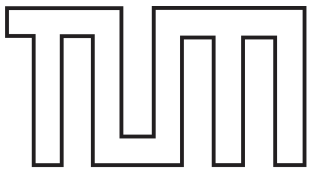


Technische Universität München
Institute for Media Technology
Prof. Dr.-Ing. Eckehard Steinbach

Diplomarbeit

Visual Localization based on Binary Features

Author:	Julian Straub, B.Sc.
Matriculation Number:	3026226
Address:	Otte-von-Stetzlingen Str. 24 86316 Friedberg
Advisors:	Dipl.-Ing. Sebastian Hilsenbeck, M.Sc. and Dipl.-Ing. Georg Schroth
Begin:	01.03.2012
End:	21.08.2012

Abstract

Visual indoor localization and navigation for hand-held devices is becoming a hot topic due to its potentially high localization accuracy and its independence from hardware installations. A typical visual localization system consists of a visual odometry system and a global localization mechanism for initialization. Additionally, a relocalization algorithm is usually employed to make the visual odometry system more robust against tracking loss (e.g. due to rapid motion). The deployment of feature-based localization algorithms on hand-held devices has mostly been avoided due to computational complexity of the feature descriptors. Binary features like Binary Robust Features (BRIEF) [CLSF10] promise to overcome this problem as they are about 40 times faster to extract than the quasi standard descriptor SURF [BTVG06]. At the same time they offer comparable matching precision under small rotations and scale changes. This thesis investigates the deployment of binary features for global pose recovery as well as relocalization within the visual odometry system. Integrated in the Parallel Tracking and Mapping (PTAM) algorithm [KM07], the developed BRIEF-based relocalization algorithm was found to yield accurate, fast and robust pose recovery even in sparsely textured and repetitive indoor environments. For global localization, a novel quantizer for binary features was developed to enable Content-based Image Retrieval (CBIR) [SZ03]. In combination with a Virtual Views database [HSH+12b] the high distinctiveness of BRIEF features can thus be leveraged to perform accurate global visual localization. This large scale visual localization system matches the precision of a state of the art SURF-based system while reducing the computational burden of feature extraction significantly. Deploying binary features for pose recovery leads to significant speedups in the feature extraction without loss in localization performance. This makes binary features ideal for mobile-device visual localization systems.

Contents

Chapter 1

Introduction

Visual indoor localization is becoming a hot topic recently because it has the potential to outperform competing technologies in terms of localization accuracy. Accurate and efficient position tracking is the key technology for indoor navigation and location-based services like for example location specific advertisements or warnings. Big companies like Google [Goo11] and Microsoft [Bin11] have launched the integration of indoor maps into their map systems already. Other companies like Nokia [Nok10], Meridian [Mer11] or IndoorAtlas [Ltd12] are developing applications for hand-held devices which are able to provide indoor navigation and location-based services.

Existing localization approaches utilize the typical built-in sensors of current high-end cellphones: inertial measurement units (IMUs) which measure accelerations and rotational speeds, magnetic field sensors, WiFi and cellphone receiver cards which can be used to determine the signal strength of nearby base stations. The accuracy of positioning solutions relying on wireless signal strength measurements is in the rang of 3-10 m [LDBL07, CPIP10, VdLH$^+$07] given a dense coverage of the building with WiFi hotspots. This is because at least three WiFi base stations have to be in range in order to allow successful triangulation [Fri33] of the cellphone. If fewer hotspots are in range, the localization accuracy degrades to room level. Some approaches are also fusing the signal strength measurements with IMU and magnetic field data [KKD11]. IndoorAtlas' localization system [Ltd12] is solely based on magnetic field fingerprints of buildings.

Visual odometry and localization systems are able to provide more accurate positioning as well as precise orientation estimates [KM07, JS11, LS12]. This is very important since a person using a localization system to navigate through a building will only perceive the system as useful if it works reliably and accurately. Especially in complex and dense indoor environments where hallways and rooms are not much wider than 2-3m, it is important to have very accurate and fast positioning in order to provide correct guidance for the user. Typically, buildings in which an indoor navigation system would be beneficial such

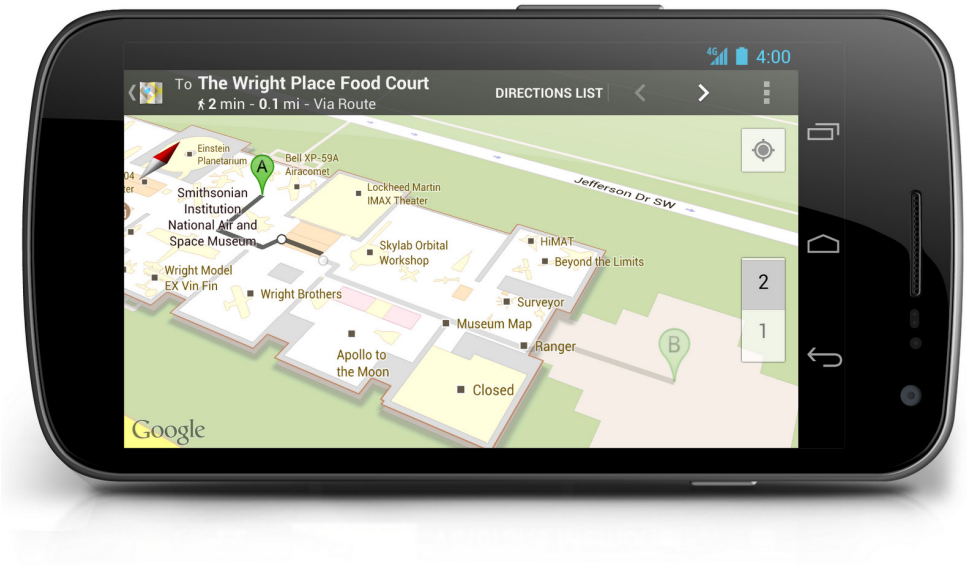


Figure 1.1: Google indoor map of the Smithsonian National Air and Space Museum in Washington D.C. on an Android phone (image from `http://googleblog.blogspot.de/2012/07/indoor-google-maps-help-you-make-your.html`).

as office buildings, universities and supermarkets with narrow rows of goods exhibit these challenging situations.

Additionally, visual localization systems scale better than WiFi-based systems since no hardware infrastructure has to be provided. WiFi-based localization requires a building to be densely populated with WiFi hotspots. This may be given in an office building but is not likely in museums or art galleries. An initial mapping of the indoor environments WiFi signal distribution is usually necessary to provide higher accuracy. Similarly, visual localization systems need an initial visual mapping of the building. Hence both techniques need an initial mapping of the environment in order to work properly but visual localization has the advantage that no additional hardware infrastructure hast to be installed in the building. This allows visual localization systems to be deployed in large scale indoor environments easily.

Unlike WiFi-based localization systems, which are limited to areas with good WiFi coverage, cameras have virtually unlimited range in indoor environments. Hence the accuracy does not suffer even in large rooms where WiFi-based systems have problems since the WiFi signal strength attenuates with $1/R^2$ as any electromagnetic wave. This limited range becomes an issue in large halls which are typically found in airports and railway stations. Here visual localization systems are advantageous since any textured structure of the building in the view of the camera can be used to determine the position of the user.

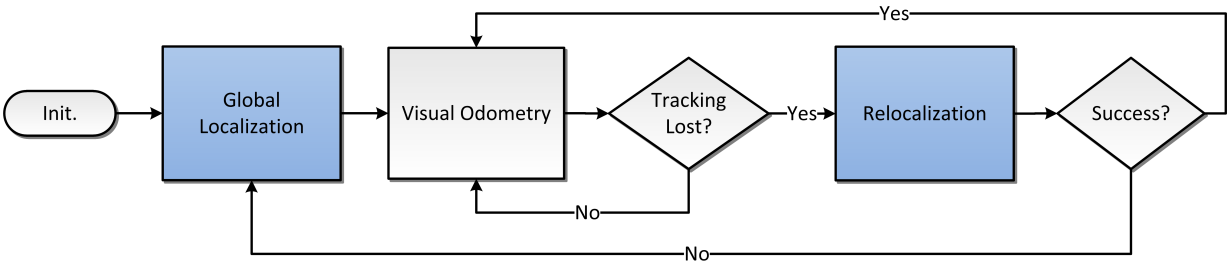


Figure 1.2: This is an overview over the proposed visual localization architecture. The main focus of this thesis is on the blue blocks: algorithms based on binary features for relocalization and global localization.

Figure (1.2) depicts the architecture of a typical visual localization system as it will be considered in this thesis. A visual odometry system is running on the hand-held device which incrementally estimates the position of the cellphone. The odometry is initialized using a global localization algorithm which mainly runs on a server. In case of tracking failure of the visual odometry system, a relocalization system first tries to recover the pose of the camera within the map on the cellphone. If this fails, the whole system is restarted from an initial pose obtained from the global localization aided by the server.

Key to visual localization on hand-held devices are highly efficient algorithms. Despite the ever increasing computational power of todays cellphones, the energy storage is is limiting the amount of computation an application can demand over an extended period of time. Thus, key to the success of a navigation solution based on visual localization techniques is a lightweight implementation that can be run in the background. This has the additional advantage that computational power is available for location-based services, augmented reality or other applications relying on the localization algorithm.

In state of the art feature-based localization system, the extraction of complex features like SURF [BTVG06] or SIFT [Low99] are the most time-consuming part of the algorithm. This is because the aim of those descriptors is invariance with respect to scale, lighting, rotation and distortion. These properties are desirable in order to obtain unique descriptors for points in the 3D environment but demand complex computations.

Recently, simple binary feature descriptors such as BRIEF were shown to perform equally well as complex descriptors like SURF or SIFT for small rotation and scale differences [CLSF10]. This is in stark contrast to any intuition because the binary descriptor is simply composed of the results of image intensity comparisons at randomly selected positions around the keypoint position. The key advantage of binary features is their rapid extraction speed which for BRIEF is in the order of 40 times faster than for SURF [CLSF10]. Additionally, the binary descriptors are more compact memory wise.

The question is: are binary features able to cope with the challenges of indoor environments like sparsely textured surfaces and highly repetitive structures? If this was the case, the

advantages of binary features, namely rapid extraction and storage efficiency, could be leveraged to enable visual localization on hand-held devices.

In this thesis, the deployment of binary features for visual localization is evaluated. As depicted in Figure (1.2) there are two different scenarios of localization that have to be solved: (1) relocalization within the map on the mobile device and (2) global pose recovery in large visual maps of buildings. A visual odometry system capable of both techniques would be able to to provide robust visual positioning of a mobile device in an indoor setting.

The first scenario of relocalization is important in order to enhance the robustness of visual odometry systems against rapid motion, occlusions in dynamic environments and bad lighting conditions. These disturbances, although probably limited to brief moments, are likely to cause the tracking mechanism of a visual odometry system to fail. Therefore it is crucial for robust operation to have a system for fast relocalization in place which can be used to restart tracking.

The second scenario is large scale visual localization. Here, the goal is to allow for initial localization of the hand-held device in extensive indoor environments. From this position estimate the visual odometry system can be started to provide accurate positioning online. Due to the storage and memory requirements, large scale visual localization will have to be aided by a server. Here it becomes important that information which has to be sent to the server to allow positioning is as compact as possible to minimize delay due to data uploading. The eight times smaller size of for example BRIEF features compared to more complex features like SURF would allow significant reduction of the amount of data to be transferred to a server.

The thesis is structured as follows: first, in Chapter (2) the visual odometry system is introduced wich is deployed for the evaluation of the relocalization algorithm. In the same chapter a detailed description of important computer vision techniques is given which are needed throughout the thesis. Second, in Chapter (3) a binary-feature-based relocalization system is proposed and all algorithms involved are described. Third, in Chapter (4) Content-based Image Retrieval (CBIR) is introduced as a method for visual localization in large image databases of extensive indoor environments. Algorithms are described that allow CBIR using binary features. An overview of the software developed during the course of this work is given in Chapter (5) in order to foster the reusability of the code. The results are presented and evaluated in Chapter (6), before a conclusion is drawn in Chapter (7). Finally, a brief outlook of possible directions for further research is given in Chapter (8).

Chapter 2

Monocular Visual Odometry

The aim of monocular visual odometry is to provide incremental movement information of a system equipped with a single camera. Areas of application for visual odometry systems include for example mobile localization on hand-held devices [KM09, LS12] or odometry for flying robots [BWSS10].

The essential problem which most monocular visual odometry system have to solve is that in order to estimate the 3D position of a feature at least two observations from different angles are required. One observation of a 3D feature point in an image only provides an observation of the direction but not the depth of the feature. With a single camera, two observations can only be obtained from frames that are distant in time and pose. This means that the algorithm has to find the corresponding observations in the two frames that originate from one feature in 3D space.

There are roughly two kinds of algorithms that can provide monocular visual odometry. On the one hand, visual simultaneous localization and mapping (SLAM) algorithms can provide both, estimation and optimization of the trajectory as well as the 3D structure of the environment over time. The focus is on finding a globally optimal description of the scene and the movement of the camera.

On the other hand, the pure visual odometry type of algorithms are mainly concerned with estimating the incremental movement of the camera. While some of them also estimate the geometry of the environment to aid the computation of the visual odometry, they do usually not try to find an optimal solution and discard or ignore knowledge of the scene which is not usefully anymore due to distance in location and time. These two measures are important to limit the amount of memory and computation time such that real-time operation becomes feasible.

2.1 Related Work

The seminal work of Davison [Dav03, DRMS07] in 2003 first demonstrated that monocular visual Simultaneous Localization and Mapping (SLAM) is possible. The algorithm uses an Extended Kalman filter (EKF) to obtain an estimate of the whole state vector consisting of the pose of the camera and the positions of features in 3D. In order to keep track of the movement of the camera, EKF updates are computed at each frame. Due to the computational burden $O(N^2)$ of the EKF updates, this system is limited to about 100 features in the map.

Sim et al. [SEG+05] first deployed a particle filter to solve the visual SLAM problem. Their system relies on a large database of SIFT feature descriptors to estimate the position of robot in 2D. With a runtime of 11.9s per frame the system is unfit for real-time operation.

This limitation was removed by Eade and Drummond [ED06] by using a top down search for features in each new frame instead of searching for extracted features in feature database. Their system relies on a FastSLAM [MTKW03] style particle filter for SLAM and is able to run in real-time for maps of up to 250 landmarks.

With Parallel Tracking and Mapping, Klein and Murray [KM07] introduced the idea of splitting the task of tracking the camera pose and estimating the 3D positions of features in the map. This has the advantage that the mapping thread can use any left over time not needed for camera pose tracking to improve the quality of the map. With this approach their system is able to construct maps with thousands of features while still performing real-time camera pose tracking. In [KM09] they even show that the PTAM paradigm allows visual SLAM to be performed on an iPhone 3G for small workspaces

The visual odometry system by Jones and Soatto [JS11] uses an Inertial Measurement Unit (IMU) in addition to a camera to estimate the trajectory of a moving car. The different components of their system are highly interconnected to allow for an very efficient implementation. For example, their RANSAC algorithm used for loop closure detection uses the direction of gravity obtained from the IMU to allow for 2-point pose estimation. Their map is represented as a graph of locations where a location is defined as a set of features one would expect to observe at the respective location. They achieve an accuracy of less than 0.5% drift in distance over runs of up to 30km.

Lupton and Sukkarieh [LS12] also use an IMU in addition to a camera to support the odometry estimation within buildings. On sequences of around 750 frames, their custom built system is able to track its path and to estimate the scale of the path using the IMU information. This is important since the true scale of a map cannot be covered solely from a monocular camera. The system estimates the trajectory with an accuracy of less than 1% of the path length.

Indelman et al. [IWKD12] utilize the incremental Smoothing and Mapping algorithm (iSAM2) [KJR+11] to fuse information from a stereo camera, an IMU and GPS

data. Smoothing tries to come up with a globally consistent trajectory, while filtering approaches like the Extended Kalman Filter (EKF) are only concerned with incorporating observations into the state estimate as they come. Sensor information which is usually obtained at different rates and times is directly integrated into the graph which represents the estimation problem. iSAM2 operates incrementally and thus enables globally consistent trajectory estimation as new observations are made.

2.2 3D Computer Vision Preliminaries

3D Computer Vision is the theory of inferring the 3D structure of a scene that is observed by a camera from different view points also called poses. This problem is challenging since the observations of the scene are in the 2D image of the camera and hence lack the information about the depths of the observations. Additionally, the transformation from 3D into the image is non-linear.

In the following first the theory of Lie Groups and Lie Algebras will be introduced. These can be used to describe rigid body transformations in 3D space. Second, the math for projecting a 3D point into the camera image given a camera model will be shown. And finally, the Levenberg Marquardt algorithm will be explained which, in conjunction with the theory of M-Estimators, can be used to robustly minimize a quadratic cost function and is thus widely used in Computer Vision.

These preliminaries are important for the understanding of consecutive sections. The level of depth in the preliminaries section is purposefully deep in order to give a thorough introduction to the most important topics which can serve as a future reference.

2.2.1 Lie Groups and Lie Algebras for 3D Vision

Poses and rigid body motion, like translation and rotation, can be interpreted as Lie Groups. This allows minimal parameterizations of those transformations within the respective Lie Algebras [MSKS04, MLS94, Agr06]. Being able to obtain a minimal parameterization is important for incremental optimization algorithms to keep the number of variables small.

A pose in 3D space is usually represented by the translation $\mathbf{t} \in \mathbb{R}^3$ and the rotation $\mathbf{R} \in \mathbb{R}^{3\times3}$ with respect to some global coordinate frame. In the homogeneous coordinate representation, the pose is defined by a 4×4 matrix

$$\mathbf{T} = \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}^T & 1 \end{array}\right) \in \mathbb{R}^{4\times4}. \tag{2.1}$$

This representation is over-parameterized since it uses twelve parameters to describe a pose in 3D space which does only have six parameters: three parameters for the position and three for the orientation.

The pose $\mathbf{T}$ in 3D space is a member of the Special Euclidean Group in 3D ($\mathrm{SE}(3)$). This group comprises all distance and orientation preserving mappings from $\mathbb{R}^3$ to $\mathbb{R}^3$. The rotation $\mathbf{R}$ is a member of the Special Orthogonal Group in 3D ($\mathrm{SO}(3)$), the group of all orientation preserving orthogonal matrices.

Each Lie Group has a Lie Algebra associated in which the Lie Group is minimally parameterized. The Lie Algebra of $\mathrm{SO}(3)$ is denoted by $\mathrm{so}(3)$. This Lie Algebra is the set of all skew symmetric matrices:

$$\mathrm{so}(3) = \left\{ [\boldsymbol{\omega}]_\times \in \mathbb{R}^{3\times 3} \mid \boldsymbol{\omega} \in \mathbb{R}^3 \right\}, \tag{2.2}$$

where the operator $[\cdot]_\times$ is constructing a skew symmetric matrix in $\mathbb{R}^{3\times 3}$ from a vector in $\mathbb{R}^3$. This operation is defined as

$$[\boldsymbol{\omega}]_\times = \left[\begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \right]_\times = \begin{pmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{pmatrix}. \tag{2.3}$$

The conversion from the Lie Algebra $\mathrm{so}(3)$ to the Lie Group $\mathrm{SO}(3)$ is performed using the exponential mapping $\exp : \mathrm{so}(3) \to \mathrm{SO}(3)$:

$$\begin{aligned} \boldsymbol{\omega} &\in \mathbb{R}^3 \\ \theta &= ||\boldsymbol{\omega}||_2 = \sqrt{\boldsymbol{\omega}^T \boldsymbol{\omega}} \\ \exp(\boldsymbol{\omega}) &= \mathbf{I} + \tfrac{\sin\theta}{\theta} [\boldsymbol{\omega}]_\times + \tfrac{1-\cos\theta}{\theta^2} [\boldsymbol{\omega}]_\times^2 = \mathbf{R} \in \mathrm{SO}(3) \end{aligned} \tag{2.4}$$

Note, that $\boldsymbol{\omega}$ describes the rotation axis around which $\mathbf{R}$ rotates about the angle $||\boldsymbol{\omega}||_2$. The inverse operation to the exponential mapping is the logarithm $\ln : \mathrm{SO}(3) \to \mathrm{so}(3)$:

$$\begin{aligned} \mathbf{R} &= \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \in \mathrm{SO}(3) \\ ||\boldsymbol{\omega}||_2 &= \arccos\left(\tfrac{tr(\mathbf{R})-1}{2} \right) \\ \ln(\mathbf{R}) &= \tfrac{||\boldsymbol{\omega}||_2}{2\sin(||\boldsymbol{\omega}||_2)} \begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix} = \boldsymbol{\omega} \in \mathbb{R}^3. \end{aligned} \tag{2.5}$$

The Lie Algebra associated with the Lie Group $\mathrm{SE}(3)$ of a pose on 3D space is denoted $\mathrm{se}(3)$. Again, there is the exponential mapping $\exp : \mathrm{se}(3) \to \mathrm{SE}(3)$ and the logarithmic mapping $\ln : \mathrm{SE}(3) \to \mathrm{se}(3)$ for the conversion between Lie Group and Lie Algebra.

The exponential map from $[\mathbf{v};\boldsymbol{\omega}] \in \mathrm{se}(3)$ to $\mathbf{T} \in \mathrm{SE}(3)$ is defined as:

$$\begin{aligned} \mathbf{v}, \boldsymbol{\omega} &\in \mathbb{R}^3 \\ \theta &= ||\boldsymbol{\omega}||_2 = \sqrt{\boldsymbol{\omega}^T \boldsymbol{\omega}} \\ \mathbf{R} &= \mathbf{I} + \tfrac{\sin\theta}{\theta} [\boldsymbol{\omega}]_\times + \tfrac{1-\cos\theta}{\theta^2} [\boldsymbol{\omega}]_\times^2 \\ \mathbf{V} &= \mathbf{I} + \tfrac{1-\cos\theta}{\theta^2} [\boldsymbol{\omega}]_\times + \tfrac{\theta-\sin\theta}{\theta^3} [\boldsymbol{\omega}]_\times^3 \\ \exp\begin{pmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{pmatrix} &= \left(\begin{array}{c|c} \mathbf{R} & \mathbf{V}\cdot\mathbf{v} \\ \hline \mathbf{0}^T & 1 \end{array}\right) = \mathbf{T} \in \mathrm{SE}(3), \end{aligned} \tag{2.6}$$

where $\mathbf{I}$ is the identity matrix. The inverse operation, the logarithmic map from $\mathbf{T} \in \mathrm{SE}(3)$ to $[\mathbf{v};\boldsymbol{\omega}] \in \mathrm{se}(3)$, can be computed by first finding $\boldsymbol{\omega}$ using the logarithmic map from $\mathrm{SO}(3)$ to $\mathrm{so}(3)$ as outlined in Equation (2.5). Then $\mathbf{v}$ can be obtained in the following way:

$$\begin{aligned} \mathbf{T} &= \left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}^T & 1 \end{array}\right) \in \mathrm{SE}(3) \\ \theta &= ||\boldsymbol{\omega}||_2 \\ \mathbf{V} &= \mathbf{I} + \tfrac{1-\cos\theta}{\theta^2} [\boldsymbol{\omega}]_\times + \tfrac{\theta-\sin\theta}{\theta^3} [\boldsymbol{\omega}]_\times^3 \\ \mathbf{v} &= \mathbf{V}^{-1}\mathbf{t} \\ \ln(\mathbf{T}) &= \begin{pmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{pmatrix} \in \mathbb{R}^6. \end{aligned} \tag{2.7}$$

In order to make use of the Lie Group and Lie Algebra formulation in iterative optimization algorithms, it is important to find the derivatives of a 3D point transformed by a member of a Lie Group with respect to the parameters of the corresponding Lie Algebra.

The derivative of the rotation of a point $\mathbf{p}$ by $\mathbf{R} \in \mathrm{SO}(3)$ with respect to the parameters of the rotation $\boldsymbol{\omega} \in \mathrm{so}(3)$ can be derived [Ead08] as

$$\frac{\partial(\mathbf{R}\cdot\mathbf{p})}{\partial\boldsymbol{\omega}} = -[\mathbf{R}\cdot\mathbf{p}]_\times \in \mathbb{R}^{3\times 3}. \tag{2.8}$$

For the coordinate transformation of a 3D point $\mathbf{p}$ by $\mathbf{T} \in \mathrm{SE}(3)$, the derivative with respect to the parameters $\mathbf{x} = \left(\begin{smallmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{smallmatrix}\right) \in \mathrm{se}(3)$ is given [Ead08] as

$$\frac{\partial(\mathbf{T}\cdot\mathbf{p})}{\partial\mathbf{x}} = \left(\begin{array}{c|c} \mathbf{I} & -[\mathbf{T}\cdot\mathbf{p}]_\times \end{array}\right) \in \mathbb{R}^{3\times 6}. \tag{2.9}$$

Examples of how Lie Algebra is utilized in computer vision algorithms can be found in successive sections.

2.2.2 From 3D to 2D Points

A central piece of math for all algorithms trying to estimate the 3D structure of the environment are the transformations involved in the projection of 3D points in world coordinates into the image that the camera observes. This projection involves a series of transformations as follows:

1. Compute the 3D position of a feature $\mathbf{p}_c$ in the camera coordinate system of the current camera pose estimate ${}^c\mathbf{T}_w$ from the 3D position of the feature $\mathbf{p}_w$ in world coordinates:
$$\mathbf{p}_c = {}^c\mathbf{T}_w \, \mathbf{p}_w. \tag{2.10}$$
If the position of the features are not given in world coordinates but in some other camera coordinate system ${}^r\mathbf{T}_w$, it is always possible to obtain $\mathbf{p}_c$ by transforming into the world coordinate system first:
$$\mathbf{p}_c = {}^c\mathbf{T}_w \, {}^r\mathbf{T}_w^{-1} \, \mathbf{p}_r. \tag{2.11}$$

2. Project the 3D position of the feature $\mathbf{p}_c = (X, Y, Z)$ in the camera coordinate system into camera plane coordinates:
$$\mathbf{p}_{\text{cp}} = \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} X \\ Y \end{pmatrix}. \tag{2.12}$$

3. Apply a distortion model like for example polynomial or field of view (FOV) distortion model [DF01], depending on the amount of distortion induced by the camera lens. The second order polynomial distortion model, which works well with low distortion lenses, is defined as:
$$\mathbf{p}_{\,\text{polynomial}} = \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} = \left(1 + k_0 r^2 + k_1 r^4\right) \, \mathbf{p}_{\text{cp}}, \tag{2.13}$$
where k_0 and k_1 are the coefficients of the polynomial distortion model and $r = \sqrt{\tilde{u}^2 + \tilde{v}^2}$. The FOV distortion model can be expressed using the same definition of r and one distortion parameter k_{FOV}:
$$\mathbf{p}_{\,\text{arctan}} = \begin{pmatrix} \bar{u} \\ \bar{v} \end{pmatrix} = \frac{1}{r k_{FOV}} \arctan\left(2r \tan\frac{k_{FOV}}{2}\right) \, \mathbf{p}_{\text{cp}}. \tag{2.14}$$

4. Project the feature position in the camera plane into the image plane using the camera calibration
$$\mathbf{z} = \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \bar{u} f_u + c_u \\ \bar{v} f_v + c_v \end{pmatrix} \tag{2.15}$$
where f_u and f_v are the focal lengths into the u and v direction and $c = \begin{pmatrix} c_u \\ c_v \end{pmatrix}$ is the optical center of the camera.

The distortion parameters, the image center and the focal lengths are commonly referred to as intrinsic camera parameters.

In the following, the transformation sequence performing the projection of a 3D point $\mathbf{p}_w$ into the image plane of the camera will be denoted by the function $\mathbf{h}(\mathbf{p}_w, \mathbf{x})$. $\mathbf{x}$ is a vector of parameters that characterizes the projection and that are variable within for example an optimization algorithm. Usually $\mathbf{x}$ will hold the 6 parameters of the camera pose in $\mathrm{se}(3)$. The parameters of the intrinsic camera calibration are obtained once in a separate camera calibration program and then left fixed.

2.2.3 The Levenberg Marquardt Algorithm

The Levenberg Marquardt (LM) algorithm is an iterative optimization algorithm for non-linear functions that optimally interpolates between the steepest descent method and the Gaus-Newton optimization algorithm [Mar63]. The algorithm thus shows more robust convergence than the Gaus-Newton algorithm and converges faster than the deepest descent.

The LM algorithm is designed to find a locally optimal solution to the weighted least squares problem. This is the problem of minimizing or maximizing the following sum of weighted squared errors with respect to a parameter vector $\mathbf{x}$:

$$\arg\min_{\mathbf{x}} E(\mathbf{x}) = \arg\min_{\mathbf{x}} ||\Phi(\mathbf{x}) - \hat{\Phi}||^2_{\mathbf{W}} = \arg\min_{\mathbf{x}} \left(\Phi(\mathbf{x}) - \hat{\Phi}\right)^T \mathbf{W} \left(\Phi(\mathbf{x}) - \hat{\Phi}\right) \quad (2.16)$$

where $E(\mathbf{x})$ is the quadratic energy or cost function, $\mathbf{W}$ is the weighting matrix and $\Phi(\mathbf{x})$ is a possibly non-linear function that predicts the observations $\hat{\Phi}$ using the parameters $\mathbf{x}$. This means that the LM algorithm can in principal be used to find (locally) optimal parameters for any continuous and differentiable function $\Phi(\mathbf{x})$.

In 3D computer vision the algorithm is mostly used to determine or refine a subset or all parameters involved in the projection process described in previous Section (2.2.2). In the camera calibration process, for example, the intrinsic camera parameters as well as the extrinsic camera parameters namely the pose of the camera relative to calibration pattern can be determined using the LM algorithm. In PTAM and other visual SLAM systems, the intrinsic camera parameters are assumed to be known from a preceding camera calibration process. The two main areas of deployment are:

- Bundle Adjustment: estimation of the 3D structure of the environment and the camera poses from which the scene is observed [TMHF00]. For this task, the parameter vector $\mathbf{x}$ comprises the 3D positions of all features and the pose parameters of all cameras observing the scene.
- Camera pose refinement: given correspondences between 3D feature positions and their observations in the image, update the camera pose such that the projection error of the features is minimized. In this case the parameter vector $\mathbf{x}$ describes the pose of the camera.

The following derivation [HZ04] for the update equation for the LM algorithm is performed for the application in 3D computer vision. Using the projection function $\mathbf{h}(\mathbf{p}_w, \mathbf{x})$ as previously defined, the quadratic cost function becomes:

$$E(\mathbf{x}) = \frac{1}{2}\sum_i w_i(\mathbf{z}_i - \mathbf{h}(\mathbf{p}_i^w, \mathbf{x}))^T(\mathbf{z}_i - \mathbf{h}(\mathbf{p}_i^w, \mathbf{x})) = \frac{1}{2}\sum_i w_i \mathbf{e}_i^T \mathbf{e}_i, \tag{2.17}$$

where $\mathbf{p}_i^w$ are the 3D points corresponding to the observations $\mathbf{z}_i$ in the image and w_i are the weights for each individual observation.

In order to make the derivation more compact, the observations, the predicted positions and errors are combined into single large vectors:

$$E(\mathbf{x}) = \frac{1}{2}(\mathbf{z} - \mathbf{h}(\mathbf{x}))^T \mathbf{W}(\mathbf{z} - \mathbf{h}(\mathbf{x})) = \frac{1}{2}\mathbf{e}(\mathbf{x})^T \mathbf{W}\, \mathbf{e}(\mathbf{x}), \tag{2.18}$$

where $\mathbf{W}$ is a square matrix with the individual weightings on its diagonal.

To find an update $\Delta\mathbf{x}$ to the parameter vector $\mathbf{x}$, the Taylor series expansion of E around the state $\mathbf{x} + \Delta\mathbf{x}$ up to the second order terms is derived:

$$\begin{aligned} E(\mathbf{x} + \Delta\mathbf{x}) &\approx E(\mathbf{x}) + \tfrac{\partial E}{\partial \mathbf{x}}\Delta\mathbf{x} + \tfrac{1}{2}\Delta\mathbf{x}^T \tfrac{\partial^2 E}{\partial \mathbf{x}^2}\Delta\mathbf{x} \\ &= E(\mathbf{x}) + \mathbf{e}(\mathbf{x})^T\mathbf{W}\mathbf{J}\Delta\mathbf{x} + \tfrac{1}{2}\Delta\mathbf{x}^T\left(\mathbf{J}^T\mathbf{W}\mathbf{J} + \mathbf{e}(\mathbf{x})^T\mathbf{W}\tfrac{\partial^2 \mathbf{e}}{\partial \mathbf{x}^2}\right)\Delta\mathbf{x}, \end{aligned} \tag{2.19}$$

where $\mathbf{J} = \left.\frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}} = -\left.\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}}$.

Next, the Gauss-Newton approximation that $\mathbf{e}(x)^T\mathbf{W}\frac{\partial^2 \mathbf{e}}{\partial \mathbf{x}^2} \approx 0$ is applied and the derivative of the second order approximation with respect to $\Delta\mathbf{x}$ is set to zero to find the optimal update step $\Delta\mathbf{x}$:

$$\frac{\partial E}{\partial \Delta\mathbf{x}} = \mathbf{J}^T\mathbf{W}\mathbf{e}(\mathbf{x}) + \mathbf{J}^T\mathbf{W}\mathbf{J}\Delta\mathbf{x} = 0 \Longrightarrow \Delta\mathbf{x} = -\left(\mathbf{J}^T\mathbf{W}\mathbf{J}\right)^{-1}\mathbf{J}^T\mathbf{W}\mathbf{e}(\mathbf{x}). \tag{2.20}$$

Computing $\Delta\mathbf{x}$ according to Equation (2.20) gives the Gauss-Newton Minimization Algorithm. In the LM Algorithm the diagonal entries of $\mathbf{J}^T\mathbf{W}\mathbf{J}$ are multiplied by a variable factor $\lambda + 1$. Hence we get the update equation:

$$\Delta\mathbf{x} = -\left(\mathbf{J}^T\mathbf{W}\mathbf{J} + \lambda\,\mathbf{I}\right)^{-1}\mathbf{J}^T\mathbf{W}\mathbf{e}(\mathbf{x}), \tag{2.21}$$

where $\mathbf{I}$ is the identity matrix. For low values of λ the LM update Equation (2.21) becomes the Gaus-Newton update Equation (2.20). If λ takes on high values, Equation (2.21) approximates $\Delta\mathbf{x} = -\frac{1}{\lambda}\mathbf{J}^T\mathbf{W}\mathbf{e}(\mathbf{x})$ which resembles a step in the direction of steepest descent.

Going back to the original formulation using sums, Equation (2.21) can be re-written as

$$\Delta\mathbf{x} = -\sum_i w_i \left(w_i\mathbf{J}_i^T\mathbf{J}_i + \lambda\mathbf{I}\right)^{-1}\mathbf{J}_i^T\left(\mathbf{z}_i - \mathbf{h}(\mathbf{p}_i^w, \mathbf{x})\right), \tag{2.22}$$

where $\mathbf{J}_i = -\frac{\partial \mathbf{h}(\mathbf{p}_i^w, \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{p}_i^w, \mathbf{x}}$.

Starting from $\lambda = 10^{-4}$ the LM algorithm computes and incorporates steps $\Delta\mathbf{x}$ and checks at each iteration whether the cost E was reduced. If E decreased the step is accepted and λ is divided by 2. Otherwise the step is discarded and λ increased by a factor of 10. This way the LM Algorithm is dynamically mixing Gradient Descent and Gauss-Newton Minimization [HZ04].

As already mentioned, in Computer Vision the LM algorithm is often used to estimate rotations or poses of a camera. In order to do this efficiently, the LM algorithm operates solely on the respective Lie Algebras $\mathfrak{so}(3)$ or $\mathfrak{se}(3)$. Updates $\Delta\mathbf{x}$ can be incorporated into the rotation or pose estimate using the exponential map from the Lie Algebra to the respective Lie Group.

2.2.4 Robust Parameter Estimation using M-Estimators

The insight which motivates M-Estimators is that instead of using a sum of squared errors cost function any other function of the errors than the squaring them can be used inside the sum. This is important since the squared error function is instable in the presence of outliers. One outlier that is far enough away from the inliers of the true model can corrupt the solution completely.

Thus Equation (2.17) can be reformulated in a more general form:

$$E(\mathbf{x}) = \sum_i \varrho(\mathbf{e}_i), \tag{2.23}$$

where $\mathbf{e}_i$ are the deviations of the model from the observations and $\varrho(\mathbf{e}_i)$ is a function of the errors which determines their contribution to the overall cost function.

In general Equation (2.23) is minimized for

$$\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} = \sum_i \frac{\partial \varrho(\mathbf{e}_i)}{\partial \mathbf{e}_i} \frac{\partial \mathbf{e}_i}{\partial \mathbf{x}} = 0. \tag{2.24}$$

Now let $\frac{\partial \varrho(\mathbf{e}_i)}{\partial \mathbf{e}_i} = \psi(\mathbf{e}_i)$ and define the weighting function

$$w(\mathbf{e}_i) = \frac{\psi(\mathbf{e}_i)}{\mathbf{e}_i} = \frac{1}{\mathbf{e}_i} \frac{\partial \varrho(\mathbf{e}_i)}{\partial \mathbf{e}_i}, \tag{2.25}$$

where $\psi(\mathbf{e}_i)$ is called the influence function. With these definitions Equation (2.24) becomes

$$\sum_i w(\mathbf{e}_i) \mathbf{e}_i \frac{\partial \mathbf{e}_i}{\partial \mathbf{x}} = 0. \tag{2.26}$$

This is the same criterion for a minimum as if the problem is solved iteratively using a weighted least squares cost function:

$$E(\mathbf{x}) = \sum_i w(\mathbf{e}_i^{k-1})(\mathbf{e}_i^k)^2, \tag{2.27}$$

where the weights w_i depend on the errors of the previous $\mathbf{e}_i^{k-1}$ iteration. To show this, the derivative of Equation (2.27) with respect to the parameters $\mathbf{x}$ is computed and the result is set to zero:

$$\frac{\partial E(\mathbf{x})}{\partial \mathbf{x}} = \sum_i w(\mathbf{e}_i^{k-1}) \frac{\partial (\mathbf{e}_i^k)^2}{\partial \mathbf{x}} = 2 \sum_i w(\mathbf{e}_i^{k-1}) \mathbf{e}_i \frac{\partial \mathbf{e}_i^k}{\partial \mathbf{x}} = 0. \tag{2.28}$$

This is the same criterion function for minimality as derived in Equation (2.26).

This means that using an iterative re-weighting least squares with a weighting according to Equation (2.25) is equivalent to minimizing a cost function as given in Equation (2.23). This provides the freedom to choose $\varrho(\mathbf{e}_i)$ such that large errors caused by outliers are attenuated in order to render the parameter estimation robust to outlier data.

There are many different weighting functions [HZ04, Zha96] which aim to make the parameter estimation more robust. The most common ones are the Tukey, Cauchy and Huber weighting functions.

The weighting is based on a standard normal distribution of errors $\tilde{\mathbf{e}}_i$. That means the errors are normalized by the standard deviation σ_e of the error distribution.

$$\tilde{\mathbf{e}}_i = \frac{\mathbf{e}_i}{\sigma_e} \tag{2.29}$$

Using this definition, the Tukey weighting function is defined as:

$$w(\tilde{\mathbf{e}}_i) = \begin{cases} \left(1 - \frac{\tilde{\mathbf{e}}_i^2}{c^2}\right)^2 & \text{if } |\tilde{\mathbf{e}}_i| \leq c \\ 0 & \text{if } |\tilde{\mathbf{e}}_i| > c \end{cases}, \tag{2.30}$$

where $c = 1.2107$ again to achieve 95% asymptotic efficiency on the standard normal distribution [Zha96].

Similarly, the Huber function is given as:

$$w(\tilde{\mathbf{e}}_i) = \begin{cases} 1 & \text{if } |\tilde{\mathbf{e}}_i| \leq c \\ \frac{c}{|\tilde{\mathbf{e}}_i|} & \text{if } |\tilde{\mathbf{e}}_i| > c \end{cases}, \tag{2.31}$$

where $c = 1.345$ to achieve 95% asymptotic efficiency on the standard normal distribution [Zha96].

And finally, the weighting according to Cauchy is defined as:

$$w(\tilde{\mathbf{e}}_i) = \frac{c^2}{c^2 + \tilde{\mathbf{e}}_i^2} \tag{2.32}$$

where $c = 2.3849$ to achieve 95% asymptotic efficiency on the standard normal distribution [Zha96].

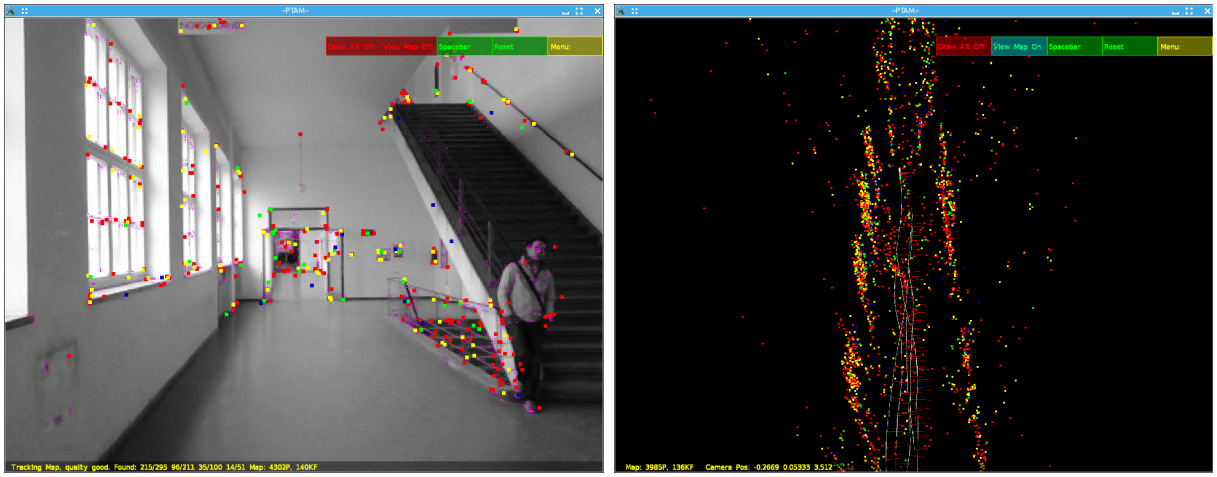
A common approach is to start out with a weighting function that does not fully ignore gross outliers like the Cauchy or the Huber weighting functions. After a small number of iterations, the weighting function is switched to the Tukey weighting function which simply discards gross outliers. This approach gives datapoints with high errors the chance to draw the model into their direction before the ones that still have high errors are completely ignored.

2.3 Parallel Tracking and Mapping (PTAM)

The Parallel Tracking and Mapping (PTAM) algorithm developed by Klein and Murray [KM07, KM09], is an efficient algorithm to perform monocular Simultaneous Localization and Mapping (SLAM). Monocular SLAM is the problem of estimating the pose of a single camera and the 3D structure of the environment at the same time. The conventional SLAM approach [DRMS07] updates the estimate of the camera pose and the structure of the environment at the same time whenever a new observation is obtained. In contrast to that PTAM splits up the task of keeping track of the pose of the camera and the mapping task into two threads: the tracking and the mapping thread.

The mapping thread estimates the 3D structure of the environment using the information about 2D-3D point correspondences supplied by the tracking thread. The tracking thread in turn uses the knowledge of 3D feature positions to find 2D-3D point correspondences which are used for camera pose tracking.

PTAMs source code is readily available from Kleins homepage [Kle08] for use in research. Since the main topic of this thesis was tho improve relocalization and to allow for global localization in buildings, the original PTAM code was adopted and extended. This chapter serves as an introduction to the concepts used in PTAM and to the algorithm itself. The relocalization algorithm of PTAM is described in depth and theoretical drawbacks are discussed for an application of this relocalization algorithm when PTAM is used as a visual odometry system. This motivates the consecutive chapter which deals in depth with feature-based relocalization. For an exhaustive description of all details of the algorithm the reader is referred to the original papers of Klein and Murray [KM07, KM09].



(a) Camera Image with tracked Features (b) Feature Point-Cloud and Keyframe Poses

Figure 2.1: The PTAM implementation of Klein [Kle08] allows to switch between a view of the camera image with the tracked features and a view of the estimated 3D feature point-cloud.

2.3.1 Tracking Thread

The tracking thread is responsible for (1) keeping track of the data association of observed features with their 3D positions and (2) for updating the pose estimate of the camera at every new frame. This is achieved by two main components that are executed for each frame: first, the algorithm searches for features at predicted locations in the current frame to establish 2D-3D point correspondences. Second, the pose estimate is refined using 10 iterations of a robust LM algorithm to minimize the projection errors of the features that were re-observed in the current frame.

These two steps are executed two times: first for a small set (50) of features at the highest image pyramid level and then for the full set of features on all pyramid levels. This coarse to fine approach ensures that the final pose refinement using all features has a good starting pose for the LM algorithm and hence converges faster.

PTAM uses the FAST [RD05] keypoint detector to find features in the incoming frames. Only keypoints with a high Shi-Thomasi score [ST94] are considered good feature points and hence used for tracking. The image patch around a keypoint serves as the feature descriptor.

Establishing 2D to 3D Point Correspondences

The frame-to-frame tracking is necessary in order to keep track of the so called data association, the correspondence of an observed feature point with its 3D position. In

order to find the corresponding 3D feature points to the features observed in the current frame, the pose at the current frame is predicted using a rotation estimate obtained from Efficient Second Order Minimization (ESM) [BM04] on the small blurred current and previous frame. Zero translation is assumed.

$$ {}^{w}\tilde{\mathbf{T}}_{c} = \Delta\mathbf{T}\,{}^{w}\mathbf{T}_{c} \tag{2.33} $$

The predicted pose ${}^{w}\tilde{\mathbf{T}}_{c}$ is then used to compute assumed feature locations in the current image from all 3D feature points in the map as outlined in Section (2.2.2).

For all predicted points in the image plane the best matching image patch is found among all close-by FAST corners. The cut-off proximity for the image patch search is 10 pixels for fine and 20 pixels for coarse pose refinement. As the similarity measure between the 8*8 patch candidates in the current frame I_c and the warped predicted patch I_w Zero Mean Sum Squared Differences (ZMSSD) is used.

$$ \mathbf{ZMSSD}(I_c, I_w) = \sum_i (I_c(i) - I_w(i) - \bar{I}_c - \bar{I}_w)^2, \tag{2.34} $$

where $\bar{I}_c$ and $\bar{I}_w$ are the mean patch intensities computed as $\bar{I} = \frac{1}{N}\sum_i I(i)$.

The feature with the most similar warped image patch is assumed to be the observation of the 3D feature in the current frame.

Pose Updates

Pose updates are computed by minimizing the difference between the observed and the predicted feature positions with respect to the estimated camera pose. This minimization problem can be formulated as a weighted Least Squares optimization:

$$ E(\mathbf{x}) = \frac{1}{2}\sum_i w_i(\mathbf{z}_i - \mathbf{h}(\mathbf{p}_i^w, \mathbf{x}))^T(\mathbf{z}_i - \mathbf{h}(\mathbf{p}_i^w, \mathbf{x})) = \frac{1}{2}\sum_i w_i \mathbf{e}_i^T \mathbf{e}_i, \tag{2.35} $$

where the errors $\mathbf{e}_i$ are defined as the deviation of an observed feature position $\mathbf{z}_i$ in the image from the prediction of the feature position $\mathbf{h}(\mathbf{p}_i^w, \mathbf{x})$ given the $\mathrm{se}(3)$ parameters $\mathbf{x}$ of the camera pose. The individual error's contribution to the overall error is weighted by w_i.

The LM algorithm with the robust Tukey weighting function as introduced in Sections (2.2.3) and (2.2.4) is deployed to update the camera pose such that the weighted error is minimized. The updates $\Delta\mathbf{x} \in \mathrm{se}(3)$ to the camera pose computed by the LM algorithm (see Equation (2.22)) are incorporated into the camera pose using the exponential map from Equation (2.6):

$$ {}^{c}\mathbf{T}_{w} = \exp(\Delta\mathbf{x}) \cdot {}^{c}\mathbf{T}_{w}. \tag{2.36} $$

The PTAM implementation iterates the optimization algorithm ten times at each frame to efficiently keep track of the pose of the camera.

2.3.2 Mapping Thread

The mapping thread estimates the 3D structure of the environment using the information about 3D-2D point correspondences supplied by the tracking thread.

The map in PTAM is represented as a set of keyframes and a set of 3D points. The keyframes hold the pose from which the frame was seen as well as all observations of 3D feature points in that frame. These observations constrain the poses of the keyframes and the 3D point positions. A sparse LM algorithm [HZ04] is used to find an optimal set of camera poses and 3D points. In Computer Vision this problem is often called Bundle Adjustment [TMHF00].

To initialize the map, PTAM extracts features in an initial frame and tracks those frame-to-frame until enough parallax is achieved. These 2D-2D correspondences are then used to first estimate the transformation between the initial and the current frame. Given this transformation the depth of each feature point can be estimated using triangulation [Fri33]. These depth estimates are used to initialize the 3D positions of feature points. PTAM keeps these two frames as the first two keyframes. The pose of the first keyframe is fixed to the origin of PTAMs internal coordinate system.

After the initialization from two different views of one scene, PTAM keeps adding keyframes to the map whenever the number of re-observed 3D features falls below a certain threshold and no other keyframe is nearby. This way the map is enlarged whenever the camera views new scenes. Running the Bundle Adjustment whenever new keyframes are added ensures global consistency of the map.

2.3.3 Keyframe-based Relocalization Algorithm

PTAMs relocalization algorithm makes use of the knowledge of all keyframe poses obtained by the Bundle Adjustment performed in the mapping thread [KM08]. Once tracking is lost every new frame is scaled down to 40×30 pixels, the mean image intensity is subtracted and the resulting image is blurred with a Gaussian kernel with standard deviation of 2.5 pixels. Blurring is performed efficiently using integral images. The same operations are used to obtain scaled down, zero mean and blurred images of the keyframes.

After these preparations, an exhaustive search over all keyframes is conducted to find the keyframe that looks most similar to the current frame. The camera pose estimate is set to the pose ${}^{w}\mathbf{T}_{c}^{*}$ of the best matching keyframe. The similarity measure used in this Nearest Neighbor search is Zero Mean Sum of Squared Differences (ZMSSD). Since all the small images are already zero mean, the ZMSSD between the current image I_q and a keyframe I_{kF} can be computed as follows:

$$\mathbf{ZMSSD}(I_q, I_{kf}) = \sum_i (I_q(i) - I_{kf}(i))^2, \tag{2.37}$$

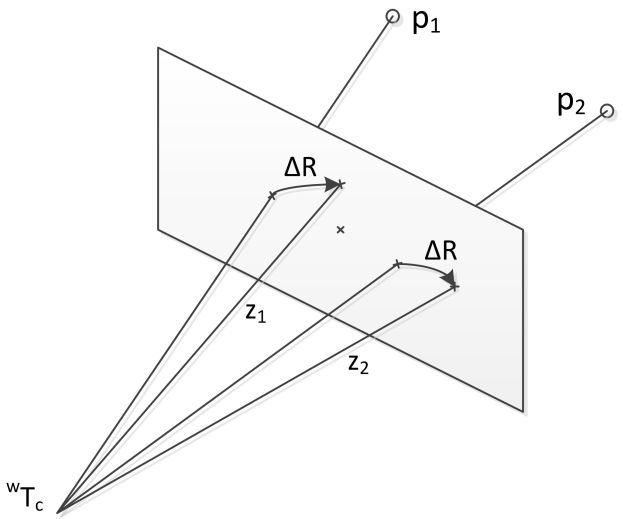


Figure 2.2: Geometry of the rotation refinement in PTAMs keyframe-based relocalization algorithm. The rotation $\Delta\mathbf{R}$ is found such that the projections of the 3D points $p_{1,2}$ are as close as possible to $z_{1,2}$.

where i iterates over all pixels in the images.

In a second step the orientation of the camera is refined. Using Efficient Second Order Minimization (ESM) [BM04] the transformation $T_{img} \in \mathrm{SE}(2)$is determined which best aligns the current frame with the small blurred keyframe. This $\mathrm{SE}(2)$ transformation is then used to estimate the difference in rotation $\Delta\mathbf{R} \in \mathrm{SO}(3)$ between the unknown current pose and the pose of the most similar keyframe ${}^{w}\mathbf{T}_{c}^{*}$.

The rotation $\Delta\mathbf{R}$ is iteratively computed by an LM algorithm in the following way: First, two points $\left(\begin{smallmatrix}5\\0\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}-5\\0\end{smallmatrix}\right)$ in the image coordinate system are transformed by $\mathbf{T}_{img}$ at the image center $c_{img} = \begin{pmatrix} w_{img}/2 \\ h_{img}/2 \end{pmatrix} = \left(\begin{smallmatrix}20\\15\end{smallmatrix}\right)$:

$$\mathbf{z}_1 = c_{img} + \mathbf{T}_{img} \begin{pmatrix} 5 \\ 0 \end{pmatrix} \text{ and } \mathbf{z}_2 = c_{img} + \mathbf{T}_{img} \begin{pmatrix} -5 \\ 0 \end{pmatrix}. \tag{2.38}$$

The same points $\left(\begin{smallmatrix}5\\0\end{smallmatrix}\right)$ and $\left(\begin{smallmatrix}-5\\0\end{smallmatrix}\right)$ translated to the image center c_{img} are undistorted and unprojected using the FOV camera model (Section (2.2.2)) to obtain 3D points $\mathbf{p}_1$ and $\mathbf{p}_2$ which were not transformed by $\mathbf{T}_{img}$. The depth of $\mathbf{p}_1$ and $\mathbf{p}_2$ can be chosen arbitrarily since only a rotation is estimated. In the PTAM implementation, the depth is set to one. A LM algorithm is used to find the rotation $\Delta\mathbf{R}$ under which the projection error of the rotated 3D points into the image becomes minimal:

$$\Delta\mathbf{R} = \arg\min_{\Delta\mathbf{R}} \sum_{i=1}^{2} \left(\mathbf{z}_i - \mathbf{h}(\Delta\mathbf{R}\mathbf{p}_i)\right)^2, \tag{2.39}$$

where $\mathbf{h}(\cdot)$ denotes the projection of a 3D point into the image as defined in Section (2.2.2). The geometry of this problem is depicted in Figure (2.2). The rotation of the pose estimate ${}^{w}\mathbf{T}_{c}^{*}$ is updated by $\Delta\mathbf{R}$ before tracking is restarted from this pose.

In conclusion, PTAM relocalization strategy is optimized for pose recovery in the proximity of already seen keyframes since it assigns the position of the most similar keyframe and only refines the rotation. This is sufficient for Augmented Reality applications in a small confined workspace since a keyframe can always be assumed to be close by. For an indoor localization scenario however, this clearly limits the situations in which relocalization is possible. For example in the standard scenario, when the user is walking straight forward, PTAM may not be able to recover from tracking loss since the camera moves away from the last keyframe.

Chapter 3

Relocalization using Binary Features

In the previous chapter, drawbacks of PTAMs original relocalization algorithm for indoor localization were discussed. Due to the limited computational and battery power of handheld devices, an efficient relocalization algorithm is of paramount importance. For this reason the objective was to utilize binary feature descriptors which are very efficient to extract. Additionally, distance computations can be performed rapidly in the Hamming space.

After examining related work in feature-based relocalization and binary features in Section (3.1), the Binary Robust Features (BRIEF) [CLSF10] descriptor is introduced in Section (3.2). Choosing to utilize binary features calls for different search strategies to find matching features for relocalization. Such suitable search strategies are described in Section (3.3). Finally, the algorithms necessary for pose recovery are described in Section (3.4) before a binary-feature-based algorithm for relocalization is introduced in Section (3.5).

3.1 Related Work

At first, related work in small scale localization is reviewed. Both, frame-based and feature-based relocalization strategies are examined. Since the aim is to utilize binary features for relocalization, an overview over existing binary feature descriptors is given in the second part of this section.

3.1.1 Relocalization Strategies

Small scale relocalization within a visual odometry system can roughly be performed in two ways: (1) frame-based or (2) feature-based. Former method does require that the map somehow stores the position and the image of a set of frames like for example the

keyframes in PTAM. Later method necessitates that feature positions are estimated in 3D and that feature descriptors are remembered by the algorithm.

Among the earliest work for frame-based visual localization is the system by Dellaert et al. [DBFT99] which utilizes a Monte Carlo Localization (MCL) algorithm. Their system uses a visual map of the ceiling to gradually localize a robot in a building. Since the algorithm is a pure localization algorithm, the map has to be supplied to the system and is not refined or updated by it.

A frame-based method proposed by Klein and Murray [KM08] was already introduced as the relocalization algorithm of PTAM in Section (2.3.3). The algorithm uses down-sampled images of keyframes (40×30 pixels) to find the most similar keyframe to the current image in the ZMSSD sense. While this method is extremely fast (1.5 ms for pose estimation within 250 keyframes), it can only localize to previously visited places where keyframes were added. In a similar vein, Reitmayr et al. [RD06] were using frames observed from estimated poses to restart their model tracking algorithm after tracking loss.

Among the earliest work for feature-based localization in a SLAM setting is the work by Se et al. [SLL05]. They utilize SIFT features [Low99] combined with RANSAC to localize a robot in a 2D map to solve the kidnapped robot problem. This is the problem of localizing a robot without any prior knowledge of its position.

Early work on feature-based relocalization in a pure visual SLAM setting can be found in [CPMCC06], where visual indexing of feature image patches is utilized to allow robust feature matching. These matches are used to recover the camera pose by deploying a 3-point algorithm combined with RANSAC.

Williams et al. [WSR07] propose a feature-based pose recovery system for the monocular SLAM system by Davis et al. [Dav03, DRMS07]. Equal to Davis' MonoSLAM algorithm, they utilize image points with high Shi-Thomasi score [ST94] as features. The algorithm finds feature matches using the correlation between keypoint image patches in the map and in the frame for relocalization. Those putative feature matches are fed into a RANSAC algorithm utilizing the 3-point algorithm for pose estimation.

In a different paper, Williams et al. [WKR07] propose a feature matching algorithm which utilizes randomized trees for fast feature matching. Each branch in the proposed randomized tree amounts to a comparison of image intensities at random image positions. In order to make their feature retrieval system robust to viewpoint changes, they train the randomized trees with 400 warped patches of newly added features. Again RANSAC is used to robustly determine a pose from a set of matched features. All in all the system performs in real-time for maps of 80 features.

Wagner et al. [WRM+08] compare two algorithms for feature-based visual localization relative to a planar object on mobile devices. On the one hand SIFT features with a forest of spill trees [LMGY04], a technique for fast approximate NN search, are utilized for feature matching. On the other hand FERNs [OFL07, OCLF10] are deployed to obtain

putative matches. FERNs use random binary tests on a feature image patch to classify the feature as belonging to one of several classes which have to be trained beforehand. Modifications to both algorithms were implemented in order to allow deployment on the cellphones. Using the matched features and assuming planarity of the scene, they estimate a homography using MLESAC [TZ00] a variant of the RANSAC algorithm. From the homography the relative transformation to the planar object is recovered. They present detailed timing evaluations on several mobile phones with 320×240 pixels cameras showing average frame rates of 15 fps for both localization algorithms. The match against around 200 to 400 SIFT features and 100 FERNs classes.

Arth et al. [AWK+09] also evaluate the performance of feature-based localization on a cellphone. They divide their environment into potentially visible sets (PVS) which are downloaded from a server as necessary. The SURF-like features are matched using exhaustive nearest neighbor search in close-by PVSs. The 3-point pose algorithm within a RANSAC is used to obtain a pose estimate. This estimate is refined using a robust Gaus-Newton algorithm.

3.1.2 Binary Features

Binary feature descriptors have gained considerable attention after the BRIEF feature descriptor by Calonder [CLSF10] exhibited similar performance for small rotation and scale differences compared to the SURF descriptor by Bay et al. [BTVG06], the quasi standard feature descriptor in computer vision. The big advantage of BRIEF over SURF is its up to 40 times faster extraction time and lower memory requirement. Additionally the distance computations between BRIEF features are faster since the Hamming distance can be used instead of the Euclidean distance.

These convincing results sparked a series of other binary feature descriptors. Among them is ORB [RRKB11] which adds rotation invariance to the binary descriptor. The rotation is estimated using the intensity centroid method [Ros99]. Image intensity comparison patterns are learned which when used in conjunction with the rotation corrected image patches are less correlated than the ones utilized by BRIEF.

BRISK [LCS11] is also a binary feature descriptor. In contrast to BRIEF and ORB, the image sampling positions are not drawn randomly anymore. Besides taking the rotation of a feature point into account, BRISK utilizes scale space theory to adapt the sampling pattern to the maximum in scale space. Thus, BRISK is rotation and scale invariant.

The latest of the binary feature descriptors is FREAK [AOV12] which draws the inspiration for the image intensity comparison pattern from the spatial organization of the human retina. The evaluation of FREAK suggests that the descriptor is rotation, scale and brightness invariant. According to the evaluation in their paper FREAK outperforms BRIEF, ORB, BRISK, SURF and SIFT [Low99] in all aspects.

All previously mentioned feature descriptors need a keypoint detector which tells the algorithm where to extract the descriptors. Since the main intention of using binary feature descriptors is speed, these descriptors are usually combined with the FAST [RD05] keypoint detector which, as its name suggests, is a very efficient corner detector. More resent binary feature descriptors, namely BRISK and FREAK, rely on an improved version of FAST the so called AGAST [MHB+10] keypoint detector. AGAST improves the order of image intensity comparisons of FAST to obtain a generic detector which is not trained on some specific environment. AGAST does detect the same corner locations as FAST.

3.2 Binary Robust Features (BRIEF)

In this work, the most basic variant of binary features namely Binary Robust Features (BRIEF) by Calonder et al. [CLSF10] is used. The BRIEF descriptor is the least complex binary descriptor and hence fastest to extract. This is because BRIEF is neither scale nor rotation invariant. The implications of these properties will be discussed in the sections of the localization algorithms. To detect where to extract BRIEF descriptors, the FAST keypoint detector [RD05] is deployed. FAST was chosen instead of AGAST since FAST is already integrated into the PTAM implementation. This might affect extraction speed but will not alter keypoint locations since they both detect the same corner points.

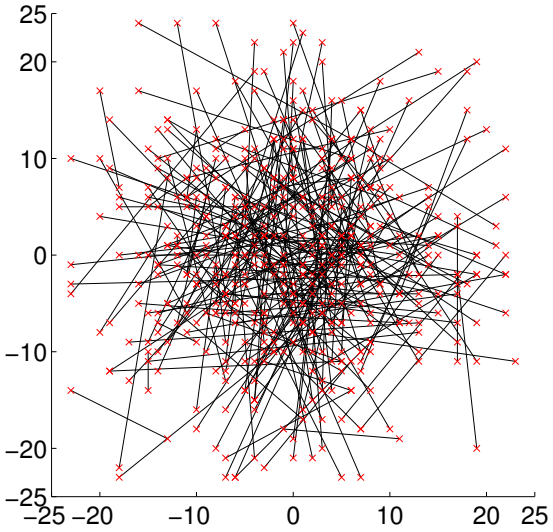


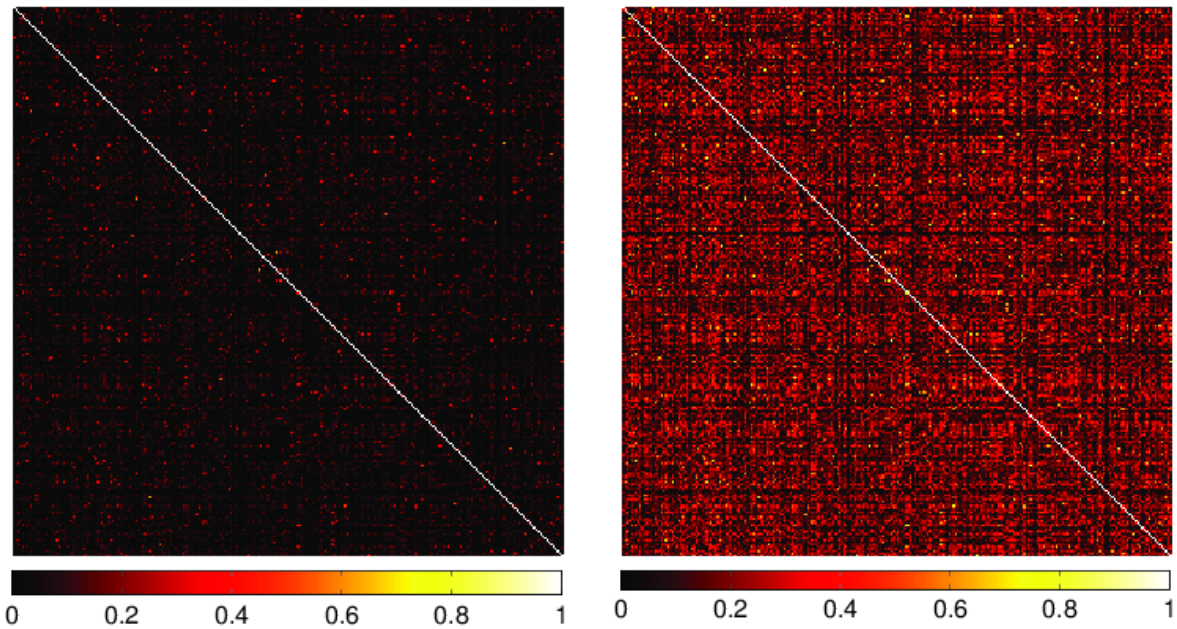
Figure 3.1: The original image intensity comparison pattern employed by OpenCV for BRIEF descriptor extraction.

The BRIEF descriptor bit string $\mathrm{BD} = (\mathrm{bit}_1, \ldots, \mathrm{bit}_M)$ is obtained by a multitude of image intensity comparisons around the feature position after smoothing the image:

$$\mathrm{bit}_i = \begin{cases} 1 & \textbf{if } I(p_{i1}) < I(p_{i2}) \\ 0 & \textbf{otherwise} \end{cases} \tag{3.1}$$

The positions for image intensity comparisons are sampled from the normal distribution $p_{i1,2} \sim N(0, \frac{S^2}{25})$ once at the beginning and are then fixed for all subsequent descriptor extractions. In the OpenCV implementation, the area in which the positions for the comparisons come to lie in is a square of 49×49 pixels. Figure (3.1) shows the image intensity comparison pattern used for BRIEF descriptor extraction in the OpenCV implementation.

While theoretically any number M of bits for the BRIEF descriptor would be possible, Calonder et. al. [CLSF10] found that $M = 256$ offers a good trade-off between recognition rates and storage efficiency. Therefore, in this work the 256 bit version of BRIEF called BRIEF32 from the OpenCV implementation is used. The smoothing is performed with a Gaussian kernel of size 9×9 pixels with a variance of 2. The OpenCV implementation performs the smoothing efficiently using integral images.



(a) Mutual Information (b) Absolute Correlation

Figure 3.2: Correlation and mutual information between bits in the BRIEF descriptors from a database of about 9M features.

The way the BRIEF descriptor is constructed results in an interesting property: since the positions are sampled independently from a normal distribution, bits in the descriptor are independent and their ordering is arbitrary. This means the BRIEF descriptor designates a point in 256 dimensional Hamming space. Hence, the Hamming distance can be used as a similarity measure between two BRIEF features. This distance measure is defined as the number of bits in two bit strings that differ.

The Hamming distance $||\mathrm{BD}_A, \mathrm{BD}_B||_H$ between two BRIEF bist strings can be computed efficiently in two steps: first, the bits that differ in the two bit strings are detected by performing an XOR between the bit-strings. Second, the number of ones in the resulting

bit string is counted. The bit counting can be performed by using a lookup table for 8 bit chunks of the BRIEF feature bit string. The lookup table consisting of 256 entries is constructed once at startup:

$$\mathrm{LUT}_{8\mathrm{bit}} = [0, 1, 1, 2, 1, 2, 2, 3, \cdots, 8]\,. \tag{3.2}$$

For every 8 bit number, it stores the amount of one-bits this bit string has. Thus, one-bit-counting can be done by summing up the number of ones of all 8 bit chunks of the M bit feature descriptor. The Hamming distance between two BRIEF features BD_A and BD_B is therefore given as:

$$||\mathrm{BD}_A, \mathrm{BD}_B||_H = \sum_{i=0}^{\lfloor M/8 \rfloor} \mathrm{LUT}_{8\mathrm{bit}}\left(\mathrm{byte}\left(\mathrm{BD}_A \oplus \mathrm{BD}_A\right)_i\right), \tag{3.3}$$

where $\mathrm{byte}\left(\cdot\right)_i$ extracts the ith byte form a bit-string and $\oplus$ denotes the XOR operator.

3.3 Nearest Neighbor Search for Binary Features

In contrast to distance metrics in continuous spaces, like the Euclidean or Manhattan distance, the Hamming distance is discrete:

$$||\mathrm{BD}_A, \mathrm{BD}_B||_H \in \{0, 1, \cdots, 256\} \tag{3.4}$$

For Nearest Neighbor searches in continuous spaces, using for example the Euclidean norm, it is nearly impossible that a query feature has the same distance to two features in the database. In contrast to that, the discrete Hamming distance makes it very likely that a query feature has the same distance to one or more features in the database.

The exhaustive NN search algorithm implemented for this thesis returns all matching features which have the same smallest Hamming distance to a query feature since the correct NN in the set of matches with equal distance cannot be determined.

Hashing-based techniques for approximate Nearest Neighbor search are commonly used for binary feature matching. More precisely Locality-Sensitive Hashing (LSH) [GIM99] can be used to search for binary features rapidly.

Recently, Muja and Lowe [ML12] proposed to use the FLANN library [ML09] to perform hierarchical clustering of a database of binary features to enable fast feature matching. They show that their algorithm outperforms LSH for very large datasets of 5M features. For smaller datasets in the order of 100k features both algorithms perform equally fast. Although they do not give the timing for preparing the hierarchical clustering, it is very likely that their algorithm takes longer to prepare than LSH. This makes it unsuitable for binary feature matching on a mobile device since hierarchical clustering of all map features would have to be performed at relocalization. Therefore, LSH will be used in this work.

3.3.1 Locality-Sensitive Hashing (LSH)

Locality-Sensitive Hashing (LSH) [IM98, GIM99] is a hashing-based technique to perform approximate Nearest Neighbor (aNN) search. The method is approximate in that there is neither a guarantee that any neighbor is found nor that a retrieved match is the nearest neighbor.

The main idea is to use a hash function that generates the same hash code for close-by feature descriptors. To be more precise, the hash function should generate the same hash codes for two BRIEF descriptors if their distance is less than r with a higher probability than if their Hamming distance is greater than r. Several hash tables $\mathcal{T}_i$ are used in parallel to increase the probability of finding the true nearest neighbor.

A locality-sensitive hash function g_i for BRIEF features composes the hash code from m randomly selected bits in the BRIEF descriptor bit strings $\mathrm{BD}_j = (\mathrm{bit}_1, \ldots, \mathrm{bit}_M)$:

$$g_i(\mathrm{BD}_j) = 0\mathrm{b}\,\langle \mathrm{bit}_{r_1} \rangle\, \langle \mathrm{bit}_{r_2} \rangle \ldots \langle \mathrm{bit}_{r_m} \rangle \in [0, \ldots, 2^m - 1]\,, \tag{3.5}$$

where 0b signals a binary number and the indices r_i are drawn uniformly and without repetition from $\{1, \ldots, N\}$ once at instantiation of the hash function.

In the following a proof will be given which shows that the hash function as defined in Equation (3.5) is indeed locality-sensitive. The necessary condition which has to be shown is that the probability of two BRIEF features BD_A and BD_B being closer than r given that they share the same hash code is smaller than the probability of them being closer than r given that they have different hash codes:

$$\begin{array}{l} P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \mid g_i(\mathrm{BD}_A) = g_i(\mathrm{BD}_B)\right) \overset{!}{>} \\ P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \mid g_i(\mathrm{BD}_A) \neq g_i(\mathrm{BD}_B)\right). \end{array} \tag{3.6}$$

In order to proof this first consider the probability of two BRIEF features having Hamming distance r:

$$\begin{array}{rcl} P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H = r\right) & = & P\left(\text{exactly } r \text{ bits in } \mathrm{BD}_A \text{ and } \mathrm{BD}_B \text{ are equal}\right) \\ & = & \binom{256}{r} P\left(\mathrm{bit}_{Ai} = \mathrm{bit}_{Bi}\right)^r P\left(\mathrm{bit}_{Aj} \neq \mathrm{bit}_{Bj}\right)^{256-r} \\ & = & \binom{256}{r} 0.5^r 0.5^{256-r} \\ & = & \binom{256}{r} 0.5^{256}, \end{array} \tag{3.7}$$

where the following identities were used:

$$\begin{array}{rcl} P\left(\mathrm{bit}_{Ai} = \mathrm{bit}_{Bi}\right) & = & P\left(\mathrm{bit}_{Ai} = 1 \cap \mathrm{bit}_{Bi} = 1\right) + P\left(\mathrm{bit}_{Ai} = 0 \cap \mathrm{bit}_{Bi} = 0\right) \\ & = & P\left(\mathrm{bit}_{Ai} = 1\right) \cdot P\left(\mathrm{bit}_{Bi} = 1\right) + P\left(\mathrm{bit}_{Ai} = 0\right) \cdot P\left(\mathrm{bit}_{Bi} = 0\right) \\ & = & 0.5^2 + 0.5^2 = 0.5 \\ P\left(\mathrm{bit}_{Aj} \neq \mathrm{bit}_{Bj}\right) & = & P\left(\mathrm{bit}_{Aj} = 1 \cap \mathrm{bit}_{Bj} = 0\right) + P\left(\mathrm{bit}_{Aj} = 0 \cap \mathrm{bit}_{Bj} = 1\right) \\ & = & P\left(\mathrm{bit}_{Aj} = 1\right) \cdot P\left(\mathrm{bit}_{Bj} = 0\right) + P\left(\mathrm{bit}_{Ai} = 0\right) \cdot P\left(\mathrm{bit}_{Bj} = 1\right) \\ & = & 0.5^2 + 0.5^2 = 0.5. \end{array}$$

In above equations the independence of bit positions in the BRIEF bit string and the uniform distribution of the image intensity comparisons results were used.

Using Equation (3.7), the probability of two BRIEF features being closer than r in the multidimensional Hamming space can be computed as

$$P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r\right) = \sum_{d=0}^{r-1} P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H = d\right) = 0.5^{256} \sum_{d=0}^{r-1} \binom{256}{d}. \tag{3.8}$$

The conditions in the conditional probabilities in Equation (3.6) are reformulated in terms of the Hamming distance between both BRIEF descriptors:

$$\begin{aligned} &P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \mid ||\mathrm{BD}_A, \mathrm{BD}_B||_H < 256 - m\right) \overset{!}{>} \\ &P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \mid ||\mathrm{BD}_A, \mathrm{BD}_B||_H > 0\right), \end{aligned} \tag{3.9}$$

where m is the length of the hash code.

Using Bayes' theorem the two sides in Equation (3.9) can be derived. The left side becomes:

$$\begin{aligned} &P(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \mid ||\mathrm{BD}_A, \mathrm{BD}_B||_H < 256 - m) = \\ &= \tfrac{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \,\cap\, ||\mathrm{BD}_A, \mathrm{BD}_B||_H < 256-m)}{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H < 256-m)} = \\ &= \tfrac{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r)}{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H < 256-m)} = \\ &= \tfrac{0.5^{256} \sum_{d=0}^{r-1} \binom{256}{d}}{0.5^{256} \sum_{d=0}^{256-m-1} \binom{256}{d}} = \\ &= \tfrac{\sum_{d=0}^{r-1} \binom{256}{d}}{\sum_{d=0}^{256-m-1} \binom{256}{d}} \end{aligned} \tag{3.10}$$

where $r < 256 - m$ was assumed. And the right side amounts to:

$$\begin{aligned} &P\left(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \mid ||\mathrm{BD}_A, \mathrm{BD}_B||_H > 0\right) = \\ &= \tfrac{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H < r \,\cap\, ||\mathrm{BD}_A, \mathrm{BD}_B||_H > 0)}{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H > 0)} = \\ &= \tfrac{P(0 < ||\mathrm{BD}_A, \mathrm{BD}_B||_H < r)}{P(||\mathrm{BD}_A, \mathrm{BD}_B||_H > 0)} = \\ &= \tfrac{0.5^{256} \sum_{d=1}^{r-1} \binom{256}{d}}{0.5^{256} \sum_{d=1}^{256} \binom{256}{d}} = \tfrac{\sum_{d=0}^{r-1} \binom{256}{d} - 1}{\sum_{d=0}^{256} \binom{256}{d} - 1} = \\ &\approx \tfrac{\sum_{d=0}^{r-1} \binom{256}{d}}{\sum_{d=0}^{256} \binom{256}{d}}. \end{aligned} \tag{3.11}$$

Plugging Equations (3.10) and (3.11) into Inequality (3.9) yields the necessary condition for locality-sensitivity of the selected hash function:

$$\begin{aligned} &\tfrac{\sum_{d=0}^{r-1} \binom{256}{d}}{\sum_{d=0}^{256-m-1} \binom{256}{d}} \overset{!}{>} \tfrac{\sum_{d=0}^{r-1} \binom{256}{d}}{\sum_{d=0}^{256} \binom{256}{d}} \\ \Leftrightarrow \quad &\tfrac{\sum_{d=0}^{256} \binom{256}{d}}{\sum_{d=0}^{256-m-1} \binom{256}{d}} \overset{!}{>} 1. \end{aligned} \tag{3.12}$$

Clearly, Inequality (3.12) is satisfied for all reasonable values of $m \in \{1, \ldots, 255\}$. Therefore the hash function as defined in Equation (3.5) is locality sensitive. □

Before aNN queries are possible, the hash tables have to be prepared by inserting pointers to all Brief features into the buckets in the hash table that the hash code of the respective BRIEF feature maps to. This initialization procedure for the hash tables is outlined in Algorithm (1).

Algorithm 1 Before queries can be performed the hash functions have to be generated and all database features have to be sorted into the hash tables $\mathcal{T}_l$.

require The number of hash tables L and the number of bits per hash code M;
Database of N BRIEF feature descriptors BD_i.
for $l = 1$ **to** L **do**
Initialize hash function g_l randomly for hash table $\mathcal{T}_l$
end for
for $l = 1$ **to** L **do**
for $i = 1$ **to** N **do**
Add BD_i to bucket $g_l(\mathrm{BD}_i)$ of hash table $\mathcal{T}_l$
end for
end for
return L prepared hash tables $\mathcal{T}_l$

Once the hash tables are initialized, aNNs can be found for arbitrary BRIEF query features as can be seen in Algorithm (2). For a query feature BRIEF_q a candidate set S is constructed from all database features in hash buckets that the query feature maps to. Multiple occurrences of the same database feature in the candidate set are discarded, before an exhaustive Nearest Neighbor search is conducted. The nearest neighbor within the candidate set is assumed to be the nearest neighbor within the whole database of Brief features.

Algorithm 2 The LSH matching algorithm searches for an approximated Nearest Neighbor match for a given query feature.

require L prepared hash tables $\mathcal{T}_l$ with hash functions g_l generating M bit hash codes;
Query feature bit-string BD_q.
Candidate set $S \leftarrow 0$
for $l = 1$ **to** L **do**
$S \leftarrow S \cup \{\text{feature descriptors from bucket } g_l(\mathrm{BD}_q) \text{ in } \mathcal{T}_l\}$
end for
Discard multiple occurrences of the same feature descriptors in S
$\{\mathrm{BD}_{m1}, \ldots, \mathrm{BD}_{mn}\} \leftarrow$ Nearest Neighbors in S
return approximate Nearest Neighbor matches $\{\mathrm{BD}_{m1}, \ldots, \mathrm{BD}_{mn}\}$

As mentioned in the previous section, there might be several database features with the same minimal distance to the query feature. In this case all potential matches are returned by the LSH search.

3.4 Pose Recovery from 2D-3D Feature Matches

Using the search algorithms introduced in the previous section, 2D-3D feature correspondences can be established. For a set of query features in the current image the best matching descriptors in a database of previously observed features are found. These keypoint descriptors, which were extracted from previous frames, have to be associated with a 3D position. This is usually done by the visual odometry system which maintains a position estimate for all features.

How these 2D-3D correspondences can be utilized to estimate the pose of the camera observing the features in a given image, will be described in the following sections. First, the so called 3-point pose algorithm will be introduced which is able to compute the pose of the camera solely from four 2D-3D feature associations. Second, it is explained how the hypothesize-and-verify algorithms RANSAC and PROSAC can utilize the 3-point algorithm to robustly estimate the camera pose.

3.4.1 3-Point Pose Algorithm

The 3-point pose algorithm solves the problem of finding the pose of a camera observing three plus one 3D points. The fourth point is necessary to resolve a four-fold ambiguity in the camera pose computation.

First, the depths of three points is computed using all four points. Knowing the depths of the observed points means knowing the 3D position of the features in camera coordinates. Thus, in a second step the transformation between the points in camera coordinates and the points in world coordinates is computed. This transformation corresponds to the pose of the camera.

Let $\mathbf{p}_1^w$, $\mathbf{p}_2^w$ and $\mathbf{p}_3^w$ be known 3D positions of the observed projections $\mathbf{z}_1$, $\mathbf{z}_2$ and $\mathbf{z}_3$ in the camera plane. The goal is to find s_1, s_2 and s_3 since they define $\mathbf{p}_i^c = \frac{s_i}{||\mathbf{z}_i||_2} \cdot \mathbf{z}_i$. Figure (3.3) depicts the geometry of the problem. Since the distances between the 3D points is the same in all coordinate frames, the problem is defined in terms of the distances:

$$\begin{aligned} a &= ||\mathbf{p}_2^w - \mathbf{p}_3^w||_2 \\ b &= ||\mathbf{p}_1^w - \mathbf{p}_3^w||_2 \\ c &= ||\mathbf{p}_1^w - \mathbf{p}_2^w||_2. \end{aligned} \tag{3.13}$$

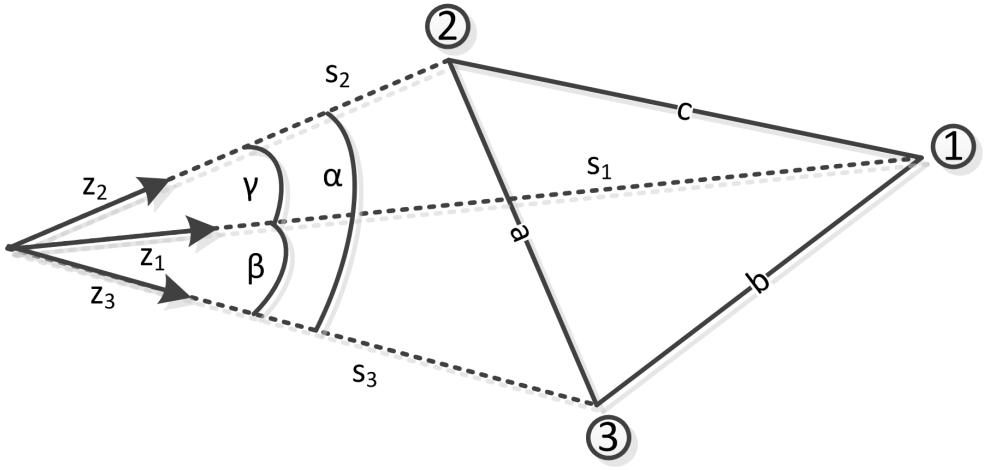


Figure 3.3: The geometry of the 3-point pose problem. The camera center is at the origins of z_1, z_2 and z_3. The camera observes the 3D points 1, 2, and 3.

The observed projections can be interpreted as the directions to the 3D points. The angles between them can be computed as

$$\begin{aligned} \cos\alpha &= \frac{\mathbf{z}_2^T\mathbf{z}_3}{||\mathbf{z}_2||_2||\mathbf{z}_3||_2} \\ \cos\beta &= \frac{\mathbf{z}_1^T\mathbf{z}_3}{||\mathbf{z}_1||_2||\mathbf{z}_3||_2} \\ \cos\gamma &= \frac{\mathbf{z}_1^T\mathbf{z}_2}{||\mathbf{z}_1||_2||\mathbf{z}_2||_2} \end{aligned} \tag{3.14}$$

The law of cosines gives three equations which need to be solved in order to obtain the desired depths s_i:

$$\begin{aligned} a^2 &= s_2^2 + s_3^2 - 2s_2s_3\cos\alpha \\ b^2 &= s_1^2 + s_3^2 - 2s_1s_3\cos\beta \\ c^2 &= s_1^2 + s_2^2 - 2s_1s_2\cos\gamma \end{aligned} \tag{3.15}$$

A good overview over the different solution strategies solving Equations (3.15) can be found in [HLON94]. In this work the solution of Fischler and Bolles [FB81] was used because of its superior accuracy and the fact that no singularities are possible. In the following the only the formulas necessary for finding the solutions are given and the reader is referred to [HLON94] for the full derivation.

Fischler and Bolles rewrite Equation (3.15) in terms of two variables

$$u = \frac{s_2}{s_1} \text{ and } v = \frac{s_3}{s_1}. \tag{3.16}$$

Then 3.15 becomes

$$\begin{aligned} a^2 &= s_1^2(u^2 + v^2 - 2uv\cos\alpha) \\ b^2 &= s_1^2(1 + v^2 - 2v\cos\beta) \\ c^2 &= s_1^2(1 + u^2 - 2u\cos\gamma) \end{aligned} \tag{3.17}$$

After some math they come up with a forth order polynomial in u

$$u^4 + \frac{\tilde{D}_3}{\tilde{D}_4}u^3 + \frac{\tilde{D}_2}{\tilde{D}_4}u^2 + \frac{\tilde{D}_1}{\tilde{D}_4}u + \frac{\tilde{D}_0}{\tilde{D}_4} = u^4 + D_3u^3 + D_2u^2 + D_1u^1 + D_0 = 0 \tag{3.18}$$

where

$$\begin{aligned}
\tilde{D}_4 &= 4b^2c^2\cos^2\alpha - (a^2 - b^2 - c^2)^2 \\
\tilde{D}_3 &= -4c^2(a^2 + b^2 - c^2)\cos\alpha\cos\beta - 8b^2c^2\cos^2\alpha\cos\gamma + \\
&\quad +4(a^2 - b^2 - c^2)(a^2 - b^2)\cos\gamma \\
\tilde{D}_2 &= 4c^2(a^2 - c^2)\cos^2\beta + 8c^2(a^2 + b^2)\cos\alpha\cos\beta\cos\gamma + \\
&\quad +4c^2(b^2 - c^2)\cos^2\alpha - 2(a^2 - b^2 - c^2)(a^2 - b^2 + c^2) \\
&\quad -4(a^2 - b^2)^2\cos^2\gamma \\
\tilde{D}_1 &= -8a^2c^2\cos^2\beta\cos\gamma - 4c^2(b^2 - c^2)\cos\alpha\cos\beta \\
&\quad -4a^2c^2\cos\alpha\cos\beta + 4(a^2 - b^2)(a^2 - b^2 + c^2)\cos\gamma \\
\tilde{D}_0 &= 4a^2c^2\cos^2\beta - (a^2 - b^2 + c^2)^2
\end{aligned} \tag{3.19}$$

The solutions to the fourth order polynomial from Equation (3.18) can be found using the following formulas obtained using an algebraic solver:

$$\begin{aligned}
u_{i1} &= T_1 - R_4 - \sqrt{R_5 - R_6} \\
u_{i2} &= T_1 - R_4 + \sqrt{R_5 - R_6} \\
u_{i3} &= T_1 + R_4 - \sqrt{R_5 + R_6} \\
u_{i4} &= T_1 + R_4 + \sqrt{R_5 + R_6}
\end{aligned} \tag{3.20}$$

where

$$\begin{aligned}
T_1 &= -\tfrac{1}{4}D_3 \\
T_2 &= D_2^2 - 3D_3D_1 + 12D_0 \\
T_3 &= \tfrac{1}{2}\left(2D_2^3 - 9D_3D_2D_1 + 27D_1^2 + 27D_3^2D_0 - 72D_2D_0\right) \\
T_4 &= \tfrac{1}{32}\left(-D_3^3 + 4D_3D_2 - 8D_1\right) \\
T_5 &= \tfrac{1}{48}\left(3D_3^2 - 8D_2\right)
\end{aligned} \tag{3.21}$$

and

$$\begin{aligned}
R_1 &= \sqrt{T_3^2 - T_2^3} \\
R_2 &= \sqrt[3]{T_3 + R_1} \\
R_3 &= \tfrac{1}{12}\left(T_2/R_2 + R_2\right) \\
R_4 &= \sqrt{T_5 + R_3} \\
R_5 &= 2T_5 - R_3 \\
R_6 &= T_4/R_4
\end{aligned} \tag{3.22}$$

It is possible that solutions u_{ij} are imaginary due to several roots in Equations (3.20) to (3.22). While imaginary u_{ij} cannot be used for pose computation, the implementation has to be able to handle imaginary numbers since intermediate results might have imaginary parts that cancel out later on.

For each of the four solutions u_j one can compute the associated value v_j as

$$v_j = \frac{(b^2 + c^2 - a^2)u_j^2 + 2(a^2 - b^2)\cos\gamma u_j - a^2 + b^2 - c^2}{2c^2(\cos\alpha u_j - \cos\beta)} \tag{3.23}$$

Using Equations (3.16) and (3.17) the depths s_i can now be computed for all real-valued solutions pairs u_i, v_i:

$$\begin{aligned} s_1 &= \sqrt{\frac{a^2}{u^2+v^2-2uv\cos\alpha}} \\ s_2 &= us_1 \\ s_3 &= vs_1 \end{aligned} \tag{3.24}$$

Only real-valued pairs u_i, v_i lead to real-valued depths s_i.

Since the fourth order polynomial has up to four real-valued solutions u_i, up to four sets of depths are computed. In order to resolve this four-fold ambiguity, a fourth 2D-3D point correspondence is needed. From these four correspondences, four sets of three correspondences each can be grouped together:

$$\begin{array}{lll} \langle \mathbf{p}_1^w, \mathbf{z}_1 \rangle ; & \langle \mathbf{p}_2^w, \mathbf{z}_2 \rangle ; & \langle \mathbf{p}_3^w, \mathbf{z}_3 \rangle ; \\ \langle \mathbf{p}_1^w, \mathbf{z}_1 \rangle ; & \langle \mathbf{p}_3^w, \mathbf{z}_3 \rangle ; & \langle \mathbf{p}_4^w, \mathbf{z}_4 \rangle ; \\ \langle \mathbf{p}_1^w, \mathbf{z}_1 \rangle ; & \langle \mathbf{p}_2^w, \mathbf{z}_2 \rangle ; & \langle \mathbf{p}_4^w, \mathbf{z}_4 \rangle ; \\ \langle \mathbf{p}_2^w, \mathbf{z}_2 \rangle ; & \langle \mathbf{p}_3^w, \mathbf{z}_3 \rangle ; & \langle \mathbf{p}_4^w, \mathbf{z}_4 \rangle ; \end{array} \tag{3.25}$$

For each of those correspondence sets, the solution set of depths is computed. The true set of depths can be found since it has to be contained four times in the set of all depths. In theory it would be sufficient to evaluate the depths for two correspondence sets to find the true set of depths. In practice however, it is more robust to compute the solutions for all four correspondence sets to find the true set of depths.

Knowing the depths s_i of all points, the positions $\mathbf{p}_i^c$ of the points in camera coordinates can be computed as

$$\mathbf{p}_i^c = \frac{s_i}{||\mathbf{z}_i||_2} \cdot \mathbf{z}_i. \tag{3.26}$$

Given the correspondences between points in camera coordinates $\mathbf{p}_i^c$ and points in world coordinates $\mathbf{p}_i^w$, the transformation from camera coordinates into world coordinates ${}^w\mathbf{T}_c$, the pose of the camera, can be computed.

First, the means of both point sets $\bar{\mathbf{p}}_c$ and $\bar{\mathbf{p}}_w$ are subtracted:

$$\begin{aligned} \tilde{\mathbf{p}}_i^w &= \mathbf{p}_i^w - \bar{\mathbf{p}}_w \\ \tilde{\mathbf{p}}_i^c &= \mathbf{p}_i^c - \bar{\mathbf{p}}_c. \end{aligned} \tag{3.27}$$

Second, the rotation ${}^w\mathbf{R}_c$ that aligns the two zero-mean point sets $\tilde{\mathbf{p}}_i^w$ and $\tilde{\mathbf{p}}_i^c$ best is computed using a small number of Levenberg Marquardt iterations. The algorithm is the same as described in Section (2.2.3) except that the cost function is now:

$$E = \tfrac{1}{2} \sum_{i=1}^{4} (\tilde{\mathbf{p}}_i^w - {}^w\mathbf{R}_c \tilde{\mathbf{p}}_i^c)^2 \tag{3.28}$$

The variables that the LM algorithm is optimizing are the $\mathrm{so}(3)$ parameters $\boldsymbol{\omega}$ of the rotation ${}^{w}\mathbf{R}_{c} \in \mathrm{SO}(3)$. This means the Jacobians J_i are

$$J_i = \frac{\partial {}^{w}\mathbf{R}_{c}\tilde{\mathbf{p}}_i^c}{\partial \boldsymbol{\omega}} = \begin{pmatrix} 0 & \tilde{\mathbf{p}}_{iz}^c & -\tilde{\mathbf{p}}_{iy}^c \\ -\tilde{\mathbf{p}}_{iz}^c & 0 & \tilde{\mathbf{p}}_{ix}^c \\ \tilde{\mathbf{p}}_{iy}^c & -\tilde{\mathbf{p}}_{ix}^c & 0 \end{pmatrix} \tag{3.29}$$

using the Lie Algebra formulation as introduced in Section (2.2.1). The update equation becomes

$$\begin{array}{rcl} \Delta\boldsymbol{\omega} & = & \sum_{i=1}^{4} \left(J_i^T J_i + \lambda I\right)^{-1} J_i^T \left(\tilde{\mathbf{p}}_i^w - {}^{w}\mathbf{R}_{c}\tilde{\mathbf{p}}_i^c\right) \\ {}^{w}\mathbf{R}_{c} & = & \exp\left(\Delta\boldsymbol{\omega}\right) {}^{w}\mathbf{R}_{c}, \end{array} \tag{3.30}$$

where $\Delta\boldsymbol{\omega} \in \mathbb{R}^3$ is the differential rotation update in $\mathrm{so}(3)$. Starting from zero rotation, the LM algorithm is iterated until the decrease in the cost function drops below 10^{-2} and convergence is assumed.

Using the means of the point sets in world and camera coordinates and the rotation ${}^{w}\mathbf{R}_{c}$, the camera pose ${}^{w}\mathbf{T}_{c}$ can be computed as

$${}^{w}\mathbf{T}_{c} = \left(\begin{array}{c|c} {}^{w}\mathbf{R}_{c} & \bar{\mathbf{p}}_w - {}^{w}\mathbf{R}_{c}\bar{\mathbf{p}}_c \\ \hline \mathbf{0}^T & 1 \end{array}\right). \tag{3.31}$$

In conclusion, the 3-point algorithm is able to estimate the pose of a camera observing four points of known 3D position. The algorithm does not depend on any prior knowledge of the pose of the camera and is hence suitable for obtaining an initial estimate on the pose of a camera.

3.4.2 Hypothesize-and-Verify driven Pose Recovery

Hypothesize-and-verify driven algorithms instantiate hypothetical underlying models of a dataset from minimal randomly selected sets of datapoints. These hypothetical models are verified and ranked according to their capability to explain the whole dataset. This capability is usually measured by the number of so called inliers. These are datapoints within a certain error threshold from the position predicted by the model.

The strength of this kind of algorithm is the inherent robustness against outliers in the dataset which cannot be explained by the model. Models instantiated from sets containing outlier datapoints will have a lower number of inliers than models created from inlier datapoints. The price for this robustness is that a good model can only be found with a certain probability and that in order to find it, a large number of models might have to be instantiated.

The basic hypothesize-and-verify algorithm commonly used for pose recovery is the Random Sample Consensus (RANSAC) algorithm [FB81].

In the case of 3D pose estimation, the RANSAC algorithm (Algorithm 3) randomly samples four 2D-3D point correspondences to compute a 3D camera pose ${}^{w}\mathbf{T}_{c}$ using the 3-point pose algorithm as described in Section (3.4.1). This model is then verified against the set of all correspondences. A correspondence $\langle \mathbf{z}_i, \mathbf{p}_i^w \rangle$ is an inlier, if the projection of the 3D point $\mathbf{p}_i^w$ under the hypothetical camera pose ${}^{w}\mathbf{T}_{c}$ is within a maximal error distance ϵ_{inlier} from the observed 2D position $\mathbf{z}_i$ in the image:

$$||\mathbf{z}_i - \mathbf{h}(\mathbf{p}_i^w, {}^{w}\mathbf{T}_{c})||_2 < \epsilon_{\text{inlier}} \tag{3.32}$$

Algorithm 3 The RANSAC algorithm for camera pose recovery from a set of 2D-3D feature correspondences.

require Model M and a set of 2D-3D point correspondences $\langle \mathbf{z}_i, \mathbf{p}_i^w \rangle$;
  $S^* \leftarrow \{\}$
  for $i = 1$ **to** J_{max} **do**
    Sample four 2D-3D point correspondences $\left\langle \mathbf{z}_j, \mathbf{p}_j^w \right\rangle$ and compute ${}^{w}\mathbf{T}_{c}$
    Find consensus set S_i for pose ${}^{w}\mathbf{T}_{c}$
    if $|S_i| > |S^*|$ **then**
      $S^* \leftarrow S_i$
      ${}^{w}\mathbf{T}_{c}^{*} \leftarrow {}^{w}\mathbf{T}_{c}$
    end if
  end for
return best model ${}^{w}\mathbf{T}_{c}^{*}$

The Progressive Sample Consensus (PROSAC) algorithm [CM05] is a variant of RANSAC which can reduce the number of models which have to be sampled drastically. The necessary condition for PROSAC to achieve these improvements is that it is possible to rank the set of feature correspondences according to their likelihood of being a correct match. Given such a quality measure, PROSAC samples from increasingly larger subsets of the sorted set of all feature matches. Qualitatively high matches are sampled first.

One measure for the likelihood of being a good feature pairing is the proximity of the matched descriptors in the feature space. In this work, the Hamming distance between matched binary features is employed as the measure of pairing quality. Matches with a lower distance are assumed to be more likely a correct pairing than ones lying further apart in the feature space.

3.5 Relocalization Algorithm

The proposed binary-feature-based relocalization algorithm utilizes BRIEF features for keypoint description, LSH for fast feature matching and RANSAC or PROSAC followed

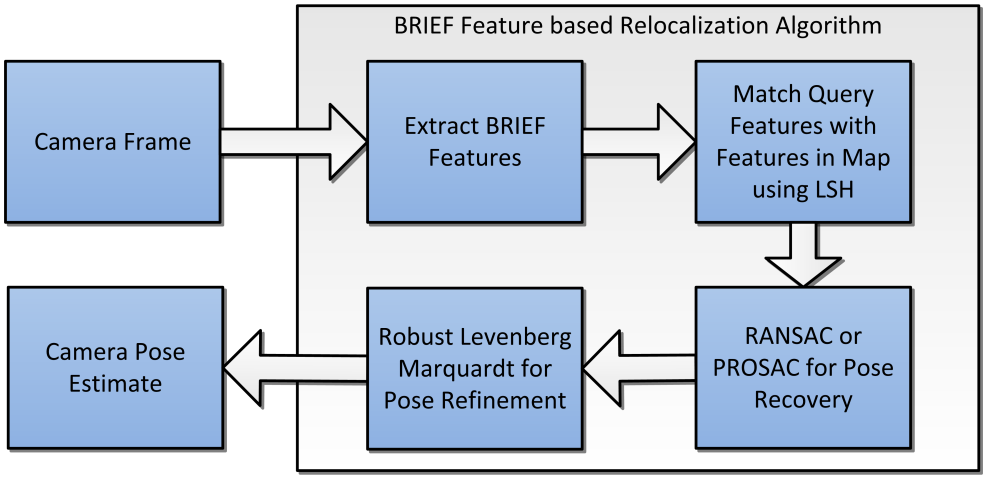


Figure 3.4: Schema of the proposed feature-based relocalization algorithm using binary descriptors.

by a few M-Estimator iterations for pose estimation. The algorithm is designed for the deployment in a visual odometry system which does estimate 3D positions of keypoints such as PTAM or other visual SLAM systems. In order for the relocalization system to work, it is necessary that the visual odometry algorithm maintains a history of BRIEF descriptor observations for each 3D keypoint. Keeping this history mitigates the impact of BRIEF's scale and rotation variance and allows to capture the appearance of a 3D keypoint from different viewpoints. Due to the minimal time requirements for BRIEF descriptor extraction (17.3 μs per descriptor [CLSF10]) this is only a slight additional computational burden. Additionally, it is not necessary to maintain a dense history of descriptor observations of the 3D points. For the implementation of this relocalization algorithm in PTAM, BRIEF descriptors were only extracted in keyframes. These are inserted only approximately every 50 frames.

In case of tracking loss, e.g. due to motion blur, rapid lighting changes or occlusions, the algorithm extracts BRIEF descriptors at image locations selected by the same keypoint detector as used by the visual odometry system. This ensures that keypoints are potentially re-observed and thus can be matched to the correct 3D features in the map.

LSH as described in Section (3.3.1) is used to establish 2D-3D feature associations. Only 3D features which have been observed by from places not further away than 25 m are taken into consideration in the matching process. Note, that 3D features might well be further away than 25 m. This is important in long hallways and large rooms. In case LSH finds several feature pairings with the same minimal distance, all these matches are added to the set of putative correspondences.

All potential 2D-3D feature associations are then fed into a hypothesize-and-verify algorithm for camera pose estimation. In this work, the hypothesize-and-verify algorithms RANSAC and PROSAC were evaluated for robust pose recovery. The 3-point pose algo-

rithm as described in Section (3.4.1) is utilized to compute potential camera poses from four 2D-3D feature matches. Care is taken that hypothetical poses are not computed from matches of one query feature to several 3D features. The pose with the largest inlier count is returned and consecutively refined using a robust LM algorithm as described in Section (2.2.3) and Section (2.2.4). The LM algorithm is run on the set of inliers from the hypothesize-and-verify algorithm to refine the pose.

Chapter 4

Global Localization based on Binary Features

Global localization within a database of 10k images and 100M features necessitates different approaches for localization than for relocalization in the map of the visual odometry system. The techniques described in the previous section are suitable for database sizes of around 1k images and 100k features. The problem with large scale feature-based localization is that feature descriptors are not descriptive enough to allow direct matching of features from query image to a database of map features. This necessitates the introduction of additional constraints to filter out wrong feature matches.

This work focuses on Content-based Image Retrieval (CBIR) [SZ03] techniques for large scale localization. Image retrieval poses the constraint that features of a query image should be visible in the best matching image in the database. Therefore it is not a matching from one feature to another but from a set of query features to a set of features in the database. The representation of an image as the set of features extracted from it, is the so called Bag-of-Features (BoF) model [GLT12]. The assumption is that the image in a database which looks most similar to the query image was taken somewhere in close proximity to the query image. Thus, by finding the most similar image which was taken at a known position the query image location can be recovered.

For image retrieval the quantization of features into so called visual words is necessary. Thus, in order to allow the usage of binary features, a k-means algorithm for binary features was developed. This so called k-Binary Means algorithm will be introduced in Section (4.2) after examining related work in Section (4.1). The use of the k-Binary Means algorithm as a quantizer in a general CBIR system is detailed out in Section (4.3). Finally, the concept of image retrieval driven localization based on Virtual Views [HSH+12b] is introduced in Section (4.4).

4.1 Related Work

The idea of localization using image retrieval has been explored e.g. by [WBB02] prior to the introduction of techniques from textual search to image retrieval by Sivic and Zisserman [SZ03]. Since this knowledge transfer first enabled really large scale image retrieval and thus large scale localization, only work utilizing these CBIR techniques will be reviewed in the following.

FAB-MAP 2.0 [CN10] is an algorithm for appearance-based SLAM which is able to build maps over 1000 km long driving sequences. They improve the bag of features representation of images by adding information about metric distances between features and thus increase the descriptiveness of the image representation. Their model can utilize an inverted index to allow fast and scalable large scale place recognition and loop closure for the appearance-based SLAM algorithm.

Schroth et al. deploy CBIR to localize a hand-held device using a database of Google Street View images [SANH+11]. They utilize a Multiple Hypothesis Vocabulary Tree (MHVT) which allows soft assignment of visual words to a query feature. This efficient algorithm can be run on a cellphone to quantize extracted features. The visual words are then uploaded to a server which performs CBIR to localize the hand-held device. In a second paper [SHC+11], Schroth et al. show how localization using CBIR can be performed on a hand-held device using partial vocabularies that are location specific. These allow precise yet storage efficient quantizers which can be downloaded to the cellphone quickly. Since the partial vocabularies are much smaller than the full vocabulary, they allow localization in the area they are generated for directly on the mobile device. In this paper an approximate k-means algorithm is used for feature quantization since hierarchical quantizers such as for example the MHVT are not suitable for the generation of partial vocabularies.

Standard CBIR algorithms for place recognition rely on a set of position associated images which are recorded once by traversing the area in which the system should operate. The downside of this approach is that the system can only localize itself to the positions at which images were taken in during collection of the database images. The Virtual Views approach by Huitl et al. [HSH+12b] effectively removes this constraint by synthesizing views of the indoor environment at positions which have not been visited while gathering database images. Their results show that the image-retrieval-based localization precision improves significantly when utilizing a Virtual Views database instead of the raw image database.

The first localization system to utilize binary features and CBIR was proposed by Gálvez-López and Tardós [GLT12]. Visual words are obtained by discretizing the binary feature space of the BRIEF features using a hierarchical vocabulary tree of depth six and a branching factor of ten. Each level of the tree is obtained by partitioning the features using a k-medoids algorithm. A geometric verification using RANSAC is performed on the images obtained from CBIR.

4.2 k-Binary Means Clustering

The k-Binary Means (kBM) is an adaption of the k-means algorithm for binary strings. It performs a clustering of a set of binary strings into K clusters where the means themselves are binary strings. This is important because it allows the consistent use of the Hamming distances as a similarity measure between the binary strings and the centroids of the clusters. In the following it is assumed that the binary strings stem from BRIEF descriptors although the algorithm can deal with any source of binary strings.

The kBM algorithm iteratively clusters the set of binary strings into K clusters. The clustering is computed such that the cost function J is minimized:

$$J = \sum_{k=1}^{K} \sum_{i,j \in C_k} ||\mathrm{BD}_i, \mathrm{BD}_j||_H, \tag{4.1}$$

where C_k is the index set of BRIEF features bit-strings belonging to cluster k. This set satisfies

$$C_k = \{i \mid \arg\min_j ||\mathrm{BD}_i, \bar{\mathrm{BD}}_j||_H = k\}, \tag{4.2}$$

where $\bar{\mathrm{BD}}_j$ is the binary string centroid belonging to the jth cluster.

After initializing the means from randomly sampled BRIEF features in the dataset, the following three steps are iterated until convergence:

1. Assign all features to their nearest centroid. The distance measure used here is the Hamming distance.
2. Balance empty and over-full clusters.
3. Recompute the centroids of all clusters.

Algorithm (4) outlines the procedure in more detail.

The centroids are computed using a voting algorithm. Every feature in a cluster votes for bit positions which are ones in its bit string. These votes are accumulated in one scoring vector with elements score_i for each bit position:

$$\mathrm{score}_i = \begin{cases} \mathrm{score}_i + 1 & \textbf{if } \mathrm{bit}_i = 1 \\ \mathrm{score}_i & \textbf{otherwise} \end{cases}. \tag{4.3}$$

A threshold of half the number of features N_{Cl} in the cluster is applied to the resulting votes score_i to obtain a binary string for the centroid:

$$\mathrm{bit}_i = \begin{cases} 1 & \textbf{if } \mathrm{score}_i > N_{Cl}/2 \\ 0 & \textbf{otherwise} \end{cases}. \tag{4.4}$$

This voting approach comes up with the same result as the naive implementation which would compute the mean bit value for each position in the binary string. The average bit

Algorithm 4 k-Binary Means clustering algorithm for BRIEF descriptors.

require The desired number K of cluster,
Database of feature descriptors BD_i.
Initialize k centroids from randomly drawn features.
Assign features in database to their nearest centroid.
$J_0 \leftarrow 256$
$n \leftarrow 0$
while $\Delta J > \epsilon \cap n < N_{\max}$ **do**
Balance empty and over-full clusters
Recompute centroids $\bar{\mathrm{BD}}_k$
Re-assign features to closest centroids (compute C_k)
$J_n = \sum_{k=0}^{K} \sum_{i \in C_k} ||\mathrm{BD}_i, \bar{\mathrm{BD}}_k||_H$
$\Delta J = J_n - J_{n-1}$
end while
return K centroids and clusters.

values would additionally have to be rounded to obtain a binary string centroid. Clearly, the voting strategy needs less computational effort than the naive implementation.

Due to the high dimensionality of the Hamming space of the feature bit strings, it can happen that there are empty clusters with only one feature and large clusters with over ten times the average cluster size. In order to get a more fine grained and balanced partitioning of the space, the kBM algorithm performs an extra step at each iteration to balance empty and over-full clusters. First, a set of all features from k_{empty} empty clusters is created and united with the features from the $k_{\text{over-full}}$ largest clusters. The number of largest clusters is determined such that the united set of features can be clustered into clusters which on average have the theoretical mean cluster size $\frac{k}{N}$. From the united set of descriptors $k_{\text{over-full}} + k_{\text{empty}}$ centroids are randomly initialized replacing the centroids of the empty and over-full clusters. Consecutively, the features from the united set are assigned to their respective closest centroid.

In order to enable kBM clustering of large datasets in the order of 10M to 100M features into 100k to 1M clusters several improvements were implemented. Firstly, instead of using exhaustive NN search to assign features to their nearest centroid, LSH as described in Section (3.3.1) is used. Secondly, computation intensive steps in the algorithm were parallelized where possible. These steps are: (1) the assignment of database features to their closest means using LSH, (2) the computation of the binary means and (3) the balancing of empty and over-full clusters.

To parallelize the assignment of all database features to their closest centroid, the set of all features is split into 1000 subsets of features. In multiple threads LSH is used to assign the features of a subset to their closest centroids. In order to avoid copying of data, the

threads work on one large vector of pairings. Since the centroids are not changed in this step of the algorithm, all threads can read from the same vector of centroids.

The recomputation of the centroids can also be parallelized since for each centroid only the features in its cluster are necessary. Thus, the recomputation of the k centroids is split among several threads. Again, each thread works on the assigned centroids which are stored in a single vector in memory.

Finally, the balancing of empty and over-full clusters can be parallelized since the main task is to assign the features in the united set of features from the empty clusters and the over-full clusters to the new randomly sampled centroids. This assignment is parallelized as previously described.

4.3 Content-based Image Retrieval (CBIR)

Content-based Image Retrieval (CBIR) is a technique for image search in large databases. CBIR was first introduced in the context of image search within videos by Sivic and Zisserman [SZ03]. Their main idea is to re-interpret the image retrieval problem in terms of text search within a database of documents. Image features are interpreted as terms (so called visual words) and images as text documents. This allows the application of the large corpus of work done in textual search to the problem of image retrieval. This idea was further developed by Yang et al. [YJHN07] who called the representation of an image by the set of its features the Bag-of-Features (BoF) representation.

By using features extracted from the query image as opposed to using the whole image itself, the query becomes robust against partial occlusion and dynamic scene changes. The query features are used to score all images in the database according to their similarity to the query image.

Before queries on the database are possible, the feature quantizer has to be trained and an inverted file for fast image lookup given a visual word has to be prepared. In this work it is proposed to utilize the k-Binary Means algorithm (see Section (4.2)) for feature quantization. The quantizer is used to assign visual word ids to all features extracted from all N images in the database. A kBM quantizer with M visual words is obtained by clustering all features extracted from the image database into M clusters. The id of the closest mean is used as the visual word id of a feature.

For the registration of all features in the database, exhaustive nearest neighbor search for the closest of the k centroids is performed. This could also be replaced by an LSH search. These visual word ids are then used to construct an inverted file which stores a list of images for each visual word in which it has been observed. The inverted file is important for scoring images in which a specific visual word has been observed without having to traverse the whole list of all images for each visual word.

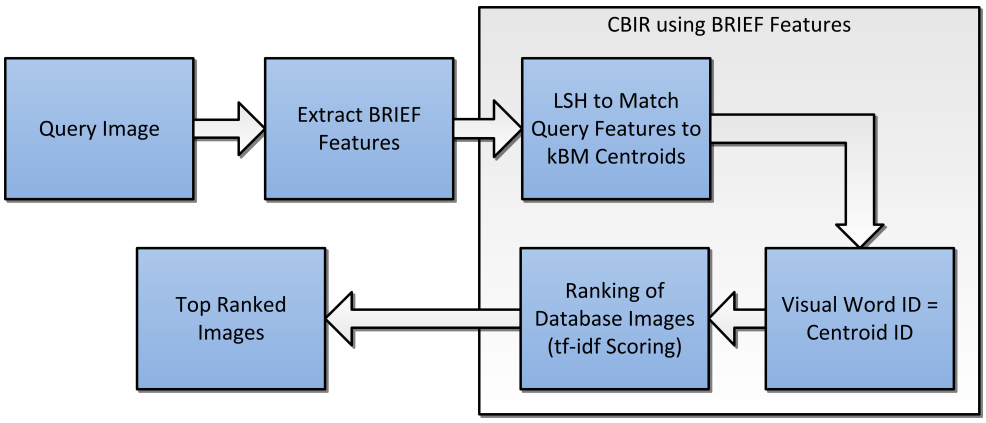


Figure 4.1: Schema of CBIR using BRIEF features. The kBM quantizer is used to facilitate the assignment of visual word ids to BRIEF features.

Employing the BoF [YJHN07] idea, each image in the database is interpreted and described as a set of visual words. The naive approach would be to simply store a vector $\mathbf{b}$ of visual word frequencies for all images in the database. This however ignores that some visual words are more descriptive than others because they are observed less often in the database images or because they are observed more often only in a small subset of database images.

To overcome those deficiencies of the naive approach the term frequency-inverse document frequency (tf-idf) weighting [YJHN07, NS06] is commonly used. The scores b_{ij} in the BoF vector $\mathbf{b}_j$ of each image are computed as:

$$b_{ij} = \frac{n_{id}}{n_d} \log \frac{N}{n_i}, \tag{4.5}$$

where n_{id} is the number occurrences of the visual word i in the d^{th} image, n_d is the number of features in the d^{th} image, and n_i is the total count of features in the database that map to the visual word i.

After these preparations images can be queried in the database as depicted schematically in Figure (4.1). First, features are extracted from the query image. Using the kBM quantizer obtained in the preparation stage, each feature is assigned a visual word id. For the query, the visual word ids are found by an approximate Nearest Neighbour search using LSH. This speeds up the query significantly. Using the visual word ids of the features in the query image, the BoF vector $\mathbf{q}$ is computed as defined in Equation (4.5), where $\frac{n_{id}}{n_d}$ is the frequency of visual word i in the query image.

Following the approach of [NS06], all BoF vectors are normalized before the L_1 metric is used to rank database images according to their distance to the query BoF vector $\mathbf{q}$. The L1 norm can be computed from the sparse BoF vectors as:

$$||\mathbf{q} - \mathbf{b}_j||_1 = 2 + \sum_{i|q_i \neq 0, d_{ij} \neq 0} \left(|q_i - b_{ij}| - |q_i| - |b_{ij}| \right), \tag{4.6}$$

where $\mathbf{b}_j$ is the visual word vector of the jth database image.

CBIR can be used for localization if the images in the database are associated with the pose from which the images was taken. The poses of the top ranked images are then returned as hypothetical positions form which the query image might have been observed. By incorporating additional information about the location or by filtering the pose over several query images from slightly different viewpoints, a unique pose hypothesis can be recovered.

4.4 CBIR Localization from Virtual Views

The problem with CBIR is that localization is only possible to poses for wich images exist in the database. Since mapping larger environments like complete buildings is a time consuming task, usually only one traversal of the area is done while recording localized images in all directions [HSH+12a].

The virtual view approach by Huitl et al. [HSH+12b] aims to solve this issue by simulating a virtual camera to compute a dense set of views of the environment. Views are extracted in a grid of 1 m edge length. For each position 16 images are computed: one for each 15^o of rotation on the spot. In order to render these views, planes are extracted from the point-cloud of the building interior. Using these planes, homographies can be computed between high resolution camera images from the dataset and the virtual camera. Given these homographies the virtual view can be composed of the views of all observable planes under the homography warping. Observability is checked using ray-casting on the planes extracted from the 3D point-cloud model of the environment.

All virtual views together constitute an image database for which CBIR as described in Section (4.3) can be utilized to search for a query image at an unknown location. The positions of the top ranked retrieved images are the desired localization hypotheses.

Note, that by extracting BRIEF features from the virtual view database, the high distinctiveness of this binary descriptor is advantageous. Since the descriptor is neither scale nor rotation invariant, a 3D keypoint will have different descriptors in different Virtual Views. This means that different Virtual Views have distinct sets of features even if they observe the same scene from a different distance or orientation. A query image can thus be localized more accurately to one specific Virtual View.

Using scale and rotation invariant features like SURF does not provide additional value when using Virtual Views. This is because the database is already composed of views of the same scene from different scales and rotations and thus handles the invariance with respect to scale and rotation implicitly. So in contrast to [GLT12], where the little invariance of BRIEF features was found to be the limiting factor for CBIR, exactly this

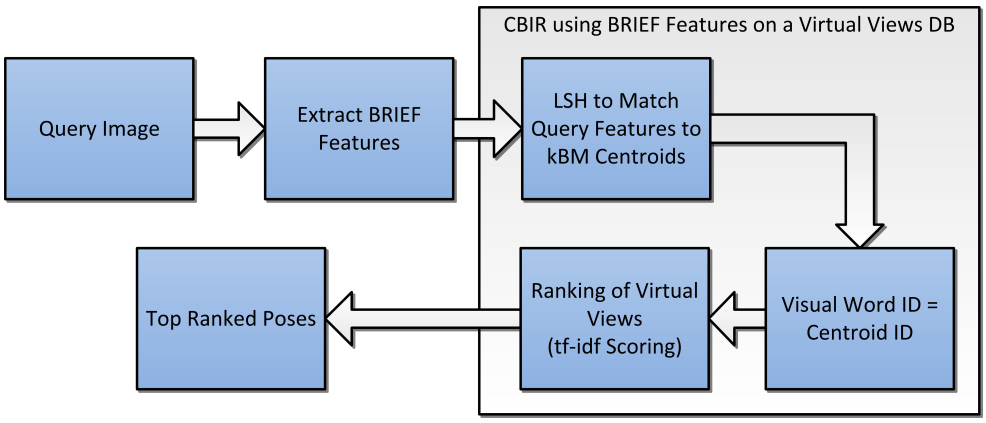


Figure 4.2: Schema of Image-retrieval-based localization using BRIEF features and a Virtual View database (DB). The poses associated with the top ranked Virtual Views are returned as global localization hypothesis.

higher distinctiveness is expected to yield better localization performance when using a Virtual Views database.

Chapter 5

Software Reference

The implementation is split into three main software parts. The first part is the implementation of the NN and aNN search for binary features as well as the kBM clustering algorithm. The second part is the implementation of the BRIEF-based relocalization for PTAM and the integration of this code into the PTAM program by Klein [Kle08]. The third part is the implementation and integration of the kBM quantizer into the CBIR system developed by Huitl [Hui10].

All code is written in C++ since the methods are mostly time-critical. Compilation of the C++ code is handled using cmake. Evaluation and test programs are written in C++ as well and are a good starting point to learn how the individual classes are used. Note that some of the test programs which were not used frequently do not take parameters from the command-line. Instead, the configuration needs to be changed in the code. Evaluation and plotting scripts for the results are implemented as MATLAB scripts and can be found in the *matlab/* folder. Also note that the image databases for CBIR were created using the MATLAB scripts developed by Robert Huitl as described in [Hui10].

In order for most of the scripts to work properly, the environment variable *NAVVIS_WORKSPACE* has to be set to the root of the workspace directory.

```
>> export $NAVVIS_WORKSPACE=/path/to/toplevel/directory/
```

It is most convenient to add this line to the end of the users *∼/.bashrc*.

All code is available from the git repositories on the LMT servers under *LMTprojects/-Navvis/julian/git*. In order for the code to work properly it is important that the folder structure is recreated as outlined in Table (5.1). An example workspace exhibiting this directory structure can be found at *LMTprojects/Navvis/julian/workspace*. The datasets used for the evaluation can be found there as well.

Directory	Description
data/	Output of data from PTAM runs
rosbags/	Rosbags for input to PTAM
dataTablet/	Data obtained from tablet
grundtruth/	Trolley groundtruth data
results/	Results and plots for evaluation
results/raw_*/	Raw results from test programs
matlab/	MATLAB scripts for analyzing and plotting results
lshKBM/tests/	Tests and evaluation code for LSH and kBM
androidDataLogger/	Datalogger application for Android
tabletDataROSPublisher/	ROS node to publish images and IMU data
ptamBriefRelocalize/	Starter scripts for PTAM
ptamBriefRelocalize/PTAM/	PTAM with BRIEF-based relocalization algorithm
ptamBriefRelocalize/test/	Tests for BRIEF-based relocalization methods
work/	CBIR code and MATLAB scripts from [Hui10]
3rdparty/TooN/	Math include files [Ros12c] used by PTAM
3rdparty/libcvd/	LibCVD Computer Vision library [Ros12b] used by PTAM
3rdparty/gvars3/	GVars3 [Ros12a] used by PTAM

Table 5.1: Directory Structure

5.1 Binary Feature Search and Clustering

The *lshKBM* folder contains implementations of the exhaustive NN and LSH search algorithms for binary features, the k-Binary Means clustering algorithm and the kBM quantizer for use in CBIR. Since the search and the clustering algorithms are template classes, their code can mainly resides in header files in *lshKBM/include*. All source files for these algorithms are stored in *lshKBM/src*.

In the *lshKBM/tests* folder the code for several test programs can be found:

lshKBM/tests/**testLsh.cpp** can be used to evaluate the precision, the timing and the percentage of matched features of the LSH algorithm. The program accepts parameters from the command-line. A list of optional parameters can be obtained by running the program with the "-h" option. Results of testLsh.cpp which are saved to *lshKBM/lshResults* can be visualized using the MATLAB script matlab/**plotLSHResults.m**. Plots from this script are shown in the evaluation chapter.

lshKBM/tests/**testBriefDesc.cpp** tests several methods of the BriefDesc class like for example the constructor, voting on bit positions, and different Hamming distance computations. The BriefDesc class is used to handle binary features descriptors.

lshKBM/tests/**testDBQuery.cpp** reads and displays information contained in an image database. Image Databases for CBIR are created using MATLAB scripts by Robert Huitl as described in [Hui10].

lshKBM/tests/**testLshImageRetrieval.cpp** performs a query of features from a PTAM dataset on an image database and displays the top-ranked images. The program creates the kBM quantizer and inverted file in case these files are not found at the specified path.

lshKBM/tests/**testLshTable.cpp** creates hash tables for LSH in different ways: sample hash function randomly or aided by covariance or mutual information matrices. The bit positions that the respective hash function will compose the hash code form are displayed.

5.2 PTAM Evaluation How-To

The original PTAM code from [Kle08] was used with slight modifications by Andreas Möller to allow PTAM to obtain camera frames via the node-based architecture of the Robot Operating System (ROS).

In the following, all steps are described which are necessary to run PTAM on a dataset collected on an Android tablet.

5.2.1 Collecting Data with an Android Device

The android data-logger application in *androidDataLogger/* can collect accelerometer and gyroscope values alongside recording a .mp4 video from the backward facing camera of the device. It was developed and tested on a Samsung Galaxy Tab 10.1. Eclipse with the Android SDK plugin was used to program, upload and debug the application.

Figure (5.1) shows the user interface of the data logger application. Via the preferences button, the user can select the rate at which the IMU data is recorded. Available logging intervals are: slow(200 ms), medium(60 ms), fast(20 ms) and maximum (1 ms to 20 ms - as fast as possible).

Data-logging can be started by simply touching the screen in the area of the camera image. Recording is stopped the same way. During data-logging the current rotational speeds and accelerations are displayed in the lower left corner. Additionally, the logging-rate setting and the path for data-logging is shown. The standard path for the log data is *Log/* but this can be changed in the preferences menu. The program automatically creates a subfolder which is named according to the start date and time of the logging sequence. In this folder, the logger places five files:

- **video.mp4**: a .mp4 video of the backwards-facing camera of the Android device.
- **videoStart.txt**: the starting time of the video in ns.

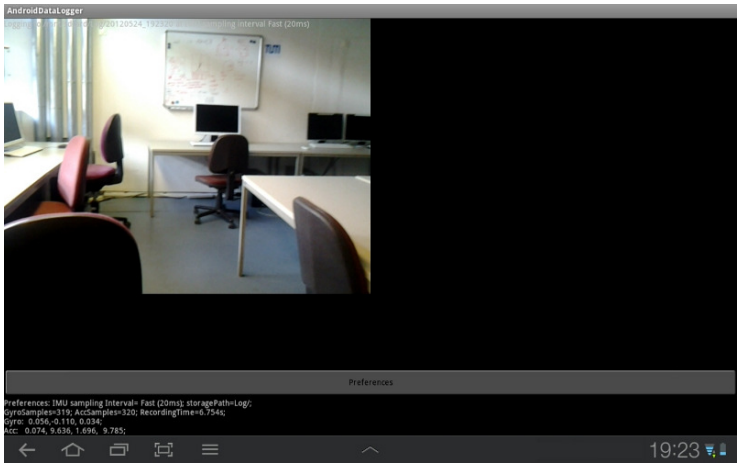


Figure 5.1: Main screen of the Android datalogger application which was developed for dataset collection.

- **videoStop.txt**: the end time of the video in ns.
- **GyroLog.txt**: an ASCII file which is composed of 4 columns: $\left(\; t_\omega \;\middle|\; \omega_x \;\middle|\; \omega_y \;\middle|\; \omega_z \; \right)$, where t_ω is the timestamp in ns of the rotationspeed $\omega = \begin{pmatrix} \omega_x & \omega_y & \omega_z \end{pmatrix}^T$.
- **AccLog.txt**: an ASCII file which consists of 4 columns: $\left(\; t_a \;\middle|\; a_x \;\middle|\; a_y \;\middle|\; a_z \; \right)$, where t_a is the timestamp in ns of the acceleration $\mathbf{a} = \begin{pmatrix} a_x & a_y & a_z \end{pmatrix}^T$.

All timings are in ns and utilize the system time of the Android operating system.

5.2.2 Converting Tablet Data into a ROS bag

In order to run PTAM offline, rosbags need to be recorded. The scripts in *tabletDataROSPublisher/scripts* together with the tabletDataROSPublisher program can be used to convert a video given as a series of frame pictures or as a .mp4 file into a rosbag.

tabletDataROSPublisher/**README.txt** contains instructions on how to setup ROS and the environment to compile the tabletDataROSPublisher ROS node.

tabletDataROSPublisher/scripts/**convertToRosbag.sh** is the central script which sets up the ROS pipeline to publish images and IMU data with tabletDataROSPublisher The ROS program *rosbag record* is then used to collect all published data into a single rosbag. The script uses *extractFramesFromMp4.sh* and *extractFrameTimestampsFromMp4.sh* to extract frames and timestamps in case the specified tablet data folder in *dataTablet* does only contain a .mp4 file.

tabletDataROSPublisher/src/**tabletDataROSPublisher.cpp** is a program to publish images and IMU data to ROS nodes. The images have to be named according to the following C printf-style naming convention: *frame%06d.txt*. Additionally, the program needs a file *FrameInfo.txt* which is a list of timestamps t_f and frame ids i_f in two columns: $\left(\; t_f \; \middle| \; i_f \; \right)$. From a .mp4 video this list can be created using the aforementioned script *extractFrameTimestampsFromMp4.sh*. For the IMU data the program expects two files: *GyroLog.txt* for the gyroscope measurements and *AccLog.txt* for the acceleration values. The content of the files has to match the layout as obtained from the Android DataLogger application as described in the previous Section (5.2.1). It is important, that all timestamps are in the same time reference.

tabletDataROSPublisher/scripts/**convertBatchToRosbags.sh** and **convertAll.sh** utilize *convertToRosbag.sh* to convert multiple datasets to rosbags.

tabletDataROSPublisher/scripts/**extractFramesFromMp4.sh** is a bash script to extract all individual frames from a recorded .mp4 video. The frames are stored as a series of images with the following naming convention: *frame%06d.txt* (C printf-style notation). This script requires the ffmpeg package to be installed.

tabletDataROSPublisher/scripts/**extractFrameTimestampsFromMp4.sh** extracts the timestamps of all frames in a .mp4 video and saves them to a FrameInfo.txt file as specified previously. Note that this requires the program ffprobe which comes with the ffmpeg packet in Linux.

For example, a command to convert the raw tablet data in *dataTablet/20120420_175544/* into a rosbag would be:

```
>> ./convertToRosbag.sh 20120420_175544/
```

5.2.3 Running PTAM

A modification by Andreas Möller to PTAM allows it to obtain the input video from a ROS image stream. The ROS integration is implemented in a shared library in *ptamBriefRelocalize/ptam_ros_wrapper/*. It is important that the *ptam_ros_wrapper/* library and PTAM itself are compiled before the following scripts can be used to run PTAM.

ptamBriefRelocalize/**README** contains instructions on how to compile PTAM and the shared library for ROS integration.

ptamBriefRelocalize/PTAM/**makeRos** is a short script to compile the shared library which allows PTAM to obtain frames from an image_transport ROS node.

ptamBriefRelocalize/**launchROS_PTAM.sh** is a bash script for the evaluation of PTAM. This script ensures that roscore is running, a specified rosbag is played and that image_transport is started to uncompress the images in the rosbag. Once the ROS pipeline

is properly setup, the script runs PTAM with an optional set of parameters. Once PTAM is finished the script creates a new folder in *data/* and copies all PTAM output into that folder. Possible parameters can be seen by running:

```
>> ./launchROS_PTAM.sh -h
-------------------------------------------------------------------------------
Launch programs necessary to process data with PTAM
-------------------------------------------------------------------------------
valid configuration parameters are:
-b |--bag <bagfile>:          .bag file to be played
-s |--start <seconds>:        Start <seconds> into the dataset
-x |--skip <frames>:          Skip <frames> frames before one is handed over to PTAM
-rt|--reloc-time <frames>:    Relocalize always after <frames> frames
-rw|--reloc-wait-time <dt>:   Wait for <dt> ms after relocalization before tracking is
                              resumed
-r |--reloc-type <typeNr>:    Use relocaliser of type <typeNr> (0=standard PTAM,
                              1= BRIEF+LSH+PROSAC+MEstimator)
-e |--end <frames>:           Stop after <frames> frames
-h |--help:                   Display this help message
```

ptamBriefRelocalize/**calibrateROS_PTAM.sh** can be used to calibrate the FOV camera model deployed by PTAM given a calibration video in a rosbag. This bash script runs sets up the ROS image publishing pipeline as described for *launchROS_PTAM.sh* and runs the original camera calibration program of PTAM.

ptamBriefRelocalize/test/**statisticsBRIEF.cpp** contains code to evaluate several statistics and probabilities of a set of BRIEF features and to save them to several files in the same directory. Among the statistics is the computation of the joint probability distribution of all bit positions in with all others, the computation of the mean and standard deviation of the bit positions and the computation of the covariance matrix between all bit positions. MATLAB scripts in *matlab/briefStatistics/* are used to visualize the BRIEF statistics. The main scripts are **plotCorrelation.m** and **plotMutualInformation.m**.

ptamBriefRelocalize/test/**testLshKbm.cpp**: similarly to *testLsh.cpp* this program can be used to evaluate the precision, the timing and the percentage of matched features of the LSH algorithm. In contrast to *testLsh.cpp* it is possible to evaluate LSH on large sets of BRIEF features taken from a database create using scripts from [Hui10]. A list of optional parameters can be obtained by running the program with the "-h" option. The results of testLshKbm.cpp are stored in *ptamBriefRelocalize/test/lshResults* and can be plotted using the MATLAB script matlab/**plotLSHResults.m**. The LSH performance plots for LSH in the kBM quantizer were plotted using the combination of those two programs.

ptamBriefRelocalize/test/**testLSHRansacLoc.cpp** can be used to evaluate the performance of BRIEF-based relocalization within PTAM generated maps. The program accepts a series of parameters which can be seen by running it with the "-h" option. Using the "-p" parameter it is possible to switch between RANSAC and PROSAC for pose recovery. Results are saved in *ptamBriefRelocalize/test/relocResults* and named according to the current machine time in order to make file names unique. The MATLAB script matlab/**evalRANSACRelocWithinPTAM.m** utilizes the results of testLSHRansacLoc.cpp

to generate the performance statistics of RANSAC and PROSAC localization shown in the evaluation chapter.

ptamBriefRelocalize/test/**testRANSACImageRetrievalLoc.cpp** contains code for the performance evaluation of Image-retrieval-based localization strategies. The set of optional parameters for this program can be seen by running it with the "-h" option.

There are several MATLAB scripts in *matlab/* which can be used to evaluate the performance of PTAM and the relocalization algorithms:

matlab/**evalDistToGT.m** is used to evaluate and plot the deviation of PTAM's pose estimates from the groundtruth trajectory collected with the trolley. Additionally statistics such as average distance to groundtruth before and after relocalization as well as time taken for relocalization is displayed in the MATLAB console.

matlab/**videoPathInMap.m** can be used to display a video of the PTAM pose estimates alongside the groundtruth positions as time progresses. This script was used to cut the video of the PTAM program's camera view next to PTAMs pose estimates and corresponding groundtruth positions.

5.3 Content-based Image Retrieval

The kBM quantizer as described in Section (4.2) as well as the option to extract BRIEF features form the database images, was integrated into the CBIR system developed by Huitl [Hui10]. The files **features.cpp** and the **cbir.cpp** were modified accordingly.

work/mser-surf/cbir/**cbir.cpp** allows the selection of the kBM quantizer via the "-q kbm" option. The file ending of the quantizer is .kbm and the inverted files are called .kbm.set. The number of distinct visual words N can be set using the option "-n N". In case of very large feature databases, the parameter "–max-samples=N_{sample}" can be used to create the quantizer from N_{sample} features that were randomly sampled from the whole database. The kBM quantizer implements the same interface as the other existing quantizers and thus supports all operations supported by *cbir.cpp*.

work/mser-surf/features/**cvfeatures.cpp** does now contain methods for extracting BRIEF descriptors at FAST corner locations. The OpenCV implementation of BRIEF and the libCVD implementation of FAST is used. The configuration struct config has to be set to the following values in order to construct a database with BRIEF features:

```
config.detector     = 'fast';
config.descriptor   = 'brief';
config.dims         = 32;
% FAST detector
config.fast_number_octaves = 1;
% BRIEF descriptor
config.brief_bytes = config.dims;
config.expand_bits = false;
% Feature descriptor type
```

```
config.type          = 'uint8';
```

work/mser-surf/configs/**config_indoor_ptam.m** is the configuration file used for evaluations of the kBM quantizer. The configuration for the generation of BRIEF descriptor databases can be seen here as well.

work/mser-surf/**do_evalVirtualViewDB.m** was used for the evaluation of the precision of Image-retrieval-based localization using BRIEF features and a Virtual Views database.

An example command for generating a k-binary means quantizer with 200k visual words and the associated inverted file would be:

```
>> ./cbir -q kbm -n 200000 --max-samples=20000000 -b /path/to/db.db
```

Note that the *–max-samples=N* option for randomly sampling a subset of size N from all features in the database is supported by the kBM quantizer and used in the example above.

CBIR queries can be issued as described in [Hui10]. Here the command is given for a querying all images in a query-database:

```
>> ./cbir -q kbm /path/to/querydb.db --query-db /path/to/db.db
          --query-all --prefix /path/to/results
```

For an in depth description of the source code and the MATLAB scripts for CBIR in the *work/* folder, the reader is referred to the Diplomarbeit of Huitl [Hui10].

Chapter 6

Evaluation

In order to measure the performance of the proposed binary-feature-based localization strategies, two scenarios where evaluated: (1) BRIEF-based relocalization in maps generated by PTAM and (2) the global localization using CBIR on a Virtual Views database of one floor of the Technische Universität München (TUM).

The main difference between those two scenarios is the scale of the map. While a typical PTAM map of 100 images contains about 30k features, the map of one floor of the TUM from the NAVVIS database [HSH+12a] has between 10M features for the raw dataset and 10M for Virtual Views.

For each of the scenarios, the key algorithms involved in the respective localization method were evaluated individually as well to aid optimal parameter selection. This includes the evaluation of LSH for direct feature matching (Section (6.1.1)) and for the use in the kBM quantizer (Section (6.2.1)) as well as the evaluation of RANSAC and PROSAC in Section (6.1.2),

6.1 Relocalization within PTAMs own Map

For the evaluation of the accuracy of PTAM and the relocalization mechanisms, a video dataset with associated groundtruth positions was recorded. To facilitate this a Samsung Galaxy Tab 10.1 tablet was mounted on the same trolley that was used to obtain the TUM indoor dataset [HSH+12a]. This setup is depicted in Figure (6.1).

On the trolley a localization algorithm was run to localize the trolley within the previously built and globally optimized map. The localization algorithm utilized the precise laser scanner of the trolley and a particle filter for 2D pose estimation. Thus, it can be assumed that the groundtruth trajectory is sufficiently accurate. For the tablet, a data-logging ap-

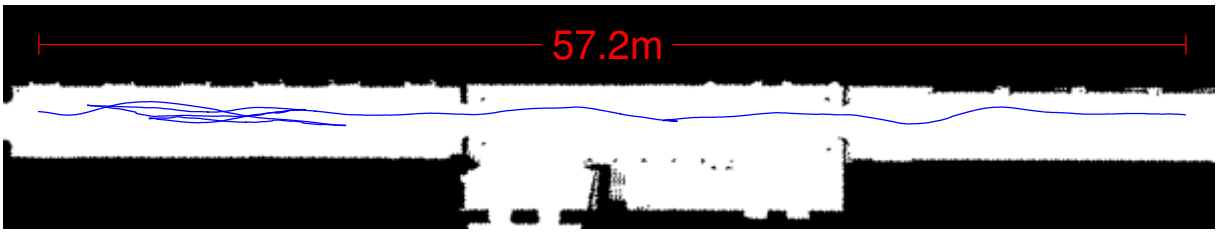
Figure 6.1: The Samsung Galaxy Tab 10.1 was mounted on the NAVVIS trolley to obtain a groundtruth trajectory for the dataset.

plication was developed to record a .mp4 video from the backwards-facing camera alongside the measurements of the built in inertial sensors of the tablet.

Synchronisation of the video and position data was enabled by inserting artificial movement features, i.e. by moving the trolley backwards and forwards once at the beginning and at the end of a dataset. These cues were manually aligned using IMU, video and positioning data. The timestamps for the frames of the video from the tablet were extracted from the .mp4 using the Linux program ffprobe and transformed into tablet time using a timestamp at the start of video recording. Unfortunately, this pretty coarse way of obtaining timestamps was the only possibility since recording individual frames with associated timestamps was not possible due to hardware limitations of the tablet. Since the timestamps of the frames are used to associate PTAM and groundtruth poses, it is likely that some deviation of PTAM from the groundtruth is due to this coarse way of obtaining the timestamps.

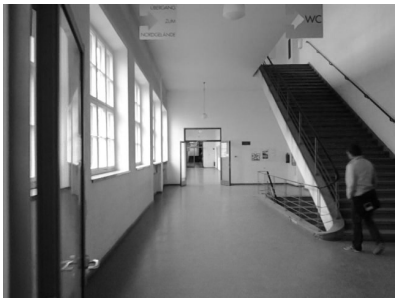
As can be seen in Figures (6.1) and (6.2b) - (6.2d), the dataset was recorded in a well lit corridor without lots of texture rich areas. The left side of the corridor is a series of windows whereas the right side is white except for several display cases. Only in the middle part the right side opens up for a large staircase.

Figure (6.2a) shows the groundtruth path which was recorded by the localization algorithm on the trolley. The run starts in the left segment of the hallway which is traversed three times before entering the middle segment. In the middle, the trolley was pushed past a staircase to enter the right segment of the hallway. The dataset has a length of 8:00 minutes and the trajectory is about 100 m long.

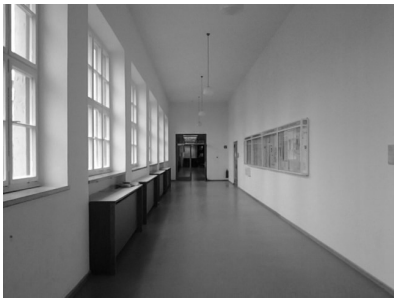


(a) Groundtruth Path

(b) Start Segment



(c) Middle Segment



(d) End Segment

Figure 6.2: Groundtruth path that the trolley was pushed with the Samsung Galaxy Tab 10.1 mounted on it. The dataset starts in the left segment of the corridor, which is traversed three times. Then, after passing a staircase in the middle segment, the path ends on the right.

The evaluation of the relocalization within PTAMs own map is split into three parts: (1) the evaluation of the aNN search algorithm LSH for BRIEF feature matching, (2) the evaluation of RANSAC vs. PROSAC pose recovery accuracy and (3) the comparison of the relocalization accuracy of PTAMs original relocalization strategy vs. the BRIEF-based relocalization algorithm.

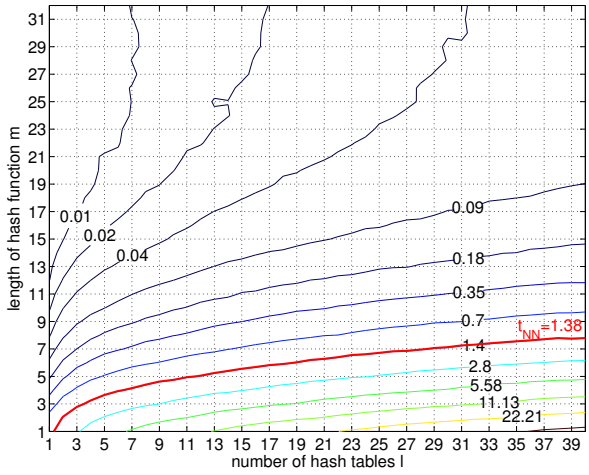
6.1.1 LSH Parameter Selection for Binary Feature Matching

LSH is dependent on two parameters: the number of hash tables L and the number of bits M that the hash code is composed off. Figure (6.3) shows the contour lines of timing, the precision and the percentage of matched features with respect to the aforementioned two parameters M and L. Those plots are averages over ten times 100 queries on a database of 15000 features.

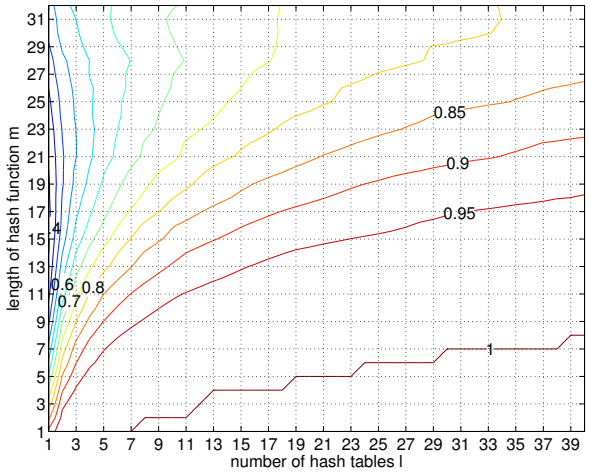
The precision is defined as the number of correctly retrieved nearest neighbors over the number of paired features:

$$\text{precision} = \frac{|\text{true nearest neighbors}|}{|\text{paired features}|} \tag{6.1}$$

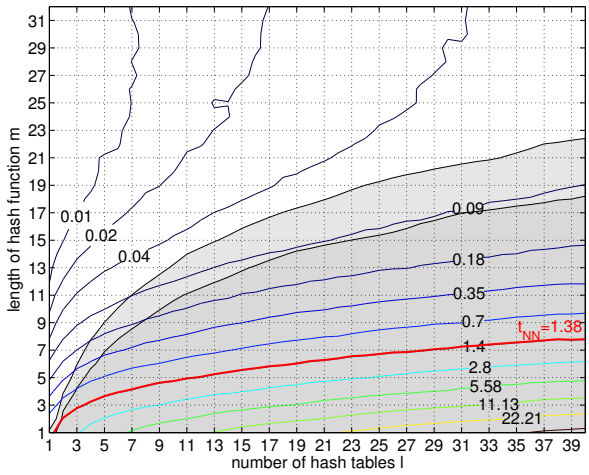
The denominator is the number of matches for the query features since LSH might return more matches than features queried. This is due to the fact that for one query feature several features in the database might have the same minimal distance. Dividing by the number of queried features would allow the precision to become greater than 100% which would not make sense.



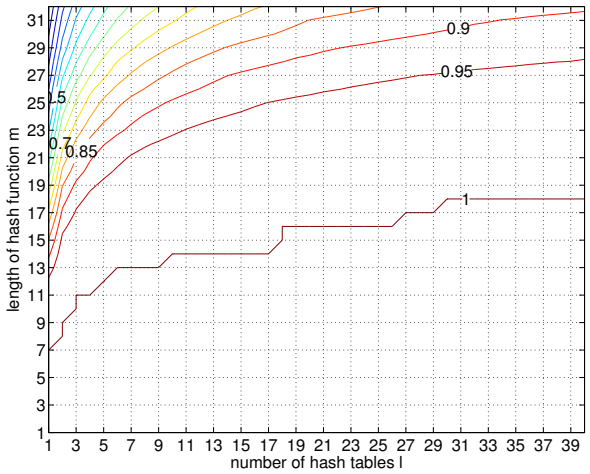
(a) LSH Timing per Query [ms]



(b) LSH Precision



(c) LSH Timing per Query [ms] with 90% and 95% Precision Areas



(d) LSH Percentage of Matched Features

Figure 6.3: Timing, Precision and percentage of matched features for LSH over k and l. The timings are given for the LSH search of the approximate NN of one feature in the database. The contour line corresponding to the timing of exhaustive NN search is plottet in red to allow easy comparison.

The size of the database was chosen by looking at the average number of features that were visible in keyframes within a radius of 25 m around the last known pose. This is the same map feature selection strategy as employed for the BRIEF-based relocalization in

PTAM. While the average number of features selected by this method was about 25k, the number dropped down to as low as 15k. The LSH parameters were optimized for the 15k map features case for security reasons: when the parameters are optimal for 15k features but the actual number of map features is 25k, the matching takes slightly longer and is more precise. In the opposite case, when parameters would be selected for 25k features but there were actually only 15k features in the map, queries would be faster and less precise. Therefore, selecting optimal features for the lower end of the number of expected map features ensures high precision at the cost of slightly longer matching times.

It is also important to look at the percentage of matched features since LSH is not guaranteed to always find a match for a given query feature as explained in Section (3.3.1). The reason is that the hash code of the query feature might map to an empty bucket in the hash table. This happens more often as the number of bits M for the hash code increases. This can be seen in Figure (6.3d). The cause for this observation is, that the number of bins that can be distinguished with M bits in the hash code is 2^M and hence grows exponentially. For N features in the database, empty buckets in the hash tables become increasingly probable for $M > log_2(N)$. With $N = 15000$ in the database of this experiment, empty buckets are likely to occur for $M > log_2(15000) = 13.87$. This can be observed in Figure (6.3d).

The plots in Figures (6.3a) and (6.3b) were used to determine a pair of parameters suitable for the use on a mobile device with a limited amount of memory and computational power. Since the memory requirements scale linearly with the number L of hash tables, a good pair of parameters will have a low value for L. A second constraint is that the precision for a query should be 95% or more. For convenience refer to Figure (6.3c) which shows the areas of precision greater than 90% and 95% shaded in gray. As can be seen from Figure (6.3d), this constraint also means that the matching probability will be 100%. This is desirable if only very few features are queried. To satisfy the requirement of high computational speed, the lowest possible computation time on the curve of 90% precision can be found in Figure (6.3c). The parameters $M = 14$ and $L = 11$ were chosen as the optimal parameters for the deployment of LSH retrieval in a mobile device. Consequently, this parameter configuration was used in all subsequent evaluations.

With a computation time of 59 μs per query LSH search with $M = 14$ and $L = 11$ is more than 23 times as fast as the exhaustive NN search which takes 1.38 ms. The memory usage of eleven hash tables storing 15000 features each is 4.38 MiB. The C++ Standard Template Library map container was used to for the implementation of the hash tables. The size in memory does not include the storage required for the BRIEF descriptors since the hash tables only hold pointers to BRIEF descriptors. The memory consumption was measured using the valgrind tool massif. If an even smaller memory footprint is desired, the parameter configuration of for example $M = 11$ and $L = 7$ can still provide 90% precision while taking about 89 μs per query. This still presents more than a 15-fold speedup over exhaustive NN search. Only 2.20 MiB of memory are necessary to store the seven hash tables of this configuration.

When comparing the timings of LSH in Figure (6.3a) with the exhaustive NN search timing of 1.38 ms per query, the question arises why LSH performs significantly worse for parameter configurations with small hash function lengths M. The reason is that there are 2^M buckets in any of the L hash tables. For small numbers of M there are only few buckets in comparison to the number of features that has to be stored in the hash tables. Therefore, the hash buckets contain many elements (on average $N/2^M$). This means, that the set of potential candidates for a query features becomes large as well (on average $L * N/2^M$). The overhead which leads to longer execution time comes from the removal of duplicated database features from the candidate set. Since smaller values of M and larger values of L lead to larger candidate set sizes, this overhead increases for smaller M and larger L as can be verified in the timing Figure (6.3a).

6.1.2 RANSAC vs. PROSAC for BRIEF-based Pose Recovery

A standard RANSAC algorithm as introduced in Section (3.4.2) was implemented and compared against the slightly more complicated PROSAC algorithm. The best pose estimate from either of the algorithms was refined using a small number of M-Estimator iterations. The 2D-3D BRIEF feature correspondences were established using LSH.

For the evaluation, relocalizations within an original PTAM map were simulated. It is important to note that for each relocalization, the algorithms did only know of the features which PTAM would have observed prior to the relocalization. This resembles exactly the situation that the algorithms would encounter in a real relocalization situation.

Using this setup, the precision and the timing of the localization procedure with respect to the maximal number of pose samples I for both RANSAC and PROSAC was evaluated. For each query frame both algorithms were run five times to account for the inherent and deliberate randomness of the algorithms. The inlier threshold was set to ten pixels.

Figures (6.4) and (6.5) show the probability of the algorithm to localize within a certain radius r around the pose estimate $\mathbf{t}_{\text{PTAM}}$ of PTAM:

$$\text{precision}(r) = P\left(||\mathbf{t}_{\text{PTAM}} - \mathbf{t}_{\text{est}}||_2 < r\right), \tag{6.2}$$

where $\mathbf{t}_{\text{est}}$ is the translation estimated by the respective algorithm.

As can be seen in Figures (6.4) and (6.5), PROSAC needs to sample less poses in order to obtain a high precision pose estimate. This becomes especially clear when looking at $I \in \{1, 3\}$. Therefore PROSAC has a higher probability of sampling a good pose. The main difference between PROSAC and RANSAC is the sorting of all feature matches according to their Hamming distance. Hence, this result confirms that a low Hamming distance between matching BRIEF features provides a good measure for the correctness of the association.

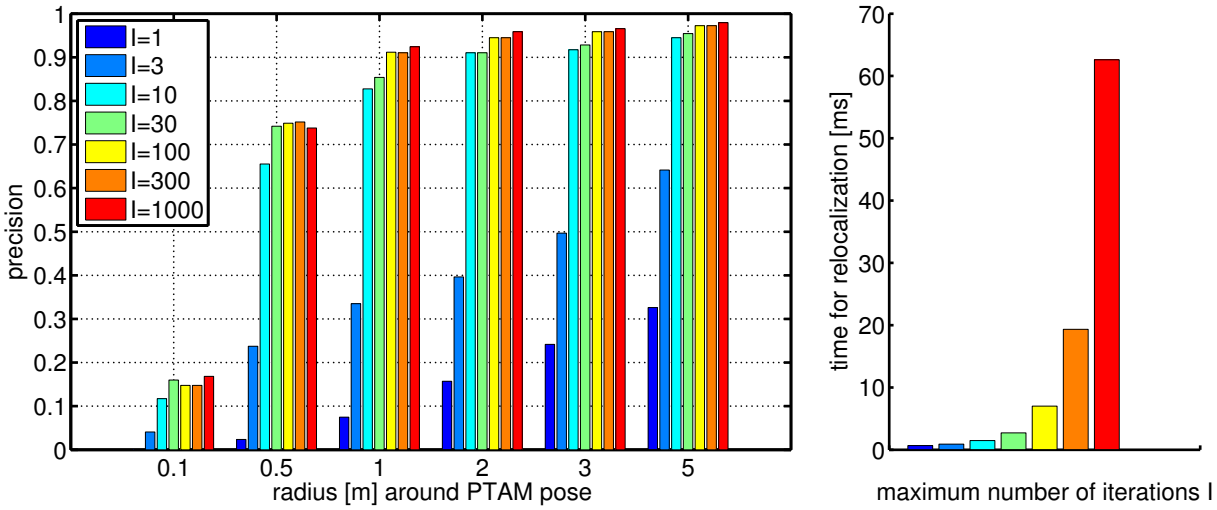


Figure 6.4: Accuracy and timing of the relocalization using LSH for feature matching and RANSAC combined with a M-Estimator for pose estimation.

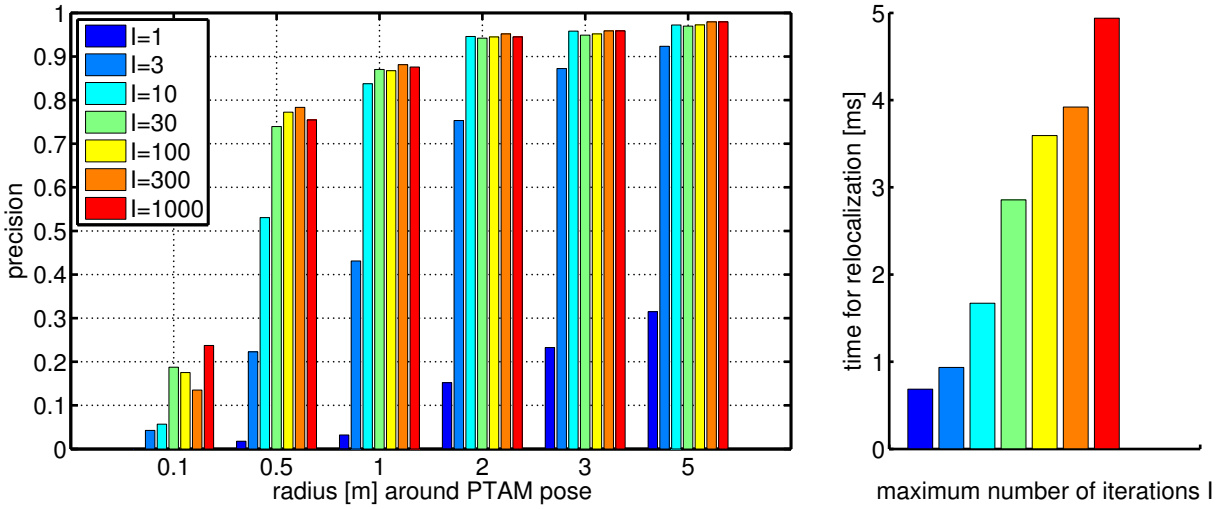


Figure 6.5: Accuracy and timing of the relocalization using LSH for feature matching and PROSAC combined with a M-Estimator for pose estimation.

The timings of RANSAC and PROSAC are displayed in the left plots in Figures (6.4) and (6.5). Especially for large numbers of sampled hypothetical poses I, RANSAC needs significantly more time than PROSAC. This is due to the maximality stopping criterion which is employed by PROSAC to terminate early in case the probability that a better solution exists and was not yet found is smaller than 1%. Hence PROSAC ist faster because the algorithm does not sample the maximal number of poses I. The increased speed leads to slightly smaller precision for large $I \in \{100, 300, 1000\}$. In this range of I, RANSAC delivers slightly more accurate position estimates. This however

comes at the prize of a more than three times longer computation time in comparison to PROSAC.

In the following evaluations of the BRIEF-based relocalization algorithm, PROSAC was used for robust pose recovery due to its higher efficiency. From the evaluation in Figure (6.5) it was deduced that sampling 100 hypothetical poses within PROSAC offers a save trade-off between a high computational speed of below 4 ms and high localization precision. If even less time should be consumed, sampling as few as 30 poses could also be sufficiently accurate while recovering a camera pose in less than 3 ms.

6.1.3 Relocalization Accuracy Comparison

The accuracy of the location estimate of PTAM is evaluated by looking at the distances of pose estimates of PTAM from the groundtruth poses. The poses are associated using their timestamps.

Before the distances can be evaluated, PTAMs trajectory has to be rescaled and transformed into the groundtruth coordinate system. The transformation of PTAM coordinate poses into the groundtruth/world coordinate system ${}^{w}\mathbf{T}_{\text{ptam}}$ is defined as

$$ {}^{w}\mathbf{T}_{\text{ptam}} = {}^{w}\mathbf{T}_{\text{trolley}} \, {}^{\text{trolley}}\mathbf{T}_{\text{ptam}} \tag{6.3} $$

where the transformation from trolley into world coordinates ${}^{w}\mathbf{T}_{\text{trolley}}$ is determined by the mounting position of the tablet and the transformation from PTAM into trolley coordinates ${}^{\text{trolley}}\mathbf{T}_{\text{ptam}}$ is a rotation, to align the forward movement direction in PTAM coordinates and in trolley coordinates.

Since PTAM can only estimate the trajectory up to scale, it is necessary to rescale PTAM to allow meaningful comparison with the groundtruth. For the following evaluation, the scale of the PTAM trajectory was estimated using the time associations between PTAM and groundtruth poses. Essentially, the scale is determined such that it minimizes the sum of squared differences between groundtruth and rescaled PTAM poses at equal times.

Before the relocalization accuracy can be evaluated, it is important to see how well PTAM is able to estimate the trajectory of the dataset. In Figure (6.6) four plots of the deviation of PTAMs trajectory estimate from the groundtruth path. These trajectories are without relocalizations. It becomes clear that while the localization accuracy is less than 0.25 m up to about 400 s into the dataset, the deviation increases drastically at the end of the dataset. This can be explained by wrong position estimation of features at the end of the hallway which have been observed almost the whole dataset. Since they were observed from far away, their constraints in the image space are fairly coarse and thus might lead to wrong position estimates. Wrong feature position estimates in turn lead to deviations of the pose estimates from the groundtruth. It is important to keep these deviations at the end of the dataset in mind when evaluating the relocalization accuracy. It is assumed, that

the coarse timestamp extraction from the .mp4 video contributes to some of the variations in the deviation from the groundtruth.

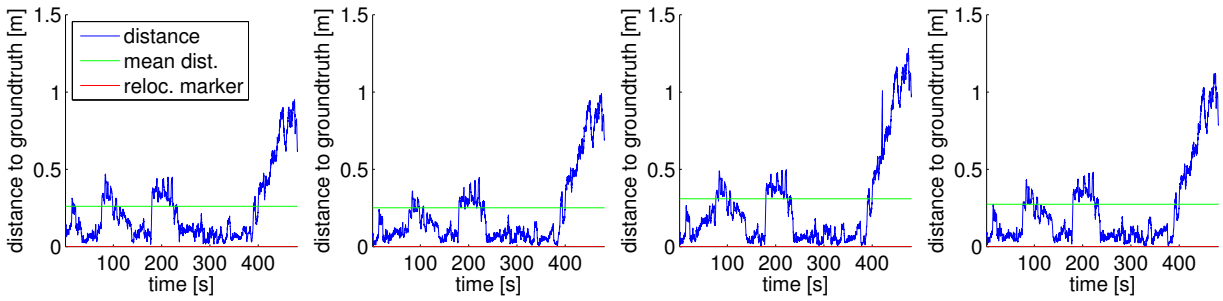


Figure 6.6: Accuracy of PTAMs trajectory estimation on the same evaluation dataset without relocalization.

The relocalization accuracy of PTAMs keyframe-based relocalization scheme is compared with the relocalization algorithm using BRIEF features, LSH for matching and PROSAC with consecutive robust LM iterations for pose recovery as proposed in Section (3.5). The behaviors of both algorithms are evaluated in two different relocalization scenarios: (1) relocalization in a previously traversed area and (2) relocalization in an area which has been observed but not passed before and therefore does not have keyframes associated with it. Figure (6.7) shows the positions that were selected for the evaluation. For this experiment code was added to the original PTAM implementation which allows to force relocalization at a specific frame in the video dataset. After artificially triggering the relocalization no frames are handed to PTAM for 10 s to simulate a longer tracking failure. After this pause normal operation is resumed and PTAM uses the first frame after the break for a first relocalization attempt.

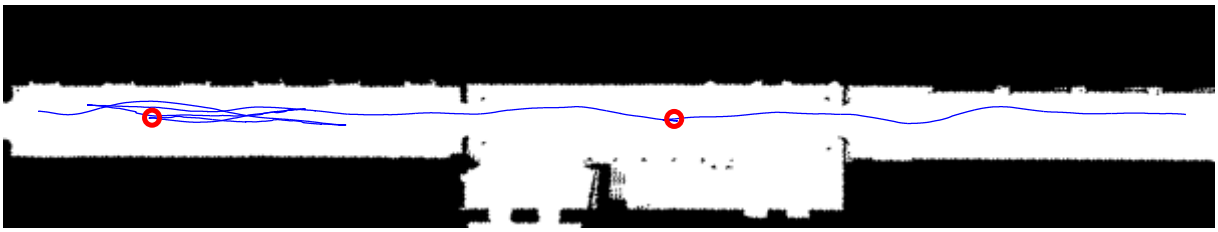


Figure 6.7: The red circles denote the positions at which relocalization was triggered for the accuracy comparison. The left position is the scenario for relocalization within a previously traversed area. The right position was selected to test pose recovery in an area which has not been visited before.

Figure (6.8) shows the deviation of PTAMs pose estimates from the groundtruth for the relocalization scenario within an already visited area. Relocalization is triggered after 1500 frames or about 130 s at the position of th left red circle in Figure (6.7). The upper row of

plots corresponding to PTAMs built in relocalization algorithm show significant deviations from the groundtruth after relocalization. This is despite of the fact that PTAM was designed for relocalization within areas densely populated by keyframes. The problem is the repetitive structure of the hallway which PTAMs native relocalization strategy was not designed to handle. As can be seen from the average distances to the groundtruth after relocalization, the hallway seems to have repetitive elements every 4 m. This could be the windows on the left side of the corridor which are about 4 m apart from each other.

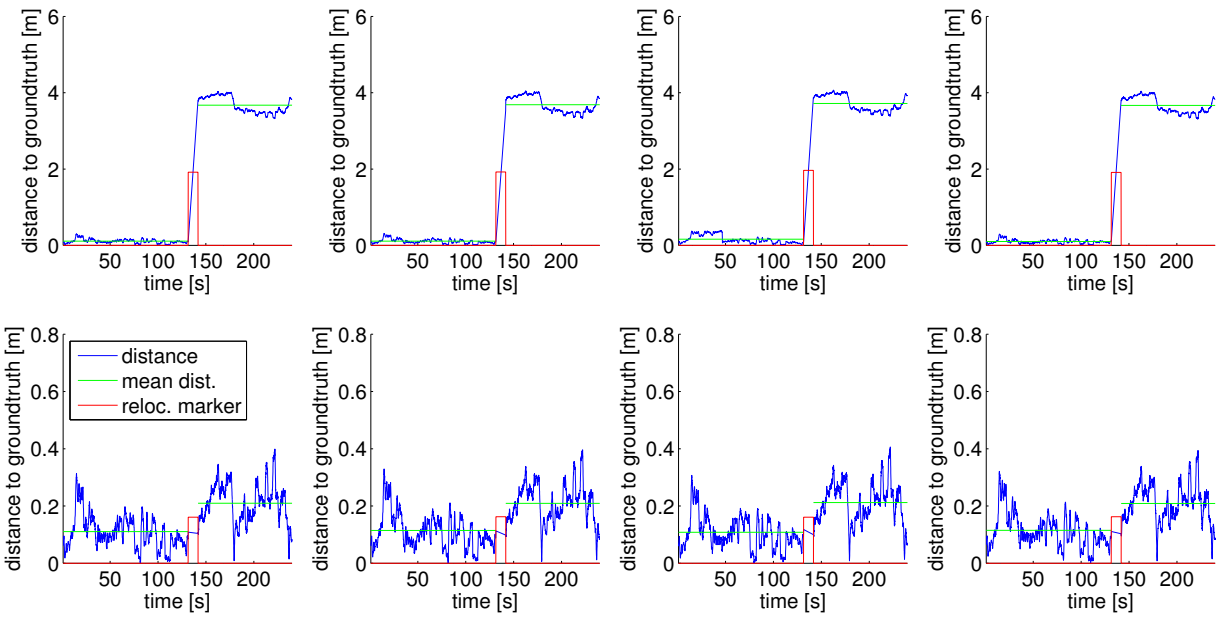


Figure 6.8: Deviation from groundtruth when relocalization is triggered in the first segment of the corridor after the second traversal. The area is thus already explored and PTAM has inserted keyframes around the relocalization position. Top row: relocalization mechanism of PTAM is used. Bottom row: BRIEF-based relocalization is used.

In contrast to that, feature-based relocalization is able to cope with the repetitive structure of the hallway. The average distance before and after relocalization are almost equal and well below 0.50 cm as can be seen in the bottom row of Figure (6.8). This shows the advantage of BRIEF descriptors which can capture the appearance of fine grained texture in the environment. The BRIEF-based relocalization algorithm is able to identify the correct location even in the presence of repetitive elements in the hallway. The keyframe-based approach, however, cannot distinguish the places because it discards all detailed information by down-sampling and blurring the images in order to make a similarity ranking using ZMSSD feasible.

Results for the relocalization scenario in an unexplored area are plotted in Figure (6.9). The position selected for to evaluate pose recovery accuracy is marked by the right red circle in Figure (6.7). The deviations for PTAMs built-in relocalization algorithm are even larger

than in the first scenario. In fact, the algorithm is never able to recover from the tracking loss which can be seen from the relocalization marker. In the second row of plots the relocalization accuracy is plotted for the proposed BRIEF-based relocalization algorithm. While at first sight it looks like the algorithm would fail to recover the pose correctly. When these plots are compared with PTAMs trajectory estimates without relocalization in Figure (6.6) it becomes clear that the large deviations after relocalization are due to the failure of PTAM to estimate the 3D structure of the environment correctly as discussed above.

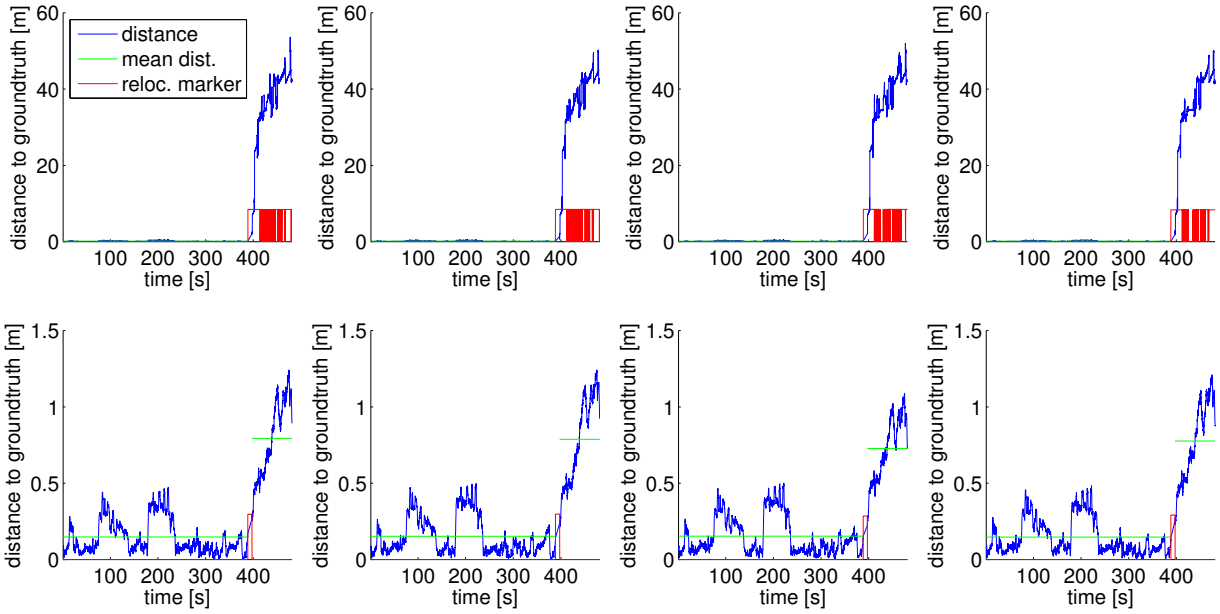
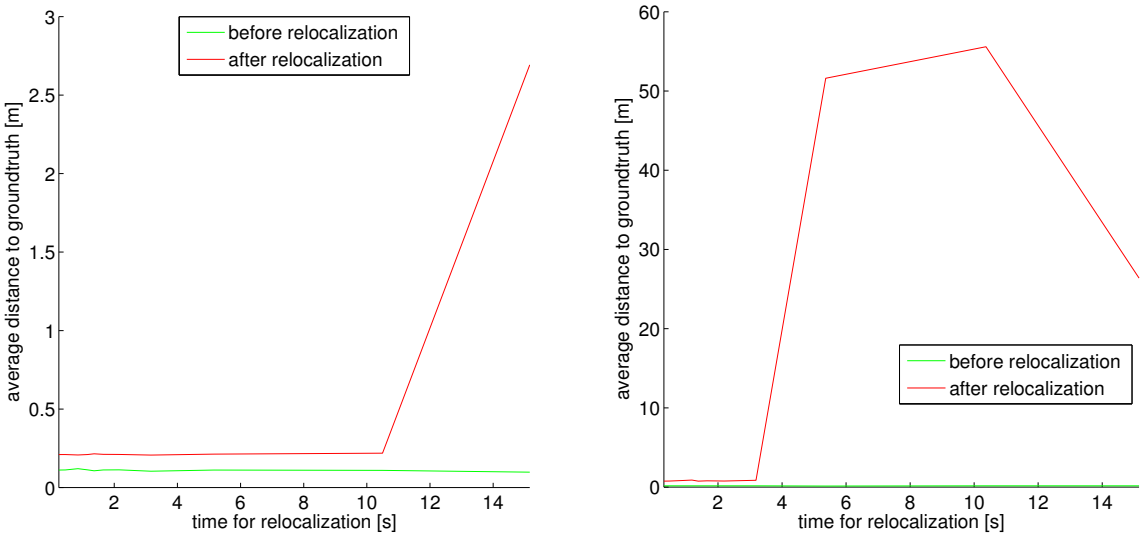


Figure 6.9: Distance to groundtruth over a longer run of PTAM. Relocalization is triggered in the last segment of the hallway. The algorithms need to relocalize in an unexplored area without keyframes. Top row: relocalization mechanism of PTAM is used. Bottom row: BRIEF-based relocalization is used.

On average the BRIEF-based relocalization takes 169 ms with a standard deviation of 36 ms and had to be executed only once in each test. This time measurement includes everything from BRIEF feature extraction to pose refinement. While relocalization takes only about 90 ms in case the number of features selected from the map is around 15k, the average number of map features is 25k as pointed out in the beginning of Section (6.1.1). Therefore matching elongates the time for relocalization. In any case, with an average timing of 169 ms, the whole BRIEF-based relocalization is much faster than extracting the same number (around 700) of SURF features from an image.

In comparison, PTAMs built in relocalization algorithm takes about 1 ms but either relocalize to an incorrect position or failed to relocalize at all. This can be assumed since the relocalization algorithm was called over and over again (see top row of Figure (6.9)).



(a) First Relocalization Scenario (b) Second Relocalization Scenario

Figure 6.10: Impact of the time it takes the algorithm to relocalize on the localization accuracy. The average distance to groundtruth is shown before and after relocalization for different timings of the relocalization algorithm. Note that the high average deviation form groundtruth for the second scenario is due to PTAMs failure to estimate the structure of the environment correctly at the end of the corridor.

Figures (6.10a) and (6.10b) show the average deviations from groundtruth before and after relocalization for the two relocalization scenarios. The mean distances to groundtruth are plotted against the time it takes the relocalization algorithm to recover the pose of the camera. For this evaluation, the BRIEF-based relocalization algorithm was used. The time for relocalization was artificially elongated to evaluate the ability of PTAM to cope with long relocalization delays. This capability is important since the pose is computed from the frame when relocalization is triggered. Due to the time taken by the pose recovery algorithm, this pose is already outdated once it is available. Hence the visual odometry system has to be able to restart tracking from a pose that does not exactly reflect the true pose at that instance in time.

From Figures (6.10a) and (6.10b) it is clear, that PTAM is able to continue tracking even if the relocalization takes up to 10 s in the first scenario and up to 3 s in the second scenario. The higher tolerance to long relocalization times in the first scenario stem from the fact that the relocalization happens in a well explored area. Compared to the average relocalization time of 169 ms a tolerance of 10 s or even 3 s is a long period of time. Therefore it is save to assume that the BRIEF-based relocalization algorithm is able to recover the camera pose fast enough to allow PTAM to continue tracking in time.

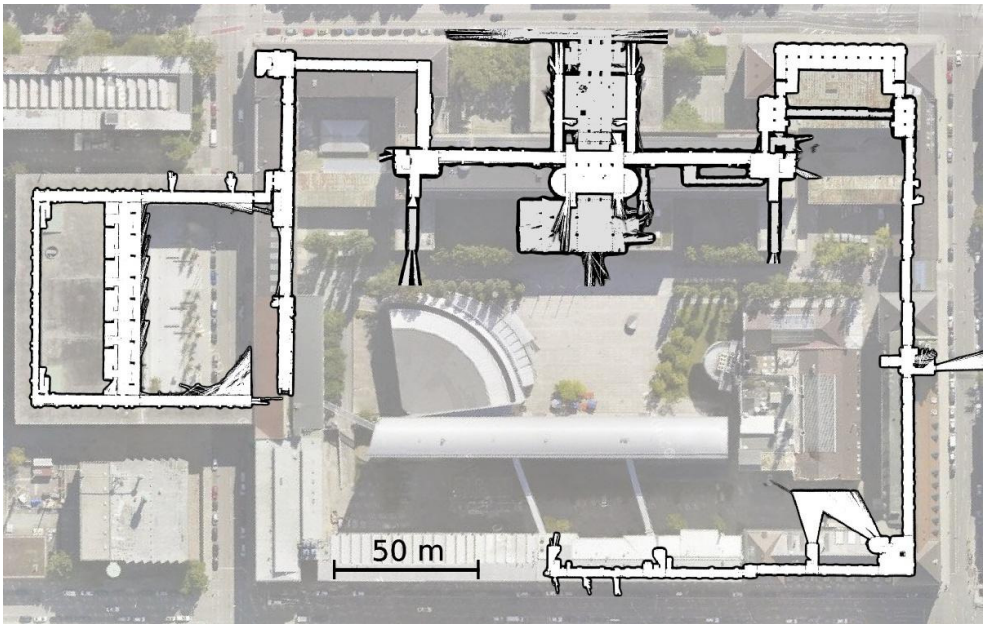


Figure 6.11: The map created while recording the image dataset used for the evaluation of global localization (from [HSH+12a]).

6.2 Localization in a Global Map

For the performance evaluation of the BRIEF-based CBIR algorithm for localization in a large map, the indoor dataset of TUM [HSH+12a] was used. The dataset is comprised of 9438 LadyBug3 images and 3146 high resolution DSLR images. These images were recorded with the trolley described in [HSH+12a] from a complete traversal of the first floor of the TUM main campus in the center of Munich. The map created during this run is displayed in Figure (6.11).

From the dataset of high resolution DSLR images and a 3D point cloud recorded with two high resolution laser scanners, a virtual views database was created using the same code as in [HSH+12b]. The virtual view dataset consists of around 100k images from which about 10M BRIEF descriptors were extracted.

Unfortunately, the lighting conditions in the database images and the Android video datasets used in previous section differ significantly. Thus a different query dataset was used. The images for the query are associated with poses in the groundtruth coordinate system and can thus be used to evaluate localization accuracy. The query dataset is the same as used in [HSH+12b] and the results are thus directly comparable.

6.2.1 LSH Parameter Selection for the kBM Quantizer

The typical quantizer size exhibiting good CBIR results consisted of 200k visual words. In order to optimally tune the performance of LSH for the kBM clustering and quantization, the performance of LSH as an aNN algorithm for matching against 200k centroids was evaluated. These results were collected on an Intel(R) Xeon(R) CPU X5660 running at 2.80GHz. For each parameter configuration 20 different randomly sampled query sets of 100 BRIEF features each was matched against the 200k centroids obtained by kBM clustering of a Virtual Views dataset.

In contrast to the line of argumentation for optimal parameter selection in Section (6.1.1), there are no memory restrictions in this scenario, since all computations are performed offline on a powerful server. This means that the parameter configuration is chosen which minimizes time consumption while maintaining a high retrieval precision. Additionally, it is important that the percentage of matched features is at 100% especially for the quantization of query features, where it is not acceptable to not get a matching centroid.

By looking at the joint plot of timing and precision performance in Figure (6.12c), $M = 19$ and $L = 31$ was found to be a good parameter set. This configuration has 90% retrieval precision at matching times of about 560 μs per query. Compared to 5.3 ms for exhaustive NN search, this presents more than a nine-fold speedup. The memory footprint of the 31 hash tables storing 200k centroids each was measured to be 138.8 MiB. Again, the hash table implementation utilizes a Standard Template Library map container which hold pointers to the binary centroids. The memory consumption was determined using the valgrind tool massif.

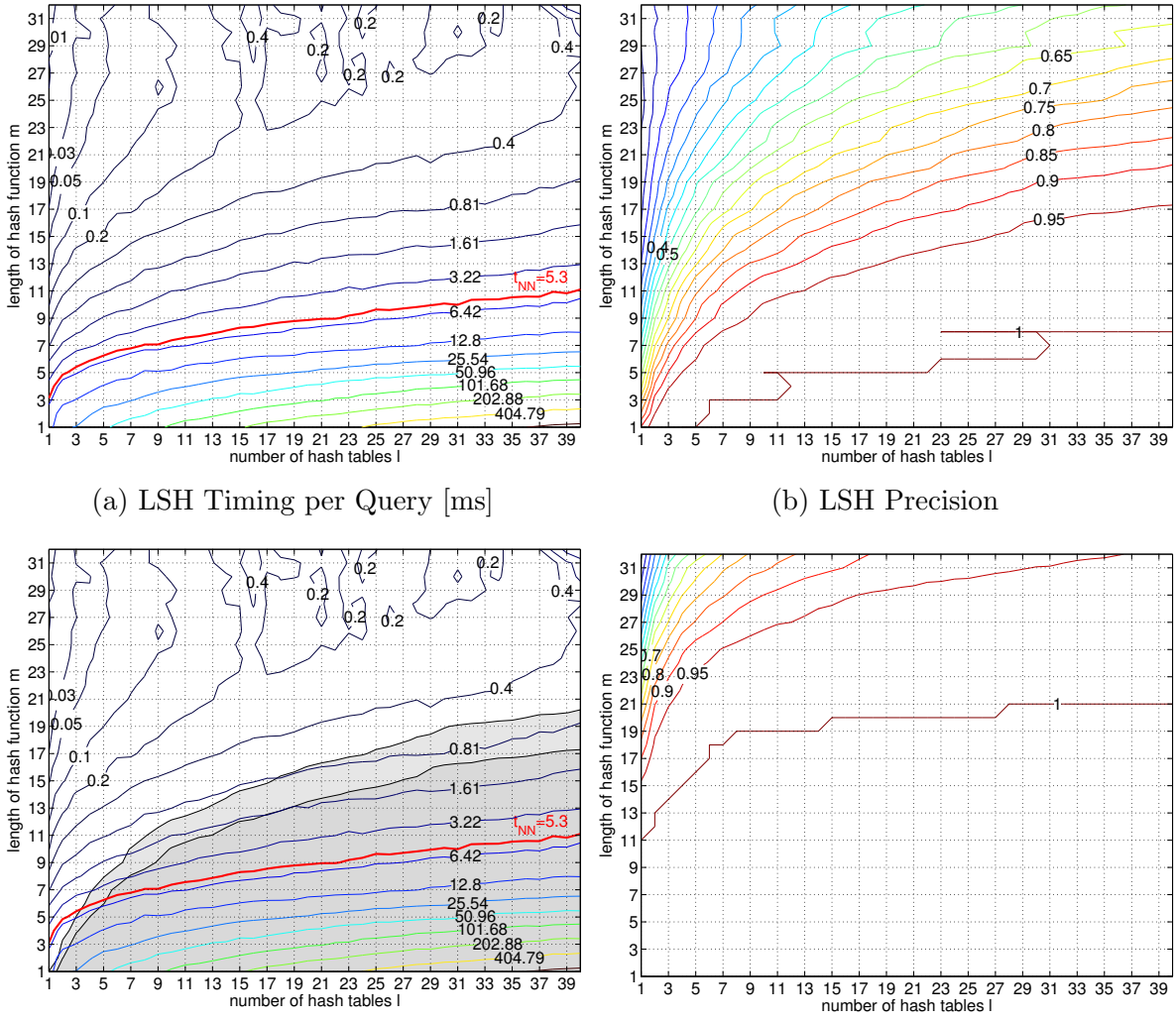
The clustering of 20Mio features into 200k clusters can be done in about 4.5 h using 24 threads on a twelve core Intel(R) Xeon(R) CPU X5660 running at 2.80 GHz. The kBM clustering is terminated after 30 iterations. This is a reasonable time for the generation of a kBM quantizer since this has to be done only once for each image database.

6.2.2 Localization Precision of Virtual Views CBIR

As outlined in Chapter (4), CBIR on a virtual views image database is used to perform large scale localization. In order to evaluate the localization precision, poses within a radius r of 5 m around the true location are declared a correct localization. Any pose outside this radius is defined as a wrong pose estimate. The localization precision at rank i is then defined as

$$\text{precision}(i) = \frac{|\text{top } i \text{ poses within radius } r|}{i}. \tag{6.4}$$

For the evaluation different vocabulary size of 100k, 200k and 500k visual words were created and their localization performance tested. Precisions are averaged over ten times



(a) LSH Timing per Query [ms]

(b) LSH Precision

(c) LSH Timing per Query [ms] with 90% and 95% Precision Areas

(d) LSH Percentage of Matched Features

Figure 6.12: Timing, Precision and percentage of matched features for LSH over k and l. The timings are given for the LSH search of the approximate NN of one feature in the database. The contour line corresponding to the timing of exhaustive NN search is plottet in red to allow easy comparison.

the same sequence of 252 query images. Multiple evaluations of of the same query image is important since LSH is used for BRIEF feature quantization. The slightly random outcome of the image retrieval stems from the parameter set of $M = 19$ and $L = 31$ found in the previous section which has only 90% precision. The average query timings were measured running 24 threads on a twelve core Intel(R) Xeon(R) CPU X5660 with 2.80 GHz clock frequency.

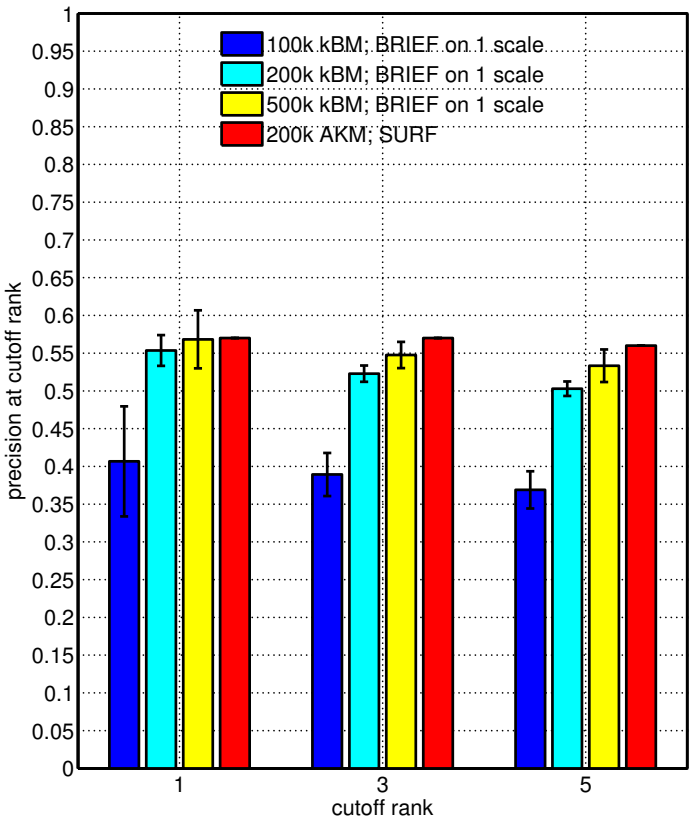


Figure 6.13: Localization precisions at ranks 1, 3 and 5 for BRIEF-based CBIR on Virtual Views databases. The feature databases were constructed from single scale images. For the kBM precision results the 3-sigma errors are plotted as well. These are unfortunately not known for the localization precision of CBIR using AKM and SURF features.

Figure (6.13) shows the results of the localization accuracy evaluation of BRIEF-based CBIR on Virtual Views databases. The significant difference in precision from the 100k vocabulary to the 200k and 500k vocabulary shows that 100k visual words are not sufficient for CBIR using binary features. While the 500k binary means quantizer shows the highest precision, the difference in performance between 200k and 500k visual words is only small. With a time of 0.90 ms, the quantization of a query feature using the 500k vocabulary takes almost three times as long as if the 200k quantizer with a timing of 0.37 ms is used. Additionally, the 500k quantizer needs 16 MB of storage whereas the 200k version is only 6.2 MB large. Therefore, the 200k visual words quantizer offers the best trade off between speed and localization precision while consuming significantly smaller disk space than the 500k quantizer.

Chapter 7

Conclusion

In this thesis, two algorithms based on binary features were developed: the first algorithm utilizes LSH to establish 2D-3D correspondences which are then used by PROSAC to estimate the pose of the camera. Deployed in a visual odometry system, this algorithm can be used for relocalization in case of tracking loss. The second algorithm allows large scale localization of a camera via CBIR from a Virtual Views database. This makes the initialization of a visual odometry system's position in large indoor environments possible. Extensive evaluations were conducted to aid parameter selection for these algorithms.

A thorough evaluation of the LSH matching algorithm was conducted to allow optimal parameter selection for fast and precise BRIEF descriptor matching. For relocalization in PTAM, a parameter set was selected offering more than 90% precision while reducing the query time by a factor of 23 in comparison to exhaustive NN search. The low memory consumption of only around 4.4 MiB for typical database sizes of 15k features makes this LSH configuration suitable for deployment on a hand-held device. By relaxing the memory constraints, a configuration for LSH within the kBM quantizer was found which, at above 90% precision, speeds up the matching process by a factor of nine. Consuming around 140 MiB of memory for a 200k visual word quantizer, this parameter set was selected for BRIEF-based CBIR on a server. Optimal parameters for differing requirements can be found using the timing and precision contour plots.

The comparison of RANSACs and PROSACs positioning accuracy indicated that the Hamming distance between matched BRIEF features can be utilized as a measure of pairing quality. This was deduced from the higher accuracy of pose recovery using PROSAC as opposed to RANSAC when sampling the same number of hypothetical poses. PROSAC prefers BRIEF matches with lower Hamming distance for pose computation. Sampling from low Hamming distance features first yields better pose estimates.

The built-in localization mechanism of PTAM was replaced with the proposed BRIEF-based algorithm. This was shown to yield robust pose recovery in typical sparsely textured and repetitive indoor environments. Hence, it can be assumed that BRIEF features can

replace other more complex feature descriptors like SURF which have been used in related work. This leads to significant speed ups: with an average duration of around 169 ms, the whole BRIEF-based relocalization procedure takes less than half the time necessary to solely extract the same number of SURF features (450 ms for 700 SURF features [CLSF10]).

Using a voting approach, the classical k-means algorithm was adapted for the clustering of bit-strings in high dimensional Hamming spaces. The resulting k-Binary Means algorithm combined with LSH for fast approximate matching was used as quantizer for binary feature bit-strings. This kBM quantizer was deployed in a CBIR system to enable large scale image-retrieval-based localization. Evaluated on a Virtual Views image database, this localization system matched the performance of a state of the art SURF-based CBIR system. Since kBM is not a hierarchical quantizer, location specific partial vocabularies as proposed in [SHC+11] could be created to allow CBIR on a mobile device.

In conclusion, it was shown that the binary feature descriptor BRIEF can successfully replace more complex descriptors like SURF without loss in localization precision. The benefits are twofold: (1) BRIEF extraction is about 40 times faster than SURF extraction and (2) BRIEF needs eight times less storage than the commonly used SURF descriptor. Therefore, the use of binary features is highly recommended. It can be assumed that binary features are key to robust and lightweight localization of hand-held devices.

Chapter 8

Outlook

This chapter summarizes possible future research directions and open questions that arose during the work on this thesis.

As reviewed in Section (3.1.2) there are several more sophisticated binary feature descriptors like ORB, BRISK, or FREAK, which promise to be more invariant to rotation and/or scale changes. While the BRIEF descriptor was used in this thesis to obtain base line results for binary descriptors, it is assumed that these more invariant binary features would yield further improvements at least for the relocalization. It would thus be interesting to evaluate the algorithms performance with the other binary features. The FREAK descriptor, which is supposedly scale and rotation invariant, would be a good starting point.

The implementation of the BRIEF-based relocalization algorithm for PTAM does right now simply store all BRIEF descriptors of all observations of a 3D feature in keyframes and builds new LSH hash tables each time relocalization is performed. This waists a lot of computation time especially if many relocalizations are performed. Thus an improvement to the current implementation would be to have a separate thread that manages the binary features. This thread would incrementally add binary features to one instance of LSH as they are observed. Additionally, this thread could take care of discarding old binary features or ones that are duplicates of features already in the hash tables. This would potentially reduce the number of binary features in the hash tables and thus speed up the matching of query features in the case of relocalization.

In related work on CBIR systems for large scale localization, a geometric post verification of the top-ranked images is performed to detect the retrieved images which can explain the query image under some rigid body motion. For this extra filtering step a hypothesize-and-verify algorithm is used to estimate the transformation between the query and the database image. If the algorithm is not able to find a transformation with enough inlier feature associations the database image is discarded as a potential match. It would be interesting to add a geometric verification step to the CBIR localization pipeline to see how much the localization precision could be improved by such an extra filtering step.

The implementation of BRIEF-based localization on a cellphone is another future direction of work. It is unclear whether the processor architecture on a cellphone can handle binary feature extraction and Hamming distance computation as efficiently as modern laptop or desktop computers. As a starting point, BRIEF-based CBIR for global localization could be ported onto an Android device.

With the implementation of large scale localization on a hand-held device, it also becomes interesting how the bit-strings of the binary features are best encoded for the transmission to a central server. This transmission is necessary to allow the server to support localization based on CBIR by i.e. transmitting a partial vocabulary of the surrounding area back to the cellphone.

Once the large scale localization in a client server architecture has been developed, the question becomes what else besides the pose estimate should be transmitted back to the hand-held device. One possibility would be to send a small set of features and their 3D positions in the world coordinate frame back to the cellphone. A visual odometry algorithm on the mobile device could use those features to initialize its own map of the environment. Not only would this allow for a very simple and fast map initialization wich would not require user interaction but it would also implicitly initialize the scale of the map on the cellphone correctly.

List of Figures

List of Tables

Bibliography

[Agr06] M. Agrawal. A lie algebraic approach for consistent pose registration for general euclidean motion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1891–1897, Beijing, China, 2006. IEEE. 7

[AOV12] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (To Appear)*, Providence, Rhode Island, 2012. IEEE. 23

[AWK+09] C. Arth, D. Wagner, M. Klopschitz, A. Irschara, and D. Schmalstieg. Wide area localization on mobile phones. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 73–82, Orlando, Florida, USA, 2009. IEEE. 23

[Bin11] Bing. Indoor maps. `http://www.bing.com/community/site_blogs/b/search/archive/2011/08/03/new-airport-maps-for-bing-and-mall-maps-come-to-mobile.aspx`, 2011. 1

[BM04] S. Benhimane and E. Malis. Real-time image-based tracking of planes using efficient second-order minimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 943–948, Sendai, Japan, 2004. IEEE. 17, 19

[BTVG06] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *European Conference on Computer Vision (ECCV)*, pages 404–417, 2006. ii, 3, 23

[BWSS10] M. Blosch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *IEEE international conference on Robotics and automation (ICRA)*, pages 21–28, Anchorage, Alaska, USA, 2010. IEEE. 5

[CLSF10] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. *European Conference on Computer Vision (ECCV)*, pages 778–792, 2010. ii, 3, 21, 23, 24, 25, 36, 71

[CM05] O. Chum and J. Matas. Matching with prosac-progressive sample consensus. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 220–226, San Diego, CA, USA, 2005. IEEE. 35

[CN10] Mark Cummins and Paul Newman. Appearance-only SLAM at large scale with FAB-MAP 2.0. *The International Journal of Robotics Research (IJRR)*, November 2010. 39

[CPIP10] K. Chintalapudi, A. Padmanabha Iyer, and V.N. Padmanabhan. Indoor localization without the pain. In *ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 173–184, Chicago, Illinois, USA, 2010. ACM. 1

[CPMCC06] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. *Advances in Visual Computing*, pages 276–285, 2006. 22

[Dav03] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1403–1410, Nice, France, 2003. IEEE. 6, 22

[DBFT99] F. Dellaert, W. Burgard, D. Fox, and S. Thrun. Using the condensation algorithm for robust, vision-based mobile robot localization. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, Ft. Collins, CO, USA, 1999. IEEE. 22

[DF01] F. Devernay and O. Faugeras. Straight lines have to be straight. *Machine Vision and Applications*, 13(1):14–24, 2001. 10

[DRMS07] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 29(6):1052–1067, 2007. 6, 15, 22

[Ead08] E. Eade. *Monocular simultaneous localisation and mapping*. PhD thesis, PhD thesis, University of Cambridge, 2008. 9

[ED06] E. Eade and T. Drummond. Scalable monocular SLAM. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 469–476, New York, NY, USA, 2006. IEEE. 6

[FB81] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 31, 34

[Fri33] Gemma Frisius. Libellus de locorum describendorum ratione. In *Cosmographica*, 1533. 1, 18

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases (VLDB)*, pages 518–529, Edinburgh, Scotland, UK, 1999. 26, 27

[GLT12] D. Gálvez-López and J.D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 2012. 38, 39, 44

[Goo11] Google. Indoor maps. `http://www.google.com/mobile/maps/`, 2011. 1

[HLON94] B.M. Haralick, C.N. Lee, K. Ottenberg, and M. Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International Journal of Computer Vision (IJCV)*, 13(3):331–356, 1994. 31

[HSH+12a] R. Huitl, G. Schroth, S. Hilsenbeck, F. Schweiger, and E. Steinbach. TUMindoor: An extensive image and point cloud dataset for visual indoor localization and mapping. In *International Conference on Image Processing (ICIP)*, Orlando, FL, USA, September 2012. Dataset available at `http://navvis.de/dataset`. 44, 54, 66

[HSH+12b] Robert Huitl, Georg Schroth, Sebastian Hilsenbeck, Florian Schweiger, and Eckehard Steinbach. Virtual reference view generation for CBIR-based visual pose estimation. In *ACM Multimedia (MM)*, Nara, Japan, October 2012. ACM 2012. ii, 38, 39, 44, 66

[Hui10] Robert Huitl. Fast image retrieval for mobile location recognition. Master's thesis, Technische Universität München, Munich, Germany, 2010. 46, 47, 48, 51, 52, 53

[HZ04] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, April 2004. 12, 13, 14, 18

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *ACM Symposium on Theory of Computing (STOC)*, pages 604–613, Dallas, Texas, USA, 1998. ACM. 27

[IWKD12] V. Indelman, S. Williams, M. Kaess, and F. Dellaert. Factor graph based incremental smoothing in inertial navigation systems. *International Conference on Information Fusion (Fusion)*, 2012. 6

[JS11] E.S. Jones and S. Soatto. Visual-inertial navigation, mapping and localization: A scalable real-time causal approach. *The International Journal of Robotics Research (IJRR)*, 30(4):407–430, 2011. 1, 6

[KJR+11] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE*

International Conference on Robotics and Automation (ICRA). IEEE, 2011. 6

[KKD11] Nisarg Kothari, Balajee Kannan, and M Bernardine Dias. Robust indoor localization on a commercial smart-phone. Technical Report CMU-RI-TR-11-27, Robotics Institute, Pittsburgh, PA, USA, August 2011. 1

[Kle08] Georg Klein. PTAM code. `http://www.robots.ox.ac.uk/~gk/PTAM/`, 2008. 15, 16, 46, 48

[KM07] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 1–10, Nara, Japan, 2007. IEEE Computer Society. ii, 1, 6, 15

[KM08] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. *European Conference on Computer Vision (ECCV)*, pages 802–815, 2008. 18, 22

[KM09] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 83–86, Orlando, Florida, USA, 2009. IEEE. 5, 6, 15

[LCS11] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2548–2555, Barcelona, Spain, 2011. IEEE. 23

[LDBL07] H. Liu, H. Darabi, P. Banerjee, and J. Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics (SMC), Part C: Applications and Reviews*, 37(6):1067–1080, 2007. 1

[LMGY04] T. Liu, A.W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. *Advances in neural information processing systems*, 17:825–832, 2004. 22

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1150–1157, Kerkyra, Corfu, Greece, 1999. IEEE. 3, 22, 23

[LS12] T. Lupton and S. Sukkarieh. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Transactions on Robotics*, 28(1):61–76, 2012. 1, 5, 6

[Ltd12] IndoorAtlas Ltd. `http://www.indooratlas.com/`, 2012. 1

[Mar63] D.W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics (SIAM)*, 11(2):431–441, 1963. 11

[Mer11] Meridian. `http://www.meridianapps.com/`, 2011. 1

[MHB+10] E. Mair, G. Hager, D. Burschka, M. Suppa, and G. Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. *European Conference on Computer Vision (ECCV)*, pages 183–196, 2010. 24

[ML09] M. Muja and D.G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications (VISSAPP)*, pages 331–340, Lisboa, Portugal, 2009. 26

[ML12] M. Muja and D.G. Lowe. Fast matching of binary features. In *Conference on Computer and Robot Vision (CRV)*, pages 404–410, Toronto, Ontario, Canada, 2012. IEEE. 26

[MLS94] R.M. Murray, Z. Li, and S.S. Sastry. *A mathematical introduction to robotic manipulation*. CRC Press Florida, 1994. 7

[MSKS04] Y. Ma, S. Soatto, J. Košecká, and S. Sastry. An invitation to 3-d vision. from images to geometric models, 2004. 7

[MTKW03] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 18, pages 1151–1156, Acapulco, Mexico, 2003. IJCAI. 6

[Nok10] Nokia. `http://research.nokia.com/news/9505`, 2010. 1

[NS06] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168, New York, NY, USA, 2006. IEEE. 43

[OCLF10] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(3):448–461, 2010. 22

[OFL07] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Minneapolis, Minnesota, USA, 2007. IEEE. 22

[RD05] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision (ICCV)*, volume 2, pages 1508–1515. IEEE, 2005. 16, 24

[RD06] G. Reitmayr and T.W. Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 109–118, Santa Barbara, CA, USA, 2006. IEEE. 22

[Ros99] P.L. Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999. 23

[Ros12a] Edward Rosten. GVars3. `http://www.edwardrosten.com/cvd/gvars3.html`, 2012. 47

[Ros12b] Edward Rosten. libCVD. `http://www.edwardrosten.com/cvd/index.html`, 2012. 47

[Ros12c] Edward Rosten. TooN. `http://www.edwardrosten.com/cvd/toon.html`, 2012. 47

[RRKB11] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2564–2571, Barcelona, Spain, 2011. IEEE. 23

[SANH+11] Georg Schroth, Anas Al-Nuaimi, Robert Huitl, Florian Schweiger, and Eckehard Steinbach. Rapid image retrieval for mobile location recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, Czech Republik, May 2011. 39

[SEG+05] R. Sim, P. Elinas, M. Griffin, J.J. Little, et al. Vision-based SLAM using the rao-blackwellised particle filter. In *IJCAI Workshop on Reasoning with Uncertainty in Robotics*, volume 14, pages 9–16, Edinburgh, Scotland, UK, 2005. 6

[SHC+11] G. Schroth, R. Huitl, D. Chen, M. Abu-Alqumsan, A. Al-Nuaimi, and E. Steinbach. Mobile visual location recognition. *IEEE Signal Processing Magazine*, 28(4):77–89, 2011. 39, 71

[SLL05] S. Se, D.G. Lowe, and J.J. Little. Vision-based global localization and mapping for mobile robots. *IEEE Transactions on Robotics*, 21(3):364–375, 2005. 22

[ST94] J. Shi and C. Tomasi. Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, Seattle, WA, USA, 1994. IEEE. 16, 22

[SZ03] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1470–1477, Nice, France, 2003. IEEE. ii, 38, 39, 42

[TMHF00] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. *Vision algorithms: theory and practice*, pages 153–177, 2000. 11, 18

[TZ00] P.H.S. Torr and A. Zisserman. MLESAC: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000. 23

[VdLH+07] A. Varshavsky, E. de Lara, J. Hightower, A. LaMarca, and V. Otsason. Gsm indoor localization. *Pervasive and Mobile Computing*, 3(6):698–720, 2007. 1

[WBB02] J. Wolf, W. Burgard, and H. Burkhardt. Robust vision-based localization for mobile robots using an image retrieval system based on invariant features. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 359–365, Washington DC, USA, 2002. IEEE. 39

[WKR07] B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1–8, Rio de Janeiro, Brazil, 2007. IEEE. 22

[WRM+08] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose tracking from natural features on mobile phones. In *IEEE/ACM International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 125–134, Cambridge, UK, 2008. IEEE Computer Society. 22

[WSR07] B. Williams, P. Smith, and I. Reid. Automatic relocalisation for a single-camera simultaneous localisation and mapping system. In *2007 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2784–2790, Roma, Italy, 2007. IEEE. 22

[YJHN07] J. Yang, Y.G. Jiang, A.G. Hauptmann, and C.W. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *ACM International Workshop on Multimedia Information Retrieval (MIR)*, pages 197–206, Augsburg, Germany, 2007. ACM. 42, 43

[Zha96] Zhengyou Zhang. M-estimators. `http://research.microsoft.com/en-us/um/people/zhang/INRIA/Publis/Tutorial-Estim/node24.html`, 1996. 14, 15