eye view. The robot circles around an object which is depicted, too. The camera is looking at the object, i.e., the robot is "looking over it's left shoulder".*

make the robot move smoothly from one control point to the next. The camera pan/tilt angles are smoothly interpolated between control points, too. Feature tracking is done by the on-board laptop. The feature data is sent to an external PC, where it is synchronized with the odometry data and stored into files.

Several problems had to be solved in the planning and execution of the experiments. Firstly, with the onboard laptop we achieve frame rates of about 3.75 Hz. This is sufficient for movement in the viewing direction but rapid turns have to be avoided since the resulting magnitude of image motion cannot be handled by the feature tracker. Consequently, we have tried to keep the velocity of image motion low for the experiments.

Secondly, pure rotations pose some problems for the reconstruction algorithms, e.g., a pan of the camera while the robot stands still. As yet, our approach cannot handle such cases at all. Corresponding features between images are related by a 2D to 2D homography in a pure rotation situation. In this case, there are a family of essential matrices that satisfy the epipolar constraint. Augmenting our method with the detection of homographies and recovering rotations from them should not be overly difficult, though. The situation becomes worse if the magnitude of the pure rotation is such that all the features that were visible before the rotation have moved out of view when the robot starts to move again. In this case, the scale is lost, i.e.,

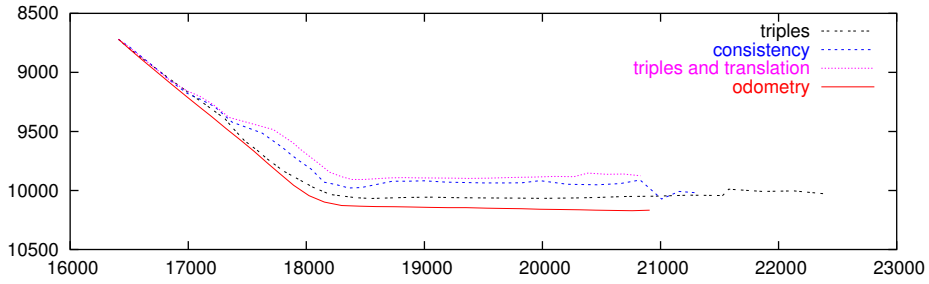


Figure 6.13: *Reconstructed trajectories for the "navigation" experiment in birds-eye view. Odometry readings are shown for comparison. The trajectories have been scaled such that the distances between the leftmost position and the mid of the bend match.*

there is an undetermined scale factor relating the motion sequences before and after the pure rotation. Without further information, the latter problem cannot be solved by Bundle Adjustment either.

Finally, to update the reconstruction to the correct scale, some metric information is needed. This so-called *datum definitions* must relate image coordinates in the image to coordinates in the world. Indeed, no solution can be obtained at all from the ORIENT software unless such information is known. In order to obtain the datum definitions, a cardboard box is marked in a special color and placed at a known location in the scene. The box can be detected in the laser range scans by it's shape and estimated position. It can be detected in the camera image by color-segmentation, too. Together, this provides the necessary datum information. When planning the trajectory, it must be observed that, unlike the camera, the laser range finder cannot be panned. For instance, when the camera is panned left all the way, as it is in the mapping scenario, a marked box in the camera's field of view cannot be detected in the laser range scan and vice versa.

6.2.2 Results

Figures 6.13 and 6.14 show the reconstructed paths for the navigation and mapping experiments. For reference, the robot path as obtained by the odometry sensors is shown, too. Respectively, the following algorithms are compared:

1. *"triples"*. This is the "sequential alignment of image triples" algorithm described in Section 2.4. The relative orientation of consecutive cameras is estimated from image pairs. Using three-image-correspondences, the relative scale of two-view pairs is determined to obtain image triples. These are concatenated to a path.

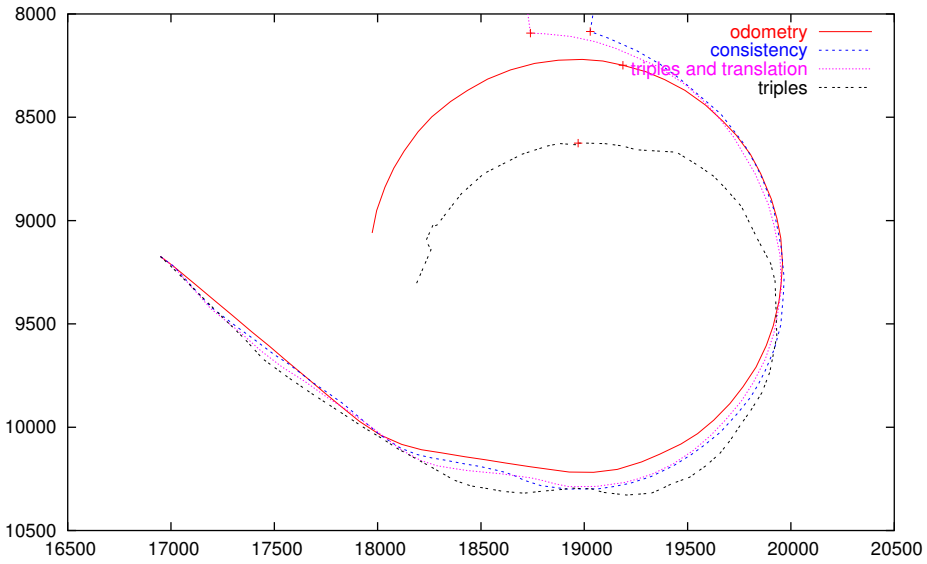


Figure 6.14: *Reconstructed trajectories for the "mapping" experiment in birds-eye view. Odometry readings are shown for comparison. The trajectories have been scaled such that the distances between the leftmost and rightmost positions match.*

2. *"consistency"*. This is the algorithm described in Section 5.4. Rotations between all image pairs in a 5 frame subsequence are estimated. A consistent set of rotations is computed from these redundant estimates. Keeping the rotation parameters fixed, the translations between consecutive frames are re-estimated. Image triples are obtained using correspondences across three views and subsequently concatenated to a path.

3. *"triples and translation"*. The relative orientation of consecutive cameras is estimated from image pairs. Keeping the rotation parameters fixed, the translation parameters are re-estimated. Image triples are obtained using correspondences across three views and subsequently concatenated to a path.

All algorithms operate on identical input data, i.e., outlier detection is performed beforehand and the detected inlier set is used by all algorithms. Since the path reconstruction is only up to scale for our methods, we fixed scale as follows. For the mapping experiment, scale factors were chosen such that the distance between the leftmost and rightmost positions in the path matches the distance between the respective odometry positions. For the navigation experiment, scale was chosen such that the distance between the starting position and the bend in the path match.

As can be seen in Figure 6.14, at one point in the mapping experiment the "consistency" and "triples and translation" fail. The estimated motion direction is almost orthogonal to the true motion. Subsequently, this causes severe error in the estimated relative scale. Scale is obtained by triangulation of scene points and the accuracy of the triangulation depends on the accuracy of the recovered relative motion. We have performed the experiments several times and such fatal mistakes inevitably occur in the processing of extended image sequences. Although this is the case for the "triples" algorithm as well, in this particular experiment, it succeeds in completing the sequence. As yet, we can offer no satisfying explanation for the circumstances that trigger these failures. This is subject to further investigations since practical applicability of the methods depends on detecting such situations and recovering from them.

Ground truth for the camera positions is not known, but odometry is supposed to be quite accurate over short distances. Odometry error is commonly modeled [17] by three contributing sources of error:

1. *distance*. Accumulating distance error is about 1 centimeter per meter. This is the least worrisome source of error.

2. *rotation*. Error of this kind occurs when the robot is changing direction. It amounts to about 8 degrees per full revolution (360 degrees).

3. *drift*. Drift error is the angular deviation when the robot is driving in a straight line. It is about 1 degree per meter.

We can judge the expected accuracy of our methods only in relation to the odometry data.

The mapping experiment is considered only up to the point where "consistency" and "triples and translation" deviate from the expected behaviour. The corresponding position is marked by a cross in the plot on all trajectories, respectively. The expected odometry error in the final position is in the order of 20 centimeters. In this experiment, "consistency" performs best. The positional deviation for the final position is in the same order of magnitude as the odometry error. For "triples" and "triples and translation", the deviation is approximately twice as large. Visually judged, both "triples and translation" and "consisteny" yield much smoother paths than "triples". Probably, accumulated scale drift is responsible for the large deviation in "triples and translation".

In the navigation experiment, expected odometry error in the final position is about 10 centimeters. The effects of scale drift are obvious for the "triples" result, mounting up to a final positional deviation that is approximately 15 times the odometry error. It is known, that the 8-point algorithm suffers from a bias in the viewing direction. Furthermore, the average image motion of the features for camera motion in the viewing direction is small

in comparison with camera motion perpendicular to the vieweing direction. Thus, the angles between triangulation rays for scene point reconstruction are smaller and consequently, relative scale estimates are less accurate. It is highly likely, that the combination of these two facts is responsible for the bad performance of "triples". Both of the other algorithms can significantly reduce scale drift. Nevertheless, in both cases positional deviation is about twice the expected odometry error.

We have also separately evaluated the rotational errors. Thereby, we have manually separated path segments corresponding to straight and circular motion, respectively. For straight segments, we have found that the magnitude of rotational error is comparable the odometry drift error: 1 degree per meter. For circular motion, the most prominent source of odometry error is rotation which is about 8 degrees per revolution. In this case, all of our algorithms perform worse. Rotational error is about 12 degrees per revolution for "consistency" and more than 20 degrees per revolution for the other two algorithms.

6.3 Discussion

Based on our experiments, we can clearly say that our method cannot rival the accuracy of odometry, as yet. This is in part due to the algorithms themselves. In part, it is due to the restrictions of the hardware we use, such as the limited resolution of the camera, image distortion introduced by the frame grabber, etc. Yet, what if odometry is not an option? For humanoid robots (or more general, legged robots) odometry is simply not available. Camera based navigation might prove as a viable alternative in these cases.

It is desirable for the task of navigation that motion estimates are instantly available. This is an important aspect in the visual motion estimation methods, we have explored. We do not necessarily require optimally accurate estimates. Instead, we are interested in obtaining reasonably good estimates in about real-time. However, to evaluate the accuracy of our results, it is useful to compare them to the optimal solution.

We will present some experimental results [27] in which our "triples and scale" algorithm is compared with the ORIENT software [25], a commercial Bundle Adjustment solution. In Bundle Adjustment, nonlinear optimization is applied to find a maximum likelihood solution, i.e., the set of camera and scene point parameters that represents the most likely explanation for a sequence of images. Two experiments were performed, resembling the navigation and mapping scenario, respectively. A mobile robot was used to collect images in which features were manually selected, subsequently. Correspondence between features was established manually, too.

The reconstructed trajectories are shown in Figures 6.15 and 6.16. The

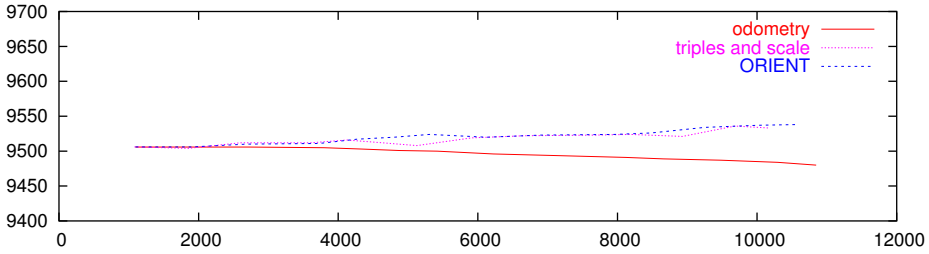


Figure 6.15: *Results for the navigation scenario with manually extracted features. The trajectory was reconstructed using "sequential alignment of image triples" and ORIENT, respectively. Odometry readings are shown for comparison.*

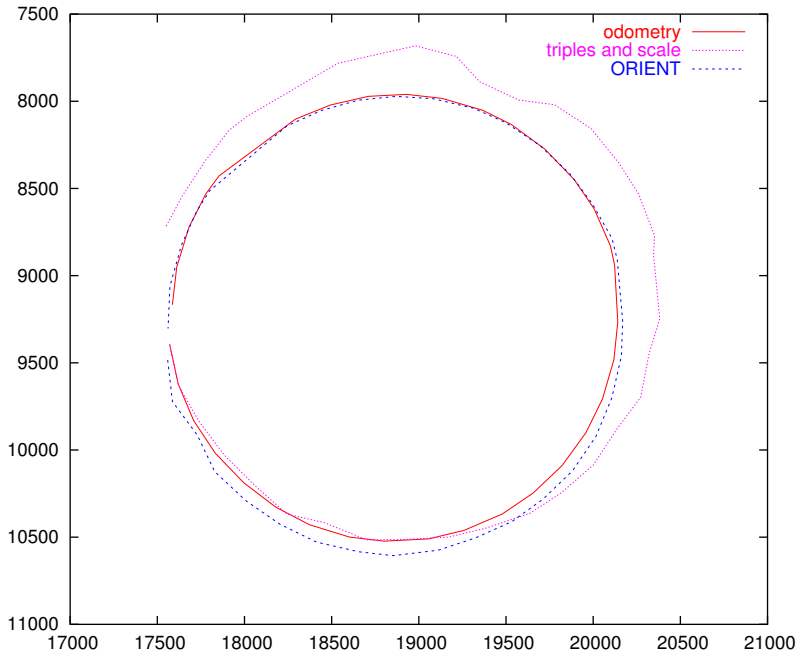


Figure 6.16: *Results for the mapping scenario with manually extracted features. The trajectory was reconstructed using "sequential alignment of image triples" and ORIENT, respectively. Odometry readings are shown for comparison.*

	Triples			ORIENT		
	t [s]	Δ_{XY} [cm]	Δ_α [deg]	t [s]	Δ_{XY} [cm]	Δ_α [deg]
Navigation	0.15	67	-0.3	10	31	0.3
Mapping	0.32	45	20.3	7	6	0.6

Table 6.1: *Runtimes and final deviations for the navigation and mapping experiments.*

overall scale of the ORIENT solution was determined by the known reference points in the scene. To fix the scale of the "triples" solution, the recovered relative scale estimates between consecutive camera pairs were fused with the distances between the corresponding odometry positions using a least squares fit, treating both as unreliable measurements. The final camera positions were measured in both experiments. Table 6.3 lists the positional and rotational deviations in the final positions, respectively. The runtime results obtained on a 1.5 GHz PC are shown as well. The advantage of the "triples" algorithm in terms of processing speed is obvious. The execution times for the Bundle Adjustment are higher by several orders of magnitude. Bundle Adjustment is an iterative optimization technique, where each iteration requires the inversion of a large coefficient matrix formed by the camera and scene point parameters. Thus, the complexity of one iteration step is $O(n^3)$, where n is the number of parameters, i.e., 3 for each scene point and 6 for each camera [41]. The complexity of triples is $O(n^2)$ for each image, where n is the number of point correspondences. That is, complexity does not increase with the length of the sequence.

In prelimiary experiments with automatically detected features we have also found that ORIENT is suspectible to outliers. A few of outliers can be detected, but computation is aborted if their number increases beyond a predefined threshold. The outliers have to be manually removed from the input. Then, the computation can be repeated. Thus, straightforward integration of ORIENT into the robot control architecture is not possible, unless the feature data is preprocessed, e.g., by the robust MSAC algorithm [38]. This is a peculiarity of ORIENT, though, and should not be generalized to Bundle Adjustment methods in the whole.

It is obvious that the accuracy of Bundle Adjustment is by far superior in the mapping experiment. When the robot has completed its path, features from the start of the sequenc become visible again. This allows to back-propagate errors throughout the sequence that otherwise would have accumulated over time. Bundle Adjustment can employ such information since it operates on the image sequence as a whole. In our method, the path is reconstructed sequentially. Since, essentially, we only look at two images

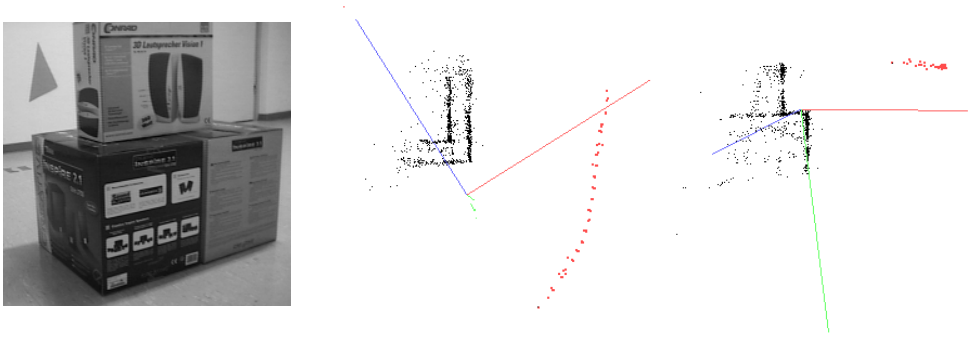


Figure 6.17: *Recovering structure of scene objects. From left to right: An image of the actual object, view from above of the reconstructed scene points, view from the side. The red dots in the reconstructed views indicate camera positions.*

at a time, the closedness of the sequence cannot be exploited. In this particular case, correspondence is easiely re-established since the feature extration is performed manually. The situation is very different if automatic feature detection is used. The feature tracker is not able of reestablishing correspondence, once it has been lost. This points us to an important capability of visual sensors, that we have as yet neglected. So far, we have looked at the camera as a *motion* sensor although information about *structure* can be extracted, too.

6.3.1 3D mapping as a potential application

For sequences of a few images, our motion estimation method should be sufficiently accurate to allow for a reconstruction of scene structure. We have performed the following experiment for a subsequence of the mapping scenario: Camera positions were estimated using the "consistency" approach. Subsequently, a 3D reconstruction of scene points is obtained by triangulation of backprojected image rays. Plots of the reconstructed scene are shown in Figure 6.17.

An aim of future work is to use such partial scene recostruction to extract visual landmarks. One could imagine an approximation of the point cloud by planar patches, may be augmented with texture from the input images. Once we succeed in obtaining models for individual objects, these could be used to recognize objects and update the position estimate in turn.

Chapter 7

Conclusion

Today, mobile robots are equipped with a variety of sensors, among which cameras are only one modality. As the complexity of application scenarios increases, the limitations of the traditional sonar and odometry sensors will become more prominent. A humanoid robot acting in a realistic, complex scenario cannot successfully represent it's environment in a 2D abstraction. It is our belief, that visual sensors will prove as an important alternative in the near future.

For mobile robots, localization and mapping are important lower-level tasks which must be reliably solved for almost every conceivable application. Clearly, data from all available sensors has to be integrated to achieve the best possible result in terms of robustness and accuracy. The aim of this thesis, though, has been to explore what can be achieved with visual sensors only.

It was indicated that both motion *and* structure can be recovered from sequences of images. Visual sensors can supply both position and distance information in this respect. In the present work, we have focussed on the properties of a single camera as a means of estimating relative motion. On the other hand, the preprocessing steps we implemented provide an abstraction from the input images that also forms a solid basis for the recovery of scene structure.

The work we presented can be divided into two parts. Firstly, we were concerned with the extraction of point features corresponding to a rigid scene from raw images. The correspondence problem, i.e., establishing a connection between the projections of a scene feature in different images is addressed, too. We used the Kanade-Lucas-Tomasi Feature Tracker [33] to locate features and track them across images. We have proposed and implemented several modifications to a widely-used implementation, thereby decreasing processing time by more than factor 5. The output of the tracker is contaminated by outliers, i.e., incorrectly tracked features or features that correspond to non-rigid parts of the scene. The presence of outliers can

severely disturb the estimation of structure and motion. Consequently, they should be removed from the data. We applied the MSAC algorithm [38] to detect outliers in the feature data.

Secondly, we have evaluated the "sequential alignment of image triples" algorithm [32] in real-world scenarios. Feature tracking and outlier detection have been used, to process images obtaine by a mobile robot equipped with a camera. Subsequently, the feature data was used to reconstruct the trajectory of the robot. We have proposed a modification to the "sequential alignment of image triples" algorithm which integrates additional constraints available in the feature data. Several redundant estimates of motion between non-consecutive views are computed and integrated. The benefits of this modification have been evaluated in both simulated and real-world experiments. This forms the main contribution of the thesis.

Our experiments have shown that indeed the trajectory of a mobile robot can be recovered efficiently. We are not able to compete with the accuracy of odometry, though. Various sources of error contribute to a degradation of accuracy over time. Rotational and translational errors of individual two-frame estimates accumulate. However, drift in the relative scale of image triples seems to the most critical problem, since the accumulation of individual errors is multiplicative. Thus, the accuracy of a position estimate obtained by our method will decrease over time.

A subject of future work is to obtain visual representations of fixed landmarks in the environment. These could be used to correct the estimated trajectory whenever they are visible. Using our position estimation approach, short subsequences of oriented cameras can be obtained. Subsequently, the structure of the scene can be partially recovered. The reconstructed "scene" is basically a number of 3D points. However, in structured environments these points can be used, to approximate planar surfaces. One could envision an approach similar to the one Liu et. al. [28] apply to map-building from 3D Laser range scans.

Appendix A

Quaternion Rotations

Rotations in space can be represented by various means. Commonly used are 3×3 rotation matrices. This representation is unique, each rotation is associated with a unique matrix. Since a rotation has only 3 degrees of freedom, there are several constraints on the 9 entries of the matrix. Each pair of columns must be orthogonal and each column must have unit norm. Furthermore, the matrix must be orthonormal, i.e. have determinant 1.

Another common representation are Euler angles. A rotation is decomposed as 3 angles, specifying rotations about the x, y and z axis respectively. Euler angles are a minimal representation, i.e., 3 numbers represent the 3 degrees of freedom. Unfortunately, for some rotations there is not a unique representation

Unit quaternions represent a rotation by 4 numbers, $\mathbf{q} = (w, x, y, z)^\top$. There is only one constraint, that $\|\mathbf{q}\| = 1$. Unit quaternions, too, are unique, each rotation is associated with a unique quaternion.

The unit quaternion representing a rotation by θ radians, $-2\pi < \theta < 2\pi$, about the unit vector $\widetilde{\mathbf{x}}$ is

$$\mathbf{q} = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2})\widetilde{\mathbf{x}} \end{bmatrix}. \tag{A.1}$$

Antipodal Quaternions. Rotation about $\widetilde{\mathbf{x}}$ by $\theta - 2\pi$ yields the same result as roation by θ. In fact, this is the same rotation, only "the other way around". The quaternions associated with both rotations are called *antipodal* quaternions. From the definition above, the rotation antipodal to $\mathbf{q}$ is

$$\begin{bmatrix} \cos(\frac{\theta-2\pi}{2}) \\ \sin(\frac{\theta-2\pi}{2})\widetilde{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} \cos(\frac{\theta}{2} - \pi) \\ \sin(\frac{\theta}{2} - \pi)\widetilde{\mathbf{x}} \end{bmatrix} = -\mathbf{q}.$$

Quaternion multiplication. Given two quaternion rotations $\mathbf{q}_1$ and $\mathbf{q}_2$, it is easy to obtain the composite rotation $\mathbf{q}$, i.e. the rotation which has to be

applied to obtain the same result as by applying $\mathbf{q}_2$ and then $\mathbf{q}_1$. Quaternion multiplication is defined as

$$\mathbf{q} = \mathbf{q}_1\mathbf{q}_2 = \begin{bmatrix} w_1 \\ \mathbf{v}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2 \\ w_1\mathbf{v}_2 + w_2\mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2 \end{bmatrix}. \tag{A.2}$$

As one would expect, quaternion multiplication is an associative but not commutative operation.

The above definition can be used, for instance, to derive the rotation $\mathbf{q}^{-1}$ which reverses the effect of rotation $\mathbf{q}$. It is required that $\mathbf{q}^{-1}\mathbf{q} = \mathbf{q}\mathbf{q}^{-1} = (1, 0, 0, 0)^\top$, and using Equation (A.2) obviously the inverse rotation to $\mathbf{q} = (w, x, y, z)^\top$ is $\mathbf{q}^{-1} = (w, -x, -y, -z)^\top$.

Rotation of a vector. When some rotation represented by $\mathbf{q}$ is to be applied to a 3-vector $\widetilde{\mathbf{x}}$, the resulting rotated vector $\widetilde{\mathbf{x}}'$ is obtained by the following quaternion multiplication

$$\begin{bmatrix} 0 \\ \widetilde{\mathbf{x}}' \end{bmatrix} = \mathbf{q} \begin{bmatrix} 0 \\ \widetilde{\mathbf{x}} \end{bmatrix} \mathbf{q}^{-1}.$$

Relation between quaternions and rotation matrices. The columns of a rotation matrix form the base vectors of the rotated frame. Thus, the rotation matrix corresponding to a quaternion $\mathbf{q} = (w, x, y, z)^\top$ can be obtained by rotating the columns of the 3×3 identity matrix, yielding:

$$\mathsf{R}_\mathbf{q} = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix}$$

The inverse operation, obtaining a quaternion from a roation matrix, is quite easy too. Given the above rotation matrix, from its diagonal entries

$$1 + \mathsf{r}_{11} + \mathsf{r}_{22} + \mathsf{r}_{33} = 4 - 4x^2 - 4y^2 - 4z^2,$$

and since $\|\mathbf{q}\| = 1$,

$$w = \pm\frac{\sqrt{1 + \mathsf{r}_{11} + \mathsf{r}_{22} + \mathsf{r}_{33}}}{2}.$$

The other components can be easily retrieved now, too:

$$\begin{aligned} x &= \frac{\mathsf{r}_{32} - \mathsf{r}_{23}}{4w} = \frac{4wx}{4w} \\ y &= \frac{\mathsf{r}_{13} - \mathsf{r}_{31}}{4w} = \frac{4wy}{4w} \\ z &= \frac{\mathsf{r}_{21} - \mathsf{r}_{12}}{4w} = \frac{4wz}{4w}. \end{aligned}$$

The choice of sign for w reflects the direction of rotation, i.e., choice between antipodal quaternions.

Bibliography

[1] Vision workbench. Available via http://www.robots.ox.ac.uk/~lav /Internal/Software/.

[2] Vxl vision libraries. Available via http://vxl.sourceforge.net/.

[3] ActivMedia, Inc. Saphira software manual version 5.3, 1997.

[4] A. Benedetti and P. Perona. Real-time 2-D feature detection on a reconfigurable computer. In *IEEE Conference on Computer Vision and Pattern Recognition*, 1998.

[5] S. Birchfield. An implementation of the Kanade-Lucas-Tomasi feature tracker. Available via http://vision.stanford.edu/~birch/klt/.

[6] S. Birchfield. Derivation of Kanade-Lucas-Tomasi tracking equation. May 1996.

[7] J. Bouget and P. Perona. Visual navigation using a single camera. In *International Conference on Computer Vision*, pages 645–652, 1995.

[8] J. Bouguet. Camera calibration toolbox for Matlab. Available via http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.

[9] J. Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. Available via http://graphics.stanford.edu/courses/cs448a-00-fall/bouget00.pdf.

[10] D. C. Brown. Decentering distortion of lenses. *Photometric Engineering*, 32(3):444–462, 1966.

[11] A. Chiuso, P. Favaro, H. Jin, and S. Soatto. Structure from motion causally integrated over time. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):523–535, 2002.

[12] A. J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *International Conference on Computer Vision*, 2003.

[13] Andrew J. Davison and David W. Murray. Mobile robot localisation using active vision. *Lecture Notes in Computer Science*, 1407, 1998.

[14] C. L. Feng and Y. S. Hung. A robust method for estimating the fundamental matrix. In *7th Conference on Digital Image Computing: Techniques and Applications*, pages 633–642, December 2003.

[15] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[16] W. Förstner and E. Gülch. A fast operator for detection and precise location of distinct points, corners and centres of circular features. In *ISPRS Intercommission Workshop*, pages 281–305, Interlaken, 1987.

[17] D. Fox. *Markov localization: A probabilistic framework for mobile robot localization and navigation.* PhD thesis, Universität Bonn, 1999.

[18] V. Govindu. *Probabilistic models for motion estimation.* PhD thesis, University of Maryland, December 1999.

[19] V. Govindu. Combining two-view constraints for motion estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[20] C. J. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, Manchester, 1988.

[21] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge Univerity Press, 2000.

[22] R. I. Hartley. In defense of the 8-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

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

[24] A. Johnson and L. Matthies. Precise image based motion estimation for autonomous small body exploration. In *5th International Symposium On Artificial Intelligence, Robotics and Automation in Space*, pages 627–634, 1999.

[25] H. Kager. ORIENT: A universal photogrammetric adjustment system. In *Optical 3-D Measurement Techniques*, pages 447–455. Herbert Wichman Verlag, 1989.

[26] L. Kitchen and A. Rosenfeld. Gray level corner detection. *Pattern Recognition Letters*, pages 95–102, December 1982.

[27] A. Knöppler and T. Pietzsch. Vergleich von Bündelblockausgleichung und sequentieller relativer Orientierung von Bildtripeln zur Roboterpositionierung. In *23. Jahrestagung der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation*, 2003.

[28] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun. Using EM to learn 3D models with mobile robots. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[29] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[30] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[31] T. Luhmann. *Nahbereichsphotogrammetrie*. Herbert Wichman Verlag, 2003.

[32] T. Pietzsch. Position estimation of a mobile robot using a single vehicle-mounted camera, 2002. Available via http://www.wv.inf.tu-dresden.de/Publications/Prediploma/pietzsch.pdf.

[33] J. Shi and C. Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[34] C. V. Stewart. Robust parameter estimation in computer vision. *SIAM Review*, 41(3):513–537, 1999.

[35] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[36] T. Tommasini, A. Fusiello, E. Trucco, and V. Roberto. Making good features track better. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 178–183, 1998.

[37] P. H. S. Torr. Bayesian model estimation and selection for epipolar geometry and general manifold fitting. *International Journal of Computer Vision*, 50(1):35–61, 2002.

[38] P. H. S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *CVIU*, 78(1):138–156, 2000.

[39] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – A modern synthesis. In *Vision Algorithms: Theory and Practice*, Lecture Notes in Computer Science, pages 298–375. Springer Verlag, 2000.

[40] F. M. Waltz and J. W. V. Miller. An efficient algorithm for gaussian blur using finite-state machines. In *SPIE Conference on Machine Vision Systems for Inspection and Metrology VII*, volume 3521, pages 334–341, Boston, MA, 1998.

[41] X. Wang and T. A. Clarke. Separate adjustment of close range photogrammetric measurements. *ISPRS Journal of Photogrammetry and Remote Sensing*, 32(5):177–184, 1998.

[42] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. Technical Report RR-2676, INRIA, October 1995.

[43] Zhengyou Zhang. A new multistage approach to motion and structure estimation: From essential parameters to euclidean motion via fundamental matrix. Technical Report RR-2910, INRIA, June 1996.