
Graz University of Technology

Institute for Computer Graphics and Vision

Master's Thesis

---

# ONLINE MODEL-BASED MULTI-SCALE POSE ESTIMATION

---

**Thomas Kempter**

Graz, Austria, December 2011

*Thesis supervisors*

Univ.-Prof. Dipl.-Ing. Dr. techn. Horst Bischof

Dipl.-Ing. Andreas Wendel

All religions, arts and sciences are branches of the same tree.

*Albert Einstein*

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

## EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am …….  ……..
(Unterschrift)

Englische Fassung:

## STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

…….  ……..
date  (signature)

# Abstract

In this thesis we propose a novel model- and point-based pose estimation approach, which is able to operate on multiple scales in real time. We build our work on a state-of-the-art visual Simultaneous Localization And Mapping (*SLAM*) approach and extend it to exploit metric prior knowledge about the geometry of a single object in the scene. Using keypoint-based localization, which actually tracks the object by its surroundings, we are robust to occlusions and can even determine a pose when the object vanishes completely. Additionally, we refine this pose based on edge information to increase accuracy when the object occupies a certain amount of the image. In our experiments we show an improvement of the mean translational localization error compared to a state-of-the-art *SLAM* system from 6.1 cm to 1.7 cm for solid objects, and from 6.9 cm to 2.6 cm for wiry objects. Furthermore, when tracking is lost due to a lack of distinctive features, a purely model-based tracking component takes over. This reduces the number of frames for which the pose cannot be estimated by more than 20%. Our approach delivers a metrically correct pose estimate relative to a known object solely based on visual input from a single camera, which is useful for different robotic applications such as the autonomous inspection of a power pylon by an Unmanned Aerial Vehicle (*UAV*).

**Keywords:** Pose Estimation, SLAM, Model-Based Tracking, UAV, Inspection, Vision

# Kurzfassung

In dieser Abschlussarbeit präsentieren wir einen neuartigen modell- und punktbasierten Ansatz zur Positionsbestimmung, welcher in Echtzeit und für verschiedene Größenordnungen funktioniert. Wir bauen dabei auf einer aktuellen bildbasierten *Simultaneous Localization And Mapping (SLAM)* Methode auf und erweitern diese so, dass Vorwissen über die Geometrie eines einzelnen Objektes in der Szene ausgenutzt wird. Zunächst wird die Lage der Kamera relativ zum Objekt basierend auf markanten Punkten in der Szene geschätzt, wobei diese Punkte hauptsächlich im Hintergrund und nicht auf dem Objekt liegen. Dadurch sind wir in der Lage sowohl mit Verdeckungen umzugehen, als auch in kompletter Abwesenheit des Objektes eine Pose zu schätzen. Wenn das Objekt prominent im Bild zu sehen ist, werden die Kanteninformationen zusätzlich zum Verfeinern der Pose verwendet. In unseren Experimenten zeigen wir eine Verbesserung des mittleren translatorischen Fehlers im Vergleich zu einem aktuellen *SLAM* System von 6.1 cm auf 1.7 cm für solide Objekte und von 6.9 cm auf 2.6 cm für drahtige Objekte. Außerdem setzen wir auf eine rein modellbasierte Tracking-Komponente, die bei einem Verlust der Verfolgungsqualität auf Grund von zu wenigen markanten Punkten automatisch übernimmt. Dadurch wird die Zahl an Bildern für die keine Pose bestimmt werden kann um mehr als 20% gesenkt. Unser Ansatz liefert eine metrisch korrekte Positions- und Lageschätzung relativ zu einem bekannten Objekt, basierend auf Bildern einer einzelnen Kamera. Dies ist für verschiedene Anwendungen in der Robotik nützlich, wie zum Beispiel für die autonome Inspektion von Hochspannungsmasten durch ein *Unmanned Aerial Vehicle (UAV)* .

**Stichworte:** Posebestimmung, SLAM, Modell-Basiertes Tracking, UAV, Inspektion, Vision

# Acknowledgments

I would like to thank my family, my parents, Alfred and Emmi, my sister Theresa and my beloved significant other, Daniela Stefani. My academic studies would not have been possible without their great support.

Additionally I would like to gratefully acknowledge all my fellow-students and friends who have helped and supported me during my years of study, above all Mario Katusic and Michael Goller. I am also grateful to those students with whom I shared a room during the work on my thesis, for giving me so many constructive suggestions, including Gottfried Graber and Markus Murschitz.

Finally I would like to express my gratitude to my master's thesis advisors Univ.-Prof. Dipl.-Ing. Dr. techn. Horst Bischof and Dipl.-Ing. Andreas Wendel for their commitment and constructive suggestions. At Graz University of Technology I have had the fortune to work with some of the best researchers in computer graphics and vision, including my supervisors and Univ.-Prof. Dipl.-Ing. Dr. techn. Gerhard Reitmayr, who always helped me with several difficulties I had to face.

Thank you!

This work has been supported by the Austrian Research Promotion Agency (FFG) project FIT-IT Pegasus (825841/10397).

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

**Contents**



The field of robotics has been constantly growing over the past decades. Along with this development, computer vision and its applications have become increasingly important. Whether we think of rescue robots, service robotics, or robots used for inspection tasks, 2D- and 3D-visual data represents one of the most important sources of information describing the current state of the environment. Data acquired from vision systems is used for instance for navigation, grasping of objects, or quality assurance inspection. In future, an increasing number of applications will ultimately depend on visual input: *Cameras are a robot's eyes.*

## 1.1 Motivation

This work is part of the *Pegasus* project[1] at the Institute for Computer Graphics and Vision at Graz University of Technology which focuses on the autonomous inspection of overhead power lines using an Unmanned Aerial Vehicle (*UAV*). Today, such inspection tasks are performed with a helicopter. This is not only time-consuming and hence quite expensive, but furthermore dangerous for the crew itself, as the power lines to inspect cannot always be shutdown during inspection. Therefore, this task is predestined for autonomous inspection by a *UAV* without putting any risk on human beings.

[1] `http://aerial.icg.tugraz.at`

During autonomous inspection of overhead power lines several tasks depending on visual input have to be performed. First, a safe navigation within the vicinity of power lines involves a proper recognition of power pylons. This knowledge is then used during visual servoing to approach and inspect several points of interest such as single insulators. In this work we focus on the reliable determination of the exact location and orientation of a camera with respect to a known object in the scene. In turn, this allows accurate navigation of a robot; whether it comes on two wheels, is supplied with an omni-directional drive, or as in our case is airborne.

## 1.2 Goals and Contributions

Our goal is to be able to reliably and accurately determine a camera's pose with respect to a single yet known object with only images and detailed prior knowledge about the object's geometry being supplied. This can be used in future work to safely navigate within the vicinity of e.g. power pylons, when thinking of overhead power line inspection. The main contributions of this work are described in the following. Furthermore, a short outline of the structure of this work is given.

**Tracking Using Background Information.** Initial experiments have shown that it is difficult to estimate a pose for complex structures like power pylons from one single image. Due to the fact that a continuous image stream is available, we decided to use a tracking based approach. After an initialization step where the a-priori model information is used, the object is tracked by its surroundings. This includes recovering sparse 3D-information of the environment. As a result, determining the pose with respect to the object is not only possible over multiple scales, but as well when parts of the object are occluded, the object is only partially visible, or even vanishes completely.

**Pose Refinement Using Prior Knowledge.** As the geometry of one object in the scene is known, this information is not only used during initialization, but furthermore used permanently in a robust pose refinement during tracking, as long as the object is within the camera's view and occupies a significant area of the image. Especially when the camera is quite close to the object the refinement is able to improve accuracy. Additionally, refinement during initialization helps to ensure a good initialization, which is most essential in terms of accuracy.

**Failure Recovery Using Model Information.** When performing inspection tasks, closeup views of parts of the known object are sometimes needed. In such cases, tracking the object by its

surroundings often fails due to the lack of features. To successfully recover from such failures, model information is used to keep up tracking. We seamlessly switch back to a background-based tracking when visiting already explored areas.

**Capability of Sensor Fusion.** The current pose of the camera is supplied in real time at all times. This allows further integration into probabilistic frameworks, together with other sensor measurements like Global Positioning System (*GPS*) localizations or pose estimates delivered by an Inertial Measurement Unit (*IMU*).

The remainder of this work is organized as follows. In Chapter 2 the related work is discussed. Chapter 3 describes a general state-of-the-art framework for Visual Simultaneous Localization and Mapping, whereas our approach is presented in Chapter 4. Experiments and pose estimation results with respect to different known objects are then given in Chapter 5. Finally, Chapter 6 presents a summary and an outlook to possible future work.

# Chapter 2

# Related Work

**Contents**



The problem of determining the position and orientation of a 3D-object in space from one or more images is known as *pose estimation*. In literature, pose estimation is often referred to as *extrinsic camera calibration*, as the external parameters of the camera, namely the relative translation and orientation with respect to the 3D-object, are determined. Sometimes the term pose estimation is also used when the relative rotation and translation of one camera between two different views is desired. In photogrammetry pose estimation is often called space resection [1].

## 2.1 Overview

When talking about a *pose* or *extrinsic calibration* we always refer to the transformation between world and camera coordinate frame, as depicted in Figure 2.1.

The mapping between a 3D-world point $\mathbf{X}$ and the corresponding 2D-image point $\mathbf{x}$, assuming

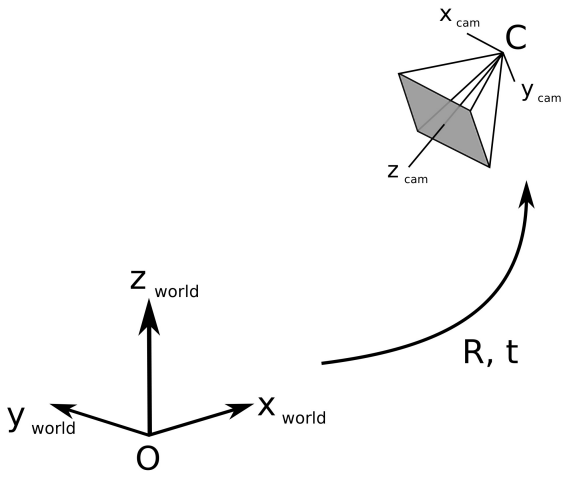



**Figure 2.1:** *The transformation between world and camera coordinate frame, consisting of* $\mathbf{R}$ *and* $\mathbf{t}$*, is shown. Usually this transformation is also referred to as pose or extrinsic camera calibration. The camera center* $\mathbf{C}$ *is the origin of the camera coordinate frame expressed in the world coordinate frame* $\mathbf{O}$*.*

homogeneous coordinates, is performed by the central projection

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} , \tag{2.1}$$

written in short matrix representation as

$$\mathbf{x} = \mathbf{P}\mathbf{X} , \tag{2.2}$$

where $\mathbf{P}$ is the $3 \times 4$ camera projection matrix. This matrix can be further broken down into

$$\mathbf{P} = \mathbf{K} [ \mathbf{R} \mid \mathbf{t} ] , \tag{2.3}$$

where $\mathbf{K}_{3\times3}$ is the camera calibration matrix, $\mathbf{R}_{3\times3}$ the rotation matrix, and $\mathbf{t}_{3\times1}$ the translation vector. The combination of $\mathbf{R}$ and $\mathbf{t}$ is the camera's *pose* or *extrinsic calibration* and has six degrees of freedom (*DOF*), where three *DOF* result from the rotation matrix and another three *DOF* from the translation vector. The camera center $\mathbf{C}$ in the world coordinate frame can be determined using

$$\mathbf{C} = -\mathbf{R}^T\mathbf{t} . \tag{2.4}$$

The camera calibration matrix

$$\mathbf{K} = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

is often referred to as *intrinsic calibration* and has five *DOF*. These are the focal lengths $f_x$ and $f_y$ of the camera in x- and y-direction, respectively, a skew factor $s$, which accounts for non perpendicular pixel elements, and the principal point offsets $p_x$ and $p_y$. All these parameters are determined during a camera calibration step [2, 3], using one of many available toolboxes like *MATLAB*'s camera calibration toolbox[1]. A detailed comparison of different camera models including their projective geometry can be found in [4].

One of the challenges when talking about pose estimation is the large six dimensional search space, as the pose is fully described by three *DOF* for its position in space, and another three *DOF* for the rotation angles around the x-, y-, and z-axis, respectively. The topic of pose estimation is almost as old as the area of computer vision, and hence a very extensively studied topic. Three different groups of algorithms are distinguished by means of their pose determination approach:

- **View- or appearance-based approaches [5–8].** For some time these approaches – especially the eigenspace approaches [5, 7, 8] – were in vogue due to their ability to handle the combined effects of shape, pose, reflections, partial occlusions, and illumination changes. These methods compare the query image to precomputed views of the object to determine its pose. The major drawback of such approaches is that they try to deal with the full six-dimensional search space by clustering the precomputed 2D-views of the object. Furthermore, due to comparing with precomputed views, these approaches are not suitable for non-solid objects such as power pylons.

- **Descriptor-based methods [9–13].** In this class of methods, artificial views of the object are generated and descriptors are extracted at salient image locations. Based on these descriptors, a classifier is trained. During the query phase, descriptors are extracted again at salient image locations, which are then classified accordingly in order to determine the object's pose. Despite their good performance in some applications, such approaches are limited to textured objects. As we are looking for a more general approach to determine the pose of objects with little or no texture, these methods are not suitable either.

- **Correspondence-based approaches [14–24].** These approaches use either points, lines, or other geometric primitives like triangles, and therefore circumvent the search in the large

[1] `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`

pose space. Yet, they have to deal with the search space resulting from establishing point correspondences if no prior knowledge is given. This search space can even grow bigger when additional features are introduced by clutter. However, this can be avoided by choosing more complex features such as lines, which reduces the search space on the one hand but makes the algorithm less robust to e.g. partial occlusions of the object.

For time critical applications like autonomous systems, with the aim on inspecting untextured objects such as power pylons, the focus is kept on feature-based approaches. In the remainder of this section an overview of related work regarding pose estimation with and without known correspondences is given.

## 2.2 Pose Estimation for Known Correspondences

In general, some prior knowledge like a few correspondences between 3D-model points $\mathbf{X}_i$ and 2D-image features $\mathbf{x}_i$ are needed for pose estimation. In other words, what we want to determine is the camera's projection matrix $\mathbf{P}$, which maps all $n$ corresponding 3D-model points to their image points as

$$\tilde{\mathbf{x}}_i \equiv \mathbf{P}\tilde{\mathbf{X}}_i \ , \tag{2.6}$$

where $\equiv$ denotes the equality up to an unknown scale factor. In other words, without further knowledge about the scene's metric geometry the pose is only estimated relatively. The $3 \times 4$ projection matrix $\mathbf{P}$ consists of the pose and the *intrinsic camera calibration parameters*, and generally has eleven *DOF*. Thus, for a minimal solution $n = 5\frac{1}{2}$ correspondences are needed, since every correspondence $\mathbf{X}_i \leftrightarrow \mathbf{x}_i$ rises two linearly independent equations in Eq. 2.6. The equation system $\mathbf{Ap} = \mathbf{0}$ has to be solved, where $\mathbf{A}$ is a $2n \times 12$ matrix. This method is also known as direct linear transformation (*DLT*) and has already been developed in 1963 during Sutherland's work on *Sketchpad* [25]. It was later introduced to the computer vision community by [4, 14]. When more than the minimal $n = 5\frac{1}{2}$ correspondences are given, the problem is over-determined and can be solved using Singular Value Decomposition (*SVD*) [4]. In this case an algebraic error is minimized with respect to further constraints, i.e. $||\mathbf{p}|| = 1$. The internal and external camera parameters can then be extracted from $\mathbf{P}$, however, in general *DLT* is not suitable for pose estimation as degenerate cases exist [4]. The Gold Standard algorithm is not to minimize an algebraic but a geometric error function. Thus, assuming a Gaussian noise distribution, the maximum likelihood estimation (*MLE*) [4] of $\mathbf{P}$ can be calculated by solving the minimization

problem

$$\min_{\mathbf{P}} \sum_i d\left(\mathbf{P}\mathbf{X}_i, \mathbf{x}_i\right)^2 \ , \tag{2.7}$$

where $d(\mathbf{a}, \mathbf{b})$ is the Euclidean distance between the image points $\mathbf{a}$ and $\mathbf{b}$. Equation 2.7 is then solved by an iterative minimization technique like Levenberg-Marquardt (*LM*) [4], which minimizes the sum of reprojection errors.

As a matter of fact, the complexity of estimating all eleven *DOF* of $\mathbf{P}$ can be reduced by simply calibrating the camera beforehand. By determining the five *DOF* of the intrinsic camera calibration matrix $\mathbf{K}$, another six *DOF* for the pose remain. Thus, only three correspondences are needed for a minimal solution of the pose problem, which is therefore referred to as the Perspective three Point problem (*P*3*P*), or in general Perspective $n$ Point problem (*PnP*) for overdetermined configurations. For the minimal configuration of three point correspondences up to four possible solutions exist, whereas for four or more correspondences the solution is unique. Six direct solutions to the *P*3*P* problem are given in [26], and their numerical stabilities are discussed. Several higher order polynomial equations are derived from three points and the resulting triangle configurations. The numerically most stable solution is that of Finsterwalder [15], which only requires finding the root of a cubic polynomial and the roots of two quadratic polynomials to solve the problem formulation. Recently Kneip et al. [27] proposed a novel parametrization of the *P*3*P*-problem. They introduced intermediate camera and world reference frames to solve the problem directly in a single stage. Again, a quartic equation with up to four solutions has to be solved, while a fourth point is used for disambiguation.

A very famous solution to the *PnP* problem requiring $n \geq 4$ non-coplanar points and a model of the object is *POSIT* [16]. The method approximates the perspective projection with a scaled orthographic projection and then calculates the pose by solving a set of linear equations using *POS* (Pose from Orthography and Scaling). The *ITerative* pose estimation is then continued using the last calculated pose. This allows *POS* to compute increasingly accurate scaled orthographic projections of the feature points in each iteration. *POSIT* only requires a single image, converges fast, is easy to implement, and does not require an initial guess like Newton-based methods. However, *POSIT* is quite sensitive to noise.

For $n \geq 5$ correspondences, [28] presented a linear method which is not degenerate for coplanar configurations. The method takes advantage of data redundancy and gives a unique solution. For $n$ points, $m = (n-1)(n-2)/2$ fourth order polynomials are derived, resulting in a set of linear homogeneous equations of the form $\mathbf{A}_{m\times n}\mathbf{t}_{n\times 1} = \mathbf{0}$, where $\mathbf{A}$ contains the coefficients of the fourth degree polynomial, and $\mathbf{t} = (1, x, x^2, ..., x^n)^T$ is a vector of the first $n$ powers of the

distance between one of the 3D-points and the camera center. Thus, $\mathbf{t}$ can be calculated using *SVD*, from which the pose can be determined subsequently.

Further closed-form solutions to the *PnP* problem, which neither require iteration nor a good initial guess, have been presented by Horn et al. [29, 30]. [29] uses unit quaternions [31] as rotation representation to simplify the solution, whereas [30] presents a solution where $3 \times 3$ orthonormal rotation matrices are used. Both solutions determine the translation as difference between the centroid of the coordinates in one system, and the rotated and scaled centroid of the coordinates in the other system.

Although the topic of estimating a pose is known for decades now, quite recent papers [17, 18, 32] deal with more efficient solutions to the *PnP* problem. Schweighofer and Pinz [17] investigated the pose ambiguities for planar targets viewed by a perspective camera. They showed that the instabilities in many applications occur due to two local minima of the error function used, however, the second minimum has almost been completely ignored. Only Oberkampf et al. [33] dealt with the second minimum in an extended version of *POSIT*, which handles coplanar points as well. However, Schweighofer and Pinz derived an analytical solution to locate the second minimum. They use the iterative pose estimation algorithm of [34] to solve for a first pose. Having a first possible solution, they reparametrize the error function and solve for a second minimum. The two possible pose estimates are then reevaluated, using each of them as an initial guess in [34], and retaining the solution with the smallest error.

An O($n$) solution to the *PnP* problem has been presented in [32]. This approach is of non-iterative nature and its complexity just grows linearly with the number of points. The central idea of the algorithm in contrast to most other approaches, which try to solve for the depths of the reference points in the camera coordinate frame, is to express the 3D-points $\mathbf{p}_i = \sum_{j=1}^{4} \alpha_{ij}\mathbf{c}_j$ with $\sum_{j=1}^{4} \alpha_{ij} = 1$ as a weighted sum of four virtual control points $\mathbf{c}_j$. Thus, the coordinates of the four control points become the unknowns of the problem, which can be determined by solving an equation system of the form $\mathbf{M}\mathbf{x} = \mathbf{0}$, where $\mathbf{x} = (\mathbf{c}_1^T, \mathbf{c}_2^T, \mathbf{c}_3^T, \mathbf{c}_4^T)^T$ is the 12-vector of the unknowns, and $\mathbf{M}$ a $2n \times 12$ matrix, which can be easily computed from 3D-to-2D correspondences. Thus, $\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i$ is expressed as weighted sum of the null eigenvectors $\mathbf{v}_i$ of $\mathbf{M}^T\mathbf{M}$, where a small constant number of quadratic equations has to be solved to pick the right weights $\beta_i$. To improve accuracy, the closed-form solution of [32] can be used as an initial guess for iterative Gauss-Newton minimization.

Schweighofer and Pinz presented a globally optimal O($n$) solution to the problem in [18]. Instead of limiting the algorithm to perspective cameras they use a general camera model formulation. Quaternions are used as representation for rotations and the object space costs are minimized by

transferring the problem into a semi-definite positive program. This transfer can be done in O($n$) time, whereas the solution of the semi-definite positive program is a global minimizer to the *PnP* problem and can be computed in a negligible amount of time for a large number of points.
All approaches presented require the knowledge of which 3D-point corresponds to which 2D-image feature. However, the question of how these correspondences have been established remains unanswered. Therefore, the next section is devoted to pose estimation for unknown correspondences, which is the far more general case.

## 2.3 Pose Estimation for Unknown Correspondences

The previous section presented solutions to the pose estimation problem for sets of known correspondences. However, if the correspondences are not given, the pose has to be estimated while simultaneously hypothesizing correspondences. Usually a small set of 3D-to-2D point correspondences is hypothesized randomly, and a pose is estimated using one of the previously discussed methods. Subsequently, the pose generated by these correspondences is evaluated using the remaining points. Therefore the 3D-points are projected to the image using the estimated pose and the nearest image feature is assigned as a corresponding 2D-point. However, standard least squares techniques usually applied to such problems fail in cases where large outliers are present and hence more robust methods need to be applied. The probably most widely used algorithm for robust estimation in the field of computer vision applications is RANdom SAmple Consensus (*RANSAC*) [19]. There, the geometric image distance between the projected 3D-point and its corresponding 2D-feature point is tested, and if the distance is below a threshold $t$, the correspondence is assumed to support the generated hypothesis. After several iterations of randomly sampling hypotheses and determining the set of inliers – also called consensus set – the largest set is taken to reestimate the pose. To guarantee with a high probability $p$ that at least one of the randomly drawn samples of $s$ points is free of outliers,

$$N = \left\lceil \frac{\log(1-p)}{\log(1-w^s)} \right\rceil \tag{2.8}$$

iterations have to be performed, where $w$ is the probability that any selected point is an inlier, and $\lceil \cdot \rceil$ the ceiling operator. Table 2.1 gives examples of the growth of iterations for $p = 0.99$ for given set sizes $s = 3 \dots 8$ and inlier probabilities $w = \{0.95, 0.90, 0.80, 0.75, 0.70, 0.60, 0.50, 0.40\}$. Due to the fast increasing number of iterations needed when more detailed models with more points are used, and additionally introduced features by clutter, *RANSAC* based approaches can become inapplicable for time

critical applications.

| **sample size** | **proportion of inliers** $w$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $s$ | 0.95 | 0.90 | 0.80 | 0.75 | 0.70 | 0.60 | 0.50 | 0.40 |
| 3 | 3 | 4 | 7 | 9 | 11 | 19 | 35 | 70 |
| 4 | 3 | 5 | 9 | 13 | 17 | 34 | 72 | 178 |
| 5 | 4 | 6 | 12 | 17 | 26 | 57 | 146 | 448 |
| 6 | 4 | 7 | 16 | 24 | 37 | 97 | 293 | 1123 |
| 7 | 4 | 8 | 20 | 33 | 54 | 163 | 588 | 2809 |
| 8 | 5 | 9 | 26 | 44 | 78 | 272 | 1177 | 7025 |

**Table 2.1:** *This table shows the number of iterations $N$ needed by RANSAC to guarantee with a probability of $p = 0.99$ that at least one of the randomly drawn samples of a set of $s$ points is free of outliers, while having an inlier probability of $w$. The number of iterations grows quickly when the inlier probability drops and/or the sample size increases.*

The probably best known and computationally very efficient algorithm when it comes to simultaneously estimating the pose and hypothesizing correspondences is *SoftPOSIT* [20]. *SoftPOSIT* combines the iterative *Softassign* algorithm [35, 36] for computing correspondences with the *POSIT* algorithm [16]. Unlike in *RANSAC* based approaches, all possible matches are treated identically throughout the search for an optimal pose. *SoftPOSIT* belongs to the group of model-based approaches as a 3D-point model of the object is needed to determine its pose in a single 2D-image. *SoftPOSIT* tries to estimate the correspondence matrix that matches 3D-model to 2D-image points and then computes the *MLE* of the unknown pose parameters. However, as *SoftPOSIT* is related to Expectation Maximization (*EM*) [37] algorithms, it converges to the minimum of a valley of the cost function. Subsequently, when *SoftPOSIT* should converge to the globally optimal pose a good initial guess is necessary. A commonly used technique to find the global optimum in *EM* algorithms is to run the algorithm from several randomly chosen initial guesses, however, many such guesses would be necessary in the large six dimensional pose parameter space. A further drawback of the method is its sensitiveness to noise, occlusions, and repetitive patterns.

Therefore, David et al. [21] extended *SoftPOSIT* to the case of line features. Lines are supposed to be more stable, less sensitive to partial occlusions, and are less likely produced by clutter and noise. However, David et al. use the line features only to determine the distances associated to point features, which are then used to initialize the correspondence matrix. Thus, the lines are only used for correspondence determination after which the normal *SoftPOSIT* operates on point features again. Yet, when carefully implemented, the runtime of *SoftPOSIT* can be retained.

Another suitable approach to the simultaneous solution of correspondence and pose is

*BlindPnP* [22]. *BlindPnP* claims to overcome the limitations of *SoftPOSIT* by taking advantage of some a-priori information on the camera pose. This knowledge is modeled as a Gaussian Mixture Model (*GMM*) that is refined progressively when correspondences are hypothesized, which considerably speeds up the exploration of the pose space. With each assigned correspondence the pose and its covariance are updated using standard Kalman filter equations. Simultaneously, the pose covariance defines search regions for further possible correspondences, which therefore become tighter with each correspondence assigned. After three correspondences have been chosen the uncertainties are so small that additional correspondences can be assigned easily, yielding an initial solution for [32]. In order to find the best pose, small sets of three correspondences are generated and the pose with the smallest reprojection error and maximum number of correspondences is retained. However, the main problem using *BlindPnP* is to detect 2D-image features which really correspond to a 3D-model point. Furthermore, *BlindPnP* is not intended to run in real time, and therefore is not suitable for online applications as would be needed by an autonomous robot.

Another model-based approach solely based on a Computer Aided Design (*CAD*) model of the object and a single image is presented in [38]. The view-based approach combines a pyramid search with a hierarchy of object views, arranged in a precomputed tree structure. Thus, the pose can be estimated in a short time, while exploiting the robustness of an exhaustive search. First, the model lines are subsampled and the resulting points located with subpixel precision by searching perpendicular to the model edges for edges in the image. Then, the pose is calculated by least-squares minimization using *LM*. The approach claims to be robust to noise, occlusions, clutter, and background changes. However, the major limitation of this approach are degenerated views during training, i.e. side views of almost planar objects, which would be detected at each pair of parallel lines.

Grimson et al. [39] search an interpretation tree of corresponding model and image features. This algorithm, however, is a combinatorial problem, which sometimes gets prohibitively expensive to compute. Therefore, often additional constraints are added to prune the search tree [40].

Holzer et al. [41] presented a pose estimation algorithm for planar objects only, based on edges. They extract normalized distance transform templates by applying distance transform [42] to edge images. Then, closed contours are extracted, small gaps are removed, and the templates are created by transforming them to a canonical scale and orientation. The patch is used to train a classifier in order to gain robustness against perspective transformations. During training the identity and pose of the templates is learned and spatial relations between contours are introduced for outlier rejection. Once the classifier has been trained, it retrieves template identity and pose simultaneously.

The pose is finally refined similar to Lucas-Kanade's template tracking [43] algorithm, however, instead of intensity images they use distance transform maps to align a template image to an input image. As the alignment is a homography, the 3D-pose can be computed using homography decomposition [44].

## 2.4 Pose Estimation for Image Sequences

Often, pose determination is needed not only for single images but for a continuous sequence of images. Initialization of such pose estimation algorithms is often difficult, however, essential to retrieve accurate results. Therefore, many algorithms use some kind of tracking approach with manual or semi-automatic initialization. When a first pose has been estimated, the pose can be refined frame by frame, while the algorithms are given a good initial guess using the pose of the last frame to solve for the new one. During tracking sometimes the 3D-scene structure is determined. Usually, the problem is then referred to as Structure from Motion (*SfM*) or Simultaneous Localization And Mapping (*SLAM*). Visual SLAM (*VSLAM*) describes the group of *SLAM* algorithms, where only images are used as sensor input.

A popular real-time *VSLAM* implementation is that of Davison [45]. He was the first to propose a real-time capable, single-camera localization by creating a map of sparse natural features. The Bayesian-based framework allows handling of uncertainties in a way that robust localization is possible. Based on this approach Davison et al. proposed *MonoSLAM* [46]. They extend the previously proposed method of monocular feature initialization by a feature orientation estimation. This successfully increases the range in which landmarks are detected and hence improves tracking. However, *MonoSLAM* only works for slow camera motions or in combination with further sensor inputs such as wheel-odometry.

Bleser et al. [23] proposed a robust real-time camera pose estimation approach for partially known scenes. The connection between the real and the virtual world is made by one known object, given as a *CAD*-model. However, the model is only used during initialization, which is semi-automatic: The model is projected to the image using a predefined pose, which the user then has to register manually to the image by moving the camera. When the projected model is close enough to register the model lines to the image gradient, initialization completes automatically. Then, point-based tracking is applied to estimate the pose using warped patches around extracted feature locations. 3D-features are initialized using triangulation, which are then refined during tracking using an Extended Kalman Filter (*EKF*) [47]. Due to uncertainty, statistical methods are incorporated, which are then used to calculate the weights during Weighted Least-Squares

(*WLS*) [4] pose estimation. The pose of the previous frame is used as initialization, whereas the refinement is implemented as *RANSAC* procedure to gain robustness.

Others [48, 49] use model-based pose estimation together with gyroscope measurements to be able to cope with fast motions and resulting motion blur. While [48] is used for Augmented Reality (*AR*) in head-mounted displays, [49] is used for outdoor *AR* and extends the former work of Klein by the usage of a texture-based model and online edge extraction. Both approaches use the edge-based tracking algorithm by Drummond and Cipolla [50].

Nistér et al. [51] presented a system that estimates motion trajectories using video input only. Point features are used to robustly estimate camera motion in real time, without any prior knowledge about neither scene nor motion. Accurate results have been achieved for sequences of several hundred meters using their *Visual Odometry* algorithm.

Recently [52] investigated the tracking of rigid objects using the integration of model-based and model-free cues in an iterated *EKF*. The model-based cue uses a wireframe-edge model of the object, whereas, the model-free cue uses automatically generated surface texture features. Thus, the approach relies on both model and texture information. The model-based tracking is similar to [50, 53], which estimates the normal flow for points along edges. The normal flow is then projected in the direction of the contour normal so that the error can be minimized using robust *M-Estimators* [54]. For the model-free cue, Harris corners are extracted and tracked for visible parts of the object by minimizing the Sum of Squared Differences (*SSD*) in RGB values. The algorithm is able to deal not only with polyhedral objects, but also with spherical, cylindrical and conical objects. However, this approach makes strong usage of texture information available on the object itself, and hence is not capable of dealing with non-solid objects such as power pylons.

Parallel Tracking And Mapping (*PTAM*) [24] is the current state-of-the-art *SLAM* system. *PTAM* is able to estimate the camera pose in a formerly unknown scene. It splits tracking and mapping into two separate tasks, which are processed in parallel. Thus, global optimization techniques can be applied, resulting in an accurate and real-time capable tracking system. *PTAM* is even capable of dealing with reasonably fast camera motions without being given further sensor measurements.

## 2.5 Summary

Several algorithms for pose estimation with, and without given correspondences have been discussed. As our work aims towards an autonomous power line inspection using a *UAV*, only real-time capable approaches are applicable. As a continuous video stream will be available,

*SLAM*-based approaches are most suitable: After an initial map has been created, a system based on *SLAM* can perform iterative pose updates in real time, while simultaneously expanding the map to be able to keep track of the camera. For this reason, our work is focused on *VSLAM* systems. This allows a platform-independent development of the algorithm supplying a vision-based pose estimate, which in future applications can be fused with other sensor measurements. As *PTAM* [24] meets all these requirements, it builds the basis of this work and is therefore discussed in detail in Chapter 3.

# Chapter 3

# Parallel Tracking and Mapping

**Contents**



Parallel Tracking and Mapping (*PTAM*) belongs to the group of Simultaneous Localization and Mapping (*SLAM*) algorithms. *SLAM* is a common problem in the field of robotics, as well as in computer vision tasks. It consists of two parts, namely the localization task and the mapping task. Each of them can be solved quite straight-forward if the other one has already been solved beforehand. In other words, it is possible to localize a robot within an already known environment on the one hand, and on the other hand a map can be built if the robot's position can be determined accurately at all times. However, the combination of *simultaneously* solving both problems has long been seen as a 'holy grail' for the mobile robotics community. Up to now, a lot of research has focused on this topic and considerable progress has been made in this field [55, 56]. The solution of the *SLAM* problem is the key towards a fully autonomous robot.

## 3.1 Overview

Many different *SLAM* implementations exist, depending on the field of application and the robot used. However, when talking about *Visual SLAM*, only camera images are used for both the localization and the mapping task. A *SLAM* system operates in an endless-loop, always trying to keep

track of the actual location of the robot, while simultaneously updating the map, as schematically depicted in Figure 3.1.

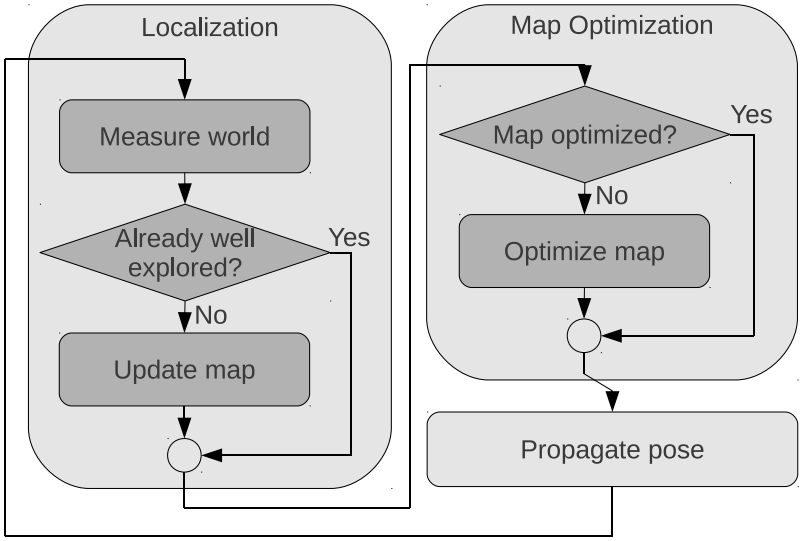



**Figure 3.1:** *A SLAM system needs a component which recognizes places already visited in order to localize itself and update the map. Furthermore, a global map optimization has to be applied due to noisy and inaccurate measurements. A good strategy to estimate the pose in the next iteration is needed to perform a fast iterative pose update during localization.*

Therefore, the main components of a good *SLAM* system are

- recognition of places already visited (localization),
- a map optimization technique, and
- a local motion estimation for pose propagation.

Each of these components is depicted as a light gray box in Figure 3.1.

The current state-of-the-art monocular real-time *VSLAM* implementation is Klein and Murray's *Parallel Tracking And Mapping* (*PTAM*) [24]. The core idea of the algorithm is to split tracking and mapping into two separate tasks executed in parallel. As a result, any robust tracking method could be used, however, *PTAM* uses a fast coarse-to-fine tracking approach. The mapping in *PTAM* is based on *keyframes* and the world is represented as a sparse 3D-point cloud. Whenever a formerly unknown area is explored, a new keyframe is created and new 3D-points are calculated using triangulation with the closest nearby keyframe. The big benefit of *PTAM* is the sophisticated optimization technique of Bundle Adjustment (*BA*) applied in the mapping thread. *PTAM* is accurate and fast; it is even able to run in real time on hardware like todays mobile phones [57].

Summing up, *PTAM* meets all necessary requirements of a good *SLAM* system. *PTAM* was originally developed as an *AR* application to overlay rendered content in the real world. Hence, the position and orientation of the camera with respect to the world coordinate frame has to be determined frame by frame. *PTAM* belongs to the group of monocular *VSLAM* approaches where only one single camera is used. Figure 3.2 finally shows a few screen-shots of *PTAM* in action.

## 3.2 Initial Pose Estimation and Map Construction

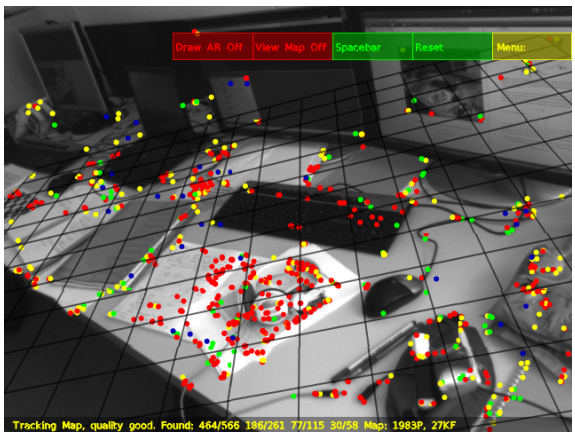
*PTAM* is able to generate and track a map from scratch without being given any prior knowledge about the scene. Only little user interaction is required to initialize the algorithm. Assuming a calibrated camera the following steps are needed to initialize *PTAM*:

- The user points the camera on a planar but well structured scene and manually initiates the start of salient image point tracking. The first frame is used as the first keyframe.
- While translating the camera sideways the system tries to track the feature points frame by frame, as depicted by Figure 3.2(a).
- The user manually initiates the end of the feature tracking process. This frame then marks the second keyframe. The two keyframes build a pair of stereo-images.
- The tracked correspondences are now used to estimate the relative camera motion up to scale. The baseline between the two cameras is then scaled to a fixed metric distance.
- Finally, 3D-map points are triangulated using the two views (see 3.2(c)). The dominant plane in the scene is determined by fitting a plane to the reconstructed points. The whole map is finally rotated in a way that the detected plane lies in the world's x-y-plane, drawn as a grid in Figure 3.2(b).

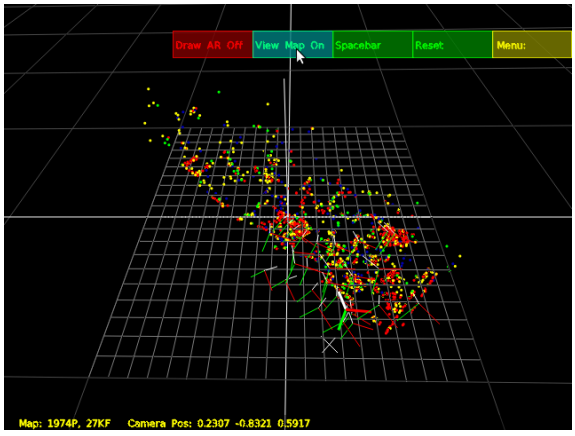
After these few steps, which only take a few seconds, *PTAM* starts tracking the camera and when needed extends the map. In the following some more details about the initialization steps are given, including important concepts of multiple view geometry [4].

**Features.** Many different features like points, lines or blobs can mark an *interesting* image location. To name just a few of the most popular detectors, there is the Harris corner detector [58], which analyzes the second-moment matrix or structure-tensor. Shi-Tomasi corner detection [59]

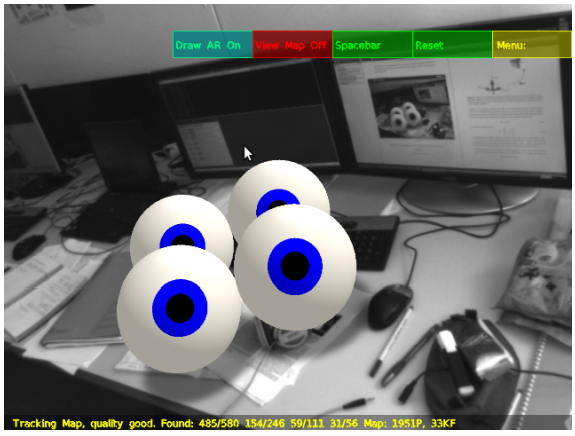

**(a)** During initialization features are tracked frame by frame. Successfully tracked features are shown as lines connecting the initial and current location of each feature.




**(b)** After initialization, the successfully tracked feature points are shown as dots. The dominant plane is drawn as a grid.




**(c)** The 3D-map points are shown as dots. The positions of the cameras, where keyframes have been dropped, are drawn as coordinate frames. The bold keyframe marks the current camera. This map has a total of approximately 2000 points and 30 keyframes.




**(d)** A possible *AR* application: Four eyeballs facing the camera are overlaid to the real world.

**Figure 3.2:** *A typical application of PTAM is the tracking of small environments like the desktop scene shown in (b). Figure (a) shows the feature tracking, (c) illustrates the 3D-map corresponding to the shown scene, and in (d) an AR model is overlaid to the scene.*

is similar to the Harris corner detector, however, Shi-Tomasi calculate the score of a possible corner by taking the minimal eigenvalue of the structure tensor, whereas Harris suggested to use the matrix's determinant and trace to avoid eigenvalue decomposition. In general the criterion of Shi-Tomasi outperforms the one suggested by Harris. Furthermore, there exists Scale Invariant Feature Transform (*SIFT*) [60], which actually uses Difference of Gaussian (*DoG*) to detect scale-space extrema, and Smallest Univalue Segment Assimilating Nucleus (*SUSAN*) [61], where a circular mask is placed over the pixel to be tested and every pixel within this nucleus is weighted according to a comparison function. *SUSAN* can be used either as edge or corner detector, where a corner is as far as possible away from the nucleus. An overview over region detectors is given in [62], whereas Maximally Stable Extremal Region (*MSER*) [63] often perform best. *MSER*s are detected by thresholding an image at all possible gray levels and monitoring regions. Those regions most stable with respect to a thresholding operation are then marked as *maximally stable*. However, *PTAM* uses a quite recently proposed feature detector called Features from Accelerated Segment Test (*FAST*) [64, 65]. *FAST* is a high-speed corner detector similar to *SUSAN* and hence very suitable for real-time applications like *PTAM*. Pixels in a Bresenham-circle around candidate points are tested by comparing their gray-value intensities as depicted by Figure 3.3. If $n$ pixels are all brighter or darker than the center pixel by a threshold $t$, the candidate point is considered to be a corner.

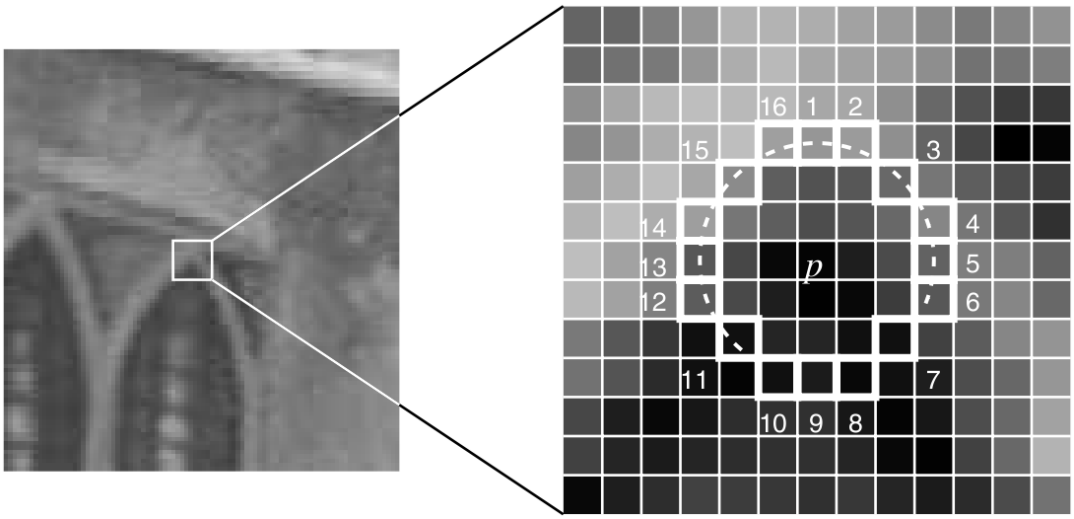



**Figure 3.3:** *FAST uses a technique called Accelerated Segment Test to determine a feature point. Gray-value intensities of pixels in a Bresenham-circle around the candidate point $p$ are compared. If $n$ neighboring pixels are all brighter or darker than the center pixel by a threshold $t$ the point is considered a feature. A circle radius of $r = 3$ pixels yields 16 Bresenham-circle pixels used for testing. Here the dashed line passes through twelve of the 16 pixels, which are all brighter than $p$. Image taken from [65].*

**Frame by Frame Feature Tracking.** When the first keyframe is stored, *FAST* features are extracted. For each feature the Shi-Tomasi score is calculated and the $N = 1000$ most salient *FAST* corners are tracked. At each salient location a small image patch $I_0(\mathbf{x})$ is extracted. Each patch is then locally searched for in the current frame $I_1$, starting at the location it has been found in the last frame. If and where the patch has been found is determined by evaluating the *SSD*

$$E_{SSD}(\mathbf{u}) = \sum_{i \in I_{0,1}} [I_1(\mathbf{x}_i + \mathbf{u}) - I_0(\mathbf{x}_i)]^2 \tag{3.1}$$

at all possible displacements $\mathbf{u} = (u, v)$ within a predefined window of fixed size. The difference between the compared patches is often referred to as *residual error*. When the patch has been found an additional criterion has to be met. At the location where the *SSD* had its maximum response another patch is extracted, which is then searched for in the last frame. Only if the backwards matching yields a location close to the starting location of the forward matching, the patch has successfully been tracked.

**Estimating Relative Camera Motion.** After tracking the most salient feature points enough correspondences between the images of the first and second keyframe hopefully remain. Using these correspondences the relative motion between the two cameras can be determined by exploiting epipolar geometry constraints. Epipolar geometry is the intrinsic projective geometry between two views and only depends on the internal parameters of the cameras and their relative pose. This mathematical relation is fully described by the fundamental Matrix $\mathbf{F}_{3\times3}$ of rank two, which maps an image point $\mathbf{x}$ from one view to it's corresponding epipolar line

$$\mathbf{l}' = \mathbf{F}\mathbf{x} \tag{3.2}$$

in the second view. When two points $\mathbf{x} \leftrightarrow \mathbf{x}'$ truly correspond, they must satisfy the epipolar constraint

$$\mathbf{x}'^{\mathbf{T}}\mathbf{F}\mathbf{x} = 0\ . \tag{3.3}$$

Figure 3.4 illustrates the epipolar geometry. When the intrinsic camera calibration matrix $\mathbf{K}$ is known, any image point $\mathbf{x} = \mathbf{P}\mathbf{X}$ can be transferred to *normalized camera coordinates* by applying $\mathbf{K}^{-1}$ to the image point $\mathbf{x}$, resulting in

$$\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x}\ , \tag{3.4}$$

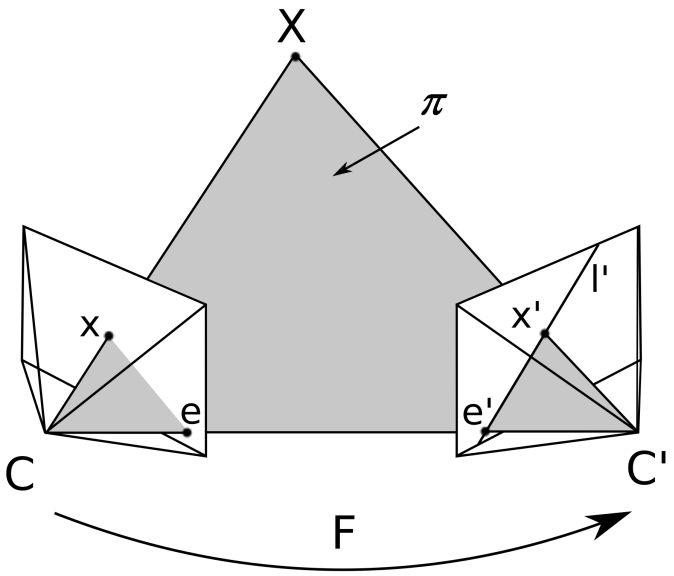



**Figure 3.4:** *The epipolar geometry is the intrinsic projective geometry between two views. A 3D-world point* $\mathbf{X}$ *is imaged as* $\mathbf{x}$ *in the first and* $\mathbf{x}'$ *in the second view, respectively. The two image points are related via the* fundamental matrix $\mathbf{F}$*, which maps a point from one view onto a line in the second view. The connection between the two camera centers* $\mathbf{C}$ *and* $\mathbf{C}'$ *is called baseline. The intersection of the baseline with the image planes of the two cameras defines the epipoles* $\mathbf{e}$ *and* $\mathbf{e}'$*, respectively. Any plane* $\pi$ *containing the baseline is an epipolar plane, which intersects the image planes in the epipolar lines, like e.g.* $\mathbf{l}'$ *for the second view.*

where $\hat{\mathbf{x}}$ is now expressed in normalized camera coordinates. In the case of normalized camera coordinates there exists a specialization of the fundamental matrix called the *essential matrix* $\mathbf{E}$. The essential matrix has fewer *DOF* compared to the fundamental matrix and is defined by

$$\mathbf{E} = [\mathbf{t}]_{\times}\mathbf{R} \,, \tag{3.5}$$

assuming a pair of normalized camera matrices $\mathbf{P} = [\mathbf{I} \mid \mathbf{0}]$ and $\mathbf{P}' = [\mathbf{R} \mid \mathbf{t}]$, respectively. When $\mathbf{t} = (t_1, t_2, t_3)^T$ is a three-vector,

$$[\mathbf{t}]_{\times} = \begin{bmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix} \tag{3.6}$$

generates a $3 \times 3$ skew-symmetric matrix. The $[\,\cdot\,]_{\times}$-operator is used to express a cross product in terms of a matrix multiplication.

The essential matrix can be directly computed using the five-point algorithm and *RANSAC* [19],

or from the fundamental matrix, if known, using

$$\mathbf{E} = \mathbf{K'}^T \mathbf{F} \mathbf{K} \, . \tag{3.7}$$

When the essential matrix has been computed, the relative motion between the two cameras can be calculated up to an unknown scale using *SVD* [4]. This yields up to four possible solutions as described in [4], however, only one solution is possible as only in one case a reconstructed 3D-point lies in front of both cameras.

*PTAM*, however, uses an approach to the decomposition of $\mathbf{E}$ similar to Faugeras and Lustman [44], where under the assumption of a piecewise planar environment the tracked point correspondences are used to estimate a homography $\mathbf{H}_{3\times3}$ between the two images. $\mathbf{H}$ is further decomposed into $\mathbf{R}$ and $\mathbf{t}$ using *SVD* under the assumptions of $h_{33} = 1$ and $|R| = 1$, with $|\cdot|$ being the determinant of the matrix. However, the solution is only defined up to scale and hence not metric, so *PTAM* assumes a baseline of ten centimeters between the two cameras and scales $\mathbf{t}$ accordingly.

**Triangulating 3D-Points.** Triangulation is the process of finding the 3D-point corresponding to several image measurements. As the motion between the two cameras has been determined, 3D-map points can be triangulated from the known point correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'$. Due to the fact that in reality the measured correspondences are noisy, they do not necessarily satisfy the epipolar constraint of Equation 3.3. Assuming a Gaussian noise model the *reprojection error* for a 3D-world point $\hat{\mathbf{X}}$, which is mapped to the image points $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ with projection matrices $\mathbf{P}$ and $\mathbf{P}'$ corresponding to $\mathbf{F}$, is minimized. $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ are then the *maximum likelihood estimates* of $\mathbf{x}$ and $\mathbf{x}'$, respectively. Now a linear triangulation method can be applied by combining $\hat{\mathbf{x}} = \mathbf{P}\hat{\mathbf{X}}$ and $\hat{\mathbf{x}}' = \mathbf{P}'\hat{\mathbf{X}}$ into an equation of the form $\mathbf{A}\hat{\mathbf{X}} = \mathbf{0}$. This equation system can now either be solved homogeneously using *SVD*, or as a set of inhomogeneous equations, where homogeneous coordinates are used, using a least-squares technique. A detailed description of optimal triangulation methods is given in [4], while the process of triangulating a 3D-point from image correspondences is depicted in Figure 3.5.

## 3.3 Iterative Pose and Map Update

*PTAM* is able to estimate the camera's pose with respect to the world coordinate frame by point-based *tracking*. After initialization a map is created, containing 3D-world points which have been determined using triangulation of corresponding points in two views. *PTAM* now estimates the

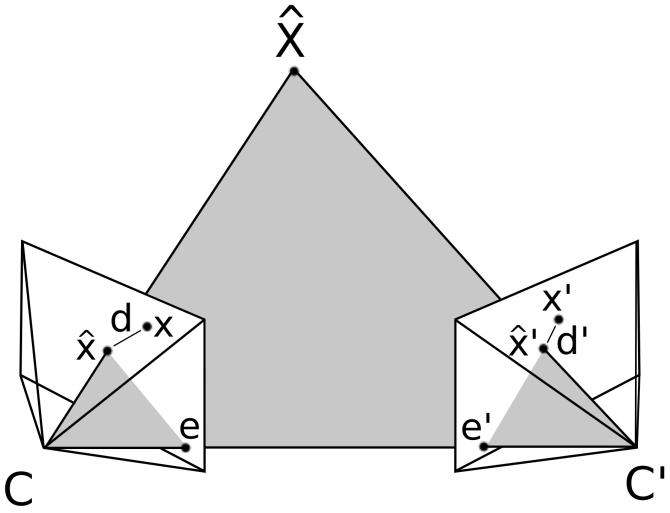



**Figure 3.5:** *The 3D-point $\hat{\mathbf{X}}$ is imaged as $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ in $\mathbf{C}$ and $\mathbf{C}'$, respectively. $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}'$ are determined by minimizing the reprojection error $d^2 + d'^2$ with respect to the epipolar constraint $\hat{\mathbf{x}}'^T \mathbf{F} \hat{\mathbf{x}} = 0$, starting from the noisy correspondences $\mathbf{x} \leftrightarrow \mathbf{x}'$ measured. As an outcome the 3D-point $\hat{\mathbf{X}}$ can be reconstructed from two noisy image correspondences by minimizing the geometric error.*

camera's position in real time using a two-stage, coarse-to-fine tracking procedure, while simultaneously expanding the map if unexplored regions are visited. In such a case, a new keyframe is inserted into the map and new 3D-points are triangulated using this keyframe and its nearest neighboring keyframe. Before we continue, the mathematical terminology used for pose representation is explained.

### 3.3.1 Pose Representation

Throughout the rest of this work a pose or rigid body transformation, consisting of a rotation and a translation, are encoded in a $4 \times 4$ matrix $\mathbf{E}$ of the Special Euclidean Lie Group *SE(3)* [66]. The six-dimensional *SE(3)* Lie Group contains the set of all 3D rigid-body transformations in $\mathbb{R}^3$. A point $\mathbf{p}$ in the world-coordinate frame $\mathcal{W}$ is then transformed to the camera-coordinate frame $\mathcal{C}$ by

$$\mathbf{p}_{\mathcal{C}} = \mathbf{E}_{\mathcal{C}\mathcal{W}} \, \mathbf{p}_{\mathcal{W}} \, , \tag{3.8}$$

where $\mathbf{E}_{\mathcal{C}\mathcal{W}}$ represents the camera's pose as a member of *SE(3)* and takes the form

$$\mathbf{E}_{\mathcal{C}\mathcal{W}} = \begin{bmatrix} & \mathbf{R}_{3\times3} & & \mathbf{t}_{3\times1} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{3.9}$$

$\mathbf{E}$ may be parameterized by a six-dimensional motion vector $\boldsymbol{\mu}$ via the exponential map, where $\mu_1$, $\mu_2$ and $\mu_3$ represent translation along $x$, $y$ and $z$ axes, and $\mu_4$, $\mu_5$ and $\mu_6$ describe the rotation around these axes, respectively. An *SE(3)* matrix $\mathbf{E}$ is calculated from a six-dimensional motion vector $\boldsymbol{\mu}$ by

$$\mathbf{E} = \exp\left(\sum_{i=1}^{6} \mu_i \mathbf{G}_i\right) \tag{3.10}$$

where $\mathbf{G}_i$ are called generator matrices, which take the values

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix},$$
$$\mathbf{G}_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{3.11}$$

Vice versa, a six-dimensional motion vector can be extracted from an *SE(3)* matrix using the logarithm operation

$$\boldsymbol{\mu} = \ln(\mathbf{E}) \ . \tag{3.12}$$

All operations on the *SE(3)* group are continuous and smooth, and hence differentiable in closed form, which is essential for tracking applications. Differentiating a motion matrix $\mathbf{E}$ at the origin $\boldsymbol{\mu} = \mathbf{0}$, the partial derivatives

$$\frac{\partial \mathbf{E}}{\partial \boldsymbol{\mu}_i} = \mathbf{G}_i \tag{3.13}$$

are simply the generator matrices.

### 3.3.2 Two-Stage Coarse-To-Fine Tracking

To continue with the pose estimation we now describe all steps of the two-stage tracking procedure stated in [24] and give some more details regarding the single steps:

1. **A new frame is acquired from the camera and a prior pose estimate is generated from a motion model.** During this step *PTAM* generates a four-level image pyramid, which makes the algorithm able to cope with *multiple scales*. This is an essential property in terms of our goal of *multi-scale pose estimation*. At level zero the full resolution image is stored, whereas the levels below contain sub-sampled images of half the resolution of the image of the higher level. On each level *FAST* corners are extracted as depicted in Figure 3.6.

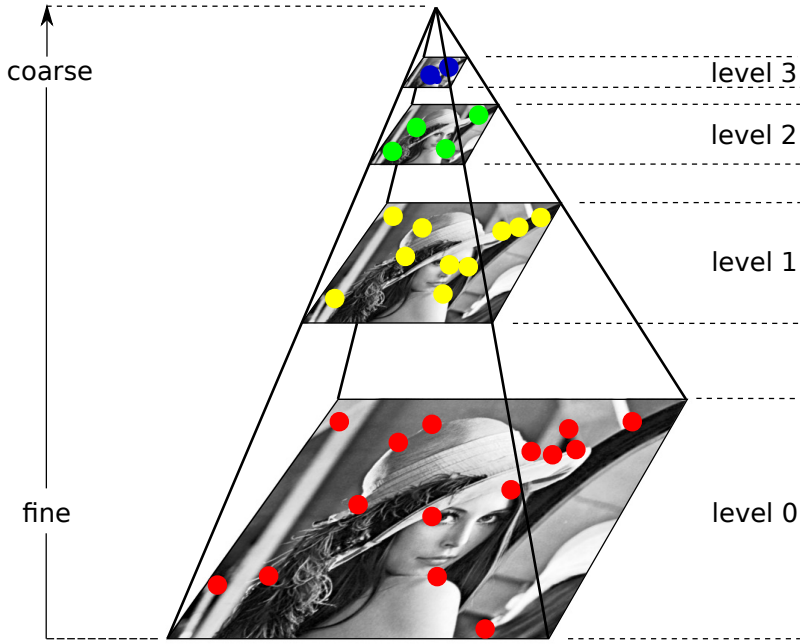



**Figure 3.6:** *For each new frame a four level image-pyramid is generated, used in the coarse-to-fine tracking approach. Each level has half the resolution and a quarter the pixels of it's parent level, starting with the original image at level zero. Furthermore, FAST corners are extracted on each level, illustrated as colored dots.*

Additionally, a decaying velocity model in form of a six-dimensional vector

$$\mathbf{v}_t = \tau \left(0.5 \ln \left(\mathbf{E}_{\mathcal{CW}} (\mathbf{E}^-_{\mathcal{CW}})^{-1}\right) + 0.5\, \mathbf{v}_{t-1}\right) \tag{3.14}$$

with $\tau < 1$ is used to propagate $\mathbf{E}_{\mathcal{CW}}$, the actual pose of the camera, forward, while a one-frame-per-second camera is assumed. This is done by applying the motion model from Equation 3.14 to $\mathbf{E}^-_{\mathcal{CW}}$, which denotes the pose of the camera at the beginning of the frame. The predicted pose of the camera in the next frame can be calculated by transforming the prior pose using the motion vector $\mathbf{v}_t$, providing a posterior pose estimate

$$\mathbf{E}^+_{\mathcal{CW}} = \exp\left(\mathbf{v}_t\right) \mathbf{E}^-_{\mathcal{CW}}\,. \tag{3.15}$$

2. **Map points are projected into the image according to the frame's prior pose estimate.** Using the camera's estimated pose $\mathbf{E}^+_{\mathcal{CW}}$ determined by Equation 3.15, all currently available map points are projected to the image. *PTAM* uses a field of view (*FOV*) camera model

with a single distortion parameter [67]:

$$\begin{aligned}\begin{pmatrix} u_j \\ v_j \end{pmatrix} &= \text{CamProj}\left(\mathbf{E}^{+}_{\mathcal{CW}}\mathbf{p}_{j\mathcal{W}}\right) \\ &= \text{CamProj}\begin{pmatrix} X_{\mathcal{C}} \\ Y_{\mathcal{C}} \\ Z_{\mathcal{C}} \end{pmatrix} \\ &= \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \frac{r'}{r} \begin{pmatrix} \frac{X_{\mathcal{C}}}{Z_{\mathcal{C}}} \\ \frac{Y_{\mathcal{C}}}{Z_{\mathcal{C}}} \end{pmatrix} ,\end{aligned} \tag{3.16}$$

where $r = \sqrt{\left(\frac{X_{\mathcal{C}}}{Z_{\mathcal{C}}}\right)^2 + \left(\frac{Y_{\mathcal{C}}}{Z_{\mathcal{C}}}\right)^2}$, and $r' = \frac{1}{\omega}\arctan\left(2r\tan\left(\frac{\omega}{2}\right)\right)$ compensate for radial lens distortion.

3. **A small number of the coarsest-scale features are searched for in the image.** In order to match a single map point in the current image *PTAM* performs a patch-based search. Therefore each map-point is first transformed into the camera-coordinate frame using the predicted pose $\mathbf{E}^{+}_{\mathcal{CW}}$ and then projected to the image plane. Around this location a patch based search is performed, again using *SSD* (see Equation 3.1) to determine patch similarity. Additionally, a patch warping is performed to account for viewpoint changes. Further details regarding the patch search can be found in [24].

4. **The camera pose is updated from these coarse matches.** For details on how the pose update is done refer to the following Section 3.3.3.

5. **A larger number of points is reprojected and searched for in the image.** When the coarse tracking is completed and the pose has been updated a first time, the same procedure is repeated using a larger number of points on the lower levels of the image pyramid.

6. **A final pose estimate for the frame is computed from all the matches found.** The final pose estimate is again computed according to Section 3.3.3.

### 3.3.3 Pose Update

From all the matches determined during patch-based search, a camera pose update can be computed as

$$\mathbf{E}^{*}_{\mathcal{CW}} = \mathbf{M}_{\mathcal{C}}\ \mathbf{E}^{+}_{\mathcal{CW}} = \exp\left(\boldsymbol{\mu}_{\mathcal{C}}\right)\ \mathbf{E}^{+}_{\mathcal{CW}} , \tag{3.17}$$

where $\mathbf{M}_{\mathcal{C}}$ accounts for a small camera motion $\boldsymbol{\mu}_{\mathcal{C}}$, which minimizes the sum of reprojection errors for the given set of correspondences. The reprojection error is defined as

$$\mathbf{e}_j \quad = \quad \begin{pmatrix} \hat{u}_j \\ \hat{v}_j \end{pmatrix} - \text{CamProj}\left(\exp\left(\boldsymbol{\mu}_{\mathcal{C}}\right) \mathbf{E}^{+}_{\mathcal{CW}}\, \mathbf{p}_{j\mathcal{W}}\right) \ , \tag{3.18}$$

where $(\hat{u}, \hat{v})^T$ is the position the patch has been found, and $\mathbf{p}_{j\mathcal{W}}$ is the $j^{th}$ 3D-point in terms of the world coordinate frame, respectively. A general approach is to minimize the sum of reprojection errors using a least squares formulation

$$\min \sum_j \left(||\mathbf{e}_j||\right)^2 \ , \tag{3.19}$$

where $||\cdot||$ is the L2-norm. Due to noise, occlusions and false matches the minimization has to be performed *robustly*, as the standard least squares is unstable when outliers are present in the data. Thus the minimization is performed robustly using *M-Estimators* [54], as they are capable of dealing with gross errors by applying a robust penalty function $\rho(e)$ to the residual error. Hence the minimization problem of Equation 3.19 becomes

$$\min \sum_j \rho\left(||\mathbf{e}_j||\right) \ . \tag{3.20}$$

The minimization is done by taking the derivative of Equation 3.20 with respect to a parameter vector $\mathbf{p}$ - which in our case has six dimensions (pose) - and set it equal to zero. Thus,

$$\frac{\partial}{\partial \mathbf{p}} \left( \sum_j \rho\left(||\mathbf{e}_j||\right) \right) = \sum_j \psi\left(||\mathbf{e}_j||\right) \frac{\partial ||\mathbf{e}_j||}{\partial \mathbf{p}} = \sum_j \frac{\psi\left(||\mathbf{e}_j||\right)}{||\mathbf{e}_j||} \mathbf{e}_j^T \frac{\partial \mathbf{e}_j}{\partial \mathbf{p}} = 0 \ , \tag{3.21}$$

where $\psi(e)$ is called influence function and is the derivative of the penalty function $\rho(e)$. If furthermore a weight function $w(e) = \psi(e)/e$ is introduced, the minimization problem of Equation 3.20 using 3.21 is equivalent to minimizing the Iteratively Reweighted Least-Squares (*IRLS*) [68] problem

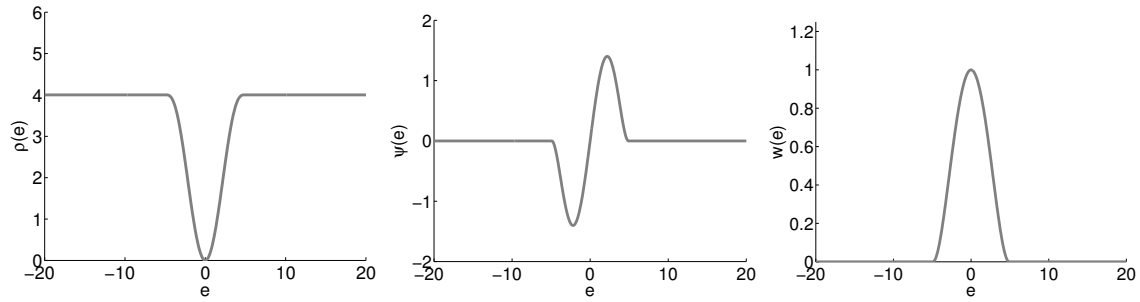
$$\min \sum_j w\left(||\mathbf{e}_j||\right) ||\mathbf{e}_j||^2 \ , \tag{3.22}$$

where $w\left(||\mathbf{e}_j||\right)$ is a robust weight function. A commonly used weight function illsutrated in Figure 3.7 is the Tukey biweight function [69], which is used throughout this work when applying

*M-Estimators*, and is defined as

$$w_{\text{TUK}}(e) = \begin{cases} \left(1 - \left(\frac{e}{c}\right)^2\right)^2 & \text{if } |e| \leq c \\ 0 & \text{if } |e| > c \end{cases} \tag{3.23}$$

where $c$ is a threshold for outlier rejection, usually determined robustly from the residual's distribution (e.g. median of standard deviation). The Tukey function even suppresses gross outliers.




**(a)** Penalty function $\rho(e)$ **(b)** Influence function $\psi(e)$ **(c)** Weight function $w(e)$

**Figure 3.7:** *This figure depicts the robust penalty function $\rho$ (a), the influence function $\psi$ (b), as well as the weight function $w$ (c) according to Tukey. These functions are used during robust iterative least-squares minimization to account for measurement errors and gross outliers.*

The optimal pose update vector $\boldsymbol{\mu}'$ is then computed using the Tukey *M-Estimator* to perform robust minimization by applying *IRLS*, after convergence yielding the pose update vector as

$$\boldsymbol{\mu}' = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \sum_j w_{\text{TUK}}\left(||\mathbf{e}_j||\right) ||\mathbf{e}_j||^2 , \tag{3.24}$$

from which $\mathbf{M}_{\mathcal{C}} = \exp\left(\boldsymbol{\mu}'\right)$ and furthermore the final pose $\mathbf{E}^*_{\mathcal{CW}}$ can be computed using Equation 3.17.

## 3.4 Map Management

*Bundle Adjustment* (*BA*) is a global optimization strategy where under the assumption of a Gaussian noise distribution the *MLE* for a set of 3D-points $\hat{\mathbf{X}}_j$ and a set of camera projection matrices $\hat{\mathbf{P}}^i$ is searched for, such that the geometric image distance $d$ between the measured image points

$\mathbf{x}_j^i$ and the projected image points $\hat{\mathbf{x}}_j^i = \hat{\mathbf{P}}^i\hat{\mathbf{X}}_j$ is minimized according to

$$\min_{\hat{\mathbf{P}}^i,\hat{\mathbf{X}}_j} \sum_{ij} d\left(\hat{\mathbf{P}}^i\hat{\mathbf{X}}_j, \mathbf{x}_j^i\right)^2 . \tag{3.25}$$

According to [4], *BA* should be used as a final step of any reconstruction algorithm. *BA* is very flexible and allows the integration of camera constraints and constraints on the position of 3D-points. Yet, *BA* has two major drawbacks, including the need of a good initialization and the growth in time of the minimization problem along with the number of parameters. *PTAM* delivers a quite good initialization where *BA* can be applied easily, however, the number of increasing parameters to optimize remains. Since each camera generally has eleven *DOF*, and each 3D-point an additional three *DOF*, a minimization for $n$ 3D-points and $m$ camera matrices involves $3n+11m$ parameters in general, or $3n+6m$ parameters in case of a calibrated camera. This results in a fast growth in time needed for global *BA*, as the number of points and cameras increase. According to the fact that most points are only observed by a small set of cameras, the resulting sparseness of the problem can be exploited using e.g. sparse Cholesky factorization or sparse *LM*, resulting in tremendous speedups. The Jacobian matrix containing the first order partial derivatives

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} , \tag{3.26}$$

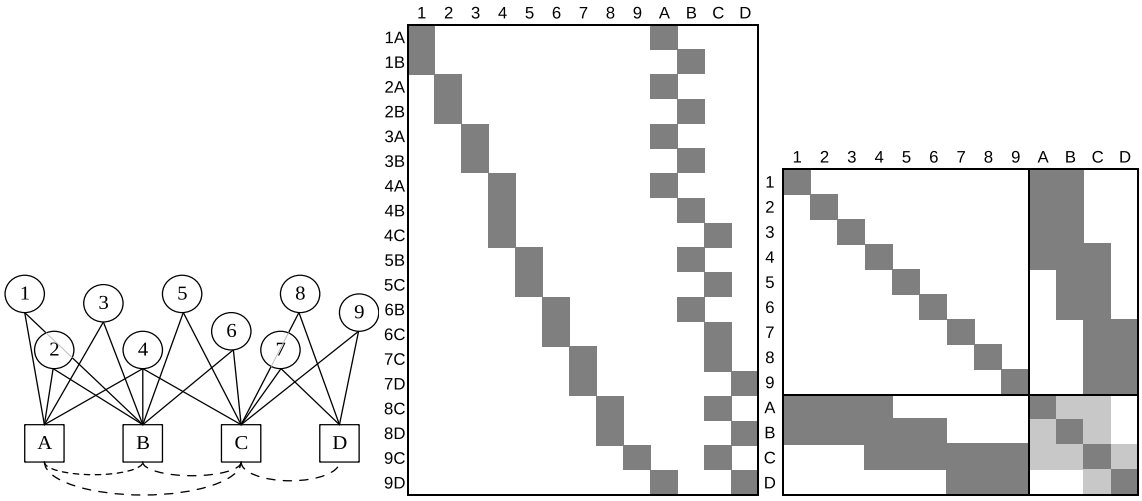
as well as the square Hessian matrix of second order partial derivatives of a function

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \frac{\partial^2 f}{\partial x_2 x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \frac{\partial^2 f}{\partial x_n x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} , \tag{3.27}$$

are used in Newton-type optimization methods like *LM*, as they occur in local Taylor expansion of a function

$$f(\mathbf{x}+\boldsymbol{\Delta}\mathbf{x}) \approx f(\mathbf{x}) + \mathbf{J}(\mathbf{x})\boldsymbol{\Delta}\mathbf{x} + \frac{1}{2}\boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}(\mathbf{x})\boldsymbol{\Delta}\mathbf{x} . \tag{3.28}$$

Since most points are only observed by a small set of cameras, $\mathbf{J}$ and $\mathbf{H}$ have a sparse, block-like structure, allowing a more efficient solution of the *BA* problem by using sparse methods. An example from [70] of an artificial *BA* problem, including the block-like structures of $\mathbf{J}$ and $\mathbf{H}$, respectively, is given in Figure 3.8.




**(a)** Example **(b)** Jacobian matrix **(c)** Hessian matrix

**Figure 3.8:** *Figure (a) describes an artificial structure from motion problem, where A-D are cameras and 1-9 feature points. The lines indicate which point is observed by which camera. (b) depicts the associated Jacobian* $\mathbf{J}$*, and (c) the Hessian* $\mathbf{H}$*. The matrices are ordered in a way that the point variables are before the camera variables. The block-like structure of the two matrices, resulting from the fact that most points are only observed by a small set of cameras, allows more effective optimization using sparse methods. Image taken from [70].*

Although sparsity is exploited, *PTAM* needs tens of seconds for full *BA* to converge when the map has more than 150 keyframes. Hence, to perform optimization in a reasonable amount of time, *PTAM* also performs local *BA*. During local *BA* the pose of the most recent keyframe and its closest neighbors, together with all map points observed by these keyframes, are optimized. This optimization is performed in the same way as global bundle adjustment, but with the difference that only a small part of the map is optimized. Only if enough computational resources are available, *PTAM* performs full *BA*. Thus, during map optimization first a local *BA* is performed for the most recent keyframe added, and if this optimization has converged global *BA* proceeds during the next iteration, as depicted by Figure 3.9. The changes made to the locations of the keyframes and the positions of the map points during optimization are directly applied to the map.

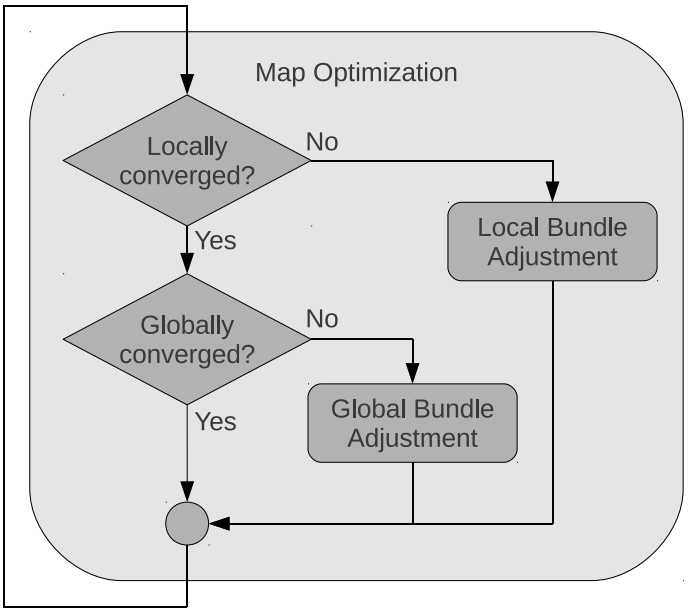



**Figure 3.9:** *During map optimization in the mapping thread PTAM always performs a local BA first. There, the most recent keyframe and its closest neighboring keyframes are optimized. If the local optimization has converged, during the next iteration the entire map is optimized until convergence has been reached.*

## 3.5 Summary

In this chapter the current state-of-the-art implementation of a visual simultaneous localization and mapping implementation, namely *Parallel Tracking And Mapping* (*PTAM*) [24] has been described. Important parts of the framework, including initialization, iterative pose and map updates as well as map management have been discussed in more detail, as these parts are important to our approach presented in the remainder of this work.

# Chapter 4

# Online Model-Based Multi-Scale Pose Estimation

**Contents**



In this chapter we present how the *VSLAM* framework introduced in Chapter 3 can be extended to make it suitable for online model-based multi-scale pose estimation. Major components include a metric initialization, a model-based pose refinement, and a model-based tracking stage.

## 4.1 Overview

Initial experiments regarding the fully automatic estimation of the camera's pose with respect to a single known object from still images have shown unsatisfactory results, especially for wiry objects. As wiry objects have little associated texture, descriptor-based and appearance-based approaches are not applicable. Correspondence-based approaches do not fulfill our real-time requirement as the large amount of unknown correspondences require robust *RANSAC*-style algorithms with a rather high number of iterations. Because of these issues, and furthermore because of the availability of a continuous image stream, we employ a *VSLAM* approach where the pose of the camera just has to undergo small changes from one frame to the next. We build our work on the keypoint-based *VSLAM* framework introduced in Chapter 3, which is already real-time capable and copes with multiple scales. However, to incorporate the prior knowledge about the object in

form of a *CAD*-model, we extend the framework in several ways.

First, we use the *CAD*-model of the object to generate a metrically correct map during initialization. Hence, a metrically correct pose is determined in every frame and can be fused with other systems such as *GPS*, or can be integrated into cartographic maps such as Open Street Map (*OSM*) [1]. During initialization some user interaction is required, however, we thus avoid object-dependent pose ambiguities right from the beginning.

After having initialized the keypoint-based tracking, the object is actually only tracked by its surroundings. This has on the one hand the advantage that the pose can be estimated over multiple scales, and that the pose estimation is robust to partial occlusions of the object. Additionally, pose estimation is even possible when the object completely vanishes, which is not the case for pose estimation algorithms based on the object alone. On the other hand, given a good prior pose estimate from keypoint-based tracking, we can use the knowledge about the object's geometry to refine the pose, which is the second extension. A refinement is only triggered if the object is present in the image, and furthermore occupies a significant amount of it. The refinement itself is robust to partial occlusions as well, and helps to improve accuracy especially when the camera is close to the object.

Keypoint-based pose estimation works well when enough discriminative world information is detected in the current image. When thinking for instance of inspection tasks, closeup views of the known object are sometimes necessary. When being that close to an object, keypoint-based tracking often fails as too few discriminative world information can be found in the image. However, the model is very prominent in such cases. As a third extension, we therefore impose a purely model-based tracking component. Model-based tracking tries to close the gap until revisiting already explored areas, where we can switch back to keypoint-based tracking. The model-based tracking component works the same way as the refinement component, however, due to the lack of a background-based pose propagation it is more affected by fast translations and rotations. We therefore just rely on this approach if keypoint-based tracking fails.

The keypoint-based *VSLAM* approach has been extended with several model-based components. A graphical overview of the entire system is given in Figure 4.1. The remainder of this chapter explains the metric initialization process, as well as the individual steps needed for pose refinement and model-based tracking.

[1] `http://www.openstreetmap.org`

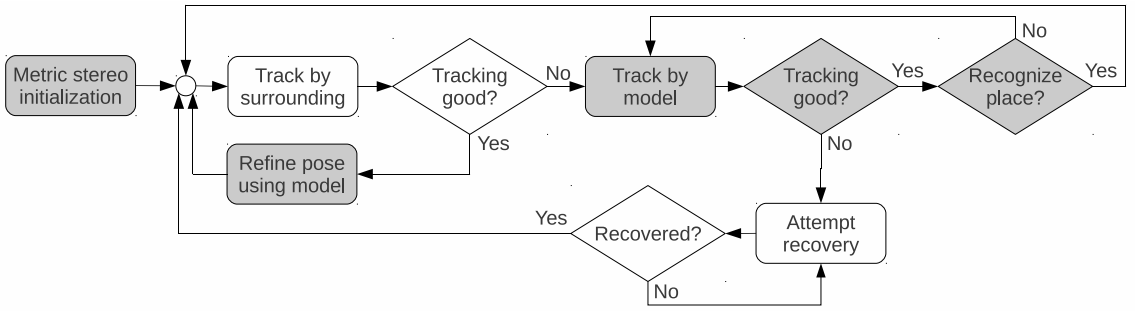



**Figure 4.1:** *System overview. The gray blocks depict the extensions. After a metric initialization using the CAD-model, the object is tracked by its surroundings. This pose is then refined to improve accuracy. When tracking the object by its surroundings fails, a model-based tracking takes over, while seamlessly switching back to a background-based tracking when visiting already explored areas. In case tracking is completely lost a recovery attempt is initiated.*

## 4.2 Initial Pose Estimation and Map Construction

Detailed information about a single object in the scene is given in form of a *CAD*-model. This means that the *VSLAM* algorithm can be initialized *metrically*. Therefore, we extend the previously introduced initialization process. Beyond the frame-by-frame feature tracking, we now need two additional steps during initialization:

- **Solving a *PnP*-problem.** The first camera pose is calculated by choosing point correspondences between 3D-object points $\mathbf{X}_i$ and the 2D-image points $\mathbf{x}_i$. This results in a *PnP*-problem, which is solved using an iterative *LM* optimization algorithm. We solve the problem by employing Zhang's method [71] for camera calibration, with the exception that the intrinsic camera parameters as well as the distortion coefficients are given a-priori, as the camera has already been calibrated beforehand. The algorithm then minimizes the sum of reprojection errors

$$\min_{\mathbf{P}} \sum_i d\left(\mathbf{P}\mathbf{X}_i, \mathbf{x}_i\right)^2 \ , \tag{4.1}$$

  where $d(\mathbf{a}, \mathbf{b})$ is the geometric image distance between the points $\mathbf{a}$ and $\mathbf{b}$ measured in pixels. As *LM* methods need an initial estimate, planar and non-planar cases of point configurations are considered. In the case of a planar configuration the initial estimate is calculated via homography estimation and decomposition, whereas in the case of non-planar points *DLT* is used to solve for an initial guess. As soon as *LM* has converged, the first camera position $\mathbf{C}_1$ is determined, and frame-by-frame feature tracking starts. Note that at least four correspondences are necessary for a unique solution, and that they have to be chosen care-

fully as outliers are not handled. Our implementation therefore requires a human operator to choose the correspondences between model and image.

- **Rescaling the baseline.** After the end of frame-by-frame feature tracking is determined by the user, a homography is estimated between the initial camera $\mathbf{C}_1$ and the translated camera $\hat{\mathbf{C}}_2$ and further decomposed into a rotation matrix $\mathbf{R}$ and a relative translation vector $\hat{\mathbf{t}}$ using [44]. However, the translation $\hat{\mathbf{t}}$ is only determined up to an unknown scale factor $s$. The scale is corrected by first choosing another correspondence between a previously chosen 3D-model point $\mathbf{X}$ and a 2D-image point $\mathbf{x}_{\hat{\mathbf{C}}_2}$. Thus, as by now two image locations of one and the same point, namely $\mathbf{x}_{\mathbf{C}_1}$ and $\mathbf{x}_{\mathbf{C}_2}$, and the relative camera translation $\hat{\mathbf{t}}$ and rotation $\mathbf{R}$ between the two views $\mathbf{C}_1$ and $\hat{\mathbf{C}}_2$ are known, the corresponding 3D-model point $\hat{\mathbf{X}}_{\mathbf{C}_1}$ can be triangulated in the camera coordinate frame of camera $\mathbf{C}_1$. After the 3D-model-point $\mathbf{X}$ has been transformed to the same coordinate frame, the disparity in length between the true $\mathbf{X}_{\mathbf{C}_1}$ and the reconstructed 3D-model-point $\hat{\mathbf{X}}_{\mathbf{C}_1}$ can be calculated, from which the scale factor
$$s = \frac{||\mathbf{X}_{\mathbf{C}_1}||}{||\hat{\mathbf{X}}_{\mathbf{C}_1}||} \tag{4.2}$$
can be derived using similar triangles, where $||\ \cdot\ ||$ depicts the length of the vector. This factor is then used to scale the baseline to its correct metric length, yielding the true metric translation vector $\mathbf{t} = s\hat{\mathbf{t}}$. This procedure is depicted in Figure 4.2.

The only constraint enforced to the initialization is the availability of a metrically defined model of the object of interest in form of a 3D-point list together with a list of connected vertices, similar to Virtual Reality Modeling Language (*VRML*)-files such as the one shown in Listing 4.1. It is important to our implementation that the model is represented using a quadrangular mesh rather than a triangular mesh, as with a triangular mesh additional edges would be introduced, which are not part of the physical object. Furthermore, the object has to be centered around the origin of the xy-plane and grows in direction of the positive z-axis. Some examples of 3D-objects and their corresponding *CAD*-models are given in Figure 4.3. By enforcing these constraints, the model coordinate system and the world coordinate system are the same, and additional transformations between these two coordinate systems during pose estimation are avoided.

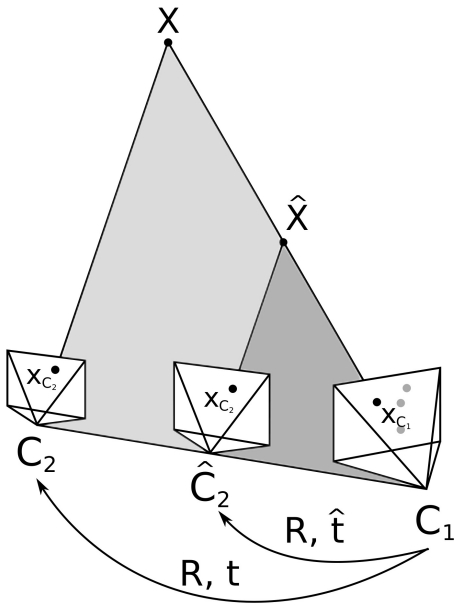



**Figure 4.2:** *The location of camera $\mathbf{C}_1$ is determined by solving a PnP-problem, after at least four point correspondences between a metric object model and the image have been chosen manually. Then, frame-by-frame feature tracking is used to track correspondences, from which a homography is estimated and subsequently the relative camera motion $\hat{\mathbf{t}}$ and rotation $\mathbf{R}$ between $\mathbf{C}_1$ and $\hat{\mathbf{C}}_2$ are extracted. The translation vector $\hat{\mathbf{t}}$ is corrected by $\mathbf{t} = s\hat{\mathbf{t}}$, where the scaling factor $s$ is determined using an additional correspondence $\mathbf{x}_{\mathbf{C}_2}$ and the disparity in length between the reconstructed $\hat{\mathbf{X}}$ and the true model-point $\mathbf{X}$. Thus, the exact location of camera $\mathbf{C}_2$ is known too.*

**Listing 4.1:** *This example shows the definition of a metric CAD-model. The definition of the 3D-point list together with a list of point indices connecting the vertices is shown. The model is defined using quadrangles rather than triangles. The connection of four points is terminated via -1.*

```
# Inventor V2.1 ascii
        Separator {
            DEF points Coordinate3 {
            point [
                # bottom
                0.089    -0.123    0.000  # vertex 0
                0.089     0.123    0.000  # vertex 1
               -0.089     0.123    0.000  # vertex 2
               -0.089    -0.123    0.000  # vertex 3
                # top
                0.089    -0.123    0.030  # vertex 4
                0.089     0.123    0.030  # vertex 5
               -0.089     0.123    0.030  # vertex 6
               -0.089    -0.123    0.030  # vertex 7
               ]
            }
```

```
        DEF connections IndexedFaceSet {
        coordIndex [
            0, 1, 2, 3, 0, -1       # bottom
            4, 5, 6, 7, 4, -1       # top
            0, 1, 5, 4, 0, -1       # front side
            0, 3, 7, 4, 0, -1       # left side
            1, 2, 6, 5, 1, -1       # right side
            3, 2, 6, 7, 3, -1       # back side
            ]
        }
    }
```

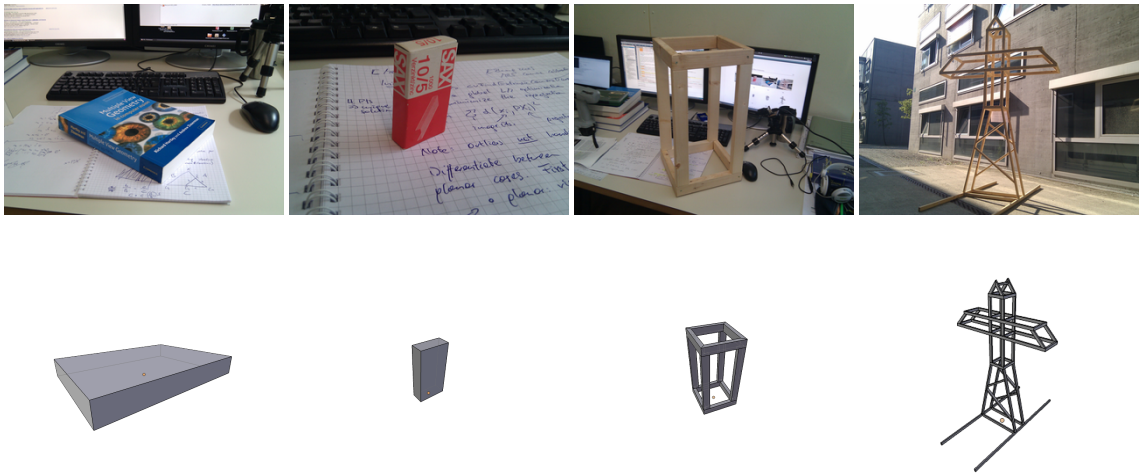
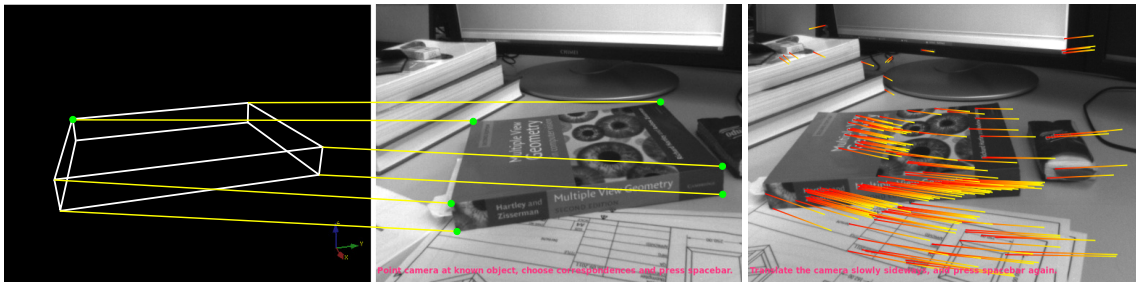


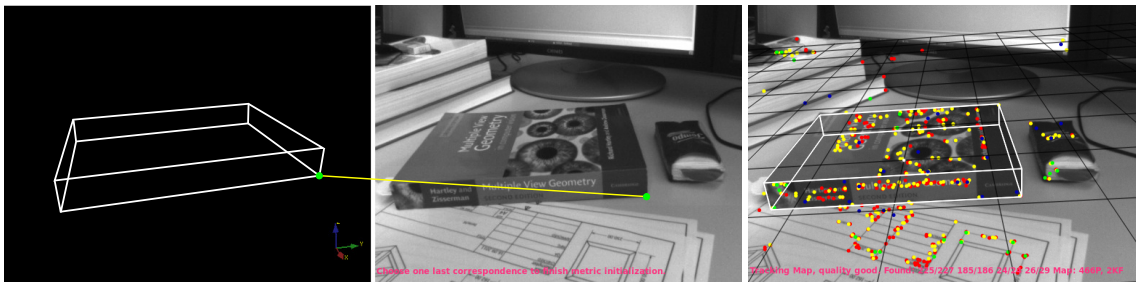
**Figure 4.3:** *The top row shows the objects of interest, namely a book, a box for staples, a wooden wiry box, and a wooden model of a power pylon. The corresponding rendered CAD-models are depicted in the bottom row, respectively.*

While manual initialization has the drawback of requiring more user interaction, it has another important advantage besides the fact that the localization and mapping tasks are now performed metrically. Difficult and object dependent ambiguities are eliminated right at the beginning, thus all pose ambiguities – multiple poses for which the object appears the same – can be neglected. Therefore, not even a cube looking the same from six different viewpoints states a problem to the proper functionality of our algorithm. This, however, is not only the result of manual initialization, but rises from the fact that the object's pose is actually determined using the entire image. Hence, the object's pose is determined by all features detected, regardless whether they are located on the object itself or its surroundings. In Figure 4.4 the procedure of metric initialization is illustrated step by step.



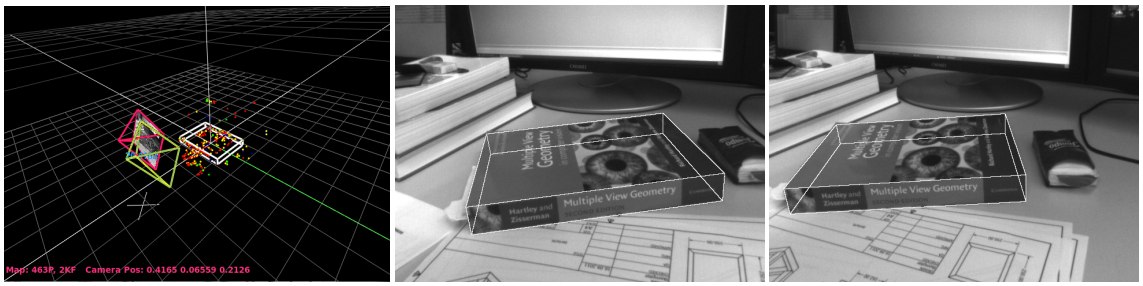
**(a)** First, at least four correspondences between *CAD*-model and image have to be selected manually. These $n$ correspondences are then used to solve the $PnP$-problem and retrieve the pose of the first keyframe. After a first pose has been retrieved, the camera has to be translated sideways and may also be rotated.

**(b)** During translation salient image locations are tracked. These are used to estimate a homography, from which a relative camera translation $\hat{\mathbf{t}}$ and rotation $\mathbf{R}$ are then extracted.



**(c)** Finally, one last correspondence has to be chosen manually to perform the scale correction of the baseline. After re-scaling the baseline to its correct metric length, simultaneous tracking and mapping of the environment begins.

**(d)** During tracking the xy-plane is drawn as grid, the detected feature points are shown as dots, and the wireframe model is overlaid to the scene.



**(e)** The 3D-map points are shown as dots, while the model is rendered as a white wireframe. The first two keyframes are shown in red and yellow, respectively.

**(f)** The image of the first keyframe with the wireframe model overlaid.

**(g)** The image of the second keyframe with the wireframe model overlaid.

**Figure 4.4:** *All steps needed for metric initialization are illustrated successively by figures (a-c). After initialization has been completed, the map is shown in figures (d) and (e). The images of the first and second keyframe are depicted by (f) and (g), respectively.*

## 4.3 Iterative Pose and Map Update

After an initial map has been generated, the algorithm has to simultaneously update the map when exploring the environment in order to successfully track its position. As described in Chapter 3, at each frame received, the current camera pose is updated iteratively during a two-stage coarse-to-fine tracking procedure. The last pose of the camera is therefore updated by forward-propagation using a motion model (Equation 3.15), giving a predicted pose $\mathbf{E}^{+}_{\mathcal{CW}}$. Using this estimated pose all map points are projected into the current image, a patch-based search is performed to determine correspondences, and a pose update vector $\boldsymbol{\mu}'$ is robustly calculated using an *M-Estimator* (Equation 3.24). The final pose $\mathbf{E}^{*}_{\mathcal{CW}}$ is then calculated as

$$\mathbf{E}^{*}_{\mathcal{CW}} = \exp(\boldsymbol{\mu}')\, \mathbf{E}^{+}_{\mathcal{CW}} \,. \tag{4.3}$$

During tracking, unknown locations in the environment are visited, and the map therefore has to be updated accordingly. *PTAM* therefore creates so-called keyframes at previously unknown locations, which are separated from each other not only by a minimum number of frames, but as well by a minimum distance in space. When a keyframe is inserted, it is integrated into the map and new 3D-map points are triangulated, added to the map, and in succession used again during tracking.

In contrast to the standard *PTAM* approach, the pose $\mathbf{E}^{*}_{\mathcal{CW}}$ is now again refined using the available model information if the model is currently present in the image and occupies a significant area. The technique we use to refine our pose is based on an edge-based tracking algorithm proposed by Drummond and Cipolla [50], which has already been successfully applied to several augmented reality applications, including e.g. work of Klein et al. [48] and Reitmayr et al. [49]. This pose update is point-based and again performed in a robust manner, thus several steps are needed:

- Interpolate additional model points on edges of the *CAD*-model and determine their visibility using the depth buffer information retrieved from a rendering component,
- perform a perpendicular search for the strongest image gradient of approximately same direction as the model's edge which is expected to be part of the object, and
- robustly calculate another pose update vector using an *M-Estimator* in combination with *IRLS*.

The refinement component is not only used to refine the pose for increased accuracy, but also for frame-to-frame tracking in cases where the keypoint-based approach loses the map. It is schematically depicted in Figure 4.5, and subsequently each block is going to be described in detail.

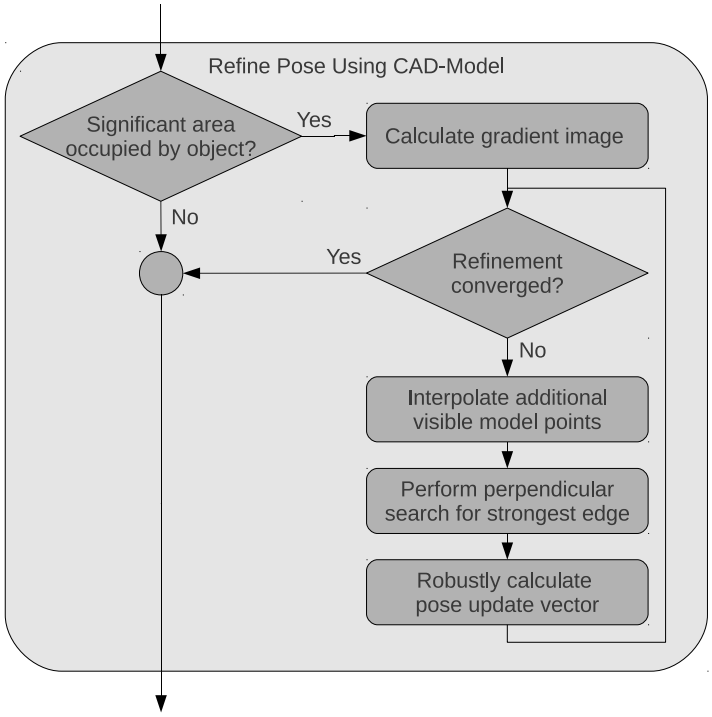



**Figure 4.5:** *The refinement component, consisting of several blocks, has been integrated into the keypoint-based tracking loop to refine the pose, or even take over tracking in cases where the map gets lost. The CAD-model is used during determination of the area occupied by the object, during interpolation of additional model points, as well as during determination of their visibility using a rendering component, and during the perpendicular search for the strongest edge.*

**Determination of the area occupied by the object.** First, when a refinement is triggered, the model points are projected to the image. Then, a bounding box is determined and the ratio between the image and the area of the bounding box is calculated. If this ratio

$$\frac{A_{\text{bounding box}}}{A_{\text{image}}} > t \tag{4.4}$$

exceeds a certain threshold $t$, a refinement is performed. Otherwise, the object is either not present in the image anyway, or too small, so that calculating a refinement based on the object's model would not improve the camera's pose notably. A few examples of the projected model points together with the determined bounding box for the 3D-model of a book are given in Figure 4.6.

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

**Figure 4.6:** *The first step needed when triggering a pose refinement is the determination of the area currently occupied by the object. A refinement is triggered in case the object is fully present in the image as in (a), or only partially visible as in (b), as long as it is large enough. However, if the object is too small as in (c) or even not present at all as in (d) no refinement is performed.*

**Computation of gradient image.** The refinement is based on gradient information extracted from the current image $I$. Therefore, the image is convolved with the derivatives of a Gaussian kernel $G_x(x, y, \sigma)$ in x-direction and $G_y(x, y, \sigma)$ in y-direction, respectively. The kernels are calculated as

$$G_x(x, y, \sigma) = \frac{\partial}{\partial x}\left(\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)\right) = -\frac{x}{\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \quad \text{and} \tag{4.5}$$

$$G_y(x, y, \sigma) = \frac{\partial}{\partial y}\left(\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)\right) = -\frac{y}{\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \quad , \tag{4.6}$$

where the standard deviation $\sigma$ is the only parameter. Pixels farther away from the center therefore have smaller influence on the result of the convolution, whereas pixels beyond $3\sigma$ have negligible effects on the result. The image derivatives are then calculated as

$$I_x = G_x * I \quad \text{and} \tag{4.7}$$

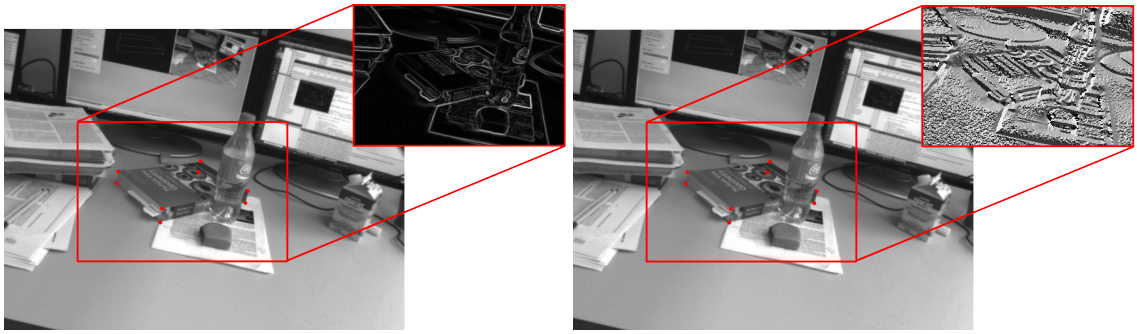
$$I_y = G_y * I \quad , \tag{4.8}$$

where $F * I$ expresses a convolution of an image $I$ with a convolution kernel $F$. Using the image derivatives $I_x$ and $I_y$, the gradient magnitude $|\nabla I(x, y)|$ as well as the gradient direction $\psi$ can be computed as

$$|\nabla I(x, y)| = \sqrt{I_x^2 + I_y^2} \quad \text{and} \tag{4.9}$$

$$\psi = \arg(I_x, I_y) \quad , \tag{4.10}$$

where arg $(x, y)$ is the angle from the x-axis to the point $(x, y)$. The previously calculated bounding box is increased by a certain percentage, and the magnitude and phase information is extracted for this region of interest only. An example is given in Figure 4.7.



**(a)** Magnitude information **(b)** Phase information

**Figure 4.7:** *The calculated bounding box is increased by a certain percentage, and gradient information is extracted according to Equations 4.9 and 4.10, respectively.*

**Interpolation of additional model points and determination of their visibility.** Typically too few model points exist in order to perform a robust refinement and additional 3D-model points need to be interpolated. Therefore, all model points are first projected into the image. Then, on each line connecting a pair of model points $(\mathbf{A}, \mathbf{B})$ the number of additionally needed 3D-points

$$N_i = \frac{d\left(\mathbf{PA}_i, \mathbf{PB}_i\right)}{s} \tag{4.11}$$

is determined, where $d(a, b)$ is the distance in pixel between two image points $a$ and $b$, $\mathbf{P}$ is the projection matrix for the actual camera pose $\mathbf{E}^*_{\mathcal{CW}}$, and $s$ is the maximum distance in pixel for two neighboring model points. After adding

$$N = \sum_i N_i \tag{4.12}$$

additional points to the 3D-model, each of these points has to be tested for visibility in the current frame. For this purpose, a *rendering component* is used where the *CAD*-model is rendered under its currently estimated pose $\mathbf{E}^*_{\mathcal{CW}}$. After rendering, the depth buffer – also called z-buffer – is retrieved [72]. Each value in the z-buffer contains the depth value of the object closest to the camera, and therefore allows testing for visibility of a 3D-world point $\mathbf{X}_{\mathcal{W}}$ by first transforming it

to the camera coordinate frame via

$$\mathbf{X}_{\mathcal{C}} = \mathbf{E}^{*}_{\mathcal{C}\mathcal{W}}\mathbf{X}_{\mathcal{W}} \; . \tag{4.13}$$

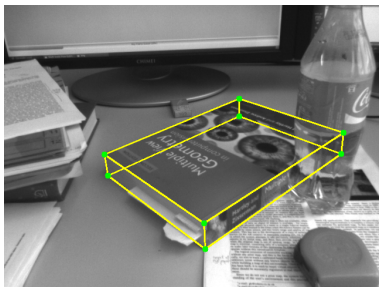
Then, the 3D-point $\mathbf{X}_{\mathcal{C}}$ is projected from the camera coordinate frame to normalized clipping coordinates $\mathbf{x}_{ncc}$ in the range of $[-1, 1]$, using

$$\begin{pmatrix} x_{ncc} \\ y_{ncc} \\ z_{ncc} \end{pmatrix} = \mathbf{P}_{4\times4} \begin{pmatrix} X_{\mathcal{C}} \\ Y_{\mathcal{C}} \\ Z_{\mathcal{C}} \\ 1 \end{pmatrix} \; , \tag{4.14}$$

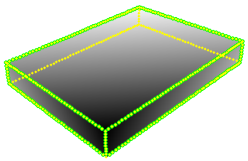
where $\mathbf{P}_{4\times4}$ is the *OpenGL* camera projection matrix which has been used during rendering. Hereafter, $(x_{ncc}, y_{ncc})^T$ are linearly transformed to image coordinates $(u_{ncc}, v_{ncc})^T$, where $(u_{ncc}, v_{ncc})^T$ is used as a lookup index in the z-buffer to retrieve the correct depth value for the 3D-point. If the difference

$$|z_{ncc} - z_{buffer}\,(u_{ncc}, v_{ncc})\,| < \tau \tag{4.15}$$

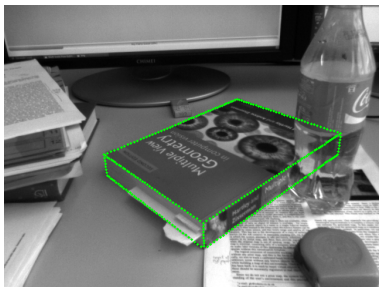
between the depth $z_{ncc}$ of the 3D-point in the normalized clipping coordinate frame and the corresponding z-buffer value $z_{buffer}\,(u_{ncc}, v_{ncc})$ is below a threshold $\tau$, the 3D-point $\mathbf{X}_{\mathcal{W}}$ is considered visible, otherwise invisible. As the z-buffer only contains values between $[-1, 1]$, where the *near*-plane is at $-1$, and the *far*-plane at $1$, these planes are always set as close to the object's boundaries as possible, in order to retrieve highest possible accuracy for the depth values. In Figure 4.8 an example of the interpolation of additional visible model points is given, starting with only the information from the *CAD*-model, the visibility test using the depth buffer from a rendering component, to the final result of the visible model points. Another example of interpolating and testing for visible points is given for a wiry object, depicted in Figure 4.9.



**(a)** Only the *CAD*-model information is available at the beginning, including the eight model points (green dots) and the edges (yellow).

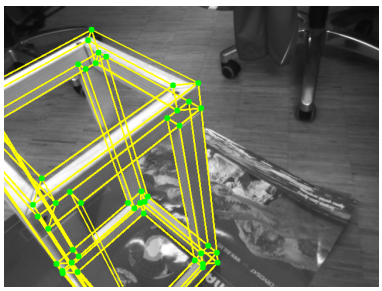


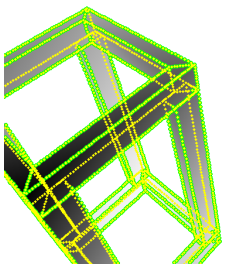
**(b)** After interpolating additional points, all of them (yellow) are tested for visibility, using depth information. Visible points are highlighted by a green circle.



**(c)** The final result of all additional and visible points is overlaid to the scene.

**Figure 4.8:** *Starting from the available CAD-model information (a) and the current pose* $\mathbf{E}^*_{\mathcal{CW}}$, $N = 782$ *additional 3D-model points are interpolated. The CAD-model is then rendered and all points are tested for visibility using the z-buffer, which can be retrieved from the rendering component. The depth buffer is depicted in (b) together with all points to test (yellow). The* 606 *visible points are encircled green, and finally overlaid to the scene in (c). Occlusions by other objects like the bottle cannot be handled here, as only the available CAD-model can be rendered and used for visibility testing.*





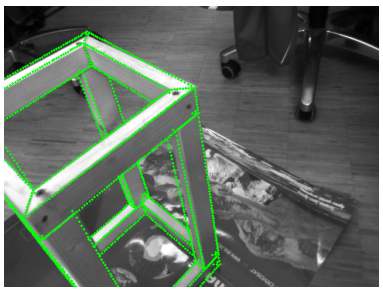


**Figure 4.9:** *This example shows the same three steps as depicted by Figure 4.8, except for a wiry object.* $N = 2403$ *additional visible 3D-model points are shown here. Despite the small cross-sections of the object, depth testing yields very accurate results. Note that parts of the object are outside the image.*

**Perpendicular search for the strongest edge.** For all model points including the interpolated ones, a linear search for the strongest gradient magnitude is performed as depicted in Figure 4.10. The search line $\eta$ is arranged perpendicular to the model's edge. The projected model point is now shifted along the perpendicular search line to the image point $(x, y)$ which maximizes

$$\max_{(x,y)} \left( |\nabla I(x,y)| \exp\left( -\frac{v^2 - \mu^2}{2\sigma^2} \right) - \lambda \sphericalangle(\boldsymbol{\psi}(x,y), \boldsymbol{\eta}) \right) , \tag{4.16}$$

where the image gradient magnitude $|\nabla I(x, y)|$ is weighted with a Gaussian windowing function centered at the projected model point's position $\mu$, the scalar $v$ denotes the distance along the search line, and $\lambda \sphericalangle(\mathbf{n_1}, \mathbf{n_2})$ is a term that penalizes orientation deviations between the perpendicular search direction $\boldsymbol{\eta}$ and the image gradient direction $\boldsymbol{\psi}(x, y)$. In short, what we are looking for is the nearest image location of the projected model point with the strongest gradient response and approximately the same direction.

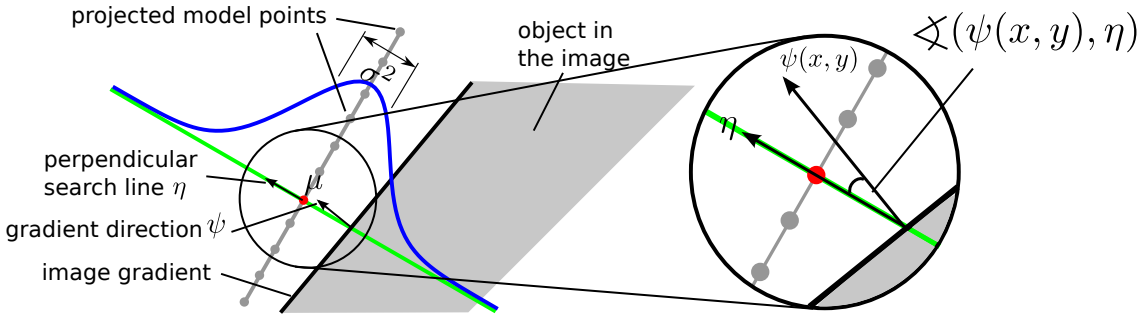



**Figure 4.10:** *For each projected model point, a linear search perpendicular to the model's edge is performed. The image point is thereby shifted along the search line $\boldsymbol{\eta}$, while looking for a strong image gradient with approximately the same direction. The gradient magnitude is weighted with a Gaussian windowing function centered at $\mu$, where $\sigma^2$ controls the expansion of the search region.*

The perpendicular search strategy proposed here is similar to that used by Berger et al. [73]. However, false matches might occur with models consisting of thin structures delimited by two edges, as it is the case with wiry structures. We try to overcome this issue by using a tight search region with small $\sigma$ during the first iteration, assuming a good initial guess in the first place. When refinement doesn't converge after the first iteration, the search region is expanded.

**Robust computation of the pose update vector.** After a new image location has been determined for each visible model point using the perpendicular search strategy, the image distances between old and new positions are assigned to an error vector $\mathbf{e}$. We then calculate a motion vector

$\boldsymbol{\mu}$ which minimizes these image errors by solving the equation system

$$\mathbf{J}\boldsymbol{\mu} = \mathbf{e}\ , \tag{4.17}$$

where $\mathbf{J}$ is a Jacobian matrix describing the effect of each element of $\boldsymbol{\mu}$ onto each element of $\mathbf{e}$ in direction perpendicular to the edge. The elements of $\mathbf{J}$ are therefore calculated by taking the partial derivatives at the current pose along the edge normal as

$$J_{ij} = \frac{\partial e_i}{\partial \mu_j} = \langle \hat{\mathbf{n}}_i\ ,\ \begin{pmatrix} \frac{\partial u}{\partial \mu_j} \\ \frac{\partial v}{\partial \mu_j} \end{pmatrix} \rangle\ , \tag{4.18}$$

where $(u, v)^T$ are the image coordinates of the $i^{th}$ projected model point, $\hat{\mathbf{n}}_i$ is the corresponding unit length edge normal, and $\langle x, y \rangle$ denotes the tensor dot product. Recalling from Equation 3.16 that a world point $\mathbf{p}_{i\mathcal{W}}$ is projected to the image coordinates $(u, v)^T$ by first transforming it to the camera coordinate frame using the actual pose estimate $\mathbf{E}^*_{\mathcal{CW}}$ and then performing the camera projection as

$$\begin{aligned} \begin{pmatrix} u_i \\ v_i \end{pmatrix} &= \text{CamProj}\left(\mathbf{E}^*_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}\right) \\ &= \text{CamProj}\begin{pmatrix} X_{i\mathcal{C}} \\ Y_{i\mathcal{C}} \\ Z_{i\mathcal{C}} \\ 1 \end{pmatrix} \\ &= \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \end{bmatrix} \begin{pmatrix} \frac{X_{i\mathcal{C}}}{Z_{i\mathcal{C}}} \\ \frac{Y_{i\mathcal{C}}}{Z_{i\mathcal{C}}} \\ 1 \end{pmatrix}\ . \end{aligned} \tag{4.19}$$

The radial lens distortion can be ignored here, as the images are undistorted prior to applying our algorithm. Thus, Equation 4.19 has to be differentiated with respect to the motion parameter vector $\boldsymbol{\mu}$ by applying the chain rule

$$\begin{aligned} \begin{bmatrix} \frac{\partial u_i}{\partial \boldsymbol{\mu}} \\ \frac{\partial v_i}{\partial \boldsymbol{\mu}} \end{bmatrix} &= \frac{\partial\ \text{CamProj}\left(\mathbf{E}^*_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}\right)}{\partial \boldsymbol{\mu}} \\ &= \underbrace{\frac{\partial\ \text{CamProj}\left(\mathbf{X}\right)}{\partial \mathbf{X}}\bigg|_{\mathbf{X}=\mathbf{E}^*_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}}}_{\text{I}} \underbrace{\frac{\partial\left(\mathbf{E}^*_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}\right)}{\partial \boldsymbol{\mu}}}_{\text{II}} \end{aligned} \tag{4.20}$$

with I:

$$\left.\frac{\partial\,\text{CamProj}\,(\mathbf{X})}{\partial\mathbf{X}}\right|_{\mathbf{X}=\mathbf{E}^{*}_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}} = \left.\frac{\partial\left(\begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \end{bmatrix}\begin{pmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \\ 1 \end{pmatrix}\right)}{\partial\mathbf{X}}\right|_{\mathbf{X}=\mathbf{E}^{*}_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}}$$

$$= \begin{bmatrix} f_x\frac{1}{X_z} & 0 & -f_x\frac{X_x}{X_z^2} \\ 0 & f_y\frac{1}{X_z} & -f_y\frac{X_y}{X_z^2} \end{bmatrix}, \tag{4.21}$$

and with II:

$$\frac{\partial\left(\mathbf{E}^{*}_{\mathcal{CW}}\mathbf{p}_{i\mathcal{W}}\right)}{\partial\boldsymbol{\mu}} \overset{\text{Eq. 3.13}}{=} \begin{bmatrix} 1 & 0 & 0 & 0 & X_z & -X_y \\ 0 & 1 & 0 & -X_z & 0 & X_x \\ 0 & 0 & 1 & X_y & -X_x & 0 \end{bmatrix}, \tag{4.22}$$

finally resulting in a $2 \times 6$ Jacobian

$$\begin{bmatrix} \frac{\partial u_i}{\partial\boldsymbol{\mu}} \\ \frac{\partial v_i}{\partial\boldsymbol{\mu}} \end{bmatrix} = \begin{bmatrix} f_x\frac{1}{X_z} & 0 & -f_x\frac{X_x}{X_z^2} & -f_x\frac{X_xX_y}{X_z^2} & f_x\left(1+\frac{X_x^2}{X_z^2}\right) & -f_x\frac{X_y}{X_z} \\ 0 & f_y\frac{1}{X_z} & -f_y\frac{X_y}{X_z^2} & -f_y\left(1+\frac{X_y^2}{X_z^2}\right) & f_y\frac{X_xX_y}{X_z^2} & f_y\frac{X_x}{X_z} \end{bmatrix}, \tag{4.23}$$

describing the influence of a small increment around the current pose on the projected point $\mathbf{p}_i$. Given all necessary mathematical expressions, Equation 4.17 may be solved for the motion-update vector $\boldsymbol{\mu}$ using a simple least-squares solution

$$\boldsymbol{\mu} = \mathbf{J}^{\dagger}\mathbf{e}\,, \tag{4.24}$$

with $\mathbf{J}^{\dagger} = \left[\mathbf{J}^T\mathbf{J}\right]^{-1}\mathbf{J}^T$ being the pseudo-inverse. However, as commonly known, least-squares solutions are not robust to outliers. As in this case gross outliers are very likely to occur due to noise, occlusions or falsely matched points during the perpendicular search, solving the above equation system in a least-squares manner is not appropriate, as the distribution of the error vector $\mathbf{e}$ is not Gaussian at all. Therefore, a robust estimation for $\boldsymbol{\mu}$ is performed, again using *IRLS* performing

$$\boldsymbol{\mu}' = \underset{\boldsymbol{\mu}}{\text{argmin}} \sum_i w_{\text{TUK}}\left(||\mathbf{e}_i||\right)||\mathbf{e}_i||^2 \tag{4.25}$$

each iteration, whereas the weights are again obtained by the use of a Tukey *M-Estimator*. A few examples for such an iterative pose update using nine iterations, respectively, are given in Figures 4.11, 4.12 and 4.13, respectively. The yellow pose depicts the pose $\mathbf{E}^*_{\mathcal{CW}}$ from the prior iteration, while the green pose marks the updated pose

$$\mathbf{E}_{\mathcal{CW}} = \exp\left(\boldsymbol{\mu}'\right) \mathbf{E}^*_{\mathcal{CW}} \,. \tag{4.26}$$

The white lines show the error vectors connecting the points on the model edges with the points found during the perpendicular search.

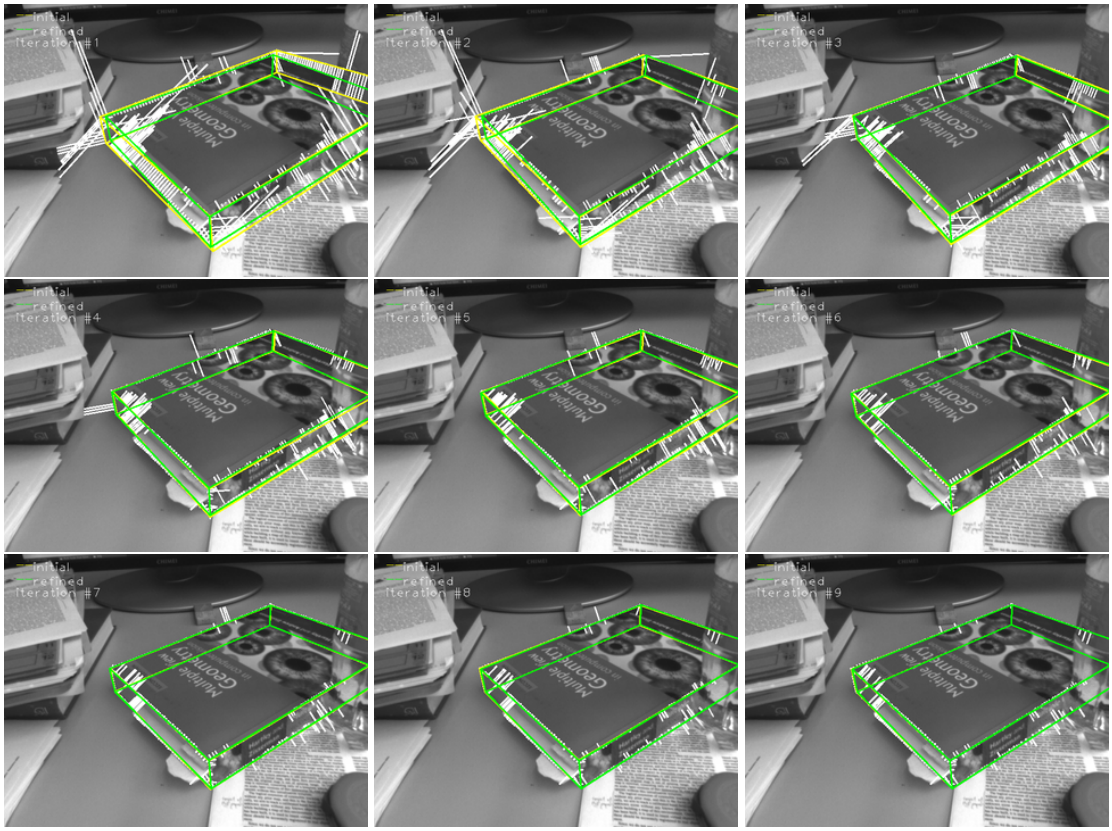



**Figure 4.11:** *An example of an iterative pose refinement for a solid object using nine update iterations is shown. The initial pose is shown in yellow, the refined pose in green. The error vectors determined during the perpendicular search are depicted by the white lines. Iteration count increases from the left to the right and from the top to the bottom.*

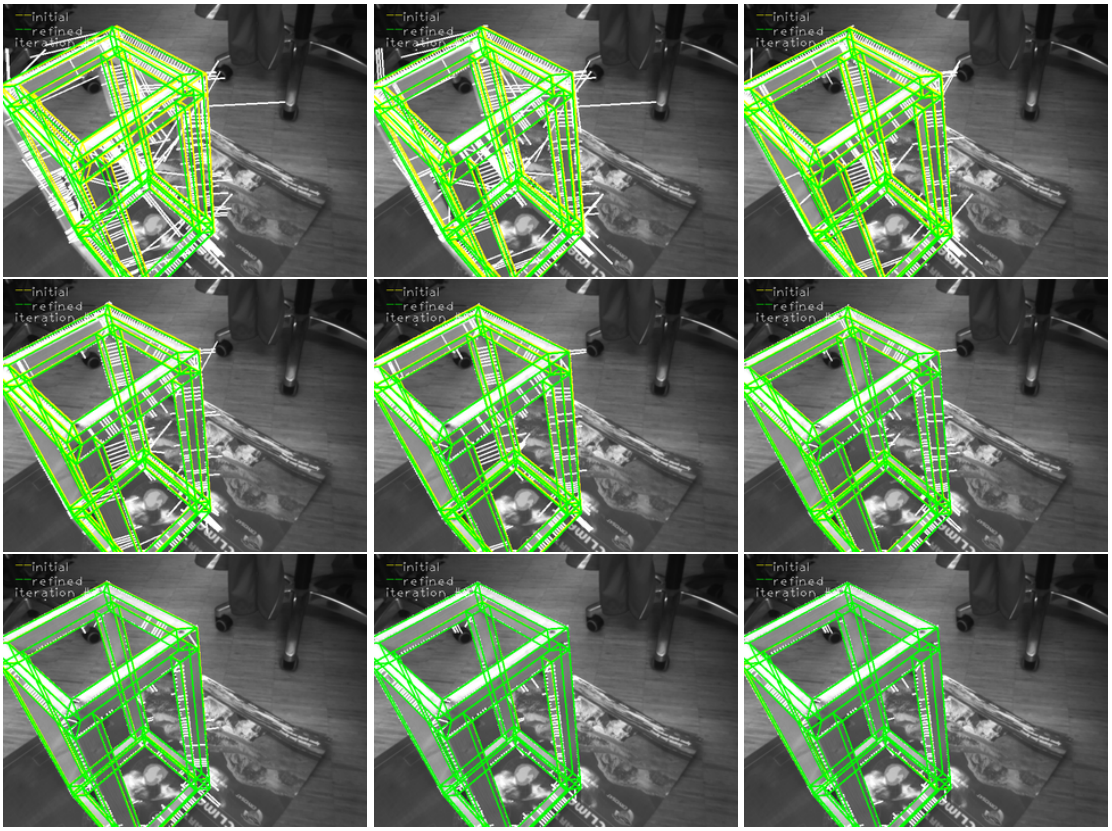



**Figure 4.12:** *An example of an iterative pose refinement for a wiry object using nine update iterations is shown. The initial pose is again shown in yellow, the refined pose in green. The error vectors determined during the perpendicular search are depicted by the white lines. Iteration count increases from the left to the right and from the top to the bottom.*

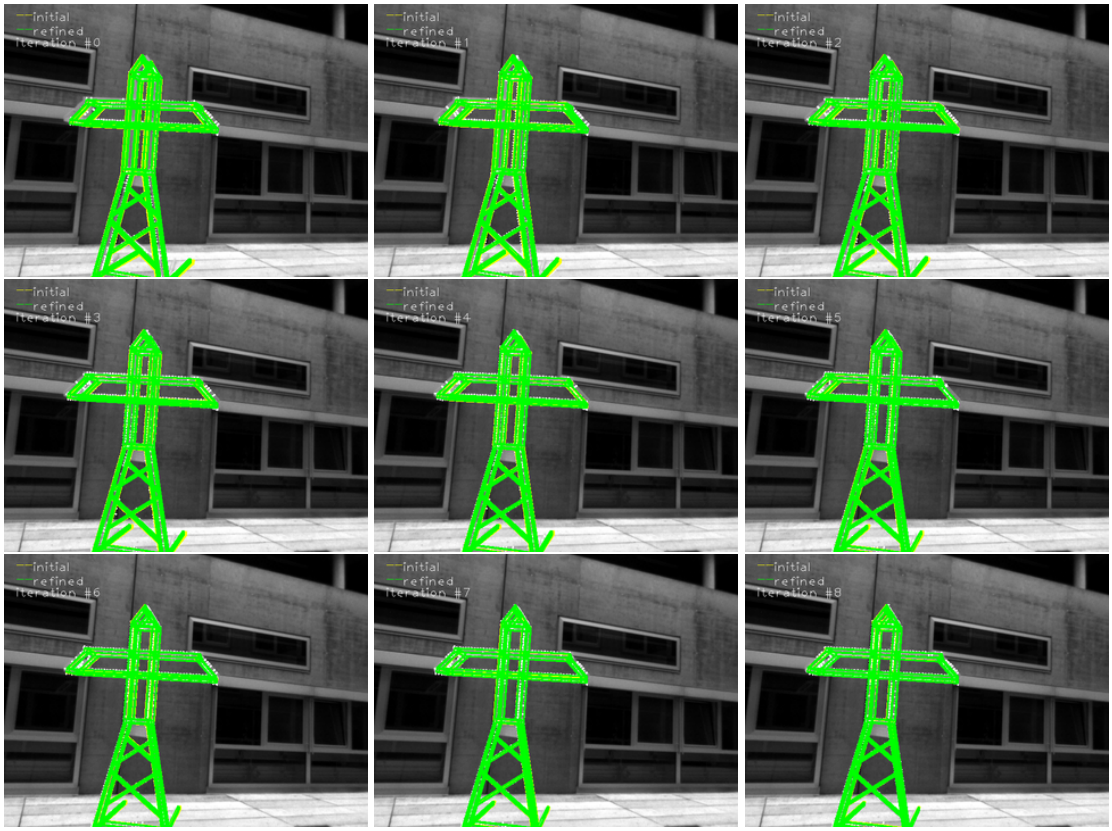



**Figure 4.13:** *An example of an iterative pose refinement for a power pylon model using nine update iterations is shown. The initial pose is shown in yellow, the refined pose in green. The error vectors determined during the perpendicular search are depicted by the white lines. Iteration count again increases from the left to the right and from the top to the bottom.*

# Chapter 5

# Experiments and Results

**Contents**



Several experiments with solid objects as well as wiry objects have been performed. Our implementation of online model-based multi-scale pose estimation runs in real time with 15-20 frames per second (*FPS*), while exploiting the computational resources of todays multi-core processors. We have evaluated our adoptions in terms of tracking accuracy by comparison to a ground-truth, acquired by an outside-in tracking system. All evaluation tasks are performed on a PC with an Intel® Core™ i5 2,66 GHz processor running Linux. Our implementation runs in *C++* and makes use of the Robotics Operating System (*ROS*) [1]. *ROS* is frequently used in robotics application to perform distributed computing, because it already cares about many common problems such as message passing.

## 5.1 Evaluation Setup

The accuracy of the algorithm has been evaluated by comparing the pose delivered by our system against an off-the-shelf outside-in tracking solution from A.R.T. GmbH[2]. The tracking system uses three A.R. tracking cameras and calculates 6 *DOF* positions of rigid arrangements of several markers – also called a target – with an accuracy of approximately $\pm$1mm. Such a target has been mounted onto the camera shown in Figure 5.1(a). Additionally, a stylus equipped with a target,

[1] `http://www.ros.org`

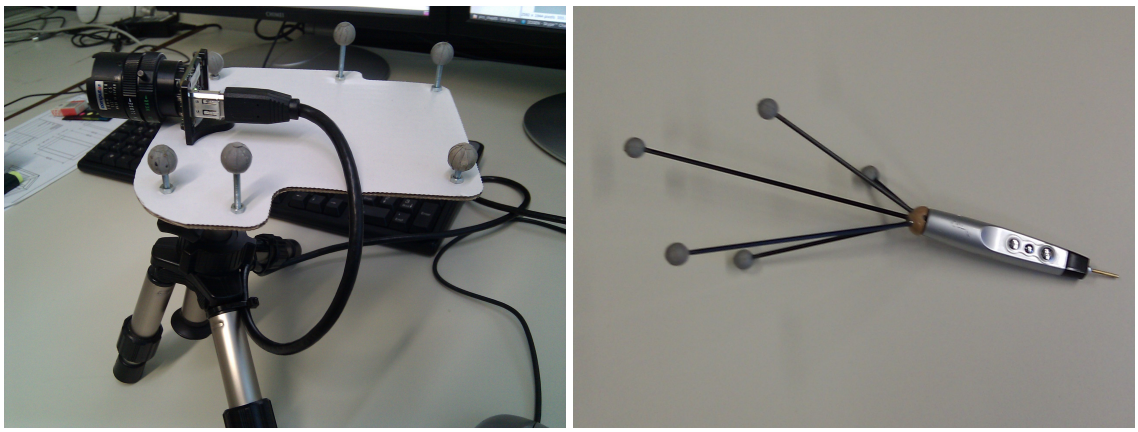
[2] `http://www.ar-tracking.de`

shown in Figure 5.1(b), was used to select the model points within the world coordinate frame of the A.R.T. system, and furthermore calculate a transformation between A.R.T. world coordinate frame and our tracking coordinate frame, by solving a 3D-similarity transformation of the form

$$\mathbf{X}_{left} = s\left(\mathbf{R}\mathbf{X}_{right} + \mathbf{t}\right) \ , \tag{5.1}$$

where $\mathbf{X}_{left}$ are the 3D-model points selected in the A.R.T. world coordinate frame $\mathcal{A}$ using the stylus, and $\mathbf{X}_{right}$ are the model points defined within our tracking coordinate frame $\mathcal{W}$. Thus, the transformation from $\mathcal{W}$ to $\mathcal{A}$ can be written as

$$\mathbf{E}_{\mathcal{A}\mathcal{W}} = \begin{bmatrix} & \mathbf{R} & & s\mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{5.2}$$



**(a)** Camera equipped with tracking target

**(b)** Stylus equipped with tracking target

**Figure 5.1:** *The camera in (a) has been equipped with a tracking target consisting of six markers, which allows us to retrieve a ground-truth for the camera's pose. A stylus in (b) has been used to select 3D-model points in the A.R.T world coordinate frame and subsequently calculate a transformation from our tracking coordinate frame to the A.R.T coordinate frame.*

Moreover, the transformation $\mathbf{E}_{\mathcal{C}\mathcal{T}}$ between the camera coordinate frame $\mathcal{C}$ and the target's coordinate frame $\mathcal{T}$, located in one of the six markers, has been calculated from 600 samples acquired during a qualitatively well initialized tracking sequence. As we had to experimentally estimate the fixed translation $\mathbf{E}_{\mathcal{C}\mathcal{T}}$, several hundred samples were used, however, we still impose a small systematic error, especially in the translational component. This is the reason for the many qualitative results shown throughout this evaluation, as these images illustrate the accuracy.

Given $\mathbf{E}_{\mathcal{CT}}$ allows us to compare the camera's world pose $\mathbf{E}_{\mathcal{CW}}^{\text{our}}$ calculated by our implementation, to the ground-truth

$$\mathbf{E}_{\mathcal{CW}}^{\text{art}} = \mathbf{E}_{\mathcal{CT}}\mathbf{E}_{\mathcal{TA}}\mathbf{E}_{\mathcal{AW}} \tag{5.3}$$

measured by the outside-in tracking system. The chain of transformations is schematically depicted in Figure 5.2. Finally, Figure 5.3(a) shows the room with the A.R.T. system setup, while Figure 5.3(b) shows a single A.R. tracking camera in action.

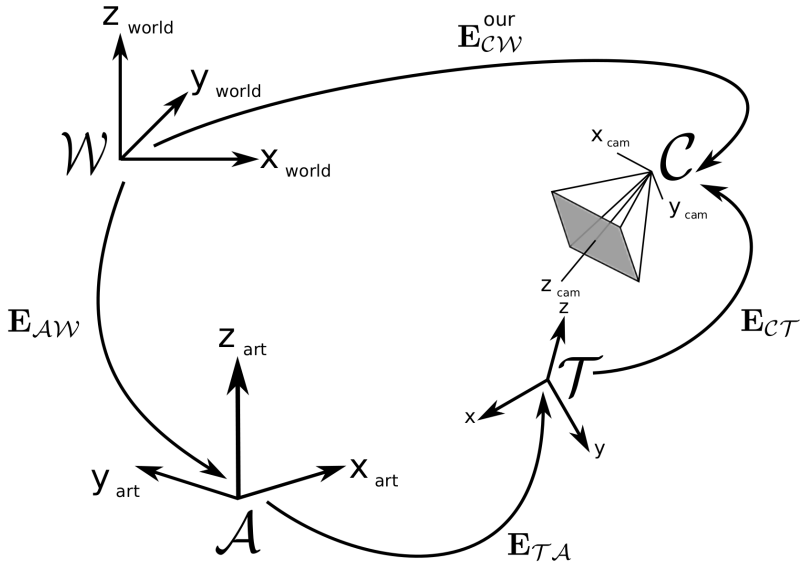



**Figure 5.2:** *To compare our camera pose $\mathbf{E}_{\mathcal{CW}}^{our}$ to the ground-truth camera pose $\mathbf{E}_{\mathcal{CW}}^{art}$ three different transformations have to be combined according to Equation 5.3. $\mathbf{E}_{\mathcal{AW}}$ is determined using a similarity transform, $\mathbf{E}_{\mathcal{TA}}$ is delivered by the A.R. tracking system, and $\mathbf{E}_{\mathcal{CT}}$ is calculated from 600 samples during a qualitatively good tracking sequence.*

## 5.2 Evaluation

Tracking accuracy is evaluated by comparing the camera's pose, determined by our implementation, against ground-truth supplied by an outside-in tracking system. Additionally, we compare the accuracy of a standard *PTAM* implementation against our implementation. Both approaches are initialized metrically, but our implementation exploits additional model information during initialization and tracking. Several different objects are used, including solid and wiry ones. For both translational and rotational error, scalar error measures are used. While for the translational error the Euclidean distance

$$t_{err} = ||\mathbf{t} - \mathbf{t}_{true}|| \tag{5.4}$$

**(a)** Evaluation setup **(b)** A.R. tracking camera

**Figure 5.3:** *The evaluation setup consisting of three A.R. tracking cameras is shown in (a). Only two of the three cameras are visible in the picture, while a single tracking camera in action is shown in (b).*

between the two camera centers is used, an attitudinal error measure based on a quaternion representation has been chosen. A quaternion is defined as $\mathbf{q} = \begin{bmatrix} \boldsymbol{\rho}^T & q_w \end{bmatrix}^T$, with $\boldsymbol{\rho} = \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}^T = \hat{\mathbf{e}} \sin\left(\frac{\theta}{2}\right)$ and $q_w = \cos\left(\frac{\theta}{2}\right)$, where $\hat{\mathbf{e}}$ is the axis of rotation and $\theta$ is the angle of rotation [74]. Thus, the angular difference between two quaternions $\mathbf{q}_{true}$ and $\mathbf{q}$ can be calculated from $q_w$ as

$$\theta_{err} = 2 \arccos\left( ||\mathbf{q}_{true}^{-1}\mathbf{q}||_w \right) , \tag{5.5}$$

where $|| \cdot ||_w$ is an operation that first normalizes the quaternion resulting from the quaternion multiplication, and then extracts $q_w$.

The parameter settings throughout this evaluation are the same. While the significant area threshold $t = 0.2$, the convolution kernel used to calculate the gradient information has $\sigma = 0.7$. The parameter chosen for the perpendicular search strategy is $\lambda = 0.8$, and the expansion of the search region is set to 30% of the maximum expansion of the bounding box. $\tau = 0.025$ is used during z-buffer thresholding.

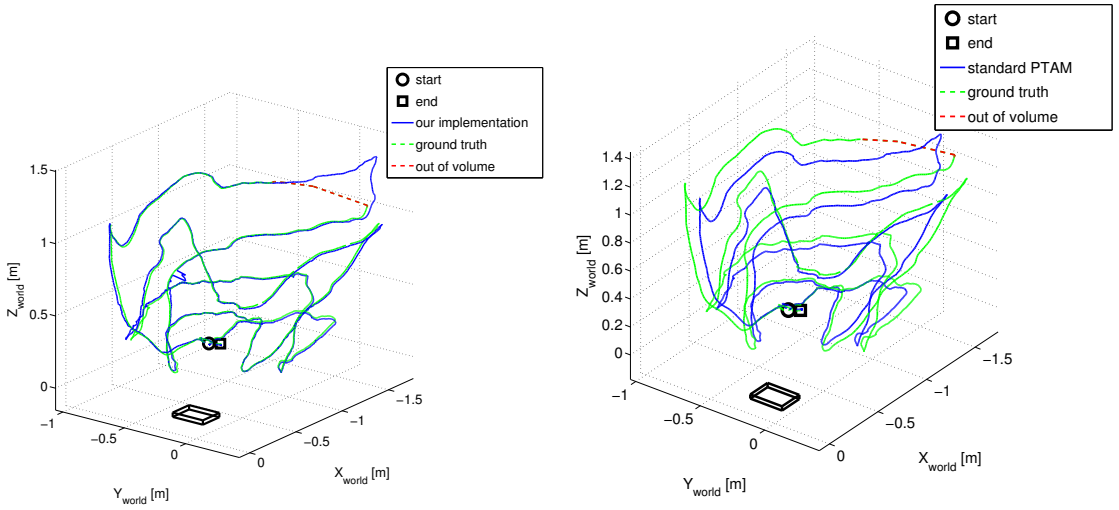
### 5.2.1 Solid Objects

First, results using a solid object are presented. The results for a two and a half minute long tracking sequence of a book are listed in Table 5.1.

| | **translational error [m]** | | | **rotational error [deg]** | | |
|---|---|---|---|---|---|---|
| | $\mu$ | $\pm\sigma$ | max | $\mu$ | $\pm\sigma$ | max |
| our implementation | 0,017 | 0,008 | 0,107 | 1,0 | 0,4 | 4,8 |
| std. *PTAM* | 0,061 | 0,034 | 0,177 | 1,2 | 0,4 | 3,7 |
| our implementation w/o global BA | 0,017 | 0,008 | 0,064 | 1,0 | 0,4 | 3,2 |

**Table 5.1:** *The accuracy of the standard PTAM implementation is compared against our implementation, where additional model information is used during initialization as well as during tracking. Furthermore, the effect of disabling global BA has been evaluated.*

From the results listed in Table 5.1 one could assume, that our implementation significantly outperforms standard *PTAM*. This, however, is mainly due to the fact that the relative motion estimation between the first and second keyframe estimated from the piecewise planar environment using [44] is erroneous. Thus, a model-based refinement during initialization is essential. When the model information is incorporated into the initialization process to refine the first two camera poses for the first and the second keyframe, respectively, then standard *PTAM* is almost as accurate as our implementation. The accuracy gained when constantly refining the pose during tracking using the model is quite small as long as the initialization is good, and is not important until being very close to the object. Furthermore, looking forward to use our implementation on a *UAV*, we tried to save computational resources by disabling global bundle adjustment, as this is the computationally most expensive task which grows rapidly with increasing map size [24]. We show that our implementation is able to track very accurately, even when disabling global optimization. However, this only holds as long as we navigate within a local environment around the object of interest, in this case within the vicinity of the book. In Figure 5.4 we compare the camera's trajectory against the ground truth acquired by the A.R.T. system for both our and standard *PTAM* implementations. Additionally, the maximum translational as well as rotational error in our implementation occurring due to a bad refinement are shown in Figure 5.5. This happens because our implementation with all features enabled sometimes drops one or two frames, as we are operating at our computational limits, which is not the case when disabling global *BA*. Therefore, although looking contradictory, the accuracy during this experiment even improves when disabling global *BA*, as shown in Table 5.1.




**(a)** Our implementation

**(b)** Standard *PTAM*

**Figure 5.4:** *The camera's trajectory of our vision based system using additional model information is compared to ground-truth obtained from an outside-in tracking system. High tracking accuracy with a mean deviation of only 1,7 centimeters is achieved over a 2:30 minute long tracking sequence of a solid object, namely a book. In the upper right corner, marked in red, the camera moved outside the tracking volume of the reference system, which has been considered during evaluation.*

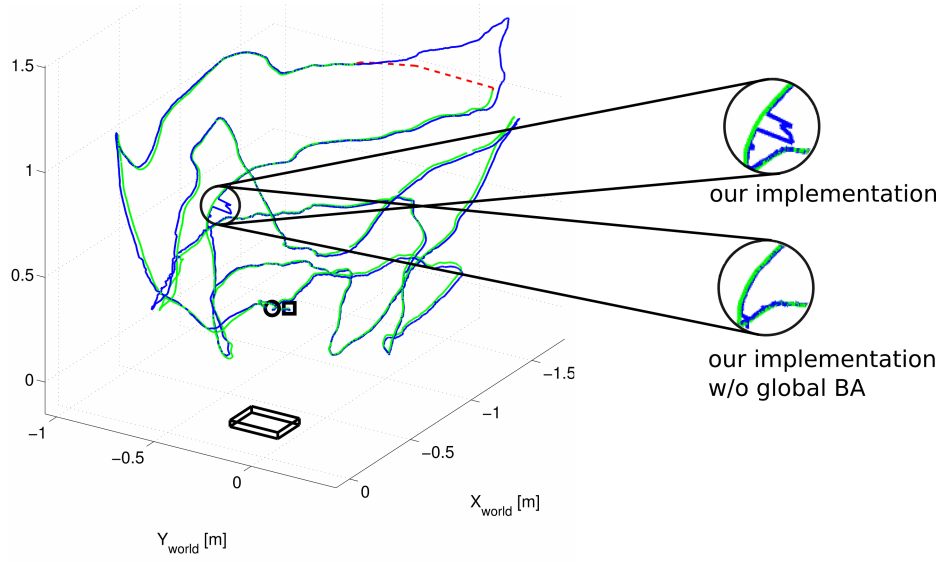



**Figure 5.5:** *When global BA is enabled additional computational resources are needed, which sometimes leads to a drop of a few frames. Thus, the refinement component can fail to refine the pose properly, which is not happening when global BA is turned off.*

Furthermore, Figures 5.8 and 5.9 show qualitative results acquired during the tracking sequence used for evaluation. We first show the problems that occur, when refinement is not used during initialization, followed by some qualitative results of our implementation. The high accuracy of our implementation is evident, especially when being close to the object. This is in our case of high importance, as we are working towards an inspection system, where closeup views of parts of the object will become important. Furthermore, our implementation can handle smaller objects (see Figure 5.6), as well as robustly deal with partial occlusions, not only during tracking but during refinement as well, as depicted in Figure 5.7.

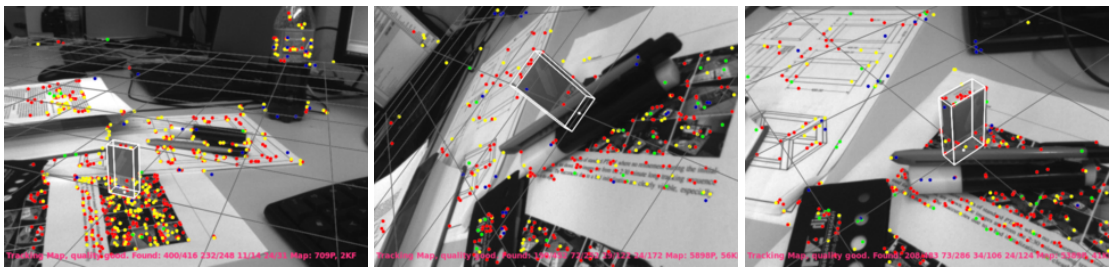


**Figure 5.6:** *A small solid object, in this case a box for staples, is successfully tracked.*

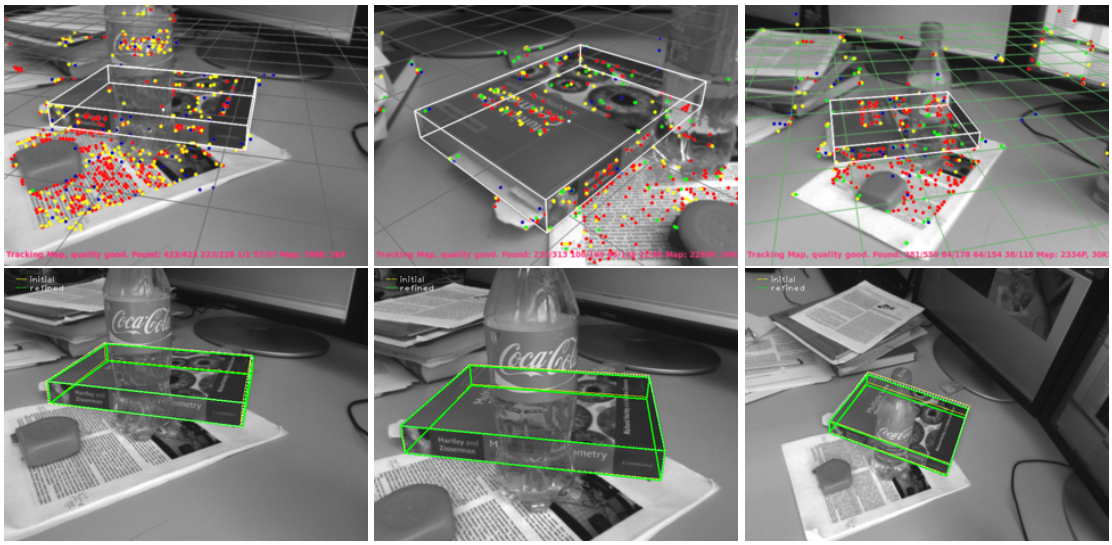


**Figure 5.7:** *The object tracking process is very robust against partial occlusions, and severe scale changes. The object is tracked using the entire 3D-information available, and thus when parts are occluded enough information from the background is available to keep up tracking. However, not only the tracking itself is robust to occlusions, but the refinement is as well. This is shown in the bottom row of this figure. The refined pose is shown in green.*

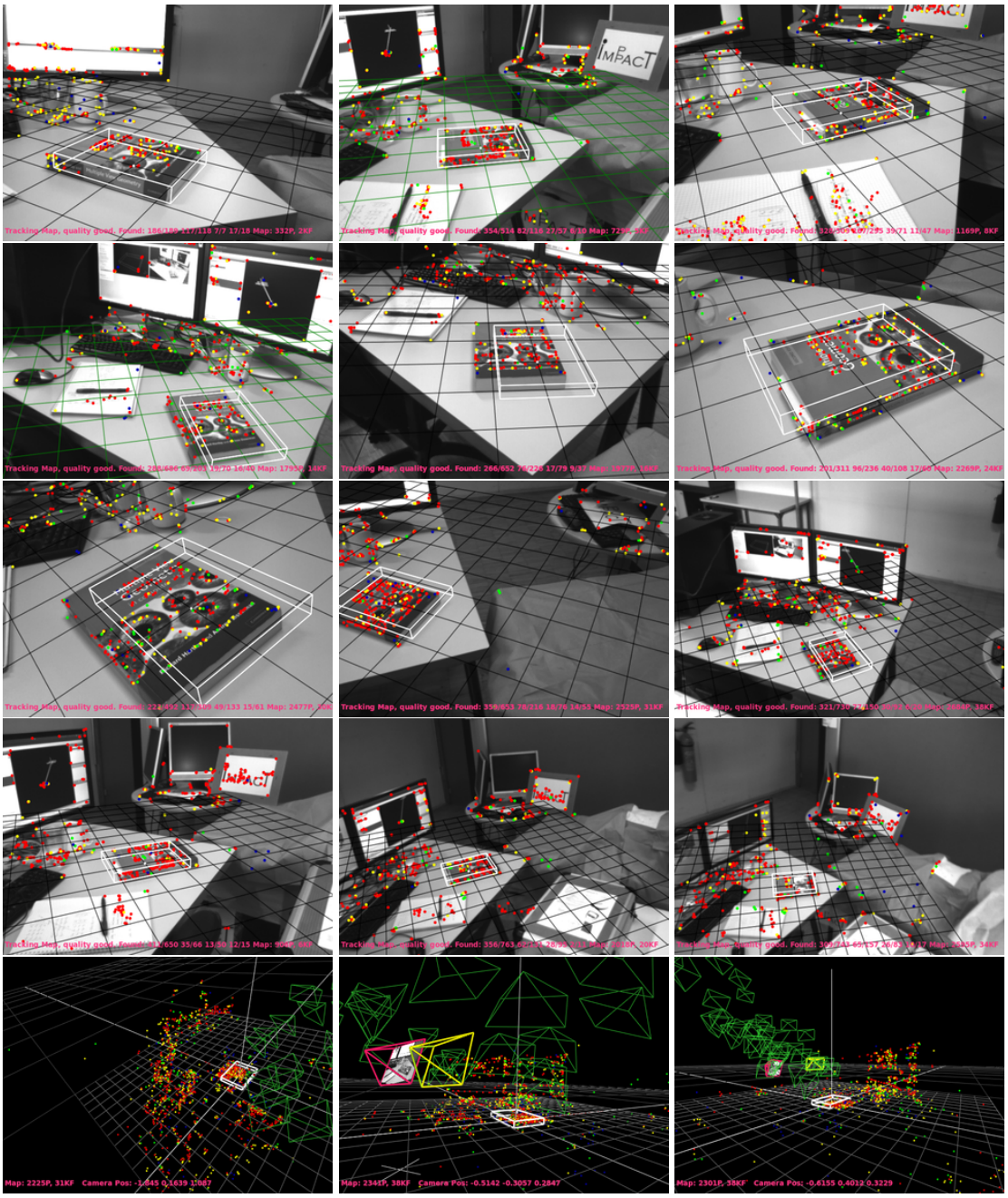


**Figure 5.8:** *A few qualitative results of standard PTAM, where no refinement during the initialization has been used, are shown here. The images have been taken from the 2:30 minute long tracking sequence used for evaluation. The inaccuracies due to a bad initialization are clearly visible, especially when close to the object. The bottom row shows the final map.*

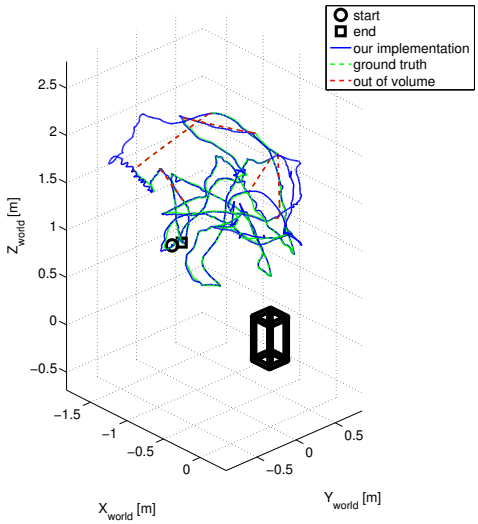
**Figure 5.9:** *A few qualitative results of our implementation are shown here. The high accuracy is visible, especially when being close to the object. Obtaining such highly accurate results is only possible when the initialization is excellent, which is assured by refining the first and second keyframe during initialization. The bottom row shows the final map.*

### 5.2.2 Wiry Objects

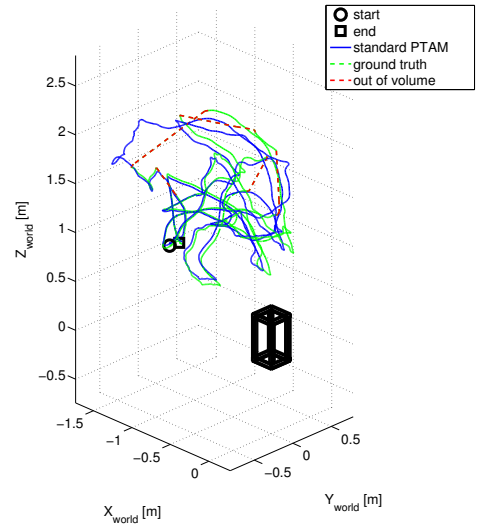
The same experiments performed for the solid object have been performed with a wiry object, namely the wiry-wooden box from Figure 4.3(c). The same observations made before hold for the results of the wiry object as listed in Table 5.2, except the maximum errors also occur when global *BA* is disabled. This is due to the fact that the computational time for refinement slightly increases with the number of model points. The accuracy strongly depends on a good initialization, achieved by using the refinement component during initialization. The camera trajectories of our and a standard *PTAM* implementation, compared to ground-truth, are shown in Figure 5.10. Although overall tracking accuracy slightly decreases compared to the solid object, the qualitative results depicted in Figure 5.12 show very satisfactory results.

| | **translational error [m]** | | | **rotational error [deg]** | | |
|---|---|---|---|---|---|---|
| | $\mu$ | $\pm\sigma$ | max | $\mu$ | $\pm\sigma$ | max |
| our implementation | 0,026 | 0,031 | 0,300 | 1,6 | 1,4 | 16,5 |
| std. *PTAM* | 0,069 | 0,029 | 0,320 | 1,4 | 1,3 | 11,3 |
| our implementation w/o global BA | 0,026 | 0,031 | 0,313 | 1,6 | 1,3 | 11,7 |

**Table 5.2:** *The accuracy of the standard PTAM implementation is compared against our implementation using the refinement component during initialization. Furthermore, the effect of disabling global BA on the accuracy of the tracking has been evaluated.*




**(a)** Our implementation




**(b)** Standard *PTAM*

**Figure 5.10:** *The camera's trajectory of our implementation has been compared to ground-truth obtained by an outside-in tracking system. Tracking accuracy with a mean deviation of 2,6 centimeters is achieved over a 2:00 minute long tracking sequence of a wiry object. The camera moved several times outside the tracking volume of the reference system, marked in red, which has been considered during evaluation.*

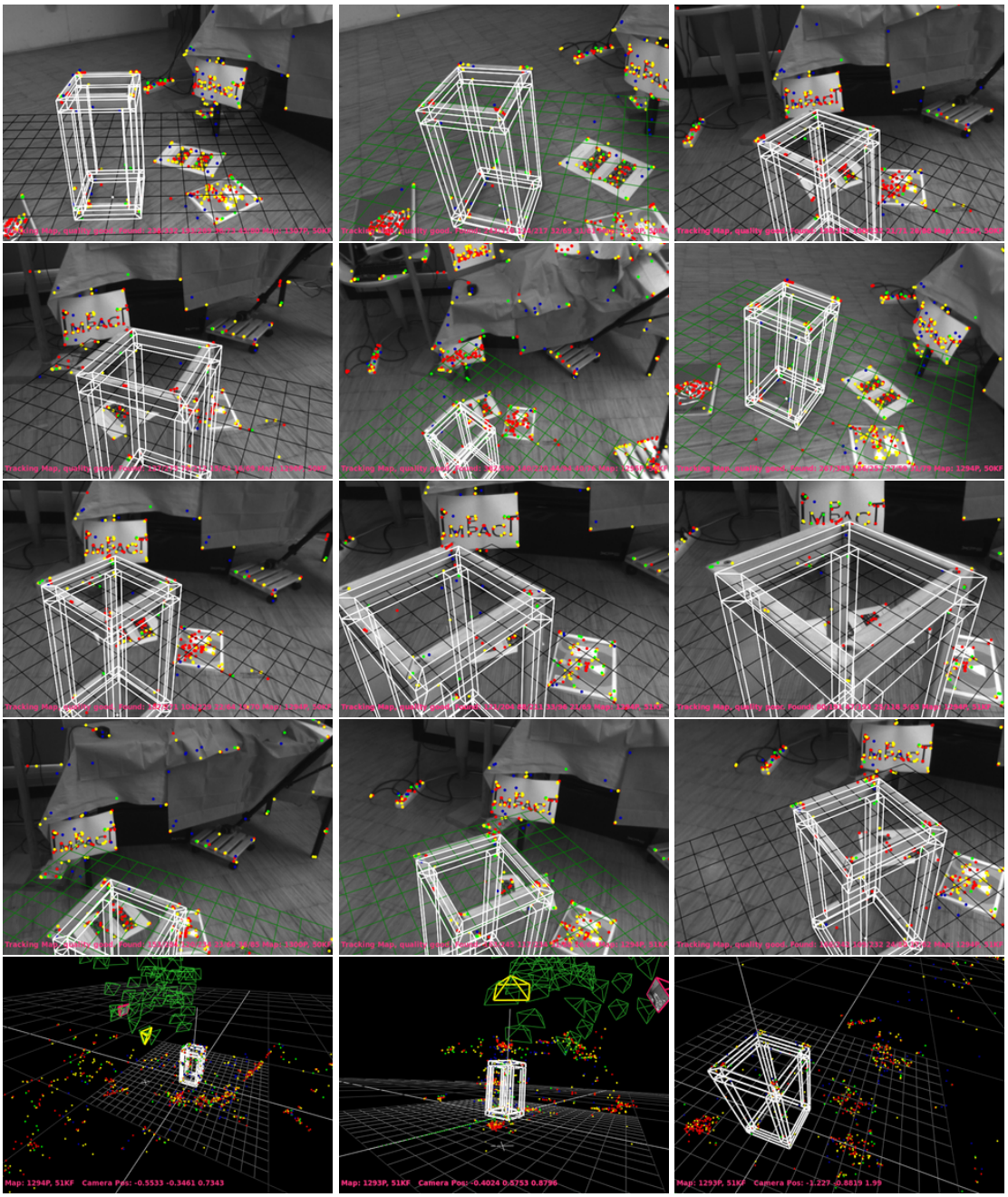


**Figure 5.11:** *A few qualitative results of standard PTAM are shown here. The inaccuracies due to a bad initialization are again clearly visible, especially when being close to the object. The bottom row shows the final map.*

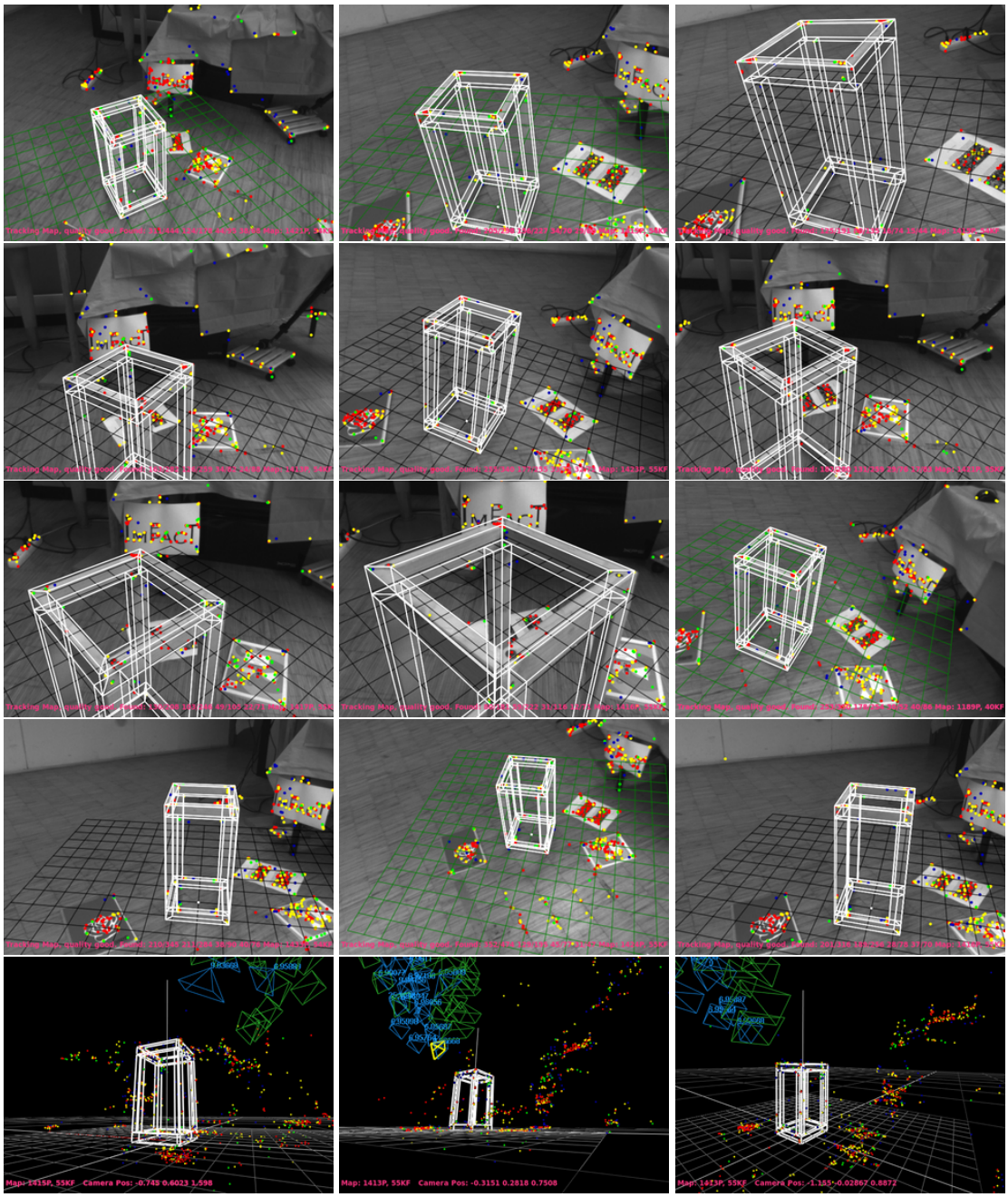


**Figure 5.12:** *A few qualitative results of our implementation are shown here. The high accuracy is visible, especially when being close to the object. The bottom row shows the final map.*

After having evaluated the accuracy for wiry objects as well, we want to show further qualitative results from two scenes where our hand-crafted wooden power pylon was used as object of interest. The results for both scenes are depicted in Figure 5.13 and Figure 5.14, respectively.

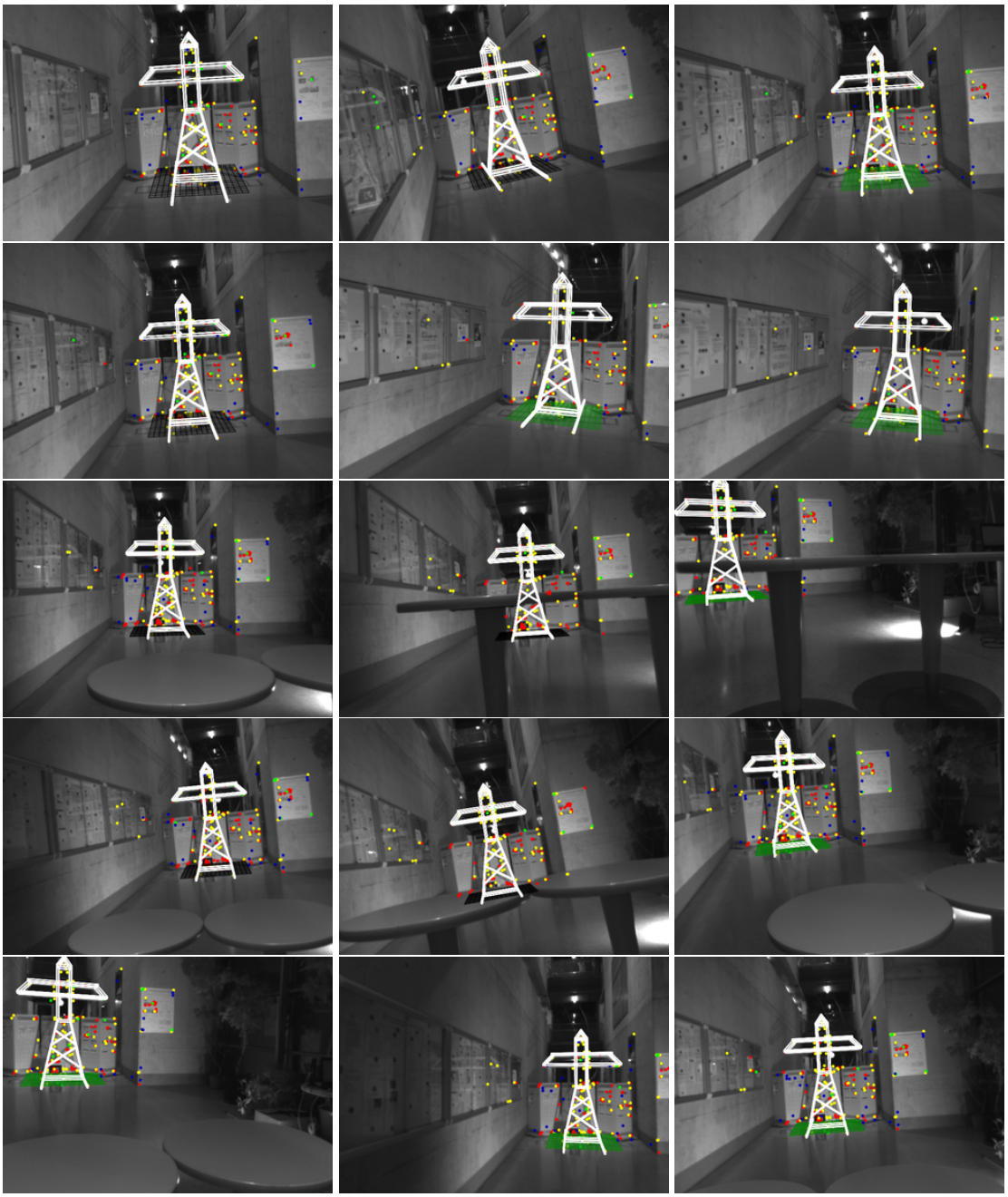


**Figure 5.13:** *A few qualitative results of our implementation are shown here for an indoor scene with the hand crafted wooden power pylon. Note the partial occlusions in the middle row.*

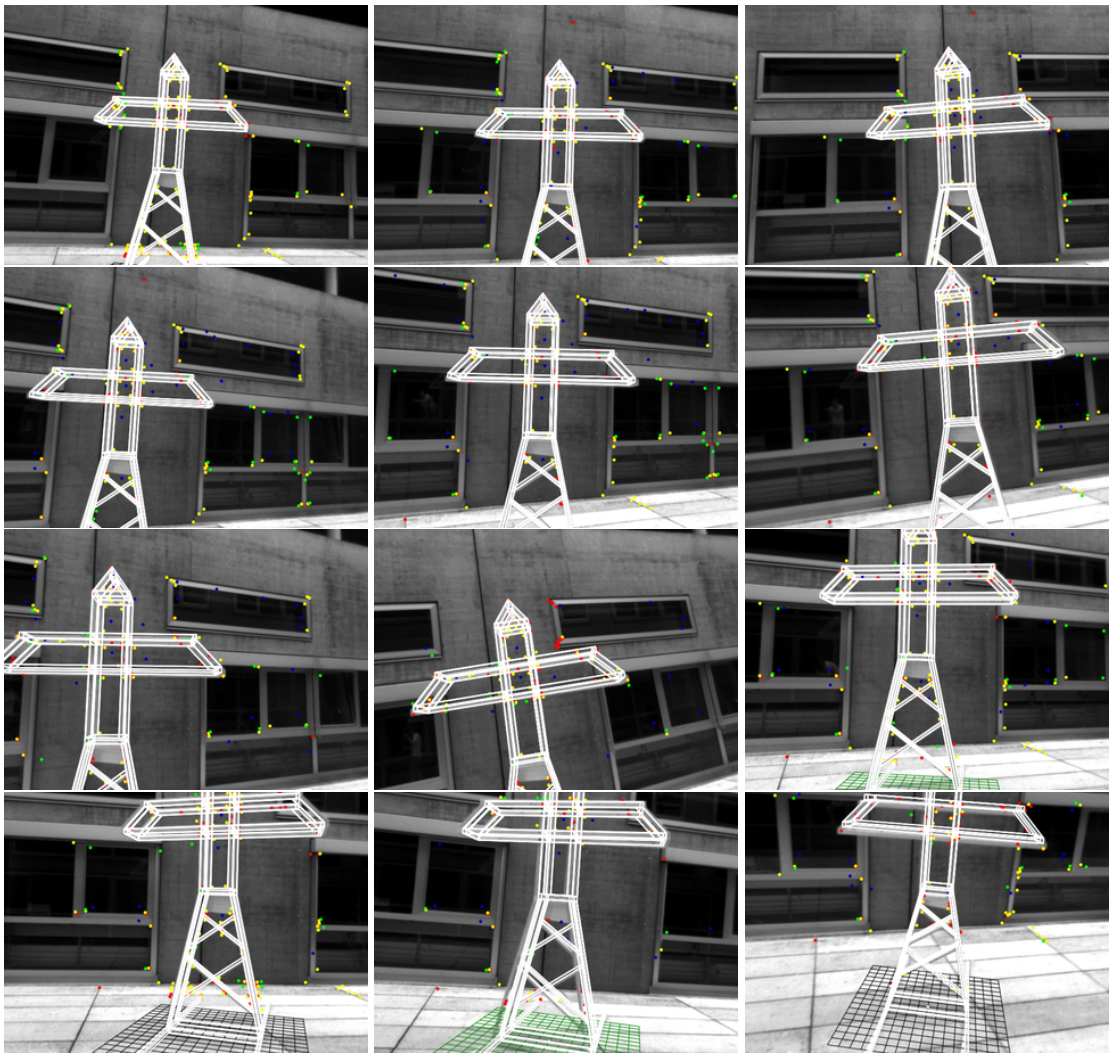


**Figure 5.14:** *A few qualitative results of our implementation are shown here for an outdoor scene with the hand-crafted wooden power pylon. In some cases the alignment is not perfect, such as depicted in the last row. This is caused by the very thin structure and the fact that despite the small search region used during the first iteration, too many wrong image points are determined during the perpendicular search.*

### 5.2.3 Overcoming tracking failures

The main benefit of our implementation, however, is to combine *PTAM*'s tracking procedure with model-based tracking in order to keep up tracking in cases where standard *PTAM* would lose it. This is especially the case when capturing closeup views of parts of the object as needed during inspection tasks. To track the model in a frame-by-frame manner, we simply use the same refinement component used during initialization. However, to ensure real-time operation, we perform

less pose update iterations, and only determine model point visibility in every fifth frame. We have simulated such closeup views using the wiry wooden box, and compared the standard *PTAM* implementation against our implementation with model-based refinement and model-based recovery. As can be seen from the results summarized in Table 5.3, loss of tracking could always be prevented for this 2:20 minute long video sequence when using model-based tracking during failure recovery. This is not true for the standard *PTAM* implementation where tracking is completely lost in 20,9%. Moreover, we could not only keep up tracking during longer periods of time, but even avoid further loss of frames by the use of model-based tracking. Qualitative model-tracking results from Figure 5.15 show very promising results. More results can be seen in the video, available online[3].

| | #frames | #frames lost | frames lost [%] | #frames recovered | frames recovered [%] |
|---|---|---|---|---|---|
| standard *PTAM* | 4110 | 860 | 20,9 | 0 | 0,0 |
| our implementation | 4110 | 272 | 6,6 | 272 | 100,0 |

**Table 5.3:** *A model-based tracking using our refinement-component takes over tracking in cases where standard PTAM tracking fails. This especially is the case when capturing closeup views of an object to inspect. During a 2:20 minute long tracking sequence including several closeups we could always recover from loss of tracking by switching to a model-based tracking stage. Furthermore, the total number of frames lost is far smaller, because by overcoming tracking failures the need to attempt a total recovery could be avoided completely.*

## 5.3 Discussion

During the experiments for solid and wiry objects of different sizes we have shown that we are able to determine the camera's pose accurately by integrating the available prior knowledge of the model into *PTAM*. We use the model information to create a metrically correct map from scratch, and have shown that a refinement of the first two keyframes ensures a good initialization. An accurate initialization is the most important step towards an accurate *SLAM* system.

Beyond using the refinement component during initialization, we perform a pose refinement in each frame when the object occupies a significant area of the image, in order to improve accuracy. When too few map-points are detected in the image, tracking usually gets lost. This is the case for instance when capturing closeup views where the background has little texture. A possible scenario would be the inspection of an insulator at a power pylon with just sky in the background.

[3] `http://aerial.icg.tugraz.at`

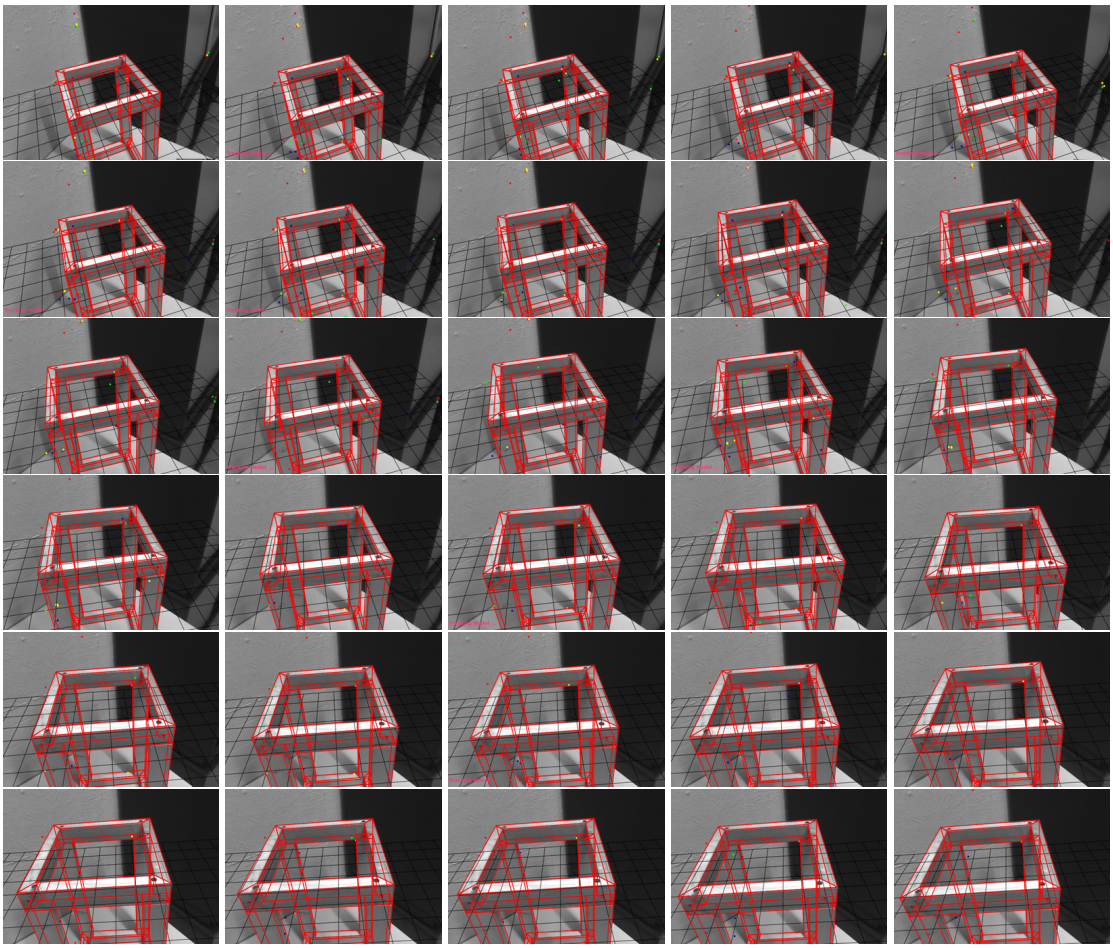


**Figure 5.15:** *When not enough features for tracking are available, we automatically switch to a model-based tracking stage in order to avoid tracking failure. Some qualitative results are shown here for thirty continuous frames.*

We showed that we can overcome such failures by seamlessly switching to a model-based tracking stage, and switch back to standard tracking if enough map-points are detected again.
A drawback of our current implementation is that during purely model-based tracking we are quite prone to fast motions, as a fast motion results in a rather big shift of the model in the image when being close to it. Besides fast motions, the refinement component can fail if too strong edges parallel to the object's edges are apparent in the scene. This is not an issue during a refinement when the prior pose estimate to refine is delivered by *PTAM*'s standard tracking procedure, however, becomes a serious problem when only performing edge-based model tracking. This, however, can be resolved by predicting the camera's transformation using the *IMU* as for instance demonstrated in [48, 49].

# Chapter 6

# Summary and Conclusion

**Contents**



During this work we have presented our online model-based multi-scale pose estimation approach, which builds upon a state-of-the-art *VSLAM* implementation. We integrated prior knowledge of our object of interest not only during initialization, but used this information during the entire tracking procedure. To conclude our work, we summarize our contributions and present an outlook to future work.

## 6.1 Conclusion

We have shown that we are capable of delivering accurate metric pose estimates in real time for different sizes and types of objects, including solid and wiry models. We are able to estimate the camera's pose over multiple scales: When far away from the object, or in cases where the object is not even present in the image, the pose is estimated using the standard keypoint-based tracking procedure. When close to the object the additional model knowledge is used to refine this pose estimate. We have learned that refining the pose is most important in terms of accuracy during the initialization step. Furthermore, the model information has been used for model-based tracking to overcome failures in cases where tracking using the background information failed. This especially avoids tracking failures when capturing closeup views of the object, for instance during inspection tasks. Our implementation only requires a single camera, and steps towards using this implementation on a *UAV* with limited computational resources have been taken. This

especially involves the disabling of the computationally expensive global map optimization, while retaining accuracy. Furthermore, the entire algorithm has been realized as a stand-alone *ROS*-node, which allows for distributed computing as often needed in mobile robotics applications.

## 6.2 Future Work

With our implementation we are able to obtain accurate metric pose estimates in real time solely based on images. However, when thinking of a *UAV* other sensors are available, namely *GPS* and *IMU* data. For future work we therefore suggest a robust fusion of all data available, including a pose prediction based on gyroscopic measurements. The *IMU* typically operates at far higher rates than a camera and thus could be used to accurately propagate the pose from one frame to the next. We think that incorporating *IMU* data especially helps to cope with fast camera movements and rotations during model-based tracking, where our implementation currently lacks accuracy. Other than that, the refinement component now performs a sequential perpendicular search for each edge point, but this could be performed in parallel on a Graphics Processing Unit (*GPU*). *GPU*s are not yet available in such small form factors that they could be used on a *UAV*, however, we think that this is going to be the case in near future.

# Appendix A

# Acronyms

## List of Acronyms

| | |
|---|---|
| *AR* | Augmented Reality |
| *BA* | Bundle Adjustment |
| *CAD* | Computer Aided Design |
| *DLT* | direct linear transformation |
| *DoG* | Difference of Gaussian |
| *EKF* | Extended Kalman Filter |
| *EM* | Expectation Maximization |
| *FAST* | Features from Accelerated Segment Test |
| *FPS* | frames per second |
| *GMM* | Gaussian Mixture Model |
| *GPS* | Global Positioning System |
| *GPU* | Graphics Processing Unit |
| *IMU* | Inertial Measurement Unit |
| *IRLS* | Iteratively Reweighted Least-Squares |
| *LM* | Levenberg-Marquardt |
| *LS* | Least-Squares |
| *MLE* | maximum likelihood estimation |
| *MSER* | Maximally Stable Extremal Region |
| *OSM* | Open Street Map |
| *PTAM* | Parallel Tracking And Mapping |
| *P3P* | Perspective three Point problem |

| | |
|---|---|
| *PnP* | Perspective $n$ Point problem |
| *RANSAC* | RANdom SAmple Consensus |
| *ROS* | Robotics Operating System |
| *SfM* | Structure from Motion |
| *SIFT* | Scale Invariant Feature Transform |
| *SLAM* | Simultaneous Localization And Mapping |
| *SSD* | Sum of Squared Differences |
| *SVD* | Singular Value Decomposition |
| *SUSAN* | Smallest Univalue Segment Assimilating Nucleus |
| *UAV* | Unmanned Aerial Vehicle |
| *VSLAM* | Visual SLAM |
| *VRML* | Virtual Reality Modeling Language |
| *WLS* | Weighted Least-Squares |

# Bibliography

[1] F. H. Moffitt and E. M. Mikhail, *Photogrammetry*. NY: Harper & Row, first ed., 1980.

[2] Z. Zhang, "Flexible Camera Calibration by Viewing a Plane from Unknown Orientations," in *Proceedings 7th International Conference on Computer Vision*, (Kerkyra, GR), pp. 666–673, 1999.

[3] J. Heikkilä and O. Silvén, "A Four-step Camera Calibration Procedure with Implicit Image Correction," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (San Juan, PR), pp. 1106–1112, 1997.

[4] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, GB: Cambridge University Press, second ed., 2004.

[5] A. Leonardis and H. Bischof, "Dealing with Occlusions in the Eigenspace Approach," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (San Francisco, CA), pp. 453–458, 1996.

[6] M. Byne and J. Anderson, "A CAD-Based Computer Vision System," *Image and Vision Computing*, vol. 16, no. 8, pp. 533–539, 1998.

[7] H. Bischof and A. Leonardis, "Robust Recognition of Scaled Eigenimages Through a Hierarchical Approach," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Santa Barbara, CA), pp. 664–670, 1998.

[8] H. Borotschnig, L. Paletta, M. Prantl, and A. Pinz, "Appearance-Based Active Object Recognition," *Image and Vision Computing*, vol. 18, no. 9, pp. 715–727, 2000.

[9] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[10] J. Pilet, V. Lepetit, and P. Fua, "Real-Time Nonrigid Surface Detection," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (San Diego, CA), pp. 822–828, 2005.

[11] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded Up Robust Features," in *Proceedings 9th European Conference on Computer Vision*, (Graz, AT), pp. 404–417, 2006.

[12] V. Lepetit and P. Fua, "Keypoint Recognition Using Randomized Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 9, pp. 1465–1479, 2006.

[13] S. Hinterstoisser, S. Benhimane, and N. Navab, "N3M: Natural 3D Markers for Real-Time Object Detection and Pose Estimation," in *Proceedings 11th International Conference on Computer Vision*, (Rio de Janeiro, BR), pp. 1–7, 2007.

[14] O. D. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*. London, GB: The MIT Press, 1993.

[15] S. Finsterwalder and W. Scheufele, *Das Rückwärtseinschneiden im Raum*. Berlin, DE: Verlag der Königlichen Akademie der Wissenschaften, 1903.

[16] D. F. Dementhon and L. S. Davis, "Model-Based Object Pose in 25 Lines of Code," *International Journal of Computer Vision*, vol. 15, no. 1, pp. 123–141, 1995.

[17] G. Schweighofer and A. Pinz, "Robust Pose Estimation from a Planar Target," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 2024–2030, 2006.

[18] G. Schweighofer and A. Pinz, "Globally Optimal O(n) Solution to the PnP Problem for General Camera Models," in *Proceedings 19th British Machine Vision Conference*, (Leeds, GB), pp. 1–10, 2008.

[19] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Communications of Association for Computing Machinery*, vol. 24, no. 6, pp. 381–395, 1981.

[20] P. David, D. F. DeMenthon, R. Duraiswami, and H. Samet, "SoftPOSIT: Simultaneous Pose and Correspondence Determination," in *Proceedings 7th European Conference on Computer Vision*, (Copenhagen, DK), pp. 79–95, 2002.

[21] P. David, D. F. DeMenthon, R. Duraiswami, and H. Samet, "Simultaneous Pose and Correspondence Determination Using Line Features," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Madison, WI), pp. 424–431, 2003.

[22] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Pose Priors for Simultaneously Solving Alignment and Correspondence," in *Proceedings 10th European Conference on Computer Vision*, (Marseille, FR), pp. 405–418, 2008.

[23] G. Bleser, H. Wuest, and D. Stricker, "Online Camera Pose Estimation in Partially Known and Dynamic Scenes," in *Proceedings 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Santa Barbara, CA), pp. 56–65, 2006.

[24] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *Proceedings 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Nara, JP), pp. 225–234, November 2007.

[25] I. E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," in *Proceedings of the 1963 Spring Joint Computer Conference*, (Baltimore, MD), pp. 6–329, 1963.

[26] R. M. Haralick, C. Lee, K. Ottenberg, and M. Nölle, "Analysis and Solutions of the Three Point Perspective Pose Estimation Problem," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Maui, HI), pp. 592–598, 1991.

[27] L. Kneip, D. Scaramuzza, and R. Siegwart, "A Novel Parametrization of the Perspective-Three-Point Problem for a Direct Computation of Absolute Camera Position and Orientation," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Colorado Springs, CO), pp. 2969–2976, 2011.

[28] L. Quan and Z. Lan, "Linear N-Point Camera Pose Determination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 8, pp. 774–780, 1999.

[29] B. K. P. Horn, "Closed-Form Solution of Absolute Orientation Using Unit Quaternions," *Journal of the Optical Society of America*, vol. 4, no. 4, pp. 629–642, 1987.

[30] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour, "Closed-Form Solution of Absolute Orientation Using Orthonormal Matrices," *Journal of the Optical Society of America*, vol. 5, no. 7, pp. 1127–1135, 1988.

[31] W. R. Hamilton, *Elements of Quaternions.* London, GB: Longmans, Green, & Co, 1866.

[32] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Accurate Non-Iterative O(n) Solution to the PnP Problem," in *Proceedings 11th International Conference on Computer Vision*, (Rio de Janeiro, BR), pp. 1–8, 2007.

[33] D. Oberkampf, D. F. DeMenthon, and L. S. Davis, "Iterative Pose Estimation Using Coplanar Feature Points," *Computer Vision and Image Understanding*, vol. 63, no. 3, pp. 495–511, 1996.

[34] C.-P. Lu, G. D. Hager, and E. Mjolsness, "Fast and Globally Convergent Pose Estimation from Video Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 6, pp. 610–622, 2000.

[35] S. Gold and A. Rangarajan, "A Graduated Assignment Algorithm for Graph Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 4, pp. 377–388, 1996.

[36] S. Gold, A. Rangarajan, C.-P. Lu, S. Pappu, and E. Mjolsness, "New Algorithms for 2D and 3D Point Matching: Pose Estimation and Correspondence," *Pattern Recognition*, vol. 31, no. 8, pp. 1019–1031, 1998.

[37] T. K. Moon, "The Expectation-Maximization Algorithm," *IEEE Signal Processing Magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[38] M. Ulrich, C. Wiedemann, and C. Steger, "CAD-Based Recognition of 3D Objects in Monocular Images," in *Proceedings IEEE International Conference on Robotics and Automation*, (Kobe, JP), pp. 1191–1198, 2009.

[39] W. E. L. Grimson and T. Lozano-Pérez, "Model-Based Recognition and Localization from Sparse Range or Tactile Data," *International Journal of Robotics Research*, vol. 3, no. 3, pp. 3–35, 1984.

[40] R. B. Fisher, "Performance Comparison of Ten Variations on the Interpretation-Tree Matching Algorithm," in *Proceedings 3rd European Conference on Computer Vision*, (Stockolm, SE), pp. 507–512, 1994.

[41] S. Holzer, S. Hinterstoisser, S. Ilic, and N. Navab, "Distance Transform Templates for Object Detection and Pose Estimation," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Miami Beach, FL), pp. 1177–1184, 2009.

[42] G. Borgefors, "Distance Transformations in Digital Images," *Computer Vision, Graphics and Image Processing*, vol. 34, no. 3, pp. 344–371, 1986.

[43] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique With an Application to Stereo Vision," in *International Joint Conference on Artificial Intelligence*, (Vancouver, CA), pp. 674–679, 1981.

[44] O. D. Faugeras and F. Lustman, "Motion and Structure from Motion in a Piecewise Planar Environment," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 2, no. 33, pp. 485–508, 1988.

[45] A. J. Davison, "Real-Time Simultaneous Localisation and Mapping with a Single Camera," in *Proceedings 9th International Conference on Computer Vision*, (Nice, FR), pp. 1403–1410, 2003.

[46] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[47] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," tech. rep., University of North Carolina at Chapel Hill, 1995.

[48] G. Klein and T. Drummond, "Robust Visual Tracking for Non-Instrumented Augmented Reality," in *Proceedings 2th IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Tokyo, JP), pp. 113–122, 2003.

[49] G. Reitmayr and T. Drummond, "Going Out: Robust Model-Based Tracking for Outdoor Augmented Reality," in *Proceedings 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Santa Barbara, CA), pp. 109–118, 2006.

[50] T. Drummond and R. Cipolla, "Visual Tracking and Control Using Lie Algebras," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Fort Collins, CO), pp. 652–657, 1999.

[51] D. Nistér, O. Naroditsky, and J. Bergen, "Visual Odometry," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Washington, DC), pp. I–652, 2004.

[52] V. Kyrki and D. Kragic, "Tracking Rigid Objects Using Integration of Model-Based and Model-Free Cues," *Machine Vision and Applications*, vol. 22, no. 1, pp. 323–335, 2011.

[53] T. Drummond and R. Cipolla, "Real-Time Visual Tracking of Complex Structures," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 932–946, 2002.

[54] P. J. Huber, "Robust Estimation of a Location Parameter," *Annals of Mathematical Statistics*, vol. 35, no. 1, pp. 73–101, 1964.

[55] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part I," *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[56] H. Durrant-Whyte and T. Bailey, "Simultaneous Localization and Mapping: Part II," *IEEE Robotics and Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[57] G. Klein and D. Murray, "Parallel Tracking and Mapping on a Camera Phone," in *Proceedings 8th IEEE and ACM International Symposium on Mixed and Augmented Reality*, (Orlando, FL), pp. 83–86, 2009.

[58] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Proceedings 4th Alvey Vision Conference*, (Manchaster, GB), pp. 147–151, 1988.

[59] J. Shi and C. Tomasi, "Good Features to Track," in *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (Seattle, WA), pp. 593–600, 1994.

[60] D. G. Lowe, "Object Recognition from Local Scale-Invariant Features," in *Proceedings 7th International Conference on Computer Vision*, (Kerkyra, GR), pp. 1150–1157, 1999.

[61] S. M. Smith and M. J. Brady, "SUSAN - A New Approach to Low Level Image Processing," *International Journal of Computer Vision*, vol. 23, no. 1, pp. 45–78, 1997.

[62] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A Comparison of Affine Region Detectors," *International Journal of Computer Vision*, vol. 65, no. 1, pp. 43–72, 2005.

[63] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust Wide Baseline Stereo from Maximally Stable Extremal Regions," in *Proceedings 15th British Machine Vision Conference*, (Kingston, GB), pp. 761–767, 2004.

[64] E. Rosten and T. Drummond, "Fusing Points and Lines for High Performance Tracking," in *Proceedings 10th International Conference on Computer Vision*, (Beijing, CN), pp. 1508–1515, 2005.

[65] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Proceedings 9th European Conference on Computer Vision*, (Graz, AT), pp. 430–443, 2006.

[66] V. S. Varadarajan, *Lie Groups, Lie Algebras and Their Representations*. Berlin / Heidelberg / NY: Springer-Verlag, second ed., 1984.

[67] F. Devernay and O. D. Faugeras, "Straight Lines Have To Be Straight," *Machine Vision and Applications*, vol. 13, no. 1, pp. 14–24, 2001.

[68] P. W. Holland and R. E. Welsch, "Robust Regression using Iteratively Reweighted Least-Squares," *Communications in Statistics-Theory and Methods*, vol. 6, no. 9, pp. 813–827, 1977.

[69] P. J. Huber, *Robust Statistics*. New York, Chichester, Weinheim, Brisbane, Singapore, Toronto: John Wiley & Sons Inc, 1981.

[70] R. Szeliski, *Computer Vision Algorithms and Applications.* Berlin / Heidelberg / NY: Springer-Verlag, 2010.

[71] Z. Zhang, "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[72] E. Catmull, *A Subdivision Algorithm for Computer Display of Curved Surfaces.* PhD thesis, University of Utah, 1974.

[73] M. Berger, T. Auer, G. Bachler, S. Scherer, and A. Pinz, "3D Model Based Pose Determination in Real-Time: Strategies, Convergence, Accuracy," in *Proceedings of 15th International Conference on Pattern Recognition*, (Los Alamitos, CA), pp. 567–570, 2000.

[74] M. D. Shuster, "A Survey of Attitude Representations," *Journal of Astronautical Sciences*, vol. 41, no. 4, pp. 439–517, 1993.