# Efficient Line and Patch Feature Characterization and Management for Real-time Camera Tracking

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

**DISSERTATION**

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

von
**Dipl.-Inf. Harald Wuest**
geb. in Spaichingen

| | |
|---|---|
| Referenten der Arbeit: | Prof. Dr. Dieter Fellner |
| | Prof. Dr. Didier Stricker |
| Tag der Einreichung: | 8. 9. 2008 |
| Tag der Disputation: | 5. 11. 2008 |

D17
Darmstadt 2008

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

# Acknowledgements

During the last years of working on this thesis I was greatly supported by the team of the Department of Virtual and Augmented Reality at the Fraunhofer IGD in Darmstadt. I am deeply grateful for a congenial environment in which to work and the help and friendship of my colleagues.

First of all, many thanks go to my PhD supervisor Prof. Dr. Dieter Fellner for his support.

I wish to extend my sincere thanks to Prof. Dr. Didier Stricker for his assistance and advice. I am deeply grateful for all his suggestions in technical and scientific problems I encountered.

Furthermore, I also thank the whole team of CAMTech at Nanyang Technological University for supporting me in all matters and making my stay there a very enjoyable time.

Finally, I express my gratitude to my parents for their persistence and motivating encouragement.

# Abstract

One of the key problems of augmented reality is the tracking of the camera position and viewing direction in real-time. Current vision-based systems mostly rely on the detection and tracking of fiducial markers. Some markerless approaches exist, which are based on 3D line models or calibrated reference images. These methods require a high manual preprocessing work step, which is not applicable for the efficient development and design of industrial AR applications.

The problem of the preprocessing overload is addressed by the development of vision-based tracking algorithms, which require a minimal workload of the preparation of reference data.

A novel method for the automatic view-dependent generation of line models in real-time is presented. The tracking system only needs a polygonal model of a reference object, which is often available from the industrial construction process. Analysis-by-synthesis techniques are used with the support of graphics hardware to create a connection between virtual model and real model.

Point-based methods which rely on optical flow-based template tracking are developed for the camera pose estimation in partially known scenarios. With the support of robust reconstruction algorithms a real-time tracking system for augmented reality applications is developed, which is able to run with only very limited previous knowledge about the scene. The robustness and real-time capability is improved with a statistical approach for a feature management system which is based on machine learning techniques.

# Contents

# 1. Introduction

## 1.1. Augmented Reality

Augmented Reality (AR) deals with the combination of real world images with computer graphics. In contrast to Virtual Reality, where the user is totally immersed into a virtual environment, in Augmented Reality the real environment is still perceived and additional virtual objects are overlaid into the user's field of vision. Azuma [3] defines an augmented reality system with the following characteristics: Virtual and real world are combined, the system is capable of handling interaction in real-time and the registration between real and virtual data has to be carried out in three dimensions. The real-time capability of an augmented reality system is a differentiation to technologies where real and virtual data are synthesized off-line, like in the movie post-production. In contrast thereto augmented reality systems require that the 3D alignment of virtual objects into the real world is performed in real-time.

A very common approach for an augmented reality system is to analyze a digital camera image to estimate the camera position and viewing direction and to overlay this image with additional virtual information. Head-mounted displays (HMDs) are widely used devices for augmented reality applications. Video-see-through devices consist of a camera and a display, on which the augmented camera image is shown. See-through displays consist of a semi-transparent screen, where the augmented image is overlaid directly on the real world image. Since the development of display technology is not in the state that an ergonomic and comfortable use is possible, many AR-applications have been implemented on tablet PCs, ultra-mobile PCs or PDAs.

A variety of augmented reality applications exist. A high potential of Augmented Reality covers the field of industrial maintenance, where a technician is supported with instructions which are directly overlaid in the technician's field of view. With such an AR maintenance system an unskilled worker is able to perform complicated repairs of motor engines, control units or other technical equipment. Other applications are in the field of architecture, tourism and entertainment, where virtual and real worlds are mixed in outdoor scenarios. Virtual constructions of buildings can be placed into real city skylines or an ancient ruin can be augmented with virtual reconstructions of a temple. Furthermore, in TV live broadcasts, like sport events, augmented reality technologies are used to superimpose additional information to clarify special situations for the viewer.

## 1.2. The Tracking Challenge

One of the most relevant problems in current research regarding augmented reality is the robust registration of a virtual model into the real scene. For a correct overlay of a virtual augmentation into an image, the camera position and viewing direction, also called the camera pose, must be known. The estimation of the camera pose is denoted as tracking.

The tracking for augmented reality applications has a number of requirements. It must be able to work in real-time with a sufficient update rate and the precision must be high enough that the virtual model is not misplaced in the real image. Furthermore, the user should not observe any latency when moving around, i.e there should be no time lag and the estimated camera pose should always match the current frame. When the user is not moving, the camera pose must be stable and no jitter should be observed.

A wide range of tracking technologies can be used for the camera pose estimation. If the object, which shall be tracked, consists of several parts, which are physically connected with measurable joints, these measures can be used to track the movements of the object. Such mechanical tracking techniques are integrated into devices like augmented reality telescopes ore movable screens. Electromagnetic sensors can be a good choice for tracking an instrument in a predefined field of activity, but have the disadvantage that metallic objects can disturb the tracking significantly. With inertial sensors like gyroscopes or accelerometers the relative motion variation of an object can be estimated. Hybrid tracking systems combine the features of several different sensor sources, e.g. a mixture of vision, inertial or electromagnetic devices. For outdoor applications GPS receivers are a possibility for a rough position estimation, but the accuracy is often too low for a precise registration.

A widely used technique for the estimation of a camera pose is optical tracking. Since for most augmented reality applications a camera image is already available, no additional sensors are needed. With computer vision approaches the complete camera parameters are estimated by analyzing the images of the observing camera. Digital cameras are consumer products and are available for very cheap prices. A simple camera can be therefore one of the most inexpensive tracking devices. Another benefit of optical methods is that it might be possible that no costly preparations of the working environment need to be made as for the electromagnetic tracking. However, the development of computer vision-based tracking systems, which are able to robustly compute the camera position and viewing direction out of an image sequence, is a very complex problem, since the requirements for augmented reality applications are quite demanding. A considerable amount of research is carried out nowadays in the area of vision-based tracking, but no complete systems, which fulfill all the needs of AR applications, have been developed so far.

To simplify the optical tracking problem, fiducial markers have been used, because the detection of specially designed markers can be carried out robustly in real-time. Such fiducial-based tracking methods have been widely used, but in many scenarios they are not applicable, because the preparation of the scene with markers is very intricate and sometimes not possible at all.

To avoid the manual positioning of markers, a tracking system must be able to use only natural features which occur in the scenario. With reference images a natural feature-

based tracking is possible, but the calibration of reference images is a too complex preprocessing step for the straightforward creation of industrial AR applications. Thus a high demand exists for a fully automatic tracking system which is able to work in real-time with a minimal previous knowledge of the scene.

## 1.3. Contributions and Overview

The goal of this thesis is the development of a real-time computer vision-based tracking system which fulfills the requirements of augmented reality applications. In the first part of this thesis line-based tracking approaches which rely on a given geometric model are investigated and further developed. Novel rendering methods are developed to create a tracking system, in which computer graphic techniques are used to create a connection between polygonal 3D models and edge-based tracking algorithms. The second part discusses the tracking of point-based image features and the reconstruction of the feature geometry. The key idea is to develop a tracking system which gathers all information which is needed for the continuous tracking during runtime. After observing and tracking the scene for a sufficient time a feature map is created which is then used for a camera pose estimation under strong illumination and aspect changes. With the support of machine learning algorithms a sophisticated feature management system is developed, which results in an overall both robust and efficient tracking framework.

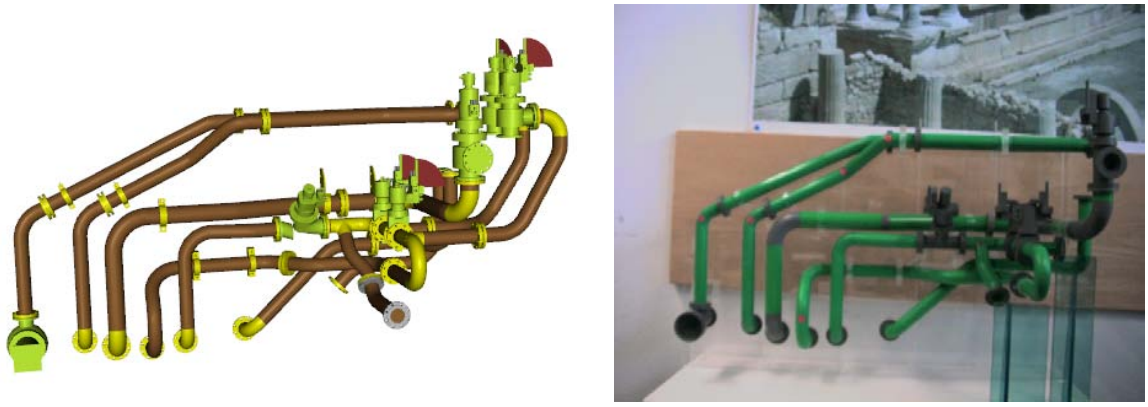
In Chapter 2 the relevant computer vision basics are discussed and the mathematical notations are introduced. The perspective camera model, on which most of our algorithms rely, is presented, and the state-of-the-art methods for robust camera pose estimation are shortly described.

Chapter 3 provides an overview of fiducial-based tracking methods.

The edge-based tracking approaches, which are implemented and further developed for our tracking system, are described in Chapter 4. The standard approach of tracking a given 3D line model is often denoted as the *RAPiD* tracker. This method is based on the orthogonal search for gradient maxima along search lines at control points on the projected model edges.

We extend this method in that way that both on-line and off-line information is used to increase the robustness and convergence behavior of the tracking system. The model geometry is combined with the visual appearance of an edge in the camera images and an adaptive learning method is used to create a most general multiple appearance representation of the control points of a model edge. A visibility test for self-occluding edges of the regarded object is carried out with the support of modern graphics hardware. We evaluate the algorithm and demonstrate that our system outperforms other purely line-based tracking systems in robustness. The tracking method and the results of our approach are presented in [118].

Since line models are rarely given, the methods of Chapter 4 are not very user friendly, because a 3D line model has to be created in such a way that it represents strong image edges in the regarded scene. Polygonal models in the VRML format are often given and



(a) (b)

Figure 1.1.: A virtual model (a) and a real model (b) of an industrial object. The task of the tracking algorithms is to align the virtual model on the real model correctly.

can be exported from the industrial construction and design process easily. Such a virtual model and the real model is shown in Figure 1.1.

In Chapter 5 we present a novel tracking method which uses rendering techniques to align a virtual model onto the real model in an image by using contour-based tracking algorithms. Our main contribution for this problem is a real-time edge model generation, in which a 3D edge model is created on the fly with only those edges which are visible for a predicted camera pose at an adequate level of detail in every frame. Parts of the algorithm are implemented on graphic shader hardware to increase the performance of the creation process. A two-stage tracking method that uses image and object information is used for a more stable handling of large camera movements. With the tracking approach presented in Chapter 4 it is possible to track any non-textured industrial object with only a given polygonal model. In contrast to other methods it is possible to track objects which mostly consist of silhouette edges from any viewing direction. The complete description and evaluation of the algorithm is published in [120].

If a scenario consists of well textured planar surfaces, point-based tracking methods are a more suitable choice to create a markerless tracking system. The detection and tracking of point features is discussed in Chapter 6.1. The main focus of the chapter is the optical flow based template alignment. We improve the template tracking with an approach, in which the scale invariance is increased by representing feature points with multiple templates of different scale levels. The tracking results are demonstrated with applications like poster trackers or tracking algorithms where a polygonal 3D model is used for the acquisition of 3D coordinates.

In Chapter 7 the tracking in unknown scenarios and the reconstruction of scene geometry is discussed. We present a system, in which feature points which do not belong to known parts of the scene are reconstructed and refined on-line for the further continuous camera pose estimation. Not only 3D coordinates, but also surface normal vectors are reconstructed and used for a precise prediction of lost or occluded feature points. Many parts of this tracking algorithm are described in [12].

In dynamic scenes with occluding objects many features need to be tracked for a robust real-time camera pose estimation. An open problem is that tracking too many features has a negative effect on the real-time capability of a tracking approach. In Chapter 8.1 a feature management method is proposed which performs a statistical analysis of the ability to track a feature and then uses only those features which are very likely to be tracked from a current camera position. Thereby a large set of features in different scales is created, in which every feature holds a probability distribution of camera positions from which the feature can be tracked successfully. As only the feature points with the highest probability are used in the tracking step, the method can handle a large amount of features in different scales without losing the ability of real-time performance. Both the statistical analysis and the reconstruction of the features' 3D coordinates are performed online during the tracking and no preprocessing step is needed. A description of the complete system is published in [119].

The derivation of different motion models for the optical flow-based template alignment with and without illumination compensation is presented in Appendix A.

## 1.4. Publications

The majority of the work described in this thesis has been peer-reviewed and presented at conferences. This is a list of the publications derived from this work:

- Wuest, Harald; Wientapper, Folker; Stricker, Didier: **Adaptable Model-Based Tracking Using Analysis-by-Synthesis Techniques.** In *Proceedings of Computer Analysis of Images and Patterns (CAIP)*, 2007.
- Koch, Reinhard; Evers-Senne, Jan-Friso; Schiller, Ingo; Wuest, Harald; Stricker, Didier: **Architecture and Tracking Algorithms for a Distributed Mobile Industrial AR System.** In *Proceedings of the 5th International Conference on Computer Vision Systems (ICVS)*, 2007.
- Becker, Mario; Bleser, Gabriele; Pagani, Alain; Stricker, Didier; Wuest, Harald: **An Architecture for Prototyping and Application Development of Visual Tracking Systems.** In *Proceedings of IEEE 3DTV-Conference: Capture, Transmission and Display of 3D Video*, 2007.
- Wuest, Harald; Pagani, Alain; Stricker, Didier: **Feature Management for Efficient Camera Tracking.** In *8th Asian Conference on Computer Vision (ACCV)*, 2007.
- Webel, Sabine; Becker, Mario; Stricker, Didier; Wuest, Harald: **Identifying Differences Between CAD and Physical Mock-ups Using AR.** In *Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.
- Wuest, Harald; Stricker, Didier: **Tracking of Industrial Objects by Using CAD Models.** In *Journal of Virtual Reality and Broadcasting 4 (JVRB)*, 2007.
- Bleser, Gabriele; Wuest, Harald; Stricker, Didier: **Online Camera Pose Estimation in Partially Known and Dynamic Scenes.** In *Proceedings of the*

*Fifth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, 2006.

- Wuest, Harald; Stricker, Didier: **Robustes Kamera-Tracking für industrielle Anwendungen im Bereich der Erweiterten Realität.** In *1. Internationales Symposium Geometrisches Modellieren, Visualisieren und Bildverarbeitung*, Stuttgart, 2006.
- Wuest, Harald; Stricker, Didier: **Tracking of Industrial Objects by Using CAD Models.** In *3. GI-Workshop der Fachgruppe Virtuelle Realität und Augmented Reality*, 2006.
- Wuest, Harald; Vial, Florent; Stricker, Didier: **Adaptive Line Tracking with Multiple Hypotheses for Augmented Reality.** In *Proceedings of the Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, 2005.
- Becker, Mario; Bleser, Gabriele; Pagani, Alain; Pastarmov, Yulian; Stricker, Didier; Vial, Florent; Weidenhausen, Jens; Wohlleber, Cedric; Wuest, Harald: **Visual Tracking for Augmented Reality: No Universal Solution but Many Powerful Building Blocks.** In *2. GI-Workshop der Fachgruppe Virtuelle Realität und Augmented Reality*, 2005.

# 2. Computer Vision Basics

## 2.1. Camera Models

In order to understand the imaging process, in this section the standard pinhole camera model is introduced. Although in recent years other camera models like omni-directional cameras have received more attention, because of their wider field of view, the pinhole camera model is appropriate for most cameras which are used for tracking in augmented reality scenarios.

### 2.1.1. Perspective camera

The principle of the pinhole camera is very old. Basically such a camera consists of a box with a small pinhole at one side, which is the optical center, and a projection plane on the other side. The image on the projection plane is formed by light rays which pass from an object through the pinhole. Thereby an upside-down image is created. For the perspective camera model the mirrored image plane in front of the optical camera center is regarded as the projective plane. In figure 2.1 the projection of a 3D point $\boldsymbol{M} = (X, Y, Z)^T$ onto a 2D point $\boldsymbol{m} = (x, y)^T$ in the image plane is depicted.

The 3D point $\boldsymbol{M} = (X, Y, Z)^T$ is expressed in the Euclidean world coordinate system $(W_c, \vec{x}_w, \vec{y}_w, \vec{z}_w)$, and the projected 2D point in the image coordinate system $(\vec{u}, \vec{v})$.

If $\tilde{\boldsymbol{m}} = (x, y, 1)^T$ and $\tilde{\boldsymbol{M}} = (X, Y, Z, 1)^T$ are the homogeneous coordinates of $\boldsymbol{m}$ and $\boldsymbol{M}$, the projection can be described by

$$s\tilde{\boldsymbol{m}} = P\tilde{\boldsymbol{M}}, \tag{2.1}$$

where $s$ is a scale factor and $P$ a $3 \times 4$ projection matrix. This equation shows that the projection of a point $\tilde{\boldsymbol{M}}$ to the 2D image point $\tilde{\boldsymbol{m}}$ is linear in projective space. The projection matrix $P$ is defined up to a scale factor and therefore has 11 degrees of freedom. These degrees of freedom consist of 6 extrinsic parameters, which describe the orientation and the translation of the camera, and 5 intrinsic parameters, which depend on the internal parameters of the camera, such as the focal length $f$.

To separate the intrinsic parameters from the extrinsic parameters, the projection matrix $P$ can be decomposed as

$$P = K\left[R|\boldsymbol{t}\right], \tag{2.2}$$

where $K$ is a $3 \times 3$ calibration matrix, which depends on the intrinsic parameters of the camera. The $3 \times 3$ rotation matrix $R$ represents the orientation of the camera coordinate

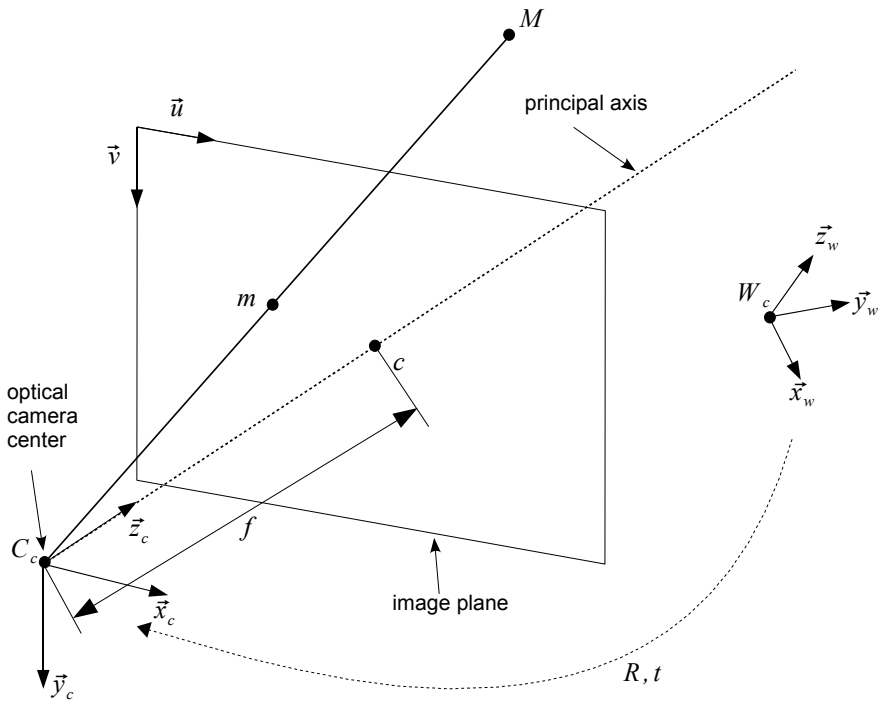



Figure 2.1.: Model of the perspective camera. $M$ is a 3D point, and $m$ its 2D projection onto the image plane.

frame, and the vector $\boldsymbol{t}$ is a 3-dimensional translation from the origin of the world frame into the origin of the camera frame.

A homogeneous 3D point $\tilde{M}$ can therefore be projected to a homogeneous 2D point $\tilde{\boldsymbol{m}}' = (\tilde{m}'_x, \tilde{m}'_y, \tilde{m}'_z)^T = s\tilde{\boldsymbol{m}}$ by the following equation:

$$\tilde{\boldsymbol{m}}' = K \left[R|\boldsymbol{t}\right] \tilde{M} \tag{2.3}$$

The Euclidean image coordinates $\boldsymbol{m} = (x, y)^T$ can be computed by homogenizing $\tilde{\boldsymbol{m}}'$:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{\tilde{m}'_x}{\tilde{m}'_z} \\ \frac{\tilde{m}'_y}{\tilde{m}'_z} \end{pmatrix} \tag{2.4}$$

### Intrinsic parameters

The upper triangular transformation matrix $K$ represents the transformation from a point in the camera coordinate system to a homogeneous point in the image plane. The matrix $K$ is also called camera calibration matrix and depends on 5 parameters. It can be written as

$$K = \begin{pmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.5}$$

where $f_u = k_u \breve{f}_u$ and $f_v = k_v \breve{f}_v$ represent the focal length of the camera in terms of pixel dimensions. The values of the normalized focal length are here denoted as $\breve{f}_u$ and $\breve{f}_v$ and the factors $k_u$ and $k_v$ are the number of pixels per unit distance in the $\vec{u}$ and $\vec{v}$ directions respectively. If all pixels are square, which is mostly the case with modern CCD cameras, then $f_x$ is equal to $f_y$. The principal point $c = (u_0, v_0)^T$ represents the image coordinate of the intersection of the principal axis and the image plane. Similarly, $u_0 = k_u \breve{u}_0$ and $v_0 = k_v \breve{v}_0$ are represented in terms of pixel dimension, where $\breve{u}_0$ and $\breve{v}_0$ are the coordinates of the principal point which are normalized to the image dimensions $k_u$ and $k_v$. Usually the principal point $c$ is very close to the center of the image. The parameter $s$, which is known as the skew parameter, is 0 in most of the cases regarding modern cameras. It is only non-zero, if the directions $\vec{u}$ and $\vec{v}$ are not perpendicular.

Often is is useful to express the camera calibration matrix $K$ independently from the image dimensions. For example, if an algorithm uses different levels of an image pyramid, image planes of different resolutions are needed to project a 3D point. The normalized camera calibration matrix

$$\breve{K} = \begin{pmatrix} \breve{f}_u & \breve{s} & \breve{u}_0 \\ 0 & \breve{f}_v & \breve{v}_0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{2.6}$$

which is independent from the image dimensions, is related to the camera calibration matrix $K$ by the following equation:

$$K = \begin{pmatrix} k_u & 0 & 0 \\ 0 & k_v & 0 \\ 0 & 0 & 1 \end{pmatrix} \breve{K} \tag{2.7}$$

### Extrinsic parameters

The $3 \times 4$ matrix $\left[R|\boldsymbol{t}\right]$ represents the Euclidean transformation of a homogeneous point $\tilde{M}$ from the world coordinate system $(W_c, \vec{x}_w, \vec{y}_w, \vec{z}_w)$ to the camera coordinate system $(C_c, \vec{x}_c, \vec{y}_c, \vec{z}_c)$.

A 3D point $M$ can be transformed to the camera coordinate system by

$$M_c = \left[R|\boldsymbol{t}\right] \tilde{M} = RM + \boldsymbol{t}. \tag{2.8}$$

Both the rotation matrix $R$ and the translation vector $\boldsymbol{t}$ depend on 3 parameters each. These 6 extrinsic parameters which define orientation and the position of the camera are often referred to as camera pose. The main task of the tracking methods is to estimate these extrinsic camera parameters.

Since the optical center of the camera $C$ in the world coordinate system is transformed to the origin of the camera coordinate system, the equation $\mathbf{0} = R\boldsymbol{C} + \boldsymbol{t}$ must hold. Therefore the optical center of the camera in world coordinates can be calculated by $\boldsymbol{C} = -R^T \boldsymbol{t}$.

### 2.1.2. Lens Distortion

The projective camera model is an ideal model of the pinhole camera. In practice, however, a significant radial distortion can often be observed, especially if the camera has a wide field of view. To model such an effect, a 2D deformation of the image can be used to compensate the radial distortion. A very common model can be described as follows. Let $\breve{\boldsymbol{m}} = (\breve{x}, \breve{y})^T$ be the normalized image coordinates of the undistorted point $\boldsymbol{m} = (x, y)^T$ and $\breve{\boldsymbol{m}}_{\boldsymbol{d}} = (\breve{x}_d, \breve{y}_d)^T$ the corresponding normalized coordinate of the distorted point $\boldsymbol{m_d} = (x_d, y_d)^T$. The relation between undistorted normalized and the undistorted observed image coordinates can be described as

$$\boldsymbol{m} = \boldsymbol{c} + \begin{pmatrix} k_u & 0 \\ 0 & k_v \end{pmatrix} \breve{\boldsymbol{m}}, \tag{2.9}$$

where $\boldsymbol{c}$ is the principal point in the image coordinate system and $k_u$ and $k_u$ are the image dimensions. For distorted coordinates the same relation holds.

If $\breve{\boldsymbol{m}} = (\breve{x}, \breve{y})^T$ is the normalized undistorted point in the image plane, and $\breve{\boldsymbol{m}}_d = (\breve{x}_d, \breve{y}_d)^T$ is the corresponding normalized distorted point, then the distorted point can be approximated with:

$$\breve{\boldsymbol{m}}_d = 1 + d_{\text{radial}}(\breve{\boldsymbol{m}}) + d_{\text{tangential}}(\breve{\boldsymbol{m}}). \tag{2.10}$$

The radial distortion can be expressed as

$$d_{\text{radial}}(\breve{\boldsymbol{m}}) = (k_1 r^2 + k_2 r^4 + k_3 r^6 + ...)\breve{\boldsymbol{m}}, \tag{2.11}$$

where $r = \sqrt{\breve{x}^2 + \breve{y}^2}$ and the factors $k_1, k_2, ...$ are the radial distortion coefficients. In most of the cases, two radial distortion coefficients are enough to model the radial distortion sufficiently.

The tangential distortion, which was introduces by Brown [14], can be computed by

$$d_{\text{tangential}}(\breve{\boldsymbol{m}}) = \begin{pmatrix} 2t_1\breve{x}\breve{y} + t_2(r^2 + 2\breve{x}^2) \\ t_1(r^2 + 2\breve{y}^2) + 2t_2\breve{x}\breve{y} \end{pmatrix}, \tag{2.12}$$

where $t_1$ and $t_2$ are the tangential distortion coefficients. Often the tangential distortion is neglected, because its influence is not very significant.

For many computer vision-based tracking algorithms, an input image is undistorted with some given distortion parameters. Then the projective camera model is applied on the undistorted images. To undistort an image efficiently, a lookup table can be used, which stores for every pixel in the undistorted image the position of the corresponding pixel in the distorted image.

Software packages like [45] or [13] exist, which use images of a reference grid to calibrate a camera and estimate both the intrinsic parameters and the radial distortion coefficients. Another method [25] to estimate the radial distortion coefficients uses the fact that straight lines in the real world always need to be straight lines in a projective image. Thereby the distortion parameters are estimated by minimizing the deviation of straightness.

### 2.1.3. Camera Pose Parameterization

For the camera pose estimation, the extrinsic camera parameter matrix $\left[R|\boldsymbol{t}\right]$ needs to be parameterized, so that it only depends on a minimum amount of degrees of freedom. As the 3-dimensional translation vector $\boldsymbol{t}$ represents 3 degrees of freedom of the camera pose, the parametrization is straightforward. As the $3 \times 3$ matrix $R$ consists of 9 elements, but only depends on 3 degrees of freedom, the parametrization of $R$ is more difficult to do well.

To ensure that the rotation matrix $R$ represents a rotation in $\mathbb{R}^3$, all the column vectors must be of unit length (3 constraints) and they must be mutually orthogonal (3 more constraints). The fact that $\det R = 1$ results from these constraints.

Several parameterizations of a rotation matrix in $\mathbb{R}^3$ exist which are useful for different purposes. An extensive description about different parameterizations of a ration matrix can be found in [106]. We describe the parametrization with Euler angles, quaternions, and the axis angle representation of a rotation in the the following sections.

#### Euler Angles

An Euler angle is the rotation around one of the coordinate axes. The rotation matrix $R$ can be composed by the three rotations around all coordinate axes. If $\alpha$ is the rotation around the $x$-axis, $\beta$ around the $y$-axis and $\gamma$ around the $z$-axis, the rotation matrix $R$ can be computed by

$$R = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.13}$$

The major drawback of using Euler angles is the fact that one rotation parameter gets lost, if two of the three rotation axes align. This problem is known as gimbal lock. Since the singularities occur typically for angles at $\pi/2$, it is not always possible to limit the legal range of rotation. Another drawback is that the interpolation between Euler angles produces poor results, because all three axes are interpolated separately. An advantage of Euler angles is the fact that they can provide an easy interface in the form of three different sliders, i.e. for virtual reality authoring tools.

#### Quaternions

A quaternion $q$ is a hyper complex number that can be written as $q = q_x + q_y i + q_z j + q_w k$ with $i^2 = j^2 = k^2 = ijk = -1$. Quaternions form a ring in a four dimensional vector space, which is closed under the multiplication operator. A unit quaternion

$$\tilde{q} = \cos\left(\frac{\theta}{2}\right) + \boldsymbol{\omega}\sin\left(\frac{\theta}{2}\right) \tag{2.14}$$

with $\|q\| = 1$ can be used to represent a rotation in $\mathbb{R}^3$ around the unit vector $\boldsymbol{\omega}$ by the angle $\theta$. The rotation of a point $\boldsymbol{x} \in \mathbb{R}^3$ can be carried out by using the quaternion multiplication

$$\text{rotate}(x) = q \circ \tilde{x} \circ \bar{q}, \tag{2.15}$$

where $\circ$ is the quaternion multiplication operator, $\bar{q}$ the conjugate of $q$ and $\tilde{x}$ is the vector $x$ extended with a zero scalar component. The major advantage of the rotation representation with a quaternion is that it overcomes the problem of singularities. Therefore quaternions are widely used for smoothly interpolating between rotations.

A problem, however, is that the rotation representation with a quaternion is over-parametrized, since a rotation in $\mathbb{R}^3$ has only 3 degrees of freedom. If some target function is minimized over the quaternion parameters, it has to be ensured that $\|q\| = 1$ with an additional constraint. Solving optimization problems with the use of unit quaternions is therefore a computational overhead and an increase in code complexity.

### Exponential map

The representation of a rotation with the exponential map, also known as axis angle representation, parameterizes the rotation matrix $R$ by a 3D-vector $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^T$. The axis around which the rotation is performed is given by the direction of $\boldsymbol{\omega}$ and the angle of the rotation is represented by $\theta = \|\boldsymbol{\omega}\|$.

The exponential map owes its name to the fact that the rotation matrix $R$ can be represented by the following infinite series expansion on an exponential:

$$R = \exp(\Omega) = \mathbb{I} + \Omega + \frac{1}{2!}\Omega^2 + \frac{1}{3!}\Omega^3 + ... \quad , \tag{2.16}$$

where $\Omega$ is the skew-symmetric matrix

$$\Omega = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix}. \tag{2.17}$$

With the help of the Rodrigues' formula the rotation matrix $R$ can be computed by

$$R = \exp(\Omega) = \mathbb{I} + \frac{\sin\theta}{\theta}\Omega + \frac{1 - \cos\theta}{\theta^2}\Omega^2. \tag{2.18}$$

The advantage compared to the quaternion representation is that the rotation is represented by only three parameters and no additional constraint is needed during an iterative optimization. Singularities are only at angles of $2n\pi$ with $n = 1, 2, 3, ....$ Luckily these singularities can be avoided by restricting the angle $\theta = \|\boldsymbol{\omega}\|$ in the range of $-\pi$ to $+\pi$.

As the exponential map representation is not over-parametrized and has only singularities in a region of the parameter space, which can easily be avoided, it is the most practical parametrization of a rotation matrix for the purpose of camera pose estimation.

## 2.2. Camera Calibration

The idea of camera calibration is to estimate all the parameters of a camera model. Here it is not assumed that intrinsic parameters of the camera are known. The problem of estimating only the external camera parameters is denoted as camera pose estimation and is described in the next section.

A very common approach to estimate all elements of the whole projection matrix

$$P = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \tag{2.19}$$

of a perspective camera is the DLT algorithm [1]. This method solves a linear system of equations, which relate a set of 3D coordinates $M_i = (X_i, Y_i, Z_i)^T$ and their corresponding projected 2D points $m_i = (x_i, y_i)^T$ in the image. Each correspondence results in two linearly independent equations:

$$x_i = \frac{p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \tag{2.20}$$

$$y_i = \frac{p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \tag{2.21}$$

If $\boldsymbol{p}$ is a vector of all coefficients of $P$, the equations can be rewritten in the form $A\boldsymbol{p} = \mathbf{0}$, where $A$ is a $2n \times 12$ matrix and $n$ the number of correspondences of 3D coordinates and 2D image points. The elements of the projection matrix $P$ can be computed by using a singular value decomposition of $A$. At least $n = 6$ correspondences are necessary to solve the system of equations. The internal and external parameters can be extracted from $P$ by using a QR decomposition, which results in an upper triangular matrix $K$ and an orthonormal matrix $R$.

The results of a linear method like DLT is often used as initialization for a further refinement with a non-linear iterative minimization method like Levenberg-Marquard, which is described in Section 2.3.3.

Many calibration tools use any kind of calibration pattern for the detection of 2D points in an image. The Camera calibration Toolbox for Matlab [13] or OpenCV [45] use a check board pattern to detect very precise 2D points and to create 2D/3D-correspondences. ArToolKit uses centroids of circular features. Another option to create a set of 2D/3D point correspondences is the usage of fiducial markers, as described in Chapter 3.

For a precise intrinsic calibration often many images of a calibration pattern are used. Thereby the estimation of the intrinsic camera parameters $K$ and the extrinsic camera parameters of every single picture $[R|t]_j$ is formulated as one problem, which is solved iteratively by a non-linear minimization.

## 2.3. Camera Pose Estimation

If the intrinsic parameters are known, the calibration process is reduced to estimating the extrinsic camera parameters, which are also denoted as the camera pose. With a given set

of $n$ correspondences between 3D world coordinates and 2D image points, the six degrees of freedom of the camera pose shall be estimated. This problem is often referred to as the Perspective-n-Point (PnP) Problem. It is also possible to use the DLT algorithm for estimating only the extrinsic parameters by simply multiplying the estimated $P$-matrix with $K^{-1}$, i.e. $[R|r] \sim K^{-1}P$, but the results are not very stable, since the problem is over-parametrized.

The problem of estimating the camera pose has been extensively studied in the literature. The methods can be classified into two categories, into iterative and non-iterative approaches. Whereas the non-iterative methods are often used to estimate the pose without any prior knowledge, i.e. for the camera pose initialization, purely iterative methods need a first guess of extrinsic camera parameters. These iterative methods are widely used for a refinement step of the camera pose or a frame-to-frame tracking.

### 2.3.1. Iterative Methods

All the iterative methods [62, 65, 24] usually define an error function depending on a given camera pose and minimize these error functions iteratively. The error function can be either defined in image space or object space. Many image space methods minimize the squared projection error over the extrinsic camera parameters $R$ and $t$:

$$[R|\boldsymbol{t}] = \arg\min_{[R|\boldsymbol{t}]} = \sum_{i}^{n} \|f(\boldsymbol{M}_i) - \boldsymbol{m}_i\|^2, \tag{2.22}$$

where $f$ is a function depending on $[R|t]$ which projects a 3D point into the image space. Non-linear minimization methods as described in section 2.3.3 are then used to find a solution.

Lu et al. describe a method in [65] which uses the following error minimization in object space:

$$[R, \boldsymbol{t}] = \arg\min_{[R|\boldsymbol{t}]} = \sum_{i}^{n} \|(I - V_i)(RM_i + t)\|^2, \tag{2.23}$$

where $V_i$ is the observed line-of-sight projection matrix defined as:

$$V_i = \frac{m_i m_i^T}{m_i^T m_i}. \tag{2.24}$$

The authors showed that their method is very accurate and computationally efficient compared to other iterative algorithms.

A very popular way to solve the pose estimation problem was presented by DeMenthon and Davis [24]. Their method, called POSIT, first computes an approximate solution by solving a linear system using the scaled orthographic projection model, then the camera pose is iteratively refined. A problem of this approach is that it cannot be applied when the points are coplanar. In [83] a similar approach is described, which handles the coplanar case. However, these two cases have to be explicitly distinguished.

SoftPOSIT [21] is another interesting method which not only handles the extrinsic camera parameters estimation, but also the determination of the correspondences. This can be useful for problems, where the connection between 3D points and 2D points is ambiguous.

### 2.3.2. Non-iterative Methods

The non-iterative approaches rely on first estimating the depth and the 3D positions $M_i^C$ of a feature point in the camera coordinate system. Then the rotation $R$ and translation $t$ from the world coordinate system to the camera coordinate system can be easily retained from aligning the points $M_i$ on $M_i^C$ with a closed-form solution [43]. Non-iterative methods usually have a high complexity, which means that they are only fast for a small number of correspondences $n$, but become very slow for a larger $n$. To overcome this problem a very efficient and accurate non-iterative algorithm was developed by Moreno et al. [30]. Their central idea is to express the $n$ 3D points as a weighted sum of four virtual control points and solving in terms of their coordinates. Thereby the complexity is reduced to $O(n)$.

As non-iterative methods do not rely on any initial guess, they are often used to compute an initial estimate of the camera pose. Iterative methods are more accurate and can be taken to refine the estimation result.

### 2.3.3. Non-linear Minimization

Often the error function which is minimized iteratively to estimate a camera pose is of a non-linear nature. This is also the case, when the camera rotation is parameterized with the axis/angle representation as described in Section 2.1.3. Non-linear minimization methods are then necessary to compute an accurate estimate of the camera pose.

Let $g(\boldsymbol{p})$ be the error function which depend on the extrinsic camera parameter vector $\boldsymbol{p}$. All the algorithms start with an initial estimate $\boldsymbol{p}_0$ and find a minimum by iteratively updating the camera pose by

$$\boldsymbol{p}_{i+1} = \boldsymbol{p}_i + \boldsymbol{\Delta}_i, \tag{2.25}$$

where $\boldsymbol{\Delta}_i$ is an update difference of the camera pose, which decreases the value of the error function $g$ in every iteration.

#### Newton's Method

The Newton's method, also called Newton-Raphson method, is an algorithm for finding roots of a real-valued function. It relies on a Taylor expansion of the function $g$:

$$g(\boldsymbol{p} + \boldsymbol{\Delta}) \approx g(\boldsymbol{p}) + J_g(\boldsymbol{p})\boldsymbol{\Delta} + \frac{1}{2}\boldsymbol{\Delta}^T H_g(\boldsymbol{p})\boldsymbol{\Delta}, \tag{2.26}$$

where $J_g(\boldsymbol{p})$ is the Jacobian and $H_g(\boldsymbol{p})$ the Hessian of $g$. A minimum of $g$ can be found, where the deviation of the right hand side of the above equation vanishes, i.e.

$$J_g(\boldsymbol{p}) + H_g(\boldsymbol{p})\boldsymbol{\Delta} = \mathbf{0}. \tag{2.27}$$

The difference of an iteration step can be computed by

$$\boldsymbol{\Delta} = -(H_g(\boldsymbol{p}))^{-1} J_g(\boldsymbol{p}). \tag{2.28}$$

The Newton method has a quadratic convergence when it is close to a solution, however, it can fail to converge if the initial value is too far from the true minimum. Another drawback is that the computation of the Hessian $H_g$ is often expensive and sometimes not possible.

### Gauss-Newton Algorithm

The Gauss-Newton method is an algorithm for finding local extrema of a function. It does not require the computation of a Hessian matrix. However, this method can only be used for optimizing a squared error function.

The difference of an iteration step can be written as

$$\boldsymbol{\Delta} = -(J_g^T(\boldsymbol{p})J_g(\boldsymbol{p}))^{-1}J_g(\boldsymbol{p})^T g(\boldsymbol{p}) = -J_g^+(\boldsymbol{p})g(\boldsymbol{p}), \tag{2.29}$$

where $J_g^+$ is the pseudo-inverse of $J_g$.

The Gauss-Newton method can be regarded as an approximation of Newton's method, especially if the values of $\|g(\boldsymbol{p})\|$ are small.

### Gradient Descent

The Gradient descent, also denoted as method of steepest descent, is a minimization method, where in every iteration a step into the direction of the negative gradient is performed. The increment $\boldsymbol{\Delta}$ of an iteration can be computed by

$$\boldsymbol{\Delta} = -\alpha J_f(\boldsymbol{p}), \tag{2.30}$$

where $\alpha$ is the step size, which is set to be a small constant in the simplest case. The algorithm always converges, but it can take many iterations to converge toward a local minimum.

### Levenberg-Marquardt

The Levenberg-Marquard method is a slight modification of the Gauss-Newton method. The increment of the estimated parameter vector can be estimated by

$$\boldsymbol{\Delta} = -(J_g^T(\boldsymbol{p})J_g(\boldsymbol{p}) + \lambda I)^{-1}J_g(\boldsymbol{p})^T g(\boldsymbol{p}). \tag{2.31}$$

The additional term $\lambda I$ is used to stabilize the convergence behavior. If the error function is decreased, the value $\lambda$ is reduced and the inclement is accepted. Otherwise the value $\lambda$ is increased. This makes the algorithm more robust, but results in a slower convergence.

For many non-linear least square problems the Levenberg-Marquardt method is widely used, because it is more robust, but has a similar convergence speed to the Gauss-Newton method.

### 2.3.4. Robust Estimation

Occlusions, reflections or small changes in a scene can often result in tracking failures of single image features. If such features are not tracked or detected correctly, there is always a presence of ambiguous or very inaccurate 2D/3D-correspondences. These spurious measurements will have a great influence on the estimated camera pose, if purely the squared projection error of all correspondences is minimized. Therefore a detection of outliers of incorrect measurements is indispensable for a robust pose estimation. Two widely used methods to reduce the influence of false measurements are the M-estimators and RANSAC.

The M-estimator method is more accurate, but an initial estimate is required. The RANSAC approach does not need an initial guess, but results are less precise.

#### M-estimators

With the robust estimation technique called M-estimators, it is possible to reduce or neglect the influence of spurious data in a least-squares minimization problem. Instead of minimizing the squared residuals $\sum_i r_i^2$, the error function to be minimized is replaced by

$$\sum_i \rho(r_i), \tag{2.32}$$

where $\rho$ is the so-called estimator function. The estimator function must be symmetric, continuously differentiable and and it must have a unique minimum at zero. A description of several estimator functions can be found in [122]. One of the widely used estimator functions for robust camera pose estimation [112] is the Tukey estimator [111], which is defined by

$$\rho_{\mathrm{Tuk}}(x) = \begin{cases} \frac{c^2}{6}[1-(1-(\frac{x}{c})^2)^3] & \text{if } |x| \leq c \\ \frac{c^2}{6} & \text{if } |x| > c\ , \end{cases} \tag{2.33}$$

where $c$ is a threshold, which is usually chosen with respect to the standard deviation of the data. In figure 2.2 the Tukey estimator function is plotted together with a least-square estimator for comparison.

The effect of the Tukey estimator is that very small residuals are handled in a leased square sense and all values $x > c$ do not have any influence on the minimization result. These very large residuals can be regarded as outliers and are therefore completely rejected.

Instead of applying the estimator function of the projection error, it is also possible to implement the robust estimation as an iterated re-weighted least-square minimization. Details on how to compute the weights can be found in [122].

#### RANSAC

Another method for robust estimation called RANSAC was first presented by Fischler and Bolles [29]. From an observed set of data a smallest possible subset of samples is randomly

selected and used to estimate the model parameters. Then it is tested, if a certain amount of the other points also fits to the model. For a robust pose estimation this means that randomly four 2D/3D-correspondences are selected and a linear method like [24] or [83] is applied to estimate a camera pose. All other 3D points of the correspondences are then projected with that camera pose into the image, and it is tested how many correspondences exist which have a smaller re-projection error than a certain threshold. Such correspondences are called inliers. All other correspondences, where the re-projection error is too big, are called outliers. If the amount of inliers is not big enough, a camera pose is estimated with another random subset of correspondences and it is tested again how many inliers exist. This process is iterated until the amount of inliers exceeds a threshold or if a maximum number of iterations is reached. The RANSAC method is usually slower and less accurate than the M-estimators method, but the advantage is that no initial estimate of the pose is needed. If the RANSAC method has been applied successfully, the pose can be refined by applying a non-linear method on all inliers.

PROSAC [18] is derivative of RANSAC, where the selection of the samples for a subset is not performed randomly, but by some quality measure of the correspondences. Top rated samples are selected with a much higher probability. The benefit of PROSAC is a performance increase, because much less iterations are needed, until enough inliers are found. The quality of a correspondence can be determined by the tracking or detection success of previous frames.

### 2.3.5. Bayesian Tracking

Camera pose estimation can also be performed with a probabilistic method called Bayesian estimation [31]. A Bayes filter estimates a dynamic system's state recursively over time using incoming noisy observations. For a Bayesian camera tracking the state $\boldsymbol{s}_t$ describes a probability distribution of the estimated camera pose. It can be simply the position and the orientation of the camera or, in addition, variables such as the translational and angular velocities.

The state probability density $p(\boldsymbol{s}_t)$ is conditioned on all available sensor data $\boldsymbol{z}_0, ..., \boldsymbol{z}_t$ available at time $t$:

$$p(\boldsymbol{s}_t) = p(\boldsymbol{s}_t | \boldsymbol{z}_0, ..., \boldsymbol{z}_t) \tag{2.34}$$

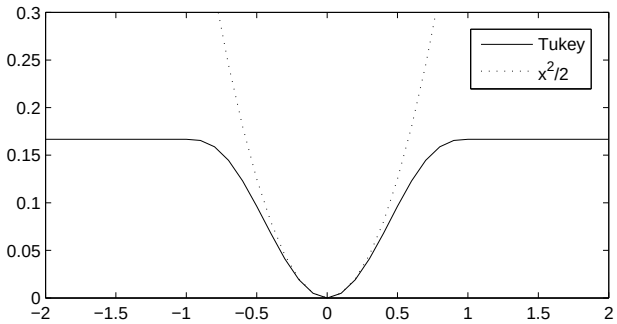



Figure 2.2.: Graph of the Tukey estimator function $\rho_{\text{Tuk}}(x)$ with $c = 1$.

The observations $\boldsymbol{z}$ can be for example image feature locations or measurements of an inertial sensor. A propagation rule for the state probability density can be written as

$$p(\boldsymbol{s}_t) = \frac{p(\boldsymbol{z}_t|\boldsymbol{s}_t)p(\boldsymbol{s}_t|\boldsymbol{z}_0, ..., \boldsymbol{z}_{t-1})}{p(\boldsymbol{z}_t|\boldsymbol{z}_0, ..., \boldsymbol{z}_{t-1})}. \tag{2.35}$$

As the state can be assumed to be a Markov process, the true state $\boldsymbol{s}_t$ only depends on the previous state $\boldsymbol{s}_{t-1}$ and the term $p(\boldsymbol{s}_t|\boldsymbol{z}_0, ..., \boldsymbol{z}_{t-1})$ can be estimated by

$$p(\boldsymbol{s}_t|\boldsymbol{z}_0, ..., \boldsymbol{z}_{t-1}) = \int p(\boldsymbol{s}_t|\boldsymbol{s}_{t-1})p(\boldsymbol{s}_{t-1})d\boldsymbol{s}_{t-1}. \tag{2.36}$$

Equation (2.36) can be regarded as a prediction step, which applies the motion model on the previous camera pose probability distribution $p(\boldsymbol{s}_{t-1})$. The update step of the filter, which corrects the predicted estimate using the sensor measurements, is described by equation (2.35).

The camera pose distribution can be represented by a Gaussian, a mixture of Gaussians or a set of particles. Using a Gaussian distribution for the camera pose probability leads to the Kalman filter, which is only able to handle a single hypothesis for the camera pose. A mixture of Gaussians or a particle filter can represent the camera pose with a more general distribution, but with the disadvantage of much higher computational costs.

### Kalman Filter

The Kalman filter is a widely used tool for camera pose estimation and the fusion of several measurement sources [42, 11]. A very detailed introduction to the Kalman filter can be found in [7] or [115]. Here only a very coarse outline of the Kalman Filter is presented.

For the linear case the measurements $\boldsymbol{z}_t$, such as image feature positions, are related to the state $\boldsymbol{s}_t$ by

$$\boldsymbol{z}_t = C\boldsymbol{s}_t + \boldsymbol{v}_t, \tag{2.37}$$

where the matrix $C$ represents a linear transformation relating the measurements to the state. The vector $\boldsymbol{v}_t$ stands for the measurement noise.

The prediction step of the Kalman filter computes the a priori state estimate $\boldsymbol{s}_t^-$ and its covariance matrix $S_t^-$ by

$$\boldsymbol{s}_t^- = A\boldsymbol{s}_{t-1}, \tag{2.38}$$

$$S_t^- = AS_{t-1}A^T + Q, \tag{2.39}$$

where $A$ is the state transition matrix describing the dynamic of the model and the matrix $Q$ represents the process noise covariance. The prediction step corresponds to equation (2.36) of the Bayes filter.

The a posteriori state estimate $\boldsymbol{s}_t$ and its covariance matrix $S_t$ are estimated by the following update step:

$$\boldsymbol{s}_t = \boldsymbol{s}_t^- + G_t(\boldsymbol{z}_t - C\boldsymbol{s}_t^-), \tag{2.40}$$

$$S_t = (I - G_tC)S_t^-, \tag{2.41}$$

The Kalman gain $G_t$ determines the amount of influence of a measurements and is computed by

$$G_t = S_t^- C^T (C S_t^- C^T + R)^{-1}, \tag{2.42}$$

where the covariance matrix $R$ represents the measurement noise.

The relation between measurements like image feature positions and the state, i.e. the six degrees of freedom of the camera pose, is, however, not linear for a fully projective camera model.

Therefore equation (2.37) has to be replaced by

$$\boldsymbol{z}_t = c(\boldsymbol{s}_t, \boldsymbol{v}_t) \tag{2.43}$$

with the non-linear function $c$. Linearizing this function by its first order Taylor approximation leads to the Extended Kalman Filter (EKF). The update step can then be written as

$$\boldsymbol{s}_t = \boldsymbol{s}^- + G_t(\boldsymbol{z}_t - c(\boldsymbol{s}_t^-, \boldsymbol{0})), \tag{2.44}$$
$$S_t = (I - G_t J_c) S_t^-, \tag{2.45}$$

where the Kalman gain $G_t$ is computed by

$$G_t = S_t^- J_c^T (J_c S_t^- J_c^T + J_v R J_v^T)^{-1}. \tag{2.46}$$

The matrices $J_c$ and $J_v$ are the Jacobians of the function $c$ with respect to the state $\boldsymbol{s}$ and the measurement noise $\boldsymbol{v}$ respectively. Usually the identity matrix is taken for the Jacobian $J_v$.

The linearization can be regarded as one iteration step of the Gauss-Newton iterative minimization. To increase the accuracy of the measurement update step, the equations (2.44) can be applied iteratively several times. This method is denoted as the Iterated Extended Kalman Filter.

Another method to increase the accuracy in a non-linear system it the Unscented Kalman Filter [51]. The mean and the covariance of a Gaussian distribution are here represented by a minimal set of carefully chosen sample points. These sample points are propagated through the true non-linear system and result in a posterior mean and covariance with a much higher accuracy than the Extended Kalman Filter.

With the aid of the Kalman filter several measurement sources can be fused into the estimation of the camera pose [42]. It is possible to make predictions of the camera pose, if measurements are not available due to occlusion or strong motion blur. With the prediction step it is also possible to compensate the latency of the computation time of the whole camera pose estimation, which is necessary for real-time see-through augmented reality applications.

However, only one hypothesis is considered with the the Kalman filter, which can be an insufficient representation of the camera pose probability distribution in ambiguous cases. A possible solution is the use of multiple weighted Kalman filters, where the probability distribution can be regarded as a Mixture of Gaussians [15].

### Particle Filter

Particle Filters, also known as Sequential Monte Carlo methods, are a more general approach to estimate a probability distribution of a dynamic system's state. The probability distribution is here represented by a set of weighted particles. Isard et. al. [46] presented a method called Condensation, which uses a particle filter for tracking contours. Approaches exist, in which particle filters are taken to estimate the camera pose using point features [87] or line features [55].

No linearization of the function, which relates the state and the measurements, is needed. This makes the particle filters a very general method, which is easy to apply on any non-linear function. However, a major drawback are the computational costs, since many particles are needed for a sufficient representation of a distribution. Therefore in many real-time applications the Kalman filter is preferred, if it can be assumed that the distribution is of a Gaussian nature.

# 3. Marker-Based Tracking

The main problem of a camera based pose estimation is the detection of image features and the creation of correspondences between 2D image features and their 3D coordinates. If enough such 2D/3D correspondences exist, the camera pose can be easily estimated with techniques described in section 2.3. To simplify the feature detection process and the creation of 2D/3D correspondences, artificially designed fiducials, also called markers, are used. These markers must be able to get detected easily with basic image processing algorithms, and they must also carry some information, which makes it possible to uniquely distinguish them among each other. The exact 3D position of every fiducial point has to be known to create the correspondences between 3D coordinates and image features.

A marker can be designed to detect only a single feature point in an image or to detect a planar region, which in most cases is a square, where the four corners are taken as image feature points. As such a planar marker can be used to create four 2D/3D correspondences, it is possible to estimate a camera pose with a single planar marker.

## 3.1. Point Fiducials

As point fiducials can be easily detected with a high subpixel accuracy, they have been widely used to track objects, human bodies or interaction devices. Active and passive point fiducials can be distinguished. Active markers are self-emitting light sources, which require some external power supply, whereas passive markers are detected only by reflected or scattered light.

### 3.1.1. Active Point Fiducials

Light Emitting Diodes (LEDs) have a high brightness intensity in contrast to the rest of the scene and can therefore be easily spotted in a camera image. Often infrared LEDs are used, because they are not visible by the user and do not interfere with other light sources. Furthermore, CMOS sensors used in consumer cameras are very sensitive to infrared light, which makes IR-LEDs easy to detect.

The HiBall Tracker [116] is an inside-out tracking system for virtual and augmented reality applications, where arrays of infrared LEDs are used to estimate the position and orientation with high accuracy and high performance. Another widely used inside-out tracking system is the controller of the Nintendo Wii, which uses an infrared camera in the controller to track an array of IR-LEDs.

Outside-in tracking systems like the one presented in [68] use a stereo camera setup where 3D positions of LEDs are reconstructed by epipolar constraints and then used to estimate the orientation and position of a head-mounted display or an interaction device.

If colored LEDs are used, it is possible to take the colors to distinguish between the different LEDs. A far more sophisticated method is to encode the ID of an infrared LED by frequency or amplitude modulation [79].

### 3.1.2. Passive Point Fiducials

Passive point markers do not depend on any power supply and are therefore less intricate for setting up a tracking system. A widely used method is to create markers with retro-reflective materials [90] and to use a directed infrared ring flash to illuminate the scene. Due to their reflective material properties, the fiducials stand out from the rest of an image taken with an infrared camera and can therefore be detected easily. By minimizing the epipolar constraints in two camera images the 3D position of such a fiducial point can be calculated. The asymmetric composition of several markers on an interaction device makes it possible to estimate its position and orientation in the 3D scene. Tracking systems based on this method are commercially available from companies like Advanced Realtime Tracking or Vicon. A similar approach for detection 2D positions in an infrared image is also used for finger tracking on a multi-touch screen [36].

Passive point fiducials can also be detected in the visible range of light. The many fiducial designs among others include black and white concentric circles [72], coloured concentric circles [17] and circular ring codes [78]. Another interesting marker design was presented by Bencina and Kaltenbrunner [9]. They segment an image into a tree of alternating black and white regions which encodes the ID of a marker. With their approach not only the position but also the 2D orientation of the marker in an image can be detected. These markers are used for detecting objects on a table based interactive surface.

## 3.2. Planar Square Fiducials

The benefit of using planar markers is that not only one 2D position of the marker center is detected, but the four corners of a marker square. With four correspondences it is possible to estimate the pose of a calibrated camera with only a single marker. The detection process of a marker can be split into two steps: The extraction of the four corners of a marker square and the detection of a marker ID.

### 3.2.1. Marker Square Extraction

ARToolkit [52] is a very popular library for detecting planar markers. It is freely available and was therefore widely used to create AR applications. The marker fiducials consist of a black border on a white background containing a black/white image. To detect a marker, first the input image is binarized, and this thresholded image is then used to detect the black border of a marker with a contour following algorithm. If closed loops of contours

are detected, a shape analysis of these contours is performed to identify square-like shapes. Therefore the four corner points are extracted approximately by searching for points on the contour with the furthest distance to a given other point. A more precise sub-pixel position is computed by the intersection of lines fitted through the contour segments of the marker edges.

A drawback with the binarization is that a fixed global threshold does not always result in a clear image, from which the black border of a marker can be extracted. To solve this problem, the threshold can be adapted to the brightness of a region of interest in the video image [84]. In [78] a method to extract marker contours in scenes with non-uniform lighting is presented, where a threshold is not applied on the image itself, but on the gradient of a logarithmic contrast enhanced image. Thereby markers can be detected both in very bright and very dark regions of an image with the same threshold.

### 3.2.2. Marker Identification

In [52] the interior of a marker consists of a black/white image template. If the four corners of the marker border have been detected, the homography $H$ can be estimated that maps the image template coordinates $\tilde{\boldsymbol{m}}_t$ on the camera image coordinates $\tilde{\boldsymbol{m}}'$ by

$$\tilde{\boldsymbol{m}}' = H * \tilde{\boldsymbol{m}}_t. \tag{3.1}$$

A correlation with the template image and the corrected interior of the marker image is performed and tested if the two images coincide. Because the detection must be rotation invariant, the template image is rotated in 90ř steps and then also correlated with the extracted marker image.

Instead of a template image, Fiala [28] used a black/white $6 \times 6$ pattern to describe a unique marker ID. He uses digital coding theory with techniques of checksums and forward error correction. Markers with such an identification code have a better inter-marker confusion rate than ARToolkit.

A similar tracking library for PDAs and smartphones called ARToolKitPlus [114] was presented by Wagner. He also uses a binary code similar to [28] for the detection of the marker ID.

# 4. Edge-Based Tracking Methods

Although fiducial marker can be detected and tracked reliably and augmented reality applications can be created very easily with software packages like ARToolKit, in many scenarios, especially in industrial setups, the preparation of a scene with artificial markers can be very intricate or sometimes not possible at all. For scenarios like industrial maintenance or outdoor environments a tracking method should only rely on natural features like contours, straight lines or distinct points. In recent years there has been a lot of research interest in the area of markerless tracking. The markerless tracking methods can be categorized in model-based methods and model-free methods. The model-based methods rely on some 3D knowledge of the scene, whereas model-free methods only use information for the camera pose estimation, which is gathered during the tracking. In this chapter the model-based methods are described.

Model-based methods can either rely on a 3D line model or a polygonal model. With a given 3D line model, edge-based tracking algorithms are often used. A textured 3D model or simply a reference image can be used for texture-based tracking methods. A given 3D polygonal model can also be used to estimate the depth of a detected feature point. A benefit of using a model is that the tracking cannot accumulate drift, and the camera pose is always estimated in the coordinate system of the given model. Placing virtual augmentations in the real scene is therefore easy, because they can be set up in the coordinate system of the reference model.

The very first tracking approaches all rely on edges, mostly because in contrast to texture-based methods, they are computational less expensive, and are therefore able to run in real-time on standard hardware of the nineteen-nineties. Edges are also very stable to a wide variety of transformations, illumination changes and reflecting materials and are thus a good choice for tracking industrial scenarios, where not many planar textures objects exist. The edge-based tracking methods can be split into two categories: The first group of approaches first extracts lines and then fits a given 3D model to the extracted lines. The other methods do not explicitly extract line features, but search for image gradient maxima along lines perpendicular to a regarded edge.

## 4.1. Explicit Line Extraction

The methods described in this section all rely on extracting line or contour features in an image and then match these features to a given model to detect an object or estimate the camera pose, with which the image was taken.

Lowe [63] presented an approach where not only the camera pose but additional parameters of the given 3D model can be estimated. To extract line features in the image, first

a Laplacian filter is applied on a special hardware board and zero crossings are analyzed to detect edges in the image. A Canny hysteresis thresholding is performed to create a 8-connected list of edge points. The resulting contour is split into straight line segments by applying a scale invariant recursive subdivision algorithm. Probabilities of a match between model edges and extracted lines are calculated according the their perpendicular distance, relative orientation and model covariance. These probabilities are then used to guide the search for a best match of image lines and model edges. When correspondences between 2D lines and 3D lines are established, the camera pose and the model parameters are updated by the result of a Gauss-Newton minimization of projected model edges and image lines.

Gennery [32] also uses a Sobel-like hardware edge detector to compute an edge map. A Kalman filter which models the position, orientation, linear velocity and angular velocity of the camera is used to predict a projection of the 3D model in the image. Matches between detected 2D lines and predicted projections of 3D lines are created by a search at control points on an edge along the vertical or horizontal direction, whichever is closer to the perpendicular. Detected measurements are weighted according to their quality, which originates from the distance and the orientation deviation.

A generic 3D vehicle model parameterized by 12 length parameters was used in [57] to detect and track moving vehicles in an image sequence. Line segments of detected image edges and projected model lines are described by their position, length and orientation. The mahalanobis distance between extracted segments and model segments is computed to find a closest match. A Levenberg-Marquard minimization is used to estimate the model parameters and the camera pose iteratively, until a stable solution is found.

A similar approach was presented by Ruf et. al. [94] to control a robot arm. Predictions are made by joint angle measurements from the robot. In addition to the tracking an on-line calibration of the kinematic chain is performed.

In [58] the extraction of line features is performed with a Hough transform. Predictions and updates of the moving object's state are also done with a Kalman filter. To speed up the detection process, the extraction of lines in an image is limited to the uncertainty region of the predicted model lines.

A solution for the simultaneous determination of the camera pose and line correspondence was presented by David et. al. [20]. Their method relies on the soft assignment of correspondences called SoftPOSIT, which was first presented in [21] for point correspondences. The camera pose can here be determined not by non-linear minimization, but by solving a linear system of equations.

## 4.2. Line Model Registration

One of the first model-based 3D tracking systems called *RAPiD*, was presented by Harris [38, 37]. His system was the first to be able to run in real-time. The tracking method is computationally very efficient, because the explicit line feature extraction is avoided, and the image is only examined where edges are expected to be found. A Kalman filter, which represents the six degrees of freedom of the parameters, i.e. the position and the

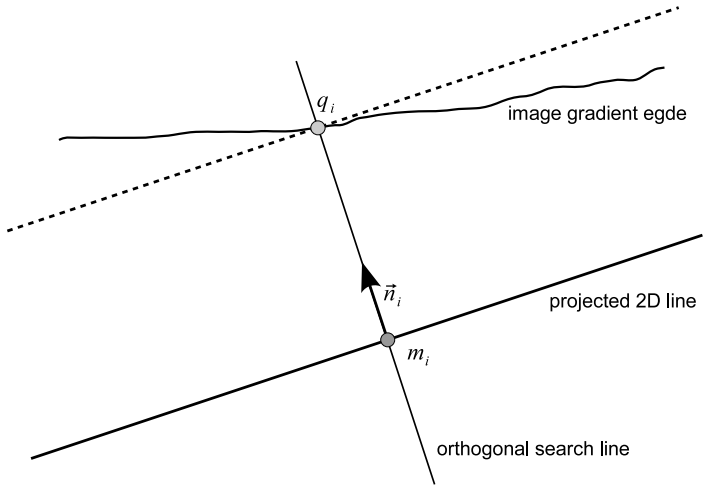



Figure 4.1.: Principle of the orthogonal search for gradient maxima. At the control points $\boldsymbol{m}_i$ on the projected line a search for gradient maxima along the normal direction $\vec{n}_i$ is performed. If a point $\boldsymbol{q}_i$ on a gradient image edge is found, minimizing the distance to a parallel line through this point is used to estimate the camera pose.

orientation of an object, is used to predict the new position of an object in the current image. At control points along a projected edge in the image a local search for gradient maxima along a one-dimensional search line is performed. The direction of the search line is perpendicular to the projected direction of the regarded model edge. The RAPiD system only searches in horizontal, vertical and diagonal directions for computational efficiency. When the first gradient maxima which is bigger than a predefined threshold, is found on the search line, this 2D point is regarded as a point of the edge on which the line model shall be aligned. In Figure 4.1 the principle of the perpendicular search is pointed out. With the measured differences between the image edges and projected model edges, the camera pose is updated by minimizing these errors.

Many modern model-based tracking methods still rely on the principles of RAPiD, because the overall computation is much more efficient than the explicit extraction of line segments. The model-based tracker developed in this thesis also uses the techniques of one-dimensional search at control points to create correspondences between 2D edges and 3D model lines.

For each frame, the pixel values on the search line for a considered point are obtained in two distinct steps. Similar as presented in [19], first the image is filtered at all pixel positions on a perpendicular search line with a 2D anisotropic Gaussian mask, whose major axis is axially parallel to the regarded edge. The pixel values are thereby smoothed only in the direction parallel to the edge and the variation of the pixel values along the search line is kept sharp. This filtering helps to make the search for gradient maxima robust against image noise without blurring the intensity values along the search line. To save computational costs, the Gaussian filter masks are precomputed for 180 angles. That mask whose major axis is most parallel to the projected edge is always taken into account. The smoothed pixel values on the search line are computed with subpixel accuracy. Then

the gradient is computed along the search line. Instead of using large precomputed filter masks as described in [19], we take advantage of the separability of the filter and first do a one-dimensional Gaussian smoothing parallel to the projected edge and then use a mono-dimensional convolution of the simple filter mask $\begin{pmatrix}-1 & 0 & 1\end{pmatrix}$ on the search line to compute the gradient with the obtained smoothed pixel values.

Maxima of the gradient on the search line can now be used as 2D measurements for the projected control points of a model edge. Either the largest gradient maximum within a certain search space can be considered as the desired 2D feature, or the closest maximum larger than a certain threshold.

Let $\mathbf{P}_{\boldsymbol{p}}$ be a function which projects the points $\boldsymbol{M}_i$, which have been sampled along the edge of the 3D line model into the 2D image points $\boldsymbol{m}_i$. The image points can be calculated by

$$\boldsymbol{m}_i = \mathbf{P}_{\boldsymbol{p}}(\boldsymbol{M}_i), \tag{4.1}$$

where $\boldsymbol{p}$ is the extrinsic camera parameter vector on which the projection function $\mathbf{P}$ depends. If $\boldsymbol{m}_i$ is the 2D position of a gradient maximum, the error to be minimized for the camera pose estimation can be described as

$$e = \sum_i \Delta(\boldsymbol{m}_i, \boldsymbol{q}_i), \tag{4.2}$$

where $\Delta$ is the distance function between the projected sample point $\boldsymbol{m}_i$ and the line through the edge feature point $\boldsymbol{q}_i$ in the image. The line through $\boldsymbol{q}_i$ is parallel to the projected line of the line model. The distance function can be written as:

$$\Delta(\boldsymbol{m}_i, \boldsymbol{q}_i) = ((\boldsymbol{q}_i - \boldsymbol{m}_i) \cdot (\boldsymbol{n}_i))^2, \tag{4.3}$$

where $\boldsymbol{n}_i$ indicates the 2D normal vector of the projected line. With respect to the aperture problem, here only the normal distance is regarded as a reprojection error of 3D model lines and image edges.

The camera pose is estimated by

$$\boldsymbol{p} = \arg\min_{\boldsymbol{p}} \sum_i \Delta(\mathbf{P}_{\boldsymbol{p}}(\boldsymbol{M}_i), \boldsymbol{q}_i). \tag{4.4}$$

A Levenberg-Marquardt minimization as described in Section 2.3.3 is used to estimate the extrinsic camera parameter vector $\boldsymbol{p}$.

## 4.3. Robust Camera Pose Estimation

However, because of ambiguities and false measurements, minimizing only the distances of control points to gradient maxima does not lead to a very stable pose estimation. False measurements, which can originate from occlusion, shadows, textures or edge ambiguities, have a high influence on the least-squares optimization result. Therefore a special treatment of outliers is indispensable for a robust camera pose estimation.

Armstrong and Zisserman [2] presented a RAPiD-like approach with a more robust pose estimation. Their method uses RANSAC [29] to detect outlying control point measurements. For a geometric primitive randomly chosen control points are evaluated and it is tested, how many other control points would fit into the solution. This process is repeated several times and the solution with the most support is finally adopted. All outlying control points are deleted and the camera pose is only computed by minimizing the reprojection error of the remaining control points.

As presented in [26] or [69] robust estimation of a camera pose can also be done by using M-estimators. An iterative re-weighted least squares solution is used to decrease the influence of outliers. Simon and Berger [102] present a robust method for tracking a known model of three-dimensional curves. The Huber [44] estimator function is applied on the error function of a local curve and outliers are rejected. In a second stage all remaining correspondences are used to estimate the pose again together with a global estimator function.

In our implementation [118] we use the Tukey estimator function [111] directly applied on the projection error for a robust outlier detection. This estimator function is also used in [113] for a robust camera pose estimation.

With the Tukey estimator function $\rho_{Tuk}$ the error to be minimized can be described by

$$e = \sum_i \rho_{\mathrm{Tuk}}(\Delta(\boldsymbol{m}_i, \boldsymbol{q}_i)). \tag{4.5}$$

Due to the nature of the Tukey estimator function as described in Section 2.3.4, outliers are rejected directly during the minimization and have no influence on the pose estimation result. Therefore the tracking system gets more robust against occlusion, shadows or falsely detected edge points.

## 4.4. Multiple Hypothesis Tracking

In many edge-based tracking approaches [2, 26, 69, 19] the largest gradient maximum within a certain search space is considered as the desired 2D feature. Another approach would be to take the first maximum larger than a certain threshold as the corresponding 2D point. However, it is never guaranteed that a nearby gradient maximum corresponds to the regarded edge.

A method to improve the tracking accuracy was presented by Berger et al. [10] In their approach the perpendicular distance of the search space is decreased by one pixel after every iteration step of the minimization process. The amount of outliers is thereby reduced, since the possible distance to the next nearby edge is decreased during the iterations. They showed that the minimization is less prone to local minima and false matches and that their method benefits from an increase in convergence and accuracy.

Vacchetti et al. [112] use not only one image edge point as measurement, but allow multiple hypotheses for potential edge-locations. During the orthogonal search not only the largest gradient maximum is taken as a point on the image edge, but several gradient

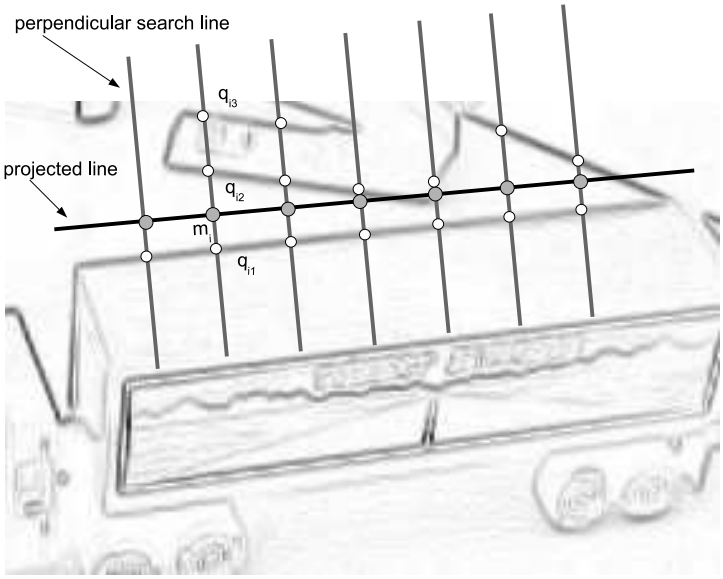



Figure 4.2.: Detection of multiple hypotheses of 2D points of a gradient edge. $\boldsymbol{m}_i$ are the control points on the projected line. The points $\boldsymbol{q}_{i,j}$ are several detected gradient maxima on the perpendicular search line. The gradient image is underlaid to visualize image edges.

maxima larger than a threshold are used as detected 2D edge feature points. During the minimization process, only the most probable hypothesis is taken into account for the distance measurement. The authors showed that considering multiple hypotheses prevents a wrong gradient maximum from being assigned to a control point on a line of the 3D model.

Figure 4.2 illustrates the detection of several hypotheses on the perpendicular search line. For every control point $\boldsymbol{m}_i$ several hypotheses of 2D points on an image edge are detected. Correspondences between a 3D control point $\boldsymbol{M}_i$ of a model line and multiple 2D edge feature points $\boldsymbol{q}_{i,j}$ are created. In [112] the estimator function $\rho^*_{\text{Tuk}}(\Delta_1, ..., \Delta_n)$ for multiple error measurements is defined as

$$\rho^*_{\text{Tuk}}(\Delta_1, ..., \Delta_n) = \min_j \rho_{\text{Tuk}}(\Delta_j), \tag{4.6}$$

where $n$ is the number of hypotheses and $\Delta_j$ is the distance of a point $\boldsymbol{m}$ to the $j^{\text{th}}$ hypothesis of a detected 2D point on an image edge. If $\boldsymbol{q}_{i,j}$ is the $j^{\text{th}}$ hypothesis for the $i^{\text{th}}$ sample point, the estimation error using multiple hypotheses can be computed by

$$e = \sum_i \rho^*_{\text{Tuk}}(\Delta(\boldsymbol{m}_i, \boldsymbol{q}_{i,1}), ..., \Delta(\boldsymbol{m}_i, \boldsymbol{q}_{i,n})). \tag{4.7}$$

As the estimator function $\rho_{\text{Tuk}}(\Delta)$ is non-decreasing for all $\Delta \geq 0$, the error can also be written as

$$e = \sum_i \rho_{\text{Tuk}}(\min_j \Delta(\boldsymbol{m}_i, \boldsymbol{q}_{i,j})). \tag{4.8}$$

Such a computation of the projection error $e$ is more efficient, since the estimator function $\rho_{\text{Tuk}}$ needs to be evaluated only once. Using this equation as the error function for the

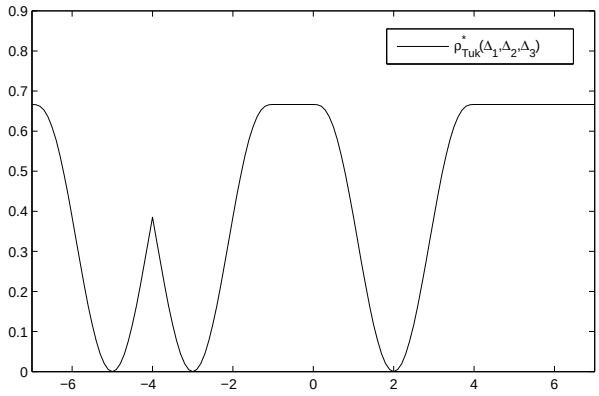



Figure 4.3.: Estimator function with multiple hypotheses at the positions $\Delta_1 = -5$, $\Delta_2 = -3$ and $\Delta_3 = 2$. Only the hypothesis which is closest to the current position of the control point is used as an error measure.

non-linear minimization guarantees that the hypothesis which has the closest distance to the projected sample point is always used. Falsely detected edge points do not have a negative influence as long as a hypothesis of the correct edge point also exists.

Figure 4.3 shows an example of the estimator function using multiple hypotheses. Only the hypothesis is used as an error measure in every iteration step of the minimization. Other hypotheses which are further away from the current estimate do not have any influence.

As large displacements shall be tracked, the search space must not be too small. But the more distinct gradient maxima are on a search line, the more hypotheses are collected, and the more distances have to be calculated during the non-linear minimization process. The computation costs have to be taken into account for a real-time performance, and therefore in our implementation we use a fixed number of hypotheses for each sample point. Only the $g$ most distinct gradient maxima are considered as potential 2D feature points. In our implementation we achieve good results with a maximum number of hypotheses of $g = 5$.

## 4.5. Visibility Test

A central question in model-based tracking is to determine all visible lines of the model from a given camera position. A static line model can only be assumed, if all lines of a model are visible in every camera frame. This can be the case if a wall in an office or the front of a building is tracked. However, in many scenarios an object shall be tracked from different viewing positions and many lines of the 3D model can be occluded by the object itself. Trying to find correspondences for occluded model lines leads to false matches, which disturbs the pose estimation. Therefore it is necessary to select only the control points which are visible.

The RAPiD approach [37] requires a pre-processing of the models to determine visibility of control points from various camera positions. Drummond [26] uses a technique, where

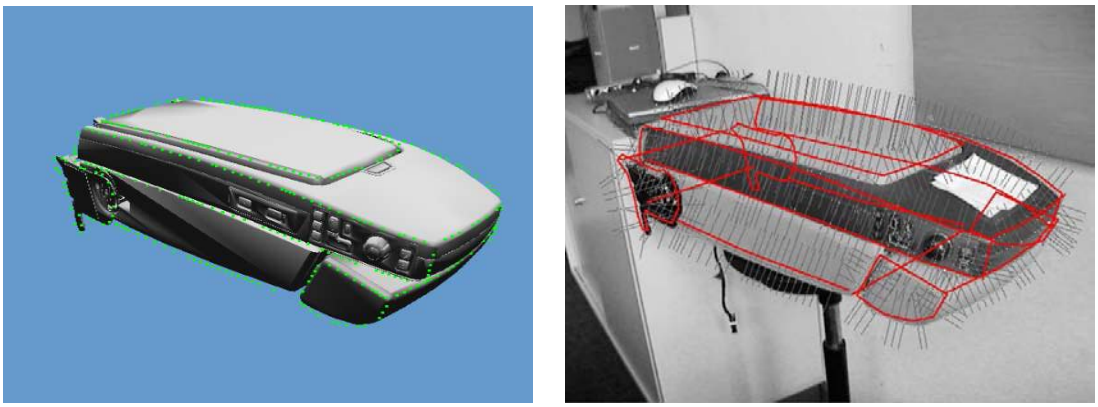


Figure 4.4.: Illustration of the visibility test. Control points which do not affect the depth buffer will not be used for tracking.

the visibility of control points is determined by rendering the model with the predicted camera viewing parameters. The 3D model is represented by a binary space partition tree, where each node consists of a plane and a list of edges. With an in-order-scan of the tree and the aid of a stencil buffer the model is rendered with only the visible edges for a given camera viewpoint. Correspondences between visible 3D control points and 2D measurements of the closest image edge are then determined by performing a perpendicular search at every projected control point.

To create a representation of a 3D model in such a binary space partitioning tree is, however, a complex preprocessing step. Nowadays polygonal surface models, which were created in the design process, are often available. These models can be used to carry out a visibility test for 3D edges. A possible solution to perform a visibility test is to carry out an intersection test between a ray starting from the 3D camera position and passing through the projected control point in the image plane and a 3D model of the tracked object. If the intersection point is not close to the 3D control point, this point is declared not visible. The main disadvantage of this method is that it is only fast enough for a small number of visibility tests. With complex 3D models and several hundred sample points, this intersection test becomes rapidly unsuitable for real-time tracking.

A much faster approach to carry out a visibility test is to render the scene on the GPU of modern graphics hardware and to test if control points on the 3D model are displayed or not. Testing the color of a pixel in the frame buffer with `glReadPixels` is rather slow. A much better solution is to use the GL extension `OCCLUSION_TEST_HP`. This extension tests, if drawing a set of geometries modifies the depth buffer. If the depth buffer is not modified, the drawn geometry is not visible. In our implementation, a VRML model of the tracked object is rendered off-screen with regard to the current camera pose into a p-buffer. Every sample point on the 3D line model is then drawn into the same p-buffer and the occlusion test is carried out. In order to visualize the result, the p-buffer can be copied into a texture and the texture can be drawn into a frame buffer. Figure 4.4 shows an example of a VRML model with all sample points of the line model drawn. The perpendicular search for gradient maxima is performed for visible sample points only. All other points are ignored.

The main advantage of this method is that it is really fast, since the visibility test is computed in hardware.

## 4.6. Texture-Based Edge Tracking

An edge of an object or an object boundary is often assumed to be observed as a high gradient in the image. Therefore many line-based tracking methods rely on interpreting image gradients as object edges. However, some edges especially of textured objects do not result in high intensity changes in the image, and gradient-based methods are then not able to detect the edge of an object.

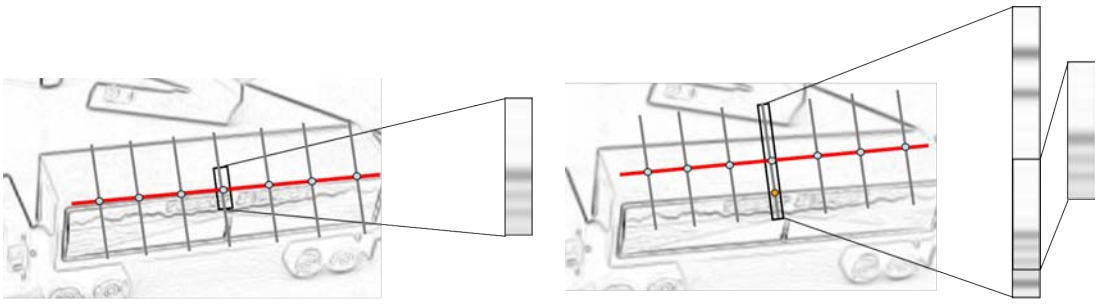
To overcome this problem, Shahrokni et al. [99] presented a method where object boundaries in the image are detected with texture segmentation techniques. Instead of using edge gradients to detect a contour in the image, they propose an algorithm based on a Markov Model to detect the transition of one texture to another on the search line. It is shown that with this texture segmentation method the tracking of textured materials in cluttered backgrounds produces more stable results than using gradients.

Kemp et al. [54] use a similar method to find points of texture change on a one-dimensional search line. However, they consider multiple hypotheses of texture change points as measurements and demonstrate the improvements by this technique.

Reitmayr [89, 88] uses another method to detect edges by their textured appearance. A given scene is rendered with the predicted camera pose and contours are extracted in the rendered image. At control points on the contours a line of pixel intensities is extracted on a line orthogonal to the current contour direction. In the current video frame a one-dimensional search for these edge points is performed by applying a normalized cross correlation along the search line. With the aid of a rendered model it is always possible to extract features in the rendered image with the correct appearance and scale. The correlation between two pixel intensity lines therefore produces good detection results. A drawback of this method is the fact that a photorealistic model must be given.

If no textured model is given, correlation-based techniques can be nevertheless useful for more accurate tracking results. In our model-based tracking system [120] we use the appearance of an edge in a previous frame to support finding the correct edge point in the current frame. This can be regarded as an prediction step.

An optical flow-based prediction for line segment matching is also presented by Chiba and Kanade [16]. They use a hierarchical optical flow estimation technique to predict the motion of line segments. Whereas they compute the optical flow in two dimensions, we only perform a one-dimensional search on a line perpendicular to the current edge. Furthermore, we do not only predict the 2D position of lines but also the camera pose and the 3D position of control points on an edge, which will be helpful for further processing, like testing the visibility.



(a) (b)

Figure 4.5.: Prediction of the control points. (a) After a successful tracking step a one-dimensional line of pixel intensities is extracted. (b) In the proximate frame the edge point is detected by a normalized cross correlation along the search line.

## 4.7. Correlation-Based Camera Pose Prediction

If a line model is given and used to track an object by a gradient-based approach, the projected lines must always be in proximity of the image edges, because the search method described in Section 4.2 is only a local search method. The convergence towards local minima can be avoided if the initial camera pose used for the minimization is a very close approximation to the real camera pose. Therefore a prediction of the camera pose from frame to frame is very beneficial for the convergence behavior.

In [120] we presented a method where the camera pose is predicted by using the appearance of an edge. As correspondences between 3D contour points and 2D image points exist, if the previous frame was tracked successfully, these correspondences can be used again to make a prediction of the camera pose in the current frame. Therefore the control points of the 3D edges are projected into the current image with the camera pose of the previous frame. A one-dimensional perpendicular search is performed, but instead of looking for gradient maxima, the point which is most similar to the 2D point in the last frame is regarded as the wanted 2D point. So if the tracking in the previous frame was successful, a one-dimensional window on every control point in the direction of the contour normal is extracted out of the previous image and a normalized cross correlation is performed along the one-dimensional scan line in the current image. For every 3D control point the point with the highest correlation result is regarded as the corresponding 2D image point. Figure 4.5 illustrates this process of predicting control points.

Because of shadows or bad lighting conditions it can happen that a projected control point is located in a very homogeneous area of the image. In such cases the correlation window will not contain distinct structures. Therefore we first test for all extracted correlation windows, if strong enough gradients exist, and use only those correspondences for the pose estimation.

The calculation of the camera pose is done by a Levenberg-Marquard minimization as described in Section 2.3.3. Only one hypothesis is used as a measurement, because ambiguities of falsely detected points are very rare.

The one-dimensional correlation technique to detect edge points always needs an exact representation of the visual appearance of a model edge. This can be achieved by rendering a photorealistic model or by extracting the appearance of an edge feature directly in an image video image. Extraction and detection of a correlation window, however, only works if the appearance is very similar. A whole sequence would be hard to track with such a method, because the appearance of an edge changes dramatically if the viewing direction, the scale and or illumination differs too much. Therefore a purely correlation-based detection of edge points is mostly suitable for a frame-to-frame prediction.

## 4.8. Validating the Tracking Success

After the pose estimation by minimizing the projection error with the Levenberg-Marquardt method, it is desirable to know, if the tracking step was successful or if the tracking failed. A very straightforward method is to check if the average projection error is smaller than a given threshold. Thereby outliers have to be considered and only the error inliers have to be analyzed. This method gives some rough estimate if the minimization of the projection error converges.

To decide if the tracking was successful or the alignment of a line model onto an image failed, we analyze the image gradient at the position of control points after the pose estimation. In an extra step all the control points which were used for the pose estimation are projected into the image again with the newly estimated pose, and the image gradient is analyzed at these 2D positions. One method to verify that a control point lies on an image edge is to check if the gradient intensity at that position is big enough that it can be regarded as an image edge. Another value which can be used to determine the tracking success is the direction of the image gradient. If the line direction of a projected control point is similar to the direction of the image gradient at the projected 2D position, this control point can be regarded as aligned correctly. As a measure for the tracking success the average aberration between image gradient directions and direction of the corresponding projected lines can be taken into account. A mixture of analyzing the gradient intensity and the gradient direction also produces good results in validating the tracking success.

If a textured appearance of a control point is given for a line-based tracking as described in Section 4.6, these one-dimensional lines of pixel intensities can also be used to validate the pose estimation. By analyzing the sum of square differences of the pixel intensities, it can be decided if the tracking step of a single control point was successful of if the tracking failed. For illumination invariance a simple lighting compensation is performed before the intensity comparison.

Any of these validation methods can be used for a semi-automatic model-based initialization. In [12] we used such a method to initialize a feature point based tracking system. A line model is projected with given camera parameters into the image, and the user has to move the camera, so that the virtual line models get close to the object in the image. If the image edges are close enough to the projected model lines, so that they lie inside the search space of the control points, the pose can be correctly estimated. If the validation

step proclaims a tracking success, the tracking system is regarded as initialized with the current camera pose.

## 4.9. Adapting the Visual Appearance

If the appearance of an edge is not given, it can be still useful to take the visual properties of an edge into account for tracking lines or contours.

However, the appearance of an edge changes, if an object is viewed from different viewing directions and different distances. Figure 4.6 demonstrates the necessity of handling multiple appearances of an edge. From different viewing positions or by the rotation of the object itself, the perpendicular lines of pixel intensities at the same control point look very different.

Tsin et al. [110] presented an online learning approach of intensity patterns to establish correct correspondences between points on the model edges and edge pixels in an image. They borrowed some ideas from keypoint recognition techniques with randomized trees [60]. In their system small segments of intensity patterns are used as descriptors. These descriptors are taken to train a randomized forest, which is then used to find the correct edge pixels on a search line by classification.

In [118] we modeled the multiple appearances of an edge with a mixture of Gaussians, which represent the pixel intensities of a line of an edge control point. Considering the visual edge properties of only the last frame will not lead to good detection results, since the appearance of an edge can change when the camera or the object is moved. A possible idea is to apply a temporal low pass filter on the edge pixel intensities, so that their state of appearance becomes more stable over time. Unfortunately, the edge of an object does not necessarily look the same in every frame of an image sequence. The visual perspective, the lighting conditions and the background are many factors that can cause the appearance of an object edge to change considerably. It is therefore necessary to describe the properties of an edge with a multi-appearance model.

Condensation [46] is an effective but costly method to maintain a probability density over time. As the number of control points to be tracked is rather high, the condensation algorithm will be computationally very costly and it is therefore not suitable for real-time tracking. A compromise between accuracy and complexity is to use a mixture of Gaussians with a fixed number of distributions.

We used the ideas of Stauffer and Grimson [104], who used a Gaussian mixture model to represent a background model with multiple appearances. We did not apply this method on a background image, but on every intensity pattern of an edge control point. Several hypotheses of an edge are maintained by this filter, and the most probable one is always used for detection.

An edge's visual property is represented by a multidimensional Gaussian distribution. In our implementation we simply used the pixel values of the correlation window as variables of the Gaussian. The dimension $m$ of the distribution is therefore equal to the size of the correlation window.

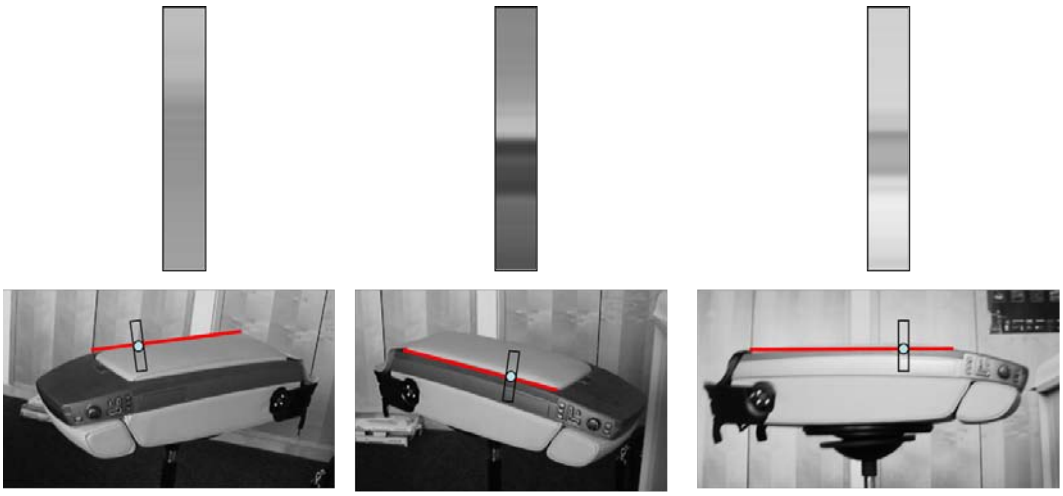


Figure 4.6.: Necessity of using multiple visual appearances. In all images the extracted line of pixel intensities looks different.

A Gaussian distribution consists of a mean $\mu$, a variance $\sigma^2$ and a weight $\omega$. The weight $\omega$ is a measure which represents what portion of data from previous frames is taken into account by this Gaussian. To avoid a costly matrix inversion, variances are calculated separately for every dimension of the Gaussian distribution. Edge property values are therefore treated independently from each other.

The probability of observing the current edge feature $x$ at time $t$ is

$$P(x_t) = \sum_{i=1}^{m} \omega_{i,t} \cdot G(x_t, \mu_{i,t}, \sigma_{i,t}), \tag{4.9}$$

where $G$ is the Gaussian probability density function.

To update the mixture model with a current value, every measured edge property $x_t$ is checked against the existing $m$ Gaussian distributions. A match is defined if the measurement is within 3 standard deviations of the distribution. If none of the distributions is a match to the current measurement, the mean of the least probable distribution is replaced by the current measurement. The variance $\sigma^2$ of this distribution is initialized with a high value, the weight $\omega$ is set to a low one. The Gaussian with the highest $\frac{\omega}{\sigma}$ is interpreted as the most probable distribution.

The weights of the $m$ distributions at time $t$ can be computed by

$$\omega_{i,t} = (1 - \alpha)\omega_{i,t-1} + \alpha M_{i,t}, \tag{4.10}$$

where $\alpha$ is the learning rate and $M_{i,t}$ is 1 for the matched distribution and 0 otherwise.

The parameters for the matched distribution are updated as follows:

$$\mu_t = (1 - \beta)\mu_{t-1} + \beta x_t \tag{4.11}$$
$$\sigma_t^2 = (1 - \beta)\sigma_{t-1}^2 + \beta(x_t - \mu_t)^2, \tag{4.12}$$

**Algorithm 1** Adaptive line tracking algorithm

```
1: for all lines l of the 3D line model do
2:    determine the number of n control points depending on the length of the projected
      line
3:    carry out the visibility test for control sample points and store the visible points in
      the set V_l
4:    for all points M_i in V_l do
5:       project 3D point M_i it into the image plane
6:       search for gradient maxima along the edge normal and consider every maximum
         larger than a certain threshold as an hypothesis for an edge point
7:       for all possible edge points do
8:          calculate the similarity with the most probable distribution of the adapted edge
            properties
9:       end for
10:   end for
11: end for
12: for all points of V do
13:   take only the hypothesis with the highest similarity as match and calculate the
      camera pose by non-linear minimization
14: end for
15: for all points of V do
16:   append the g most significant gradient maxima to the list of possible hypotheses
17:   apply the minimization again with the estimated pose of the previous step as initial
      guess
18: end for
19: for all points of V do
20:   update that appearance model which is most similar to the current control point
21: end for
```

where $\beta = c \cdot \alpha$ is a second learn rate depending on the learn rate $\alpha$. The learn rate $\alpha$ is controlled by the accuracy of the pose estimation. Good estimates set the learn rate to a higher value, whereas unconfident ones result in an update step with a smaller learn rate.

A significant advantage of this method is that a dramatic visual change of an object edge in the image, e.g. when viewing an object from a totally different direction, does not destroy the existing state of appearance. The original edge property remains in the mixture of Gaussians. When the camera is turned back to an earlier position, the previous visual properties of an edge still exist with the same $\mu$ and $\sigma^2$ but with a lower $\omega$, and will quickly become the most probable distribution again. Another benefit is that occlusions do not disturb the adapted edge properties too much, since it is likely that different looking edges are assigned to different distributions.

To clarify the adaptive line tracking algorithm with multiple hypotheses the high-level pseudocode for processing one frame of an image sequence is pointed out in Algorithm 1.

## 4.10. Selection of Control Points

The number of control points has a significant influence on the robustness and the runtime of the tracking system. The more control points are used, the more stable the pose estimation result gets. However, every control causes additional computational costs and decreases the performance of the tracking system. Selecting a suitable number of control points can therefore be regarded as a tradeoff between robustness and real-time performance. In addition the correspondences between 3D control points and image edge points should be distributed evenly over the current image to result in a robust pose estimation. If fixed sample points on the 3D line model are used, the density of their projection in the image is not uniformly distributed. The length of the projected lines depends on the perspective of the camera and on the distance to the considered object. The number of sample points should therefore depend on the length of the projected line and not on its length on the 3D model. If a tracking method only detects gradient maxima on a search line as described in Section 4.2, uniformly distributed control points can be created on the projected line. We achieved good result by creating a control point after every $10^{\text{th}}$ pixel on the projected line.

However, if an adaptive approach as in Section 4.9 is used, the edge properties of a control point must always belong to the same point of the tracked object in order to justify the adaptive appearance filter algorithm. To overcome this problem we computed control points along the line of the 3D model and projected only a subset of these control points into the current image. The size of the subset of projected control points is determined by the length of the projected line.

As the control points shall be evenly distributed, they are numbered on the 3D line in a tree-like recursive way as illustrated in Figure 4.7.

The first $n$ control points are then used to calculate 2D/3D-correspondences, where the number $n$ can be computed by

$$n = \frac{\text{length of the projected line}}{\text{control point density}}. \tag{4.13}$$

With such a numbering it can be guaranteed that the selected control points are always evenly distributed along the projected line.

Figure 4.8 shows two images of the same object with different distances of the camera to the object, and therefore with different numbers of control points.

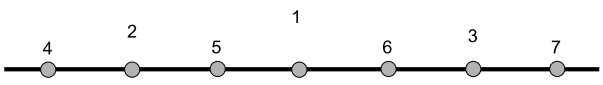
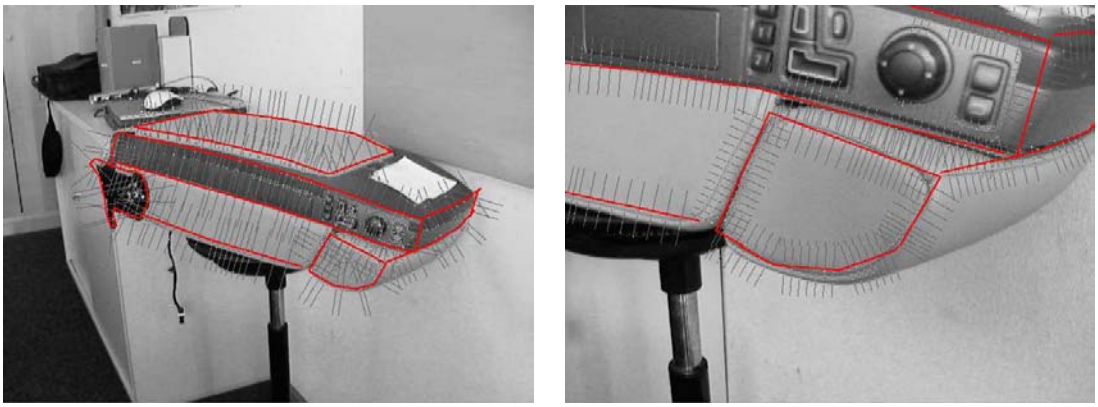



Figure 4.7.: The recursive numbering of the control points causes that the subset of the first $n$ points is more evenly distributed.



(a) (b)

Figure 4.8.: Two images of the same tracked object showing that the control points are always evenly distributed. The number of control points depends on the length of the projected line.

## 4.11. Evaluation of the Line Tracking Methods

Image sequences of several scenes were recorded and used as input for different tracking methods. The camera pose was initialized either manually or by detecting markers. In the first experiment, an object in a static camera sequence of 949 frames was tracked with different approaches: single hypothesis ($M_a$), multiple hypotheses ($M_b$), single hypothesis with the adaptive method ($M_c$), and finally multiple hypotheses with the adaptive method ($M_d$) as described in the algorithm of Section 4.9.

If method $M_a$ is used, many sample points are assigned to false edges. False correspondences lead to an inexact pose estimate and eventually to the loss of the track. Using method $M_b$ significantly improved the tracking results, as the line model sticks much better on the object in the image sequences. However, when fast motion occurs, the estimated pose is always several frames behind the real camera pose. It also sometimes happens that the minimization process is stuck in a local minimum, thus not leading to the desired result. Method $M_c$, that uses only the most likely hypotheses with regard to the previous measurements, produces better results during large movements, but still has problems if other edges are near the line to be tracked. False correspondences lead to a bad adaptation of the edge properties and quickly cause the edge filter to learn wrong properties of the edge. If method $M_d$ is applied, the problems described above are avoided. The camera pose is estimated correctly during the whole sequence. Taking into account multiple hypotheses makes the edge filter much more likely to be updated with the properties of the correct edge. Therefore, the adapted state is more accurate and leads to a better result in finding the most probable gradient maximum. Figure 4.9 illustrates the results of the four tracking methods described above.

Since the computational complexity is higher for $M_d$, it is slightly slower than the other ones. On a 2.8 GHz Pentium IV, it needs about 60 milliseconds for an iteration step with an image size of 640x480 pixels, whereas $M_b$ takes about 50 milliseconds on average for the image sequence.

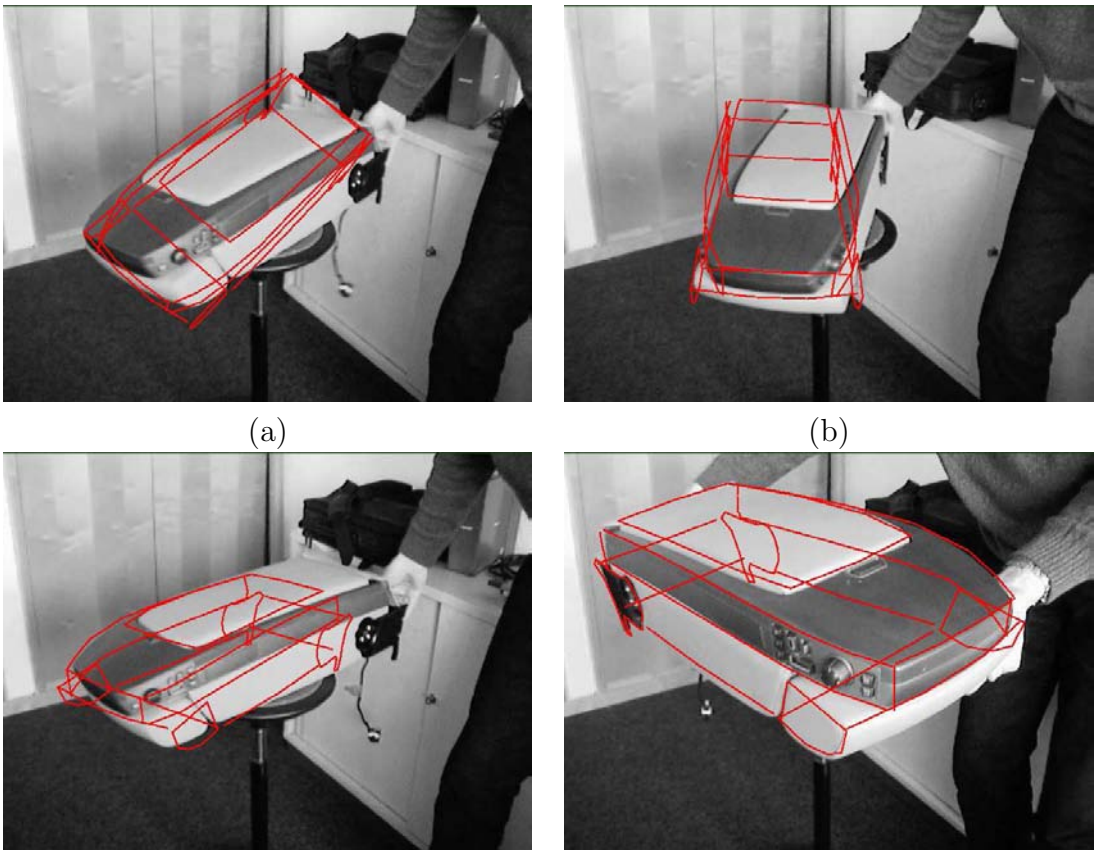
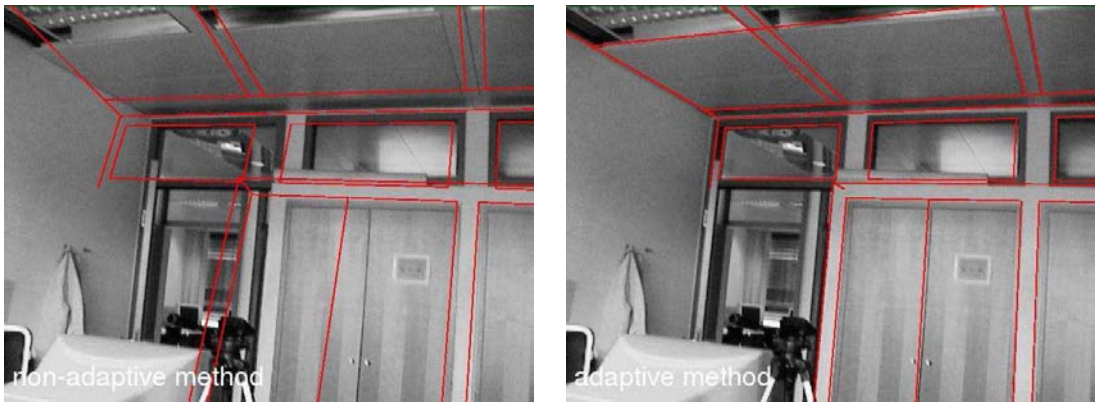


Figure 4.9.: Results of the different tracking methods: (a) the single hypothesis method loses the track at frame 451, (b) the multiple hypotheses method fails at frame 428, (c) the adaptive method fails at frame 462, (d) the adaptive approach with multiple hypotheses stays on track during the whole sequence.

In another experiment, the indoor environment of an office building was tracked with both $M_c$ and $M_d$ (see Figure 4.10). Again, it can be observed that fast movements are handled much better by the adaptive algorithm. With the non-adaptive approach, tracking sometimes fails, because too many wrong 2D/3D-correspondences are used for the pose estimation.

Figure 4.13 shows the result of our adaptive line tracking algorithm in an industrial scenario. The camera path can be tracked successfully throughout the sequence as long as enough parts of the object are visible in the camera image, where lines in the manually created line model are available. Only when the camera turns away from the scene, the tracking fails.

To measure the run-time as a function of the number of sample points, another object was tracked from different distances. As seen in Section 4.10, the number of sample points depends on the length of the projected lines, and therefore on the distance between the camera and the tracked object. This means that an iteration step of the tracker needs more computational time when the object is close to the camera and less time when it is further away. Figure 4.11 shows an object tracked from different distances and a scatter




(a) (b)

Figure 4.10.: Tracking the office sequence with the non-adaptive (a) and the adaptive method (b).

plot illustrating the run-time of an iteration step versus the number of used sample points. The sample point density was chosen so that the distance between two adjacent search lines was at least 10 pixels. To increase the frame rate of the tracking, the sample point density can be decreased. However, if the sample point density is too small, the estimated 6 degrees of freedom of the camera start jittering. The tracking algorithm needs about 50 milliseconds on average for an iteration step, which means that it can run at a frame rate of about 20Hz.

If some parts of an object are occluded, it is still possible to estimate the camera pose (see Figure 4.12). Due to the robustness towards outliers of the Tukey estimator function, the estimated pose is correct even with a certain amount of outliers. If a higher occlusion proportion is expected, the Tukey constant $c$ of equation (2.33) can be set to a lower value. The Tukey constant shall not be too small though, otherwise tracking results become unstable and start jittering. If there is very little or no occlusion, the Tukey constant can be set to a higher value.

The adaptive approach, which tries to maintain the visual appearance of a line control point, improved the tracking robustness by using only the most probable edge for every sample point during the first minimization step, so that the pose estimation does not get stuck in local minima. After the first minimization, the resulting 6 degrees of freedom lie around the desired minimum of the error function. The way through many local minima caused by multiple hypotheses is thereby avoided. A uniform distribution of sample points leads to a balanced set of 2D/3D correspondences and keeps the computation costs low. Finally, using multiple hypotheses helps to find the correct edge out of many possible gradient maxima in the image and leads to more accurate measurements that are used for learning the visual properties of an edge.

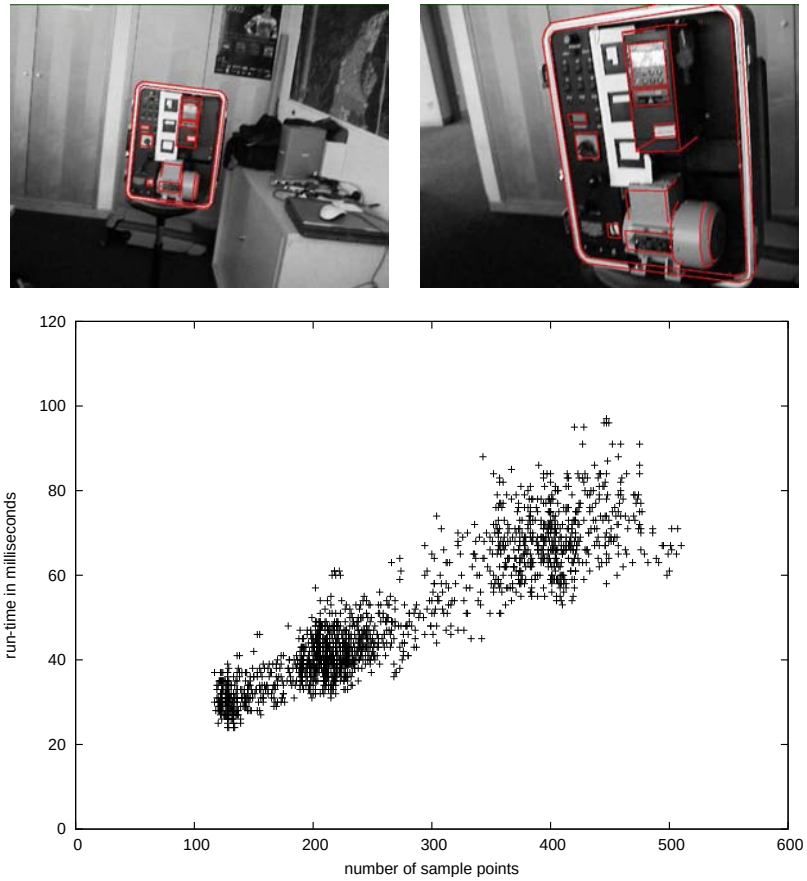



Figure 4.11.: Scatter plot of the run-time versus the number of sample points used for tracking.

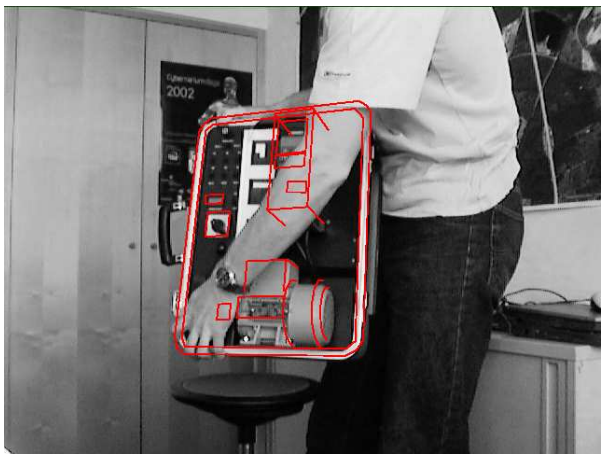


Figure 4.12.: Tracking an object with occlusion. The markers are only used for initialization.

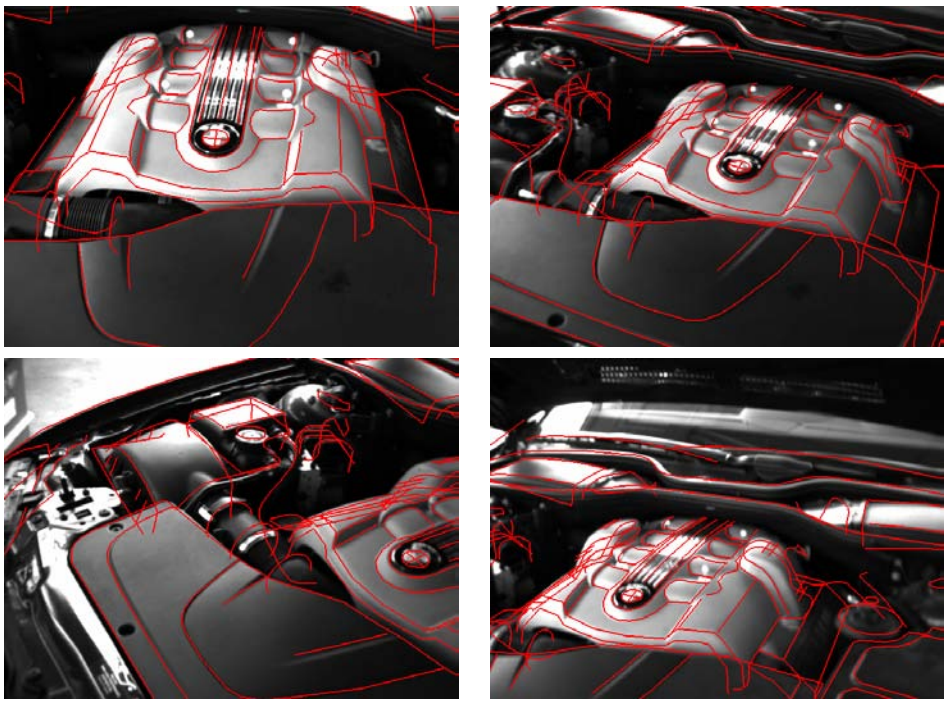


Figure 4.13.: Tracking the engine hood of a car with the adaptive line tracking method. The line model is manually created.

# 5. Line Tracking Based Analysis by Synthesis Techniques

Three-dimensional line models which are needed for a line-based tracking system as described in Section 4 are usually never available. The CAD models which are created in the industrial engineering process do not consist only of those lines, which are useful for tracking, but of much very detailed geometry, which does not have any properties in common with object edges seen in a video image. Polygonal models of industrial objects often exist and can be of advantage if an object shall be detected or tracked in an image sequence. The challenge is to find a good connection between the geometric data of an object and its visual appearance in an image.

Some existing line-based tracking systems used rendering techniques to obtain a set of control points for a given camera viewing direction. Drummond and Cipolla [26] present an approach that is based on identifying edges in a rendered CAD model. They use a model represented by a binary space partitioning tree to render object edges with correct visibility.

A method of extracting line features out of a textured scene is described by Reitmayr [89]. He extracts contours in a rendered image and uses the texture information at control points to find these edges in a video image.

Roston [91] uses another interesting approach for generating contours of silhouette edges of an object on the fly. He models the objects to be tracked with implicit surfaces and calculates the apparent contours for a given camera viewing position analytically. This method can be fast enough for real-time tracking if the objects have a limited complexity. For our purpose this method is, however, unfeasible, because the geometry of industrial objects is never represented by implicit surfaces, and the creation of such implicit shapes would be a very tough preprocessing step.

An offline generation of a contour model was presented in [86]. Several rendered images with different lighting conditions and camera viewing directions are used to create 3D clouds of contour points. Connected 3D contours are then created with the Euclidean Minimum Spanning Tree algorithm. This method requires a huge amount of computation time of several hours and the quality of resulting contour models is rather poor.

In this thesis we present a model-based approach which generates a 3D line model out of a surface model of an object and uses this line model to track the object in the image. For the line model generation a polygonal model of the object is rendered with a predicted camera pose and contours are extracted by analyzing discontinuities of the z-buffer and the normal buffer or by detecting edges in the frame buffer. The generated 3D line model is projected into the current image and the extrinsic camera parameters are estimated

by minimizing the error between the projected lines and strong maxima of the image gradient.

## 5.1. Line Model Generation

In order to ensure the robustness and accuracy of the line tracking algorithm, the set of lines of the model should be congruent as best as possible with the set of image line features or high gradient points. Model lines abundant in the image as well as cluttering image line features not contained in the model may degrade the performance of the tracker. However, it is a difficult task to obtain a perfect congruency, as many high gradient points in the image arise from shadows, reflections of smooth surfaces or changes in surface color. Therefore, the main objective of the line model generation step should be to acquire many good candidates of lines that are most likely to be visible in the image, as well, while preserving an affordable amount of computational costs.

In many line-based tracking systems [118, 92] the 3D model used for tracking is made up of geometric three-dimensional lines. The line model has to be provided once in advance and usually has to be constructed manually. It remains basically unmodified throughout the whole tracking procedure. By performing a visibility test under the current viewing direction, lines hidden by the object itself are ignored. In contrast thereto our approach uses a polygonal surface model stored in a VRML file. Often such models are already at hand and can be provided by our partners as their creation is part of the manufacturers' engineering process of the object to be tracked. Our tracking algorithm performs an analysis of this surface model during each iteration step in order to extract a reliable, view-dependent line model. The computational time is kept to a minimum by making use of rendering standard graphics hardware.

One possible approach for the line model generation is to render the scene and to extract 2D-line features from the synthetic image in just the same manner as for the line feature extraction of the real images of the camera. We then can include model lines corresponding to high gradient points in the images resulting from changes in color or intensity. However, when working on tracking systems for industrial augmented reality applications, we experienced that the material properties of the models provided, such as textures or colors, are rarely modeled correctly or do not exist at all. Thus they usually do not contain valuable information which can be used for the line model generation. Furthermore, it is essential to know the illumination conditions exactly or at least to have an accurate estimation of them.

In contrast thereto, one approach for the real-time line model generation presented here is based only on the geometric properties of the object. It is related to the silhouette generation of polygonal models for the purpose of non-photorealistic rendering: Isenberg et. al. [47] describe methods which create silhouettes of a model in both object space and image space. Object space methods analyze the normal vectors of triangles in relation to the camera viewing direction. Image space methods render the scene and analyze the output of the rendered images. Whereas object space methods can produce curves with much higher precision, they come along with higher computational costs. Image

space methods are computational less expensive and can be implemented by using pixel shading hardware as presented in [76]. Hybrid algorithms which combine the advantages of both image space and object space methods exist [82]. As real-time performance is an important criterion in our application we use only an image space method as detailed in [81].

By focusing solely on the geometric properties of the models, usually more reliable line models can be obtained. A common assumption is that model edges normally are also visible in the image as high gradient points, since adjacent surfaces with different orientation usually have a different brightness under illumination. Building upon this basic idea our algorithm creates a view-dependent line model as follows:

First the surface model is rendered with the predicted camera pose. To keep sampling artefacts as low as possible, the near and far clipping planes are adjusted to exactly match the bounding volume of the object. In a second step an edge map is created by analyzing discontinuities in the z-buffer or normal-buffer. Two types of edges are of interest: step edges (also called $C^0$ edges), which correspond to a surface partially occluded by another one, and crease edges ($C^1$) as the locations of two adjacent surfaces with different orientation. For the purpose of their detection we present and discuss several filtering methods detailed in the Subsections 5.1.1 and 5.1.2. Another method for extraction edges out of the frame-buffer is described in section 5.1.3.

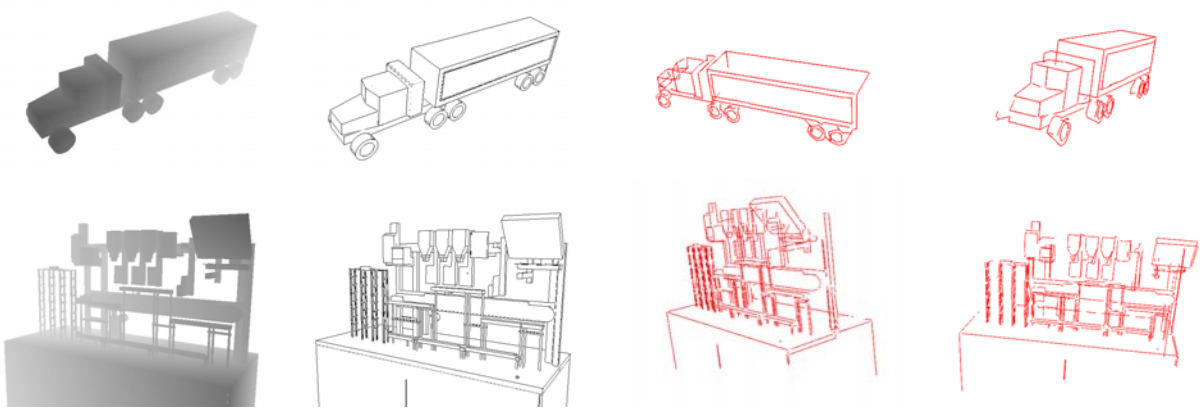
The last step consists of extracting the 3D-lines of the edge map by means of a Canny-like edge extraction algorithm. A 3D contour in the world coordinate system is computed by un-projecting every pixel in the edge map with the information stored in the z-buffer and applying an inverse transformation of the current camera rotation and translation parameters. For the visualization a recursive subdividing algorithm splits the 3D contours into straight line segments. The tracking approach described in Section 4 needs only control points and the direction of the 3D contour at the control points. During the contour following of the edges in the edge map these control points are generated directly and used as input later on for the registration step.

### 5.1.1. Edge Map Generation using the Depth Buffer

Discontinuities in a z-buffer image are changes in depth and can be regarded as a point on an edge of an object according to the given camera viewing direction. Having rendered the surface model with the predicted camera pose, a second order differential operator is applied on the z-buffer image to generate an edge image. As one approach in our implementation we use the following simple 2D Laplacian filter mask in order to find points belonging to silhouette edges:

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

For silhouette edges the Laplace operator returns a high absolute value both on the object border and on the neighboring pixel in the background. When the value of a pixel in the



(a) (b) (c) (d)

Figure 5.1.: 3D Line extraction using discrepancies in the z-buffer. (a) shows a z-buffer image. (b) illustrates an edge image obtained by a Laplacian Filter. (c) and (d) show the generated 3D line model of different views.

Laplace image is positive, the regarded pixel is located on that side of an edge which is further in the depth. When the Laplace filter results in a negative value, the regarded pixel lies on that side of an edge which is closer to the near plane. Since we are interested in the 3D coordinates of pixels, which are located on the edge of an object and not on a background surface or the far plane, we only consider pixels as a silhouette edge which have a negative response of the Laplacian filter. Therefore, it can be guaranteed that silhouette pixels are actually located on the rendered object. At strong depth discontinuities the Laplace filter produces only a one pixel thick contour, which is very desirable for the contour following later on.

For the purpose of the line model generation, other edge detection filters like the Sobel filter, as suggested in [95], do not lead to good results, since $C^0$ discontinuities are more than one pixel thick and it cannot be distinguished on which side of a silhouette a pixel is located. Another problem with first order differential operators like the Sobel filter arises due to the fact that pixels on a very steep surface are all regarded as edge pixels.

Figure 5.1 illustrates z-buffer images of some objects, the resulting edge maps and the generated line model from two different viewing positions. Here the threshold applied on the Laplace filtered z-buffer image is chosen, so that not only the silhouette edges but also the crease edges are generated. If objects are built by cube-like geometric primitives, then crease edges can be generated if the Laplace threshold is set to a very low value. As finding a good threshold depends on many factors like the distance of the object to the camera or the distance between the near and the far plane, it is almost impossible to produce a clear edge map of crease edges for different models or different camera viewing positions with a fixed threshold. Therefore the z-buffer method is mostly suitable for finding silhouette edges.

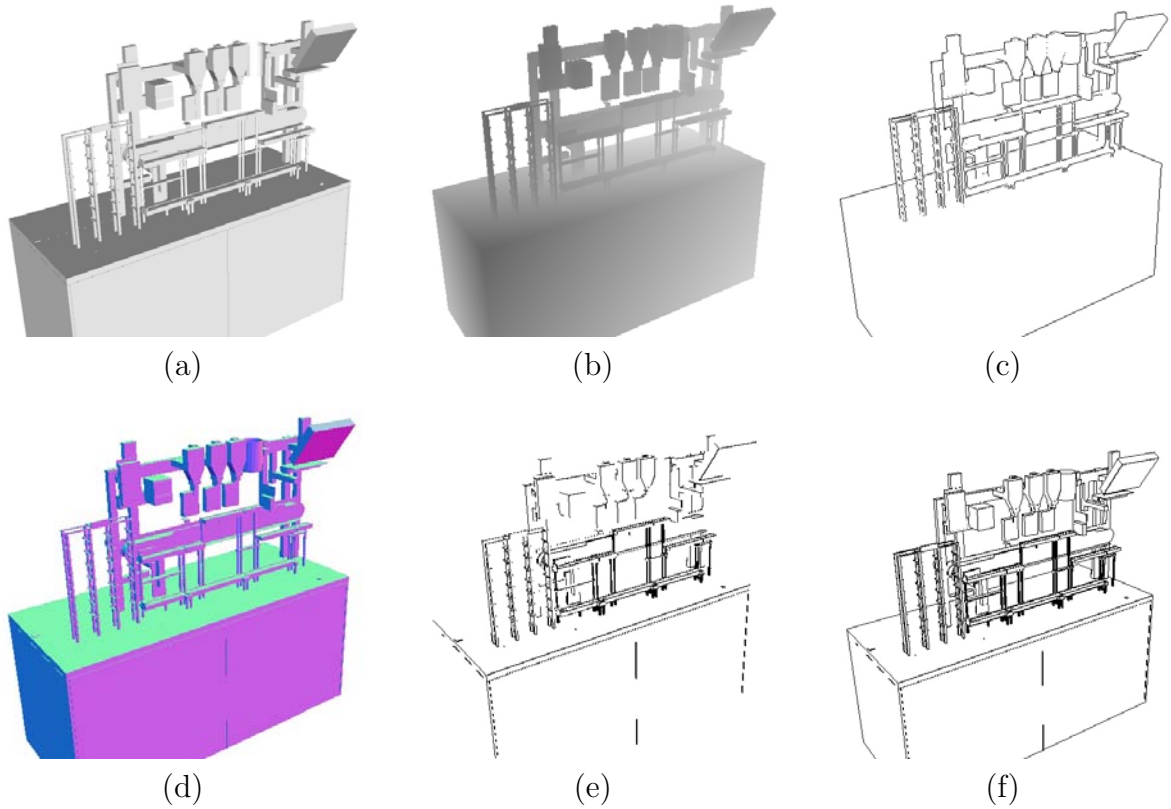


Figure 5.2.: Edge map generation: (a) shows a rendered object and (b) the depth buffer of that object. (c) illustrates the silhouette edges. The normal map is shown in (d) and the edges of the normal map in (e). The combined edge image can be seen in (f).

### 5.1.2. Edge Map Generation using the Normal Buffer

If the threshold which is applied on the Laplacian filtered depth buffer image is small enough, it is also possible to detect crease edges with the method described in the previous section. In [12] only the depth buffer was used to generate an edge map of both crease and silhouette edges. However, due to the nonlinear nature of the z-buffer, such a threshold is not easy to define, and the quality of the edge image soon gets very poor, if the threshold is chosen too small. Another disadvantage of only using the z-buffer is that for crease edges the value returned by the Laplace operator also depends on the viewing angle of the camera.

A far more promising approach is the additional extraction of edges of the normal map as described in [47]. A normal map is a buffer which instead of the $(r, g, b)$ color components consists of the $(x, y, z)$ coordinates of the surface normal vectors at each pixel. In [41] a method is described, which creates a normal map by placing several light sources of different color on the axes of the camera coordinate system. Nowadays graphics hardware offers more direct ways to create a normal map. In our implementation we simply use a fragment shader which transforms the surface normal $\mathbf{n}$ into the camera coordinate

system and puts it into the range between 0 and 1 by

$$\begin{pmatrix} r \\ g \\ b \end{pmatrix} = \frac{1}{2}\left(M_N \mathbf{n} + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}\right), \tag{5.1}$$

where $M_N$ is the $3 \times 3$ matrix which transforms the surface normal from the world coordinate system into the camera coordinate system. To extract edges out of the normal map, simply for every pixel the angle between neighboring normal vectors is calculated and tested, if it exceeds a predefined threshold. Thereby an edge image is created which represents changes in surface orientation.

Figure 5.2(d) and (e) show a normal map of a rendered scene and the resulting crease edge image. The z-buffer and the edges resulting by a Laplacian filtering are also shown in (b) and (c). A combined edge map which includes both silhouette and crease edges can bee seen in Figure 5.2(f).

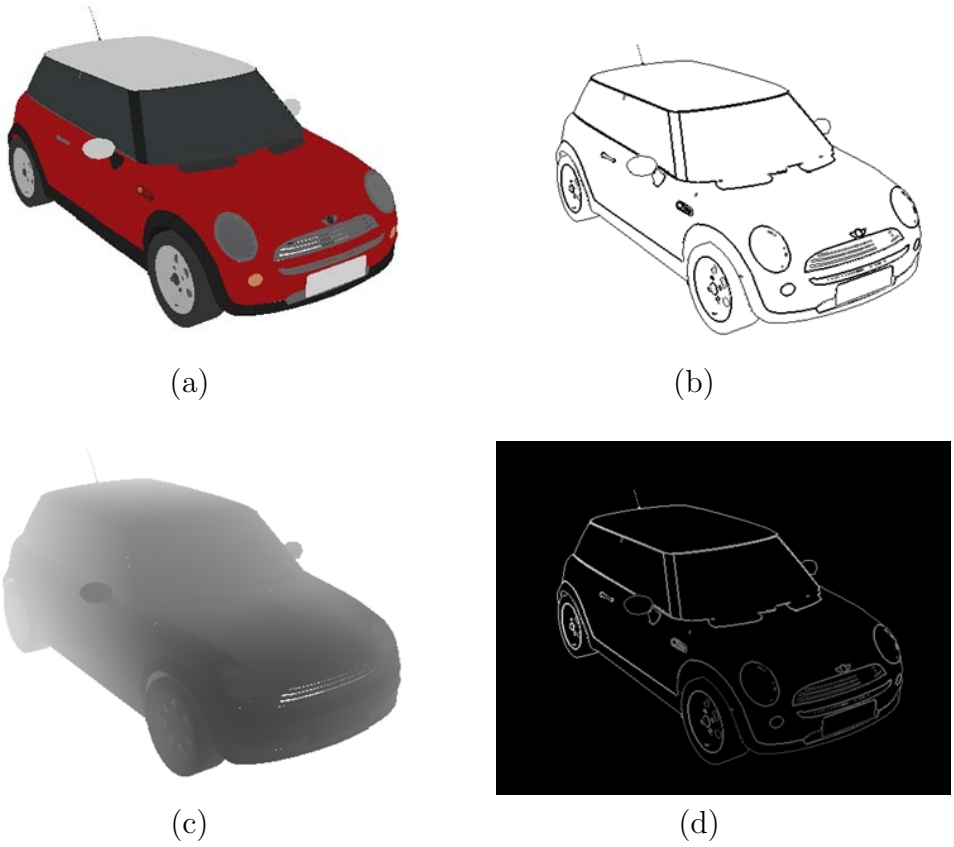
The whole process of creating an edge image is done on the graphics hardware with two rendering passes. In the first pass the scene is rendered into a texture, whereas the normal vectors are stored in the $(r, g, b)$ color components and the depth is stored in the alpha channel of the texture. In the second pass a fragment shader creates the combined edge map and stores the edge image in only one color channel. The depth information is thereby stored in the intensity value of that pixel. Only one color channel has to be read back from the graphics hardware into the main memory, which is another performance advantage compared to an edge extraction on the CPU [89], where all four channels would have to be read back, if the normal map was used as well.

### 5.1.3. Edge Map Generation using the Frame Buffer

In high detailed scenes with lots of triangles the edge map obtained by geometrical properties like the depth or the surface orientation gets very fuzzy and the resulting contours are not very distinct. If material properties of an object exist or the scene is textured, edges can also be extracted out of the frame-buffer. Edges in an image are then assumed to represent material boundaries.

In order to make the edge map generation independent from any light source and the viewing direction, the scene is rendered only with ambient lighting. The gradient of the frame buffer is computed with the simple mask $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ in the x- and y-directions. The edge extraction with a Sobel filter produces thicker lines, which is not beneficial, since the contour following works best with a one pixel thick edge. Smoothing in any direction like the Sobel filter does, is not necessary, since the rendered image does not contain any noise.

Again the edge image is created in two rendering passes. In the first pass the scene is rendered with ambient lighting into a texture and the z-buffer value is stored in the alpha channel. In the second pass the image gradient of the $(r, g, b)$ image is calculated. Pixels which are on the far side of a silhouette edge are detected as described in Section 5.1.1 and suppressed in the edge image. Thereby it can be guaranteed that all edge pixel are located on the object, which is necessary for getting the correct 3D coordinates. The depth value of all edge pixels is stored in only one color channel, which is read back for further processing. Figure 5.3 shows the generation of material boundary edges.



(a) (b)

(c) (d)

Figure 5.3.: Generation of an edge image with material properties. The image is rendered with only ambient lighting (a) and edges are extracted with a first order differential operator (b). The $2\frac{1}{2}$D edge image which is read back from the GPU (d) consists of the depth values of the z-buffer.

## 5.2. Experimental Evaluation

In Algorithm 2 the method we used for evaluating the line model generation is described. A prediction step as described in Section 4.7 was used to make the tracking system more capable of handling large camera movements. The first camera pose is defined by the user. It must be close to the real camera pose, so that the registration of the generated line model can be successful. The generation step produces a line model which is only used for visualization and a set of 3D control points, which are used for tracking. If the registration step fails, no camera pose prediction is performed in the proximate frame. All images are undistorted with given radial distortion parameters, before they are processed by the algorithm.

### 5.2.1. Evaluation of Synthetic Image Sequences

All the tests are done on a Pentium 4 with 2.8GHz and a ATI Radeon 9700Pro graphic card. To evaluate the robustness and the accuracy, the algorithm is tested on a synthetic

**Algorithm 2** Tracking with generated line models

1: **if** the previous frame was tracked successfully **then**
2: Create correspondences between the 3D control points of the last generated model, and the 2D points obtained by the correlation along the line perpendicular to the regarded edge.
3: Predict camera pose by minimizing the re-projection error of the correspondences.
4: **end if**
5: Generate a new line model to the predicted camera pose.
6: Apply the line model registration.
7: **if** the registration was successful **then**
8: Extract a one-dimensional window of pixel intensities at every control point.
9: **end if**

image sequence first. A virtual model of a toy car is rendered with a predefined camera path, where the camera moves half around the model and back again. The images, which are rendered with a resolution of $512 \times 512$ pixels, and the very first camera pose are used as input for the tracking method. Only z-buffer edges are regarded for this test for the line model generation. After every processed frame the 6 degrees of freedom of the estimated camera pose are stored and compared with the ground truth data. In Figure 5.4 these values are plotted separately for every parameter. It can be observed that there is always a small error, but the method is capable of tracking the camera path throughout the whole synthetically generated sequence correctly.

The difference of the values between the real and the estimated camera pose are shown in Figure 5.5. Euler angles in radians are used to represent the three parameters of the camera rotation.

In Table 5.1 the mean error and the standard deviation of every component of the extrinsic camera parameters can be seen. As for most parameters the error alternates around 0, the mean of the z-component of the translation error is clearly above 0. This means that the estimated camera pose is always further away or that the tracked object in the image seems smaller than it really is. The reason for this fact is that the extracted silhouette edges are always on the object and the gradient edges in the image have their peak between the object border and the background pixel. Therefore the extracted silhouette edges have an error of half a pixel which mostly affects the z-component of the camera translation. By analyzing the standard deviations it can be seen that the uncertainty of the camera

| parameter | mean error | standard deviation |
|---|---|---|
| $x$ | $-0.0148$ | 0.1182 |
| $y$ | $-0.0462$ | 0.1068 |
| $z$ | 0.5608 | 0.3335 |
| $\alpha$ | 0.0003 | 0.0259 |
| $\beta$ | $-0.0022$ | 0.0163 |
| $\gamma$ | $-0.0031$ | 0.0229 |

Table 5.1.: Average error and standard deviation of the extrinsic camera parameters.

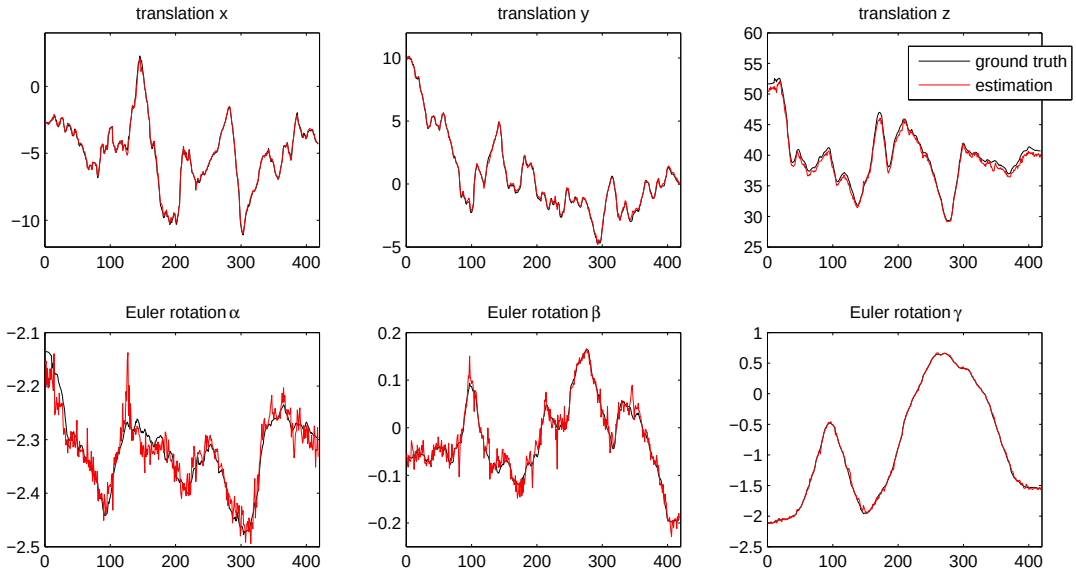



Figure 5.4.: Comparison of the estimated pose and the ground truth of the camera path.

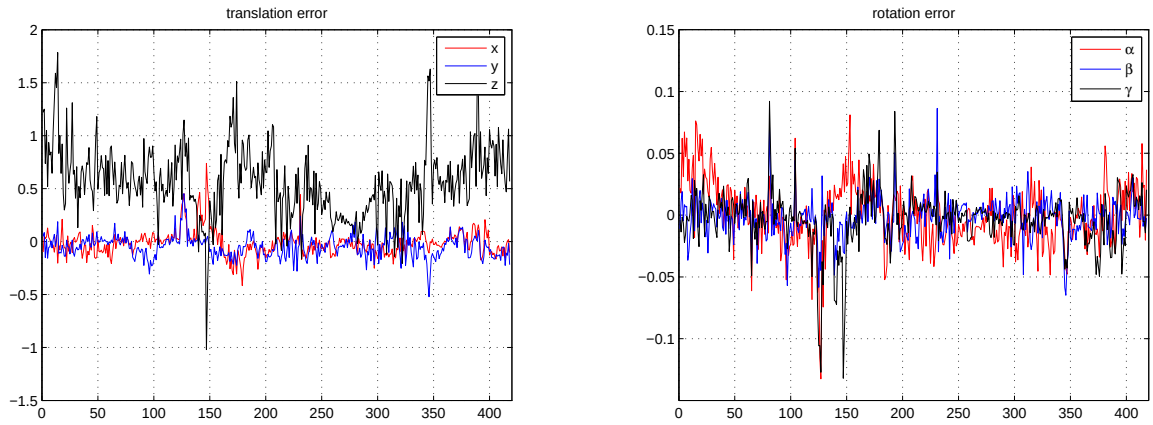
z-direction is significantly larger than the other dimensions. The mean rotation error in radians is 0.0217, which is an average rotation error of 1.242 degrees.

The computational costs of the individual steps are shown in Table 5.2. Retrieving the depth buffer from the GL system requires a major part of processing time. Better performance might be able with newer hardware like PCI Express boards. The creation of correspondences in the prediction step is also very time-consuming, which mostly can be attributed to the normalized cross correlation with sub-pixel accuracy. Together with the image acquisition and visualization the system runs with a frame-rate at 20Hz.

Both the accuracy and the runtime of the tracking highly depends on the resolution of the rendered image, which is used to generate the 3D line model. A comparison of the image resolution and the runtime is shown in Table 5.3. With an increasing image resolution a more detailed 3D line model can be extracted and therefore the result of the pose

| **prediction step** | **time in ms** |
|---|---|
| create correspondences | 10.46 |
| predict pose | 2.12 |
| **tracking step** | |
| render model / read depth buffer | 12.94 |
| extract edges | 6.84 |
| create correspondences | 8.10 |
| estimate pose | 2.42 |
| **total time** | 42.88 |

Table 5.2.: Average processing time of the individual steps.




(a) (b)

Figure 5.5.: Error between the estimated and the real camera pose of a synthetic image sequence. In (a) the components of the camera translation error are plotted, (b) shows the rotation error as the difference of Euler angles.

| resolution | std. dev. (trans/rot) | runtime in ms |
|---|---|---|
| $384 \times 384$ | 0.2499 / 0.0267 | 31.74 |
| $512 \times 512$ | 0.1862 / 0.0217 | 42.88 |
| $768 \times 768$ | 0.1420 / 0.0241 | 81.28 |
| $1024 \times 1024$ | 0.0981 / 0.0116 | 120.41 |

Table 5.3.: Comparison between image resolution, the average standard deviation of the error and the runtime.

estimation gets more precise. As expected the runtime increases, since not only a larger image has to be analyzed in the line model generation step, but also more control points on the contours are extracted and used in the tracking step. To reduce the processing time in the tracking step, the minimum distance between extracted control points can be increased, which would lead to a smaller number of correspondences between control points on the extracted 3D contours and maxima of the image gradient. The length of the edge search and the termination criterion of the minimization also have an influence on robustness and runtime. Altogether the proper choice of the thresholds is a tradeoff between the performance and the accuracy and robustness.

The same sequence of the virtual model of a toy car is used to compare the geometry-based approach with the method using material boundaries as edges. Again the result of the pose estimation is stored and compared with the given ground truth data. It is analyzed how both the results of the edge map generation with geometry edges and with material edges affect the camera pose estimation.

Figure 5.6 shows the error of the 6 extrinsic camera parameters for the different line model generation methods. For this particular example it can bee seen that the method using material edges produces more accurate results. However, this is not surprising, since this method produces a clearer edge map, if correct material properties are given. Again the

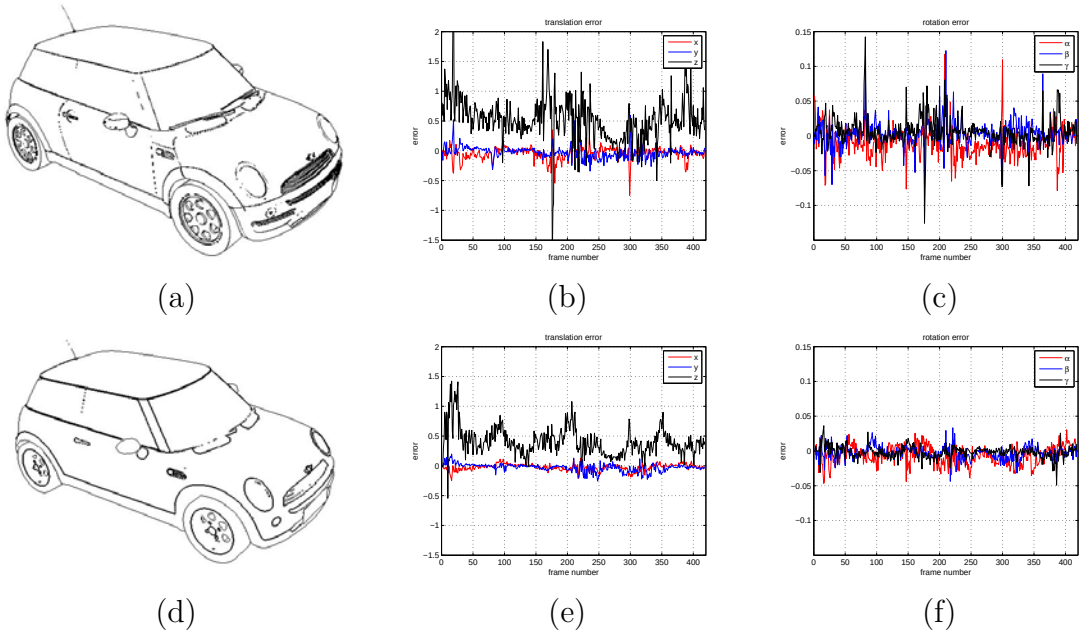



Figure 5.6.: Comparison of the error between the estimated and the real camera pose of geometry edges (a) and material edges (d). In (b) and (c) the error of the geometry edges are plotted, (e) and (f) show the error by using the material edges.

error of most of the parameters alternates around 0, except the camera z-translation, which is clearly above 0. It seems that the estimated camera is further away from the tracked object. As already discussed, the reason for this artefact is that the extracted silhouette edges of the image are are not between the object and the background, but on the object, which produces an error of half a pixel.

An analysis of the processing time is carried out to compute the average computational costs for every individual step. The results are shown in Table 5.4. The difference to the processing time analysis of Table 5.2 is that this time the edge map generation is performed on the GPU with the aid of fragment programs.

Only one 8 bit buffer holding the edge map with the depth information encoded in every pixel is read back from the graphics card to the main memory, which is a significant processing time benefit compared to [89], where both frame-buffer and depth-buffer need to be accessed. Compared to the method, with an edge map generation on the CPU, which required 19.78 milliseconds in total, the approach using the pixel shader only needs 10.90 milliseconds. When a 16 bit depth buffer is used, about 1.9 additional milliseconds are needed for reading back the GL buffer. However, for this synthetic test sequence no significant positive effect on the accuracy of the pose estimation can be observed.

### 5.2.2. Evaluation of Real Image Sequences

The algorithm is tested on several image sequences showing different objects. In the first sequence the tracking approach is tested with an industrial production line. The 3D model

| **prediction step** | **time in ms** |
|---|---|
| create correspondences | 11.42 |
| predict pose | 2.39 |
| **tracking step** | |
| render model / create edge map | 3.98 |
| read GL buffer | 6.92 |
| create correspondences | 8.56 |
| estimate pose | 3.10 |
| **total time** | 36.37 |

Table 5.4.: Average processing time of the individual steps using the edge map generation on the GPU.

Figure 5.7.: Tracking an industrial production line.

consists only of geometric data and no material properties. The edges used for tracking are generated with the combined z-buffer and the normal-buffer method as depicted in Figure 5.2. After initializing the first camera pose manually, it is possible to estimate the correct camera pose throughout the whole sequence. Occluded edges do not appear in the line model, since a new line model is generated in every frame. In Figure 5.7 some frames of this sequence are shown.

In another sequence as shown in Figure 5.8, a model of several pipes was used as an object to be tracked. Again no correct material properties exist, which makes the frame-buffer method inapplicable, and only the geometry-based approach is used. As pipes do not

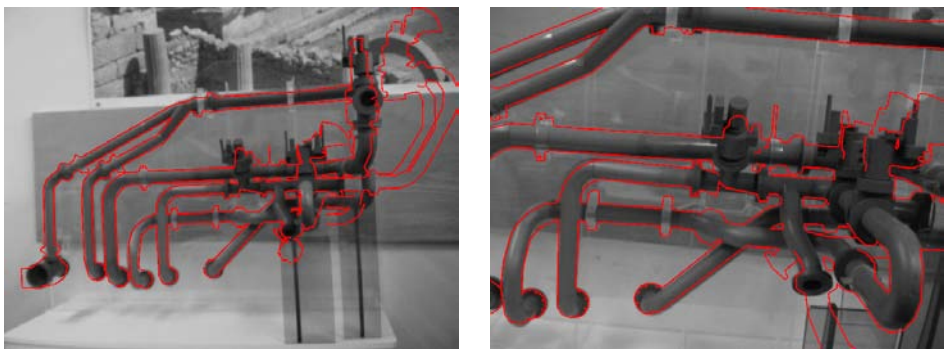


Figure 5.8.: Tracking a pipe model.

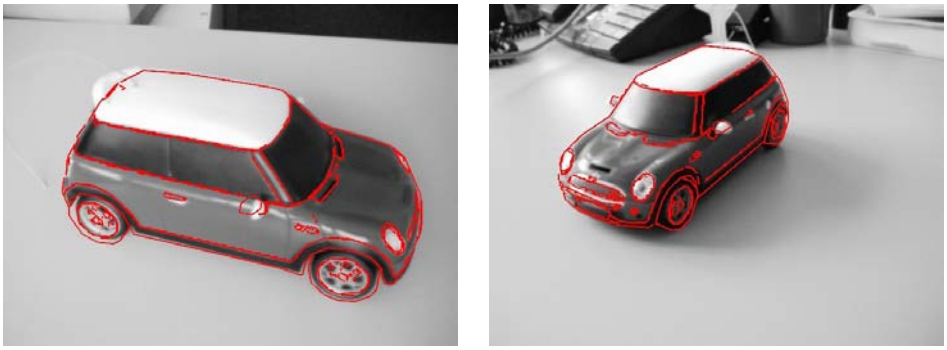


Figure 5.9.: Tracking a toy car by using the frame-buffer approach.

consist of many crease edges, almost all extracted contours are part of the silhouette of the model, which means that the z-buffer method provides the most usable edges in this case. A few pipes of the virtual model do not exist in the real model, but the line model registration is robust enough for a successful tracking through the whole sequence. In such a scenario it is indispensable to create a new line model in every frame, since the position of silhouette edges change, when the camera is moved around the object.

The frame-buffer method is evaluated by tracking the toy car, whose edge map generation is shown in Figure 5.3. The model and a frame of the sequence can be seen in Figure 5.9. Both the method based on geometry and the one based on appearance are tested. For this sequence the frame-buffer based line models produced tracking results with a more accurate camera pose and less jitter. This is not surprising, since the synthetic image sequence has already pointed out that the line model generated by material edges is of a higher quality and produces better tracking results.

All the sequences are tested without the prediction step as well. If large movements, especially fast rotations of the camera occur, the registration step does not produce correct results. The parameters of the camera pose get stuck in local minima and the overall tracking fails. Therefore a rough estimation of the camera pose is really necessary to handle fast camera movements.

## 5.3. Conclusion

In this section a flexible tracking method was presented, which is able to track an object with a given polygonal model by generating 3D contours on the fly and aligning these contours on the image gradient. Our method never runs into any scaling problems, since a line model is generated in every frame with an adequate level of detail. A major improvement is the ability of tracking occluding edges and silhouette edges of objects like tubes or pipes. The advantage of this model-based approach is that no drift can be accumulated during the tracking, since through the model a very significant connection between the virtual and the real world exists. Furthermore, the method is very invariant to lighting changes. No preprocessing steps like the generation of a line model or the calibration of reference images is necessary. Depending on the scene, a method for the

line model generation has to be chosen. If correct material properties exist, the frame-buffer method produces slightly better results. For large camera movements, a prediction of the camera pose for the model generation helps that the pose estimation converges in the proximate frame. If not enough significant edges of the generated line model appear in the image, the tracking gets very unstable. Problems also occur in highly detailed scenes, where the generated line model consists of too many contours, which have only small commonalities with edges in the image.

# 6. Detection and Tracking of Point Features

## 6.1. Tracking vs. Detection

If a scenario consists of textured planar surfaces, point-based tracking methods have been widely used with success for object or camera tracking. In this chapter we assume that a point cloud of 3D points exist. Every point has a unique descriptor, which represents the appearance of the point. In a tracking step correspondences between 3D points and 2D image points are found and the re-projection error is minimized with respect to the camera pose parameters.

This chapter focuses on the detection and the tracking of image points, which is solely a 2D problem. Point-based tracking methods can be classified into two categories. The first group of approaches are local search methods which track a point in the proximity of an area, where the point is assumed to be. A reasonable prediction of its 2D position is necessary to find a point successfully. If the regarded point is located beyond the search space, it usually cannot be found. The other category of approaches does not really rely on tracking points but on detecting them in an image. The difference is that no prediction of the position needed for a local search is necessary, and the point with its feature descriptor can be found in the whole image. This problem is known as wide baseline matching.

In practice the point-based tracking methods relying on a local search are usually used for the tracking of feature points from frame to frame, because they are computationally more efficient than wide baseline matching methods. The tracking by detection approaches are often used for initializing a tracking system, since no previous knowledge about the camera pose is needed.

For a robust detection, matching and tracking of points, the feature point descriptor should be invariant to lighting changes, and different transformations like rotation or scale.

## 6.2. Interest Point Detection

The detection of interest points is both needed for a frame-to-frame tracking and a matching of points, which are extracted in two different images. The requirements for the properties of an interest point detector are very similar in both cases. The area around the point should be textured, so that a significant description of the patch is possible. The points should be different from each other, because ambiguities would perturb the

matching process. A very important criterion is that the detection should be repeatable, which means that the detector should cope noise, illumination changes or aspect changes. Especially for feature point matching it is important that points are located reliably and the detected position does not jitter. For real-time tracking systems, the detection of feature points should be able in an affordable amount of computation time.

Many corner detectors rely on a very similar manner to extract corner features. First a function is applied on the image and computes the corner strength at every pixel. Thresholding the corner strength results in an image where only pixel positions with a strong corner-ness remain. Finally a non-maxima suppression discards all the positions in the image, where the corner strength function does not have a local maximum.

A very popular method is the Harris corner detector [39], which relies on the analysis of the autocorrelation matrix

$$H = \begin{pmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{pmatrix} \tag{6.1}$$

computed at each image location. The elements of $H$ sum over a window of the image gradients, where $I_x$ and $I_y$ are the first derivatives of the image intensity in $x$ and $y$ direction. The window function can be a simple square or circular window. Weighting the derivatives with a Gaussian window function helps to make the detector more robust against noise, but come along with a higher computational effort. Harris defined the corner strength by

$$c = |H| - \kappa(\text{trace}H)^2, \tag{6.2}$$

where $\kappa$ has to be determined empirically. In the literature values in the range from 0.04 to 0.15 have been reported as suitable. The corner strength $c$ can be regarded as a measure of image curvature.

Shi and Tomasi [100] used the smallest of the eigenvalue $\lambda_1$ and $\lambda_2$ of $H$ as corner strength function:

$$c = \min(\lambda_1, \lambda_2) \tag{6.3}$$

They argued that under affine transformation this function of the corner strength is a more accurate measure. They also showed that image locations with two high eigenvalues can be tracked reliably under affine deformations.

The SUSAN corner detector [103] uses a circular mask for detecting corner features.

Lowe [64] used a DoG (Difference of Gaussians) filter to estimate the corner strength of image points. Scale invariance of the detector is achieved by applying the convolving kernel at different resolution levels of an image pyramid. Keypoints of low contrast are rejected if the ratio of the eigenvalues of $H$ is too large and represents a strong gradient in only one direction.

Mikolajczyk and Schmid [73, 74] developed an affine invariant interest point detector, which is based on the Harris corner detector. First the approximate position and the scale are extracted by a multi-scale Harris detector. Then an iterative procedure converges to a stable point and shape, which is invariant under affine transformation.

A descriptor called *Maximally Stable Extremal Regions* was presented by Matas [70]. Instead of regarding corners as interest points regions of uniform intensity, extracted by a watershed algorithm, are taken as image features and matched across different images.

Lepetit and Fua [59] presented a very simple and efficient method to extract corner features by considering the intensities along a circle centered on each candidate point. If the intensity of two diametrically opposed pixels on this circle is approximately the same, the regarded point at the center is not considered as an interest point. The corner strength needed for a the non-maxima suppression is computed by an approximation of the Laplacian of Gaussian.

A similar but and very efficient approach called FAST (Features from Accelerated Segment Test) was presented by Rosten and Drummond [92]. Their method analyses the intensity values on a circle of 16 pixels surrounding the corner point. If at least 12 contiguous pixels are all above or all below the intensity of the center by some threshold, this point is regarded as a corner feature. In [93] a machine learning approach is used to determine some pixel position for an acceleration of the detection test. For reasons of efficiency we use the FAST feature detector in our implementation.

## 6.3. Wide Baseline Matching

If significant points are detected in an image, a description of these feature points has to be extracted. With these feature descriptions a matching between feature points in different images can be performed. A descriptor is composed of pieces of information which are gathered in the neighborhood of the detected point, therefore they are denoted as local descriptors. A database of feature points together with their descriptors can be used to detect and identify objects. In general the descriptor consists of a feature vector of several dimensions. In the matching process the distance between different feature vectors is then used as a matching criterion.

For the purpose of camera-based tracking, wide baseline matching techniques are often used to initialize a tracking system with the aid of some reference data, which can be calibrated reference images or photorealistic 3D models.

### 6.3.1. Local Feature Descriptors

Numerous local feature descriptors exist and all of them have to fulfill some requirements for a successful matching. A local descriptor must be distinct to avoid ambiguities during the matching process. Furthermore, it must be invariant under various transformations. Depending on the application, the feature descriptor is required to be invariant under rotation, scale or a full affine transformation.

A common technique is to use the Normalized Cross Correlation (NCC) of image intensities of a patch that is extracted at the feature position. If the mean intensity is removed and the standard deviation is set to unit length it is equivalent to using the Sum of Squared Differences (SSD) of two extracted patches. Such a descriptor is invariant to

lighting changes, but not invariant to transformations like rotation or scale. Therefore it is mostly used for matching points between two consecutive frames. In [22] such a method is used for the tracking of point features.

Rosten [92] describes the feature vector with image intensities on a circle around the interest point, the same circle which was used for the feature detection.

A feature vector can also be composed by several image filter responses. Schmidt and Mohr [96] use relatively high order image derivatives, where each is rotationally invariant. Therefore the whole feature descriptor is also invariant to rotation transformations.

Feature point matching with an intensity spin image representation was presented by Johnson [50]. A spin image consists of a two-dimensional histogram, where one dimension is the distance from the center and the other dimension the pixel intensity. This descriptor is also rotationally invariant.

Spacial frequency techniques like the Fourier-Mellin transform were used by Stricker [105] to achieve rotation invariance.

Lowe [64] introduced a descriptor called SIFT (Scale Invariant Feature Transform). His idea is to describe an image patch by a histogram of image gradient direction. Rotation and scale invariance is achieved by extracting the descriptor at a canonical scale and orientation. The patch is quantized in a 4 location grid and the gradient direction is quantized into 8 orientations. This leads to a feature vector of 128 elements.

Ke and Sukthankar [53] claimed that this description vector size is larger than necessary. They perform a Principal Component Analysis (PCA) on all the extracted feature vectors and only the top 20 basis vectors are taken as descriptor.

Another variation of SIFT is presented by Mikolajczyk and Schmid [75]. Their method called GLOH (Gradient Location Orientation Histogram) computes a gradient histogram on a log-polar location grid with three bins in radial direction. With 17 location bins and 16 bins of the quantized gradient orientations a histogram with 272 bis is created. The dimensionality is also reduced by a PCA to a description vector of 128 elements.

Grabner et. al [34] speed up the detection process of SIFT features by using an integral image for scale space computation.

SURF (Speeding Up Robust Features) is another method presented by Bay et al. [8], which also takes advantage of an integral image for speeding up the detection and extraction of a feature point. The approach is similar to SIFT, but instead of gradient orientation histograms the sum of Haar wavelet responses in horizontal and vertical direction is calculated on a $4 \times 4$ grid of subregions of the patch.

A comparison between different descriptors is carried out by Mikolajczyk [75], and he concludes that SIFT-based descriptors perform best.

### 6.3.2. Feature Matching Strategies

For most of the feature descriptors the Euclidean distance is used as a similarity measure to create matches. Comparing every feature descriptor of one image with every feature descriptor of another image has a computation complexity of $O(n^2)$.

Algorithms like kd-trees do not provide any speedup if the dimensionality of a feature vector is too high. Lowe [64] proposes a Best-Bin-First alternative, which is an approximate in the sense that it returns the nearest neighbor in feature space with high probability.

Rosten [92] sorts the feature vectors by their mean intensity. A binary search is performed first to find the feature vector with the closest mean, followed by a linear search for the feature with the smallest SSD. An early exit strategy speeds up the matching process by discarding a match if the maximum SSD error is exceeded during the computation of the sum.

### 6.3.3. Classification Techniques

Lepetit et al. [61] treat the wide baseline matching of keypoints as a classification problem, where each class corresponds to a feature point. A set of image patches of a point under various different transformations is used as a training set for this supervised learning technique.

In [61] the dimensionality of the image patches is reduced by a PCA and the view sets of each feature are clustered using k-means. A nearest neighbor classifier is used to match a feature point of another image. The speed of the system was improved by taking randomized trees as a weak classifier [60]. A feature is then detected by combining the classification results of all the trees. The method is very fast and robust against illumination changes and various transformations. However, the classifier has to be trained with all the transformations which the system is supposed to detect. A complex preprocessing is required to train the system, since the computational burden is shifted to a training phase.

An online version to learn randomized lists for reinitializing the camera tracking is presented by Williams et al. [117]. The classifier is trained during the tracking in an extra thread, which runs in the background, and after the camera tracking gets lost, the tracking is initialized by recognizing the keypoints by classification.

## 6.4. Optical Flow-Based Tracking

Optical flow is the apparent motion of gray value structure in consecutive images. The estimation of the optical flow is based on the assumption of intensity conservation over time. If $I(\boldsymbol{x}, t)$ is the intensity value at the image position $\boldsymbol{x}$ at time $t$ and $\boldsymbol{d}$ the motion translation vector, this assumption can be expressed as

$$I(\boldsymbol{x}, t) = I(\boldsymbol{x} + \boldsymbol{d}, t + dt). \tag{6.4}$$

In 1981 Lucas and Kanade [66] proposed one of the most popular methods for the local estimation of optical flow. Many image alignment techniques nowadays rely on their principle of minimizing image intensity differences. A solution for the displacement vector $\boldsymbol{d}$ is estimated by minimizing the residue error defined by

$$\epsilon = \sum_{x} [I(\boldsymbol{x} + \boldsymbol{d}, t + dt) - I(\boldsymbol{x}, t)]^2. \tag{6.5}$$

An iterative Newton-Raphson style algorithm is performed to solve this minimization problem. As this optical flow estimation approach is a local search method, the convergence radius is very limited. To achieve a higher possible range, where image points can be tracked successfully, the Lucas and Kanade algorithm is often applied on several levels of an image pyramid. With this coarse-to-fine technique the convergence range is duplicated with every level of the image pyramid level. The image borders have to be handled with special care, and the intensity difference must only be summed up on areas in the image, which are visible in both image frames. A window function, e.g. a bell-like Gaussian function, can be used to give pixels in the middle of an image patch more influence than pixels near the border of the regarded area.

Whereas in the original approach the optical flow estimation was used for image registration, Tomasi and Kanade [108] use the same approach for tracking point features from frame to frame. This approach, however, only calculates the translation vector at an image point. The tracking of feature points from frame to frame is prone to considerable drift and has therefore only limited capabilities for a robust camera pose estimation.

## 6.5. Template-Based Tracking

Shi and Tomasi [100] extend the method of Lucas and Kanade for affine image transformations. A quadratic patch is extracted around a feature point when it is observed first and then it is used as a reference template. They also address the problem of the detection of feature points which can be tracked stably under the affine transformation model. In regions where image structure exists in only one direction the full optical flow cannot be estimated. This problem is known as the aperture problem. Shi and Tomasi argue that image structure has to be present in all image directions and therefore use an interest point detector as already described in Section 6.2, where the smallest eigenvalue of the structure matrix has to be significantly large. This tracking method where a template patch is tracked under an affine transformation model is also denoted as the Kanade-Lucas-Tomasi (KLT) tracker in the literature.

A more general algorithm for tracking a template in a gray value image is presented by Hager and Belhumeur [35]. A geometric warping function $g(\boldsymbol{x};\boldsymbol{p})$ is used to represent more general warps than simple translations. They also switch the role of the template and the image for efficiency reasons and call this method the inverse additive approach. The efficiency benefit results from the shift of computation from the tracking step to the pre-processing step. If $I(\boldsymbol{x},0) = T(\boldsymbol{x})$ is the reference template and $g(\boldsymbol{x};\boldsymbol{p})$ the warp function of an image point $\boldsymbol{x}$ with the parameter vector $\boldsymbol{p}$, the error to be minimized for the forward additive approach is written as

$$\epsilon = \sum_x [I(g(\boldsymbol{x};\boldsymbol{p}) - T(\boldsymbol{x})]^2. \tag{6.6}$$

To optimize this expression, the parameter increment $\boldsymbol{\Delta p}$ is iteratively estimated by minimizing

$$\epsilon = \sum_x [I(g(\boldsymbol{x};\boldsymbol{p}+\boldsymbol{\Delta p}) - T(\boldsymbol{x})]^2, \tag{6.7}$$

where after every iteration the parameter vector is updated by $\boldsymbol{p} \leftarrow \boldsymbol{p}+\boldsymbol{\Delta p}$. The algorithm stops if $\boldsymbol{\Delta p}$ is insignificantly small or a maximum number of iterations has been reached. The inverse additive algorithm simply switches the role of the image $I$ and the template $T$.

A compositional approach of Shum and Szeliski [101] iteratively estimates an incremental warp $g(\boldsymbol{x}, \boldsymbol{\Delta p})$ by minimizing

$$\epsilon = \sum_x [I(g(g(\boldsymbol{x}; \boldsymbol{\Delta p}); \boldsymbol{p})) - T(\boldsymbol{x})]^2, \tag{6.8}$$

with respect to $\boldsymbol{\Delta p}$. The warp function is updated with $g(\boldsymbol{x}, \boldsymbol{p}) \leftarrow g(\boldsymbol{x}, \boldsymbol{p}) \circ g(\boldsymbol{x}, \boldsymbol{\Delta p})$.

Baker and Matthews [5] introduce another method denoted as the inverse compositional approach for tracking an image template. They approximately minimize

$$\epsilon = \sum_x [T(g(\boldsymbol{x}; \boldsymbol{\Delta p})) - I(g(\boldsymbol{x}; \boldsymbol{p}))]^2, \tag{6.9}$$

with respect to $\boldsymbol{\Delta p}$. After every iteration the update of the warp function is computed by $g(\boldsymbol{x}, \boldsymbol{p}) \leftarrow g(\boldsymbol{x}, \boldsymbol{p}) \circ g(\boldsymbol{x}, \boldsymbol{\Delta p})^{-1}$.

Baker and Matthews show that this inverse compositional method is capable of handling a more general set of warps, especially homographies, whereas the inverse additive approach of Hager and Belhumeur can only be used with affine warps. The inverse compositional approach is the most general and efficient method and is therefore most commonly used nowadays.

To solve the minimization problem of equation (6.9) a first order Taylor expansion is performed and results in

$$\epsilon = \sum_x [T(g(\boldsymbol{x}; \boldsymbol{0})) + \nabla T \frac{\partial g}{\partial \boldsymbol{p}} \boldsymbol{\Delta p} - I(g(\boldsymbol{x}; \boldsymbol{p}))]^2. \tag{6.10}$$

Solving for $\boldsymbol{\Delta p}$ yields

$$\boldsymbol{\Delta p} = H^{-1} \sum_x \left[ \nabla T \frac{\partial g}{\partial \boldsymbol{p}} \right]^T [I(g(\boldsymbol{x}; \boldsymbol{p}) - T(\boldsymbol{x})], \tag{6.11}$$

where the Hessian matrix $H$ is computed by

$$H = \sum_x \left[ \nabla T \frac{\partial g}{\partial \boldsymbol{p}} \right]^T \left[ \nabla T \frac{\partial g}{\partial \boldsymbol{p}} \right]. \tag{6.12}$$

The Jacobian $\frac{\partial g}{\partial \boldsymbol{p}}$ is evaluated at $(\boldsymbol{x}, \boldsymbol{0})$. The efficiency of the inverse compositional algorithm comes from the fact that the Hessian matrix $H$ does not depend on the parameter vector $\boldsymbol{p}$. Therefore it is constant during the iterations and the matrix $H^{-1}$ can be pre-computed. However, if the intensity values of the template $T$ or the size of the area are changed during the minimization, the Hessian must be re-computed. The template must be therefore always fully located inside the image and cannot be changed so that efficiency benefits of the algorithm pay off.

More detailed deviation for different warp functions are given in Appendix A. Another very detailed explanation and comparison of these different template-based tracking methods can also be found in [6].

### 6.5.1. Illumination Compensation

If planar patch features are tracked with a long lifetime, severe illumination changes can occur. This happens because a patch is viewed from a different viewing direction and the lighting changes or simply because a camera controller is auto adjusting the shutter or the gain of the camera. Therefore in most real-life scenarios it is indispensable to regard changes of illumination.

Tommasini et al. [109] use a photometric normalization with a zero mean SSD residual computation to make the tracking more robust against lighting changes. They also add a robust rejection rule to detect tracking failures.

Another method for illumination compensation described by Hager and Belhumeur [35] uses a set of illumination basis images, which are all captured under different illumination. An additional parameter per basis image is used to describe the contribution of every basis image to the current image.

Zhu el al. [123] achieve lighting invariance by minimizing the differences of normalized gradient images instead of intensity discrepancies.

A precise photometric model for a transformed template patch is presented by Jin et al. [49]. They extend the Shi-Tomasi tracker by two additional illumination parameters. With the contrast compensation factor $\lambda$ and the brightness correction $\delta$ they minimize the following error function with respect to $\boldsymbol{\Delta p}$:

$$\epsilon = \sum_x [\lambda I(g(\boldsymbol{x}; \boldsymbol{p} + \boldsymbol{\Delta p}) + \delta - T(\boldsymbol{x})]^2 \tag{6.13}$$

However, this forward additive formulation has a lack of efficiency, since the Hessian matrix $H$ needs to be recomputed in every frame.

Zinßer et al. [124] combine the benefits of the more efficient inverse compositional approach with the additional illumination parameters presented by [49]. This leads to minimize the error function which is given by

$$\epsilon = \sum_x [\tilde{\lambda} T(g(\boldsymbol{x}; \boldsymbol{\Delta p})) + \tilde{\delta} - (\lambda I(g(\boldsymbol{x}; \boldsymbol{p})) + \delta)]^2, \tag{6.14}$$

where $\tilde{\lambda}$ and $\tilde{\delta}$ are the incremental changes of the illumination parameters. The values $\lambda$ and $\delta$ are still those parameters that $T(\boldsymbol{x}) = \lambda I(g(\boldsymbol{x}; \boldsymbol{p})) + \delta$ holds.

Again the first-order Taylor expansion around the identity warp $g(\boldsymbol{x}, \boldsymbol{p})$ is used to approximate this error function:

$$\epsilon = \sum_x [\tilde{\lambda} T(g(\boldsymbol{x}; \boldsymbol{0})) + \tilde{\delta} + \tilde{\lambda} \nabla T \frac{\partial g}{\partial \boldsymbol{p}} \boldsymbol{\Delta p} - (\lambda I(g(\boldsymbol{x}; \boldsymbol{p})) + \delta)]^2. \tag{6.15}$$

This equation can be rewritten in matrix form by

$$\epsilon = \sum_x [h(\boldsymbol{x})^T q - (\lambda I(g(\boldsymbol{x}; \boldsymbol{p})) + \delta)]^2 \tag{6.16}$$

with

$$h(\boldsymbol{x}) = \left( \nabla T(\boldsymbol{x}) \frac{\partial g}{\partial \boldsymbol{p}} \quad T(\boldsymbol{x}) \quad 1 \right)^T \tag{6.17}$$

$$q = \left( \tilde{\lambda} \Delta \boldsymbol{p}^T \quad \tilde{\lambda} \quad \tilde{\delta} \right)^T \tag{6.18}$$

The size of the vectors $h(\boldsymbol{x})$ and $q$ is 2 elements larger than the number of parameters of the vector $\boldsymbol{p}$.

Finally by solving the least squares problem of equation (6.16) the parameter update vector can be computed by

$$q = \left( \sum_x h(\boldsymbol{x}) h(\boldsymbol{x})^T \right)^{-1} \left( \sum_x h(\boldsymbol{x}) \lambda I(g(\boldsymbol{x}; \boldsymbol{p})) + \delta) \right). \tag{6.19}$$

Since the vector $h(\boldsymbol{x})$ is independent from all parameters $(\boldsymbol{p}, \lambda, \delta)$ and the current image $I$, the inverse of the matrix composed of the dyadic products of $h(\boldsymbol{x})$ can be precomputed for every feature and then be re-used in every frame.

After every iteration the contrast parameter $\lambda$ is updated with $\lambda \leftarrow \frac{\lambda}{\tilde{\lambda}}$ and an update of the brightness parameter $\delta$ is computed by $\delta \leftarrow \frac{\delta - \tilde{\delta}}{\tilde{\lambda}}$. An example of an affine transformation model as it is presented in [124] can be found in Appendix A.

## 6.5.2. Drift Prevention

Tracking only the translation of a feature point is very fast, since the image border handling is easy and the computation of the inverse of the $2 \times 2$ matrix $H$ is simple. But as the estimation of the displacement vector can never be absolutely accurate, tracking only the translation of a point will produce feature drift. Therefore it is necessary to take the intensity values of the initial patch into account to track the feature and to monitor, whether the feature point kept its visual appearance. A template-based illumination invariant method as described in the previous section prevents the accumulation of drift, since the reference template is never changed. However, there are also some disadvantages, if only a reference template is used. For a successful alignment a complex model like the affine illumination invariant algorithm of Jin et at. [49] is necessary. Because of the high dimensionality of the parameter space such methods can easily run into convergence problems. If the initial parameter values are not close enough to the solution, the Newton-Raphson iterations do not converge, especially if brightness and contrast are estimated as well. It is also difficult to determine the borders of a warped patch, if it is not totally inside the image.

Zinßer et al.[124] use a two-stage approach to solve this problem. Pure translation from frame to frame is estimated first, then this translation vector and the affine and illumination parameters of the previous frame are used to initialize the minimization of the discrepancy of the initial patch and the current frame. This method has a much higher rate of tracking successes since the feature position is already almost correct, when the reference template is aligned. In our implementation we achieved good results with 5 levels of a Gaussian pyramid for tracking the translation from frame to frame and only one pyramid level to track the initial patch with the affine illumination invariant method.

## 6.6. Improvements

### 6.6.1. Multiresolution Tracking

If the scale of the warp function $g(\boldsymbol{x};\boldsymbol{p})$ does not change significantly during the tracking, the initial patch, which was extracted from the lowest image pyramid level, usually converges during the minimization. However, if the scale gets too small or too large, sampling errors of the image intensities become too large and the patch registration may not converge. To avoid this problem, we propose an approach which uses patches of different resolution levels for tracking. When a feature occurs first, a patch window is cut out of the image pyramid at every pyramid level. For the affine brightness invariant patch registration, we always select the patch which has the most similar resolution to the predicted warp.

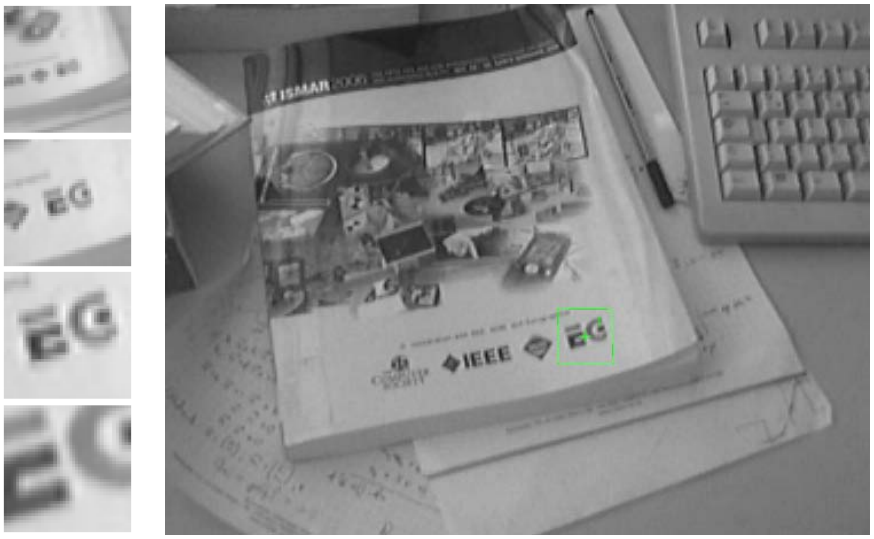
For an affine model we use the determinant of the affine transformation $A$ as a scale measure. If $l$ shall be the resolution level of the patch to be selected for registration, the following inequality must hold:

$$t \leq 2^l|A| < 2t, \tag{6.20}$$

where $t$ is a threshold which defines the decision boundary between two resolution levels. For the affine image registration a patch of the resolution level $l$ has to be warped with the matrix $A' = 2^l A$. In our implementation we choose $t = 0.8$, so the norm of the affine transformation $A'$ of a patch is always between 0.8 and 1.6.
If a feature becomes too small during the tracking, a patch with a lower resolution, which was extracted from a higher image pyramid level, is used.

A patch of higher resolution, which was extracted from a lower image pyramid level, should be used, if the scale of the affine transformation becomes too large. Since patches with a resolution higher than the bottom image of the image pyramid do not exist from the beginning of the feature track, the stack of patches has to be extended with patches of higher resolutions during the tracking. If the scale of a patch is more than twice as large as the initial warp, then, presuming the feature has been tracked successfully in the current frame, we insert a new patch of the regarded point at the bottom of the patch stack. The new patch is extracted using the current warp and brightness parameters and the new warp function is scaled appropriately. With the affine invariant model the affine matrix is simply multiplied by 0.5. This is necessary, since all patches of a feature need to have the same basis of transformation parameters. This approach enables the back-matching under a strong increase of scale. With back-matching we denote the procedure of comparing the current tracked image patch to an initial reference patch, which was acquired when the feature was seeded.

An illustration of a single tracked feature with the affine illumination invariant method is presented in Figure 6.1. A frame of a test sequence is shown together with 4 resolution levels of the patch template. Here we use the two-stage approach with the affine illumination invariant model, which is described in Section 6.5.2. In Figure 6.2 the regarded patch in different frames is illustrated. The first row (a) simply shows the feature point in the current frame, i.e. the patch which is extracted for the frame-to-frame optical flow




(a) (b)

Figure 6.1.: Example of tracking a single patch. In (a) the stack of reference templates in different resolution can be seen, (b) shows a frame of the test sequence together with the tracked patch. In this frame the second resolution level is used.

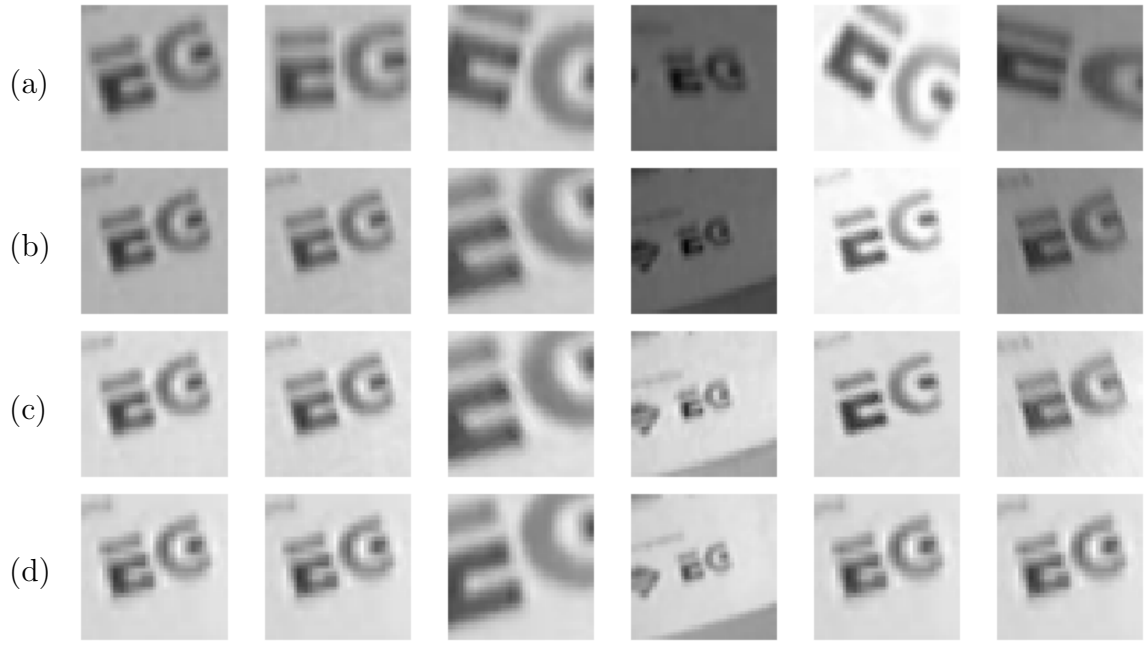
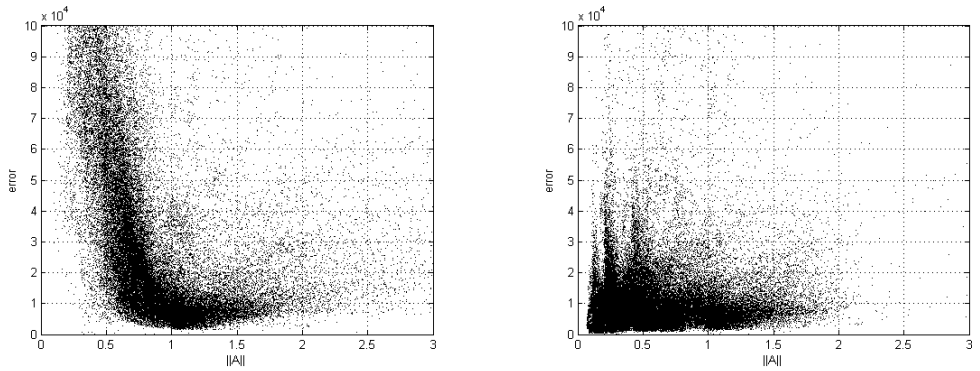



Figure 6.2.: Illustration of the lighting invariance patch tracking with multiple resolution levels. In (a) the patch in the current frame is drawn, (b) shows the extracted affine invariant regions, (c) the extracted patches with the additional illumination compensation and in (d) the reference templates of the currently used resolution level are shown for comparison.




(a) (b)

Figure 6.3.: The residue vs. the scale of the affine warping matrix. (a) shows the results of the single-scale, (b) of the multi-scale approach.

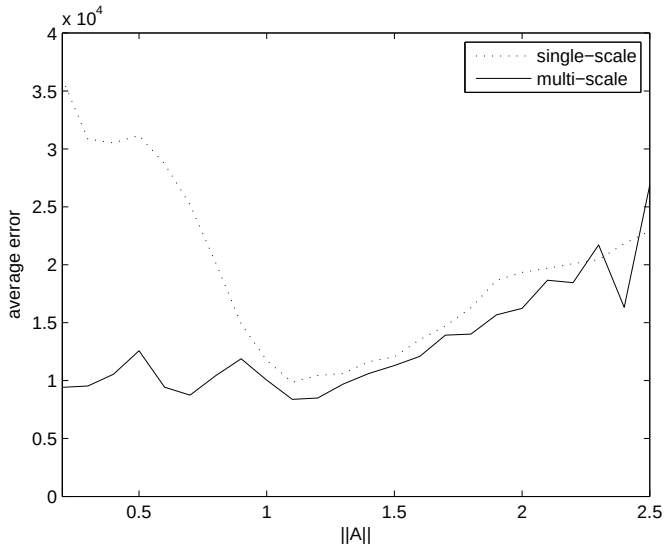



Figure 6.4.: The average error vs. the scale of the affine warping matrix.

estimation. In the second row (b) the affine invariant region of the feature in the current frame can be seen. The extracted patches with the affine model and illumination compensation can be seen in (c) and for the comparison the reference template of the used resolution level in row (d).

To evaluate the approach, we captured a sequence with strong changes in scale and plotted the intensity residuals against the scale of the affine warping matrix for every tracked feature point. Figure 6.3 shows that the errors produced by the single-resolution approach become larger especially for a decreasing scale. The reason for this is that sampling errors grow significantly if a patch is warped into a very small region. Using the multi-resolution approach, patches corresponding to higher image pyramid levels are selected if the predicted warping is too small, and therefore the error does not increase for affine transformations of a smaller scale.

Figure 6.4 shows the average residue of a successfully tracked feature according to its scale. It can be clearly seen that the average error of all tracked features is smaller for the multi-scale approach, which makes it much easier to distinguish, whether a feature has been tracked successfully or whether the tracking step has failed.

If a feature cannot be tracked successfully, its position is predicted as described in Section 7.4. Thus, temporarily lost features have a good chance to be tracked again in the succeeding frames.

### 6.6.2. Updating the Template

Updating the template is often avoided because small alignment errors can accumulate and the template might drift away from the initially extracted patch. However, the initial patch is not always the best visual representation of a surface, since reflections, shadows or strong camera noise can result in a poor representation of an extracted patch.

Segvic et al. [98] use a running average Gaussian estimation of every pixel's gray value of the image area which the template is aligned with. Thereby the template is not compared directly with the intensity values of the current image, but with a temporarily smoothed image region. Due to this noise suppression the residual error is reduced, but the original template remains untouched.

Matthews et al. [71] propose a strategy, in which the template is updated if the parameter difference of the warps from the initial template and the current template is smaller than a given threshold. Thereby the reference template is replaced by the current region of the image if the parameters of the alignment to the initial patch do no differ significantly from the warp parameters to the current template.

In our system we propose a method which does not replace the whole reference template, but updates the template image by calculating an incremental intensity mean for every pixel. To avoid drift, an update is only performed if the alignment was successful, i.e. if the SSD between reference patch and current patch is small enough.

If $C(\boldsymbol{x})$ is the number of measurements, which contributed to calculate the mean intensity of a pixel, the incremental mean can be computed with

$$T(\boldsymbol{x}) = \frac{1}{C(\boldsymbol{x}) + 1} \left[ (I(g(\boldsymbol{x}; \boldsymbol{p})) + C(\boldsymbol{x}) T(\boldsymbol{x}) \right]. \tag{6.21}$$

After updating a pixel of the template the value $C(\boldsymbol{x})$ is incremented by 1. Every pixel of the reference patch needs its own contribution counter $C$, because it is not guaranteed that the whole patch can always be updated. This might happen, if the whole patch does not lie completely inside the image, which is often the case especially on higher image pyramid levels. With this update method is is not only possible to refine intensity values of the initially extracted patch, but also to extend areas of a patch which could not be initialized, because parts of it were outside the image, when the feature was observed first.

Since the contribution of the current image intensities gets lower with every update, the influence of the first images never gets lost and drift is avoided.

If the camera does not move, applying an update of the template in every frame may not lead to the desired results, since a feature is always observed from the same viewing direction and reflections might incorporate into the reference template of the patch. We only perform an update if a significant amount of camera translation has occurred since the last template update. Thereby it is guaranteed that the input for an update step is the appearance of a patch of a different viewing position.

A disadvantage of updating the template is that the intensity gradients and the inverse of Hessian matrix, which is needed for the alignment process, needs to be recomputed after every update.

### 6.6.3. Robust Image Alignment

If intensity discrepancies are added over a whole patch area as in equation (6.13), it is assumed that the patch is completely planar, that there are no occlusions and that there is a global linear change of illumination. Neither of these assumptions is always true in real life scenarios. Patches are not always totally part of a planar surface, areas of the scene can be occluded by interacting persons or by other objects, and spotlights or reflections cannot be modeled by a global linear lighting model.

Therefore the detection of outliers during the template tracking is worth considering. Hager and Belhumeur [35] use a robust estimator function for the detection of occlusions. They modify the error function by solving a robust optimization problem of the form

$$\epsilon = \sum_x \rho(I(g(\boldsymbol{x};\boldsymbol{p}) - T(\boldsymbol{x}))), \tag{6.22}$$

where $\rho$ is one of a wide variety of robust estimator functions [122]. Instead of the estimator function $\rho$ the same problem can also be expressed with a weighting matrix $M(\boldsymbol{x})$. The authors also used morphological operators to remove outliers of the weighting matrix and showed that with this robust optimization approach a more stable tracking is possible under partial occlusion of the template. An evaluation of different estimator functions has been carried out by Theobald et. al. [107].

Baker and Matthews [4] also used a weighting matrix for a more stable and more efficient minimization. Stability comes from only taking pixels with a high confidence into account, and efficiency results from the fact that computation costs can be reduced if only the most reliable pixels are selected.

They also present an iteratively re-weighted least squares algorithm for the inverse compositional approach With the weighting matrix $M(\boldsymbol{x})$ the problem can be expressed by

$$\epsilon = \sum_x M(\boldsymbol{x})[T(g(\boldsymbol{x};\boldsymbol{\Delta p})) - I(g(\boldsymbol{x};\boldsymbol{p}))]^2. \tag{6.23}$$

Solving for $\boldsymbol{\Delta p}$ yields

$$\boldsymbol{\Delta p} = H^{-1} \sum_x M(\boldsymbol{x}) \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right]^T [I(g(\boldsymbol{x};\boldsymbol{p}) - T(\boldsymbol{x})], \tag{6.24}$$

where the matrix $H$ is computed by

$$H = \sum_x M(\boldsymbol{x}) \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right]^T \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right]. \tag{6.25}$$

After every iteration the weighting matrix $M(\boldsymbol{x})$ is estimated according to the current residual.

The drawback of this iteratively re-weighted approach is the loss of efficiency. If the weights are re-estimated in every iteration, the matrix $H^{-1}$ cannot be precomputed any more, but must also be calculated in every iteration.

Ishikawa et al. [48] avoid the re-computation of the whole Hessian by subdividing the image into a grid of blocks and pre-compute Hessian for every grid element. Blocks of outliers are determined and the sum is only calculated over valid block elements. This approach is more efficient, since the sum over every block element can be pre-computed.

The iteratively re-weighted least squares approach is only beneficial, if really large templates are tracked, e. g. a whole face of a person as in [35]. For the purpose of tracking feature points for camera pose estimation, it is more advantageous to use smaller templates, because many industrial scenarios seldom consist of large planar surfaces, and if patches are smaller, more features can be tracked with the same computational costs.

Therefore in our tracking system we use a larger set of small templates and reject a feature patch completely if the tracking has failed. For efficiency reasons we rather recomputed the weights only after a successful tracking step, because the inverse of the $H$-matrix only needs to be calculated once per frame and not in every iteration of the feature tracking step.

To achieve lighting invariance we integrate a weighting matrix into the illumination invariant method of the inverse compositional approach. The term to minimize for the robust alignment can then be written as

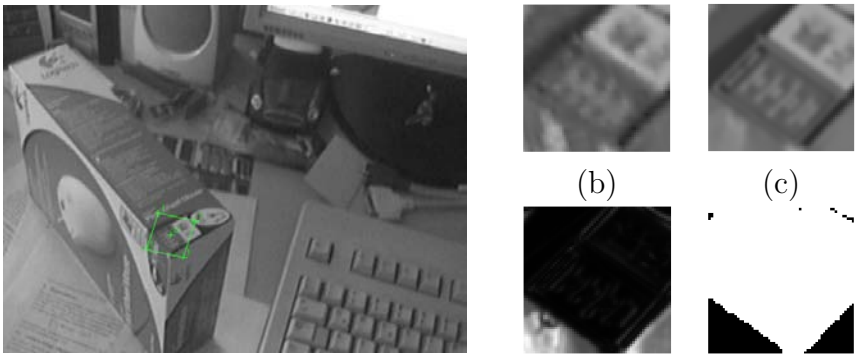
$$\sum_x M(\boldsymbol{x}) \left[(\lambda T(g(\boldsymbol{x}; \boldsymbol{\Delta p})) + \delta - I(g(\boldsymbol{x}, \boldsymbol{p}))\right]^2. \tag{6.26}$$

The parameter update can be similarly computed as in Section 6.5.1. With the vector $h(\boldsymbol{x})$ of equation (6.17) the new parameter vector $q$ can be computed by

$$q = \left(\sum_x M(\boldsymbol{x}) h(\boldsymbol{x}) h(\boldsymbol{x})^T\right)^{-1} \left(\sum_x M(\boldsymbol{x}) h(\boldsymbol{x}) \lambda I(g(\boldsymbol{x}; \boldsymbol{p})) + \delta)\right). \tag{6.27}$$

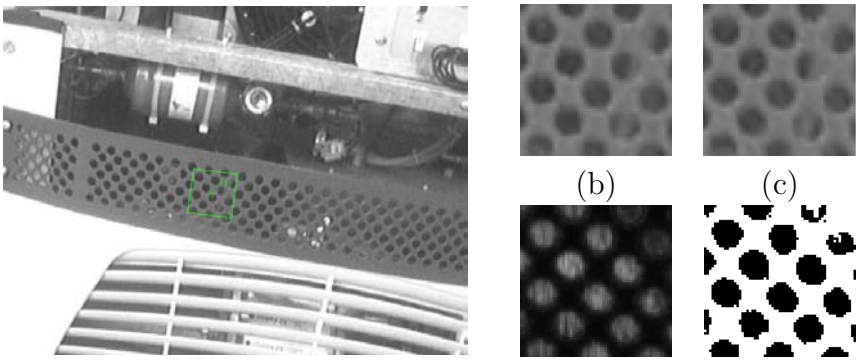
Our goal for a robust feature tracking is not to track a template under partial occlusion and extreme lighting variation, but to acquire a valid area of the patch, which is a stable representation of the patch, and to use only those areas for the alignment. If some areas of a template, for example, are not part of the planar surface, these pixels should always be regarded as outliers and not contribute to the template alignment.

With a given weighting matrix $M(\boldsymbol{x})$, which assigns every pixel an influence value for the minimization result, the computation of the parameter vector increment is only slightly more expensive.



(a) (b) (c) (d) (e)

Figure 6.5.: Illustration of the mask generation. In (a) the scenario can be seen, (b) shows the currently extracted patch feature, (c) the incremental mean, (d) the variance and (e) the mask of the patch.



(a) (b) (c) (d) (e)

Figure 6.6.: Another example of the mask generation in an industrial scenario. Again the current patch (b), the incremental mean (c), the variance (d) and the mask of a patch (e) can be seen. The mask clearly distinguishes pixels from the planar surface from the background.

To simplify the computation we do not use a weighting matrix, but a binary mask to select the pixels, which are taken into account for the feature tracking. Pixels where the value of the binary mask is 0 are not regarded at all. The overall computational cost can therefore be decreased if a binary mask is used and many pixels are masked out.

A binary mask can also be used to integrate only over those areas of a patch which are located inside of an image. If a patch is extracted out of an image, it happens especially at higher pyramid levels that the template is not completely located inside the current image, and some parts do not contain any valid intensity information. By setting the mask values $M(\boldsymbol{x})$ of these pixels to 0, only valid pixels are taken for the template tracking.

### 6.6.4. Template Mask Generation

With a patch mask $M(\boldsymbol{x})$ it is possible to minimize the intensity discrepancy between only those pixels of an image and a patch which are really part of the planar surface. The acquisition of such a mask $M(\boldsymbol{x})$ is described in this section.

Sometimes an extracted patch does not always lie totally on a planar surface. Pixels which are not part of the surface are not a useful contribution for the alignment step. For the acquisition of a high quality template it is desired to generate a mask for a patch which selects only those pixels which really lie on the planar surface of an object. Our approach for the generation of a template mask relies on the analysis of the intensity variance of a pixel. Similar to the incremental mean computation, the update of the intensity variance $S^2(\boldsymbol{x})$ of a pixel can be approximated by

$$S^2(\boldsymbol{x}) = \frac{1}{C(\boldsymbol{x})+1}\left[((I(g(\boldsymbol{x};\boldsymbol{p})) - T(\boldsymbol{x}))^2 + C(\boldsymbol{x})S^2(\boldsymbol{x})\right]. \tag{6.28}$$

To decide if a pixel is used for the tracking step, a mask $M(\boldsymbol{x})$ is created by

$$M(\boldsymbol{x}) = \begin{cases} 1 & \text{if } S^2(\boldsymbol{x}) < c, \\ 0 & \text{otherwise.} \end{cases} \tag{6.29}$$

A good value for the threshold $c$ depends on the camera noise and must be chosen experimentally.

In Figure 6.5 a single reference template, its incremental mean, the variance and the generated mask are shown. After moving around the camera for a while, the variance of areas which are not part of the patch feature's plane increases significantly and the patch mask clearly represents only the planar part of the template. Another example is illustrated in Figure 6.6.

With such a generated template mask it is possible to detect only the truly planar regions of a patch, and only those areas of a patch can be taken into account for the template alignment which clearly belong to a planar surface.

This only works if the surroundings of the patch which are not part of the surface do not consist of homogeneous areas. If this were the case, the intensity variance of a pixel would be small and those pixels would not be masked out.

However, if the surroundings are homogeneous, e.g. consist of a white wall in the background or completely black holes in a surface, they do not have a large effect on a misalignment of a template patch.

## 6.7. Camera Tracking Applications with Point Features

### 6.7.1. Poster Tracker

An easy way to set up a markerless augmented reality application is simply to replace the artificial marker with a more natural marker like a well textured poster. If a reference

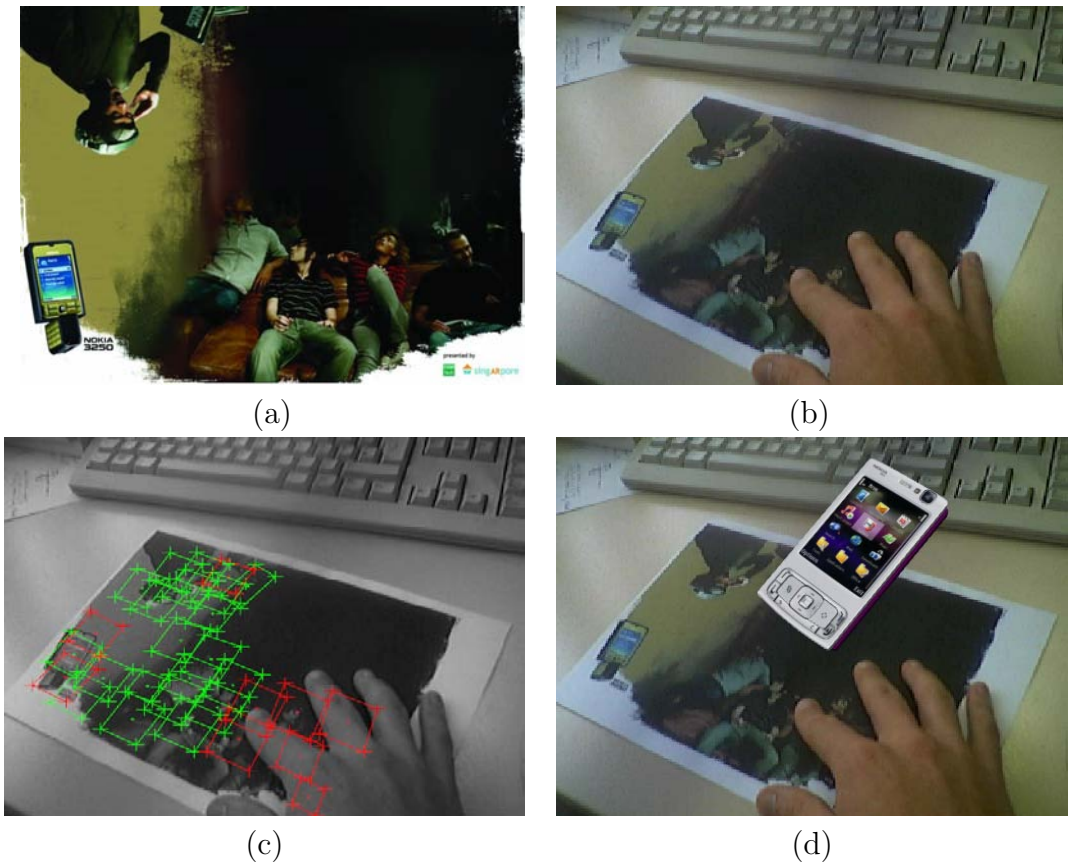
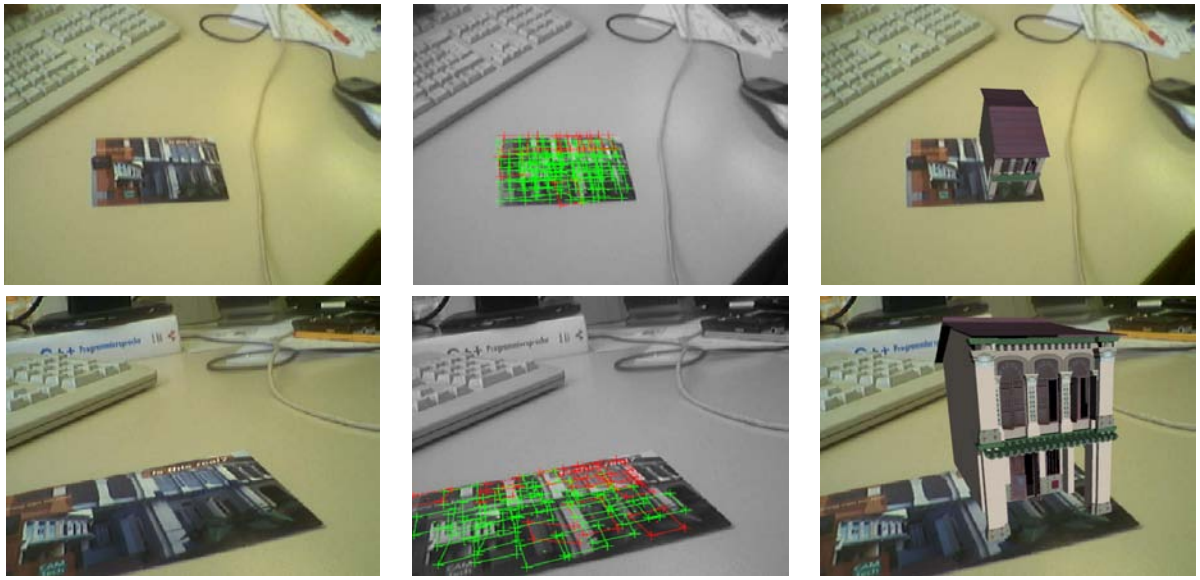


Figure 6.7.: Tracking a poster with partial occlusion. In (a) the reference image is shown, in (b) a current frame of a camera image can be seen. The tracked template patches are overlaid in image (c), where the colors green and red indicate tracking success and failure respectively. In (d) a virtual augmentation is placed in the camera image.

image is known, and the geometry of the reference image, i. e. the four 3D coordinates of the poster corners, is given in the world coordinate system, all virtual 3D information can be modeled in the same coordinate system as the reference image for a correct overlay of the augmentation onto the camera image. Therefore the creation of an AR-application with reference images can be similar in complexity to the use of planar fiducials.

To track a planar poster we created a system which uses several methods at different stages of the tracking process. As for the most markerless tracking systems, the problem of estimating the camera pose is divided into an initialization phase and a tracking phase. The main difference is that for the initialization no prior information about the camera pose is given as in the tracking step, where the camera parameters of the last frame are always known. With a given reference image the problem of initializing the camera tracking can be regarded as a wide baseline matching of feature points between the reference image and the current camera image. Of the many methods described in Section 6.3 we chose the keypoint classification technique with randomized trees to create correspondences between image pairs. By assuming that the reference image is located in the z-plane of the world coordinate system, the 3D coordinates of all detected feature



(a) (b) (c)

Figure 6.8.: Tracking a poster with large scale and aspect changes. In (a) the current camera image can be seen. The tracked template patches are overlaid in (b) and in (c) an augmentation demonstrates that the camera pose is estimated correctly.

points are also known. With these 2D/3D-correspondences the extrinsic parameters of the camera can be estimated.

After the successful initialization of the camera pose feature points are projected into the current image with the camera pose obtained from the initialization step and then tracked from frame to frame with a template-based approach. Patches are extracted from the reference image at several levels of an image pyramid and aligned with the current video image. To increase the robustness and convergence range, the feature points are first tracked from frame to frame with a simple optical flow translation estimation. This 2D position is then used to initialize the iterative template alignment of every feature patch. The affine motion model with illumination compensation is used for the template tracking. After every tracking step it is tested with a sum of squares distance check, if the tracking of a feature has been successful. Features where the tracking failed are not taken for the camera pose estimation. Lost features are predicted with the successfully estimated camera pose of the current frame. Thereby it is possible that lost features can be tracked again, if they are visible in the camera image. An outline of our poster tracking algorithm can be found in Algorithm 3.

In Figure 6.7 a frame of a simple poster tracker application is illustrated. The poster is partially occluded and it can be seen that occluded features are not tracked successfully, but remain at the locations where they are expected, because they are predicted with the camera pose, which is estimated with the correctly tracked features.

Figure 6.8 shows another tracking example, where a postcard is used as a reference image. Two frames of a sequence show that the multiscale template-based tracking approach is

**Algorithm 3** Tracking with a reference image

**Initialization:**

1: Extract keypoints in reference image
2: Generate randomized trees for keypoint classification
3: Extract patches from reference image in several resolution levels

**Tracking:**

1: **if** camera pose is not valid **then**
2: Extract keypoints in current camera image
3: Classify keypoints with randomized trees
4: Estimate camera pose
5: **end if**
6: Reproject invalid template features with current camera pose
7: **if** last frame was tracked successfully **then**
8: Predict 2D feature position with optical flow estimation
9: **end if**
10: Track template patch with affine model and illumination compensation
11: Estimate pose with all successfully tracked features

capable of handling different scales and that the reference image can be tracked under large aspect changes. If the reference image is observed with a very flat viewing direction, an increasing number of features, where the tracking fails, can be observed. However, the tracking range of the template alignment is much larger than the possible camera positions, where the tracking system initializes successfully.

### 6.7.2. Texture-Based Tracking with Polygonal Models

For augmented reality application with industrial scenarios, CAD models of the objects, which shall be tracked, are often created in the manufacturing process. These CAD model can be a useful support for the tracking of these objects or the observing camera. The edge-based tracking method described in Section 5 is one possibility to track an object with a given polygonal 3D model. If a model consists of long distinct edges, very good tracking results are achieved with this method. However, using only the edge information can result in a poor tracking quality if no distinct geometry edges exist in every visible area of a 3D model.

Texture-based tracking methods are another possibility to estimate the object's movement and can be very beneficial if the visual appearance on an object consists of significant image structure. If tracked 2D point features shall be used to estimate the extrinsic parameters of a camera, it is necessary to know the 3D coordinate of every point. Only with correspondences between 2D image points and 3D coordinates of these points, the camera pose can be computed by minimizing the projection error.

Since we regard only 3D models without visual properties as material colors or textures here, the visual information of a feature point cannot be obtained from the model, and

must be taken out of the camera image. However, with the given 3D model geometry it is possible to acquire the 3D information of a tracked 2D feature point in the image.

### Acquisition of 3D Coordinates

One possibility to get the 3D coordinate of an image point is to use rendering techniques as presented by Vacchetti et al. [113]. Their idea is to assign a unique color to each triangle of the model. After rendering the model with the same resolution as the camera image, the color at the position of every tracked 2D feature point is used to index the triangle of the model, which corresponds to the 2D image point. The triangle spans a plane and the intersection point of the view ray from the camera position through the feature position in the image can be computed, which results in the 3D coordinate of the feature point. A similar approach for the acquisition of 3D coordinates is used in [85]. The positive fact of this method is that the 3D position can be calculated very precisely, if the model is very detailed.

A much more direct way to compute the 3D coordinate, where no extra color coding is necessary, is to use the depth buffer of a rendered image. Reitmayr et al. [89] use this depth buffer method to acquire the 3D coordinates of line control points, which are extracted of a rendered image. After rendering a model with the current camera pose, the z-buffer is read back from the graphics hardware. At every 2D feature position the 3D coordinate in the camera coordinate system can be computed with the given value of the z-buffer. If $z_B$ is the value of the OpenGL z-buffer in the range between 0 and 1, the z-value $z_C$ in the camera coordinate system can be computed by

$$z_C = \frac{1}{(2z_B - 1)\frac{n-f}{2fn} + \frac{f+n}{2fn}}. \tag{6.30}$$

The values $f$ and $n$ are the distances to the near and the far plane respectively, which were used for rendering the model.

With the intrinsic camera parameter matrix $K$ and the extrinsic camera parameters $R$ and $\boldsymbol{t}$, the homogeneous image point $\tilde{\boldsymbol{m}} = (x, y, 1)^T$ can be transformed into the 3D position $\boldsymbol{M}$

$$\boldsymbol{M} = R^{-1}(z_C K^{-1}\tilde{\boldsymbol{m}} - \boldsymbol{t}). \tag{6.31}$$

The 3D point $\boldsymbol{M}$ is here given in the world coordinate system. For a best possible depth resolution, the near and far planes are set in such a way that the bounding volume of the object lies exactly between the clipping planes.

The z-buffer method has the advantage that for a large amount of feature points the 3D coordinates can be determined with very little computational costs. A drawback of the rendering-based approaches is that the frame buffer or the depth buffer has to be read back. Since most graphic cards have a very poor performance in transferring data back to the main memory, this step is the main bottle neck of the approach.

Another method for the acquisition of a 3D coordinate of a given 2D feature point is to use ray casting techniques. An intersection test of the camera viewing ray with the

model geometry results in the 3D position of an observed feature point. This intersection test can be very fast, especially if a model consists of a hierarchy of many bounding volumes. If only the 3D position of very few feature points is demanded, using such geometric intersection tests can be more efficient than rendering the whole model and reading back a whole rendering buffer. However, with many feature points, rendering techniques can help to reduce the computation time. Depending on the complexity of the model and the number of feature points, there is always a break-even point, where either the ray intersection or the rendering method is more efficient. With geometric models of industrial scenarios we experienced that the acquisition of 3D coordinates with the depth buffer has the better performance.

If the camera pose is known, features can be extracted out of an image and corresponding 3D coordinates can be obtained with the 3D model geometry. Vacchetti et al. [113] use a set of calibrated reference images together with the model geometry to acquire 2D/3D correspondences between image points and model points. With calibrated reference images the initialization of the camera tracking can be performed by matching feature points between the reference image and the current camera image. Since the 3D coordinate of a feature in the reference image is known, the matched feature in the current frame can be associated with the same 3D coordinate, and with the resulting 2D/3D correspondences the camera pose of the current frame can be estimated.

However, the calibration of reference images is an inconvenient pre-processing step, which is not applicable for many high level AR application developers. To avoid the use of reference images, we initialize our tracking system with the edge-based techniques as described in Section 5, where a line model is generated out of a polygonal model and then aligned with the image gradient. With a correctly initialized camera pose 2D features are extracted out of the current camera images and the 3D coordinate is obtained with the given polygonal model.

New features are extracted in areas of the current image, where features do not exist yet, if the number of successfully tracked features falls below a lower bound. We experienced that the minimum number of 30 features is a good compromise between robustness and real-time capability. The whole tracking method is outlined in Algorithm 4.

The tracking method is tested with a small industrial object. The polygonal model of the object which is used for the tracking setup, can be seen in Figure 6.9(a). The frame, where the tracking is initialized, is shown together with the overlaid line model in Figure 6.9(b). In this particular example a manually created line model is used to initialize the tracking. In Figure 6.10 the KLT features, which have obtained a valid 3D coordinate from the model geometry, can be seen. Features which have not been tracked successfully are colored red. On silhouette edges of the object and on the reflecting plexiglass surfaces for some of the features the tracking fails. The virtual augmentation which is rendered with the estimated camera pose is overlaid on the images, which is shown in Figure 6.11.

An industrial scenario, where augmented reality could be a very beneficial support for maintenance, is the engine hood of a car. We also tested our tracking method for such a scenario with the tracking method described in Algorithm 4. The polygonal model of the regarded engine, the extracted line model and the initialization frame are shown in Figure 6.12. Again the line model is just used to estimate the very first camera pose and

(a) (b)

Figure 6.9.: In (a) the polygonal model of an industrial maintenance scenario is shown. Only the geometry of the model is used for the determination of the feature's 3D position. In (b) the frame of the sequence is shown, where tracking is initialized with a line model. For this scenario the line model is created manually.

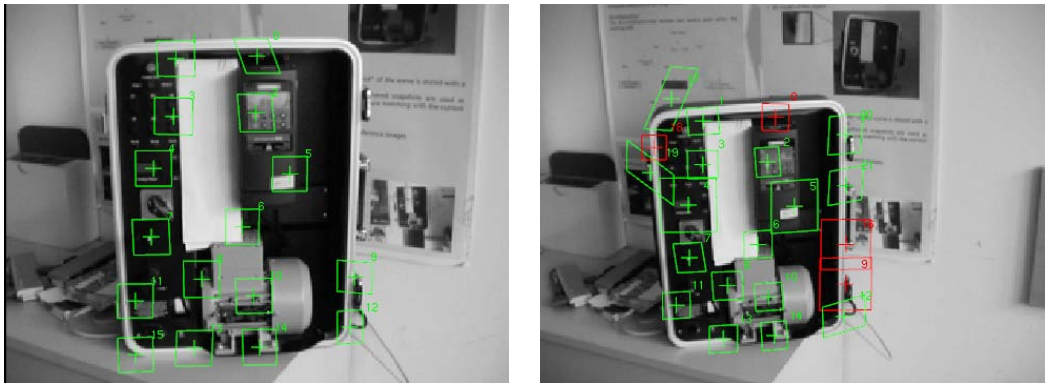


Figure 6.10.: All KLT-features which are located on the object are used for the camera pose estimation. Features where the tracking has failed are colored red.

Figure 6.11.: A virtual augmentation is rendered with the estimated camera pose onto the camera image. If enough KLT-features can be tracked, the virtual arrows are always overlaid correctly in the scene.

**Algorithm 4** Point-based tracking with a polygonal model

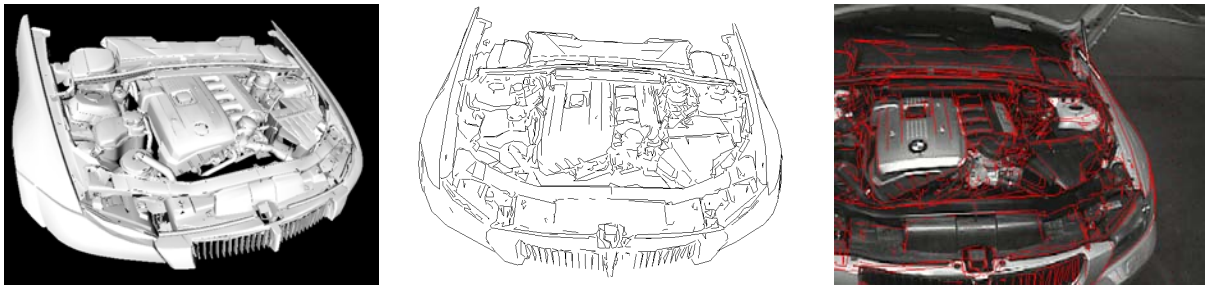
```
1: build image pyramid of current image
2: if current pose is not valid then
3:    try to initialize tracking with line based method
4: else
5:    for all features which are located inside the previous image do
6:       estimate feature translation from previous frame to current frame
7:       estimate full affine transformation with illumination compensation from reference
         patch to current frame
8:    end for
9:    compute camera pose with all successfully tracked features
10: end if
11: if estimated camera pose is valid then
12:    project all lost features into the image with the current camera pose
13:    if number of successfully tracked features is smaller than a lower bound then
14:       detect new feature points in areas of the image, where no feature exists yet
15:       for all newly detected feature points do
16:          extract patches at feature position at all levels of the current image pyramid
17:       end for
18:       acquire 3D coordinate of the feature with the given polygonal model
19:    end if
20: end if
```

the camera tracking is continued by tracking KLT feature points. The camera is moved steadily around the engine, and the scene is observed under a variety of different viewing positions and different scales. Throughout the whole sequence the camera pose can be estimated correctly. If the number of successfully tracked features is limited, slight jitter can be observed. The results of the KLT feature tracking step is visualized in Figure 6.13. Some frames of the regarded sequence with additional virtual information are shown in Figure 6.14. As augmentation three simple arrows pointing to some area of interest are overlaid.

### Using the Line Model for Drift Prevention

Our point feature tracking algorithm itself does not drift, since the alignment of the reference template is always the final step of the feature tracking methods. However, the acquisition of the 3D coordinates is not free of errors, and these errors have an influence on the further estimation of the camera pose. If new features are extracted and their 3D coordinates are obtained from the model geometry, the errors of the feature positions and 3D coordinates are accumulated. Therefore the tracking method of Algorithm 4 is not drift free.

Since the 3D geometry of the extracted line model, which is used for the initialization, is static, the line tracking algorithm does not produce any drift. This benefit can be used to eliminate drift, if the line model initialization step is inserted into the feature point tracking algorithm. The most reasonable time when the line model tracking should be



(a) (b) (c)

Figure 6.12.: Semi-automatic initialization with a polygonal model. In (a) the rendered polygonal model is shown, in (b) the extracted line model and in (c) the frame in which the tracking is initialized can be seen.

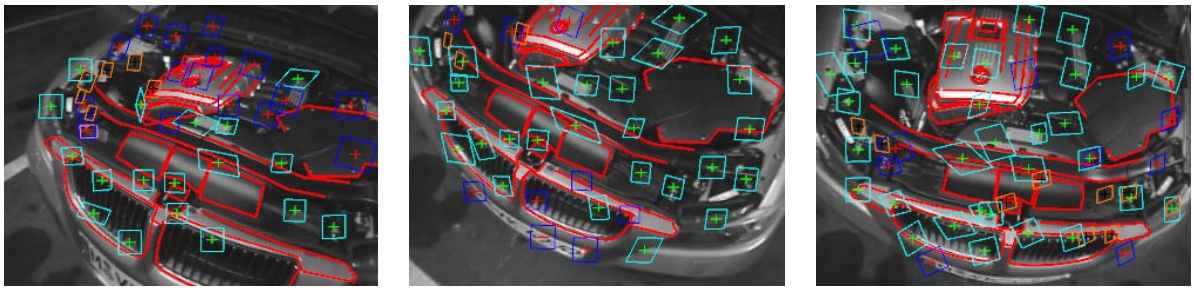


Figure 6.13.: Visualization of the KLT feature tracking. The line model is only used for initializing the first camera pose.

called, is before the extraction of new KLT features, because then it is guaranteed that the virtual model is correctly aligned with the model in the image. The error of the feature points is not accumulated, because the camera pose which is estimated with other feature points is not used for the acquisition of 3D coordinates. However, a drawback of using the line tracking method to prevent drift is that the whole tracking system does not run at constant speed, because every time when new features are extracted, a slowdown of the frame rate can be observed.

### 6.7.3. Reinitialization with SIFT Features

If the scene is completely occluded or the user looks away from the object of interest for a short time, the KLT features cannot be tracked any more and the camera pose estimation fails. If only the semi-automatic initialization with the line model is used, the user is forced to re-initialize the tracking by moving the observing camera to the position, where the line model can be aligned successfully. A more sophisticated and user-friendly method is the acquisition of calibrated reference images during the tracking and the use of these images for a wide base line matching algorithm to re-initialize the camera tracking. In our tracking system we used the SIFT descriptor for this purpose. A similar approach is implemented in the distributed mobile AR system, which was presented in [56].

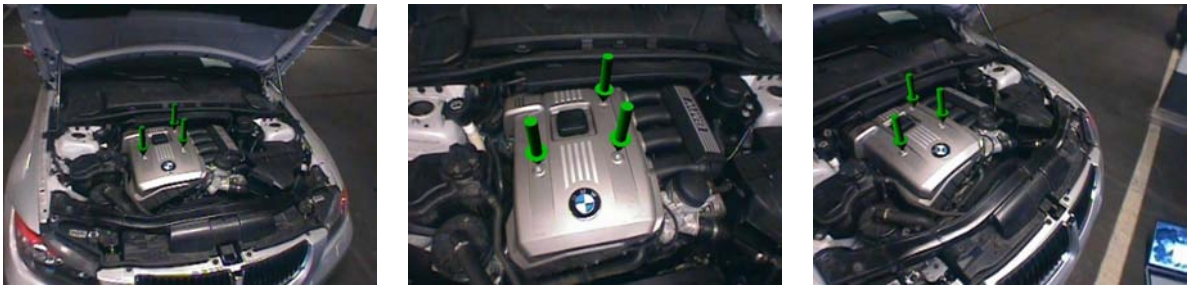


Figure 6.14.: Tracking of a motor engine. The arrows are rendered with the camera pose, which is obtained from the KLT-feature tracking.

For this re-initialization step the tracking system collects a set of calibrated reference images during the successful tracking. A valid camera pose is stored together with the extracted SIFT features of the regarded camera frame. Since the 3D coordinates can be obtained from the model geometry, a set of 2D/3D correspondences is created for every captured reference frame. If the tracking is interrupted, the re-initialization algorithm is called with the proximate image frame. SIFT features are extracted from the current camera image and matched to the 2D/2D correspondences to one of the reference views. With the matches new correspondences of 3D coordinates and 2D points in the camera image are created and used to calculate the camera pose. A new reference image and a corresponding camera pose are stored only if no reference image with a similar camera pose exists. By analyzing the histograms, the reference images are ordered according to similarity, and only those which resemble the current camera image are used in decreasing order of similarity for the re-initialization of the camera pose.

With this re-initialization module, the user of the AR system can turn the observing camera away from the scene, and as soon as known parts of the scenario appear in the camera image again, the tracking is reinitialized without requiring any user interaction.

Matching techniques like the classification of feature points with randomized trees can only be used for the initialization and not for the re-initialization, since the generation of the trees is a time-consuming processing step. Therefore such approaches are only applicable, if the generation of classification trees is moved in a preprocessing step.

## 6.8. Conclusion

Point feature tracking techniques can contribute to very beneficial solutions for the camera pose estimation problem. We implemented a tracking system, where optical flow based methods and template tracking approaches are used for the frame-to-frame tracking. We improved the template-based tracking by considering multi-resolutions of reference templates, which results in the to possibility to track a feature robustly under a variety of different scales. Furthermore, we developed an update method and a template mask generations method to increase the stability of the template alignment.

These feature tracking approaches were used to create a camera tracking system with a

given model geometry. Simple models like planar posters, but also complex industrial scenarios can be tracked with our method.

Wide baseline matching algorithms are used for the initialization and re-initialization of our tracking system. For some scenarios we used a line model for the very first estimation of the camera pose, but continued to track with point features. This can be helpful, if the scene consists of textured structures, in which point features can be tracked reliably.

The use of point features instead of only line features is an essential improvement to lead to a more stable behavior of large displacements. Since template-based features can be aligned with greater robustness and precision, the overall tracking is more stable and less jitter can be observed. However, if a scene hardly consists of distinct point features, like a collection of pipes, point-based methods might not produce as good results as line-based methods.

Future work in the area of point-based tracking will be a more efficient implementation of the tracking algorithms directly on the graphics hardware. Since the architecture of GPUs provides excellent parallel computation units, such approaches could result in higher frame rates with large image resolutions.

# 7. Tracking in Unknown Scenes

## 7.1. Introduction

If augmented reality applications shall be created with scenarios, where not the complete model geometry is known, the tracking algorithms described in the previous chapter are not applicable. Since with those methods 3D coordinates can only be gathered at image positions which correspond to a given model, the camera tracking can only work if parts of that model are visible. However, there is an urgent demand for a tracking system that can estimate the camera pose in a scene, where only limited or even no reference geometry is given. With such a system the camera tracking could continue, even if the reference object is not visible in the camera image any more.

Since the 3D position of a feature cannot be taken from a given model geometry, the only choice is the reconstruction of 3D coordinates from corresponding image points in multiple views. With two or more views a 3D point can be reconstructed by triangulation, if the camera parameters, i.e. projection matrices, are known. Structure from motion (SFM) techniques allow the simultaneous determination of 3D points and projection matrices in each view. Such methods are used widely in photogrammetric application like the reconstruction of virtual reality models [125] or the determination of the camera motions, also denoted as match moving. Commercial products for the estimation of the camera path of a video stream exist[1], and are mainly used in cinematography for post-production and special effects. For those purposes the sequences can be processed off-line and the amount of computational workload is not critical. Such methods are not applicable for augmented reality applications since for the seamless integration of virtual objects into video streams, the camera pose has to be estimated in real-time.

But nevertheless structure from motion techniques can be used for the online estimation of the camera parameters. This chapter handles the real-time reconstruction of scene geometry with the intent to use the reconstructed geometry for further continuous camera tracking. First the reconstruction of feature points is discussed, furthermore an approach for the reconstruction of surface normals is presented. The reconstructed scene geometry is then used for the prediction of lost feature points.

## 7.2. Online Reconstruction of Point Features

By analyzing the relation between multiple views and the correspondences between image feature points, the scene structure and the camera motion between the different views can

[1] see http://www.realviz.com

be retrieved. The typical structure from motion approach is first to estimate the camera motion, then reconstruct an initial estimate of the feature positions and finally refine both camera parameters and feature positions. If no previous knowledge of the scene exists, the camera motion between two image frames can be calculated with epipolar constraints, i.e. the estimation of the essetial matrix and the decomposition into camera rotation and camera translation. If some correspondences between image points and 3D coordinates exists in both images, the camera pose can also be estimated by minimizing the projection error.

If the camera parameters are known for all regarded views, the 3D position $\boldsymbol{M}$ of a feature point can be reconstructed by triangulation [40]. The classical approach is the solution of the following least squares problem

$$\boldsymbol{M} = \arg\min_{\boldsymbol{M}} \sum_i \|\boldsymbol{m}_i - f_i(\boldsymbol{M})\|^2 , \tag{7.1}$$

where $f_i$ is the projection function, which projects the 3D point $\boldsymbol{M}$ into the image of the $i^{\text{th}}$ view, and $\boldsymbol{m}_i$ is the observed feature point in the image frame $i$. A linear solution can be calculated by singular value decomposition and optionally refined with a nonlinear method like Levenberg-Marquardt.

The whole reconstruction process is usually carried out in a sequential way, where the camera parameters are computed between consecutive views. The estimates of the projection matrices of all views and the reconstructed 3D points of all features can be refined with a bundle adjustment. An iterative non-linear optimization method is used to minimize a weighted sum of all projection errors with respect to the feature positions and camera parameters.

Another very promising approach is the simultaneous localization and mapping (SLAM), where a map is built incrementally with the simultaneous estimation of the current camera pose. Statistical methods like the extended Kalman filter or particle filters are used to keep track of the feature positions and the camera state. A complete SLAM system for tracking a monocular camera for augmented reality applications is presented by Davidson [23]. A benefit of such filtering techniques is that other devices like inertial sensors can be incorporated into the tracking process. However, a drawback is that the method gets computationally very complex with an increasing number of features.

In [12] we present a system, where the consequent error propagation of the SLAM approach is combined with robust and efficient traditional SFM and pose computation methods. The difference to a full SLAM system, which maintains a huge covariance matrix with the correlation of all features and the camera pose, is the decoupling of scene reconstruction and pose estimation. This strategy makes the system more flexible with the insertion and deletion of features and more efficient with a larger number of feature points. The triangulation is implemented as a pseudo-RANSAC procedure to increase robustness especially against wrong feature localizations. If a 3D feature is visible in the current image, its 3D position and covariance is updated with the new measurement using the extended Kalman filter measurement update, which reduces uncertainty over time.

## 7.3. Reconstruction of Surface Normals

For a robust tracking system it is desired that lost features can be used for the tracking once again, after they have been occluded or moved out of the image frame for a certain time. Therefore unsuccessfully tracked features need to be predicted so that the local search can be carried out successfully in the proximate camera image. If a planar template is used to track a single point, such as the KLT feature tracker or a homogeneous image alignment, not only the feature position but also the warp function needs to be predicted. A correct prediction of the warp function is only possible if the true surface normal vector of the template is known. A rough approximation can be made by assuming that the surface normal is equal to the viewing direction of the camera in that image frame, when the feature is observed for the first time. With a reconstruction of the true surface normal a more precise prediction of the warp is possible.

A method of reconstruction the surface orientation of a planar region is presented by Molton et al. [77]. They estimate the surface normal by a gradient-based image alignment technique where the parameters of the normal vector are used as the degrees of freedom during the minimization of intensity differences. Our method is similar to the one presented by Favaro et al. [27], where an extended Kalman filter is used to iteratively refine the estimate of the surface normal orientation.

### 7.3.1. Relation between Camera Motion and Image Transformation

If we consider two camera frames, where $(R_0, \boldsymbol{t}_0)$ and $(R_1, \boldsymbol{t}_1)$ are the extrinsic camera parameters of the first and the second camera respectively, the world coordinates of a 3D point $\boldsymbol{M}_w$ are transformed into the two camera coordinate systems by

$$\boldsymbol{M}_{c0} = R_0 \boldsymbol{M}_w + \boldsymbol{t}_0 \tag{7.2}$$
$$\boldsymbol{M}_{c1} = R_1 \boldsymbol{M}_w + \boldsymbol{t}_1. \tag{7.3}$$

Substituting $M_w$ in the second equation leads to the transformation of a point from the first camera coordinate system to the second camera coordinate system, which can be computed by

$$\begin{aligned} \boldsymbol{M}_{c1} &= R_1 R_0^T (\boldsymbol{M}_{c0} - \boldsymbol{t}_0) + \boldsymbol{t}_1 \\ &= (R_1 R_0^T) \boldsymbol{M}_{c0} + (\boldsymbol{t}_1 - R_1 R_0^T \boldsymbol{t}_0). \end{aligned} \tag{7.4}$$

For the rotation difference $\Delta R$ and the translation difference $\Delta \boldsymbol{t}$ we get

$$\Delta R = R_1 R_0^T \tag{7.5}$$
$$\Delta \boldsymbol{t} = \boldsymbol{t}_1 - R_1 R_0^T \boldsymbol{t}_0 \tag{7.6}$$

and the equation (7.4) can be rewritten as

$$\boldsymbol{M}_{c1} = \Delta R \boldsymbol{M}_{c0} + \Delta \boldsymbol{t}. \tag{7.7}$$

If $\boldsymbol{n}_w$ is a unit normal vector of a plane $P$ in the world coordinate system, this normal vector can be transformed into the coordinate system of the first camera by

$$\boldsymbol{n}_{c0} = R_0 \boldsymbol{n}_w. \tag{7.8}$$

Now, if $\boldsymbol{n}_{c0}$ is the plane normal with respect to the first camera pose and $d$ is the distance from the plane to the optical center of the first camera, then for all points $\boldsymbol{M}_{c0}$ on the plane $P$ the equation

$$\boldsymbol{n}_{c0}^T \boldsymbol{M}_{c0} = d \tag{7.9}$$

must hold. The distance $d$ can be calculated by $d = \boldsymbol{n}_{c0}^T \boldsymbol{X}_p$ with any 3D point $\boldsymbol{X}_p$ located on the plane $P$.

Substituting equations (7.9) into equation (7.7) leads to

$$\begin{aligned} \boldsymbol{M}_{c1} &= \Delta R \boldsymbol{M}_{c0} + \Delta \boldsymbol{t} \frac{\boldsymbol{n_{c0}}^T \boldsymbol{M}_{c0}}{d} \\ &= (\Delta R + \frac{\Delta \boldsymbol{t} \boldsymbol{n_{c0}}^T}{d}) \boldsymbol{M}_{c0}. \end{aligned} \tag{7.10}$$

The linear transformation of a 3D point on the plane $P$ from the first camera coordinate system into the second camera coordinate system can also be described by

$$\boldsymbol{M}_{c1} = H \boldsymbol{M}_{c0}, \tag{7.11}$$

where

$$H = \Delta R + \frac{\Delta \boldsymbol{t} \boldsymbol{n_{c0}}^T}{d} \tag{7.12}$$

denotes a homography mapping form $\boldsymbol{M}_{c0} \in \mathbb{R}^3$ to $\boldsymbol{M}_{c1} \in \mathbb{R}^3$.

With the intrinsic camera matrix $K$, the point $\boldsymbol{M}_c$ in the coordinate system of a camera can be transformed into image coordinates by

$$s \tilde{\boldsymbol{m}} = K \boldsymbol{M}_c, \tag{7.13}$$

where $s$ is a factor representing the scale ambiguity and $\tilde{\boldsymbol{m}}$ the homogeneous point in the image.

If $\tilde{\boldsymbol{m}}_0 = \frac{1}{s_0} K \boldsymbol{M}_{c0}$ is the homogeneous point in the first camera image and $\tilde{\boldsymbol{m}}_1 = \frac{1}{s_1} K \boldsymbol{M}_{c1}$ is the homogeneous point in the second camera image, the relation between $\tilde{\boldsymbol{m}}_0$ and $\tilde{\boldsymbol{m}}_1$ can be denoted as

$$\tilde{\boldsymbol{m}}_1 \sim K H K^{-1} \tilde{\boldsymbol{m}}_0, \tag{7.14}$$

where $\sim$ indicates equality up to a scale factor.

The homography $H_I$, which maps a homogeneous image point on the plane $P$ from the first to the second camera image can be written as

$$H_I = \lambda K H K^{-1}. \tag{7.15}$$

This homography $H_I$ is the transformation which is, for example, estimated with an iterative image alignment method as the one presented in Section 6.5.

In order to compute $H$ from the image homography $H_I$, the scale factor $\lambda$ has to be determined. The image homography $H_I$ can be transformed into the camera coordinate system by

$$H_L = K^{-1} H_I K. \tag{7.16}$$

If the relation between $H_L$ and the motion parameters of the camera ($R$,$\boldsymbol{t}$) and the structure parameters ($\boldsymbol{n}$,$d$) is written as

$$H_L = \lambda H = \lambda(\Delta R + \frac{\Delta \boldsymbol{t}\boldsymbol{n}^T}{d}), \tag{7.17}$$

then the scale factor $\lambda$ can be computed by

$$|\lambda| = \sigma_2(H_L), \tag{7.18}$$

where $\sigma_2(H_L)$ is the second largest singular value of $H_L$. The proof can be found in [67], p.135.

By assuming that the normal vector $\tilde{\boldsymbol{n}}_{c0} = d\boldsymbol{n}_{c0}$ is non-unit, the value $d$ of equation (7.17) can be neglected, and the normal vector must satisfy the following equation

$$(\frac{1}{\lambda} H_L - \Delta R) = \Delta \boldsymbol{t} \tilde{\boldsymbol{n}}_{c0}^T. \tag{7.19}$$

Transposing and multiplying this equation with $\Delta \boldsymbol{t}$ results in the least squares solution for $\tilde{\boldsymbol{n}}_{c0}$:

$$\tilde{\boldsymbol{n}}_{c0} = \frac{1}{\Delta \boldsymbol{t}^T \Delta \boldsymbol{t}} (\frac{1}{\lambda} H_L - \Delta R)^T \Delta \boldsymbol{t}. \tag{7.20}$$

Then the unit normal $\boldsymbol{n}_{c0}$ and the scale factor $d$ can be determined by normalizing $\tilde{\boldsymbol{n}}_{c0}$. Since this equation computes the normal vector $\boldsymbol{n}_{c0}$ in the coordinate system of the first camera, it must be transformed into the world coordinate system by

$$\boldsymbol{n}_w = R_0^{-1} \boldsymbol{n}_{c0}. \tag{7.21}$$

Now the vector $\boldsymbol{n}_w$ can be regarded as a measurement of the surface normal, which will be used for a robust estimation of the surface orientation.

## 7.4. Feature Prediction

If an image feature is occluded or cannot be tracked because of reflections, these feature points needs to be predicted so that the local search of the template-based tracking can be successful. With no proper prediction the starting point for the iterative image alignment

would be further away from the solution and the chances for convergence are getting lower with every frame, at which the tracking of this feature failed. Moreover, feature points which move out of the camera image and are not visible in the current frame any more should not be discarded. If the camera moves back and an already reconstructed feature point gets visible again, this points needs to be predicted so that the feature tracking step can be carried out correctly.

The most straightforward method for feature prediction is to use the camera pose of the last frame, assuming that the pose estimation was successful. A more sophisticated approach is to use a Kalman filter with a kinematic motion model. With such a filter the current camera pose can be extrapolated for a proximate camera frame, which results in a more precise prediction. If other input devices like an inertial sensor are integrated into the tracking system, they can also be used to provide a more accurate prediction of the camera pose.

### 7.4.1. Image Position Prediction

If the approximate camera pose is known, it can be used to predict all feature points, where the tracking step failed in the last frame. Predicting the position of a template is fairly simple. If $\boldsymbol{M}$ is the reconstructed 3D point, the image position of this feature can be estimated by projecting the point $\boldsymbol{M}$ into the image with the current camera parameters. With the intrinsic camera matrix $K$ the homogeneous image position $\tilde{\boldsymbol{m}}'$ can be computed with

$$\tilde{\boldsymbol{m}}' = K(R\boldsymbol{M} + \boldsymbol{t}), \tag{7.22}$$

where the camera rotation and translation used for the prediction is given by $R$ and $\boldsymbol{t}$.

### 7.4.2. Warp Prediction

If a warp shall be predicted in the image, the 3D geometry of the template needs to be available. A simple approximated method for predicting an affine warp is to use the unit vectors of the affine transformation. When a feature is detected for the first time, it is assumed that the initial captured patch is observed from an orthogonal direction of the patch plane, and the 2D unit vectors of the affine transformation are un-projected with the 3D position of the regarded feature point. This yields two vectors which approximately describe the orientation of the template in 3D space. If offline models are used, e.g. a reference image of a poster, these 3D unit vectors of the affine transformation can be determined exactly.

For a prediction of an affine warp with a given camera pose, the 3D unit vectors can also simply be projected into the image, and the affine warp can be derived from the projected unit vector.

For a more accurate tracking of templates in camera images with a wide field of view, the warp function can be modeled with a homography. A homography correctly models the transformation of a plane for a perspective camera projection. In equation (7.12) it

can be seen that the surface normal of a plane is related to the homography mapping of a template between two images. If a surface normal of a template is reconstructed or given from a 3D model, this normal vector $n_w$ can be used to predict the homography. Together with the 3D position $\boldsymbol{M}_w$ and the relative camera rotation $\Delta R$ and relative camera translation $\Delta \boldsymbol{t}$, the homography of an image patch can be calculated.

Since the equation

$$d = \boldsymbol{n}_{c0}^T \boldsymbol{M}_{c0} = \boldsymbol{n}_{c0}^T (R_0 \boldsymbol{X}_p + \boldsymbol{t}_0) \tag{7.23}$$

must be satisfied for any point $\boldsymbol{X}_p$ on the plane $P$, the predicted homography $\tilde{H}$ can be calculated by

$$\tilde{H} = \Delta R + \frac{\Delta \boldsymbol{t} \boldsymbol{n}_{c0}^T}{\boldsymbol{n}_{c0}^T (R_0 \boldsymbol{M}_w + \boldsymbol{t}_0)}. \tag{7.24}$$

Since we are interested in a prediction of a warp function in the image coordinate system, the homography $\tilde{H}$ has to be projected into image space. For this homography, which transforms a point in the image of the first camera to the second camera, we get

$$\tilde{H}_I \sim K \tilde{H} K^{-1}. \tag{7.25}$$

The homography $\tilde{H}_I$ describes the transformation from a template in the initial camera image to the current camera image.

If a feature is initialized at the 2D image position $\boldsymbol{p} = (p_x, p_y)^T$ the homography which represents the current warp of the template is initialized with

$$H_0 = \begin{pmatrix} 1 & 0 & p_x \\ 0 & 1 & p_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{7.26}$$

To obtain the homography in the image of the current camera pose, $\tilde{H}_I$ has to be transformed by the initial homography $H_0$. Finally the prediction of the homography in the current camera image can be computed by

$$H^* = \tilde{H}_I H_0. \tag{7.27}$$

The undefined scale factor of $H^*$ can be eliminated by normalization with the last element of $H^*$. With the warp parameters of $H^*$ the iterative template alignment with a projective warp function is initialized.

### 7.4.3. Prediction of the Illumination Parameters

Since the illumination can change significantly if the feature has not been observed for a long time, the illumination parameters also need to be predicted. This is very important for the convergence behavior, because if the illumination parameters are too far away from the solution, the whole iteration process is likely to diverge.

If $\mu_t$ and $\mu_0$ are the mean intensity values of the current and the initial patch, and if $\sigma_t$ and $\sigma_0$ are their standard deviations respectively, then the contrast $\lambda$ and brightness $\delta$ can be predicted by

$$\lambda = \frac{\sigma_t}{\sigma_0} \tag{7.28}$$

$$\delta = \mu_t - \frac{\sigma_t}{\sigma_0}\mu_0. \tag{7.29}$$

The predicted illumination parameters $\lambda$ and $\delta$ describe the illumination correction of a patch, which is extracted out of the current image, that $T(x) = \lambda I(g(x; \boldsymbol{p})) + \delta$ holds.

## 7.5. Experimental Evaluation

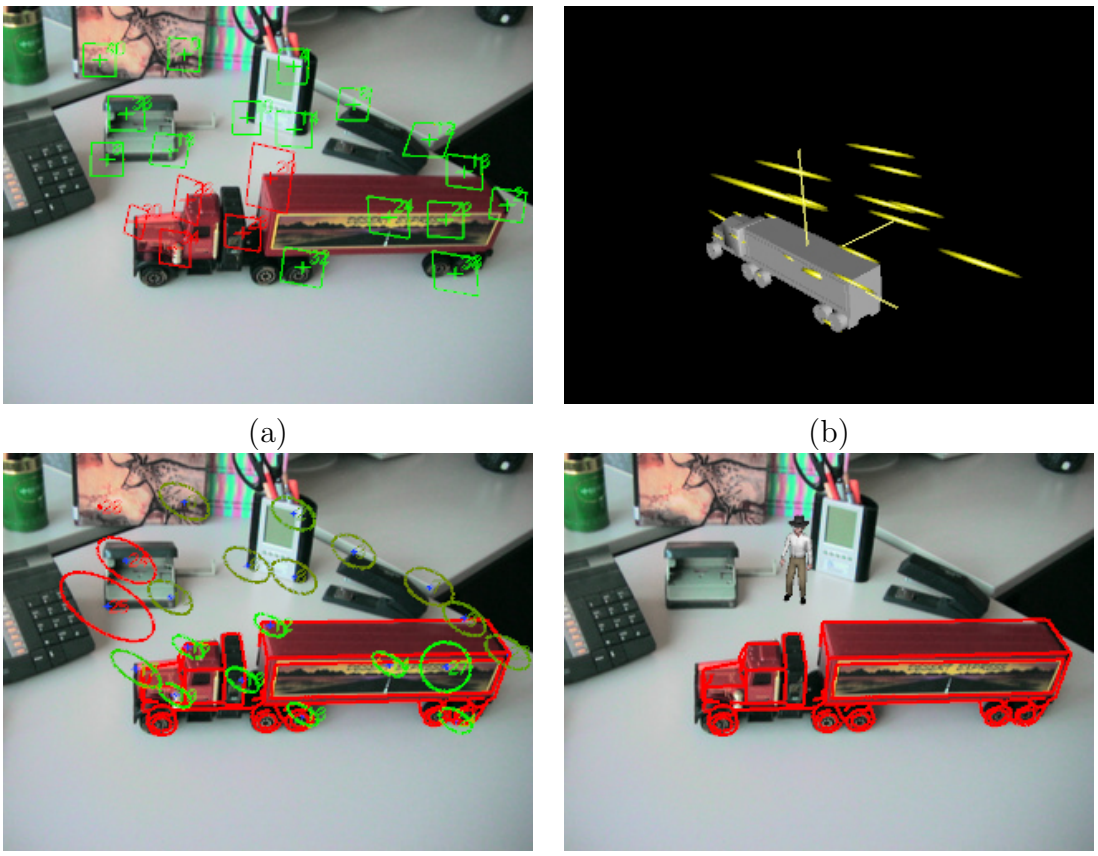
### 7.5.1. Tracking in Partially Known Scenes

We evaluated the tracking system with the integrated reconstruction of feature points on several image sequences. In the first sequence, as illustrated in Figure 7.1, the camera pose estimation is tested with a desktop scenario. A toy truck is used as a reference object and the tracking system is initialized with a generated line model of the toy truck. At the beginning the extrinsic camera parameters are set in such a way that the projection of the generated line model is in the center of the image. If the projected lines are close enough to the real object in the image, the line model is aligned and the first point features are initialized both on the model and on other areas of the image. During the point feature tracking phase the line model is just used for augmenting the scene. After the tracking is initialized, the camera pose is estimated by using only point features on the object, since these are the only points with a known 3D coordinate. Other features are detected, triangulated and refined during the further tracking. When the camera moves away from the truck, it is still possible to estimate the camera pose correctly. After the truck reappears in the image, features obtained in previous tracking steps are re-acquired, and no drift is visible in the augmentation.

For a clearly arranged visualization of the different steps of the tracking system we composed a result video with four image frames, which all point out different properties of the tracked features. In Figure 7.1 such an arrangement is illustrated.

All the four images represent the state of all the current features at the same time step. In image (a) the results of the 2D feature tracking step with KLT features is visualized. The green rectangles are successfully tracked features, for the red ones the tracking failed. Mostly features which are located on object borders differ too much from the initially extracted template and can thus not be tracked successfully.

In (b) the 3D model of the reference object is shown from a different point of view together with the 3D covariances of the reconstructed feature points, which are represented by ellipsoids. It can be observed that the features have an initially high uncertainty along the viewing direction of the camera. While moving the camera through the scene, the feature positions are refined with an extended Kalman filter. The uncertainty of the feature positions is thereby decreased, which is visualized by shrinking ellipsoids.



(a) (b)

(c) (d)

Figure 7.1.: Illustration of the feature tracking and reconstruction process. All images represent the states of the features at the same camera frame. In (a) the results of the 2D feature tracking step are pointed out. In (b) the reference object and the 3D covariances of the reconstructed feature points are shown. Figure (c) illustrates a projection of the 3D covariances in the image plane. Features with a high absolute value of the covariance are colored red, features with a small uncertainty are colored green. In (d) the original image is augmented with the line model of the reference object and a virtual character standing on the table.

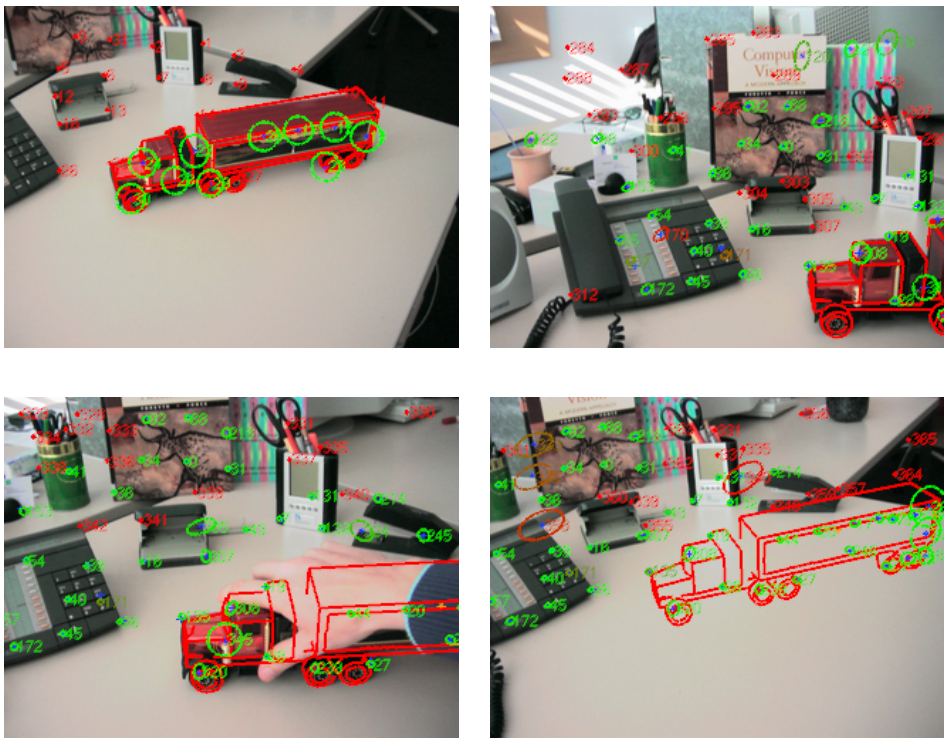


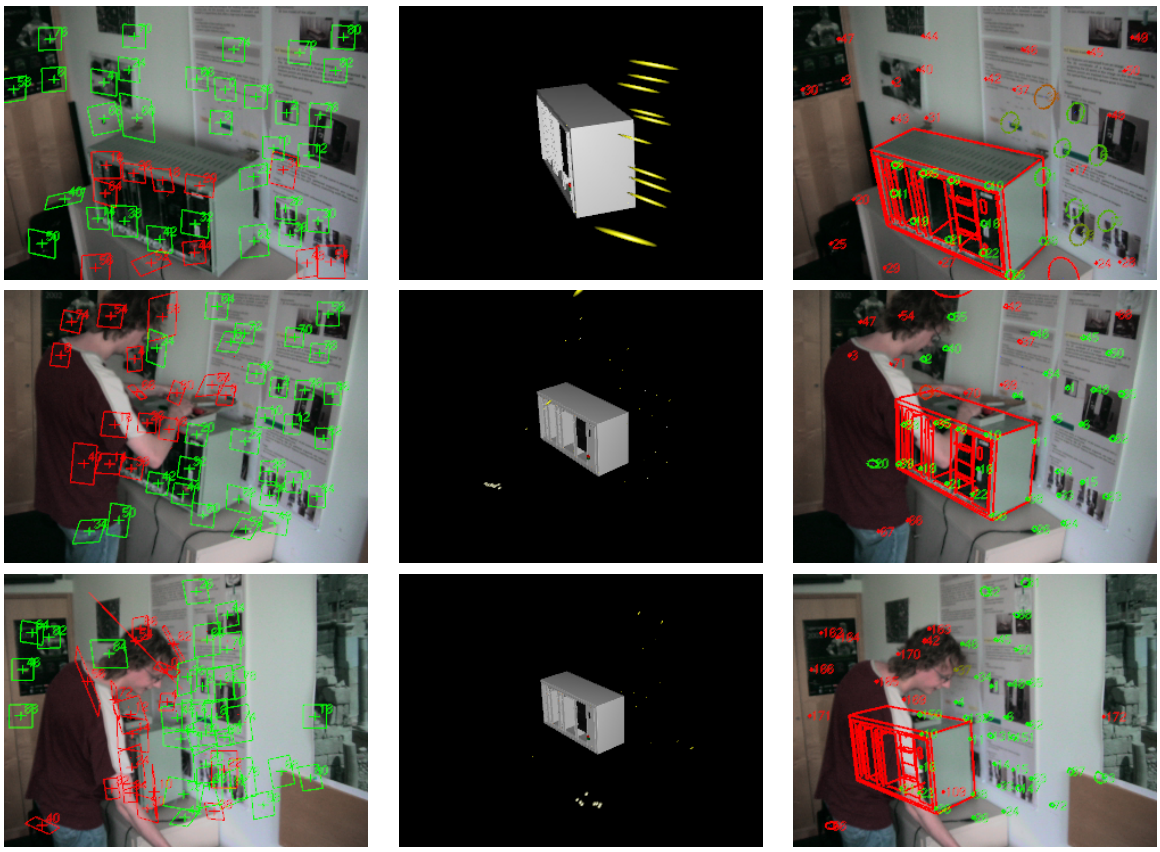
Figure 7.2.: Tracking results showing the ability of handling occlusion and even the total removal of the object from the scene.

Image (c) of Figure 7.1 illustrates the same uncertainty regions of the feature points as a projection in the image plane. The absolute value of the uncertainty is color coded in that way that the color value is shifted with increasing precision from red to green.

Finally in (d) the original image is overlaid with the line model with which the tracking was initialized and an additional virtual character standing on the table. The purpose of this frame is to evaluate, if the camera pose is estimated properly and that virtual objects are always placed correctly in the scene.

Figure 7.2 shows the tracking results of another sequence of the same scenario with an image size of 320x240 pixels. This time the initialization object is occluded and removed from the scene. Since enough other features have been triangulated and refined successfully, it is still possible to keep tracking. The line model, which is used for augmentation, sticks at the same position in the real world.

In another sequence an industrial control unit is used as a reference object. In Figure 7.3 some frames of this sequence can be seen. Again the tracking is initialized with a line-model, which is generated out of a given VRML-model. After the initialization feature points are extracted in the whole image, but only those points which are located on the known geometry can be used instantly for the camera tracking. The other features are triangulated and refined when the camera is moved through the scene. Again it can be observed that due to the refinement process the uncertainties of the reconstructed feature points shrink during the tracking. When a person moves into the scene, some feature are occluded and the 2D tracking step for these features fails. In the first column of Figure 7.3



(a) (b) (c)

Figure 7.3.: Tracking results demonstrating the robustness against occlusion. In (a) the KLT features are shown, in (b) the covariances of the reconstructed features, and in (c) the feature with their 2D unceratinties can be seen.

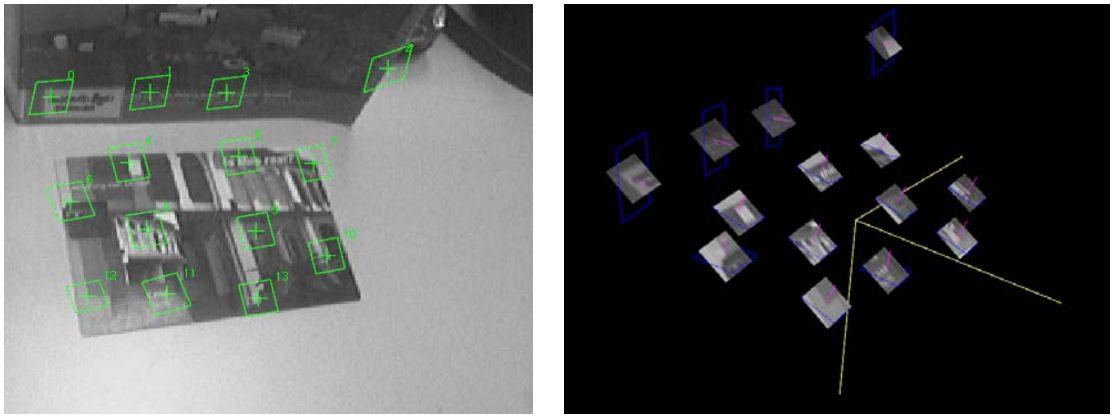
all those occluded features are colored red. Since enough valid 2D features are available in every frame, the pose can be estimated successfully despite the occlusion.

### 7.5.2. Runtime Analysis

We analyzed the processing time needed for tracking and reconstruction on a Pentium IV with 3 GHz. The results are shown in Table 7.1. The time for processing one frame strongly depends on the number of features in the current field of view. In this sequence on average 28.5 features were used for the tracking in every frame and 38.81 milliseconds

| | 2D registration + pose estimation | reconstruction |
|---|---|---|
| Avg. | 28.84ms | 9.97ms |
| Min. | 10.55ms | 4.15ms |
| Max. | 67.63ms | 32.86ms |

Table 7.1.: Average, minimum and maximum time in milliseconds, which is needed for processing one frame of the truck sequence.



(a) (b)

Figure 7.4.: In (a) a frame of the image sequence is shown together with the tracked template patches. In (b) the reconstructed 3D patched and their surface normal direction are drawn. The blue rectangle illustrates the corresponding plane of the feature.

were measured for the total time needed for 2D feature registration, pose estimation and reconstruction. The feature map contained 38.4 features on average.

For the acquisition of the images and the rendering of some virtual objects some additional processing time is needed. Altogether the system can handle frame rates up to 20Hz, when a reasonable number of features is used for tracking.

### 7.5.3. Surface Normal Reconstruction

For the evaluation of the reconstruction of surface normals a sequence of a desktop scene is used. The camera tracking is initialized with a reference image and a randomized trees keypoint classification. The tracking is tested with both updating the template and keeping the template as it is when it was captured at its first appearance. With both methods the camera can be traced throughout the whole scene.

To analyze the quality of the surface normal reconstruction the 3D patches together with their normals are rendered. In figure 7.4 a frame of the image sequence and the reconstructed planar patches together with their surface normal directions are shown. At the beginning when a feature is observed first, the normal vector points towards the camera position, but when the camera is moved around the scene, the estimate of the normal vectors converges towards the true orientation of the surface normal direction of a feature. At some points where only a poor alignment of a template patch is possible, i.e. on edges or object borders, the reconstructed normal cannot be estimated correctly.

# 8. Feature Management

## 8.1. Introduction

Tracking texture-based point features is a widely used technique for the estimation of the camera pose. If a limited set of planar template patches is tracked successfully in an image sequence, the extrinsic parameters can be computed robustly in real-time. These approaches are very promising if the feature points are located on well textured planar regions. However, in industrial scenarios objects often consist of reflecting materials and poorly textured surfaces, where feature tracking approaches like the KLT-tracker have a constricted success rate. Because of spotlights or occluding objects the area of camera positions where a feature point has the same visual appearance can be very limited.

Increasing the number of features can help to ensure a robust camera pose estimation, but as the 2D feature tracking step makes up a large amount of the computation time, the overall tracking performance gets very poor. Thus the number of features has to be limited, because tracking too many features has a negative effect on the real-time capability of a tracking approach. A choice for a reasonable number of features can be therefore regarded as a tradeoff between real-time capability and robustness.

Using only a subset of those features which are visible from a given viewpoint can avoid this problem. If a feature map consists of many features, not all feature have the same prospect of tracking success for a given camera position. For example, partially occluded features or features which are observed under a strong aspect change have a lower chance to be tracked successfully than other features. The difficulty of a sophisticated choice of a subset of feature points is to select those features which are most likely to be tracked from a given camera viewpoint and therefore are the best candidates for a robust camera pose estimation.

Najafi et al. [80] present a statistical analysis of the appearance and shape of features from possible viewpoints. In an offline training phase they coarsely sample the viewing space at discrete camera positions and create cluster groups of viewpoints for every model feature according to similar feature descriptors. Thereby a map is created which gives information about the detection repeatability, accuracy and visibility from different viewpoints for every feature. During the online phase this information is used for a selection of good features.

We present a method for a feature management which does not rely on any preprocessing, but performs an online estimation of the tracking probability of every feature. The ability to track a feature is observed during the runtime and a distribution of camera positions of tracking successes and tracking failures is created. These distributions are represented by a mixture model with a constant number of Gaussians. A merge operation is used to

keep the number of Gaussians fixed. The resulting tracking probability, which not only models the visibility but also the robustness of a feature, is then used to decide which features are most suitable to be tracked at a given camera position. The robust camera pose estimation is solved by using the Levenberg-Marquardt minimization and RANSAC outlier rejection. An evaluation demonstrates the quality of the probability distribution and the benefit of the computation time.

## 8.2. Feature Tracking and Map Management

For a reasonable feature map management, a single feature must be able to be tracked as long as possible. Therefore the 2D feature tracking must be invariant to deformations, illumination and scale. We used the template-based approach, which is described in Section 6.5 with an affine warp function and an additional brightness correction. To improve the convergence behavior of the template alignment the tracking is performed in two steps as presented in Section 6.5.2. Pure translation from frame to frame is estimated first on several levels of the image pyramid, and then the template patch is iteratively aligned at the resulting image position of the first stage.

With such a tracking system, the tracking of feature points is invariant under deformations because of the affine warping model and invariant against illumination changes. An additional scale invariance is realized by maintaining a set of templates of several resolution levels. This method extracts a template patch in different resolution levels of the image pyramid and always selects that patch which has the most similar resolution to the predicted affine transformed patch. If the desired resolution of the patch does not exist, it is extracted out of the current image after a successful tracking step.

A feature is regarded as tracked successfully if the iterations of the alignment converge and the discrepancy of the template and the extracted image patch at the current feature position is smaller than a given threshold. Successfully tracked features are reconstructed by triangulation and further refined by an Extended Kalman Filter as described in Section 7.2.

The functions of the feature management are the extraction of new features, the estimation of the feature tracking probability, the selection of good features for a given camera position and the removal of features which are not of any use for further tracking. The whole management shall be an incremental process which runs in real-time and only uses a limited amount of memory. The tracking probability of a feature is denoted as the probability if a feature is able to be tracked successfully at a given camera position. In the following section the sequential estimation of this probability is described.

## 8.3. Tracking Probability

### 8.3.1. Probability Density Estimation

As the rotation around the camera center does not have any influence on the visibility of a point feature, if the feature is located inside the image, only the position of the camera in world coordinates is regarded as a useful information to decide whether a feature is worth tracking. What is known about the ability to track a feature at a given camera position are the observations of its tracking success in previous frames. The problem of modeling a probability distribution $p(\boldsymbol{x})$ of a random variable $\boldsymbol{x}$, given a finite set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ of observations, is known as density estimation. A widely used nonparametric method for creating probability distributions are Kernel density estimators. To obtain a smooth density model we choose a Gaussian kernel function. For a $D$-dimensional vector $\boldsymbol{x}$ the probability density can be denoted as

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi\sigma^2)^{D/2}} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_n\|^2}{2\sigma^2}\right), \tag{8.1}$$

where $N$ is the number of observation points $\boldsymbol{x}_n$, and $\sigma$ represents the variance of the Gaussian kernel function in one dimension.

Every observation of a feature belongs to one element of the class $C = \{s, f\}$, which simply holds the information whether the tracking step was successful (s) or the tracking failed (f). The probability density of the camera position is estimated for every element of the class $C$ separately. Let $p(\boldsymbol{x}|C = s)$ be the conditional probability density of the camera position for successfully tracked features and $p(\boldsymbol{x}|C = f)$ the conditional probability density for unsuccessfully tracked features. The marginal probability of tracking successes is given by $p(C = s) = \frac{N_s}{N}$ and for tracking failures by $p(C = f) = \frac{N_f}{N}$, where $N_s$ and $N_f$ are the number of successful and unsuccessful tracking steps respectively, and $N$ is the total number of observations.

The probability $p_t(\boldsymbol{x})$ if a feature can be tracked at a given camera position $\boldsymbol{x}$ is estimated with

$$p_t(\boldsymbol{x}) = p(C = s|\boldsymbol{x}). \tag{8.2}$$

When applying the Bayes' theorem, the tracking probability can be written as

$$\begin{aligned} p_t(\boldsymbol{x}) &= \frac{p(\boldsymbol{x}|C = s)p(C = s)}{p(\boldsymbol{x})} \\ &= \frac{p(\boldsymbol{x}|C = s)p(C = s)}{p(\boldsymbol{x}|C = s)p(C = s) + p(\boldsymbol{x}|C = f)p(C = f)} \\ &= \frac{p(\boldsymbol{x}|C = s)N_s}{p(\boldsymbol{x}|C = s)N_s + p(\boldsymbol{x}|C = f)N_f}. \end{aligned} \tag{8.3}$$

The estimation of probability densities by using Equation (8.1), however, has the major drawback that with an increasing number of observations the complexity for storage and

computation is increasing linearly with the number of observations, which is not feasible for an online application. Our approach for the density estimation is based on a finite set of Gaussian mixtures.

The use of mixture models for an efficient computation of clusters in huge data sets has already been addressed. In [97] the Iterative Pairwise Replacement Algorithm (IPRA) is proposed, which is a computationally efficient method for conditional density estimation for very large data sets where kernel estimates are approximated by much smaller mixtures. Goldberger [33] uses a hierarchical approach to reduce large Gaussian mixtures to smaller mixtures by minimizing a KL-based distance between them. Zhang [121] presents another efficient approach for simplifying mixture models by using a $L_2$ norm as distance measure between the mixtures. Zivkovic [126] presents a recursive solution for estimating the parameters of a mixture with a simultaneous selection of the number of components.

We use a method which is similar to [97], but instead of clustering a large data set we use the method for an online density estimation with a finite mixture model. A mixture with a finite number of Gaussians is maintained for both the successfully and unsuccessfully tracked features. Now we regard the multivariate Gaussian mixture distribution of the successfully tracked features, which can be written as

$$p(\boldsymbol{x}|C=s) = \sum_{k=1}^{K} \omega_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \Sigma_k) \quad \text{with} \quad \sum_{k=1}^{K} \omega_k = 1, \tag{8.4}$$

where $\boldsymbol{\mu}_k$ is the $D$-dimensional mean vector and $\Sigma_k$ the $D \times D$ covariance matrix. The mixing coefficients $\omega_k = \frac{N_k}{N_s}$ hold the information how many observations $N_k$ have affected this Gaussian $k$. The probability distribution $p(\boldsymbol{x}|C=f)$ is defined in the same way. Together with Equation (8.3) the tracking probability for a given camera position can be estimated.

The mixture model is built and maintained as follows. Depending on the tracking success, an observation is assigned to a class $C$, which means that either the distribution $p(\boldsymbol{x}|C=s)$ or the distribution $p(\boldsymbol{x}|C=f)$ is updated. First for every observation a Gaussian kernel function is created where every kernel can be regarded as a Gaussian of the mixture model. If the maximum number of mixtures $K$ is reached, the two most similar mixtures are merged and a new Gaussian is created by taking the kernel function from the proximate observation.

### 8.3.2. Similarity Measure

A similarity matrix is maintained where the similarity of all Gaussians among each other is stored.

Scott [97] defined the similarity measure between two density functions $p_1$ and $p_2$ as

$$\text{sim}(p_1, p_2) = \frac{\int_{-\infty}^{\infty} p_1(x)p_2(x)dx}{(\int_{-\infty}^{\infty} p_1^2(x)dx \int_{-\infty}^{\infty} p_2^2 dx)^{1/2}}. \tag{8.5}$$

Equation (8.5) can be considered as a correlation between the two densities.

If $p_1(x) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_1, \Sigma_1)$ and $p_2(x) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_2, \Sigma_2)$ are normal distributions, the similarity measure can be calculated by

$$\text{sim}(p_1, p_2) = \frac{(2^D|\Sigma_1\Sigma_2|^{1/2})^{1/2}}{|\Sigma_1 + \Sigma_2|^{1/2}} \exp(\Delta) \tag{8.6}$$

with

$$\Delta = -\frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T(\Sigma_1 + \Sigma_2)^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2). \tag{8.7}$$

This equation follows from the fact that

$$\int_{-\infty}^{\infty} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_1, \Sigma_1)\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_2, \Sigma_2) = \mathcal{N}(\boldsymbol{0}|\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2, \Sigma_1 + \Sigma_2). \tag{8.8}$$

The two Gaussians for which the similarity measure of Equation (8.5) is smallest are used for the merging step which is described in the next section.

### 8.3.3. Merging Gaussian Distributions

The merge operation of the two most similar Gaussians is carried out as follows. Now we assume that the $i^{th}$ and the $j^{th}$ component are merged into the $i'^{th}$ component of the mixture. Since a mixing coefficient represents the number of observations which affect a distribution, the new number of observations is $N_{i'} = N_i + N_j$, and therefore $\omega_{i'}$ is updated by

$$\omega_{i'} = \omega_i + \omega_j. \tag{8.9}$$

The mean of the new distribution can be calculated by

$$\begin{aligned} \boldsymbol{\mu}_{i'} &= \frac{1}{N_{i'}} \sum_{n=1}^{N_{i'}} \boldsymbol{x}_n = \frac{1}{N_{i'}}(\sum_{n=1}^{N_i} \boldsymbol{x}_n + \sum_{n=1}^{N_j} \boldsymbol{x}_n) \\ &= \frac{1}{N_{i'}}(N_i\boldsymbol{\mu}_i + N_j\boldsymbol{\mu}_j) = \frac{1}{\omega_{i'}}(\omega_i\boldsymbol{\mu}_i + \omega_j\boldsymbol{\mu}_j). \end{aligned} \tag{8.10}$$

After the mean is computed, the covariance $\Sigma_{i'}$ can be updated as follows

$$\begin{aligned} \Sigma_{i'} &= \frac{1}{N_{i'}} \sum_{n=1}^{N_{i'}} (\boldsymbol{x}_n - \boldsymbol{\mu}_{i'})(\boldsymbol{x}_n - \boldsymbol{\mu}_{i'})^T \\ &= \frac{1}{N_{i'}} \sum_{n=1}^{N_{i'}} \boldsymbol{x}_n\boldsymbol{x}_n^T - \boldsymbol{\mu}_{i'}\boldsymbol{\mu}_{i'}^T \\ &= \frac{1}{N_{i'}} (\sum_{n=1}^{N_i} \boldsymbol{x}_n\boldsymbol{x}_n^T + \sum_{n=1}^{N_j} \boldsymbol{x}_n\boldsymbol{x}_n^T) - \boldsymbol{\mu}_{i'}\boldsymbol{\mu}_{i'}^T \\ &= \frac{1}{N_{i'}} (N_i(\Sigma_i + \boldsymbol{\mu}_i\boldsymbol{\mu}_i^T) + N_j(\Sigma_j + \boldsymbol{\mu}_j\boldsymbol{\mu}_j^T)) - \boldsymbol{\mu}_{i'}\boldsymbol{\mu}_{i'}^T \\ &= \frac{1}{\omega_{i'}} (\omega_i(\Sigma_i + \boldsymbol{\mu}_i\boldsymbol{\mu}_i^T) + \omega_j(\Sigma_j + \boldsymbol{\mu}_j\boldsymbol{\mu}_j^T)) - \boldsymbol{\mu}_{i'}\boldsymbol{\mu}_{i'}^T. \end{aligned} \tag{8.11}$$

After the merge operation, the $j^{th}$ component can be used by a new observation to represent a new Gaussian. It can be regarded as a Kernel estimate with a Gaussian kernel function. For a new observation, the camera position is assigned to $\boldsymbol{x}_j$ and the covariance is set to $\sigma^2\boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix and $\sigma$ determines the size of the Parzen window. The parameter $\sigma$ affects the smoothness of the resulting mixture model and must be chosen with respect to the world coordinate system. If, for example, the camera position vector is given in centimeters, with $\sigma = 5$, a convincing probability distribution can be created for an indoor camera tracking. The weight $\omega_j$ is initialized with $\omega_j = \frac{1}{N_c}$, where $N_c$ is the number of observations of the assigned class.

## 8.4. Feature Population Control

### 8.4.1. Feature Selection

Features which have a precisely reconstructed 3D coordinate have no need for any reconstruction or refinement step. If we know that such features are not very likely to be tracked from the current camera position, it is probably not of any use for the pose estimation and it can be disregarded for a tracking step. Features which do not have a valid 3D coordinate are selected for the tracking step in every case, because it is important that a feature point gets triangulated fast, and an exact 3D position is reconstructed, so that the feature will be beneficial for the camera pose estimation.

Before the tracking step all features which have not been tracked successfully in the last frame are projected into the image with the last camera position in order to provide a good starting position for the features in the iterative alignment. The tracking probabilities of all features which are located inside the current image are calculated with Equation (8.3) and the features are sorted by their probability in descending order. Now the feature tracking is applied on the sorted list of features until a minimum number of features has been tracked successfully. In our implementation we stop after 30 successfully tracked features with a valid 3D coordinate, which should be enough for a robust pose estimation.

The benefit of this approach is that the total number of tracked features is kept at a minimum if most of the features are tracked successfully, but if there are lots of tracking failures due to occlusion or strong motion blur, as many features as needed are tracked until a robust camera pose estimation is possible.

### 8.4.2. Feature Extraction

Most point-based feature tracking methods use the well known Harris Corner Detector [39], which is based on the eigenvalue analysis of the gradient structure of an image patch. Another simple but very efficient approach called FAST (Features from Accelerated Segment Test) was presented by Roston et al. [92]. Their method analyses the intensity values on a circle of 16 pixels surrounding the corner point. If at least 12 contiguous pixels are all above or all below the intensity of the center by some threshold, this point is

regarded as a corner feature. For reasons of efficiency we used the FAST feature detector in our implementation.

To avoid too many features and overlapping patches, a new feature is only extracted if no other feature points exist within a minimum distance to this feature in the image. New features are extracted if the total number of features with $p_t(\boldsymbol{x}) > 0.5$ for the current camera position $\boldsymbol{x}$ falls below a given threshold.

### 8.4.3. Feature Removal

In order to decide if a feature is valuable for further tracking, a measure of usefulness has to be defined. If the tracking probability $p_t(\boldsymbol{x})$ for any camera position $x$ is smaller than 0.5, a feature can be regarded as dispensable. The correct computation of the maximum of $p_t(\boldsymbol{x})$ with the expectation maximization algorithm for every feature is computationally too expensive.

When $\boldsymbol{\mu}_{k,s}$ are the Gaussian means of the mixture model representing successfully tracked features, we approximate the maximum of the tracking probability by evaluating $p_t$ at all positions $\boldsymbol{\mu}_{k,s}$ by the following equation:

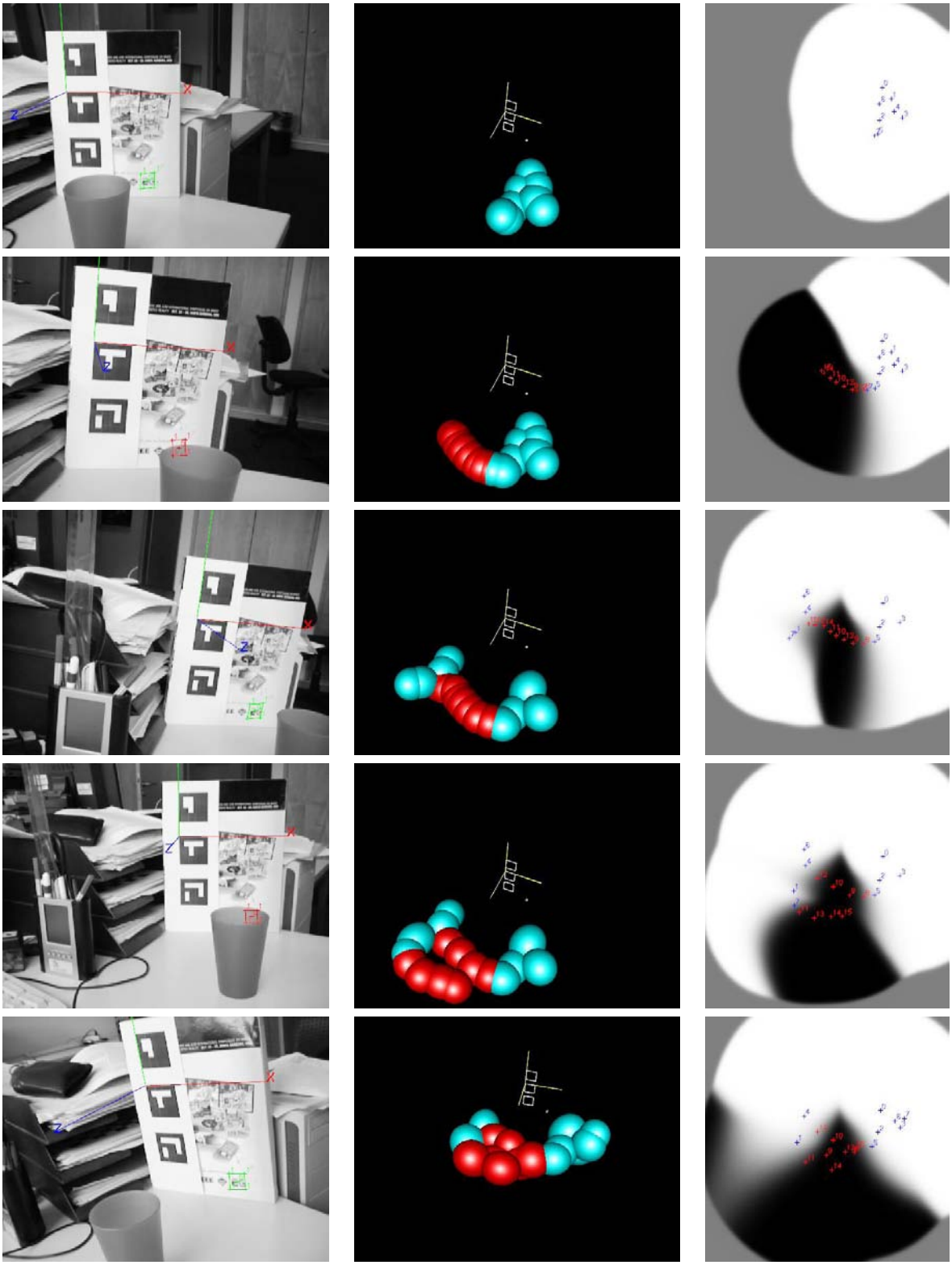
$$p_{\max} \simeq \max_k p_t(\boldsymbol{\mu}_{k,s}). \tag{8.12}$$

If $p_{\max} < 0.5$ holds, then no camera position exists where this feature is likely to be tracked, and it can be removed from the feature map without the concern of losing valuable information.

If a feature point gets lost and the 3D coordinate of that feature has not been reconstructed yet, this feature is removed as well, because without a valid 3D coordinate it is not possible to re-project the feature back into the image for further tracking.

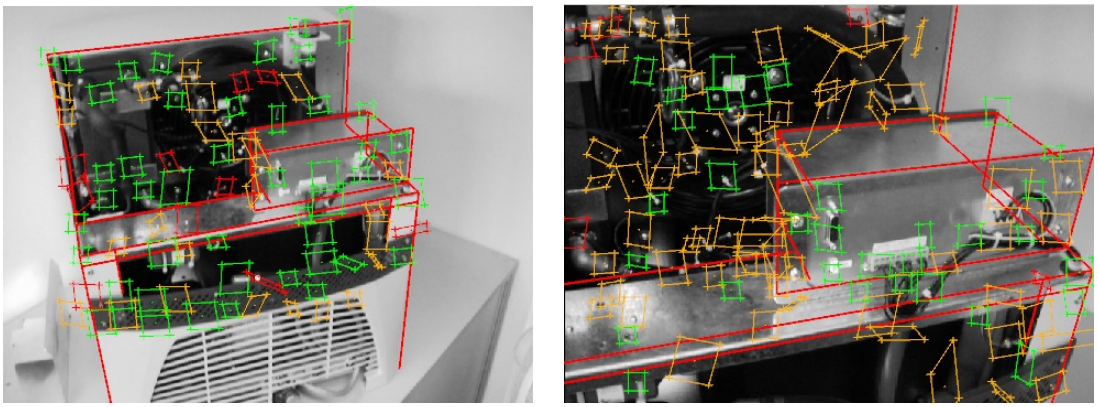
## 8.5. Experimental Results

To evaluate if the tracking probability distribution of a single feature is estimated correctly the following test scenario is created. The camera pose is computed by tracking a set of planar fiducial markers, which are located in the x/y-plane. A point feature is extracted manually on the same plane. In Figure 8.1(a) some frames of this sequence can be seen. When the camera is moved around, the point feature gets lost while it is occluded by an object, but it is tracked successfully, when it gets visible again. The Gaussian mixture model is visualized in Figure 8.1(b) by a set of confidence ellipsoids, which are drawn in blue and red for $p(\boldsymbol{x}|C = s)$ and $p(\boldsymbol{x}|C = f)$ respectively. The number of Gaussians is limited to 8 for each mixture model in this particular example. In Figure 8.1(c) the probability distribution $p_t(\boldsymbol{x})$ in the x/z-plane together with the Gaussian means is shown. It can be seen that the camera positions where the point feature was visible or occluded is correctly represented by the mixture model of tracking successes or tracking failures respectively. The probability distribution clearly illustrates that the tracking probability falls to 0 at camera positions where the feature is occluded.



(a) (b) (c)

Figure 8.1.: Probability density map of camera position for a single feature. In (a) some frames of the test sequence are shown. (b) visualizes the Gaussian mixture models of camera positions where the feature has been tracked successfully (blue) and where the tracking failed (red). In (c) the tracking probability in the x/z-plane can be seen.



(a) (b)

Figure 8.2.: Feature tracking and pose estimation results of a test sequence showing an industrial scenario. The green rectangles represent the successfully tracked features, red symbolizes tracking failures and orange features have been disabled due to a too small tracking probability.

An image sequence showing an industrial scenario is used for the further experiments. Two frames of the sequence can be seen in figure 8.2. The tracking of planar patches is very difficult in this sequence, because the regarded object does not consist of many well-textured planar parts, which is assumed for the template-based feature tracking. Many features can only be tracked from a limited viewpoint area, and therefore only a subset of all available features can be tracked successfully in most of the image frames.

In order to evaluate the quality of the tracking probability estimation, all available features are used as an input for the tracking step and it is observed whether the features compared to their tracking probability are tracked successfully or not. In Figure 8.3 histograms are plotted which show the number of successfully and unsuccessfully tracked features with their corresponding tracking probability. It can be seen that the major part of features with a high tracking probability has been, indeed, tracked successfully.

An analysis of the processing time is carried out on a Pentium 4 with 2.8GHz and a firewire camera with a resolution of $640 \times 480$ pixels. The average computational costs for every individual step are shown in Table 8.1. Without the feature extraction, the tracking system can run at a frame rate of 20Hz.

If no feature selection is performed, on average 93.9 features are used in the feature tracking step, and only 49.0% of all features can be tracked successfully. The average runtime of the tracking step is at 64.36 milliseconds. With the selection of the most probable features, on average only 48.94 features are analyzed per frame in the tracking step. The success rate of the feature tracking is at 83.0% and the mean computation time is lowered to 29.08 milliseconds with no significant difference of the quality of the pose estimation.

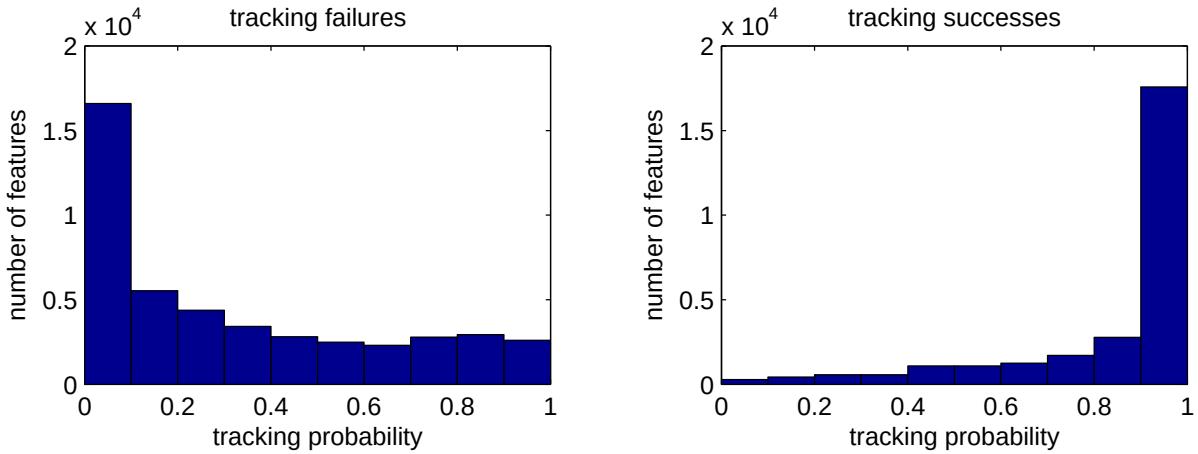



Figure 8.3.: Histograms of successfully and unsuccessfully tracked features with their corresponding tracking probability.

| **prediction step** | **time in ms** |
|---|---|
| build image pyramid | 10.53 |
| feature selection and tracking | 29.08 |
| pose estimation | 2.74 |
| update feature probability | 1.94 |
| reconstruct feature points | 5.53 |
| extract new features | 5.93 |
| **total time without feature extraction** | 49.82 |

Table 8.1.: Average processing time of the individual steps of the tracking approach.

## 8.6. Conclusion

We have presented an approach for real-time camera pose estimation which uses an efficient feature management to store many features and to track only those features which are most likely to be tracked from a given camera position. The tracking probability for every feature is estimated online during the tracking and no preprocessing is necessary. Features which are only visible in a limited area of viewpoints are only tracked at those certain camera positions and ignored at any other viewpoints. Even if they are occluded for a long time, reliable features are not deleted, but kept in the feature set as long as a camera position exists from which the feature can be tracked successfully. Not only the visibility, but also the robustness of a feature is represented by the tracking probability. Tracking failures due to reflections or spotlights at certain camera positions are also modeled correctly.

# 9. Conclusion

## 9.1. Summary

In this thesis computer vision-based tracking techniques have been developed, which aim at the creation of augmented reality applications with minimal requirements for the preparation of reference data.

Edge-based tracking methods have been used to create a tracking system, where a given 3D line model is used as a reference object and the camera pose is estimated by aligning the line model onto the image gradient. The state-of-the-art approach has been extended with an adaptive method, which learns the edge's visual appearance by creating a set of multiple appearances of every control point of the edge model. This on-line information, which is gathered during the successful tracking, helps to improve the robustness of the edge-based model tracker especially for large camera movements.

To avoid the manual creation of a line model, a method for the automatic generation of contour models was developed. With a given polygonal model the 3D contours are created in real-time with the support of pixel shader graphics hardware. The extraction of the line model was directly integrated into the edge-based model tracking, and a system was created which is able to align a virtual model onto a real model at high frame rates. The presented two-stage approach first tracked an image edge from frame to frame and then performed the 3D model registration. The result was an edge-based tracking system which is able to track very poorly textured objects and even scenarios where only silhouette edges can be used as distinct features. The usability aspects of this method are very high and outperform other generation techniques which have been proposed recently.

The development of point-based tracking systems led to promising results for the markerless camera pose estimation. Texture- based tracking methods like the template alignment with various warp functions have been investigated and improved to establish robust point-based tracking algorithms. Additional scale invariance was achieved with the acquisition of template patches of multiple resolution levels and the selection of that resolution during the tracking step which is most appropriate in the current camera image. It was experimentally shown that such tracking methods can be used for the successful estimation of the camera pose, if the geometry of the scene is known.

Reconstruction algorithms were developed and a point feature based tracking system was created, which is able to learn features in parts of the image with unknown scene geometry. Not only the 3D coordinates, but also the surface normal directions were reconstructed and a map of features was created, which makes the tracking system more stable, the longer a scene is observed.

To counteract the problems of real-time capability if the feature map becomes too large, a novel feature management system was developed. With a statistical analysis of the ability to track a feature which is carried out during the runtime of the tracking, it is possible to select only a limited subset of all features which are visible from a given camera viewing position. The resulting tracking system is able to estimate the camera pose with an equal quality but with much less demand for processing time.

## 9.2. Future Work

Although many difficulties of the camera pose estimation could be solved with methods presented in this thesis, there are still many open problems. A variety of possibilities exist to improve the current state of our tracking system.

The problem of the re-initialization is totally disconnected with the current feature tracking system. A combination of the local search and a wide baseline matching could help to create a re-initialization module which can be better integrated into the whole tracking process. The possible solution could be the integration of the randomized tree classifier directly with the KLT patches.

Another possibility for future improvements could be the reconstruction of line features and the integration of line features into a feature map. Poorly textured scenarios might be handled better if additional lines are tracked and reconstructed during runtime.

The whole implementation could be improved by shifting pixel related computation onto the graphics hardware. The tracking of KLT feature with the support of pixel shaders might be a good option to save processing time.

The feature management system has the disadvantage that dynamic scenes are not modeled correctly by the system. An extension of the approach is necessary so that the system is beneficial for non-static scenes.

Furthermore, the integration of other sensors, like inertial devices or GPS receivers, could be a reasonable improvement of the robustness of the tracking system.

# Appendix A.

# Derivations of the Inverse Compositional Image Alignment

This appendix describes the computation of the intensity based image alignment which is presented in Section 6.5. The equations of the parameter update for several different motion models are derived.

The inverse compositional approach of Baker and Matthews [5] minimizes the following error function with respect to $\Delta\boldsymbol{p}$:

$$\epsilon = \sum_x [T(g(\boldsymbol{x}; \boldsymbol{\Delta p})) - I(g(\boldsymbol{x}; \boldsymbol{p}))]^2, \tag{A.1}$$

where $T(x)$ is the template, $I(x)$ the current image and $g(\boldsymbol{x}; \boldsymbol{p})$ a warp function with the warp parameters $\boldsymbol{p}$. Substituting $T(g(\boldsymbol{x}; \boldsymbol{\Delta p}))$ with its first-order Taylor expansion and setting the derivatives of equation (A.1) with respect to all parameters $\Delta\boldsymbol{p}$ of to zero, results in the linear system:

$$H\Delta\boldsymbol{p} = \boldsymbol{b}, \tag{A.2}$$

with

$$H = \sum_x \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right]^T \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right] \tag{A.3}$$

$$\boldsymbol{b} = \sum_x \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right]^T [I(g(\boldsymbol{x}; \boldsymbol{p}) - T(x)]\,. \tag{A.4}$$

The values of $\nabla T = (T_x, T_y)$ are the image gradient of the template in x- and y-direction. The Jacobian $\frac{\partial g}{\partial \boldsymbol{p}}$ consists of the partial derivatives of the warp function $g$ and is evaluated at $\boldsymbol{p} = \boldsymbol{0}$, i.e. with those parameters which represent the identity warp. For better readability, we introduce the vector $h(x)$, which is defined by

$$h(x) = \left[\nabla T \frac{\partial g}{\partial \boldsymbol{p}}\right]^T. \tag{A.5}$$

The matrix $H$ and the vector $\boldsymbol{b}$ can then be written as

$$H = \sum_x h(x)h(x)^T \tag{A.6}$$

$$\boldsymbol{b} = \sum_x h(x)I_t, \tag{A.7}$$

where $I_t = I(g(\boldsymbol{x};\boldsymbol{p})) - T(x)$ is the difference between the warped image and the template. If $H$ is invertible, the parameter increment $\Delta\boldsymbol{p}$ can then be computed by

$$\Delta\boldsymbol{p} = H^{-1}\boldsymbol{b}. \tag{A.8}$$

The warp function can now be updated by

$$g(\boldsymbol{x},\boldsymbol{p}) \leftarrow g(\boldsymbol{x},\boldsymbol{p}) \circ g(\boldsymbol{x},\boldsymbol{\Delta p})^{-1}. \tag{A.9}$$

With this update of the warp function the new parameter vector $\boldsymbol{p}$ can be computed.

## A.1. Translation

For the purely translational case, the warp function can be expressed by

$$g(\boldsymbol{x};\boldsymbol{p}) = \boldsymbol{x} + \boldsymbol{p}, \tag{A.10}$$

where $\boldsymbol{p} = \begin{bmatrix} t_1 & t_2 \end{bmatrix}^T$ is the translation increment in x- and y-direction. Since the Jacobian of equation (A.10) is $\frac{\partial g}{\partial \boldsymbol{p}} = \left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, the vector $h(x)$ can be computed by $h(x) = \begin{bmatrix} T_x & T_y \end{bmatrix}^T$. For the matrix $H$ and the vector $\boldsymbol{b}$ we get

$$H = \sum_x \begin{bmatrix} T_x^2 & T_x T_y \\ T_x T_y & T_y^2 \end{bmatrix} \tag{A.11}$$

$$\boldsymbol{b} = \sum_x \begin{bmatrix} T_x \\ T_y \end{bmatrix} I_t. \tag{A.12}$$

After computing the parameter increment by $\Delta\boldsymbol{p} = H^{-1}\boldsymbol{b}$, the parameter vector $\boldsymbol{p}$ can simply be updated by

$$\boldsymbol{p} \leftarrow \boldsymbol{p} - \Delta\boldsymbol{p}. \tag{A.13}$$

## A.2. Scale

If changes in scale are regarded additionally to the translation, the warp function can be written as

$$g(\boldsymbol{x};\boldsymbol{p}) = s\boldsymbol{x} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}, \tag{A.14}$$

where $s$ is the scaling factor. With the parameter vector $\boldsymbol{p} = \begin{bmatrix} t_1 & t_2 & s \end{bmatrix}^T$ and with the Jacobian

$$\frac{\partial g}{\partial \boldsymbol{p}} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \end{bmatrix} \tag{A.15}$$

the vector $h(x)$ can be computed by

$$h(x) = \begin{bmatrix} T_x & T_y & xT_x + yT_y \end{bmatrix}^T . \tag{A.16}$$

The matrix $H$ and the vector $\boldsymbol{b}$ are calculated by

$$H = \sum_x \begin{bmatrix} T_x^2 & T_x T_y & xT_x^2 + yT_y \\ T_x T_y & T_y^2 & xT_x + yT_y^2 \\ xT_x^2 + yT_y & xT_x + yT_y^2 & (xT_x + yT_y)^2 \end{bmatrix} \tag{A.17}$$

$$\boldsymbol{b} = \sum_x \begin{bmatrix} T_x \\ T_y \\ xT_x + yT_y \end{bmatrix} I_t. \tag{A.18}$$

With $\Delta\boldsymbol{p} = \begin{bmatrix} \tilde{t}_1 & \tilde{t}_2 & \tilde{s} \end{bmatrix}^T = H^{-1}\boldsymbol{b}$ the inverse warp with the warping parameters $\Delta\boldsymbol{p}$ are calculated by

$$g(\boldsymbol{x};\boldsymbol{p})^{-1} = \frac{1}{s}\boldsymbol{x} - \frac{1}{s}\begin{pmatrix} t_1 \\ t_2 \end{pmatrix}. \tag{A.19}$$

The warp composition $g(g(\boldsymbol{x};\Delta\boldsymbol{p})^{-1};\boldsymbol{p})$ results in the following update of the warp parameters:

$$\boldsymbol{p} = \begin{pmatrix} t_1 \\ t_2 \\ s \end{pmatrix} \leftarrow \begin{pmatrix} t_1 - \frac{s}{\tilde{s}}\tilde{t}_1 \\ t_2 - \frac{s}{\tilde{s}}\tilde{t}_2 \\ \frac{s}{\tilde{s}} \end{pmatrix} \tag{A.20}$$

## A.3. Rotation

A warp function which consists of a translation and a two-dimensional rotation can be written as

$$g(\boldsymbol{x};\boldsymbol{p}) = R(\theta)\boldsymbol{x} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}, \tag{A.21}$$

with the rotation matrix

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}. \tag{A.22}$$

If the parameter vector is defined as $\boldsymbol{p} = \begin{bmatrix} t_1 & t_2 & \theta \end{bmatrix}^T$, the Jacobian of the warp function is

$$\frac{\partial g}{\partial \boldsymbol{p}} = \begin{bmatrix} 1 & 0 & -x\sin(\theta) - y\cos(\theta) \\ 0 & 1 & x\cos(\theta) - y\sin(\theta) \end{bmatrix}. \tag{A.23}$$

Evaluating the Jacobian at $\boldsymbol{p} = \boldsymbol{0}$ yields

$$\frac{\partial g}{\partial \boldsymbol{p}} = \begin{bmatrix} 1 & 0 & -y \\ 0 & 1 & x \end{bmatrix}. \tag{A.24}$$

With the vector $h(x) = \begin{bmatrix} T_x & T_y & -yT_x + xT_y \end{bmatrix}$ the approximated Hessian $H$ and the vector $\boldsymbol{b}$ can be computed by

$$H = \sum_x \begin{bmatrix} T_x^2 & T_x T_y & -yT_x^2 + xT_x T_y \\ T_x T_y & T_y^2 & -yT_x T_y + xT_y^2 \\ -yT_x^2 + xT_x T_y & -yT_x T_y + xT_y^2 & (-yT_x + xT_y)^2 \end{bmatrix} \tag{A.25}$$

$$\boldsymbol{b} = \sum_x \begin{bmatrix} T_x \\ T_y \\ -yT_x + xT_y \end{bmatrix} I_t. \tag{A.26}$$

The inverse function of the warp defined in equation (A.21) can be written as

$$g(\boldsymbol{x}; \boldsymbol{p})^{-1} = R(-\theta)\boldsymbol{x} - R(-\theta) \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}. \tag{A.27}$$

With the parameter increment $\Delta\boldsymbol{p} = \begin{bmatrix} \tilde{t}_1 & \tilde{t}_2 & \tilde{\theta} \end{bmatrix}^T$ the update of the warp parameters can be calculated by

$$\boldsymbol{t} = \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \leftarrow \boldsymbol{t} - R(\theta - \tilde{\theta})\tilde{\boldsymbol{t}} = \begin{pmatrix} t_1 - (\cos(\theta - \tilde{\theta})\tilde{t}_1 - \sin(\theta - \tilde{\theta})\tilde{t}_2) \\ t_2 - (\sin(\theta - \tilde{\theta})\tilde{t}_1 + \cos(\theta - \tilde{\theta})\tilde{t}_2) \end{pmatrix} \tag{A.28}$$

$$\theta \leftarrow \theta - \tilde{\theta}. \tag{A.29}$$

## A.4. Affine Transformation

The full affine model uses, in addition to the translation vector $t$, a $2 \times 2$ affine transformation matrix $A$. The affine warp function can be expressed as

$$g(\boldsymbol{x}; \boldsymbol{p}) = A\boldsymbol{x} + t = \begin{pmatrix} 1 + a_1 & a_2 \\ a_3 & 1 + a_4 \end{pmatrix} x + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}. \tag{A.30}$$

With the parameter vector $\boldsymbol{p} = \begin{bmatrix} t_1 & t_2 & a_1 & a_2 & a_3 & a_4 \end{bmatrix}^T$ the Jacobian is computed by

$$\frac{\partial g}{\partial \boldsymbol{p}} = \begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix} \tag{A.31}$$

With the vector $h(x) = \begin{bmatrix} T_x & T_y & xT_x & yT_x & xT_y & yT_y \end{bmatrix}^T$ the matrix $H$ can be computed by

$$H = \sum_x \begin{bmatrix} T_x^2 & T_x T_y & xT_x^2 & yT_x^2 & xT_x T_y & yT_x T_y \\ T_x T_y & T_y^2 & xT_x T_y & yT_x Ty & xT_y^2 & yT_y^2 \\ xT_x^2 & xT_x T_y & x^2 T_x^2 & xyT_x^2 & x^2 T_x T_y & xyT_x T_y \\ yT_x^2 & yT_x T_y & xyT_x^2 & y^2 T_x^2 & xyT_x T_y & y^2 T_x T_y \\ xT_x T_y & xT_y^2 & x^2 T_x T_y & xyT_x T_y & x^2 T_y^2 & xyT_y^2 \\ yT_x T_y & yT_y^2 & xyT_x T_y & y^2 T_x T_y & xyT_y^2 & y^2 T_y^2 \end{bmatrix}. \tag{A.32}$$

If the inverse of the affine incremental warp function is given by

$$g(\boldsymbol{x};\Delta\boldsymbol{p})^{-1} = \tilde{A}^{-1}x - \tilde{A}^{-1}\tilde{t}, \tag{A.33}$$

the composed warp can be calculated with

$$g(g(\boldsymbol{x};\Delta\boldsymbol{p})^{-1};\boldsymbol{p}) = A(\tilde{A}^{-1}\boldsymbol{x} - \tilde{A}^{-1}\tilde{\boldsymbol{t}}) + \boldsymbol{t}. \tag{A.34}$$

For the update of the translation vector we get $\boldsymbol{t} \leftarrow \boldsymbol{t} - A\tilde{A}^{-1}\tilde{t}$ and the affine matrix is updated by $A \leftarrow A\tilde{A}^{-1}$. With the updated $A$ and $\boldsymbol{t}$ the new warp parameters $\boldsymbol{p}$ can be calculated.

To simplify the update step it can be assumed that for small parameter updates $g(\boldsymbol{x};\boldsymbol{p})^{-1} \approx g(\boldsymbol{x};-\boldsymbol{p})$ holds. The affine warp can then be updated with $A \leftarrow A\tilde{A}$ and the translation vector with $\boldsymbol{t} \leftarrow \boldsymbol{t} + A\tilde{t}$. Although is is a rough approximation, this update strategy works surprisingly well and avoids the inversion of the warp function.

## A.5. Affine Model with Illumination Correction

In Section 6.5.1 the template alignment with illumination compensation for a general warp function is discussed.

For the affine model as described in [67], the vector $h(x)$ of Equation (6.17) can be computed with

$$h(x) = \begin{bmatrix} T_x & T_y & xT_x & yT_x & xT_y & yT_y & T & 1 \end{bmatrix}^T, \tag{A.35}$$

and for the matrix $H$ we get

$$H = \sum_x \begin{bmatrix} T_x^2 & T_xT_y & xT_x^2 & yT_x^2 & xT_xT_y & yT_xT_y & T_xT & T_x \\ T_xT_y & T_y^2 & xT_xT_y & yT_xTy & xT_y^2 & yT_y^2 & T_yT & T_y \\ xT_x^2 & xT_xT_y & x^2T_x^2 & xyT_x^2 & x^2T_xT_y & xyT_xT_y & xT_xT & xT_x \\ yT_x^2 & yT_xT_y & xyT_x^2 & y^2T_x^2 & xyT_xT_y & y^2T_xT_y & yT_xT & yT_x \\ xT_xT_y & xT_y^2 & x^2T_xT_y & xyT_xT_y & x^2T_y^2 & xyT_y^2 & xT_yT & xT_y \\ yT_xT_y & yT_y^2 & xyT_xT_y & y^2T_xT_y & xyT_y^2 & y^2T_y^2 & yT_yT & yT_y \\ T_xT & T_yT & xT_xT & yT_xT & xT_yT & yT_yT & T^2 & T \\ T_x & T_y & xT_x & yT_x & xT_y & yT_y & T & 1 \end{bmatrix}. \tag{A.36}$$

The illumination parameters $\lambda$ and $\delta$ are independent from the warp parameters and can be updates as described in Section 6.5.1 for all warp functions. The parameters of the warp function, though, depend on the illumination parameter $\lambda$. For one iteration the linear solution is represented by the vector

$$q = \begin{pmatrix} \tilde{\lambda}\Delta\boldsymbol{p}^T & \tilde{\lambda} & \tilde{\delta} \end{pmatrix}^T, \tag{A.37}$$

with the warp parameter increment $\Delta\boldsymbol{p}$ and the illumination parameters $\tilde{\lambda}$ and $\tilde{\delta}$. By regarding these equations it can be seen that the elements of the vector $\boldsymbol{q}$ which correspond to the warp parameter vector $\Delta\boldsymbol{p}$ have to be divided by $\tilde{\lambda}$, before the warp parameters can be updated just as in the model without the illumination compensation.

## A.6. Homography

Let $C$ be the projective transformation defined by

$$C = \begin{pmatrix} 1+p_3 & p_4 & p_1 \\ p_5 & 1+p_6 & p_2 \\ p_7 & p_8 & 1 \end{pmatrix}. \tag{A.38}$$

The parametrization is chosen in that way, that the elements of the parameter vector $\boldsymbol{p} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{bmatrix}^T$ represent similar transformations as in the models with pure translation and the full affine warp.

The warp function representing this homography can be written as

$$g(\boldsymbol{x};\boldsymbol{p}) = \frac{1}{p_7 x + p_8 y + 1} \begin{pmatrix} (1+p_3)x + p_4 y + p_1 \\ p_5 x + (1+p_6)y + p_2 \end{pmatrix}. \tag{A.39}$$

Computing the Jacobi matrix of this function yields

$$\frac{\partial g}{\partial \boldsymbol{p}} = \frac{1}{p_7 x + p_8 y + 1} \begin{pmatrix} 1 & 0 & x & y & 0 & 0 & \frac{-x((1+p_3)x+p_4y+p_1)}{p_7x+p_8y+1} & \frac{-y((1+p_3)x+p_4y+p_1)}{p_7x+p_8y+1} \\ 0 & 1 & 0 & 0 & x & y & \frac{-x(p_5x+(1+p_6)y+p_2)}{p_7x+p_8y+1} & \frac{-y(p_5x+(1+p_6)y+p_2)}{p_7x+p_8y+1} \end{pmatrix}, \tag{A.40}$$

and for the evaluation at $\boldsymbol{p} = \boldsymbol{0}$ we get

$$\left.\frac{\partial g}{\partial \boldsymbol{p}}\right|_{\boldsymbol{p}=\boldsymbol{0}} = \begin{pmatrix} 1 & 0 & x & y & 0 & 0 & -x^2 & -xy \\ 0 & 1 & 0 & 0 & x & y & -xy & -y^2 \end{pmatrix}. \tag{A.41}$$

With the vector $h(\boldsymbol{x}) = \begin{bmatrix} T_x & T_y & xT_x & yT_x & xT_y & yT_y & -x^2T_x - xyT_y & -xyT_x - y^2T_y \end{bmatrix}^T$ the approximated Hessian is calculated by

$$H = \sum_x h(\boldsymbol{x}) h(\boldsymbol{x})^T. \tag{A.42}$$

If $\tilde{C}$ is the homography of the solution vector $\Delta\boldsymbol{p}$, then the update of the warp function is given by the warp composition

$$C \leftarrow C\tilde{C}^{-1}. \tag{A.43}$$

After homogenizing $C$ and subtracting the identity matrix, the new warp parameters $\boldsymbol{p}$ are obtained.

# Bibliography


[1] Abdel-Aziz, Y., and Karara, H. Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry. In *ASP Symposium on Close Range Photogrammetry* (Falls Church, US-VA, 1971), American Society of Photogrammetry (ASP), pp. 1–18.

[2] Armstrong, M., and Zisserman, A. Robust object tracking. In *Proceedings of the Asian Conference on Computer Vision* (1995), vol. I, pp. 58–61.

[3] Azuma, R. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments 6*, 4 (1997), 355–385.

[4] Baker, S., Gross, R., Matthews, I., and Ishikawa, T. Lucas-kanade 20 years on: A unifying framework: Part 2. Tech. Rep. CMU-RI-TR-03-01, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, February 2003.

[5] Baker, S., and Matthews, I. Equivalence and efficiency of image alignment algorithms. In *Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition* (December 2001), vol. 1, pp. 1090 – 1097.

[6] Baker, S., and Matthews, I. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision 56*, 3 (March 2004), 221 – 255.

[7] Bar-Shalom, Y. *Tracking and data association*. Academic Press Professional, Inc., San Diego, CA, USA, 1987.

[8] Bay, H., Tuytelaars, T., and Gool, L. J. V. Surf: Speeded up robust features. In *ECCV (1)* (2006), pp. 404–417.

[9] Bencina, R., and Kaltenbrunner, M. The design and evolution of fiducials for the reactivision system. In *Proceedings of the 3rd International Conference on Generative Systems in the Electronic Arts (3rd Iteration 2005)* (Melbourne, Australia, 2005).

[10] Berger, M., Auer, T., Bachler, G., Scherer, S., and Pinz, A. 3d model based pose determination in real-time: Strategies, convergences, accuracy. In *Proc. International Conference on Pattern Recognition (ICPR)* (Barcelona, Spain, September 2000), vol. 4, pp. 567–570.

[11] Bleser, G., Wohlleber, C., Becker, M., and Stricker, D. Fast and stable tracking for ar fusing video and inertial sensor data. In *International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (2006), P. U. of West Bohemia, Ed., pp. 109–115.

[12] Bleser, G., Wuest, H., and Stricker, D. Online camera pose estimation in partially known and dynamic scenes. In *ISMAR* (2006), pp. 56–65.

[13] Bouguet, J.-Y. Camera calibration toolbox for matlab.

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

[15] Chen, R., and Liu, J. S. Mixture kalman filters. *Journal Of The Royal Statistical Society Series B 62*, 3 (2000), 493–508. available at http://ideas.repec.org/a/bla/jorssb/v62y2000i3p493-508.html.

[16] Chiba, N., and Kanade, T. A tracker for broken and closely spaced lines. In *Proceedings of the 1996 International Society for Photogrammetry and Remote Sensing Conference (ISPRS '98)* (1998), vol. XXXII, pp. 676 – 683.

[17] Cho, Y., and Neumann, U. Multiring fiducial systems for scalable fiducial-tracking augmented reality. *Presence: Teleoper. Virtual Environ. 10*, 6 (2001), 599–612.

[18] Chum, O., and Matas, J. Matching with prosac " progressive sample consensus. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 220–226.

[19] Comport, A., Marchand, E., and Chaumette, F. A real-time tracker for markerless augmented reality. In *ACM/IEEE Int. Symp. on Mixed and Augmented Reality, ISMAR'03* (Tokyo, Japan, October 2003), pp. 36–45.

[20] David, P., DeMenthon, D., Duraiswami, R., and Samet, H. Simultaneous pose and correspondence determination using line features. In *CVPR* (Los Alamitos, CA, USA, 2003), IEEE Computer Society, pp. 424–431.

[21] David, P., Dementhon, D., Duraiswami, R., and Samet, H. Softposit: Simultaneous pose and correspondence determination. *Int. J. Comput. Vision 59*, 3 (2004), 259–284.

[22] Davison, A. J. Real-time simultaneous localisation and mapping with a single camera. *iccv 02* (2003), 1403.

[23] Davison, A. J., Reid, I. D., Molton, N. D., and Stasse, O. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 6 (2007), 1052–1067.

[24] Dementhon, D. F., and Davis, L. S. Model-based object pose in 25 lines of code. *Int. J. Comput. Vision 15*, 1-2 (1995), 123–141.

[25] Devernay, F., and Faugeras, O. Straight lines have to be straight: automatic calibration and removal of distortion from scenes of structured enviroments. *Mach. Vision Appl. 13*, 1 (2001), 14–24.

[26] Drummond, T., and Cipolla, R. Real-time tracking of complex structures with on-line camera calibration. In *BMVC* (1999).

[27] Favaro, P., Jin, H., and Soatto, S. A semi-direct approach to structure from motion. *The Visual Computer 192* (2003), 1–18.

[28] Fiala, M. Artag, a fiducial marker system using digital techniques. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern*

*Recognition (CVPR)* (Washington, DC, USA, 2005), vol. 2, IEEE Computer Society, pp. 590–596.

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

[30] F.Moreno-Noguer, V.Lepetit, and P.Fua. Accurate non-iterative o(n) solution to the pnp problem. In *IEEE International Conference on Computer Vision* (Rio de Janeiro, Brazil, October 2007).

[31] Fox, D., Hightower, J., Liao, L., Schulz, D., and Borriello, G. Bayesian filters for location estimation. In *IEEE Pervasive Computing* (2003), pp. 24–33.

[32] Gennery, D. B. Visual tracking of known three-dimensional objects. *Int. J. Comput. Vision 7*, 3 (1992), 243–270.

[33] Goldberger, J., and Roweis, S. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds. MIT Press, Cambridge, MA, 2005, pp. 505–512.

[34] Grabner, M., Grabner, H., and Bischof, H. Fast approximated sift. In *ACCV (1)* (2006), pp. 918–927.

[35] Hager, G. D., and Belhumeur, P. N. Efficient region tracking with parametric models of geometry and illumination. *IEEE Trans. Pattern Anal. Mach. Intell. 20*, 10 (1998), 1025–1039.

[36] Han, J. Y. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th annual ACM symposium on User interface software and technology (UIST)* (New York, NY, USA, 2005), ACM, pp. 115–118.

[37] Harris, C. Tracking with rigid models. *Active vision* (1993), 59–73.

[38] Harris, C., and Stennett, C. Rapid - a video rate object tracker. In *Proceedings of British Machine Vision Conference (BMVC)* (1990), pp. 73–77.

[39] Harris, C., and Stephens, M. A combined corner and edge detector. In *Proc. Alvey Vision Conf* (Univ. Manchester, 1988), pp. 147–151.

[40] Hartley, R. I., and Sturm, P. Triangulation. *Computer Vision and Image Understanding 68*, 2 (November 1997), 146–157.

[41] Hertzmann, A. Introduction to 3d non-photorealistic rendering: Silhouettes and outlines. In *SIGGRAPH 99*. ACM Press, 1999, ch. Course Notes.

[42] Hol, J., Schön, T., Luinge, H., Slycke, P., and Gustafsson, F. Robust real-time tracking by fusing measurements from inertial and vision sensors. *Journal of Real-Time Image Processing 2* (Nov 2007), 149–160.

[43] Horn, B. K. P., Hilden, H. M., and Negahdaripour, S. Closed-form solution of absolute orientation using orthonormal matrices. *J. Opt. Soc. Am. A 5*, 7 (1988), 1127.

[44] Huber, P. *Robust Statistics*. Wiley, New York, 1974.

[45] Intel. Open source computer vision library (opencv).

[46] Isard, M., and Blake, A. Condensation | conditional density propagation for visual tracking. *IJCV 29* (1998), 5–28.

[47] Isenberg, T., Freudenberg, B., Halper, N., Schlechtweg, S., and Strothotte, T. A developer's guide to silhouette algorithms for polygonal models. *IEEE Comput. Graph. Appl. 23*, 4 (2003), 28–37.

[48] Ishikawa, T., Matthews, I., and Baker, S. Efficient image alignment with outlier rejection. Tech. Rep. CMU-RI-TR-02-27, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 2002.

[49] Jin, H., Favaro, P., and Soatto, S. Real-Time feature tracking and outlier rejection with changes in illumination. In *IEEE Intl. Conf. on Computer Vision* (July 2001), pp. 684–689.

[50] Johnson, A., and Hebert, M. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 21*, 5 (May 1999), 433 – 449.

[51] Julier, S. J., and Uhlmann, J. K. New extension of the kalman filter to nonlinear systems. In *Signal Processing, Sensor Fusion, and Target Recognition VI* (1997), vol. 3068, pp. 182–193.

[52] Kato, H., and Billinghurst, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality* (Washington, DC, USA, 1999), IEEE Computer Society, p. 85.

[53] Ke, Y., and Sukthankar, R. Pca-sift: a more distinctive representation for local image descriptors. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)* (2004), vol. 2, pp. II–506–II–513 Vol.2.

[54] Kemp, C., and Drummond, T. Multi-modal tracking using texture changes. In *British Machine Vision Conference* (2004), pp. –.

[55] Klein, G., and Murray, D. Full-3d edge tracking with a particle filter. In *Proc. British Machine Vision Conference (BMVC'06)* (Edinburgh, September 2006), BMVA.

[56] Koch, R., Evers-Senne, J.-F., Schiller, I., Wuest, H., and Stricker, D. Architecture and tracking algorithms for a distributed mobile industrial ar system. In *Proceedings of the 5th International Conference on Computer Vision Systems, ICVS07* (March 2007).

[57] Koller, D., Danilidis, K., and Nagel, H.-H. Model-based object tracking in monocular image sequences of road traffic scenes. *Int. J. Comput. Vision 10*, 3 (1993), 257–281.

[58] Kosaka, A., and Nakazawa, G. Vision-based motion tracking of rigid objects using prediction of uncertainties. In *IEEE International Conference on Robotics and Automation* (1995).

[59] Lepetit, V., and Fua, P. Towards recognizing feature points using classification trees. Tech. rep., EPFL, CVLAB, 2004.

[60] Lepetit, V., Lagger, P., and Fua, P. Randomized trees for real-time keypoint recognition. In *Conference on Computer Vision and Pattern Recognition, San Diego, CA* (June 2005).

[61] Lepetit, V., Pilet, J., and Fua, P. Point matching as a classification problem for fast and robust object pose estimation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2004).

[62] Lowe, D. G. Fitting parameterized three-dimensional models to images. *IEEE Trans. Pattern Anal. Mach. Intell. 13*, 5 (1991), 441–450.

[63] Lowe, D. G. Robust model-based motion tracking through the integration of search and estimation. *International Journal of Computer Vision 8*, 2 (August 1992), 113–122.

[64] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[65] Lu, C.-P., Hager, G. D., and Mjolsness, E. Fast and globally convergent pose estimation from video images. *IEEE Trans. Pattern Anal. Mach. Intell. 22*, 6 (2000), 610–622.

[66] Lucas, B. D., and Kanade, T. An iterative image registration technique with an application to stereo vision (darpa). In *Proceedings of the 1981 DARPA Image Understanding Workshop* (April 1981), pp. 121–130.

[67] Ma, Y., Soatto, S., Kosecka, J., and Sastrys, S. S. *An invitation to 3D vision, from images to models*. Springer Verlag, 2003.

[68] Madritsch, F., and Gervautz, M. Ccd-camera based optical beacon tracking for virtual and augmented reality. *Comput. Graph. Forum 15*, 3 (1996), 207–216.

[69] Marchand, É., Bouthemy, P., and Chaumette, F. A 2d-3d model-based approach to real-time visual tracking. *Image Vision Comput. 19*, 13 (2001), 941–955.

[70] Matas, J., Chum, O., Urban, M., and Pajdla, T. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference* (2002), vol. 1, pp. 384–393.

[71] Matthews, I., Ishikawa, T., and Baker, S. The template update problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 6 (June 2004), 810 – 815.

[72] Mellor, J. P. Enhanced reality visualization in a surgical environment. Tech. rep., Cambridge, MA, USA, 1995.

[73] Mikolajczyk, K., and Schmid, C. An affine invariant interest point detector. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part I* (London, UK, 2002), Springer-Verlag, pp. 128–142.

[74] Mikolajczyk, K., and Schmid, C. Scale & affine invariant interest point detectors. In *International Journal of Computer Vision* (Hingham, MA, USA, 2004), vol. 60, Kluwer Academic Publishers, pp. 63–86.

[75] Mikolajczyk, K., and Schmid, C. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell. 27*, 10 (2005), 1615–1630.

[76] Mitchell, J., Brennan, C., and Card, D. Real-time image-space outlining for nonphotorealistic rendering, 2002.

[77] Molton, N. D., Davison, A. J., and Reid, I. D. Locally planar patch features for real-time structure from motion. In *Proc. British Machine Vision Conference* (Sep 2004), BMVC.

[78] Naimark, L., and Foxlin, E. Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In *ISMAR* (2002), pp. 27–36.

[79] Naimark, L., and Foxlin, E. Encoded led system for optical trackers. In *Proceedings of the Fourth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 150–153.

[80] Najafi, H., Genc, Y., and Navab, N. Fusion of 3d and appearance models for fast object detection and pose estimation. In *Lecture Notes in Computer Science, 7th Asian Conference on Computer Vision (ACCV)* (2006). traning phase and online-phase, statistical evaluation of the appearance.

[81] Nienhaus, M., and Döllner, J. Edge-enhancement - an algorithm for real-time non-photorealistic rendering. In *WSCG* (2003).

[82] Northrup, J. D., and Markosian, L. Artistic silhouettes: A hybrid approach. In *Proceedings of the First International Symposium on Non Photorealistic Animation and Rendering (NPAR) for Art and Entertainment* (June 2000).

[83] Oberkampf, D., DeMenthon, D. F., and Davis, L. S. Iterative pose estimation using coplanar feature points. *Comput. Vis. Image Underst. 63*, 3 (1996), 495–511.

[84] Pintaric, T. An adaptive thresholding algorithm for the augmented reality toolkit. In *Second IEEE Intl. Augmented Reality Toolkit Workshop* (2003).

[85] Platonov, J., Heibel, H., Meier, P., and Grollmann, B. A mobile markerless ar system for maintenance and repair. In *ISMAR* (2006), pp. 105–108.

[86] Platonov, J., and Langer, M. Automatic contour model creation out of polygonal cad models for markerless augmented reality. In *Proceedings of sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)* (Nara, Japan, Nov. 13–16 2007).

[87] Pupilli, M., and Calway, A. Real-time camera tracking using a particle filter. In *Proceedings of the British Machine Vision Conference* (September 2005), BMVA Press, pp. 519–528.

[88] Reitmayr, G., and Drummond, T. Initialisation for visual tracking in urban environments. In *Proceedings of sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)* (Nara, Japan, Nov. 13–16 2007), pp. 161–160.

[89] Reitmayr, G., and Drummond, T. W. Going out: Robust tracking for outdoor augmented reality. In *Proceedings of sixth IEEE and ACM International Symposium*

*on Mixed and Augmented Reality (ISMAR)* (Santa Barbara, CA, USA, October 22–25 2006), IEEE and ACM, IEEE CS, pp. 109–118.

[90] Ribo, M., Pinz, A., and Fuhrmann, A. A new optical tracking system for virtual and augmented reality applications. In *IEEE Instrumentation and Measurement Technology Conference* (2001).

[91] Rosten, E., and Drummond, T. Rapid rendering of apparent contours of implicit surfaces for realtime tracking. In *British Machine Vision Conference* (June 2003), pp. 719–728.

[92] Rosten, E., and Drummond, T. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision* (October 2005), vol. 2, pp. 1508–1511.

[93] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *European Conference on Computer Vision* (May 2006), vol. 1, pp. 430–443.

[94] Ruf, A., Tonko, M., Horaud, R. P., and Nagel, H.-H. Visual tracking of an end-effector by adaptive kinematic prediction. In *Proceeding of the International Conference on Intelligent Robots and Systems, IROS* (Grenoble, France, September 1997), vol. 2, IEEE/RSJ, pp. 893–898.

[95] Saito, T., and Takahashi, T. Comprehensible rendering of 3-d shapes. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1990), ACM Press, pp. 197–206.

[96] Schmid, C., and Mohr, R. Local greyvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 5 (1997), 530–535.

[97] Scott, D. W., and Szewczyk, W. F. From kernels to mixtures. In *Technometrics* (2001), vol. 43, pp. 323–335.

[98] Segvic, S., Remazeilles, A., and Chaumette, F. Enhancing the point feature tracker by adaptive modelling of the feature support. In *European Conf. on Computer Vision, ECCV'2006* (Graz, Austria, May 2006), LNCS.

[99] Shahrokni, A., Drummond, T., and Fua, P. Texture boundary detection for real-time tracking. In *European Conference on Computer Vision* (2004), vol. 3022, pp. 566–577.

[100] Shi, J., and Tomasi, C. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)* (1994), pp. 593 – 600.

[101] Shum, H.-Y., and Szeliski, R. Systems and experiment paper: Construction of panoramic image mosaics with global and local alignment. *International Journal of Computer Vision 36*, 2 (2000), 101–130.

[102] Simon, G., and Berger, M.-O. A two-stage robust statistical method for temporal registration from features of various type. In *Proceedings of the Sixth International Conference on Computer Vision (ICCV)* (Washington, DC, USA, 1998), IEEE Computer Society, p. 261.

[103] Smith, S. M., and Brady, J. M. SUSAN – A new approach to low level image processing. Tech. Rep. TR95SMS1c, Chertsey, Surrey, UK, 1995.

[104] Stauffer, C., and Grimson, W. E. L. Adaptive background mixture models for real-time tracking. In *CVPR* (1999), pp. 2246–2252.

[105] Stricker, D. Tracking with reference images: A real-time and markerless tracking solution for outdoor augmented reality applications. In *In Proc. of VAST* (2001).

[106] Stuelpnagel, J. On the parametrization of the three-dimensional rotation group. *SIAM Review 6*, 4 (1964), 422–430.

[107] Theobald, B.-J., Matthews, I., and Baker, S. Evaluating error functions for robust active appearance models. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition* (April 2006), pp. 149 – 154.

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

[109] Tommasini, T., Fusiello, A., Trucco, E., and Roberto, V. Making good features track better. In *CVPR* (1998), pp. 178–183.

[110] Tsin, Y., Genc, Y., Zhu, Y., and Ramesh, V. Learn to track edges. In *IEEE 11th International Conference on Computer Vision (ICCV)* (2007).

[111] Tukey, J. W. *Exploratory data analysis*. Addison-Wesley, Reading, MA., 1977.

[112] Vacchetti, L., Lepetit, V., and Fua, P. Combining edge and texture information for real-time accurate 3d camera tracking. In *Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR)* (November 2004).

[113] Vacchetti, L., Lepetit, V., and Fua, P. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence 26*, 10 (2004), 1391–1391.

[114] Wagner, D., and Schmalstieg, D. Artoolkitplus for pose tracking on mobile devices. In *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)* (2007).

[115] Welch, G., and Bishop, G. An introduction to the kalman filter. *ACM SIGGRAPH 2001 Course Notes* (2001).

[116] Welch, G., Bishop, G., Vicci, L., Brumback, S., Keller, K., and Colucci, D. The hiball tracker: high-performance wide-area tracking for virtual and augmented environments. In *Proceedings of the ACM symposium on Virtual reality software and technology (VRST)* (New York, NY, USA, 1999), ACM, pp. 1–ff.

[117] Williams, B., Klein, G., and Reid, I. Real-time SLAM relocalisation. In *Proc. International Conference on Computer Vision* (2007).

[118] Wuest, H., F.Vial, and Stricker, D. Adaptive line tracking with multiple hypotheses for augmented reality. In *ISMAR* (2005), pp. 62– 69.

[119] Wuest, H., Pagani, A., and Stricker, D. Feature management for efficient camera tracking. In *ACCV* (2007), pp. 769–778.

[120] Wuest, H., Wientapper, F., and Stricker, D. Adaptable model-based tracking using analysis-by-synthesis techniques. In *CAIP* (2007), W. G. Kropatsch, M. Kampel, and A. Hanbury, Eds., vol. 4673 of *Lecture Notes in Computer Science*, Springer, pp. 20–27.

[121] Zhang, K., and Kwok, J. Simplifying mixture models through function approximation. In *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman, Eds. MIT Press, Cambridge, MA, 2007.

[122] Zhang, Z. Parameter estimation techniques: a tutorial with application to conic fitting. *Image and Vision Computing Journal 15*, 1 (1997), 59–76.

[123] Zhu, G., Zhang, S., Chen, X., and Wang, C. Efficient illumination insensitive object tracking by normalized gradient matching. *Signal Processing Letters, IEEE 14*, 12 (December 2007), 944–947.

[124] Zinsser, T., Grässl, C., and Niemann, H. Efficient feature tracking for long video sequences. In *DAGM* (2004), pp. 326–333.

[125] Zisserman, A., Fitzgibbon, A., and Cross, G. Vhs to vrml: 3d graphical models from video sequences. *icmcs 01* (1999), 9051.

[126] Zivkovic, Z., and van der Heijden, F. Recursive unsupervised learning of finite mixture models. *IEEE Trans. Pattern Anal. Mach. Intell. 26*, 5 (2004), 651–656.

# Curriculum vitae

## Personal Information

| | |
|---|---|
| Name: | Harald Wuest |
| Address: | Wenckstr. 22, 64289 Darmstadt<br>Germany |
| Birthday: | March 23rd, 1978 in Spaichingen |

## Educational and Academic Background

| | |
|---|---|
| 03/2004 | Graduation at the University of Mannheim, Germany<br>Majors in computer graphics, image processing and pattern recognition |
| 09/2001 - 08/2002 | Exchange student at the University of Waterloo, Canada |
| 10/1998 - 03/2004 | Computer engineering at the University of Mannheim, Germany |
| 1988-97 | Secondary school Gymnasium Spaichingen<br>Abitur with majors in Mathematics and Physics |

## Work experience

| | |
|---|---|
| since 05/2004 | Research associate at *Fraunhofer IGD* in Darmstadt, Germany<br>Camera pose estimation for augmented reality applications |
| 03/2007 - 12/2007 | Research exchange at the *Centre for Advanced Media Technology (CamTech)*, Nanyang Technological University, Singapore<br>Development of Augmented Reality applications |
| 12/2002 - 02/2004 | Freelancer at *Volume Graphics* in Heidelberg, Germany<br>Programming and optimization of algorithms for voxel data processing |
| 09/2001 - 09/2002 | Tutor of a first-year student group in a lecture "Applied Computer Science II" at the University of Mannheim |
| 08/2000 - 11/2000 | Internship at *Fraunhofer CRCG (Center for Research in Computer Graphics)* in Providence, Rhode Island, USA<br>Development of a Java bytecode obfuscation framework |
| 04/1999 - 09/2001 | Student Assistant at the University of Mannheim<br>Programmer for the project ViPa (Virtual Patient Eye Surgery Simulation) |