Graz University of Technology

Institut for Computer Graphics and Vision

Master's Thesis

LARGE-SCALE ROBOTIC SLAM THROUGH VISUAL MAPPING

Christof Hoppe

Graz, Austria, November 2010

Thesis supervisors

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

Dipl.-Ing. Dr.techn. Matthias Rüther

Abstract

Simultaneous Localization and Mapping (SLAM) in a three-dimensional environment is an essential requirement for autonomous mobile robots to accomplish high level tasks. An emerging sensor for SLAM is the digital camera, because it is cheap, small, has low weight and can be applied in many different application areas like marine, aerial or land robotics. Today's camera-based solutions, called *visual SLAM*, are limited to small environments like desktop or office scenes because of geometric error propagation and limited scalability.

In this master thesis, we developed a SLAM system that allows us to handle large-scale environments using a stereo-camera mounted on a wheeled robot. Our approach extends a keyframe-based method for augmented reality applications by adding appearance-based loop detection and correction. Furthermore, we propose a method for incorperating other sensor information like odometry into the visual SLAM framework. We are hereby able to preserve connectivity between camera poses even if visual features are absent. To maintain map accuracy without sacrificing excessive computation time, we combine feature descriptors of different strength for data association.

In the experiments, we show that our approach is able to handle trajectories of several hundred meters and containing several thousand visual features. The resulting three-dimensional maps have correct metric scale. The absolute trajectory error is below one percent. On a standardized benchmark dataset providing groundtruth trajectories, our system outperforms other visual SLAM algorithms by a factor of two.

Keywords. visual SLAM, bundle adjustment, loop detection, stereo camera, vocabulary tree, SURF features

Kurzfassung

Für viele Aufgabenbereiche autonomer, mobiler Roboter ist die gleichzeitige Erstellung einer dreidimensionalen Karte und die Lokalisierung in dieser (*Simultaneous Localization and Mapping (SLAM)*) eine unerlässliche Voraussetzung um die ihnen übertragenen Aufgaben zu lösen. Um eine dreidimensionale Rekonstruktion zu ermöglichen, werden als Sensoren häufig digitale Kameras verwendet, da sie billig, leicht und in vielen Einsatzbereichen wie der Unterwasser- oder Luftfahrtrobotik einsetzbar sind. Heutige kamerabasierte SLAM Verfahren sind, aufgrund von Fehlerfortpflanzung und geringer Skalierbarkeit, in der Lage Karten nur von räumlich sehr begrenzten Bereichen zu erstellen.

In dieser Masterarbeit wurde ein SLAM Verfahren für ausgedehnte Bereiche entwickelt, das die Bilder einer Stereokamera verwendet. Das in dieser Arbeit vorgestellte Verfahren basiert auf einem Ansatz, der für die Positionsbestimmung einer Kamera in Augmented Reality Programmen entwickelt wurde. Dieser wurde um eine Erkennung schon besuchter Orte erweitert und ein Verfahren zur Reduktion der Fehlerfortpflanzung implementiert. Ebenso wurde eine neuartige Methode zur Sensorfusion entwickelt, die Informationen anderer Sensoren in den visuellen Bereich transformiert. Somit kann eine Verbindung zwischen verschiedenen Kamerapositionen erstellt werden, selbst wenn in Bildern keine visuellen Features vorhanden sind. Um den Berechnungsaufwand trotz hoher Kartierungsgenauigkeit gering zu halten, wurden unterschiedliche starke visuelle Features miteinander kombiniert.

Die Experimente zeigen, dass der hier präsentierte Ansatz in der Lage ist, mehrere hundert Meter lange Trajektorien zu rekonstruieren. Die dabei erstellten maßstabsgerechten, dreidimensionalen Karte bestehen aus mehreren tausend visuellen Features. Der bei der Rekonstruktion der Robotertrajektorie gemachte absolute Fehler lag bei unter einem Prozent. Der hier entwickelte Ansatz erreichte, verglichen mit anderen visuellen SLAM Verfahren, auf einem standardisierten Datensatz eine doppelt so hohe Genauigkeit.

Acknowledgments

First and foremost I want to thank my supervisors Prof. Dr. Horst Bischof, Dr. Matthias Rüther and Katrin Pirker for continuously supporting me when facing problems, discussing problems, and carefully proofreading my master thesis.

I feel obliged to thank all the colleagues from the RoboCup team in Graz and Kassel, who awakened my interest in all kinds of robotics and on Computer Vision: Michael Reip, Stephan Gspandl, Christof Rath, Christoph Zehentner, Roland Reichle, Theresa Rienmüller, and Philipp A. Baer, who proofread my thesis and gave me a lot of hints to this work.

Last but not least, I would like to thank my parents, my brothers and sister, and my girlfriend Johanna. They gave me the possibility to study and showed understanding and patience in times of intense work. Especially Johanna accompanied me through my ups and downs during the master thesis and encouraged me to complete this work successfully.

This work was supported by the Austrian Research Promotion Agency (FFG) and Federal Ministry of Economics, Family Affair and Youth (BMWFJ) within the Austrian research Studio Machine Vision Meets Mobility.

Contents

List of Figures

Chapter 1

Introduction

Today, the application areas of robots can be grouped into three classes: service robotics, like lawn-mower or pool cleaner, autonomous moving vehicles, and special robots, like unmanned planes or underwater robots. In the future, robots will be used in many more areas to help the user in his every day life. In most applications, the robot will move autonomously and interact with the environment. Three examples for autonomous mobile robots are shown in Figure 1.1. Their application vary from personal assistant robots that supports the user in his everyday live, autonomous moving cars up to unmanned airplanes that are used for surveillance tasks.

A key problem for autonomously moving vehicles is the estimation of their position in

Figure 1.1: Three different application areas of autonomous robots. (a) A personal assistant robot developed by Fujitsu. It navigates autonomously and communicates with the user taken from [12]. (b) A vehicle has to find a way from the desert to the city. (c) Autonomous planes are used by the military for surveillance.

the environment within a given map. In many real applications maps are not available at the outset or the existing maps are not compatible with the robot's sensor. Also, the environment may change over time so that static maps become invalid and should be updated autonomously. To use a robot in such environments it has to incrementally learn the map while it is moving. In literature this problem is called *Simultaneous Localization And Mapping* (SLAM). SLAM is more complex than localization, because map building and localization has to be solved simultaneously. More than often this is a chicken-and-egg problem: The robot requires a map to determine its pose in the world, whereas the pose is also required for constructing the map.

SLAM has become an important research field in robotics in the last years, but it has not been solved completely. In the past, research concentrated on SLAM in the two-dimensional case as SLAM was mostly used by wheeled robots. A lot of methods have been developed for different kinds of sensors and they achieved reasonable results. In three dimensions, SLAM is more complex and current methods are not able to build maps larger than several hundred square meters. The accuracy of today‘s SLAM methods is limited and decreases while the map gets larger, because the computational effort grows at least linearly in map size. Thus, the goal of this thesis is to develop a SLAM algorithm for mapping large environments in three dimensions with high accuracy.

The underlying problem to be solved by SLAM algorithms is to handle noise that is introduced by the sensors and actors of a robot. The problem is illustrated in Figure 1.2. The robot is given a signal to move straight forward along a corridor. Since the motion is perturbed by noise, the exact new position cannot be calculated by odometry measurements alone. The noise allows to determine the new position only in a probabilistic sense. In the same way, the sensor data for environment reconstruction is also perturbed by noise and therefore each measurement has also an uncertainty. However, if it is possible to align the measurements and to find correspondences between different measurements, the pose of the robot can be corrected.

To create a three-dimensional map, the robot has to be equipped with a sensor system that is able to perceive all three dimensions. Potential sensors are pan-tilt laser scanners or time-of-flight cameras. They directly provide range measurements of their surroundings, however, they are often expensive, large and heavy. Another sensor that is able to reconstruct its environment are cameras. A camera has several advantages:

- widely-used and cheap
- low weight and small

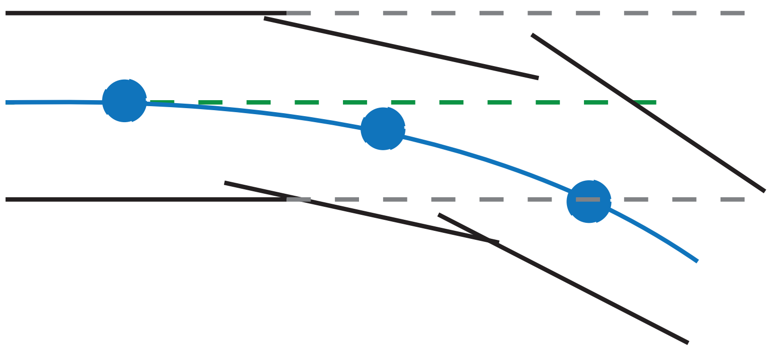


Figure 1.2: Basic SLAM illustration. The robot (blue dot) is given the command to follow the green, dashed path that is bordered by straight walls (gray dashed). The blue line illustrates the path reconstructed by an internal movement sensor (e.g. odometry) and the reconstructed environment with another sensor (e.g. laser scanner).

- insensitive to most external effects like vibrancy
- low power consumption

These characteristics allow to use cameras in a wide range of applications and products, e.g. in today's mobile phones. Furthermore, camera systems are well-studied in computer vision and used in many applications beside SLAM, like 3D reconstruction. Unfortunately, image processing is a relatively complex task. Despite that, cameras are the state-of-the-art sensors for three-dimensional SLAM applications. Algorithms based on camera information are called *visual SLAM*.

The use of a camera as the sensor for SLAM involves a large number of challenges. In contrast to other sensors, like a time-of flight camera that directly produce three-dimensional environment information, this information has to be calculated from at least two images of the same object taken from different points of view. In the reconstruction step, these projections have to be localized in the images for calculating three-dimensional information. This is known as the *corresponding point problem* and one of the basic problems in computer vision. Because today's corresponding point detectors generate outliers, the visual SLAM algorithm has to cope with. Furthermore, reconstruction requires textured images. For example, if the camera faces a white wall, the three-dimensional pose of the wall cannot be determined. In the opposite case of well-textured images, there can be several hundred corresponding points. A reliable visual SLAM algorithm has to handle both extreme cases.

Two classes of visual SLAM algorithms can be distinguished: Probabilistic and optimization based methods. Probabilistic algorithms, like Extended Kalmann Filters or Particle Filters, achieve reasonable results in small environments. Their main disadvantage is their high computational complexity, which grows at least linearly with the number of measurements. This limits the application area to small environments with less measurements. The second class formulates the SLAM problem as a geometric optimization problem. Given a number of measurements of the same object and the positions the observer locations, the algorithm optimizes measurements and locations simultaneously. In computer vision, this method is known as *bundle adjustment* and formerly used in photogrammetry. This method promises to work in large environments with several thousand measurements and it is also able to handle outliers. Recent research showed that bundle adjustment outperforms probabilistic approaches with respect to scalability, computational complexity, and accuracy [36].

Our approach is based on the *Parallel Tracking and Mapping* (PTAM) software developed by Klein et al. [19] in 2007, which is used to estimate the pose of a single camera in an unknown, arbitrary environment in real-time. In PTAM, bundle adjustment is applied to solve the SLAM problem. Experiments showed highly accurate pose estimation while the computational complexity is constant with respect to map size. Figure 2.5 shows an example of PTAM for tracking a single camera in an office environment. Since PTAM is used for augmented reality, it is designed for a limited area and cannot be applied for large scale SLAM directly. Because PTAM uses a monocular camera, the reconstructed map does not have true scale, which is unacceptable for metric SLAM. Furthermore, the strategy for map building is inadequate for large scale mapping and PTAM has no support for loop closing.

In this thesis, we develop a true scale metric SLAM system based on PTAM. We changed the monocular camera setup to a stereo camera rig that allows accurate three-dimensional reconstruction. We replaced the identification of corresponding image points with patch-based correlation by the stronger SURF descriptor. We added an appearance-based loop detection mechanism and perform loop closing using bundle adjustment. Furthermore, we propose a method for preserving connectivity between frames, even if they do not share visual image features.

We evaluate the accuracy and scalability of our system on two different datasets. The first dataset is a self-recorded trajectory and comprises four cycles of an indoor environment. The second image sequence is provided by the Rawseeds project. In three experiments, we show our improvements with respect to PTAM. In the first experiment, we changed PTAM from monocular camera into the stereo camera approach. Second, we demonstrate the improvements in accuracy and scalability by using a more robust feature descriptor. Last, we add loop closing to our system, which decreases the number of map features while increasing accuracy. We evidence on the Rawseeds dataset that our approach outperforms other stereo visual SLAM algorithms concerning accuracy. Furthermore, we give a runtime analysis, that shows the ability to run our system in real-time.

This thesis is structured as follows. In Chapter 2, we give an overview of the state-of-the-art in visual SLAM and explain the principles of PTAM in more detail. Chapter 3 deals with underlying principles of our approach: the mathematical background of multiview geometry, feature detection and multiview reconstruction. This chapter contains also the theory of bundle adjustment and methods to speed up optimization. In Chapter 4, we discuss the localization within a given map. In the following chapter, we present the process of iterative map building, loop detection and correction, and a fall-back method that is used, if visual SLAM is not possible due to less textured images. In Chapter 6, we evaluate our approach and last, we outline the proposed method, discuss the results of Chapter 6 and give an outlook on future work.

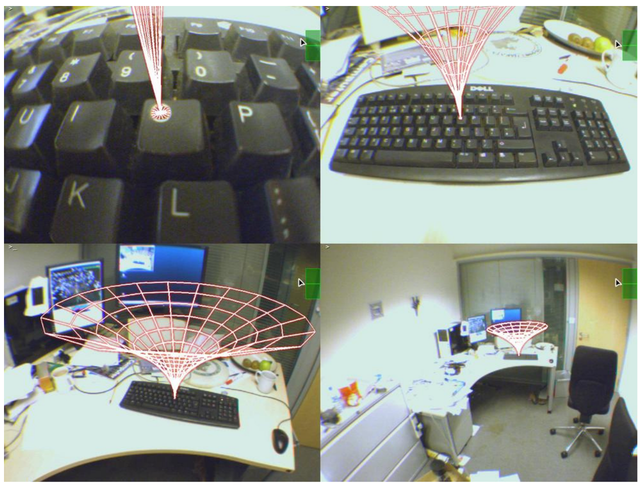


Figure 1.3: Example application of PTAM. PTAM is initialized on the right-top image. Although the camera moves closer, the overlaid object is registered correctly.

Chapter 2

Related Work

Historically, the research of SLAM topic is studied in two fields of computer vision, which led to two different notations. In photogrammetry, the problem is called *structure from motion* (SfM) and focusses on building a map from the measurements of a moving sensor. The trajectory of the sensor and the processing time is secondary. Since most SfM algorithms are batch processes, the main focus does not lie on incremental map building. For Simultaneous Localization and Mapping (SLAM), the incremental approach is clearly required. This requires real-time algorithms and incremental approaches for map building. With the help of increasing computational power, classical SfM approaches are adopted to operate in an incremental manner.

In contrast to SfM methods, which rely on global optimization methods like bundle adjustment, today's visual SLAM methods mostly describe the problem as a question of probability theory.

2.1 Probabilistic Approaches

In this section we present approaches that formulate the SLAM problem in a probabilistic manner. Given the measurements z and the robot control input u, we calculate

$$p(s_t, m_t | z^t, u^t),$$

where p denotes the probability of a robot's pose s and a map m at time t. The online SLAM problem is to compute the maximum probability of all possible maps and poses, which can be formulated as a Bayesian network. Figure 2.1 illustrates the network structure.

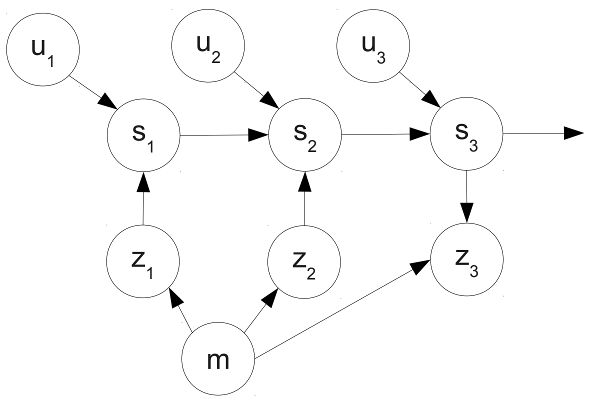
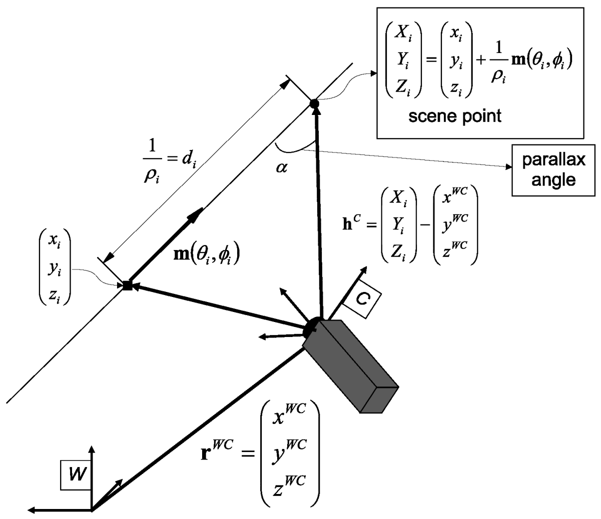


Figure 2.1: Bayesian network problem statement. In online SLAM, we estimate the probability of a robot's pose s along with the map m.

Extended Kalmann Filter SLAM

Davison and Kita [8] proposed a SLAM algorithm that makes use of the *Extended Kalmann Filter (EKF)* [37]. The map consists of a number of local image features, that are encoded as 3D positions in a global Euclidean coordinate system. The 6 *Degrees of Freedom* (DOF) robot pose, as well as all map points are combined to a state vector x, whereas their uncertainties are expressed as a covariance matrix. In each time step new features may be



$\begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \frac{1}{\rho_i}\mathbf{m}(\theta_i, \phi_i)$
scene point
parallax angle
$\frac{1}{\rho_i} = d_i$
α
$\mathbf{h}^C = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} - \begin{pmatrix} x^{WC} \\ y^{WC} \\ z^{WC} \end{pmatrix}$
$\begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$
$\mathbf{m}(\theta_i, \phi_i)$
C
$\mathbf{r}^{WC} = \begin{pmatrix} x^{WC} \\ y^{WC} \\ z^{WC} \end{pmatrix}$
W

Figure 2.2: Inverse Depth Parameterization. A point is expressed by a 6 tuple. This representation leads to a more linear problem and can also handle points with large uncertainties on the z-axis.

added, which causes a growth of the state vector and the covariance matrix.

This approach has some limitations. Although SLAM in this formulation is highly non-linear, the EKF linearizes the problem at the current state, which leads to a non optimal solution. The error produced by the linearization also depends on the uncertainties contained in the underlying data and grows with higher uncertainty. Furthermore, the EKF requires estimates of process and measurement noise, which are sometimes difficult to appraise. Because the state vector as well as the covariance matrix may grow in each time step, the computational complexity also increases and, hence, the algorithm is only able to handle a limited amount of features. Furthermore, the algorithm is not robust against outliers. A false measurement influences the whole filter and it is impossible to correct it afterwards. So, an efficient outlier detection mechanism like a RANSAC [11] algorithm has to be implemented to prevent defective measurement updates. An advantage of this approach is that it can be easily implemented. This approach serves as basis for a large number of enhancements.

Montiel et al. [25] tackled the problem of non-linearity by changing the representation of map points. Instead of describing map points in a common Euclidean coordinate system, they parameterize them by six parameters. Their representation is called *Unified Inverse Depth Parameterization* and promises to lead to a more linear problem, which can be handled more adequately by the EKF. Furthermore, the inverse depth parametrization is able to handle map points with large depth uncertainties, which is useful in the initialization phase of the map when using only a single camera or if map points are generated by two views with a small baseline. Figure 2.2 illustrates the parametrization of a map point. The disadvantage of this representation is that each map point is represented by six parameters and therefore state vector and covariance matrix are squared by size compared to the Euclidean representation. The same authors showed in [5] that in many cases the Euclidean representation is sufficient and they developed a classification in which cases a map point should be represented by the inverse depth representation instead of the Euclidean one.

A classical approach to deal with large maps is to split the global map into several local ones. Each local map has a limited number of features, which reduces the computational complexity of the EKF. Paz et al. [22] proposed a Divide and Conquer algorithm that reduces the EKF SLAM computational complexity from $O(n^3)$ to $O(n^2)$. Their map representation is based on the inverse depth representation [25].

Particle Filter SLAM

Another group of probabilistic SLAM algorithms is based on Rao-Blackwallized particle filters [9].

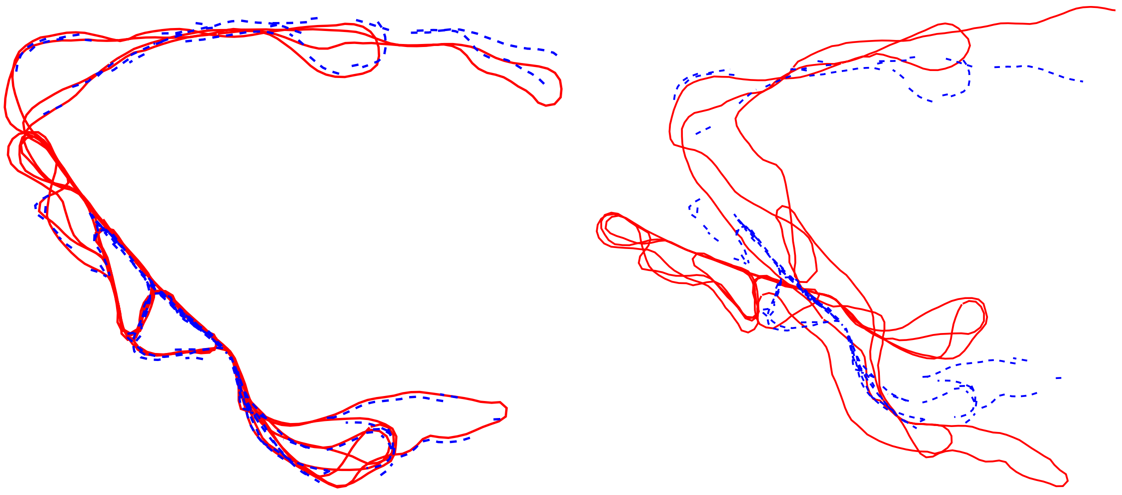
Montemerlo et al. [24] divided the SLAM problem in a robot localization part and a part of feature estimation depending on the robot's pose. The robot pose is estimated by a particle filter, whereas each feature is tracked by a Kalmann filter conditioned by the robot pose. The combination of particle and Kalmann filters is called Rao-Blackwallized particle filter. They demonstrated the performance of their system on a SLAM problem using a laser scanner. Assuming a particle filter with N particles and M landmarks, the system consists of $N \times M$ Kalman filters which has complexity of $O(M\ N)$ for an update using a naive implementation. Montemerlo et al.[24] reduced the computational time to $O(M\ log\ N)$ by using a tree structure and they show that their approach is able to handle up to 50,000 features using up to 100 particles. Figure 2.3 illustrates the accuracy of Montemerlos method (called FastSLAM) compared to the standard EKF approach.

Sim et al. [33] adopted the idea of FastSLAM for the visual domain. They equipped a robot with a stereo camera, extracted SIFT features and triangulated their 3D position. SIFT descriptors are also used for data association. Their approach is not constrained to a motion model based on odometry information, but the motion between two consecutive frames is estimated by landmark-based visual odometry. Using this approach, they created a map from two connected laboratory rooms with a total trajectory length of 67.5 m. The main disadvantage is the high computational cost. The mean processing time for each frame was 11.9 s for a 4000 frame trajectory, which is far away from real-time. The same authors extended their approach by changing the proposal distribution of the particle filter, which enables their SLAM algorithm to close large loops. They also reduced the mean processing time per frame to 1.5 s.

In [40] Zhou et al. combined a time of flight camera (TOF) and a conventional camera in order to build a three-dimensional dense map. A Rao-blackwellized particle filter was used to estimate the robot's trajectory with utilizing SIFT based three-dimensonal image points.

Non-metric SLAM

Another class of SLAM algorithms are appearance based methods that have a quite different map representation. All algorithms discussed before, present distinct image features with their corresponding 3D coordinates, whereas appearance based methods store only

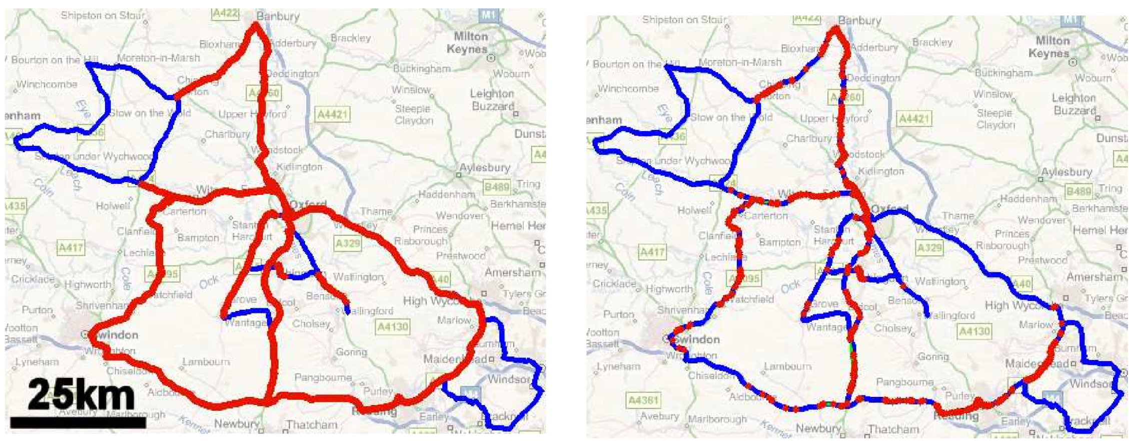


(a) (b)

Figure 2.3: Comparison FastSLAM vs. EKF. Montemerlo [24] compared the FastSLAM to the standard EKF algorithm on a simulated dataset. (a) shows the result of the FastSLAM, whereas (b) is the outcome of the EKF using a laser scanner as input sensor. The dashed line illustrates groundtruth, whereas the solid line shows the reconstructed trajectory.

plain images. Hence, the SLAM algorithm does not result in a full 6D pose estimate, but in a probability, wether an observed place has been visited anytime before.

Cummins et al. [7] focus on the bag of words approach. They assume that each picture is represented by visual words like SIFT descriptors. Given a set of features, the



(a) (b)

Figure 2.4: FAB-MAP Experiment. (a) The groundtruth of a large trajectory. Blue parts of the trajectory are visited only once, whereas the red parts are visited at least two times. (b) The detected loop closures are shown in blue taken from [7].

algorithm, called FAB-MAP, returns the probability that this set comes from a certain location. They do not only learn the similarity between pictures, but they also take into account that there is limited evidence if only non-distinctive features are in the sample set. This is important for maps containing repetitive structures like corridors to reduce false-positive matches. Their algorithm is linear in time and is therefore applicable for real-time loop detection. In order to show the performance of their algorithm, they acquired 103,000 omni-directional images on a 1000 km trajectory. The location of the image was estimated by a GPS sensor. Images that were acquired from the same trajectory on a second run, were compared to the previously stored images. On the second run, they detected 2,819 loop closures, where six of them were false-positive matches. Figure 2.4 shows the result of the 1000 km trajectory. The authors mention that large parts of the trajectory were taken from highways that contain only few distinctive visual features.

2.2 Geometric Approachs

The second large class of SLAM algorithms is inspired by optimization methods formerly used by *structure from motion* (SfM) systems. Here, the main idea is to reformulate the SLAM problem as an optimization problem that can be solved by a least-square solver (also known as bundle adjustment). Bundle adjustment optimizes the constructed 3D structure and camera poses by minimizing the reprojection error. In the past years, bundle adjustment became more and more important for SLAM due to the increased processing power.

A very popular SLAM approach based on bundle adjustment was developed by Klein et al. [19]. Their algorithm, called Parallel Tracking and Mapping (PTAM), was designed for tracking a monocular camera in a small environment. They divided the SLAM problem into two tasks: Tracking the cameras pose within the map and extending the map whenever the camera explores new regions. To reduce computational complexity, the map is extended only at certain points in time by so-called *keyframes*. A keyframe is an image linked to a 3D camera pose and is used to reconstruct distinctive image points by triangulation. As triangulation requires at least two projections of the same object, corresponding points have to be identified in other keyframes by using patch-based correlation. In order to enhance the accuracy of the map, a fixed number of keyframes is optimized by bundle adjustment at each map extension.

The pose of the camera within the map is tracked by a visual odometry approach. Map points that are already stored in the map are identified in the current image using patch-based correlation. The pose is optimized by minimizing the reprojection error of the recognized map points. Robust optimization methods based on M-Estimators are used to reduce the influence of false matches.

Since map extension and camera tracking have different computational requirements, these tasks can be executed in two seperate threads. Tracking has to be performed in real-time, whereas a map update may be much slower. So, the computationally expensive bundle adjustment can be used to optimize both structure and motion, when a new keyframe is added to the map. The bundle adjustment used in PTAM has a complexity of $O(N^2M)$, where N is the number of keyframes and M is the number 3D points. To keep real-time properties, only the n previously added keyframes are optimized. The authors showed that their system is able to cope with maps that consist of several thousand map points in real-time at a highly accurate level. Even large scale differences as shown in Figure 2.5 can be handled.

Since feature extraction is based on corners, PTAM is prone to motion blur. The authors extended their method by using lines as a second image feature [20] to be more robust against this kind of distortions. Furthermore, they implemented a simple recovery strategy that allows global localization if the tracking gets lost.

PTAM has shown that bundle adjustment can be used even in real-time systems to improve the quality of a map. One of the major disadvantage is the data association by using a patch-based approach, which produces large numbers of false associations. The implementation of PTAM is highly optimized for small workspaces, where the camera moves

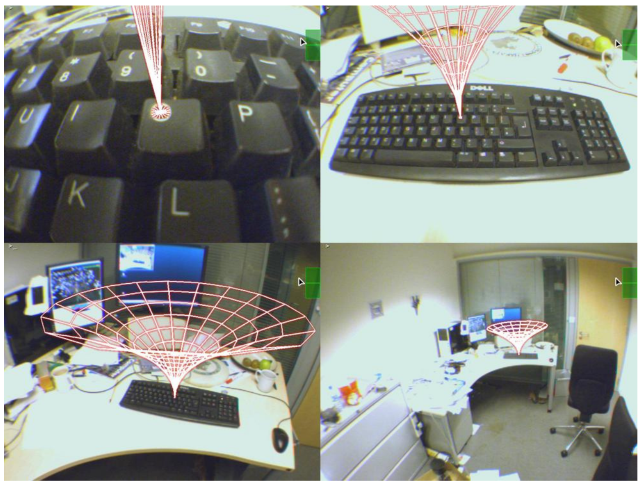


Figure 2.5: Example of a scene tracked by PTAM. PTAM is initialized on the upper right image and is then moved closer and further from the initialized object (taken from [19]).

in an already explored region most of the time. In case of exploring new regions quickly PTAM fails, because even local bundle adjustment is too time consuming. Furthermore, PTAM has not implemented loop detection and loop closing, which is an important feature for large scale SLAM applications.

Castle et al. [3] modified PTAM for generating maps of large environments by adding multimap support. The tracking and mapping is unchanged but the algorithm is extended to switch between several maps.

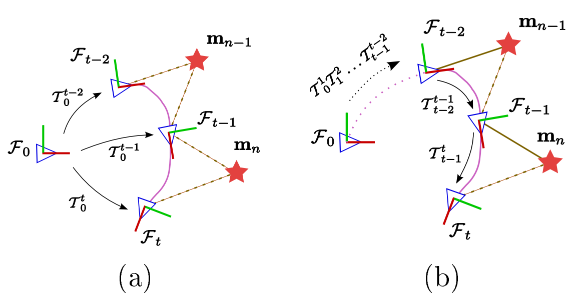


Figure 2.6: Relative representation of positions and map points as proposed by [32]. Figure (a) illustrates the orientation of cameras and features within a fixed coordinate system. In Figure (b) a camera is represented relative to its previous pose. Map points are stored relative to the camera pose of their first appearance.

Because of bundle adjustment's complexity, loop closures can not be calculated in real-time. Sibley et al. [32] tackle this problem by changing the representation of keyframes and map points. Instead of using a global world coordinate frame, all points and keyframes are represented in a relative manner (see Figure 2.2). This description of poses and map points prevents a propagation of large reprojection errors through the whole loop. Even loops with a large displacement require only a few iterations to converge. One disadvantage of the solution is that it cannot be transformed easily to an Euclidean representation. Hence, calculations based on absolute positions like path planning are difficult to perform in this framework. Based on Sibley's relative bundle adjustment (RBA), Mei et al. [23] used RBA in a high accurate stereo SLAM framework. They integrated relative bundle adjustment together with the FAB-MAP [7] loop detection. Furthermore, they attached great importance to corresponding point detection between the stereo images. They evaluated their framework on a 2.26 km outdoor dataset containing 51,000 images. As shown in Figure 2.2, relative bundle adjustment reduced the absolute position error of former 25 m to approx. 10 cm.

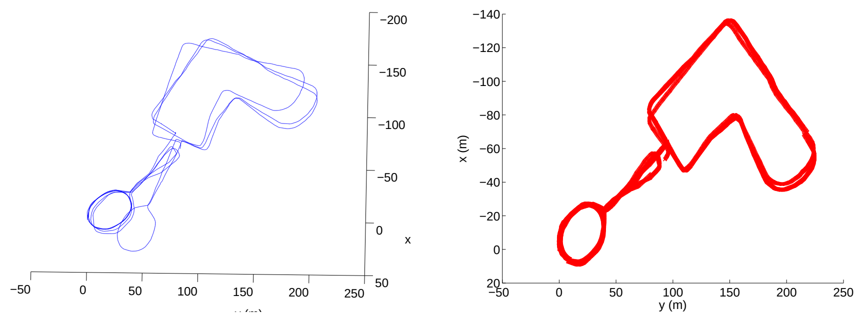
In 2009, Holmes et al. [17] replaced the representation of map points and cameras in

PTAM from fixed Euclidean coordinates to a relative representation as proposed by Sibley et al. [32]. Furthermore, they partitioned the RBA into a local and a global version. The local version allows a fast optimization ($O(n)$) around the current camera pose in real-time, whereas a global RBA is performed as a background task. This allows to add keyframes in real-time and overcomes the problem of PTAM when exploring unknown environments quickly. Unfortunately, extensive experiments have not been performed yet.

Konolige et al. [21] reduced the computational cost of large bundle adjustment by reformulating the constraints of the optimization problem. In standard bundle adjustment a camera pose and the reconstructed structure is optimized by minimizing the reprojection error. In general, this involves several hundreds parameters and equations. They replaced this large number of constraints by only a few constraints that nearly have the same conditions like all constraints together. This led to an enormous reduction of parameters and equations. They showed that the time for optimizing a loop of 370 meters took 35 ms compared to several seconds or minutes.

There exists also approaches that combine EKF and bundle adjustment. Eade and Drummond [10] build a map of local submaps, in which multiple images share a nearly linear observation model and combine them by a graph structure. On a global level, the graph structure is then optimized by bundle adjustment, whereas the local submaps are tracked by an EKF filter. They showed that their map representation avoids inconsistency in the EKF filter and performs in real-time with several hundred landmarks. The accuracy of their approach is near to offline bundle adjustment.

Strasdat et al. [36] compared the performance of EKF-based SLAM algorithms and



(a) (b)

Figure 2.7: Results of SLAM system proposed by [23] using relative bundle adjustment. (a) The trajectory produced by the stereo SLAM approach without loop detection and correction. (b) The trajectory corrected with relative bundle adjustment.

methods that make use of bundle adjustment. They analyzed both approaches in simulations, as well as on real image sequences, in terms of accuracy and computational costs. They concluded that bundle adjustment approaches are predominant with regard to accuracy and therefore EKF-based methods have a niche field in systems with low processing power.

2.3 Summary

In this chapter, we discussed different approaches for solving the SLAM problem developed in the past. We focused on approaches based on EKF filters and methods based on geometric optimization, which became popular in the last years.

The SLAM methods implemented with EKF filters are basically able to handle only a limited area because the complexity increases drastically with the number of features. To overcome this limitation, the idea of submaps has been developed [10] that are optimized on a global level. Therfore, EKF-based algorithms for large scale SLAM problems consists of at least two different mechanism for map building.

In the recent years, methods based on geometric optimization became more and more popular, because it turned out that they are able to handle large numbers of features efficiently. PTAM was one of the first systems that implemented bundle adjustment as the underlying optimization method for map and pose estimation. Even though their implementation is optimized for a limited area, the authors showed, PTAM is able to cope with large numbers of visual features in realtime with very high accuracy. Confirmed by the results of Strasdat et al. [36], our approach presented in this master thesis therefore is based on bundle adjustment.

To create a metric and accurate SLAM system, we extended PTAM in several ways. Our systems uses a calibrated stereo camera pair which guarantees an accurate metric reconstruction of the environment. Furthermore, it turned out that a more complex feature descriptor enhances the accuracy of map building. Since PTAM is developed for tracking a camera in unknown environments, it has no support for global localization. We solve this problem by an appereance based method which is also applied for loop detection. Loop detection and correction is essential for accurate mapping particular for large scale SLAM and is carried out by bundle adjustment in our approach.

Chapter 3

Theory and Background

Contents

The goal of a visual SLAM algorithm is to keep track of a robot's trajectory while exploring an unknown environment and creating a visual map. A robot equipped with one or more cameras observes its three-dimensional surrounding and maintains a three-dimensional map. In this chapter, we outline the techniques and algorithms used in the SLAM context. Section 3.1 introduces basic aspects of multiview geometry like frame transformations, the perspective camera model, and camera calibration. In the following deals with feature extraction and feature matching in different images. Three-dimensional reconstruction using two or more images is discussed in Chapter 3.3. We also introduce the concept of *Bundle Adjustment* and explain the relevant mathematical background.

3.1 Multiview Geometry

This section elaborates on the mathematical background of frame transformations that is important for tracking objects in three dimensions. Furthermore, we introduce the pinhole camera model and show a basic method for estimating the relative orientation between two cameras.

3.1.1 Rigid Body Motion

The tracking of an object in an Euclidean space can be seen as the estimation of the objects coordinate system $\mathcal{F}$ with respect to a world coordinate frame $\mathcal{W}$. In the following, we elaborate on two mathematical concepts that describe the transformation from $\mathcal{F}$ to $\mathcal{W}$.

Frame Transformation

Any orientation in 3D can be described by three consecutive rotations around three different axes. The three rotated angles refer to as the *Euler angles.* The resulting orientation depends on the rotation axes order. A common ordering used in aviation is the ZYX convention, i.e. the first rotation is performed around the Z-axis (yaw angle), the second around the Y-axis (pitch angle), and the last around the X-axis (roll angle).

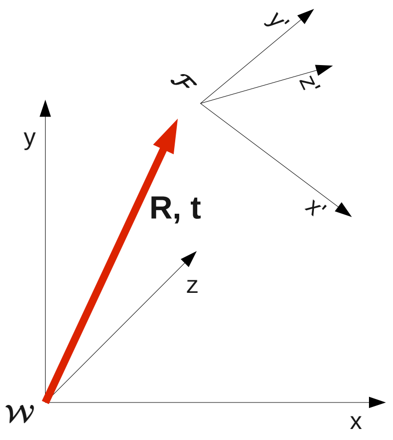


Figure 3.1: Transformation between a coordinate frame $\mathcal{W}$ and a coordinate frame $\mathcal{F}$. The position of $\mathcal{F}$ can be expressed by a rotation matrix R and a translation vector t with respect to the origin of $\mathcal{W}$

Euler angles are often used to visualize orientations because they are easily understood by humans. In order to transform the representation of a point $P^{\mathcal{W}}$ to a point $P^{\mathcal{F}}$, the Euler angles are transformed to a matrix representation. Any two dimensional rotation around one of the three axes can be expressed by a 3×3 matrix, e.g. a rotation around the z-axis by angle ϕ can be written as

$$R_z(\alpha) = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since each orientation can be expressed by three consecutive two dimensional rotations, an arbitrary orientation can be expressed by

$$R_x(\gamma)R_x(\beta)R_z(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma) & \cos(\gamma) \end{bmatrix} \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin\beta) & 0 & \cos(\beta) \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The rotation matrix has several useful properties:

$$R^{-1} = R^T$$
$$det(R) = 1$$
$$\|Rv\| = \|v\|$$

As we will see later, the representation of the rotation as a matrix is convenient for coordinate frame conversion. However, it is not the most memory efficient method for encoding a rotation due to its enormous redundancy: A rotation matrix has nine entries whereas a rotation can be encoded by three parameters.

In the field of computer vision, another orientation description is used: Every rotation can also be expressed by the rotation of angle Θ around an arbitrary rotation vector v with $\|v\| = 1$, which is called *axis angle* representation. Because $\|v\| = 1$, this representation has only three parameters and is therefore a minimal representation of a rotation. The conversion between the axis angle and the matrix representation is given by the Rodrigues rotation formula [26].

As shown in Figure 3.1, coordinate frame $\mathcal{F}$ has a translation displacement that is typically expressed by a 3×1 translation vector t. t describes the origin of $\mathcal{F}$ with respect to $\mathcal{W}$:

$$t = \begin{bmatrix} x \\ y \\ z \end{bmatrix}^{\mathcal{W}}$$

The whole coordinate transformation comprises the translation by vector t and the rotation R. Denoting $p^{\mathcal{W}}$ a point in the coordinate frame $\mathcal{W}$ and $p^{\mathcal{F}}$ the same point expressed in $\mathcal{F}$, the transformation from frame $\mathcal{F}$ to frame $\mathcal{W}$ is calculated by

$$p^{\mathcal{W}} = Rp^{\mathcal{F}} + t. \tag{3.1}$$

The inverse operation from frame $\mathcal{W}$ to frame $\mathcal{F}$ is defined by

$$p^{\mathcal{F}} = R^T p^{\mathcal{W}} - R^T t. \tag{3.2}$$

Equations 3.1 and 3.2 can be simplified by the use of homogenous coordinates. In this representation, a three-dimensional point p_h is described by an arbitrary multiple of the vector $(X, Y, Z, 1)$, where X, Y, and Z are the Cartesian coordinates of p_h. According to Equation 3.1, the transformation of point $p_h^{\mathcal{F}}$ to the point $p_h^{\mathcal{W}}$ can by written as

$$p_h^{\mathcal{W}} = M p_h^{\mathcal{F}},$$

where M is a 4×4 matrix constructed of R and t:

$$M = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

3.1.2 Perspective Camera Model

A camera describes the mapping between a 3D point $\mathbf{X}$ in a world coordinate frame $\mathcal{C}$ and a 2D point $\mathbf{x}$ on an image plane. Several camera models exists, but in this thesis we

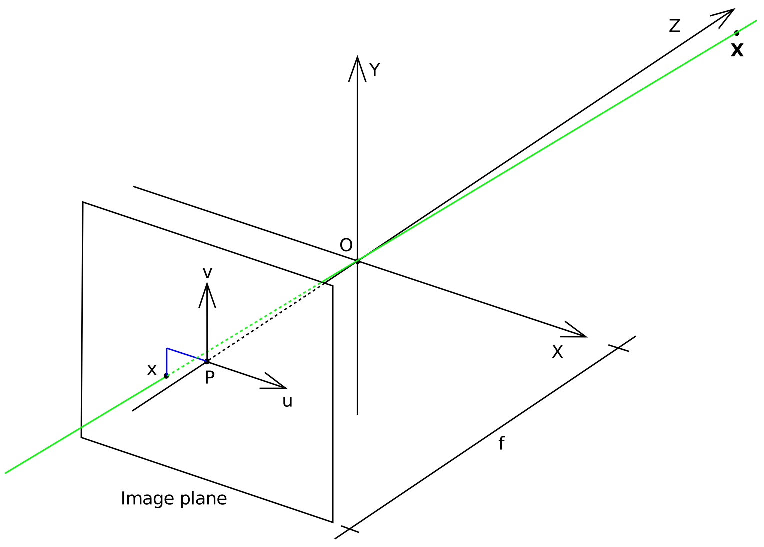


Figure 3.2: Pinhole camera model. A 3D point $\mathbf{X}$ is projected to the 2D point $\mathbf{x}$ in image plane by ray that is going through the pinhole O. The focal length f is the distance between the pinhole and the principal point P.

concentrate on a standard pinhole camera that can be used to model a standard camera. A pinhole camera defines the central projection of a 3D point X onto a 2D image plane, resulting in point x. A central projection in general can be described as

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

where $\mathbf{x}$ is a homogenous 3×1 vector $(u, v, w)^T$ representing the image point and $\mathbf{X}$ is a 4×1 vector describing the world point in homogenous coordinates. P denotes the 3×4 projection matrix, which can be decomposed into two parts: extrinsic parameters (C) and intrinsic camera parameters (K). The intrinsic parameters describe the camera's internal configuration and is represented by a 3×3 matrix

$$K = \begin{pmatrix} f & s & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{pmatrix},$$

where f is the focal length, s the skew factor, and p_x and p_y are the principal point coordinates of the camera.

The extrinsic parameters C represent the external orientation and position of the camera with respect to $\mathcal{C}$. The orientation R is a 3×3 rotation matrix and the translation T is a 3×1 vector. The matrix C is composed of

$$C = (R|t)$$

Multiplying K and C results in the projection matrix P

$$P = K[R|t]$$

This model is valid for an optimal pinhole camera. Practically all image devices are equipped with a lens that produces so-called lens distortion, i.e. straight lines are not depicted as straight lines in the image. The distorted pixel $p_d = (x_d, y_d)$ can be corrected to the undistorted pixel $p_u = (x_u, y_u)$ by

$$p_u = \begin{pmatrix} x_u \\ y_u \end{pmatrix} = \begin{pmatrix} p_x + L(r)(x_d - p_x) \\ p_y + L(r)(y_d - p_y) \end{pmatrix},$$

where p_x und p_y denote the center of the radial distortion; typically the principal point of the camera. The function $L(r)$ is the radial distortion function that can be approximated by a Taylor series:

$$L(r) = 1 + \kappa_1 r^2 + \kappa_2 r^3 + \kappa_3 r^4 + \kappa_4 r^5, \tag{3.3}$$

where r is the Euclidean distance between the radial distortion center and the distorted pixel p_d.

Both, the intrinsic parameters K and the radial distortion parameters κ_i are determined in a calibration process. A more detailed description of camera calibration techniques can be found in [39].

3.1.3 Relative Camera Calibration

Assuming a robot equipped with more than a single camera, it is useful to estimate the relative orientation between the cameras. If all cameras are mounted on a rig, the calibration process can be performed only once. Here, we present the calibration process between two cameras P_l and P_r. The algorithm can be extended for an arbitrary number of cameras.

Defining P_l as the reference camera, the pose of P_r is determined with respect to P_l as follows: Given a planar calibration target that is at least partially visible in both cameras, we extract the corresponding points X_l and X_r for a target point X. The transformation

$$X_r = R \cdot X_l + T \tag{3.4}$$

describes the the pose of camera P_r relative to the reference camera P_l. R is 3×3 rotation matrix and T a 3×1 translation vector. This equation can be solved by 4 corresponding non-collinear image points in P_l and P_r. The rotation and translation is stored in a homography matrix

$$H = (R|T)\,. \tag{3.5}$$

The projection matrix of the right camera is then altered by the homography:

$$P_r = K_r \cdot H, \tag{3.6}$$

where K_r is the intrinsic camera parameter camera P_r. Since P_l is the reference camera it defines the origin of the camera coordinate system and is set to the canonical pose.

3.2 Salient Image Points

The detection of corresponding points between two or more images is one of the key problems in computer vision and is not solved yet. Due to the importance and the complexity of this problem, myriads of different approaches exist. Most approaches are based on so called keypoints: A keypoint is a image area that is stable under global and local perturbations like noise, different points of view, lighting conditions and so on. This should guarantee that the same area is recognized in another image that is taken under different conditions. Furthermore, each salient point is connected to a feature descriptor that represents the image patch around the salient point. In order to find corresponding points in different images, these feature descriptors are compared to each other.

3.2.1 Feature extraction

Two types of descriptors can be distinguished: patch-based and feature-based descriptors. A patch-based descriptor uses the gray values of the keypoints neighborhood, whereas a feature-based descriptor extracts advanced information like gradients around the keypoint. In this section we present two feature-based and one patch-based descriptor.

Patch-based Descriptors

Initially, corner points are extracted from a gray-scale image. A common corner detector is the Harris corner detector [15]. An improved approach is the FAST corner detector [30] that significantly reduces the computational costs. The descriptor of the keypoint is the $n \times m$ image patch neighborhood.

This descriptor can be used in limited applications, because it is not invariant to viewpoint and lightning changes. Furthermore, it is not robust against high-frequency image noise and it has a weak discriminative property. Hence, the descriptor can be used in applications where prior knowledge about the corresponding image position is available and the viewpoint between both images varies only a little.

There are several extensions to this basic approach in order to be more robust against viewpoint changes. These techniques fit a warped version of the image patch to a corresponding keypoint [1].

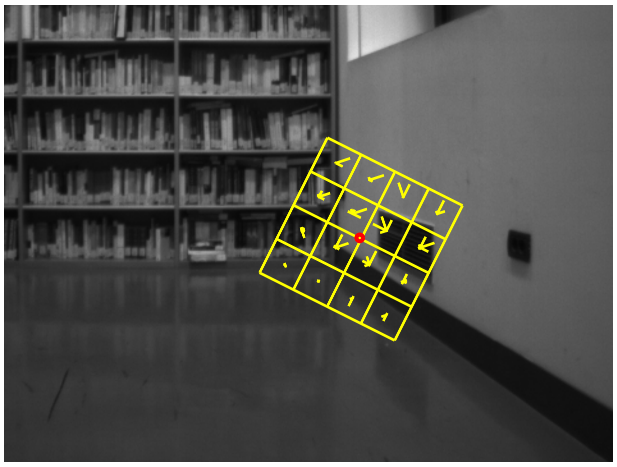


Figure 3.3: Illustration of a SIFT feature. The red dot marks the position of the keypoint. The vectors in the yellow grid show the gradients of all 16 images regions (zoomed).

Scale Invariant Feature Transform

The *Scale Invariant Feature Transform* (SIFT) combines a *Difference of Gaussians* (DOG) based keypoint detector with a feature descriptor based on gradient histograms. The 16×16 neighborhood of the detected keypoint is subdivide in 4×4 image patches and a gradient histogram with 8 bins for each image patch is computed. Therefore, the neighborhood of the keypoint is described by 16 gradient histograms of length 8. This results in a 128 byte feature vector.

This state-of-the art descriptor is robust against lighting, rotation, scale, and viewpoint changes. Hence, SIFT allows the identification of corresponding image points in presence of wide range image variations. Nevertheless, it is very distinctive and can be adopted without geometric constraints.

The drawback of this descriptor is high computational cost for keypoint detection and feature descriptor calculation.

Speeded-up robust features

The *Speeded-Up Robust Features* descriptor (SURF) [2] is partly inspired by the SIFT descriptor and promises faster computation. Both algorithms make use of spatial distribution of gradients. The algorithm proposed by Bay et al. combines a Harris keypoint detector with a feature descriptor based on Haar wavelet coefficients. In order to speed up computation, they use integral images.

The keypoint detection is similar to SIFT performed in a scale space and the interest points are identified by a Harris corner detector. Furthermore, the dominant orientation of each keypoint is detected. The neighborhood is described by responses of a Haar wavelet filter and results in a 64 byte feature descriptor for each keypoint. The authors show that their approach is not only faster but also more robust against noise and some viewpoint changes.

3.2.2 Feature Matching

Basically, all matching algorithms based on interest points can be described by the following workflow. Given two images P_1 and P_2, interest points of both images are extracted. Then, each interest point is assigned a descriptor that expresses its local neighborhood. In the matching step, the descriptors of P_1 are compared to all descriptors in P_2. If two are similar, they are identified as corresponding points.

There exist several measures that define the similarity of keypoints. Here, we can also distinguish between measures that are defined for patch-based descriptors and feature-based descriptors.

The similarity measure for SIFT or SURF features is relatively simple. Since the de-

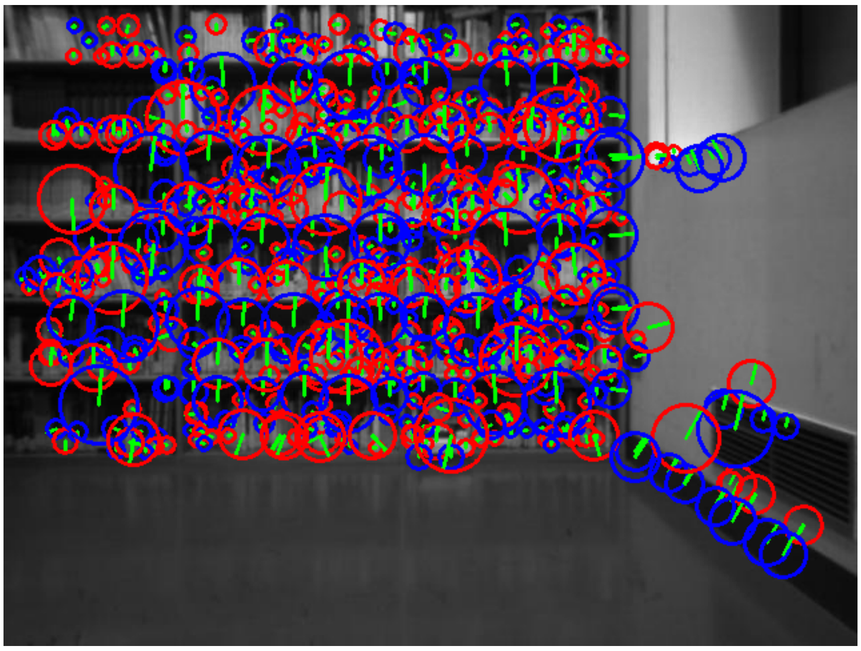


Figure 3.4: SURF Feature Descriptor. The center of the circle is the detected keypoint, whereas the size of circle refer to the scale of the keypoint. The green line indicates the dominant direction of the image patch.

scriptor itself is robust against viewpoint and lightening changes, the matching is carried out by a nearest neighbor search in feature space. Typically, the Euclidean distance between two feature vectors are used as similarity measure. Because it cannot be guaranteed that a certain feature is contained in the set of features, a threshold or another criteria has to be defined to prevent false matches. Instead of a threshold on the absolute feature distance, Lowe et al. propose a threshold on the distance between the nearest and the second nearest neighbor. If those two are too close, the match is discarded. This guarantees that only distinct features are matched.

In the naive implementation, the computational cost for recovering m features in another feature set containing n elements, is $O(m \cdot n)$. This prevents a real-time implementation if both m and n are large. Several approaches exist for reducing the computational cost for matching, e.g. the bag-of-words method presented in [28] or a nearest neighbor search in a k-dimensional-tree (KD-tree).

In order to compare the similarity of patch-based descriptors, many different measures have been developed. Given two image patches I and J, basically all measures somehow calculate a pixel wise correlation. For example, the *Sum of Absolute Differences* (SAD) is calculated as follows:

$$SAD = \sum_{W} |I - J|,$$

where $\sum_W I$ denotes the sum of all pixel inside the window around I. A measure that is more robust against illumination changes is the *Zero Mean Sum of Squared* (ZMSSD) distance and is defined as follows:

$$ZMSSD = \sum_{W} ((I - \bar{I}) - (J - \bar{J}))^2, \tag{3.7}$$

where $\bar{I}$ denotes the mean intensity of all pixel in patch I.

The advantage of the patch-based correlation is the low computation cost, because a feature extraction step is not required. In applications that provide an approximate guess for the corresponding patch location, ZMSSD can be a proper measure for corresponding point identification.

3.3 Multiview Reconstruction

One part of our SLAM algorithm is to reconstruct the environment by using one or more images. In this chapter, we explain the triangulation of a 3D point given two corresponding image points. Furthermore, we discuss the bundle adjustment method that optimizes the reconstructed 3D structure if a 3D point is visible in more than two images.

3.3.1 Triangulation

In Section 3.1.2, we explained the mapping of a 3D point X to a 2D image point x. This section elaborates on the inverse problem, which is the reconstruction of a 3D point given 2D image points.

Given the image point x in homogenous coordinates $x = w(u, v, 1)$ and splitting the projection matrix P into rows, the formula $x = P \cdot X$ can be rewritten as follows:

$$\begin{aligned} wu &= P_1^T X \\ wv &= P_2^T X \\ w &= P_3^T X, \end{aligned}$$

where P_i denotes the i-th row of P. Replacing w in the first two equations leads to the linear equation system

$$\begin{aligned} P_3^T X u - P_1^T X = 0 \\ P_3^T X v - P_2^T X = 0 \end{aligned}$$

Since X is a 4×1 vector, the equation system is under-determined and at least two image points $x_l = (u_l, v_l, 1)$ and $x_r = (u_r, v_r, 1)$ taken from two different images P_l, P_r are used to solve this equation system. Hence, X is calculated by

$$\begin{pmatrix} P_{l,3}^T u_l - P_{l,1}^T \\ P_{l,3}^T v_l - P_{l,2}^T \\ P_{r,3}^T u_r - P_{r,1}^T \\ P_{r,3}^T v_r - P_{r,2}^T \end{pmatrix} X = 0 \tag{3.8}$$

X is then the eigenvector corresponding to the smallest eigenvalue of the matrix.

Geometrically, Equation (3.8) are the mathematical solution of the intersection of two rays going through the image points x_l and the principal point of P_l, respectively x_r and P_r.

In real applications, the corresponding points in the image are disturbed by noise. Equation (3.8) can only be solved by approximation. The accuracy of the solution depends on the noise and the baseline between both cameras. Figure 3.5 shows, that especially the uncertainty in z-direction highly depends on the baseline. So, a larger baseline allows a more accurate reconstruction of three-dimensional points.

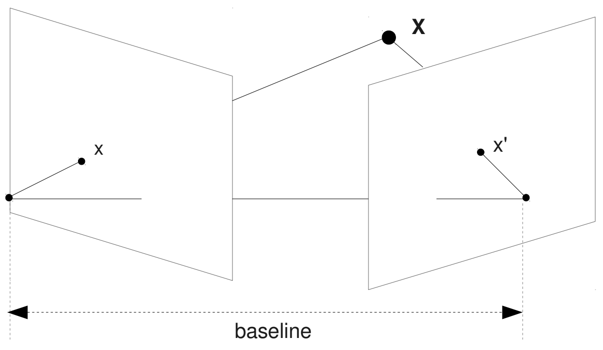


Figure 3.5: Geometrically, the triangulation is the intersection of two rays going through the principal point and the image point x_l, respectively x_r. Assuming a small baseline between both cameras, small changes in x_r or x_l results in a large variation in the reconstructed point X.

3.3.2 Bundle Adjustment

The triangulation of a 3D point requires at least two projections taken from two different points of view. If more than two projections exist, the triangulation formula is overdetermined and due to noisy corresponding point estimation, $x = P \cdot X$ cannot be satisfied exactly. Furthermore, the triangulation algorithm requires precise camera poses, whereas in many applications the poses cannot be estimated exactly. Hence, the accuracy of the reconstructed 3D point depends on the noise introduced by inexact correspondences and camera pose estimation.

In this chapter, we introduce the concept of bundle adjustment that optimizes the camera poses and the reconstructed 3D points simultaneously. We also discuss the problem of mismatched corresponding points (outliers) and some implementation details.

Consider a 3D point X_j that is visible in a set of cameras P^i. x^i_j denotes the image point of the j-th 3D point seen* in the i-th camera. Bundle adjustment is defined as the problem to find a set of camera matrices P^i and 3D points X_j that minimizes the

*A measurement denotes a point in image space that is identified as the projection of a certain 3D point. Typically, a 3D point is linked to an interest point that is used to identify a measurement.

reprojection. Assuming Gaussian noise, the maximum Likelihood solution is given by

$$\min_{X_j, P^i} \sum_{ij} d(\hat{P}^i \cdot \hat{X}_j, x_j^i)^2, \tag{3.9}$$

where $d(x, y)$ denotes the Euclidean distance between x and y.

Typically, the minimization problem consists of a very large number of parameters to be optimized, i.e. each projection matrix has 11 parameters and each 3D point has 3 DOF. Practically, a real-world bundle adjustment problem consists of several thousand 3D points visible in some hundreds cameras. Hence, the optimization problem enfolds several thousand parameters.

The problem can be reformulated to a standard non-linear least-squares optimization problem. All projection matrices P_i and 3D points X_j are rearranged to a parameter vector $\beta = (\beta_1, \ldots, \beta_n)$. The measurements x_i^j are converted to a $1 \times m$ vector $\mathbf{x}$. Given $\mathbf{x}$, we try to refine the parameters β such that the Euclidean distance between the reprojected 3D points and the measurement is minimal.

Due to the nonlinear relationship between the parameters and the measurements, a direct solution of the problem cannot be formulated. Assuming the number of measurements

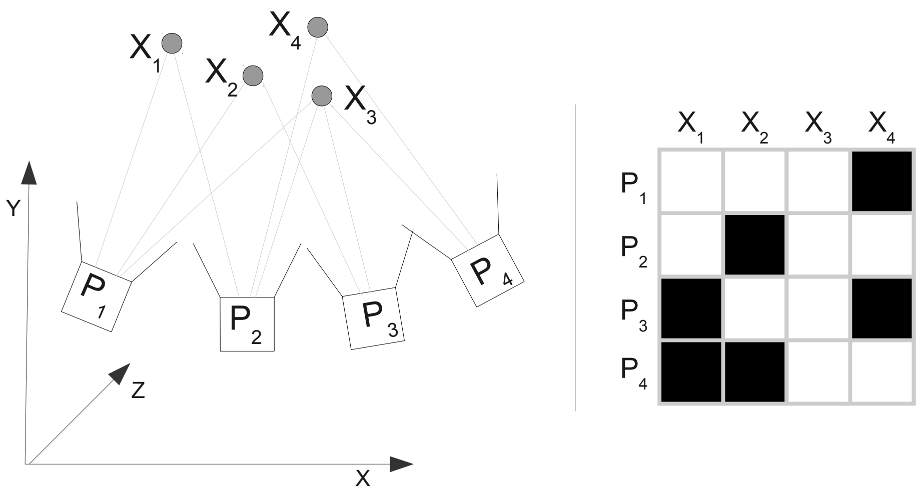


Figure 3.6: Bundle Adjustment Problem. Four 3D points A, B, C and D are measured by four cameras. Due to scene geometry or matching errors, not every point is visible in every camera. The information which point is visible in which camera can be represented in a *connectivity matrix*. If entry i/j is black, the point j is not visible in camera i. For example, point D is visible in camera 2 and 4.

m is greater than the number of parameters n ($m > n$), we have to deal with an overdetermined nonlinear system of equations. So, an iterative nonlinear least-squares solver is used to estimate an optimal parameter vector β.

Let $F(\beta)$ be the nonlinear vector model function called *objective function*,

$$F(\beta) = \begin{pmatrix} f_1(\beta) \\ \vdots \\ f_m(\beta) \end{pmatrix}$$

where $f_m(\beta) : \mathbb{R}^n \to \mathbb{R}$ computes the reprojection error of one 3D point X_j in a single camera P^i:

$$f_m(\beta) = d(P^i\, X_j, x_i^j)^2,$$

which are also known as residuals. In the following, we use f_m instead of $f_m(\beta)$. In order to optimize the parameter vector β, we minimize the squared Euclidean norm:

$$\|F(\beta)\|_2^2 = F^T(\beta)F(\beta) \tag{3.10}$$

$$= \sum_{i=1}^{m} f_i(\beta)^2, \tag{3.11}$$

describing a function from $\mathbb{R}^n \to \mathbb{R}$.

Formally, the minimum of an objective function $\mathbb{R}^n \to \mathbb{R}$ is defined as follows: A function f has a *local minimum* at point x^*, if $f(x^*) \leq f(x)$ and $\|x^* - x\| < \epsilon$, $\epsilon > 0$. A point x^* is called *global minimum*, if $f(x^*) < f(x), \forall x$.

A necessary but not sufficient condition for a minimum at x^* of a differentiable function $f : \mathbb{R} \to \mathbb{R}$ is that the gradient at x^* is zero: $f'(x^*) = 0$. This property can be extended to a multidimensional function $f : \mathbb{R}^n \to \mathbb{R}$ by defining the gradient of f as follows:

$$\nabla f(x) = (\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n})^T.$$

Hence, a point x^* may be a minimum of f, if the gradient vanishes:

$$\nabla f(x^*) = 0. \tag{3.12}$$

Putting Equations (3.12) and (3.10) together, a minimum $\hat{\beta} \in \mathbb{R}^n$ of $F(\beta)$ holds the following equation:

$$\nabla(F^T(\hat{\beta})F(\hat{\beta})) = 2J^T(\hat{\beta})F(\hat{\beta}) = 0, \tag{3.13}$$

where $J(\hat{\beta})$ is the Jacobian:

$$J(\beta) = \begin{pmatrix} \frac{\partial f_1(\beta)}{\partial \beta_1} & \cdots & \frac{\partial f_1(\beta)}{\partial \beta_n} \\ \vdots & & \vdots \\ \frac{\partial f_m(\beta)}{\partial \beta_1} & \cdots & \frac{\partial f_m(\beta)}{\partial \beta_n} \end{pmatrix} \tag{3.14}$$

Basically, all nonlinear optimization methods work iteratively. Starting from an initial guess $\beta^{(0)} = (\beta_1^{(0)}, \dots, \beta_n^{(0)})$, the parameter vector β is optimized iteratively:

$$\beta^{(k+1)} = \beta^k + \Delta\beta, \tag{3.15}$$

where $\Delta\beta$ is called *shift vector*.

There exist many different methods for calculating the shift vector $\Delta\beta$. The simplest method is the gradient descent, which is defined as follows:

$$\Delta\beta = -\lambda_k J_k F(\beta^{(k)}), \tag{3.16}$$

where λ_k is a positive step width that can be adjusted in each iteration. In order to guarantee that this algorithm reaches a local minimum, λ has to be small, which leads to slow convergence.

A common faster algorithm is the *Gauss-Newton* method. The shift vector $\Delta\beta$ can be derived by solving

$$(J_k^T J_k)\Delta\beta = -J_k^T F(\beta^{(k)}), \tag{3.17}$$

where $\beta^{(k)}$ denotes the parameter vector at iteration k, and J_k the Jacobian matrix evaluated at $\beta^{(k)}$. This equation can be solved by Cholesky- or QR-decomposition. Here, convergence behavior depends on the guess of $\beta^{(0)}$. If $\beta^{(0)}$ is close to a minimum, the algorithm converges nearly quadratically, whereas a bad initial guess reduces the convergence speed or it even diverges.

The *Levenberg-Marquardt* algorithm combines the Gauss-Newton and the gradient descent method to overcome the divergence problem. The shift vector $\Delta\beta$ is calculated by:

$$\Delta\beta = -(J_k^T J_k + \lambda I)^{-1} J_k F(\beta^{(k)}), \tag{3.18}$$

where λ is a damping factor adjusted in each iteration. If $\lambda = 0$, Equation (3.18) turns into the Gauss-Newton method (3.17). If $\lambda \to \infty$, $\Delta\beta$ Equation (3.18) turns into the gradient descent method as described in Equation (3.16).

At iteration $k = 0$, λ is set to a large value in order to start with the gradient descent method. If it gets closer to a minimum, λ is set to smaller values so it behaves like the Gauss-Newton method. The algorithm is repeated until a stopping criterion is reached, such as a maximum number of iterations or $\|\Delta\beta\|$ is below a threshold.

An issue of the Levenberg-Marquardt algorithm is the evaluation and the inversion of the Jacobian matrix, because a typical bundle adjust problem consist of several thousand parameters and equations. Since the Jacobian matrix of a bundle adjustment problem is sparse, the inversion of J^TJ can be avoided. Therefore, β is constructed by concatenation of two vectors $\mathbf{a}$ and $\mathbf{b}$

$$\beta = (\mathbf{a}^T|\mathbf{b}^T),$$

where $\mathbf{a} = (\mathbf{a}_1^T, \mathbf{a}_2^T, \ldots, \mathbf{a}_i^T)$ contains the parameters of the projection matrices P_i and $\mathbf{b} = (\mathbf{b}_1^T, \ldots, \mathbf{b}_j^T)$ is a vector containing the 3D point parameters X_j. Reordering the parameters in that way leads to a block structured Jacobi matrix $J = [A|B]$, where A and B are the Jacobians of F with respect to $\mathbf{a}$ and $\mathbf{b}$. Figure 3.7 shows the block structure of

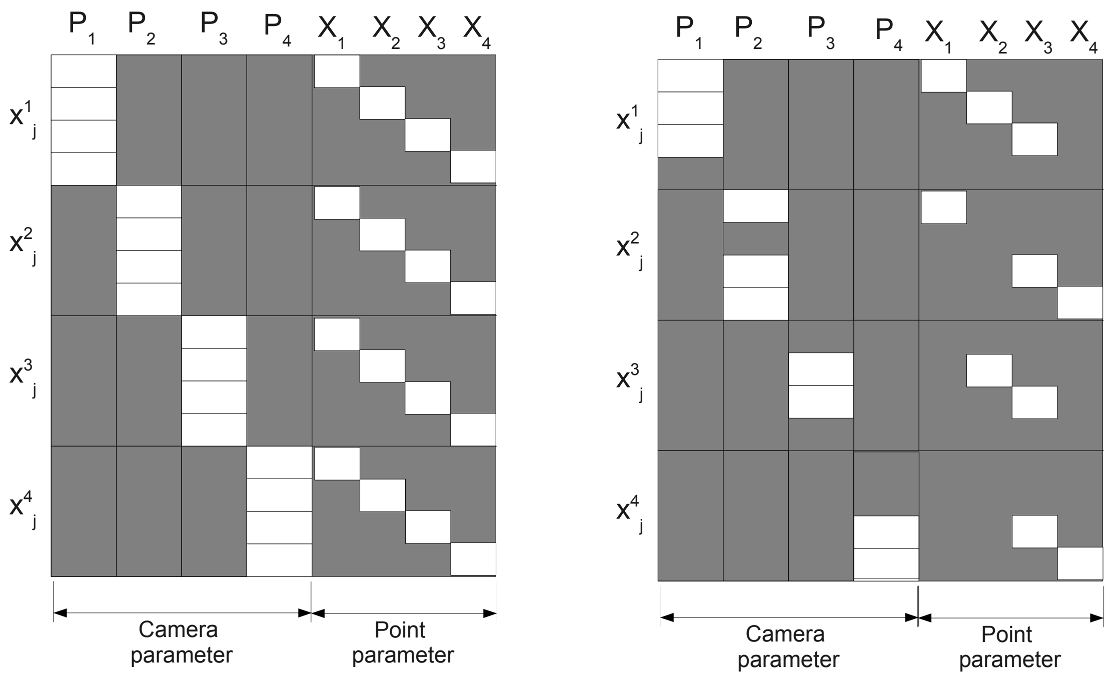


Figure 3.7: Jacobian Structure of Bundle Adjustment. The left figure shows the block-wise structure of the Jacobian matrix. In this example, each map point X_j is visible in each camera P_i. The right figure is the Jacobian matrix of the example given in Figure 3.6. If a map point is not visible in a certain camera, all entries in the Jacobians are zero.

the rearranged Jocobian matrix. Equation (3.18) can be rewritten as

$$\left[\begin{array}{c|c} A^T A & A^T B \\ \hline B^T A & B^T B \end{array}\right] \left(\begin{array}{c} \Delta\beta_a \\ \hline \Delta\beta_b \end{array}\right) + \lambda I = \left(\begin{array}{c} A^T F(\beta^{(k)}) \\ \hline B^T F(\beta^{(k)}) \end{array}\right), \tag{3.19}$$

Adding the damping factor to the diagonal entries of $A^T A$ and $B^T B$, alters them to $(A^T A)^* = U*$ and $(B^T B)^* = V^*$. So, Equation (3.19) can be rewritten as

$$\begin{bmatrix} U^* & W \\ W^T & V^* \end{bmatrix} \begin{pmatrix} \Delta\beta_a \\ \Delta\beta_b \end{pmatrix} = \begin{pmatrix} \epsilon_A \\ \epsilon_B \end{pmatrix}, \tag{3.20}$$

where ϵ_A and ϵ_B rewrite of the right side of the equation only. In order to solve this system of equations, both sides are left multiplied by

$$\begin{bmatrix} I & -WV^{*-1} \\ 0 & I \end{bmatrix}.$$

This results in a system of equations where the top right block vanishes:

$$\begin{bmatrix} U^* - WV^{*-1}W^T & 0 \\ W^T & V^* \end{bmatrix} \begin{pmatrix} \Delta\beta_a \\ \Delta\beta_b \end{pmatrix} = \begin{pmatrix} \epsilon_A - WV^{*-1}\epsilon_B \\ \epsilon_B \end{pmatrix}$$

The camera update parameters $\Delta\beta_a$ can be computed by solving the top half of equations:

$$(U^* - WV^{*-1}W^T)\Delta\beta_a = \epsilon_A - WV^{*-1}\epsilon_B$$

The point parameters are calculated by back-substition of $\Delta\beta_a$ in

$$V^*\Delta\beta_a = \epsilon_B - W^T\Delta\beta_a$$

A bundle adjustment problem often consists of thousands of parameters to be optimized. The computational amount can be reduced by making use of the Jacobian's sparseness, which stems from the fact that not every 3D point is visible in every camera. In an optimized implementation, this reduces the computational amount as well as the memory consumption. For more implementation details, we refer to the book *Multiview Geometry* of Hartley & Zisserman [16].

3.3.3 Robust Bundle Adjustment

The standard bundle adjustment optimizes the reprojection error in a least-squares sense, which is less robust in the presence of outliers. Outliers are image points identified spuriously as projections of a 3D point X_j. In many cases, these points produce large reprojection errors and therefore have strong influence on the optimization result. In this chapter, we present several approaches to reduce the influence of outliers.

Random Sample Consensus

Random Sample Consensus (RANSAC) [11] is a common method for robust model fitting. The idea is to create a hypothesis from a randomly selected subset of all measurements. Then, the remaining measurements are tested, if they are represented by the model as well. If they support the model, they are considered as inliers. The procedure of sampling and testing is repeated several times. The model with the highest number of inliers is selected as resulting model.

In case of bundle adjustment, the optimization problem is solved with the minimum number of required measurements. Then the reprojection error of the remaining observed 3D points is calculated and the set of inliers is collected. A measurement is classified as inlier, if the reprojection error is below a certain threshold.

The RANSAC algorithm is still robust in presence of a large set of outliers. The disadvantage is that no upper bound on the number of iterations can be given to achieve a certain model quality. The computational amount of a RANSAC fitting depends on the complexity of model fitting, which is relatively high in bundle adjustment. So, this limits the use of RANSAC to small bundle adjustment problems with a few number of points and views.

M-Estimator

The sensitivity to outliers in bundle adjustment arises from the quadratic error function. Hence, the idea to limit the influence of outliers is to find a robust error weighting function. In general, bundle adjustment can written as

$$min \sum_m \rho(f_m), \tag{3.21}$$

where $\rho : \mathbb{R} \rightarrow \mathbb{R}$ is a symmetric, positive-definite function that has a unique minimum. In case of the standard bundle adjustment formulation, $\rho(\cdot)$ is chosen as $\rho = \frac{1}{2}x^2$, which results in Equation (3.9). In order to be robust against outliers, $\rho(\cdot)$ is replaced by a so-called M-Estimator. The necessary condition for a minimum $\hat{\beta}$ of the previously defined error function is given by

$$\sum_m \psi(f_m)\frac{\partial f_m}{\partial \beta_n} = 0, \tag{3.22}$$

where ψ is defined as the derivative of ρ:

$$\psi(x) = \frac{\partial \rho(x)}{\partial x}$$

and is called *influence function* of ρ. Based on the influence function, we define a *weighting function* w

$$w(x) = \frac{\psi(x)}{x},$$

which guarantees that large residuals have less influence in the result. Equation (3.22) turns into a system of equations of weighted gradients with respect to parameter β:

$$\sum_m w(f_m) f_m \frac{\partial f_m}{\beta_n} = 0. \tag{3.23}$$

This system of equations can also be solved by the Levenberg-Marquardt algorithm, where the additional weights alters the equation of the shift vector update as follows:

$$\nabla\beta = -(J_k^T w_k J_k + \lambda I)^1 w_k J_k F(\beta^{(k)}) \tag{3.24}$$

where w_k is a diagonal weighting matrix depending on the residuals in the k-th iteration.

In Figure 3.9, two M-Estimators and the standard quadratic error function are shown. Huber's M-Estimator [18] combines the standard squared error function with a linear weighting and is defined as

$$\rho(x) = \begin{cases} \frac{x^2}{2} & \|x\| < k \\ k(\|x\| - \frac{k}{2}) & \|x\| \geq k \end{cases} \tag{3.25}$$

where k is a tuning factor that defines the change between the squared and the linear portion of the M-Estimator. As shown in Figure 3.9, measurements with residuals smaller than k get constant weight, whereas residuals greater than k have decaying influence. But obviously, each measurement affects the optimization result.

In contrast, the M-Estimator designed by Tukey [38], is a so-called cut-off estimator, where the robust cost function is given by

$$\rho(x) = \begin{cases} \frac{k^2}{6}(1-[1-(x/k)^2]^3) & if\|x\| \leq k \\ \frac{k^2}{6} & if\|x\| > k \end{cases}. \tag{3.26}$$

k is again a tuning factor that controls the influence of a measurement dependent on its residual. If the residual is above k, the weighting function vanishes and does not influence the result. Hence, the Tukey function binary classifies measurements as in- and outliers.

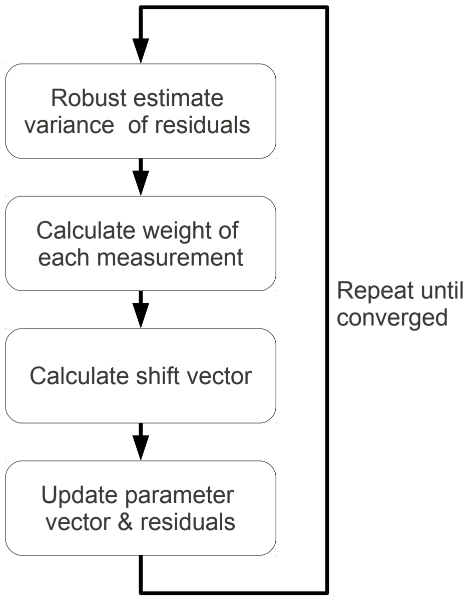

Robust estimate variance of residuals
Calculate weight of each measurement
Calculate shift vector
Update parameter vector & residuals
Repeat until converged

Figure 3.8: Robust optimization process. First, the variance of the residuals are calculated, followed by determining the weights for each measurement. Then, the shift vector is calculated. At last, the parameter vector is updated and the new residuals computed.

Both M-Estimators require a constant k that is a threshold for outlier identification and therefore the estimation of this parameter is a critical part. In an iterative optimization method, the residuals decrease typically with the number of iterations and setting k to a constant value is not suitable. However, the expected percentage of outliers can be specified, which typically varies between 5% and 10%. Assuming $F(\beta)$ can be modeled by a zero mean Gaussian distribution with variance σ, 95% of residuals are accepted as inliers, if $k = 4.6851\ \sigma$ for Tukey and $k = 1.345\ \sigma$ for Huber.

Due to the sensitivity to outliers, the residuals variance should not be calculated from the dataset directly, but a robust variance estimator should be used. Rousseeuw [31] proposes the following variance estimator:

$$\sigma \approx 1.4826(1 + \frac{5}{m-n})\sqrt{\tilde{x}}, \tag{3.27}$$

where m is the length of X and n is the dimensionality of the residuals. The inner fraction $\frac{5}{m-n}$ compensates effects that occur if m is small. For a more detailed explanation of the constants we refer to [31].

The complete workflow for a robust optimization procedure is shown in Figure 3.8. After estimating the variance of the residuals using Equation 3.27, the tuning factor based on the variance is calculated and the individual weight for each residual is determined. This step is followed by the calculation of the shift vector $\nabla\beta$. At last, the parameter vector β is updated and the new residuals are recalculated. All steps are repeated until a stopping criteria like a maximum number of iterations is reached.

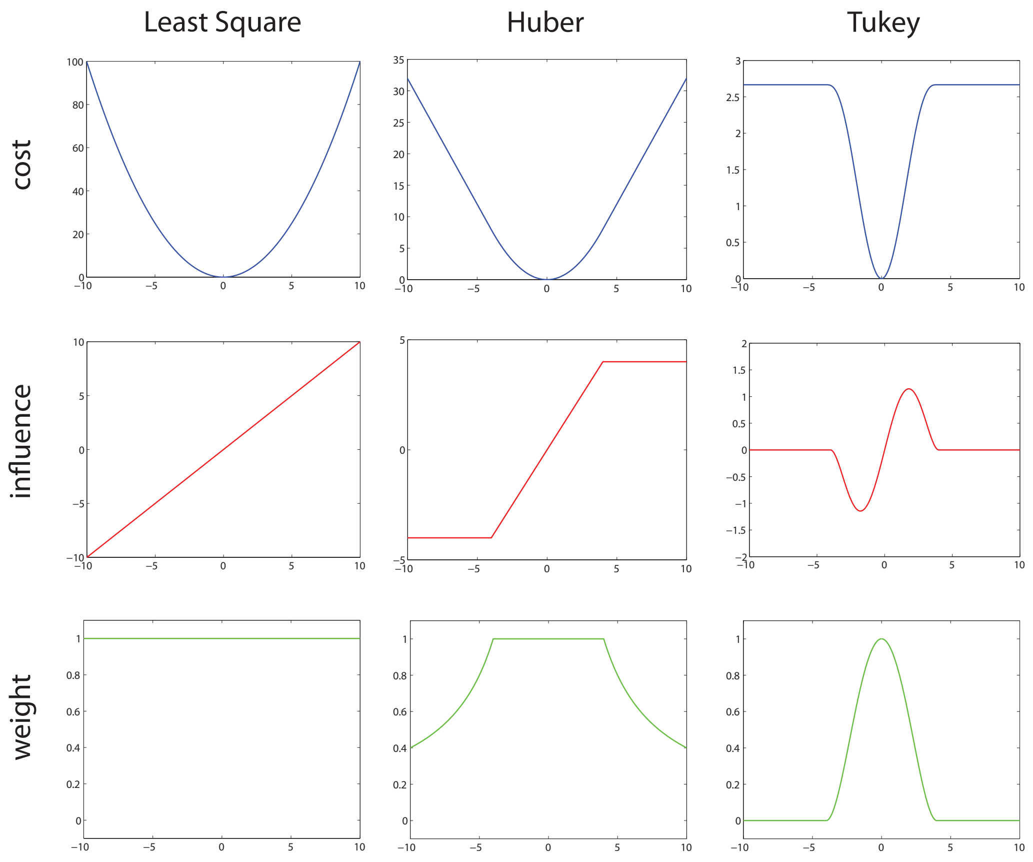


Figure 3.9: This figure shows three different error weighting functions. The weighting function of the Least Squares function w (last row) is constant, which indicates that each measurement has the same portion in the result. Huber and Tukey are two well-known M-Estimators. Their weighting functions prefer measurements with small errors.

Chapter 4

Localization

Contents

Every SLAM algorithm comprises two parts: Building a map and localizing within an existing map. In this chapter, we focus on the localization part and assume the map is already known. Creation of the map is described later in Chapter 5.

Localization is the process of determining the robot pose within a given map. A *pose* is a 6-tuple which describes the translational offset (three parameters) and the orientation (three parameter) with respect to the map origin. An accurate pose estimate is a key component in many autonomous robot applications. If the robot does not know its position in the environment, any high-level interaction task becomes difficult. Some authors state that the localization problem is one of the fundamental problems to provide a robot with real autonomous capabilities.

The localization task has a number of different instances of increasing difficulty:

- Local Localization
- Global Localization
- Kidnapped Robot

In the *local* localization, also called *pose tracking*, the robot knows its initial pose in the map. The goal is to keep track of the pose while the robot navigates in its environment.

This method is used whenever an initial pose is available.

A global localization is performed, if no initial pose is available. In the global localization, also known as *wake-up* problem, the robot may be started anywhere on the map and it has to localize itself from scratch.

A much harder issue is the *Kidnapped robot* problem. Here, the robot is localized exactly within the map but then the robot gets blindfolded und moved (*kidnapped*) to a new position. Hence, the robot also has to detect that circumstance and has to perform a global localization.

Localization can be performed in two different kinds of environment: Static and dynamic. In the static case, the environment at localization time is identical to the environment at map creation time and the robot is the single moving object. Localization in a dynamic environment is much harder due to other moving objects. The map and the current environment differ and therefore a robust localization is much more difficult.

In this chapter, we discuss our approach for solving the local and global localization in a static environment using one or more cameras. The chapter is organized as follows: In Section 4.1, the structure of the given map is outlined. In Section 4.2, we describe the pose tracking within a given map followed by the global localization method.

4.1 Map

Since the map is the fundamental data structure in a SLAM algorithm, its design is very important. The requirements on the map are versatile and sometimes oppositional: a representation of the map should be usable for many tasks like navigation, localization, obstacle avoidance. Ideally, it is memory efficient, detailed, and readable by humans. So, often a tradeoff between the requirements has to be found. Its representation also depends on the type of SLAM algorithm and vice versa.

The first idea for a map representation is a set of all acquired images, because they are provided directly by the camera and no further treatment is required. However, the appearance of the images depends on a large number of parameters, e.g. illumination or view-point to name only a few. Furthermore, a map of images is not memory efficient.

As known from other fields in computer vision like object detection, local features as described in Section 3.2 are often a sufficient way to encode image information. Using a local feature descriptor like SIFT or SURF reduces the amount of memory required and the influence of viewpoint and illumination. The drawback of a local feature based map is the increased computational amount for feature extraction. However, they can be used to identify the same point in different images, which is important for many pose estimation algorithms.

Motivated by the visual odometry algorithm proposed by Nister et al. [27], who localizes a robot against local features, in our approach a three-dimensional position is assigned to each local feature. This enables us to identify three-dimensional points in the current image and to calculate the pose and orientation of a image with respect to the given points.

Klein et al.[20] enhanced Nesters idea and introduced the *keyframe* concept. They noticed that image sequences along a robot path contain a lot of redundant information. Hence, they build a map from such images containing new information. The method presented in this thesis is closely related to this approach.

Based on the idea of tracking a camera by visual local features, the map consist of so-called *keyframes*, denoted by K_l:

$$Map = \{K_1, \dots, K_l\} \tag{4.1}$$

Each keyframe K_l is a set of projection matrices P_i and map points M_j:

$$K_l = \begin{cases} P_i & i = 1, \dots, I \\ M_j & j = 1, \dots, J \end{cases} \tag{4.2}$$

The number of projection matrices as well as the number of map points may vary in each keyframe.

A projection matrix P_i is defined as

$$P_i = K_i[R_i|\boldsymbol{t}_i], \tag{4.3}$$

where the rotation matrix R_i and translation t_i of the cameras are given with respect to a global world coordinate frame $\mathcal{C}$. K_i is the intrinsic camera calibration matrix. $P_{l,i}$ denotes the i^{th} projection matrix in keyframe l. A map point M_j is defined as:

$$M_j = \begin{cases} \boldsymbol{X}_j = triang(x, x', P, P') \\ d(x_i, P_i) \\ n(x_i, P_i) \end{cases} \tag{4.4}$$

$\boldsymbol{X}_j$ denotes the triangulated 3D position of two corresponding image points x, x' extracted from images P and P' (P and P' are not necessarily two images of the same keyframe).

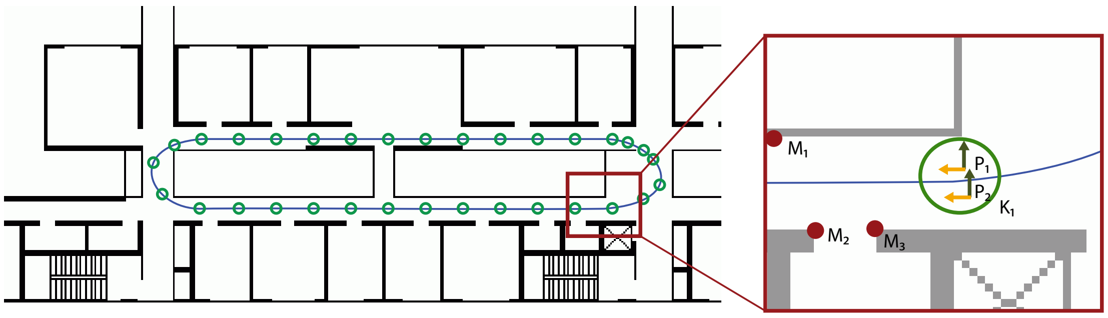


Figure 4.1: Map representation. The map shows the trajectory of a moving robot (blue). The green dots indicate the positions a new keyframe is added to the map. The display detail illustrates a keyframe that consists of two images. Three map points (red) are triangulated using both images.

$d(x, P)$ describes a local interest point descriptor such as SIFT or SURF of x in image P and $n(x, P)$ is a local neighborhood around x in image P.

The keyframe concept is a tradeoff between memory consumption and accuracy of the reconstructed environment. The difficulty in this concept is to find the correct frequency of keyframes. In case of insufficient keyframes, the structure of the environment underpresented and localization gets impossible. On the other hand, if too many keyframes are stored, the memory consumption increases drastically and only a limited area can be mapped. In our implementation, we figured out that an average distance of 40 cm is a good tradeoff between accuracy and memory consumption. Figure 4.1 illustrates the map representation.

The use of the map for localization is presented in the next section.

4.2 Local Localization

In local localization the task is to calculate the current pose given a single image and a position estimate of the last taken image. To solve this task, we identify existing map points M_j in the current image and orient the pose of the camera with respect to the identified map points.

A projection matrix at time t is defined as

$$P_t = K \underbrace{[R_t|\boldsymbol{t}_t]}_{C_t}, \tag{4.5}$$

where K is the intrinsic camera matrix and C_t defines the pose with respect to $\mathcal{C}$.

The localization comprises two steps:

- Predicting pose C_t based on C_{t-1}
- Refine pose C_t

The approximate pose C_t is predicted using C_{t-1} and a velocity based motion model. Second, the predicted camera pose is refined using known map points M_j.

4.2.1 Prediction

In order to predict the pose C_t of the robot, we make use of a decaying velocity model.The current velocity v_t is calculated using C_{t-1} and C_{t-2}:

$$v_t = \frac{C_{t-1} - C_{t-2}}{1fps}$$

In order to be robust against fast camera movements v_t is smoothed over time:

$$v_t = 0.5 \cdot v_{t-1} + 0.5 \cdot v_t \tag{4.6}$$

The prior position C_t is then estimated by the recursive function

$$C_t = C_{t-1} + \alpha \cdot (v_t \cdot 1fps)$$

A disadvantage of this simple model is the assumption of a constant frame rate in equation (4.2.1). Lacking frames are not considered and therefore affects the accuracy of the prediction. A simple solution is the use of timestamps to get a more proper velocity estimation in case of missing frames.

Another issue is the accuracy of the rotation estimation. A small error here accumulates to a large global displacement after a certain time. Especially abrupt rotations with high speed are not predicted well by this model. Therefore, the rotation of the camera is determined in an additional step.

Klein et al. developed a simple and fast method performing that task. They subsampled the original image to 40x30 pixel and applied a Gaussian blur of $\sigma = 0.75$ pixel. This image is then aligned to the previous subsampled image by minimizing the sum-of-squared distances. They use second-order minimization over three parameters in image space: translation in x and y direction and rotation. At most 10 iterations are allowed for convergence. The resulting transformation is then converted to the best-fit 3D pose of the camera. A more detailed description of this method can be found in [20].

4.2.2 Correction

The predicted pose of the robot is based on the motion model, which assumes smooth movement between frames. This assumption does not hold if the robot increases or decreases its speed or starts to rotate. Therefore, the map information in conjunction with the current camera image of the robot is used to optimize C_t.

The refinement step can be divided in two parts: All map points M_j are determined that are visible in P_t and their corresponding location in P_t is identified. This 3D - 2D correspondence is used to optimize pose C_t by minimizing the reprojection error of M_j in the image P_t.

4.2.2.1 Map Point Association

In order to determine the corresponding image location of a map point M_j in image P_t, all X_j are reprojected to the current frame:

$$\hat{x}_j = P_t X_j \tag{4.7}$$

If $\hat{x}_j$ is located within P_t, M_j is potentially visible. To identify the exact position of the map point in the P_t, the neighborhood of $\hat{x}_j$ is examined. All points of interest in that area are compared to the neighborhood stored in M_j using zero mean sum of squared differences (ZMSSD). If the difference is below a certain threshold, M_j is considered as found in P_t. This results in a list of correspondences between map points M_j and their corresponding image points x_j in the current image.

4.2.2.2 Pose Correction

Using a set of 3D - 2D correspondences, we can estimate the pose of the frame with respect to the map points.

Given is the set of map points X_j and their corresponding image coordinates $\boldsymbol{x}_j$. Both quantities are represented in homogeneous coordinates. The pose C_t of the camera can be computed by

$$\boldsymbol{x}_j = K C_t X_j, \tag{4.8}$$

assuming the intrinsic camera matrix K is known *a-priori* and C_t has 6 DOF. The equation system (4.8) can be solved with at least three 3D - 2D correspondences. If more corresponding points exist, the equation is over-determined and it can be solved in a least-squares sense. That means, the *reprojection error* is minimized:

$$\epsilon(C_t) = \sum_j d(x_j, KC_tX_j)^2 \tag{4.9}$$

where $d(.)$ denotes the Euclidean distance between two points in image space.

In a least-squares solution, outliers have great influence on the optimized frame position. To be more robust against false 3D - 2D matches, a minimization of a M-Estimator weighted distance is reasonable. So, equation (4.9) turns into the following objective function:

$$\epsilon(C_t) = \sum_j \rho(d(x_j, KC_tX_j)), \tag{4.10}$$

where $\rho(\cdot)$ denotes the M-Estimator function. We decided to use Turkeys [38] function and assumed that 5% of the measurements are outliers. So, k is defined as $k = 4.6851\sigma$, where σ is the robustly determined variance of all measurements using Equation 3.27. The minimum of (4.10) is found after at most ten iterations of a reweighted least squares algorithm.

4.3 Global Localization

Localizing a robot within an existing map without an initial pose is a much harder task than keeping track of a pose. A common approach for such a global localization problem is a particle filter that assumes that the robot is located at each point with same probability at first. This probability distribution is updated after each new measurement. So, after a certain time the distribtion should have changed towards one or more poses is much more likely than all the others.

Our approach to solve the task is somewhat different. As described in Section 4.1, our map consist of keyframes that are taken periodically from the robots path. Hence, the keyframes are a good representation of the environment. If the robot starts at an unknown position, the idea is to compare the current image P_t to all keyframe images. If we find a keyframe image that is similar to the current image, we use the pose of that keyframe to initialize the pose tracking discussed in Section 4.2.

The detection of similar images is a well known task in computer vision. A sufficient but time consuming method is to compare strong local features of P_t and all keyframe images. Such a solution is time consuming and grows exponentially with the number of projections. Thus, it is not applicable on large-scale environments and for real-time applications.

A similar task is to match a certain image against a database of thousands of images. Nister et al.[28] proposed an efficient solution for this problem. The algorithm is divided in three steps: An offline learning phase followed by an insertion and a query step. In the learning phase, they extract local features of from large number of randomly selected images and cluster them hierarchically with the $k - Means$ algorithm, i.e. all features are divided in k cells. Recursively, each cell of features is sub-divided in k cells again. The number of recursions is defined by a fixed number L. The result is a tree of depth L whose leafs contain the local descriptors. The cluster centers of each cell are the inner nodes of the tree. At the end of training only the structure and the inner nodes of the tree are stored. This tree is also called *vocabulary tree*.

To insert a new image in the database, all local features of this image are propagated down in the generic tree. The path of the feature in the tree is presented as a single integer and used later for scoring. Furthermore, each node in the tree that has been passed by a feature stores a reference to the features image. This *inverted file* structure is used in the query step.

In a database query, the local features of the query image are also propagated down in the tree. The similarity of two images can be expressed by the similarity of their feature paths in the tree. The inverted file structure speeds up the computation of the similarity because only pictures have to be taken into account that are stored in the passing nodes.

Nister et al. show that the retrieval quality is improved by creating the quantization tree from a large number of features. Therefore, the tree is computed off-line from a large number of randomly selected images. ETH Zurich's Computer Vision Group provides such a pre-calculated quantization tree. They used SIFT features for representing local image regions. Their implementation also provides an interface for insertion and retrieval. Given one sample picture, a query results in a list of the n most similar images (nearest neighbors) contained in the tree.

In our application, the vocabulary tree is filled with all features taken from any keyframe. On a global localization request, the tree is inquired for the most similar image I_n. In order to verify the result, all local descriptors of P_t are matched against I_n. If the number of matching descriptors exceeds a certain threshold, the pose of I_n is used to initialize the tracking algorithm. Because the real pose of P_0 may differ from I_n,

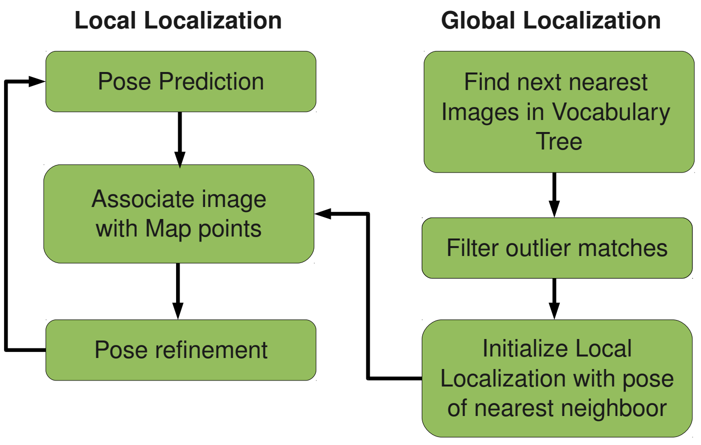

Local Localization
Pose Prediction
Associate image with Map points
Pose refinement
Global Localization
Find next nearest Images in Vocabulary Tree
Filter outlier matches
Initialize Local Localization with pose of nearest neighboor

Figure 4.2: Localization workflow. Given a predicted pose, the current image is associated with map points using a similarity metric like ZMSSD. The pose is refined by minimizing the reprojection error. If no initial pose is available, a global localization is performed. The most similar images are requested from the vocabulary tree and outliers are filtered out. The pose that is connected to the most similar image is used for initializing the local localization.

the ZMSSD map point association cannot be applied. Hence, we are using the SURF descriptors stored in I_n to identify the image location of map points in P_t.

Figure 4.2 illustrates the workflow the local and global localization.

4.4 Discussion

In this section, we discuss the advantages and limitations of our approach for local and global localization. Local localization is based on the visual odometry approach, where an image is matched against three-dimensional map points. The global localization is solved by comparing the current image against the database created by all keyframe images.

The main advantage of our local localization approach is the very low computational cost compared to other state-of-the-art approaches. Given an inital pose, the approximate pose of the robot is calculated by a simple velocity estimation. This pose is then refined by associating the current image with map points. Since we are using patch-based correlation as feature descriptor for map points, the computational effort for data association is low. Pose correction by minimizing the reprojection error is performed by at most 10 iterations of a re-weighted least squares algorithm and the number 3D-2D correspondences is typical between 50 and 300. Thus, the costs for optimization is negligible; on modern computers the local localization can be performed in realtime.

However, using patch-based correlation as a feature for data association introduces some limitations. We compare the neighborhood $n(.)$ of a map point M_j to the neighborhood of the reprojected position r_j. This presumes that the predicted pose of P_t is similar to the real pose. Otherwise, the patch-based corresponding point detector would searche in a wrong image area. Moreover, the patch-based finder has low discriminative property. In order to allow small lightening and viewpoint changes, the threshold used in the matching step has to be set to a relatively high value. This may lead to a large number of outliers and the erroneous 3D-2D association results in a wrong pose estimate, especially if the predicted pose differs from the real pose.

To overcome this problem, we could use strong local image features like SURF or SIFT for data association. In that case, the reprojected image position r_j is only used to decide if a map point may be visible in P_t. The function $l(M_j, P_t)$ is then calculated by comparing $d(.)$ of M_j to all local interest point descriptors of P_t. Thus, the 3D-2D association is less dependent on the prediction step. However, the computational cost is much higher than the patch-based method, because local interest points have to be computed in each frame P_t.

Furthermore, motion blur is a serious problem in images taken from a fast moving or rotating camera. On images distorted by motion blur, map point association using the presented methods fails. The patch-based algorithm fails because motion blur distorts the image patches of P_t. The local interest point association also fails because the keypoint detector extracts only a small number of keypoints and the their descriptors also differ from an unblurred image. Klein et al. propose in [20] a method to track blurred images. Their map additionally exists of edges extracted from keyframes because edges are preserved in images with motion blur. Hence, in the presence of motion blur, they use edgelets instead of map points to associate the current image with the map.

Like all feature-based localization algorithms our approach requires textured images. In un-textured images, the extraction of features is impossible and therefore the current image cannot be associated with map points. The predicted pose can not be refined and is only calculated by the motion model.

The global localization as described in this chapter was implemented as a proof of concept. It is used in cases local localization fails and no knowledge about the robot's pose is available.

The global localization approach based on the vocabulary tree assumes that each position of the mapped area has a unique appearance. This is not true for areas with self-similar texture like corridors. Here, all keyframes are similar and it is difficult to get a global pose using only a single image. In that case, probability based methods like particle filters are useful to estimate the pose over time. The idea here is to query the n nearest neighbors of P_0 and each neighbor serves as a pose hypothesis. Another method is proposed by Cummins et al. [7]. They also compare the similarity of images, but they take into account that the query image has to contain features that are unique for a certain place.

Another issue arises from the quantization of the vocabulary tree. Typically, the number of false matches increases with the size of the database. In that case, a post processing step has to be performed in order to detect false matches. A common method is to estimate the fundamental matrix between the query image P_t and the most similar images using a RANSAC [11] algorithm.

Chapter 5

Visual Map Building

Contents

In the previous chapter, we discussed the localization of a robot within a given map. Unfortunately, in many real world applications a map of the environment is not available. Even in many indoor applications, a construction plan of the building is not sufficient for navigation since most plans do not include information about furniture, other objects or changes in the environment. Furthermore, a constructions plan of a building typically does not contain the information needed for visual localization. Since navigation decisions performed by an autonmous robot are based on the map, an outdated or inaccurate map may cause unintended behavior of the robot. In order to guarantee an up-to-date map, it is desirable to learn the map while navigating. Assuming a correctly localized robot pose, the visual mapping task is relatively easy. When dealing with noisy robot positions and inaccurate sensor information, however, localizing and mapping becomes difficult. Thus, the goal of a SLAM algorithm is to generate a map that, given noisy sensor information, creates the most accurate map.

This chapter is structured as follows: The first section deals with the workflow of our SLAM system. Section 5.2 describes the map initialization using a stereo pair and the map extension strategy. The map update and optimization strategies are given in Section 5.3. In Section 5.4, we describe the loop detection and correction algorithm. In Section 5.5, we propose a sensor fusion method that preserves connectivity of keyframes even if visual features are not available. Section 5.6 summarizes this chapter and discusses the advantages and disadvantages of our approach.

5.1 SLAM Workflow

The workflow of our SLAM algorithm is shown in Figure 5.1. After initalizing the map with a stereo image pair, a new image is aquired from a single camera. The pose of this camera is then localized with respect to the existing map and the state of the robot is determined. If it is exploring, the map is extended and checked for loop detection is triggered afterwards. This process is started over by acquiring the next image.

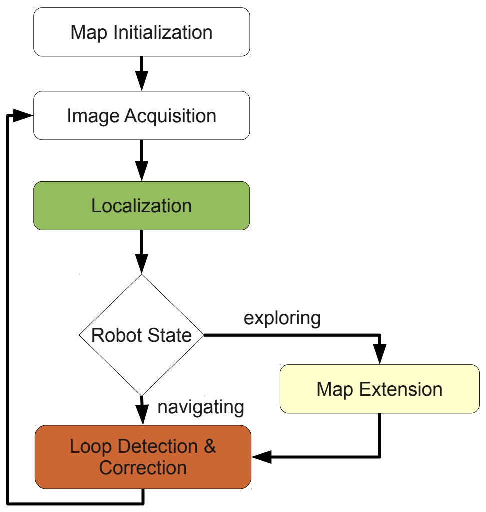


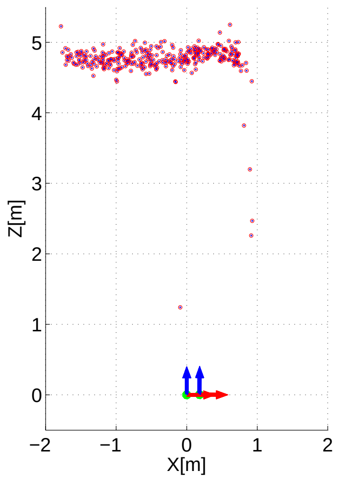
Figure 5.1: Flowchart of our basic SLAM algorithm. After map initialization, the robot grabs a new image from the camera and starts the localization. If the robot explores unknown environment, the map is extended. Otherwise, the next image is captured and the localization starts again.

The workflow shows the advantage of seperating the localization process from the map building process. Each time a new image is acquired, the pose of the robot has to be determined, whereas the map only has to be updated, if the robot explores an unknown environment. Assuming the robot explores a new environment with limited speed, the map update process is invoked much less frequently than the localization process. Therefore, the map update can be more expensive than the localization.

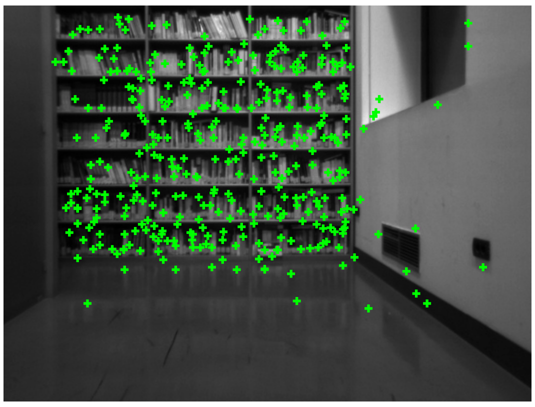
5.2 Map Initialization

The initial map is constructed from two images aquired with a calibrated stereo rig. In both images SURF features are extracted and matched. Since the relative orientation between both cameras is known, the 3D position of the corrsponding image points can be calculated as described in Section 3.3.1. The triangulated 3D points are stored along with the SURF descriptors.

The two oriented cameras define the world coordinate system $\mathcal{C}$. The first view is the origin of $\mathcal{C}$, whereas the x and y coordinates of the map are parallel to the image plane. The z coordinate indicates the depth of the scene. The orientation of the second view is given with respect to the origin. Afterwards, both views are stored in a single keyframe.



(a)



(b)

Figure 5.2: Map Initialization. In (a), the left camera represents the origin of the world coordinate frame $\mathcal{C}$, whereas the right camera pose is given with respect to the origin. (b) shows the image taken frame the left camera with extracted SURF features.

5.3 Iterative Map Building

The initial map is sufficient for localization while the robot navigates in a very limited area. The map has to be extended whenever the robot leaves the mapped area. In this section, we explain the criteria for map extension, the mechanism of map extension itself, and the map optimization using bundle adjustment.

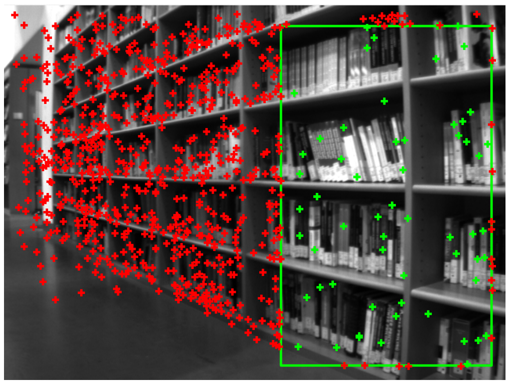


Figure 5.3: Map extension. The green marked recetangle illustrates the region that includes all rediscovered map points during localization. If this area, compared to the image size, is below a threshold and there are enough extracted keypoints (red), this frame is used to extend the map.

The workflow of the SLAM algorithm is outlined in Figure 5.1. The robot grabs new images $P_{t,n}$ from its cameras and starts the localization process followed by a check wether the robot moves to a not yet mapped area. In that case, the map is extended. Otherwise the next image for localization is acquired.

Map Extension

The map extension is triggered depending on the number of map points M_j recognized at the localization phase. Accordingly, depending on the number and distribution of map points rediscovered in the current image P_t, two states of the robot can be distinguished:

- Exploration
- Navigation

The state is determined as follows: The convex hull in image space of the reprojected map points is calculated. If the area of the convex hull is below a certain threshold (e.g. 40% of the image area), the number of salient image points outside the covered area is counted. If there are enough keypoints, this image and its corresponding stereo image is used to extend the map and stored as a new keyframe. Figure 5.3 shows a picture where the convex hull of rediscovered map points covers 38% of the image and about 700 extracted keypoints are outside the hull. Hence, this image is used for map extension. If the robot navigates in an already mapped area, new keyframes and map points are created only if the robot has moved a certain distance.

A map update comprises two steps: storing of a new keyframe as well as the triangulation of new map points, which is performed as follows: All existing map points are reprojected to the current view P_t and matched using a patch-based descriptor (correspondences identified in the localization phase are re-used here). All keypoints that are already represented by a map point and keypoints in their $n \times n$ neighborhood are not used for creating new map points. This prevents clusters of map points and contributes to a roughly uniform distribution of map points in the image. The remaining keypoints are used to find corresponding points in the stereo image. Since the corresponding point in the second image cannot be predicted precisely, we spend the SURF feature descriptor. Afterwards, the 3D positions of the corresponding image points are calculated, transformed to the global coordinate frame $\mathcal{C}$, and stored in the map. Figure 5.4 illustrates the steps of the map point creation process.

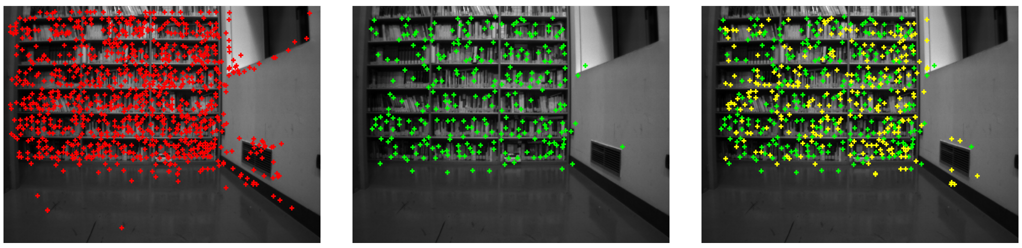

(a) (b) (c)

Figure 5.4: Map Point Creation. (a) shows the current camera image P_t with extracted Harris keypoints. The green crosses in (b) mark all image locations that are already represented by map points. (c) shows old measurements (green) and new image locations (yellow) that are used for new map points. For each yellow marked image location, a corresponding image patch in the stereo frame could be identified.

Map Optimization

The accuracy of the reconstructed 3D points depends on the accuracy of the estimated camera poses. As described in Chapter 3.3.2, the accuracy of the triangulation as well as the pose estimation can by increased by bundle adjustment, if more than two image points of the same 3D point are available.

Initially, a map point is triangulated from two corresponding image points. In order to obtain more measurements, all subsequent keyframes are checked for corresponding image locations for a certain map point. If a measurement could be identified, the location within the keyframe is stored in the map point. This represents the connectivity matrix as shown in Figure 3.6.

Since each map point consists of a SURF feature descriptor and an image patch, two different methods can be used for identification of new measurements for existing map points. The advantage of SURF features is their reliability, whereas the number of recognized map points is relativly low compared to the patch-based approach. Referring to Chapter 4.4, the reliability of the patch-based method strongly depends on the accuracy of the estimated pose, the image was taken from. To decouple the map optimization step from the pose tracking, we decided to combine the SURF detector and the patch-based method as follows.

Every time a new keyframe is stored, we search for corresponding points in this image using SURF features as well as patch-based correlation. We assume that SURF matches are more reliable and therefore, we compare the median SURF reprojection error to the median reprojection error, calculated using patch-based correspondence detection. If both reprojection errors are in the same range, the quality of corresponding image points found by patch-based correlation is acceptable and they are used for optimization. Otherwise, we apply the results of the SURF detector.

Since optimizing the complete map is computational demanding, we decided to optimize only a subset of keyframes and map points. This set contains the last n keyframes and all recognized map points. Furthermore, all keyframes that have measurements of the recognized map points are also added to the optimization set. It turned out that $n = 10$ is sufficient for an accurate map.

This subset of the map is then optimized by robust bundle adjustment as described in Section 3.3.3 using the Tukey M-Estimator error function. We parametrized the keyframe pose by three Euler angles and the 3×1 translation vector. The map point is represented by its three Euclidean coordinates.

5.4 Loop Closing

Because map building and optimization is performed iteratively, small errors in pose estimation and reconstruction sum up and may cause large absolute errors. For example, if the tracker makes a small rotational error and the robot moves several meters in this direction, the resulting error gets large. Figure 5.6 shows the resulting map of a 70 m loop, which contains a small displacement in X-direction. The only chance to detect and correct this error is re-visiting an already explored region after an arbitrarily long excursion.

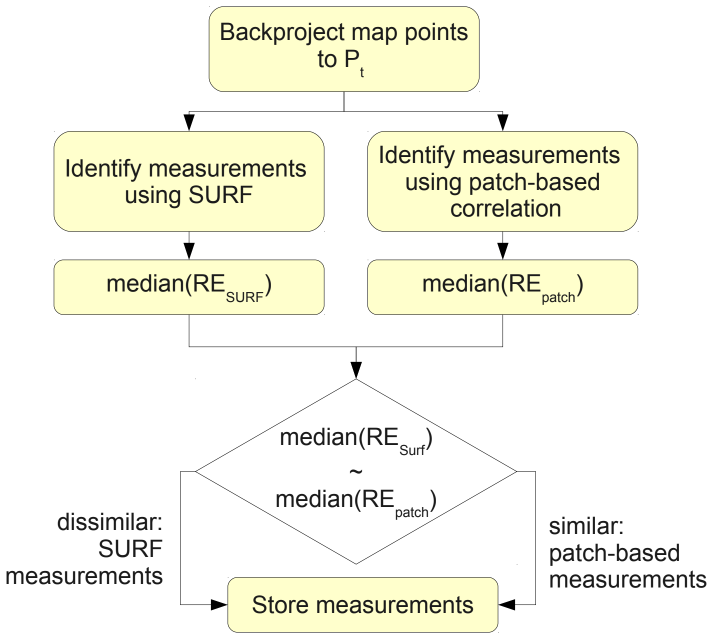

Backproject map points to P_t
Identify measurements using SURF
Identify measurements using patch-based correlation
median(RE_{SURF})
median(RE_{patch})
median(RE_{Surf}) ~ median(RE_{patch})
dissimilar: SURF measurements
similar: patch-based measurements
Store measurements

Figure 5.5: Quality check patch-based correlation. Assuming SURF features are more reliable than patch-based correlation, we compare the median reprojection error of the identified correspondences. If both are in the same range, we use the patch-based correspondences, otherwise the points detected by the SURF features.

This is known as 'loop closing task' and can be split in two parts:

- Identification of an already visited place
- Global map and trajectory correction using bundle adjustment

Because the absolute error made in map building can be arbitrary large, the pose of the robot cannot constrain the search area for a loop detection and therefore loop detection is similar to the global localization task.

Below, we discuss loop detection and explain the loop rectification method.

5.4.1 Loop Detection

In general, loop detection is similar to global localization. In order to recognize a loop, the current image is compared to all keyframes stored in the map. The decision of a loop detection is based on the structure of the most similar images.

In order to detect a loop closure, the robot inquires the vocabulary tree of keyframes periodically (e.g. every 10^{th} captured image) to identify the n most similar images as

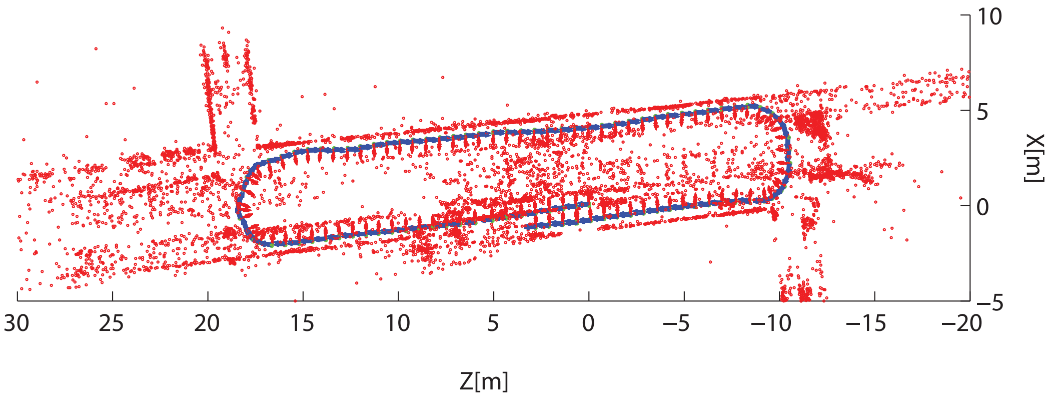


Figure 5.6: Uncorrected Loop. The resulting map of a 70 m trajectory shows a small displacement in X-direction, although the robot returns to the same location.

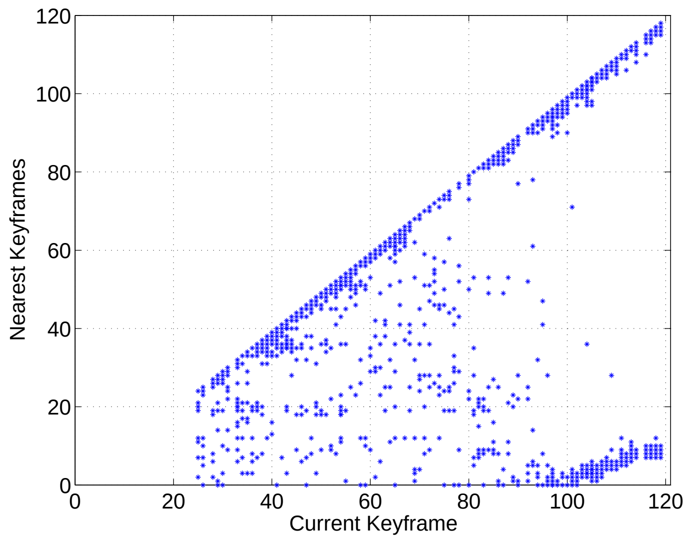


Figure 5.7: Typical structure of the nearest neighbors in a loop. Up to keyframe 95, the nearest neighbors are located on the diagonal, which indicates that the current keyframe is similar to quite recently stored keyframes. The off-diagonal points are outliers. Starting from keyframe 95, the structure changes in two parallel lines, which indicates a loop closure.

described in the global localization (Section 4.3). Dependending on the robot's trajectory, two categories of the result can be identified. If the robot has not yet finished a loop, the most similar images are typically keyframes that have been added to the map quite recently. In contrast, if the robot finished a loop, the result contains recently added keyframes as well as keyframes stored a long time before. Due to quantization effects of the vocabulary tree, the result may be polluted by outlier images.

Loop detection therefore consits of two parts: An outlier detection and a classification

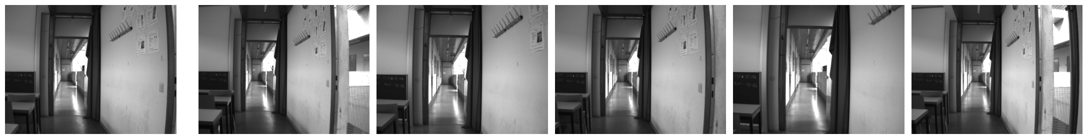


Figure 5.8: Loop Detection. The first image is the query image. The other five images are the nearest neighbors (NN) in ascending order. The numbers below the images denote the corresponding keyframe number. The query image is keyframe 102 and the first NN in image space is the keyframe quite recently stored. All other NN are stored a long time before.

part. Outlier keyframes are removed by comparing each keyframe to the current frame using SURF features. If there are only few corresponding points, the keyframe is removed from the result. Next, the result is classified into two sets: recently added keyframes and old keyframes (K_B). Some heuristics on the old keyframe set are performed to reduce the probability of an erroneous loop detection, e.g. K_B contains more than a certain number of frames and these frames are located nearby. If all heuristics are fullfilled, the current frame is stored as a new keyframe K_E and added to the map.

Figure 5.7 shows the typical nearest neighbor (NN) structure of a loop. The x-axis denotes the most recent keyframe (query image) and the y-axis the number of the resulting keyframes stored in the vocabulary tree. A point in a certain column means that the corresponding keyframe is one of the nearest neighbors of the current keyframe. The points on the diagonal show that images in spatial neighborhood are also related in image space. Up to keyframe 95, a part of the nearest neighbors are randomly distributed over all keyframes. These outliers are filtered out as described above. Starting with keyframe 100, a more regular structure of the nearest neighbors gets visible: The query result is split in two parallel lines. This indicates that the current keyframe is similar to recently stored keyframes as well as old keyframes, which indicates a loop. Figure 5.8 shows the nearest neighbors of an example keyframe.

5.4.2 Loop Correction

Before the trajectory can be rectified, duplicate 3D points have to be removed and the last and the first frame of the loop have to be connected by common map points.

The first step is to merge duplicate map points. A duplicate map point is represented by two distinct map points which correspond to the same world point. This is possible due to keyframes at the end of the loop that have a relatively large absolute localization error

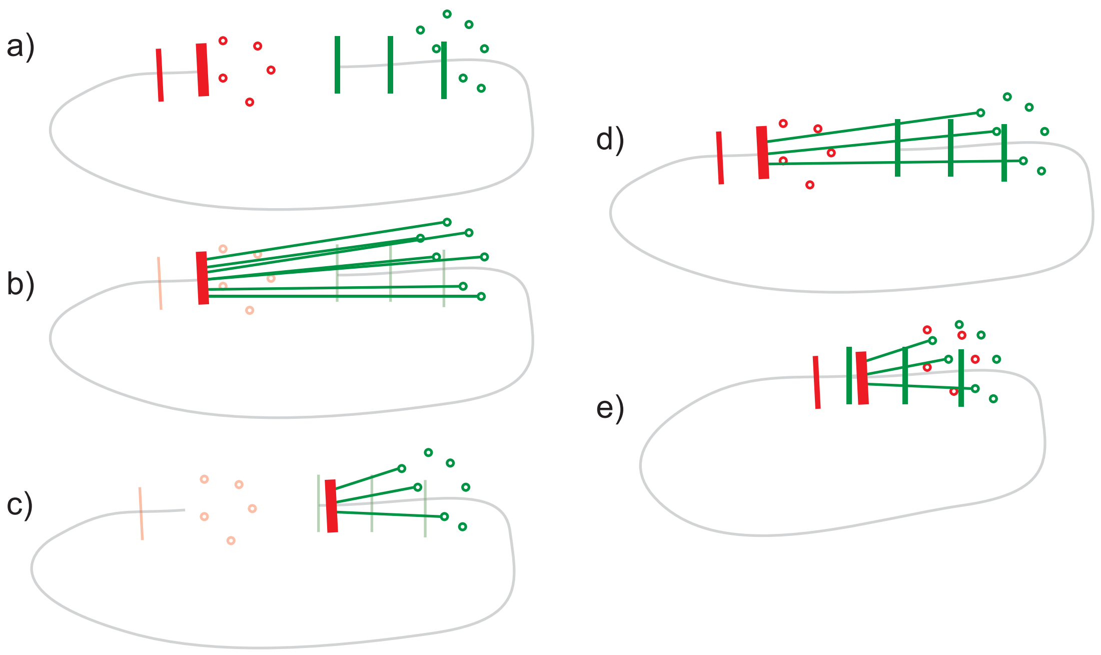


Figure 5.9: a) Situation before Loop Closing. Green map points are visible in keyframes at the beginning of the loop and red points are visible only at keyframes at the end. b) Green map points are identified in the last keyframe using SURF. c) The pose of last keyframe is optimized using only the green map points. This identifies outlier measurements. d) The last keyframe is reset to the old pose. e) Least squares bundle adjustment is performed on all map points and measurements.

and therefore do not match to map points generated in the first keyframes of the loop. Hence, the map contains two map points representing the same 3D point.

To remove duplicates, the SURF feature descriptors of all map points created in last i keyframes are compared to all generated map points at the beginning of the loop. If two points M_l and M_f are matched, all measurements of M_l are copied to the measurements list of M_f, and M_l is deleted from the map.

The loop rectification is performed by the bundle adjustment algorithm. Due to robust iterative map optimization, nearly all map points have a small reprojection error except map points that are visible both at the end and at the beginning of the loop. To achieve fast convergence, as many correspondences as possible between the first and the last keyframe are identified using the SURF features.

Now, the loop can be rectified by bundle adjustment. In contrast to iterative map optimization, we are not using a robust error function but the standard least squares function. Based on the fact that erroneous measurements are removed by the iterative optimization performed after each map update, we can assume that the measurements inside the loop are free of outliers. Outlier measurements between the first and the last keyframe of the loop are identified by robustly localizing the last keyframe with respect to the first keyframe of the loop. Removing these outlieres, we obtain a set of measurements that is free of outliers and we can use the squared error function as an objectiv function. Figure 5.10 shows the result of a loop that has been optimized by bundle adjustment.

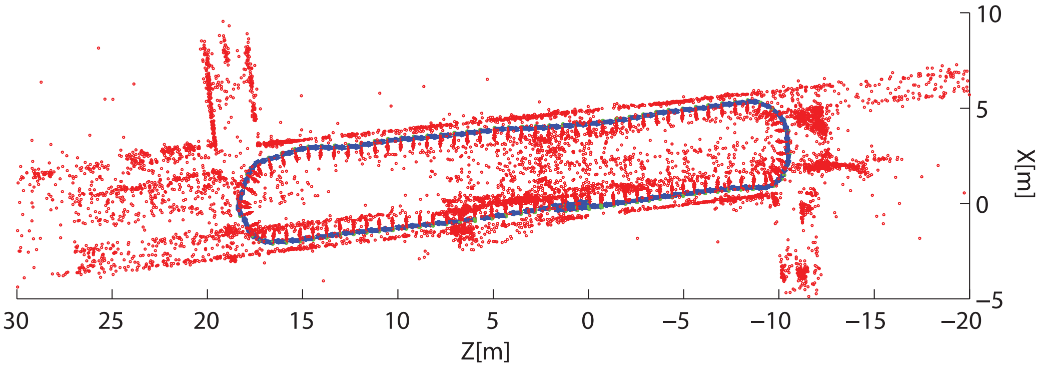


Figure 5.10: Corrected Loop. The loop of Figure 5.6 corrected using bundle adjustment.

5.5 Sensor Fusion

In the previous sections, we explained a visual SLAM algorithm based solely on a stereo camera rig. In real applications, a robot is equipped with additional sensors like odometers or inertial measurement units, which can be also used for robot localization. In this chapter, we present an idea of fusing different sensor information using so-called *synthetic visual features*. This approach enables us to use visual optimization methods for fusing non-visual sensor information. Furthermore, we can use the idea of bundle adjustment for loop optimization although visual features are missing.

We assume a robot that is equipped with a stereo camera rig and another sensor that delivers information about the robot's movement, e.g. an odometer or an IMU. Without loss of generality, we elaborate our approach using the stereo camera and the odometry information of the robot.

Given an initial pose P_0, the pose of the stereo camera rig in the next time step P_1 can be estimated in two ways:

- Visual SLAM

- Shifting the camera using odometry information

To combine information from both sensors, we generate so-called synthetic map points at time $t = 0$. Instead of a real map point, the 3D pose of a synthetic map point is not triangulated using two corresponding image points, but their position is chosen randomly in front of both images. The map point is reprojected to both images and these positions are stored as measurements. The camera pose at time $t = 1$ is estimated using odometry as well as visual odometry as described in Chapter 4. This results in two poses, namely $P_{1,C}$ and $P_{1,O}$. A new measurement for a synthetic map point M_s at $t = 1$ is generated by reprojecting M_s to the camera image $P_{1,O}$. The reprojected image position is then stored as a measurement of M_s in $P_{1,C}$. So, $P_{1,C}$ contains measurements of real map points and synthetic map points. $P_{1,O}$ is no longer needed and therefore not stored in the map. To calculate the fused pose P_1, $P_{1,C}$ is optimized using bundle adjustment.

An important question not discussed in this thesis is how to model the uncertainty of the sensor. The number of synthetic map points and their position with respect to the camera obviously influence on the optimization result. Hence, an approach for modeling the uncertainty could be to vary the number of synthetic map points or to correlate the position of the synthetic map points with the uncertainty of the odometry.

Synthetic map points can also be used to solve another problem. Loop closing requires each keyframe to be connected to other keyframes by sharing common map points. This is a problem for keyframes, that are less textured and contain only projections of a few map points. Normally, they produce a gap in the loop and prevent optimizing the loop using bundle adjustment. To overcome this problem, we use synthetic features. If few map points are visible in the current frame, the pose P_1 is calculated by shifting the camera pose by odometry. If the camera has moved a certain distance, a new keyframe is stored in the map. In order to connect this keyframe to previous ones, we create synthetic map points. A random position is assigned to each synthetic map point, so that the reprojected position lies within the last two keyframes. Since a map point needs to be visible at least in two cameras, we reproject this point to other, previously generated keyframes and store the reprojected position as a new measurement for this map point.

5.6 Discussion

Each map point is constructed from a pair of corresponding image points x, x' in the images P, P'. Typically, x and x' are image locations identified by a salient image point detector such as DOG keypoint detector (SIFT) or Harris Corners (SURF). This implies that image locations, i.e. areas of the environment, with less texture are not represented by map points. This leads to problems in less structured environments,such as corridors with non-textured walls. A similar problem is motion blur in images. DOG keypoints and Harris corners extract image locations with fast changes in x- and y-direction. In a blured image, most of the corners disappear and therefore no map points are generated. Hence, the number of potential map points depends on the number of extracted salient image features.

In addition to the number of salient image points, the discriminative property of the descriptor is responsible for the quantity and the quality of new map points. A strong descriptor like SIFT or SURF produces few false matches and therefore achieves a high reconstruction accuracy but the number of correspondences is relatively low. On the other hand, a patch-based matcher finds a lot of corresponding points but also a high number of false positives. When finding further correspondences for exiting map points, we combined both methods to get more accurate matches. This makes the map building process more independent of the local localization part. Especially scenes containing abrupt movement changes are often handled incorrectly by the tracker. Figure 5.11 shows two maps of the same environment where the 5.11(a) is constructed by using measurements identified by patch-based correlation and 5.11(b) is optimized by measurements extracted using SURF features.

We decided to apply the SURF matcher because a strong feature descriptor increases the accuracy of the trajectory and makes the map building process somewhat independent from the local localization.

Loop closing consists of two subtasks: loop identification and map optimization. The detection of a loop is similar to the global localization task and therefore has similar problems, e.g repetitive structure of the surrounding. This is a serious problem in the following case: The robot moves along a corridor and turns into another corridor with similar appearance. The query result of the vocabulary tree therefore consists of recently added and older keyframes, which leads to a loop closing detection. It is very difficult to distinguish between a real loop and an erroneously detected loop. This problem could be solved by a particle filter, which tracks several hypotheses.

Some other issues arise directly from the bundle adjustment algorithm. Even highly optimized implementations are not able to solve an equation system in realtime that consists of several thousand equations. So, loop closing is a very expensive task and can take

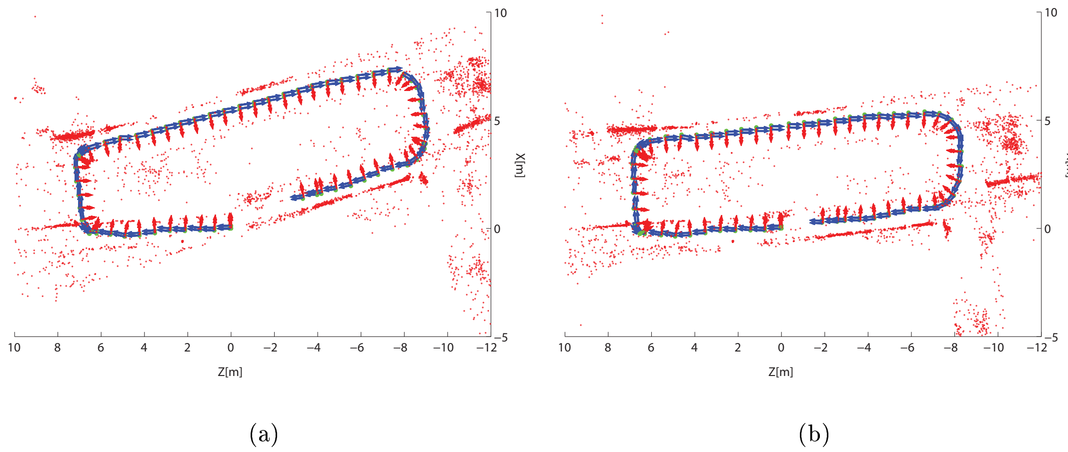


Figure 5.11: Comparison of Patch-based vs. SURF-based map building. Both maps are the results of the same image sequence. In 5.11(a) we used the positions obtained by the local localization process without checking them by SURF features. The positions in the right map was corrected using SURF features as shown in Figure 5.5. The right map is closer to the real trajectory.

up to several hours for large trajectories.

In this thesis, we present an approach for sensor fusion in image space. We use synthetic image features in case real image features are not available. Instead of simply shifting the camera by odometry, this allows us to perform loop closing, even with frames that contain few image features.With this idea different sensor information can be fused in a single optimization step. However, for taking the uncertainty of the sensors into account, a deeper mathematical inspection is required.

Chapter 6

Experiments

Contents

The goal of this thesis is to develop a SLAM approach that maps large environments. We evaluated our approach regarding robustness, accuracy, and scalability on two different datasets. The *Rawseeds* dataset [4] is a well-known benchmark and contains images of an indoor and outdoor exploring robot. We decided to evaluate our approach on an indoor trajectory of 774 m. The second dataset is a self-recorded indoor image sequence and contains a loop of about 69 m length It is explored 4 times by the robot. The dataset is recorded at the *Institute of Computer Vision and Graphics* (ICG) at the Technical University of Graz. To show the accuracy of our approach, we compare our results to groundtruth data that is provided by the datasets.

Our experiments are organized as follows: Since our approach is based on PTAM [19], we evaluate each enhancement on both datasets separately. In our first experiment, we evaluate the PTAM approach that is extended by stereo camera support. This demonstrates that PTAM is basically able to handle large scale environments. In the second experiment, we replaced the original patch-based correlation feature detector by SURF features for map building as described in Chapter 5.3. This experiment shows the increased robustness against fast camera movements. In the following trial, we add sensor fusion support to our software. This enables us to perform loop closing even if keyframes are not connected by map points. We show that loop closing increases accuracy and reduces the number of map points. Finally, we compare our method to results of other visual reconstruction methods to demonstrate the accuracy of our method.

6.1 Evaluation Metrics

In order to show the accuracy and scalability of our approache, we define several metrics. Because groundtruth is given in 2D, we fit a plane to the camera centers and project them on this plane. As an accuracy measure we use the *Absolute Pose Error* (ATE) between the trajectory achieved by our approach and the provided groundtruth (GT). ATE measures the Euclidean distance between the groundtruth position at time t_j to the estimated position at the same time. Since our trajectory is sampled only by the keyframe poses, which is less frequent than groundtruth, we interpolate our trajectory such that a corresponding pose estimate is available for each groundtruth value. It is also necessary to align both trajectories, because of different coordinate systems. Rawseeds proposes to align both trajectories by minimizing the ATE in a least squares sense and the accuracy of the reconstructed trajectory is then given by the mean of the remaining ATE errors. A further accuracy measure is the length of the reconstructed trajectory compared to the length of the groundtruth trajectory.

Since groundtruth is only provided for the Rawseeds dataset, we compare the trajectory of the ICG sequence to a map constructed using a laser scanner that is also mounted on the robot. The map and trajectory are recovered by the GMapping algorithm proposed by Stachniss et al [35]. The implementation is provided by the openSLAM[*] project.

The precision of the reconstructed map is presented by overlaying the resulting visual feature map with a 2D floorplan. The scalability of the system is shown by the number of visual feature points over time.

[*]http://www.openSLAM.org

6.2 Datasets

Rawseeds Dataset

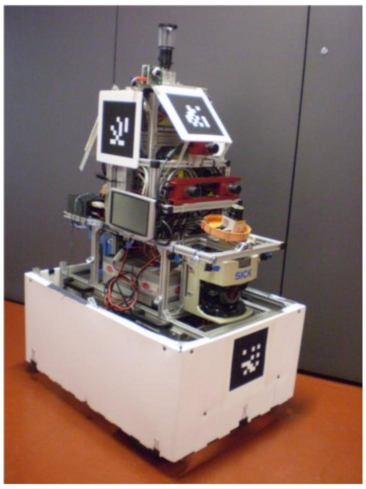


Figure 6.1: Robot used to record the Rawseed datasets.

The goal of the Rawseeds Project is to generate benchmark datasets for SLAM algorithms. Rawseeds provides sequences for static, dynamic, indoor, outdoor, natural, and artificial lighting environments. All datasets are taken by a robot, which is equipped with several sensors like laser scanners, cameras, inertial measurement unit (IMU), an odometer, and GPS. The robot used for data acquisition is shown in Figure 6.1. Additional to the sensor readings, all datasets come with groundtruth for the robots trajectory and a 2D floor plan of the environment.

In our experiments, we use two calibrated gray-scale cameras mounted at the front of the robot and looking in direction of motion. In the last experiment, we additionally use odometry information whenever visual SLAM is not possible due to absence of image features. Along with the image sequences, the intrinsic and extrinsic camera calibration parameters as well as the relative orientation between the robot coordinate frame and the cameras are given. The baseline between of stereo setup is approximately 18 cm.

For evaluation, we selected the *Bicocca 2009-02-25b* dataset. It is a static indoor sequence with artificial lighting. We decided to use this sequence because the outdoor sequences contain a large number of frames over or underexposed. Furthermore, the sequences were taken from a relatively fast moving robot and so, especially at cornerings, motion blur occurs. The dataset comprises a sequence of 26,000 images per camera taken with a framerate of 15 Hz. The image size is 640x480 pixel and the robot moved with an

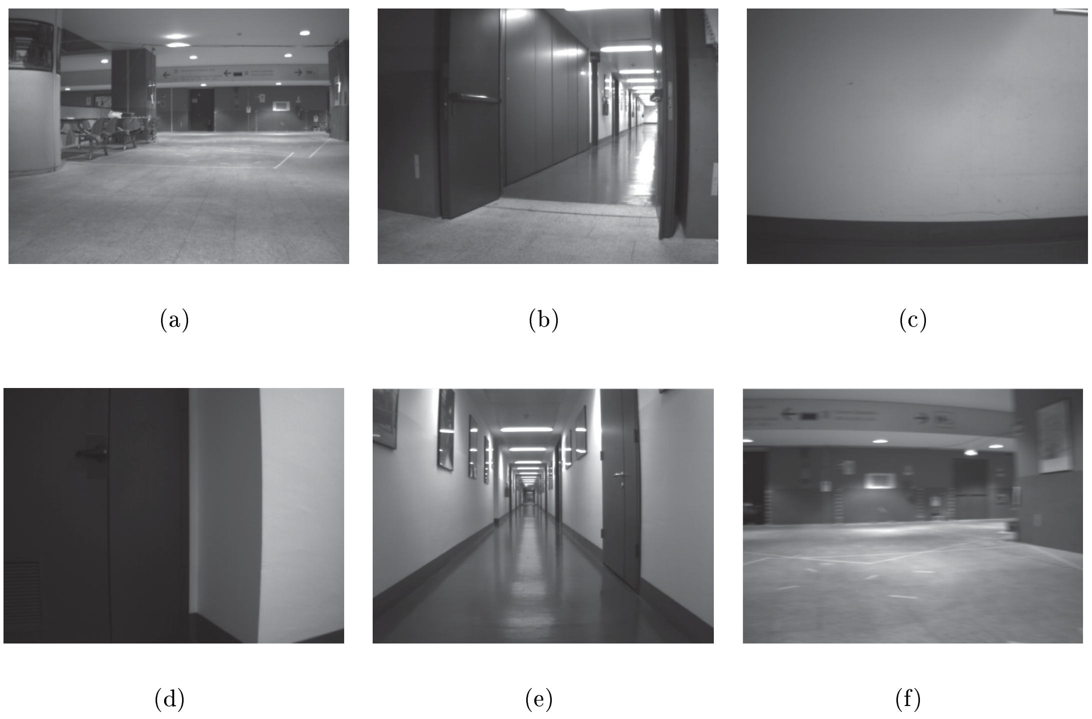


Figure 6.2: Six sample images of the Rawseeds Bicocca 2009-02-25b dataset, which has a length of 774 m. The robot starts at a wide hallway (a) and turns to a long corridor (b). Images (c) and (d) show the turning point at the end of a corridor. In Image (e), the robot is on the way back to the starting hallway. An image that is blurred because of fast motion in the hallway is shown in (f).

average speed of 0.5 m/s. The environment is a university building, that consists of long, weakly textured corridors, wide and small hallways as well as a large library. The terrain is relatively smooth with small ridges between the different parts of the building. Figure 6.3 illustrates the floor plan and the trajectory of the moving robot of the dataset.

A selection of images taken at different points of the robots trajectory is shown in Figure 6.2. The robot starts at a wide hallway (Figure 6.2 (a)) and turns then to a narrow, weakly textured corridor (Figure 6.2 (b)). At the end of the corridor (Figure 6.2 (c) and (d)), visual odometry is not possible. Figure 6.2 (e) shows that most of the image features in the corridor are far away from the robot, which is a problem for an accurate 3D reconstruction. At the end of the trajectory, the robot turns quickly, which causes motion blur (Figure 6.2 (f)).

Due to computational reasons and the similarity of most parts of the trajectory, we decided to use only a small part of the whole trajectory for evaluation. This part starts in a

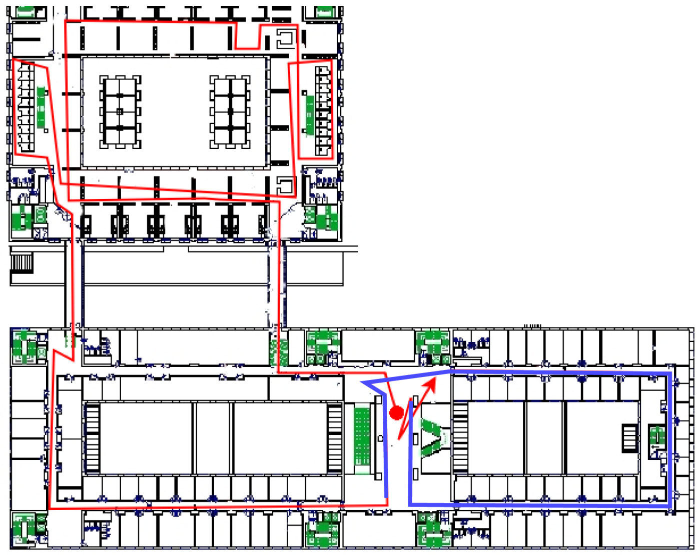


Figure 6.3: Floorplan of Rawseeds dataset Bicocca 2009-02-25b. The red and the blue line illustrate the approximate trajectory of the robot. The blue line is the part we evaluated in our experiments.

wide hallway, directs to a loop of corridors and ends in the starting hallway. It has a length of approx. 190 m. Furthermore, the dataset contains some parts, we left out because they are difficult to handle for visual SLAM. The sequence is recorded at night under artifical ligthing conditions and a part of the trajectory directs to a tunnel of glass. Here, a lot of objects mirror in the glass and move simultaneously with the robot. However, this violates our assumption of a static environment. The estimated time for calculating the whole trajectory is about one hour and the consumed is about 4 GiB.

ICG Dataset

The ICG dataset is a self-recorded indoor sequence taken from a robot at the Institute of Computer Vision and Graphics (ICG) at the Technical University of Graz. The robot is endowed with a stereo camera rig, a SICK laser range scanner and an odometer. The sequence is static and consists of a 69 m loop that has been traversed four times. We recorded 4600 images of 640x480 pixel at a framerate of 7.5 Hz. The baseline of the stereo rig is 12 cm and the robots speed is 0.5 m/s. The intrinsic and extrinsic camera calibration is performed by the method proposed by Zhang [39].

The challenge of this dataset are tight curves, which cause small overlap between the images. Some parts of the sequence consist of fast, non-smooth camera movements. Figure

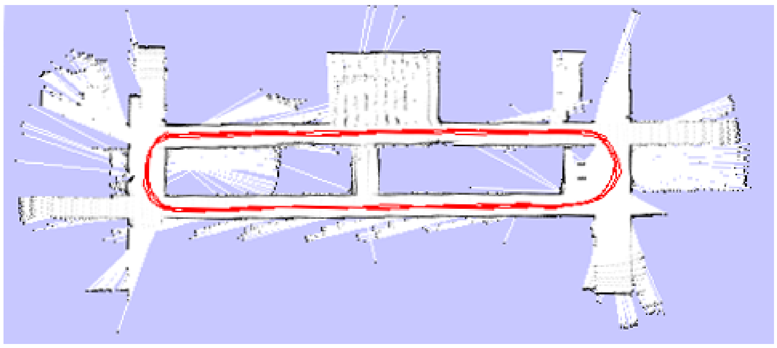


Figure 6.4: Groundtruth of the ICG dataset generated by GMapping that uses laserscanner data.

6.4 shows the floor plan of this dataset. The red line is roughly the trajectory of the robot. It is driven four times. Sample images are shown in Figure 6.5.

As reference, we used the trajectory created by the GMapping [13], which generates a map using laser scanner data and is an extension of the FastSLAM method [14]. Figure 6.4 shows the resulting map and trajectory.

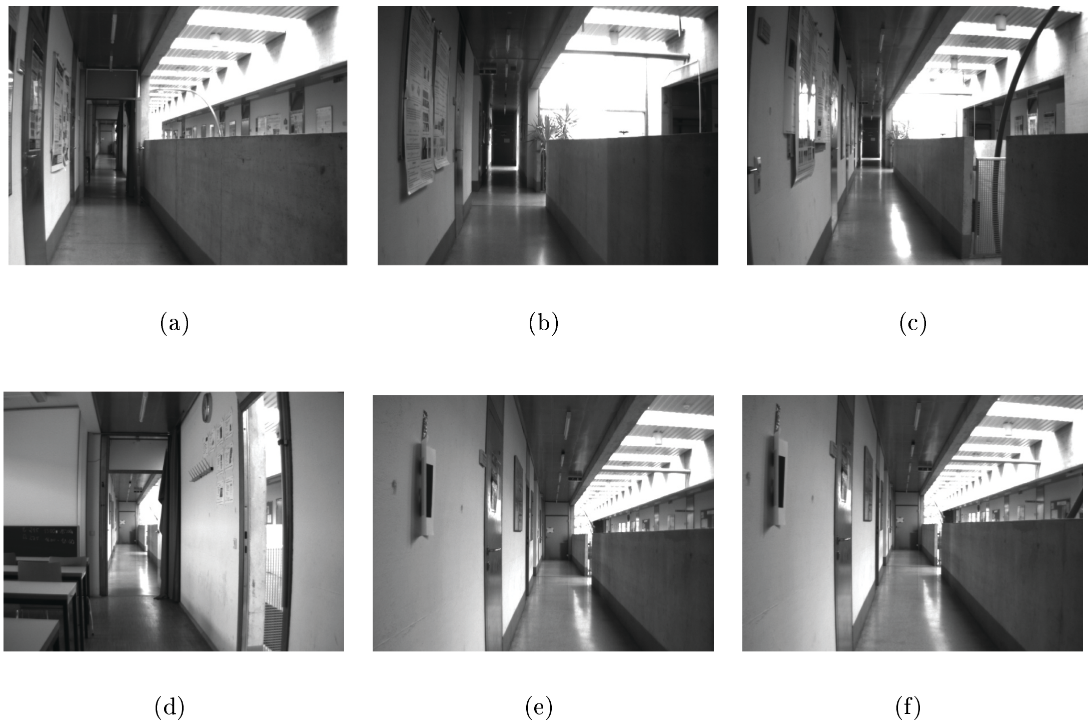


Figure 6.5: ICG sample images. The largest part of images shows long but well-textured corridors. A short part of the trajectory (d) directs to a small classroom.

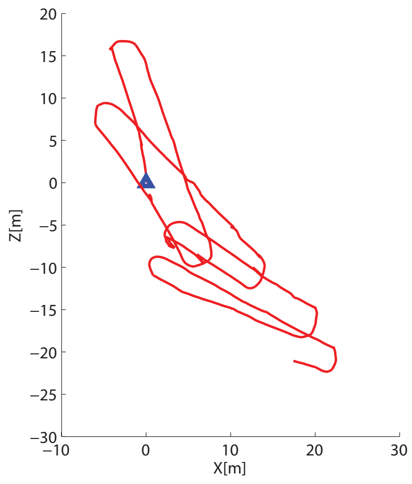
6.3 Evaluation of Stereo PTAM

In the first test, we used the source code provided by Klein et al. [19] and modified several parts to get PTAM working on both datasets. Since our goal is to develop an accurate metric SLAM algorithm, we added stereo camera support to PTAM as described in Section 5.3. To demonstrate the weakness of the patch-based correlation, we do not use SURF features in this experiment. In case of a stereo camera, a keyframe exists of two images and a new map point is triangulated from corresponding points of the stereo image pair. Point correspondences are identified by patch-based correlation using the epipolar constraint. Furthermore, we changed the multithreaded implementation of PTAM to singlethreaded, which guarantees reproducible results and assures that bundle adjustment is performed after each map extension. Finally, we changed the map extension strategy as described in Section 5.3. Otherwise, the map update rates are too slow and visual odometry is not possible. The distance between two keyframes is about 0.40 m, if the robot moves on a straight line.

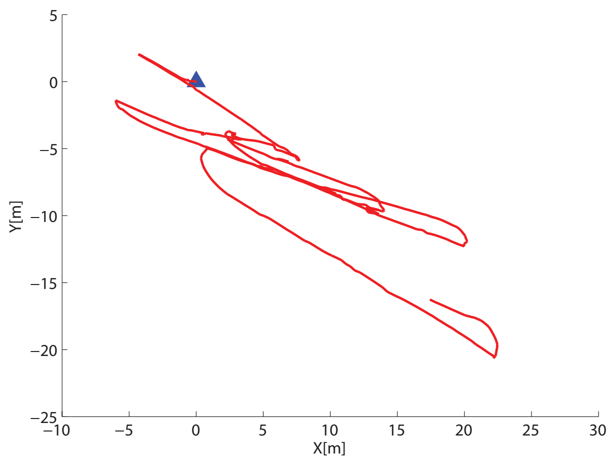
On the ICG dataset, the modified PTAM version leads to poor results. The estimated trajectory (Figure 6.6) differs fundamentally from the trajectory generated by GMapping (Figure 6.4).

The reason is seen after 15 m: Although the robot moves forward, the estimated camera pose rests at the same position. Not before reaching the end of the first half of the loop, the robot adds a new keyframe, and localization and map building continues. This situation shows a disadvantage of PTAM, which follows from the patch-based correlation approach for recognizing 3D points. To relate the current frame to existing map points, all map points are reprojected to the assumed pose and are identified in a small neighborhood by comparing image patches. A false pose estimation before reprojection so results in wrong map point measurements. The pose estimation in the next timestep is affected by the erroneous map point association and therefore the calculated pose is wrong as well. This shows that a single erroneous pose estimate may corrupt the whole trajectory. Furthermore, keyframes are only added if the current cameras pose has at least a distance of 0.40 m from an existing keyframe. If the local localization detects spuriously no movement no keyframes are added, which results in the effect seen after 15 m.

The resulting trajectory is far from being planar (Figure 6.6). This is why we omit an evaluation of the accuracy. Accordingly, the constructed map (Figure 6.7) is distorted and a comparison with groundtruth is useless. The whole map consists of 88,092 point features and 1,042 keyframes.



(a)



(b)

Figure 6.6: Performance of the modified PTAM on the ICG dataset. The blue triangle marks the start of the trajectory. Plot (a) shows a top view of the trajectory (XZ-plane). The sideview in Plot (b) illustrates that the estimated motion is far from beeing planar.

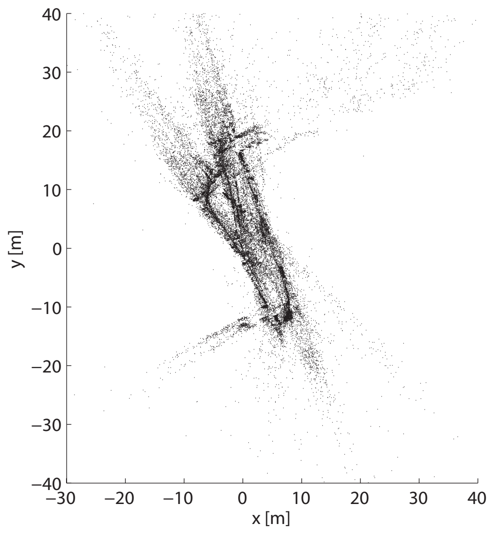


Figure 6.7: Resulting map of the ICG dataset generated by PTAM after 200 keyframes. It consists of 32,000 map points. Due to erroneous pose estimation the map is distorted.

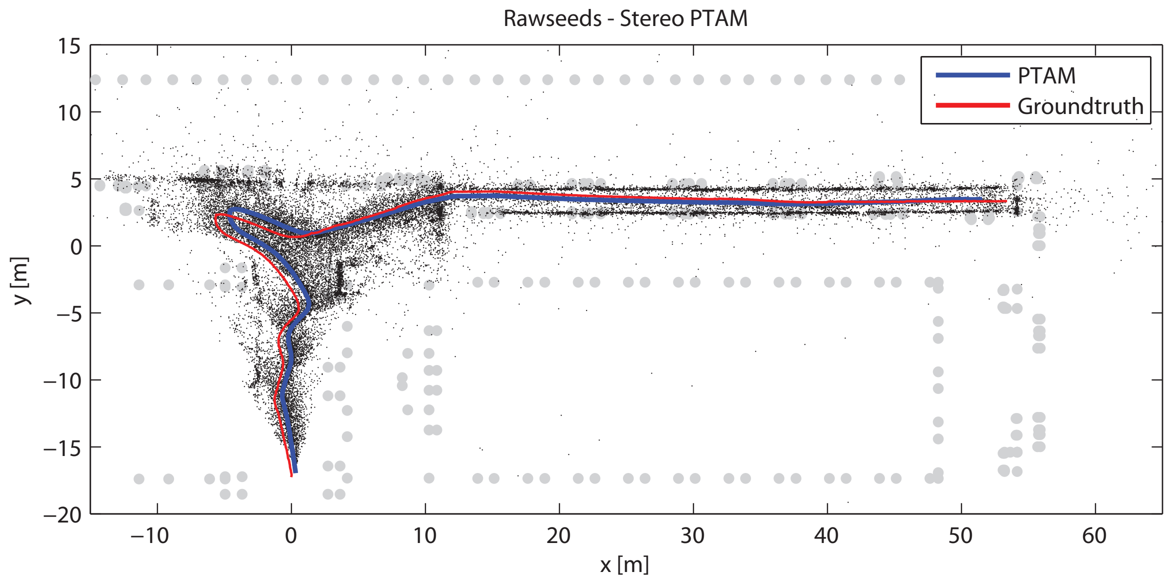


Figure 6.8: Trajectory generated by PTAM on the Rawseeds dataset. The light gray bold dots indicate walls. Each black dot is a map point projected on the fitted plane. The trajectory estimated by PTAM is similar to the groundtrouth and the reconstructed map points are consistent with the floor plan.

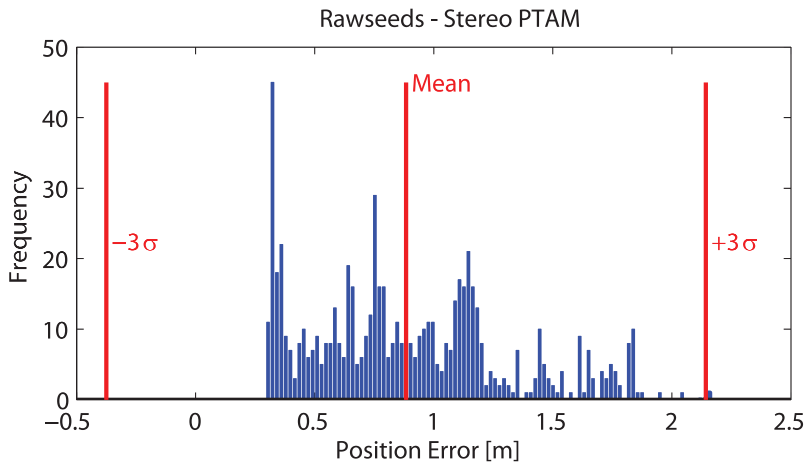


Figure 6.9: Error distribution of Stereo PTAM on the Rawseeds scene. The mean error is 0.88 m and the standard deviation 0.42 m. All errors lie within a 3 σ range.

As shown in Figure 6.8, PTAM performs much better on the Rawseeds dataset, because of a very smooth robot motion. In contrast to the ICG dataset, the framerate is doubled and therefore the distance between two frames is bisected and thus, the mean traveled distance between two frames in the Rawseeds scene is 0.03 m. As seen in Figure 6.2 (c), at the end of the first corridor, the robot faces a white wall that contains no visual image features. For that reason, visual odometry is not possible and mapping is stopped. The mean error between groundtruth and the trajectory estimated by PTAM is 0.884 m with a standard deviation of 0.42 m (see Figure 6.9). The error histogram shows a systematic error of 0.40 m, which is caused by the definition of the ATE error. Groundtruth and estimated trajectory are aligned by a least-squares minimization. So, the histogram indicates a small scale-drift of 0.40 m of the estimated trajectory.

Summing up, PTAM can be used for map building when stereo cameras are used and the entire trajectory is texture-rich. The ICG dataset shows the limitations of the original PTAM implementation. Only smooth trajectories with high framerate can be processed accurately. So, the most important task for real SLAM applications is to make the approach more robust against fast and rough motion. Since the weakness is induced by the patch-based correlation for feature matching we partly replaced it by the SURF descriptor as described in Chapter 5 in the next experiment.

6.4 Evaluation of PTAM using SURF Features

In this experiment, we extend PTAM by using SURF features as a point correspondence detector as described in Chapter 5. In order to show repeatability of the SURF PTAM, we modified the implementation in the local localization: Instead of associating all map points to the current frame, we only use the map points generated in the last 20 keyframes. This has the effect that map points generated during the last visit are not used for local localization. Hence, each cycle is treated independently and processing time of local localization is independent of map size.

As demonstrated on the ICG dataset, Stereo PTAM is not able to deal with sequences that contain fast camera movements. Figure 6.11 illustrates the projected 2D trajectory estimated by our approach. The squared mean distance between all camera centers and the fitted plane, is 0.041 m, which indicates planar motion. By visual inspection, one can see that the estimated trajectory is very similar to the GMapping result. This is also expressed by the ATE error of 0.23 m and a standard deviation of 0.14 m.

Figure 6.12 is the projected map generated by our approach overlaid by the gridmap

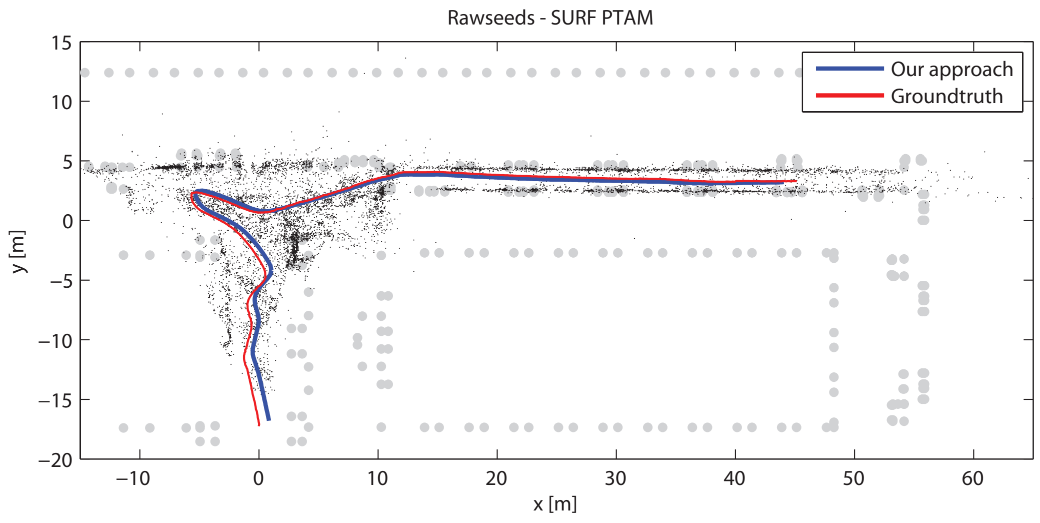


Figure 6.10: Rawseeds Dataset processed by SURF PTAM. The trajectory is very similar to groundtruth and the map is less dense compared to Stereo PTAM.

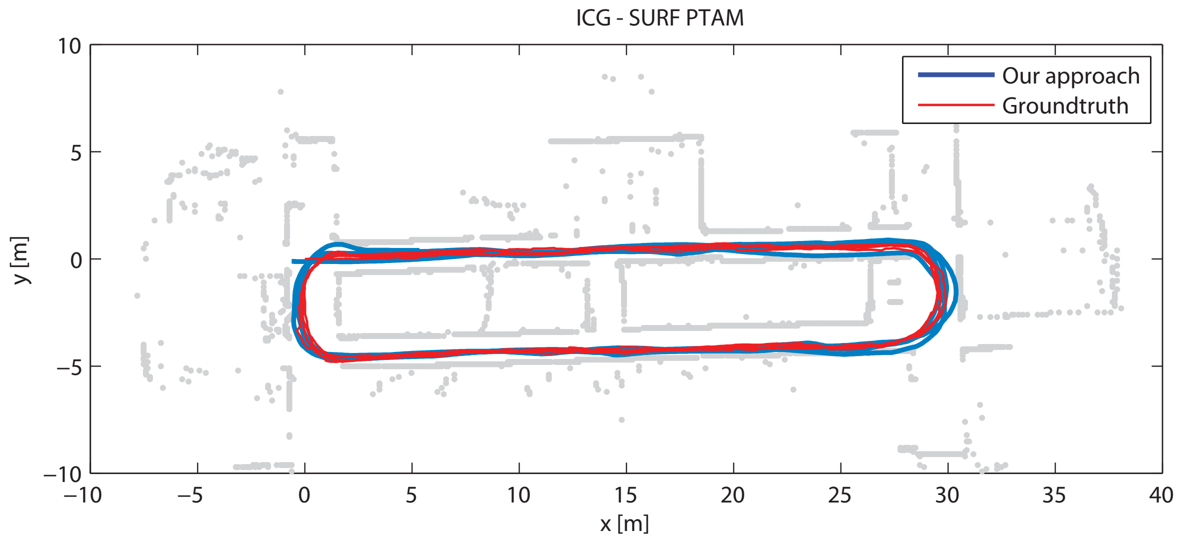


Figure 6.11: Trajectory of the ICG Dataset estimated with our approach using SURF features. The gray lines indicate the walls of the building generated by GMapping.

generated by GMapping. The basic structure of the building is visible, however some walls are not represented in the map, e.g. the inner walls of the loop. Since we did not perform loop closing, walls are added twice. Furthermore, this sequence shows the repeatability of our approach. Although each loop is handled independently, the resulting map of all four

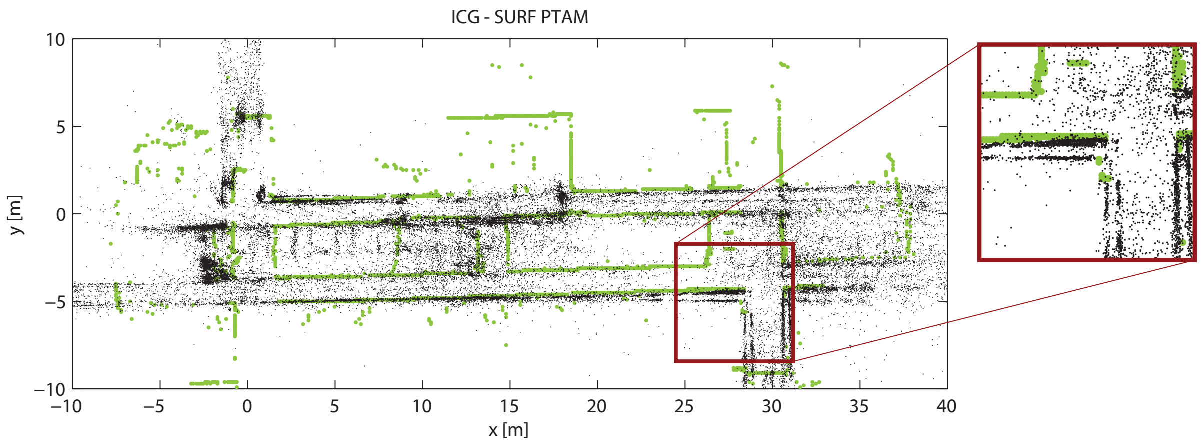


Figure 6.12: Bird's eye view of the map created by SURF PTAM. The map generated by GMapping (green) overlaid by map points (black) of our algorithm. The resulting map of all four loops is consistent although all loops are treated independently. This shows the repeatability of SURF PTAM.

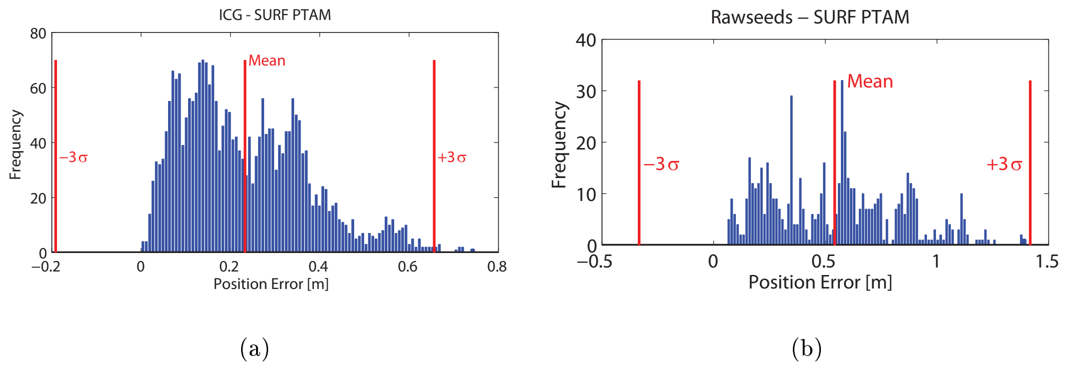


Figure 6.13: Error distribution of PTAM using SURF features. The largest ATE error on the ICG dataset is smaller than 0.5 m (a). In Histogram (b), the error distribution on the Rawseeds dataset is given. The maximum error on this sequence amounts to 1.2 m.

loops is consistent. It is also noticeable that the scale of the trajectory is constant over time, which is a basic requirement for building large metric maps.

As presented in Section 6.3, PTAM performs quite well on the Rawseeds dataset. The main disadvantage here is the enormous number of map points. Traveling about 83 m, PTAM created 31,450 map points, where most of them are generated from false point correspondences resulting in inaccurate 3D positions. These points are useless for local localization as well as for map representation. Figure 6.10 shows the trajectory and the map estimated by SURF PTAM. The trajectory has a better ATE while the map contains only 9,749 map features. The ATE of the trajectory reduces to 0.54 m with a standard deviation of 0.29 m. The estimated trajectory length is 77.91 m (groundtruth 75.05 m). The traveled distance is similar to Stereo PTAM, because the SURF descriptor cannot overcome the problem of little texture.

6.5 Evaluation of PTAM with Loop Closing

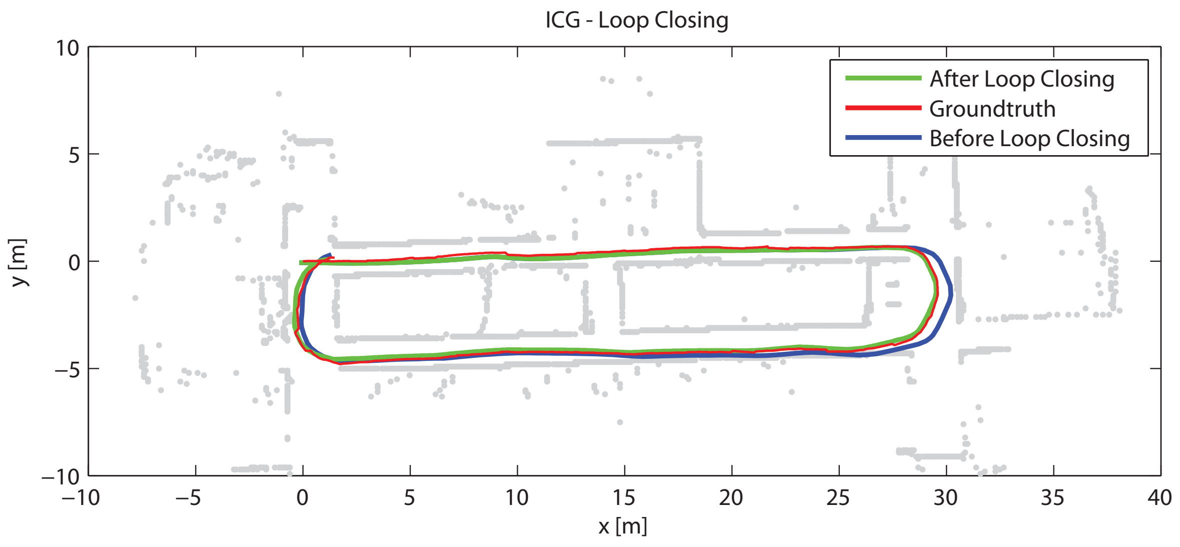


Figure 6.14: ICG-Trajectory before and after loop closing. The blue trajectory is the result of the first cycle without loop detection and correction. The green trajectory is the corrected on.

Due to the incremental nature of SLAM, small errors sum up to a large one at the end of the trajectory. This error can be recognized if the robot visits a single place twice. The method for loop detection and error correction is explained in Section 5.4. We tested the performance of loop closing on the ICG as well as on the Rawseeds scene.

As shown in the last experiment, our approach without loop closing performs quite well on the ICG dataset. Even after a trajectory of 70 m, the error is relatively low and therefore loop correction changes the trajectory only minimally. Figure 6.14 shows the trajectory before and after loop closing compared to the GMapping trajectory. The optimized loop is shortened and fits better to the laser trajectory. The map of all four loops is plotted in Figure 6.15. Since loop closing enables the reuse of map points created at a previous visit, the map of the whole trajectory consists of 31,678 image features, which is a reduction

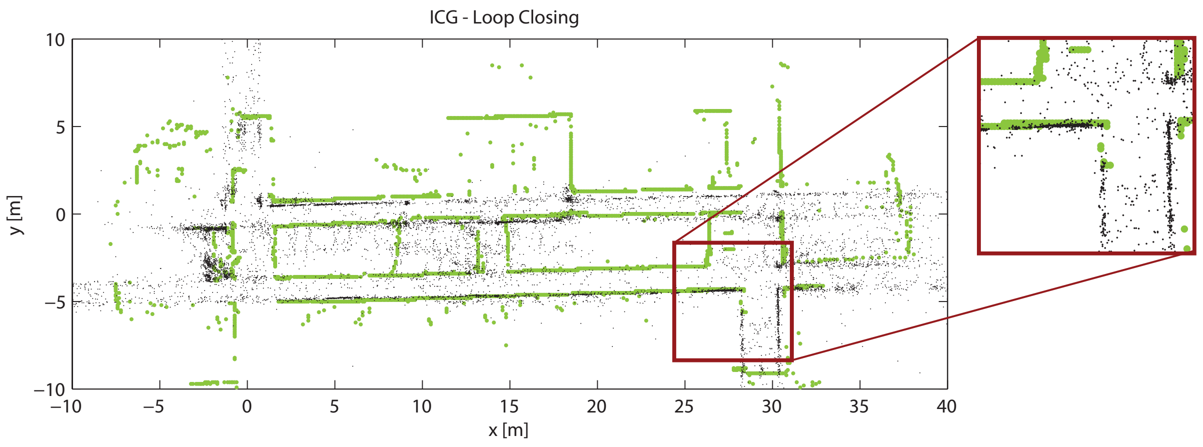


Figure 6.15: ICG map after loop closing. Compared to Figure 6.12, the map is more consistent.

of 30% compared to the map created without loop closing. The map also shows less inconsistent features like walls that are represented twice.

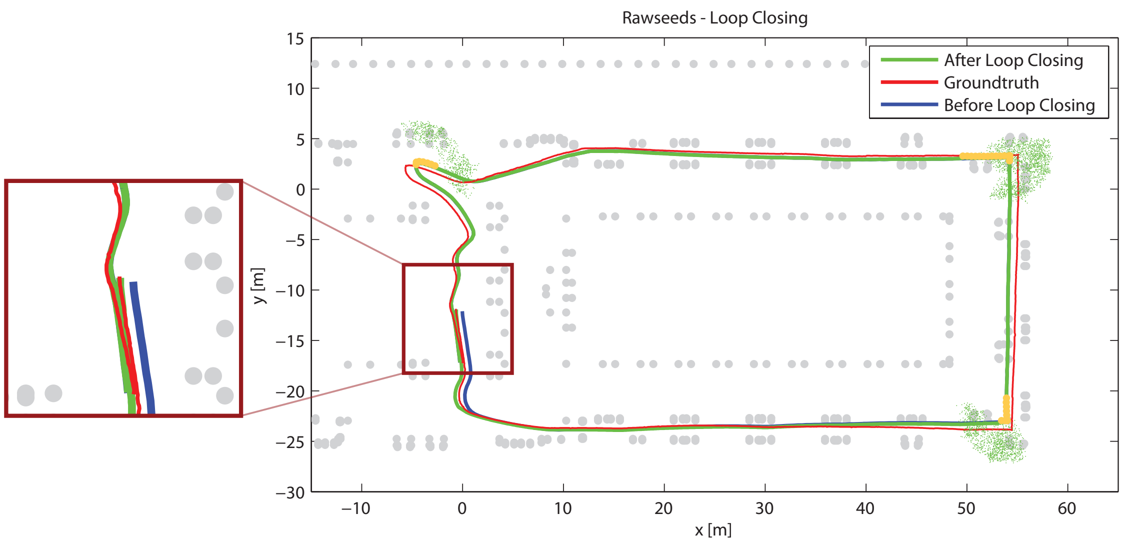


Figure 6.16: Map containing synthetic image features. The yellow lines mark the three parts of the trajectory odometry is used for keeping track of the camera. The green dots are synthetic map points, which are added for loop closing.

The Rawseeds dataset is more complex for loop closing. As shown in Section 6.2, the scene includes frames that show little texture and therefore cannot be tracked by visual

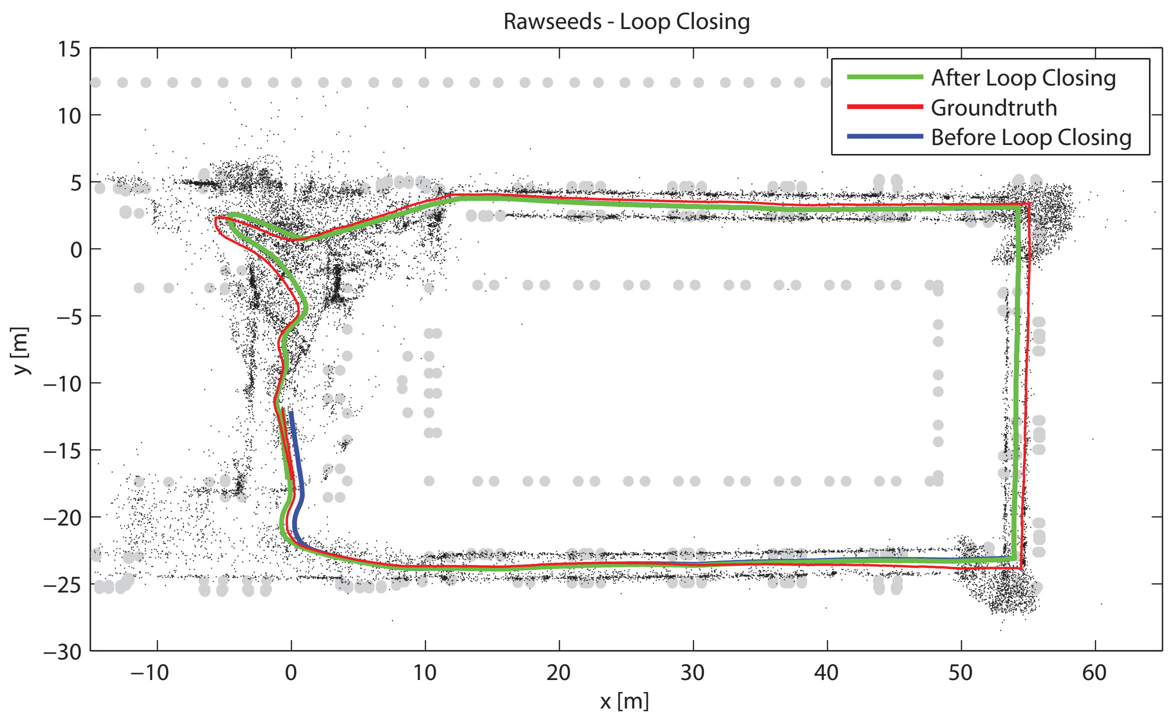
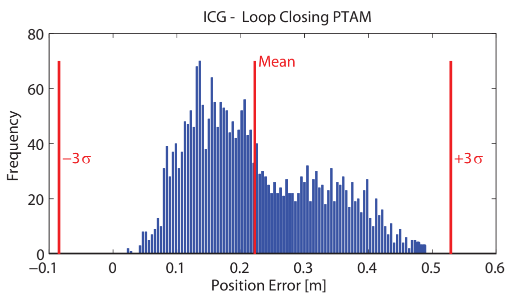


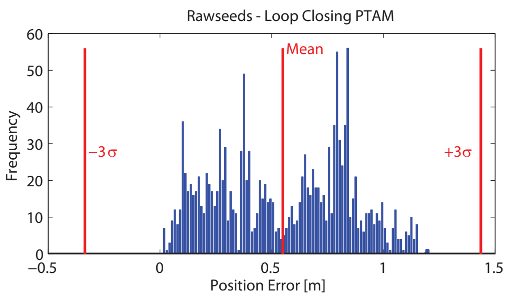
Figure 6.17: Trajectory before and after loop closing. After loop closing the error of the last camera drops from 0.67 m to 0.06 m.

image features. In these cases we keep track of the camera by using odometry information as described in Chapter 5.5. If a keyframe is added to the map that has less than 20 or no image features, we generate up to 80 3D points in front of this keyframe, reproject them back to the last 10 images, and store these positions as measurements for the synthetic feature. This enables us to keep track of the camera using odometry and performing loop closing even if there are frames without visual features.

Figure 6.16 demonstrates the parts which are tracked by odometry and displays the synthetic map points (green dots) that connect keyframes without visual features to the rest of the loop. We added 3,680 synthetic map points to connect 46 keyframes containing less than 30 or no visual image features. It also displays the situation before loop closing. The last tracked camera pose differs by 0.67 m from the groundtruth data. After loop closing, this decreases to 0.06 m (Figure 6.17). The mean ATE decreases from 0.60 m before loop correction to 0.55 m after optimization, which shows that loop correction is able to achieve a higher precision. The error distribution is shown in Figure 6.18(a).



(a)



(b)

Figure 6.18: Error distribution when loop detection and correction is implemented. (a) shows the histogram of error of the ICG dataset and (b) the distribution on the Rawseeds sequence.

6.6 Comparison

To show the accuracy of our method, we compare our results to two other visual SLAM solutions that are provided by the Rawseeds project. Both methods, the CI-Stereo Graph SLAM [29] as well as the Hierarchical Trinocular SLAM [34] system are EKF approaches that reduce computational time by splitting the map into submaps. The main differences between both methods are the applied image features, which are SURF points in the CI-Stereo method, and straight lines in the Trinocular approach.

Since CI-Stereo Graph is based on SURF features, it has similar problems with less structured images and also uses odometry. Furthermore, they implemented an appearance-based loop detection system that is similar to ours. Since CI-Stereo Graph SLAM uses the same image features and loop detection system, it is ideal to evaluate the performance of the underlying optimization algorithm.

The Trinocular SLAM method extracts line segments as image features and triangulates their 3D position. They also use submaps to limit processing time and stop uncertainty propagation through the whole map.

For accuracy evaluation, we aligned all three trajectories in a least squares sense to groundtruth and calculated the ATE for each. The results are shown in Table 6.1. The Trinocular SLAM gets the worst trajectory error with an ATE of 2.55 m followed by the CI-Stereo Graph SLAM (1.12 m). Our approach outperforms with an ATE of 0.55 m. Furthermore, our approach also has the minimal standard deviation of 0.30 m.

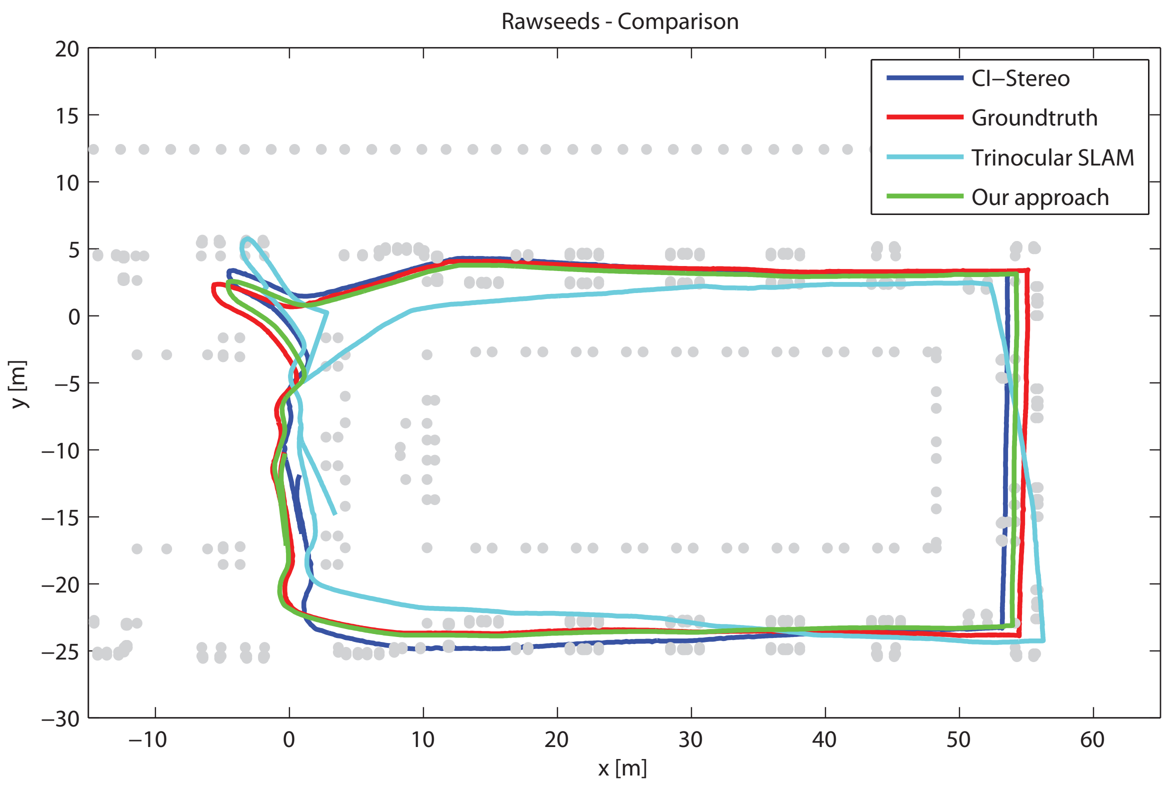


Figure 6.19: Reconstructed trajectory of H-Trinocular, CI-Stereo Graph, and our approach. The H-Trinocular method has major difficulties to reconstruct the trajectory. The path reconstructed by the CI-Stereo Graph [29] method is similar to our trajectory. The path reconstructed by our approach is most similar to the groundtruth.

Algorithm	mean ATE [m]	std ATE [m]
H-Trinocular SLAM	2.55	1.14
CI-Stereo Graph	1.12	0.51
Our approach	0.55	0.30

Table 6.1: Our method compared to other visual SLAM methods. Our algorithm has the best ATE followed by the CI-Stereo Graph SLAM. The Trinocular system is the worst method on this dataset.

6.7 Processing Time

In this thesis, we focused on precise reconstruction of the environment and accurate trajectory estimation, but we did not optimize our approach with respect to processing time. For completeness, we analyze the computational costs on the Rawseeds dataset when applying SURF PTAM with loop closing. Concerning runtime, our method can be split into two parts:

- Local localization
- Map building and loop detection

PTAM uses the FAST corner algorithm for interest point extraction. This is very fast, but results in too many corners. Because of this, we make use of the Harris corner detector that is more complex but provides better repeatability. On a 3.33 GHz Intel Core 2 Duo processor, the overall runtime for pose estimation of a single frame increases from 40 ms using FAST corners to 142 ms (see Figure 6.20). Most of the time is spent on interest point detection (120 ms) and preprocessing, which includes loading and image undistortion. The association of existing map points with the current image using patch-based correlation took only 1 ms. Pose refinement by minimizing the reprojection error finally has a processing time of 2 ms. These processing times demonstrate that most of the time is spent on extracting interest points and the processing time for pose optimization is negligible.

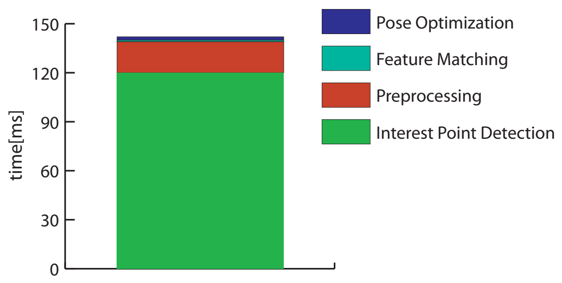


Figure 6.20: Time spent for local localization at a single frame. Most of the time is used for preprocessing the frame and interest point detection (139 ms). Feature matching using patch-based correlation and pose optimization requires 3 ms.

The map extension process is more expensive because not only interest points have to be selected but also feature descriptors have to be extracted and compared. The average time for generating the SURF descriptor is about 587 ms for each stereo pair. Creating new map points, rechecking corresponding image points using SURF, and performing bundle adjustment on the last 10 keyframes requires 330 ms, where most of the time is spent on bundle adjustment (229 ms). Apart from loop closing, this time is also constant with respect to the map size. Only the number of extracted local image features influences the time of map extension.

Experiment	GT length [m]	estimated length [m]	mean ATE [m]	std ATE [m]	map points
Rawseeds PTAM	83.36	81.45	0.88	0.42	31,450
Rawseeds SURF	75.08	77.91	0.54	0.29	9,749
Rawseeds loop closing	186.17	181.50	0.55	0.29	31,678
ICG SURF	272.84	272.97	0.23	0.14	45,350
ICG loop closing	272.95	266.90	0.22	0.10	31,678

Table 6.2: Results of all experiments. The first two columns are the groundtruth length (GT length) of the trajectory and the estimated length (Est. length). The third and fourth show the mean ATE, resp. standard deviation of the ATE. The last column displays the number of map points.

The third part of our approach deaks with loop detection and loop closing. Each time a new keyframe is stored, the vocabulary tree is updated and a query for the nearest neighbors is performed. In our implementation, this uses about 190 ms.

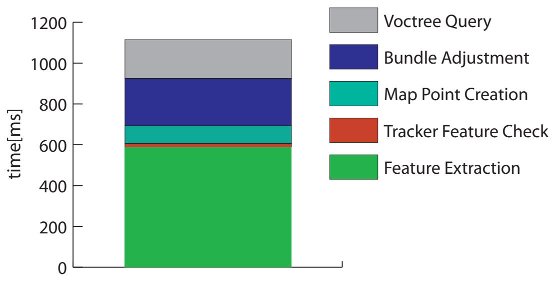


Figure 6.21: Processing time of map extension per keyframe.

Loop closing using bundle adjustment is the most time consuming part and heavily depends on the map size. The complexity of bundle adjustment on the whole loop is $O(N^3M)$, where N is the number of camera positions and M is the number of map points. The loop of the Rawseeds dataset consist of 802 camera positions, 23,000 map points and 230,000 measurements. The full optimization for this dataset took approximately 4 hours.

6.8 Discussion

In our experiments, we compared the slightly modified PTAM approach with our extensions. Since we are interested in a precise metric SLAM algorithm, we extended PTAM by adding stereo support. The first experiment illustrates the performance of this version. As shown on the Rawseeds dataset on a smooth trajectory containing enough visual features, PTAM produces acceptable results. The ICG datasets exhibits the problems introduced by the weak patch-based feature matching. PTAM cannot deal with fast movements, low framerate, and missing frames, which results in a distorted trajectory and map. Furthermore, the patch-based correlation approach generates a large number of map points, which are contaminated by outliers. In order to be memory efficient, it is appreciated to reduce the number of map points.

In the second experiment, we used a stronger feature descriptor for map point initialization and correspondence detection. This increases robustness against fast movements and missing frames as well as accuracy. Using SURF features, the ICG dataset can be processed successfully and a low ATE of 0.23 m compared to the GMapping result shows the accuracy in this scene. Since we used a cell size of 0.1 m for building the map with GMapping, our results are near to the quantization error. Furthermore, this experiment shows the repeatability of our approach. Because we used only the map points generated in last 20 keyframes for local localization, each loop is treated independently and nevertheless similar trajectories and maps are reconstructed. On the Rawseeds sequence, the main advantages of the stronger feature descriptor is the reduced number of map points. This drops from 31,450 features using patch-based correlation to 9,749 map points. Despite the reduced number of map points the accuracy is increased by reducing the ATE error from 0.88 m to 0.54 m. These results clarify that the use of a strong feature detector and descriptor is reasonable for reducing the number of map points as well as for improving accuracy.

In the last experiment, we tested our loop detection and correction approach. On the ICG dataset, we achieved a very small ATE even without loop closing. Hence, loop correction does not reduce the error drastically. The stronger effect is shown in the reduced number of created map points. If we detect a loop, we use the map points created in the 20 nearest keyframes instead of the 20 last created keyframes for local localization. This increases the number of recognized map points in a new keyframe and therefore the number of newly created map points is reduced. Table 6.2 shows the difference in the number of map points with and without loop detection. Instead of 45,350 map points treating all loops independently, the map contains only 31,678 map features when reusing older keyframes.

To process the Rawseeds dataset that contains parts that cannot be tracked by visual odometry, we added synthetic map features. This enables us to perform loop correction although we use odometry information. We showed that synthetic map points are a possibility to reconstitute the connectivity between keyframes that do not share common map points. With this techniques we reached an ATE of 0.55 m with a standard deviation of

0.29 m.

In order to asses the accuracy of our method, we compared our results to two other visual SLAM solutions for the Rawseeds dataset. The CI-Stereo Graph uses the same feature descriptor as well as the same method for loop detection, which enables us to evaluate the underlying optimization methods that are the EKF filter and bundle adjustment, respectively. Table 6.1 shows that our method achieved an ATE of 0.55 m that is the half of ATE gained by the CI-Stereo Graph method. Compared to the H-Trinocular SLAM approach, our results are five times better. This shows that this method has major problems to process this sequence.

The results of the accuracy test show that our approach is capable of reconstructing the environment with high metric precision and it outperforms EKF-based methods. The processing time analysis showed that the bottlenecks of our approach are feature detection and loop closing using bundle adjustment. The processing time for feature detection can be drastically reduced by using an SURF detector implementation that is implemented on a GPU. Cornelis et al. [6] presented an implementation that process about 100 640x480 pixel images per second. Feature detection and descriptor creation so would take less than 10 ms, which is sufficient for real-time application. Speeding up bundle adjustment is more challenging, since the problem itself is computational demanding with a complexity of $O(N^3)$. So there came up several ideas to reduce the number of 3D features and frames in optimization. Sibley et al. [32] stated that they can perform loop closing with a fixed number of keyframes and therefore are constant in time. This would be a large step for performing SLAM on large scenes in real-time.

Finally, the performed experiments provide evidence that our approach is able to create accurate metric maps and trajectories of a moving robot. Furthermore, apart from loop correction, the presented method is constant in time with respect to map size. We are able to deal with sequences that cannot be tracked by visual odometry and we outperform other existing visual SLAM approaches.

Chapter 7

Conclusions & Future work

This thesis is concerned with the 3D reconstruction of the trajectory a moving robot covered and its environment using a stereo camera system. As demonstrated in Chapter 6, we are able to generate maps of environments up to several hundred meters with high accuracy using a calibrated stereo camera rig. In order to optimize the system behavior when exploring the same area several times, we implemented a loop closing detection and correction step. This increases accuracy and reduces the number of map points. Apart from loop correction, our approach is constant in time regarding map size.

The proposed method can be distinguished in a localization and a map building part. The localization part estimates the pose of a single camera with respect to a given map. We identify map points in the current image and optimize the cameras pose by minimizing the reprojection error of the map points. The feature matching is assumed by the method proposed by Klein et al. [19] and based on patch-based correlation. This is a fast but sometimes weak method, because it requires a good initial pose estimation. We solved this problem by checking the quality of feature matches with a stronger feature descriptor from time to time.

Since the goal was precise reconstruction of a map and a trajectory, we focused on the map building part. The map consist of so-called *keyframes* and map points. A keyframe consists of two images taken from the stereo cameras simultaneously. Corresponding image points are identified using the SURF feature descriptor and triangulated to a 3D map point. Since the baseline between the cameras is small, the resulting 3D position for far distant objects may be inaccurate. To optimize the 3D position, we recognize existing map points in succeeding images and bundle adjustment optimizes camera poses and map points. Because bundle adjustment is computationally demanding, we use a sliding window mechanism to optimize only the recently added keyframes.

Due to the incremental nature of SLAM methods, small errors may sum up and result in an inaccurate map and trajectory. We compensate this problem by implementing an appearance-based method for loop detection and correction. We correct the camera poses and the map by performing bundle adjustment on the whole loop. The processing time for loop correction is the only step in our algorithm that depends on the map size.

We demonstrated that our system performs very well and with high accuracy. This is a good starting point for future research. In the following, we outline research challenges and open questions that may lead to further improvements of our proposed approach.

In Chapter 6, we showed that the loop detection mechanism works for relatively small scenes that have little repetitive texture. This method fails on long trajectories, where parts of the trajectory have similar texture, e.g. corridors or highways. In this case, similarity of images does not indicate that they are taken at the same place. A solution for this problem is presented in [7]. The authors also compare images using a vocabulary tree but additionally, they use only those features that are characteristic for a certain place. While exploring the environment, their approach learns a distribution of features that are characteristic and ignores features that do not allow inference for a certain place. This reduces the number of false-positive loop detections significantly. The authors demonstrate that their approach is able to detect loops in trajectories of several hundred kilometers without any false detection. Hence, this is might be a reasonable approach for loop detection for large-scale SLAM systems.

Our experiments show that loop detection is also important for map size reduction. Instead of 45,000 features generated when using no loop detection, the map contains only 30,000 features when reusing features generated in a previous cycle. After the first cycle, when each place is visited for the first time, the map contains approx. 12,000 features. In each further cycle, 6,000 new map points are generated. In an optimal solution, where the environment does not change and the robot drives the same way, the number of map points would stay constant over all four cycles. In our implementation, a new map point is created from an interest point if no existing map point can be matched with this interest point. Thus, the constant number of newly constructed map points indicates that a large number of map points are not rediscovered in the later cycles. That means that map points which do not account for localization in later cycles could be removed to decrease the map size.

Our local localization approach mainly depends on camera information and uses odometry information in exceptional cases when visual odometry is not possible. The accuracy and robustness can be further increased by combining different sensors in every localization step. In Chapter 5.5, we presented an idea for fusing different sensor information in image space. We generate synthetic 3D points that indicate the estimated movement of a single sensor. The idea is promising because different sensors can be combined in a single optimization step. Howecer, a deeper mathematical inspection is required for a reasonable use of this idea. Another problem of local localization and map building is motion blur. Since our method is based on local interest points, i.e. corners, motion blur prevents corner detection. Klein et al. [20] demonstrated that lines can be extracted even if the image is blurred by fast camera motion. Hence, constructing the map of interest points and lines improves local localization in case of motion blur.

The processing time analyzed in Section 6.7 shows that most computational time is used for feature detection, which prevents our implementation from being applicable under real-time conditions. A solution for this problem could be the use of a GPU-accelerated SURF implementation as presented in [6]. This approach is able to process more than 100 frames per second. This way localization and incremental map building can be performed in real-time. However, this requires powerfull and energy consuming hardware. Because of the today's limited energy density of battaries, a long running mobile robot with such demanding hardware is not possible yet.

In order to run the application online on a robot, we also have to split localization and map building into two parallel threads, so that map extension does not interfere localization. The bottleneck of map building is loop correction. A full optimization of the map may take several hours. To overcome this problem, Sibley et al. [32] presented a solution for full loop closure in constant time. However, their algorithm requires a largely different representation of map points and keyframes than available in our approach. Another interesting method is presented by Konolige et al.[21] who try to reduce the number of features.

Originally, PTAM was developed for tracking a monocular camera in a unknown environment. We changed the implementation to a stereo setup for different reasons: The initialization of new map points requires a certain baseline between two camera poses to achieve a precise reconstruction. Especially in indoor situations where the camera turns without translation this is a serious problem. Datasets like the ICG scene for example with tight curves, cannot be tracked by a monocular camera using our approach. The second reason for using a stereo camera is map initialization: In order to build a true scale map, the distance between the first two images used for map initialization, has to be known. This also points out a drawback of SLAM algorithms based on bundle adjustment: At each time step they require fully initialized 3D map points. In contrast, EKF-based methods are able to handle map points with unknown distance to the camera by using the inverse depth representation [25]. Both, map initialization and small baseline between cameras are a minor problem in the EKF framework. So, a challenging question is if the inverse depth representation can be exploited by our approach.

We presented a method for a visual SLAM in the field of autonomous, wheeled robots based on PTAM developed by Klein et al. We have shown that our method outperforms other methods with respect to metric accuracy and map size using bundle adjustment as underlying optimization method. In order to achieve these results, we proved that a strong feature descriptor is essential, since accuracy of SLAM depends on the accuracy of feature matching. Our proposed idea for sensor fusion allowed us to use bundle adjustment for loop correction, even if parts of the sequence are tracked by odometry. We further showed that a loop detection mechanism improves precision and reduces the map size when visiting the same place several times. The results on the Rawseeds benchmark dataset confirm the conclusion stated by Strasdat et al. [36] that bundle adjustment is able to outperform EKF-based visual SLAM algorithms.

Bibliography

[1] Baker, S. and Matthews, I. (2001). Equivalence and efficiency of image alignment algorithms. In *Proceedings of the 2001 IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1090 – 1097.

[2] Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). Surf: Speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110:346–359.

[3] Castle, R. O., Klein, G., and Murray, D. W. (2008). Video-rate localization in multiple maps for wearable augmented reality. In *Proc 12th IEEE Int Symp on Wearable Computers, Pittsburgh PA, Sept 28 - Oct 1, 2008*, pages 15–22.

[4] Ceriani, S., Fontana, G., Giusti, A., Marzorati, D., Matteucci, M., Migliore, D., Rizzi, D., Sorrenti, D. G., and Taddei, P. (2009). Rawseeds ground truth collection systems for indoor self-localization and mapping. *Auton. Robots*, 27(4):353–371.

[5] Civera, J., Davison, A., and Montiel, J. (2007). Inverse Depth to Depth Conversion for Monocular SLAM. In *IEEE International Conference on Robotics and Automation*.

[6] Cornelis, N. and Van Gool, L. (2008). Fast scale invariant feature detection and matching on programmable graphics hardware. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW 0́8. IEEE Computer Society Conference on*, pages 1–8.

[7] Cummins, M. and Newman, P. (2009). Highly scalable appearance-only slam -fab-map 2.0. In *Robotics Science and Systems (RSS)*, Seattle, USA.

[8] Davison, A. and Kita, N. (2001). Sequential localization and map-building for real-time computer vision and robotics. *RobAS*, 36(4):171–183.

[9] Doucet, A., Freitas, N. d., Murphy, K. P., and Russell, S. J. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *UAI '00: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 176–183, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[10] Eade, E. and Drummond, T. (2007). Monocular slam as a graph of coalesced observations. In *Proc. 11th IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil.

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

[12] Fujitsu Laboratories Ltd. (2010). Fujitsu begins limited sales of service robot enon. `http://www.fujitsu.com/global/news/pr/archives/month/2005/20050913-01.html`.

[13] Grisetti, G., Stachniss, C., and Burgard, W. (2005). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*.

[14] Grisetti, G., Tipaldi, G. D., Stachniss, C., Burgard, W., and Nardi, D. (2007). Fast and accurate slam with rao-blackwellized particle filters. *Robot. Auton. Syst.*, 55(1):30–38.

[15] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151.

[16] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition.

[17] Holmes, S., Sibley, G., Klein, G., and Murray, D. W. (2009). A relative frame representation for fixed-time bundle adjustment in sfm. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 2631–2636, Piscataway, NJ, USA. IEEE Press.

[18] Huber, P. (1974). *Robust Statistics*. Wiley, New York.

[19] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan.

[20] Klein, G. and Murray, D. (2008). Improving the agility of keyframe-based SLAM. In *Proc. 10th European Conference on Computer Vision (ECCV'08)*, pages 802–815, Marseille.

[21] Konolige, K. and Agrawal, M. (2008). Frameslam: From bundle adjustment to real-time visual mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077.

[22] Lina María Paz, Pedro Piniés, J. D. T. and Neira, J. (2008). Large scale 6dof slam with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957.

[23] Mei, C., Sibley, G., Cummins, M., Newman, P., and Reid, I. (2010). Rslam: A system for large-scale mapping in constant-time using stereo. *International Journal of Computer Vision*. Special issue of BMVC, accepted for publication.

[24] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI.

[25] Montiel, J., Civera, J., and Davison, A. (2006). Unified inverse depth parametrization for monocular slam. In *Proceedings of Robotics: Science and Systems*, Philadelphia, USA.

[26] Murray, R. M., Sastry, S. S., and Zexiang, L. (1994). *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA.

[27] Nister, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 1:652–659.

[28] Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *CVPR06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2161–2168, Washington, DC, USA. IEEE Computer Society.

[29] Pedro Piniés, L. M. P. and Tardós, J. D. (2009). In *IEEE Int. Conf. Robotics and Automation*, pages 3913–3920, Kobe, Japan.

[30] Rosten, E. and Drummond, T. (2006). Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443.

[31] Rousseeuw, P. J. and Leroy, A. M. (1987). *Robust regression and outlier detection*. John Wiley & Sons, Inc., New York, NY, USA.

[32] Sibley, G., Mei, C., Reid, I., and Newman, P. (2009). Adaptive relative bundle adjustment. In *Robotics Science and Systems (RSS)*, Seattle, USA.

[33] Sim, R., Elinas, P., and Griffin, M. (2005). Vision-based slam using the rao-blackwellised particle filter. In *In IJCAI Workshop on Reasoning with Uncertainty in Robotics*.

[34] Sorrenti, D. G., Matteucci, M., Marzorati, D., and Furlan, A. (2009). Benchmark solution to the stereo or trinocular slam - bicocca 2009-02-25b bp `http://www.rawseeds.org/rs/rawseeds/rs/assets/solutions_data/4adc84637e0a0/BS_milan_Bicocca_25b.pdf`.

[35] Stachniss, C., Grisetti, G., and Burgard, W. (2007). Analyzing gaussian proposal distributions for mapping with rao-blackwellized particle filters. In *International Conference on Intelligent Robots and Systems (IROS)*, San Diego, CA, USA.

[36] Strasdat, H., Montiel, J. M. M., and Davison, A. J. (2010). Real-time monocular SLAM: Why filter? In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, US.

[37] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

[38] Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley.

[39] Zhang, Z. (1998). A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334.

[40] Zhou, W., Miró, J. V., and Dissanayake, G. (2008). Information-efficient 3-d visual slam for unstructured domains. *IEEE Transactions on Robotics*, 24(5):1078–1087.