RWTH AACHEN UNIVERSITY

Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Informatik 8 (Computer Vision)
Fakultät für Mathematik, Informatik und Naturwissenschaften
Prof. Dr. Bastian Leibe

Master Thesis

Keyframe-based Visual-Inertial SLAM With Relocalization Using Stereo Cameras

vorgelegt von

Anton Kasyanov
Matrikelnummer: 351011
September 2016

Erstgutachter: Prof. Dr. Bastian Leibe
Zweitgutachter: Prof. Dr. Leif Kobbelt

Angeleitet von: Dr. Jörg Stückler, M.Sc. Francis Engelmann

Eidesstattliche Versicherung

Anton Kasyanov	351011
Name	Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel

Keyframe-based Visual-Inertial SLAM With Relocalization Using Stereo Cameras

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, September 2016	
Ort, Datum	Unterschrift

Belehrung:

§ **156 StGB: Falsche Versicherung an Eides Statt**
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ **161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt**

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten dementsprechend.

Die vorstehende Belehrung habe ich zur Kentnis genommen:

Aachen, September 2016	
Ort, Datum	Unterschrift

Contents

1 Introduction

Combining the information from visual and inertial sensors had become popular in robotics in recent years, due to the fact that it can be used to accurately estimate the visual odometry or perform Simultaneous Localization And Mapping (SLAM) [LLB+15] [MTK+02] [MR07a] [Dav03]. However, most systems implement just visual odometry, leaving out localization and mapping. Such systems are only locally consistent and localize the camera only within a small limited time frame. On the other hand, robot navigation usually requires a global map to be built with the further possibility to localize the agent with absolute reference. Other applications where good localization is required include Virtual Reality and Augmented Reality. Storing the map with low space requirements and precise relocalization within this map is an area of ongoing scientific research, including this thesis. In order to build a SLAM system, we are going to use the existing visual odometry layer OKVIS [LLB+15], and extend it with an additional layer for loop closure detection, map optimization and further relocalization.

OKVIS is implemented such that it can work with a single or multiple cameras. We are going to use the stereo pair of cameras (Manta G-046, operating at 20 FPS and 780 x 580 resolution) to get better SLAM precision compared to a monocular systems. Furthermore, an industry-level Xsens MTi-G IMU is used in the setup, it provides all the needed inertial information at a decent update rate (200 Hz). These devices will be assembled into a single setup and calibrated. Good calibration is vital for odometry, so it will be an important part of the implementation process.

From the technical point of view, both OKVIS and our SLAM layer are going to be composed into a single Robot Operation System (ROS) node for ease of usage, while different library parts are actually going to run in parallel to achieve the best performance. OKVIS is real-time capable so we will try to keep this and make SLAM real-time capable too. Camera images are going to be first described by OKVIS and then passed to SLAM. Similar images will be matched by DBoW2 [GLT12] and then, the relative transformation will be calculated by RANSAC 3D-2D estimation [KF14]. Such transformations will result in addi-

tional constraints in the pose graph leading to loop closures. The pose graph will be optimized globally with respect to constraints between keyframes, including the ones introduced by loop closure detection. Graph optimization will improve the accuracy in pose estimation. The estimation improvement will be demonstrated in the Evaluation chapter.

After revisiting the same location, even with loop closure and graph optimization, OKVIS will produce redundant keyframes containing the landmarks that were already mapped. Therefore, a clever management of the map and stored keyframes will be performed.

Relocalization is also going to be an important part of the thesis. The processed sequence can be saved as a map on disk. This map can further be reused for relocalization in the world frame during subsequent runs. For example, the home-cleaning robots use the variation of this technology to navigate indoors and remember the map during every run.

The key contribution include:

- **SLAM layer** - use loop closures to perform global trajectory optimization.
- **Relocalization** - find camera pose in the reference frame of the saved map.
- **Continued SLAM in the old map** - combine both maps images and trajectories after relocalization.
- **Hardware setup** - assemble and calibrate to perform our own experiments.

1.1 Thesis structure

In Chapter 2 we are going to briefly overview existing visual-inertial odometry approaches. In Chapter 3, we have put the explanation and overview of the methods that are used throughout the thesis including OKVIS, FAB-MAP, DBOW2, as well as a general SLAM problem overview. Chapter 4 explains the algorithm that was developed in the thesis to perform Keyframe-based Visual-Inertial SLAM. Schemes and plots are included to help the reader better understand the approach. Furthermore, in Chapter 5 we describe the actual implementation of the algorithm. The hardware setup is described as well as the software structure. We have included the software documentation, so our work can be further reused by other students. Finally, in Chapter 6 we evaluate our approach using both EuRoC MAV dataset [BNG+16] and the sequences that were recorded with our own hardware setup.

2 Related work

Many recent SLAM approaches contain two layers. In first layer, the camera position is estimated locally using keyframes. In the second SLAM layer, the global constraints are utilized to optimize the trajectory globally and improve the estimation. Two examples of such approach are ORB-SLAM [MAMT15] and LSD-SLAM [ESC14].

ORB-SLAM was presented in 2015 by Raul Mur-Artal et al. This is a feature-based monocular SLAM that can deliver a real-time performance. It uses bundle adjustment and survival of the fittest approach to build a map. ORB-SLAM features include loop closure detection and relocalization. In ORB-SLAM a simple approach is used for loop closure detection - the similarity between bag-of-words representations. In contrast, LSD-SLAM by Jakob Engel et al. successfully uses a direct (feature-less) SLAM algorithm for a monocular system. Similar methods use image retrieval techniques like DBoW2 [GLT12] and FAB-MAP 2.0 [CN11].

Authors of FAB-MAP 2.0 show that a probabilistic framework can be used to detect the similarity between images in order to detect when the same place is revisited. However, we have also studied an alternative approach DBoW2 [GLT12], which is another algorithm for visual place recognition that was introduced in 2012 by Galvez-López and Juan D. Tardos. It is based on bag of words model (like FabMap) however due to novelties, the computation speed is much faster then other approaches. These methods can build and query a map to find similar images. We evaluate both approaches in this thesis and pick DBoW2 to use in our SLAM system.

Apart from using visual measurement, various approaches add inertial measurement to improve the estimation. However, most of them use filtering or sliding window approaches that provide only locally consistent estimations. This leads to drift accumulating on a global map scale. State-of-art approaches use tight coupling for visual and inertial measurements [MR07a, LLB+15, UESC16, FCDS15].

In "Keyframe-based visual–inertial odometry using nonlinear optimization" [LLB+15] Leutenegger et al. formulate the problem of non-linear optimization with respect to camera and IMU constraints between frames. This is a

big difference compared to many visual odometry methods that use filtering approaches. The optimization window includes only few recent frames. Old frames and keyframes are marginalized out to keep the problem small. This method also detects 3D landmarks and uses them to improve pose estimation. A similar approach is "Direct Visual-Inertial Odometry with Stereo Cameras" by Usenko et al. [UESC16]. The authors propose to use the minimization of the combined photometric and inertial energy functional to simultaneously estimate camera pose, velocity and IMU biases. Moreover, instead of using manually designed keypoints, geometry is estimated in the form of semi-dense depth maps.

Apart from traditional Bayesian approaches, there are some algorithms based on deep neural networks. For example PoseNet [KGC15] allows for relocalization within a known location based only on a single image. However, this approach is not flexible and does not generalize to unseen locations. On the other hand, RGB-D approaches [SGZ+13, GSCI15] use machine learning techniques to detect loop closures and to estimate camera pose.

The problem of visual localization usually includes the process of offline map building and processing [LSCU12, VH12, MSUK14, LSB+15]. For example, approach by Lim [LSCU12] builds an offline map using structure-from-motion. The resulting map consists of 3D interest points. During localization phase, the keypoints are detected in the camera image and their descriptors are matched to the map points. The relative pose is computed using RANSAC 3D-2D matches (e.g. [KF14]). In another approach by Middelberg et al. [MSUK14], the local matching is happening on the mobile device, while the global alignment happens on the remote server.

In this thesis we build the visual-inertial SLAM system that features an online relocalization and mapping in the reference map.

3 Preliminaries

3.1 Simultaneous Localization and Mapping

3.1.1 Introduction

The problem of Simultaneous Localization and Mapping arises when the robot doesn't know its position and it doesn't have a map of the surroundings. Only the measurements $z_{1:t}$ and controls $u_{1:t}$ are available. So SLAM solves both problems simultaneously: building a map and localizing within this map.
There are two forms of SLAM from a probabilistic point of view: *online* and *full* [TBF05]. Online SLAM estimates the posterior over the momentary pose along with the map:

$$p(\mathbf{x}_t, m | z_{1:t}, u_{1:t}) \tag{3.1}$$

Here $\mathbf{x}_t$ is the pose at a time t, m represents a map, $z_{1:t}$ are measurements over time and $u_{1:t}$ are controls (see Figure 3.1). This form of SLAM is called online, because it estimates variables at exactly time t. Most of the algorithms to solve online SLAM are incremental, i.e. they drop past data once it has been processed.

In the *full* SLAM form, the posterior is calculated over the full path $\mathbf{x}_{1:t}$ along with a map, instead of just using the last pose $\mathbf{x}_t$ (see Figure 3.2). The probabilistic formulation of the full form is:

$$p(\mathbf{x}_{1:t}, m | z_{1:t}, u_{1:t}) \tag{3.2}$$

with the only difference of using $\mathbf{x}_{1:t}$ instead of $\mathbf{x}_t$ like in online form. It can be shown, that the online SLAM form is the result of integrating out past poses from the full SLAM:

$$p(\mathbf{x}_t, m | z_{1:t}, u_{1:t}) = \int ... \int p(\mathbf{x}_{1:t}, m | z_{1:t}, u_{1:t}) dx_1 dx_2 ... d\mathbf{x}_{t-1} \tag{3.3}$$

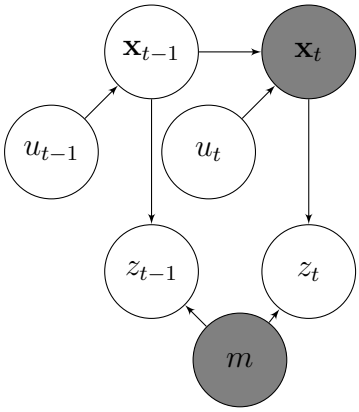


Figure 3.1: Graphical model of online SLAM that uses only current position.

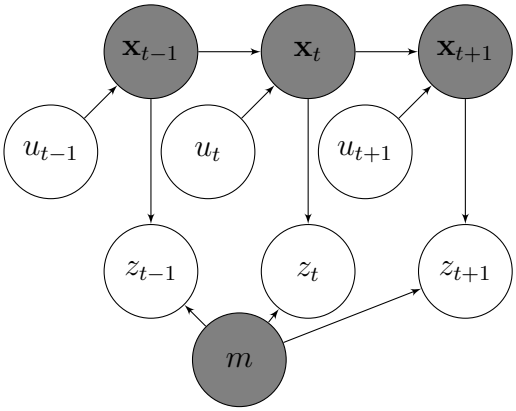


Figure 3.2: Graphical model of full SLAM that uses the full path.

In further subsection we are going to focus on the full SLAM problem. Another characteristic of SLAM is the nature of the estimation problem. SLAM can be viewed as a combination of continuous and discrete components. The continuous component is related to the robot pose estimates and the location of the map landmarks. The discrete component shows the correspondence of the landmarks. When the landmark is detected it should be classified if it is the repeated detection of the same object or it is a new object that was never seen before. It can be useful to make the existence of correspondence variables explicit.

3.1.2 Graph Formulation

The full SLAM problem can be represented as a sparse graph. Such a graph results in a sum of non-linear constraints. Optimization of the set of constraints leads to the maximum likelihood map and a set of robot poses. An example of such a graph can be seen in Figure 3.3. There are five poses $\mathbf{x}_0, ..., \mathbf{x}_4$ and two map features m_1, m_2. Solid lines are *motion* edges that link subsequent poses. Dashed lines are *measurement* edges that link poses to measured features.

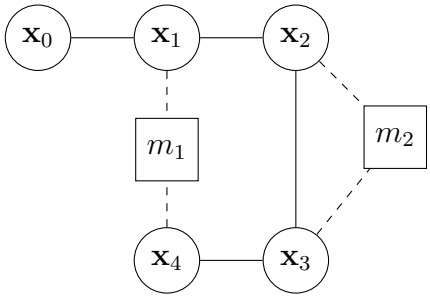


Figure 3.3: Example of the sparse graph SLAM formulation. $\mathbf{x}_t$ represent poses and m_j represent map features. Solid lines link poses and dashed lines link poses with map features.

To build the graph, let's assume we have a set of measurements $z_{1:t}$ and a set of controls $u_{1:t}$. The nodes of the graph are robot poses $\mathbf{x}_{0:t}$ and map features m_j. Edges between the nodes represent soft constraints in the graph. The constraints are equivalent to entries in an information matrix (denoted as Ω) and an information vector (denoted as ξ). Adding an edge to the graph results in local update of Ω and ξ.
The motion constraint between poses $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ is formulated as:

$$[\mathbf{x}_t - g(u_t, \mathbf{x}_{t-1})]^T R_t^{-1} [\mathbf{x}_t - g(u_t, \mathbf{x}_{t-1})] \tag{3.4}$$

where u_t provides an information between relative pose between time $t-1$ and t, g is a kinematic motion model and R_t is the covariance matrix of the motion nose.
The measurement constraint between the pose $\mathbf{x}_t$ and map feature m_j is formulated as:

$$[z_t^i - h(\mathbf{x}_t, m_j)]^T Q_t^{-1} [z_t^i - h(\mathbf{x}_t, m_j)] \tag{3.5}$$

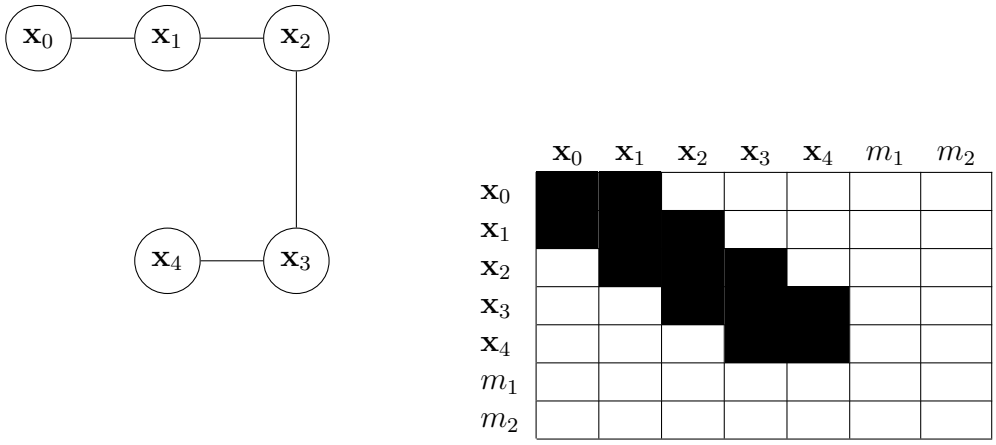
where h is a measurement function and Q_t is the covariance of the measurement noise.
Having all constraints in place, we can form the target function J:

$$\begin{aligned} J = \mathbf{x}_0^T \Omega_0 \mathbf{x}_0 &+ \sum_t [\mathbf{x}_t - g(u_t, \mathbf{x}_{t-1})]^T R_t^{-1} [\mathbf{x}_t - g(u_t, \mathbf{x}_{t-1})] \\ &+ \sum_t [z_t^i - h(\mathbf{x}_t, m_j)]^T Q_t^{-1} [z_t^i - h(\mathbf{x}_t, m_j)] \end{aligned} \tag{3.6}$$

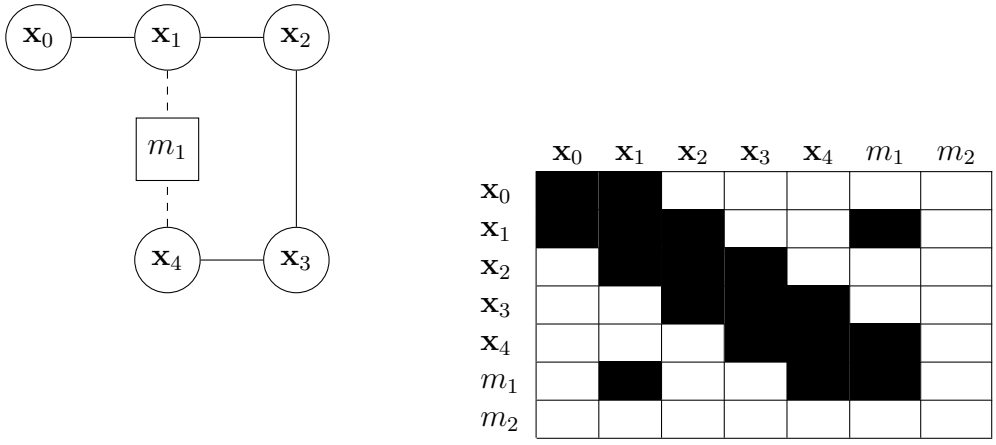
Minimizing J leads to the most likely map and the most likely robot path. Note, the $\mathbf{x}_0^T \Omega_0 \mathbf{x}_0$ term - *anchoring constraint* that initializes the first robot post at $(0, 0, 0)$.
An example of building an information matrix is shown in Figure 3.4. It can be seen, that off-diagonal elements are all zeros with two exceptions:

- Between any consecutive poses $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$ there will be a non-zero element representing the motion link.
- Between map feature m_j and pose $\mathbf{x}_t$ if m_j was observed from position $\mathbf{x}_t$.

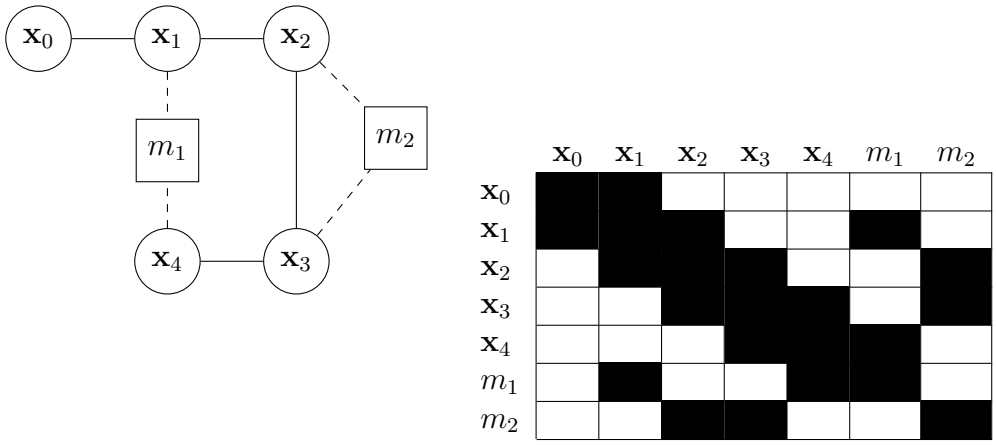
It means that the information matrix is also sparse and only a linear (in terms of graph nodes) number of elements are non-zeros.



Step 1: Initial entries for robot poses $\mathbf{x}_0, ..., \mathbf{x}_4$.



Step 2: Adding edges from $\mathbf{x}_1$ and $\mathbf{x}_4$ to m_1.



Step 3: Adding edges from $\mathbf{x}_2$ and $\mathbf{x}_3$ to m_2.

Figure 3.4: An example of information matrix building. Here white cells are zero elements, black cells are non-zero elements.

3.2 OKVIS

3.2.1 Introduction

OKVIS was introduced in 2015 by Stefan Leutenegger et al [LLB+15]. It is a variation of the full SLAM system. The authors have formulated a probabilistic cost function using reprojection errors of landmarks and inertial terms. The basic notion is same as the full SLAM in the previous subsection, however instead of using the full pose and measurements history, OKVIS uses a limited set of most recent poses. This way the problem is kept tractable and real-time processing is possible. The optimization is limited to a bounded window of keyframes that can be divided in time by arbitrary intervals related by linearized inertial terms. OKVIS supports mono camera systems as well as multiple camera systems.
Since OKVIS doesn't use the full global set of measurements, it can be argued if OKVIS is a SLAM system or a visual-inertial odometry. Generally OKVIS is a visual-inertial odometry system with locally consistent estimations. From now on, our improvements over OKVIS that add globally consistent estimation features are called the SLAM layer.
The processing pipeline starts with Harris corner detector [HS88] and BRISK descriptor [LCS11]. The last pose is propagated using the acquired IMU measurements to obtain an uncertain estimate. A local map with known 3D position of landmarks is available at this point. To obtain correspondences, 3D-2D matching is performed with discarding of outlier points. Next, 2D-2D matching is performed to associate keypoints without known 3D landmark correspondences. Finally, a relative pose estimation via 3D-2D matches and RANSAC filtering [KF14] is performed between the current frame and newest keyframe.

3.2.2 Notation

The variables that are going to be estimated are the robot states at frame k (here we use frame k instead of time t) $\mathbf{x}_R^k$ and landmarks $\mathbf{x}_L$. Here subscript letters define reference frame, W for world reference frame, S for IMU reference frame. A point P represented in frame A is written as ${}_A\mathbf{r}_P$. Quaternion that transforms from frame B to frame A is denoted as $\mathbf{q}_{AB}$. Velocity that corresponds to frame A is written as ${}_A\mathbf{v}$.
The variable $\mathbf{x}_R$ consists of a robot position ${}_W\mathbf{r}_S$, body orientation quaternion $\mathbf{q}_{WS}$, velocity ${}_S\mathbf{v}$, the gyroscope biases $\mathbf{b}_g$ and the accelerometer biases $\mathbf{b}_a$.

Therefore, $\mathbf{x}_R$ is defined as:

$$\mathbf{x}_R = [{}_W\mathbf{r}_S^T, \mathbf{q}_{WS}^T, {}_S\mathbf{v}^T, \mathbf{b}_g^T, \mathbf{b}_a^T]^T \tag{3.7}$$

The full state can be decomposed into pose error state $\mathbf{x}_T = [{}_W\mathbf{r}_S^T, \mathbf{q}_{WS}^T, {}_S]^T$ and speed/bias error state $\mathbf{x}_{sb} = [\mathbf{v}^T, \mathbf{b}_g^T, \mathbf{b}_a^T]^T$.

In practical formulations of the visual odometry, a nonlinear optimization finds the camera poses and landmark positions via minimizing the reprojection errors of the landmarks. Figure 3.5 (left) shows such an example. If we introduce the inertial measurements, they create temporal constraints between poses as well as between speed and IMU bias estimates 3.5 (right). The authors wanted

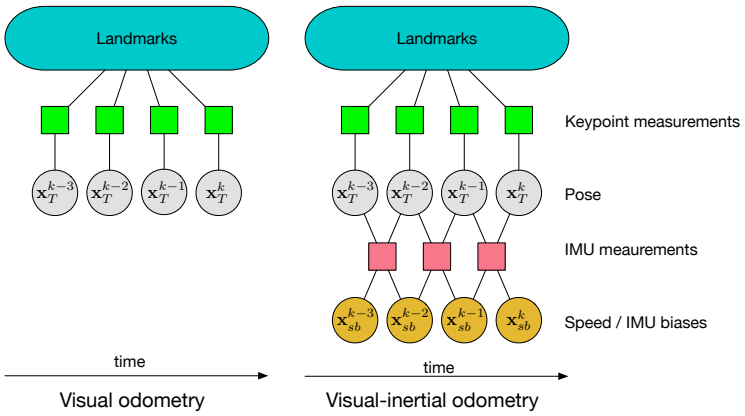


Figure 3.5: The difference between visual odometry (left) and visual-inertial odometry (right).

to formulate the problem as a joint optimization of a cost function $J(\mathbf{x})$ that contains both reprojection and temporal errors:

$$J(\mathbf{x}) = \underbrace{\sum_{i=1}^{I}\sum_{k=1}^{K}\sum_{j\in L(i,k)} \mathbf{e}_r^{i,j,kT}\mathbf{W}_r^{i,j,k}\mathbf{e}_r^{i,j,k}}_{\text{visual}} + \underbrace{\sum_{k=1}^{K-1}\mathbf{e}_S^{kT}\mathbf{W}_S^{k}\mathbf{e}_S^{k}}_{\text{inertial}} \tag{3.8}$$

where $\mathbf{e}_r$ is the reprojection error term, $\mathbf{e}_S$ is a temporal error term from IMU, i is a camera index, k is the camera frame index, the landmarks visible in kth frame and ith camera are represented by $L(i,k)$. Moreover, $\mathbf{W}_r^{i,j,k}$ is the information matrix of the landmark measurement and $\mathbf{W}_S^k$ is the information matrix of the kth IMU error. The reprojection error is adopted from [BFF11] and is formulated as:

$$\mathbf{e}_r^{i,j,k} = \mathbf{z}^{i,j,k} - \mathbf{h}_i(\mathbf{T}_{CiS}^k\mathbf{T}_{SW}^k\mathbf{l}^j) \tag{3.9}$$

where $\mathbf{h}_i(.)$ denotes the camera projection model and $\mathbf{z}^{i,j,k}$ are the measurement image coordinates. The IMU error term is formulated as:

$$\mathbf{e}_S^k(\mathbf{x}_R^k, \mathbf{x}_R^{k+1}, \mathbf{z}_S^k) = \begin{pmatrix} {}_w\hat{\mathbf{r}}_S^{k+1} - {}_w\,\mathbf{r}_S^{k+1} \\ 2[\hat{\mathbf{q}}_{WS}^{k+1} \otimes \mathbf{q}_{WS}^{k+1}] \\ \hat{\mathbf{x}}_{sb}^{k+1} - \mathbf{x}_{sb}^{k+1} \end{pmatrix} \tag{3.10}$$

that is just a difference between the predicted and actual states.

3.2.3 Keyframes and marginalization

OKVIS distinguishes two kinds of frames and it keeps the S most recent frames including the current frame and M keyframes that could have been taken far in the past. The decision if the new frame is a keyframe is simple: if the hull of the projected landmarks covers less then 50% of the image, then the frame is treated like a keyframe.
The initially marginalized error term is built from the first $M+1$ frames $\mathbf{x}_T^k$ with their speed and biases. When a new frame $\mathbf{x}_T^c$ (where c means index of current frame) is inserted into optimization window, the marginalization operation is applied. If the oldest frame $\mathbf{x}_T^{c-S}$ is not a keyframe, it has to be marginalized out with its speed and bias sates. The landmarks of this frame should be dropped. See Figure 3.6.
If $\mathbf{x}_T^{c-S}$ is a keyframe, then simply dropping all the landmarks would be too big information loss, so instead, OKVIS marginalizes out the landmarks that are visible in this keyframe, but not in the newer frames. See Figure 3.7.

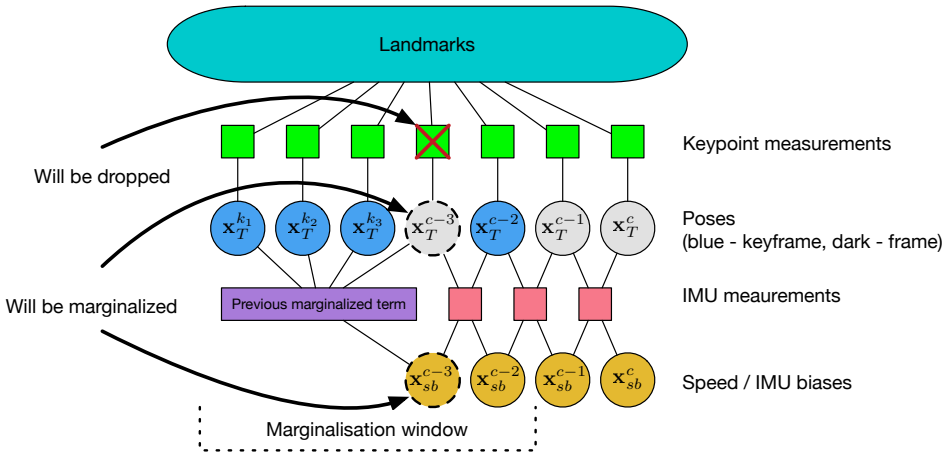


Figure 3.6: The example of the non-keyframe marginalizing. Here a regular frame as well as its speed and bias states are marginalized out. The corresponding landmarks are simply dropped.

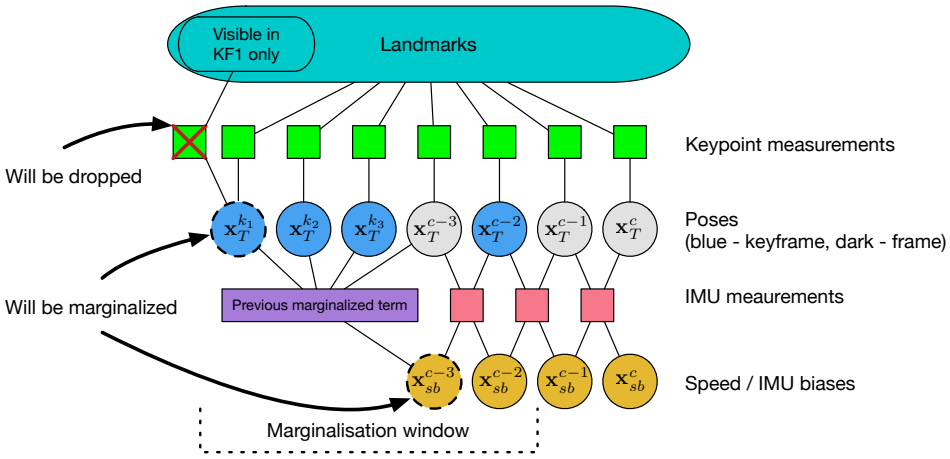


Figure 3.7: In this case, the keyframe frame is marginalized out. The corresponding landmarks that are not seen in the newer keyframes are marginalized out too.

3.3 Image similarity

In order to detect loop closures, we need to be able to reliably tell from the image pair that they were taken in the same location pointing in the same direction. This can be easily solved by comparing complete images pixel by pixel, but any change in the pose, illumination or environment would render this approach not usable. The image similarity measure has to be robust to such changes (see Figure 3.8). We have compared two existing approaches: *FAB-MAP 2.0* [CN11] and *DBoW2* [GLT12]. They both claim to be able to solve the task of detecting the similarity between images from different points of time.

Figure 3.8: The example of two images that are not exactly the same, but should be matched as similar to perform the loop closure.

3.3.1 FAB-MAP 2.0

FAB-MAP [CN08] was originally developed in 2008 by Mark Cummins and Paul Newman. The main motivation was the problem of of appearance-based place recognition at very large scale. Later, in 2009 the authors proposed an updated version called FAB-MAP 2.0 [CN11].
The main data structure that is used is a bag-of-words model. First, a vocabulary is created by clustering feature vectors for all images in the training set. Second, features from the test images are associated with the visual words from the vocabulary. Authors use SURF feature descriptor [BTVG06], but in general, other feature descriptors can be used.

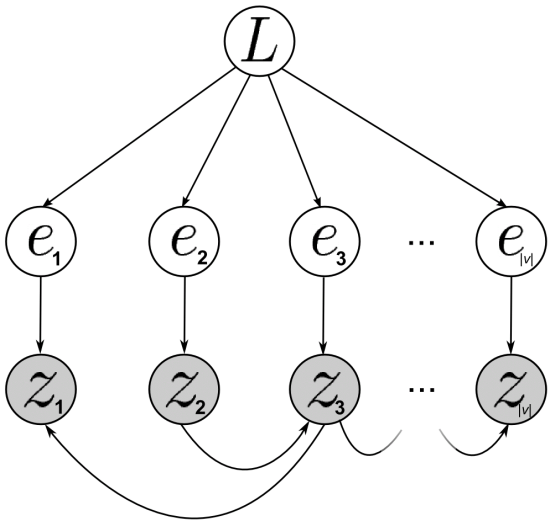


Figure 3.9: Graphical model of FAB-MAP system. Here locations L generate existence variables e. Observed z_i are conditioned on e_i via detector model and in each other by the Chow Liu tree. Image credit: [CN11].

The system builds a probabilistic model over the bag-of-words. An observation (a camera image in our case) is denoted as $Z_k = \{z_1, ..., z_{|v|}\}$, where k represents the time when the observation was captured, $|v|$ is the size of the vocabulary and z_i is a binary variable referring to presence of the i^{th} word in the observation. The world is modeled as a set of discrete and disjoint locations $\mathcal{L}^k = \{L_1, ..., L_{n_k}\}$. For each location there is an appearance model:

$$\{p(e_1 = 1|L_i), ..., p(e_{|v|} = 1|L_i)\} \tag{3.11}$$

Here e_i represents feature existence. A location can correspond to some real-world area, like a room. A detector model associates e_i with z_i:

$$\mathcal{D} : \begin{cases} p(z_i = 1|e_i = 0), & \text{false positive probability} \\ p(z_i = 0|e_i = 1), & \text{false negative probability} \end{cases} \tag{3.12}$$

The authors note, that in most cases visual words occurrences are correlated. For example, if there are car wheels in the image, then not far there are also car doors. Such dependencies are learned by a tree-structured Bayesian network via Chow Liu algorithm [CL68]. This algorithm delivers the optimal approximation to the joint distribution over word occurrence within the space of tree-structured networks (see Figure 3.9).

Having a probabilistic appearance model, mapping and localization can be implemented as a recursive Bayes estimation problem. Then a probability density function over location up to time k is given by:

$$p(L_i|\mathcal{Z}^k) = \frac{p(Z_k|L_i, \mathcal{Z}^{k-1})p(L_i|\mathcal{Z}^{k-1})}{p(Z_k|\mathcal{Z}^{k-1})} \tag{3.13}$$

Where $p(L_i|\mathcal{Z}^{k-1})$ is a location prior (obtained from previous estimate by transforming the prior using a motion model), $p(Z_k|L_i, \mathcal{Z}^{k-1})$ is the observation likelihood and $p(Z_k|\mathcal{Z}^{k-1})$ is a normalization term.

To define the observation likelihood, authors assume that current and past observations are independent, and use the Chow Liu model, giving:

$$p(Z_k|L_i) = p(z_r|L_i) \prod_{q=2}^{|v|} p(z_q|z_{p_q}, L_i) \tag{3.14}$$

where z_{p_q} is a parent of z_q in the Chow Liu tree and z_r is a root. Further expansion leads to:

$$p(z_q|z_{p_q}, L_i) = \sum_{s_{e_q} \in \{0,1\}} p(z_q|e_q = s_{e_q}, z_{p_q}) p(e_q = s_{e_q}|L_i) \tag{3.15}$$

After computing the pdf over locations, data association can be made. The observation Z_k can be used to initialize a new location or to update the model of an exiting one. Each component of the model is updated as follows:

$$p(e_i = 1|L_j, \mathcal{Z}^k) = \frac{p(Z_k|e_i = 1, L_j)p(e_j = 1|L_j, \mathcal{Z}^{k-1})}{p(Z_k|L_i)} \tag{3.16}$$

The main difference between versions 2.0 and original FAB-MAP is that in 2.0 authors wanted to improve the scalability. For this, they use inverted index retrieval architecture. It extends applicability by more than two orders of magnitude in scale.

3.3.2 DBoW2

DBoW2 [GLT12] is another algorithm for visual place recognition that was introduced in 2012 by Galvez-López and Juan D. Tardos. It is based on bag of words

model (like FAB-MAP) however due to novelties, the computation speed is much faster then other approaches. A bag of words discretizes a binary space and uses a direct index next to an inverse index.

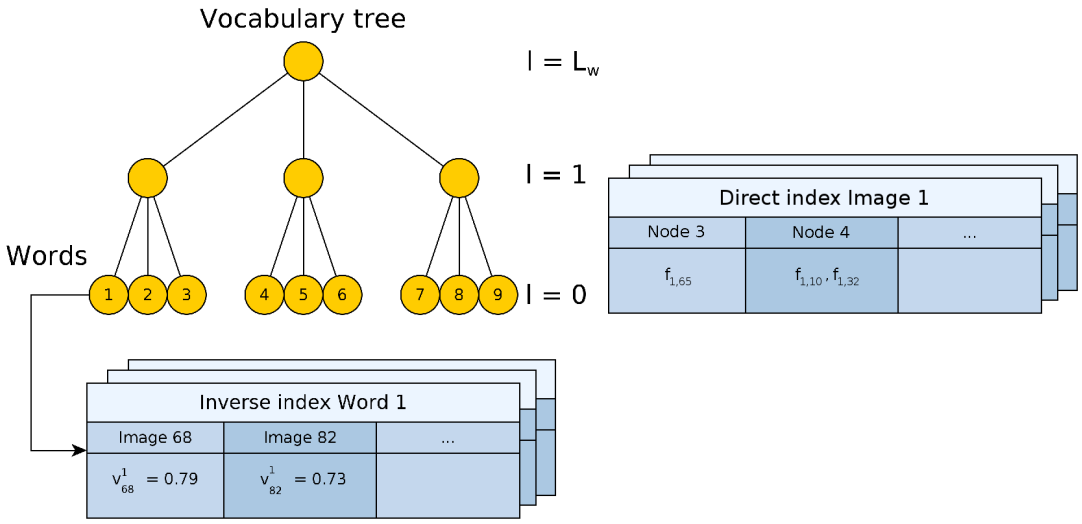


Figure 3.10: DBoW2 vocabulary and indexes example. Here the leaf nodes are the vocabulary words. The inverse index contains the weight of the words in the images. The direct index contains the features and their associated nodes. Image credit: [GLT12].

In order to detect revisited places, an image database is built using a hierarchical bag of words. The database also includes both direct and inverse indexes (see Figure 3.10). As usual, the visual vocabulary is trained offline by discretizing the descriptor space into W visual words. Hierarchical vocabularies use a tree as its structure. The tree is formed by clustering features associated with each level, resulting in W leaves that are the words of the vocabulary. Each word has a different weight that is calculating using term frequency - inverse document frequency (tf-idf). To convert an image I_t taken at time t into a bag-of-words vector v_t, the descriptors traverse the tree from top to leaves by selecting nodes that minimize the distance. Then the similarity between two vectors can be calculated as a L_1 score:

$$s(v_1, v_2) = 1 - \frac{1}{2}\left|\frac{v_1}{|v_1|} - \frac{v_2}{|v_2|}\right| \tag{3.17}$$

Apart from the bag of words, there is also an inverse index. For every visual word w_i, the inverse index stores a list of images where this word can be found. This way, it is fast to query for images that have some visual words in common. The inverse index is updated when a new image is added and is accessed when images are being searched in the database.

When a new image I_t is taken, it is converted into a bag-of-words vector v_t. This vector is searched in the database resulting in candidate pairs $< v_t, v_{t_1} >$,..., $< v_t, v_{t_j} >$ sorted by their normalized score:

$$\eta(v_t, v_{t_j}) = \frac{s(v_t, v_{t_j})}{s(v_t, v_t - \delta_t)} \tag{3.18}$$

Images that are close in time are grouped together into *islands* and are treated as a single match. This is done to remove the competition between similar images during the database query. Such matches $< v_t, v_{t_{n_i}} >$,...,$< v_t, v_{t_{m_i}} >$ are converted into $< v_t, V_{T_i} >$ if the gaps in timestamps are small. The ranking for the islands look as following:

$$H(v_t, V_{T_i}) = \sum_{j=n_i}^{m_i} \eta(v_t, v_{t_j}) \tag{3.19}$$

The authors have compared their approach with FAB-MAP 2.0. As it appeared, both approaches show comparable precision and recall (see Table 3.1). The important difference, however, is that DBoW has much lower execution time. Note that the number of images for Malaga6L dataset is different, because FAB-MAP 2.0 performed better using lower frame frequency. We have performed our own evaluation in the Evaluation chapter.

Table 3.1: DBoW2 vs FAB-MAP 2.0. Table credit: [GLT12].

Method/Dataset	Images	Precision (%)	Recall (%)
DBoW2 Malaga6L	869	100	74.75
FAB-MAP 2.0 Malaga6L	462	100	68.52
DBoW2 CityCentre	2474	100	30.61
FAB-MAP 2.0 CityCentre	2474	100	38.77

4 Keyframe-based Visual-Inertial SLAM

This chapter describes which algorithms are implemented in this thesis and how they are used.

4.1 SLAM

The main focus of this thesis lies on creating a Simultaneous Localization and Mapping (SLAM) on top of an existing visual-inertial library OKVIS.

4.1.1 Motivation

The biggest difference between the two is that OKVIS is considering only a short temporal local window during optimization, while SLAM is designed in a way that enables to use a global view of the optimization. The short temporal local window usually leads to errors accumulating over time. Thus, having a global state, it is possible to use clues from the past to improve the trajectory estimation. Namely, two important features of SLAM are developed in this thesis:

- Loop closures
- Relocalization

SLAM keeps track of visited locations and is able to detect if the same location is revisited. With the help of this knowledge it is possible to see if the trajectory has accumulated spacial drift. If it does, then extra factors that link similar images together in the global keyframe graph can optimize the trajectory and reduce the accumulated error (see Section 4.2.1 for details). This approach is usually used when the agent moves in loops, this process is called *Loop Closure*. This technique is especially useful, when the agent moves on long trajectories that eventually close.
Once we have a complete optimized trajectory along with the keyframe image

data as a map, this map can be reused during subsequent SLAM runs. For example, the robot can first record a big discovery sequence in a new environment, and then save the sequence as a map. The map can be reused while navigating in local areas to provide the robot's pose in the global map frame. This process is usually referred to as *Relocalization*. Note that after establishing the map correspondence, the SLAM continues using images from both new and old maps.

4.2 Loop closures

4.2.1 Graph optimization

We are using the library called *gtsam* to build and optimize the pose graph. The vertices of the graph are poses estimated by OKVIS. Essentially these vertices are subject to optimization (in *gtsam* terminology they are called *Values*). We distinguish two types of edges (*Factors*) between the poses:

- Relative pose between subsequent frame poses $\mathbf{T}_{k-1}^{k}$, that is calculated from OKVIS
- Relative pose between keyframes that were detected as a loop closure $\mathbf{T}_{i}^{j}$. The relative transformation between the two is taken from RANSAC 3D-2D matches algorithm.

Note, that if we just use OKVIS poses, then the optimization would not change the graph, because OKVIS already provides the optimized results.

Having such a graph structure we are able to account for loop closures and improve the estimation. An example of the graph can be seen in Figure 4.1.

Now we can formalize the described above as a cost function that should be minimized:

$$J(\mathbf{x}) = \underbrace{\sum_{k=1}^{K} ||\mathbf{x}_k - \mathbf{T}_{k-1}^{k}\mathbf{x}_{k-1}||_{\Sigma_k}^2}_{\text{OKVIS relative constraints}} + \underbrace{\sum_{(i,j)\in S} ||\mathbf{x}_j - \mathbf{T}_{i}^{j}\mathbf{x}_i||_{\Sigma_{ij}}^2}_{\text{loop closure constraints}} \tag{4.1}$$

where k is a camera frame index, $\mathbf{x}_k$ is the SLAM estimated camera pose, $\mathbf{T}_i^j$ is the relative transformation between pose i and j, S is a set of similar image correspondences. Each information matrix Σ_k is a diagonal matrix with its non-zero elements being proportional to images overlay percentage, i.e. the bigger the size of the shared part between two images, the higher the correlation between their poses. The overlay percentage is computed by OKVIS to make a keyframe creation decision. The relative transformations between subsequent poses T_i^{i+1} are extracted from OKVIS estimated trajectory. The relative transformations

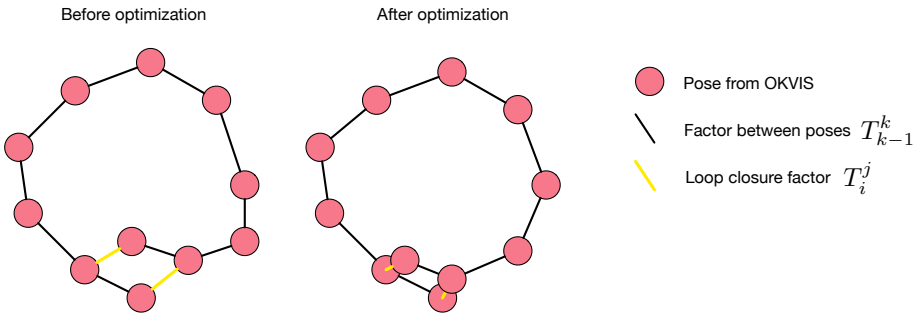


Figure 4.1: An example of gtsam pose graph. Black edges are calculated from OKVIS estimated poses, yellow edges are the result of loop closure detection. Before the optimization, starting and finishing positions are not quite the same. After optimization with respect to loop closure constraints, the starting and finishing positions are well aligned.

between frames that were detected as similar T_i^j are calculated using RANSAC 3D-2D relative pose estimation. The norm in Equation 4.1 is defined as follows:

$$||\mathbf{a}||_{\Sigma}^{2} = \mathbf{a}^{T}\Sigma^{-1}\mathbf{a} \tag{4.2}$$

This way we minimize the difference of the resulting graph from OKVIS estimation while also satisfying the spatial constraints from similar image matches (loop closure constraints). The minimization of $J(\mathbf{x})$ happens using *gtsam* library.

4.2.2 Algorithm

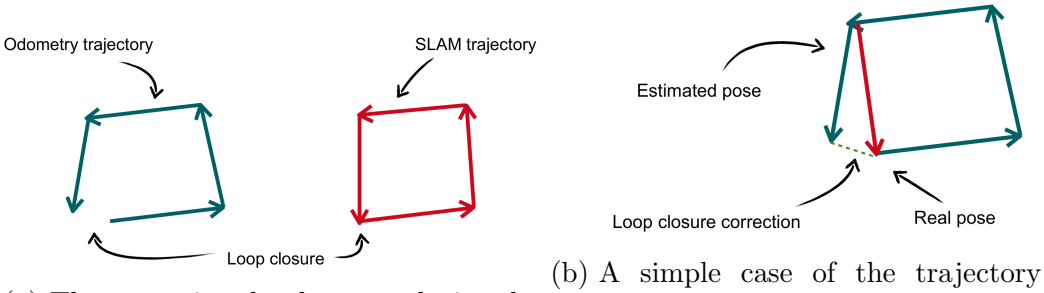
The loop closure is an essential part of the SLAM layer. The optimization of the global trajectory happens when the same area is being revisited (Figure 4.2a and Figure 4.2b). How do we detect that the agent has been here before? We compute the similarity between the current image frame and all frames in the past. If the similarity measure is high enough, then we assume that these landmarks were seen before.

In order to compute the similarity we use two steps:

- DBoW2 - *Bags of Binary Words for Fast Place Recognition in Image Sequences* [GLT12] for initial image comparisons.
- Relative RANSAC with 3D-2D matches [KF14] to geometrically verify DBoW2 and to compute 3D transformation between image poses.

It is important to mention that SLAM uses two notions for the frames - *the frame* and *the keyframe*. Namely, the keyframe is a frame that is detected as a key by OKVIS. The criterion is how many keypoints are shared with the last keyframe. If less then 60% then OKVIS makes this frame a new keyframe. From the SLAM side, the regular frame poses are only added to the pose graph, no other processing is being made. On the other hand, the keyframes are checked for image similarity and loop closures. You can see the references to *frame id* and *keyframe id*, the association between these two is stored in the *meta.txt* file of the map.
The problem with computing the image similarity is that, in most cases, the current frame looks much like the one before it, but this is not what we need for the loop closure detection (the agent hadn't moved out of the area). This problem is solved by tracking if there are frames in between the two that have completely different landmarks and keypoints. If they do have such landmarks, then we can safely assume that the agent had moved out of the area and then back again.
Summing up all the above, we can formulate the general loop closure algorithm:

```
1: procedure ConsumerLoop                          ▷ Runs while there is data
2:     for OKVIS has processed frame do
3:         frame = getFrameFromOkvis()
4:         if not frame.isKeyframe then                          ▷ Regular frame
5:             addPoseToPoseGraph(frame.pose)
6:             continue
7:         end if
8:         addToDBoWIndex(frame.image)
9:         match = findDBoWMatch(frame.image)
10:        if match is not empty then
11:            tf = computeRelativeTransform(frame.image, match.image)
12:            if tf is not empty then
13:                addExtraEdgeToPoseGraph(tf)          ▷ Loop closure detected
14:                optimizePoseGraph()
15:            end if
16:        end if
17:    end for
18: end procedure
```



(a) The correction that happens during the loop closure.

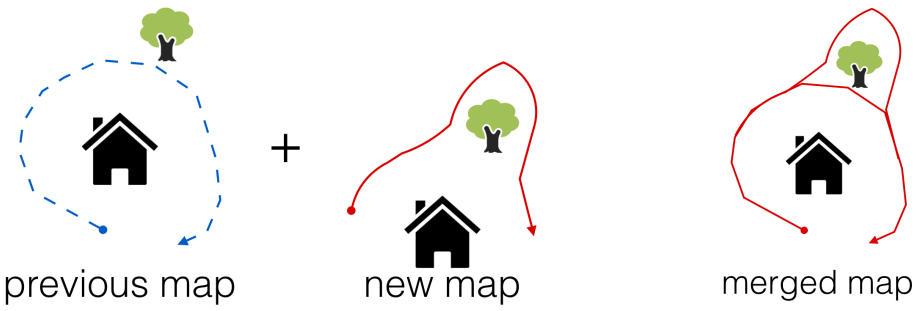
(b) A simple case of the trajectory to showcase the difference between odometry without loop closure detection and SLAM.

Figure 4.2: Loop closure graphical examples.

4.3 Relocalization

4.3.1 Motivation

There are cases, when knowing the agent's position in relation to the starting point is not enough. Sometimes, it is more useful to know the position in relation to some already built map. For example, during a first run, the robot can explore the whole environment and build a map. Any subsequent runs, the robot can just reuse the already stored map to relocalize itself in the map. A cleaning robot can build a map of the apartment and if the cleaning is interrupted, it can use the position in the map to return and finish the job. What we wanted to achieve is a mechanism to save the map and relocalize within this map if needed. Such functionality should run next to SLAM and OKVIS without interrupting or changing them. An illustration can be seen in Figure 4.3. Here, the second run trajectory is relocalized within the saved map and these trajectories are aligned together. After the trajectories are aligned and the maps are fused together, the SLAM continues as usual, making use of the images from both maps.



(a) Two initial maps (trajectories) before relocalization.

(b) One merged map that uses shared landmarks.

Figure 4.3: Example of how relocalization works.

4.3.2 Algorithm

So the general idea is to find a shared trajectory piece and align both trajectories together. This way, we can have the position in the map coordinate frame. In the notation we distinguish new map ($\mathbf{x}^{new}$) and old saved map ($\mathbf{x}^{old}$). The camera frame indexes are k for $\mathbf{x}^{new}$ and p for $\mathbf{x}^{old}$; The following steps were implemented to archive this:

- New trajectory is started in the origin $(0, 0, 0)$ point, like any other OKVIS trajectory (Figure 4.5 *1)*). The optimization has the same form as in regular SLAM:

$$J(\mathbf{x}^{new}) = \sum_{k=1}^{K} ||\mathbf{x}_k^{new} - \mathbf{T}_{k-1}^{k}\mathbf{x}_{k-1}^{new}||_{\Sigma_k}^2 + \sum_{(i,j)\in S^{new}} ||\mathbf{x}_j^{new} - \mathbf{T}_i^j\mathbf{x}_i^{new}||_{\Sigma_{ij}}^2 \quad (4.3)$$

- Every keyframe is checked for the match with the old map's keyframes. DBoW2 and RANSAC are used the same way as for loop closure detections. If the similar image is found in the map, this keyframe association is saved as a relocalization hypothesis H (Figure 4.5 *2)*):

$$H_i = [\mathbf{T}_k^p] \quad (4.4)$$

  where k and p is a pair of similar images from both maps.

- If multiple hypotheses exist for neighbor keyframes, they are merged together into single hypothesis (Figure 4.5 *3)*).

$$\begin{aligned} \text{if } H_i = [\mathbf{T}_k^p], H_j = [\mathbf{T}_{k+1}^{p+1}] \\ \text{then } H_i = [\mathbf{T}_k^p, \mathbf{T}_{k+1}^{p+1}], H_j \text{ is dropped} \end{aligned} \quad (4.5)$$

- Once there is a long enough hypothesis that consists of four keyframe associations, the relocalization transformation between two maps $\mathbf{T}_{maps}$ is calculated. This transformation moves new trajectory points from $(0, 0, 0)$ origin, to the old map's origin (Figure 4.5 *4)*).

$$\begin{aligned} H_i &= [\mathbf{T}_k^p, ..., \mathbf{T}_{k+3}^{p+3}] \\ \mathbf{T}_{maps} &= \mathbf{T}_{k+3}^{p+3} \end{aligned} \quad (4.6)$$

- Internally, two trajectories are merged together including keyframe graphs and images. This allows to reuse old map's images to create additional factors in the graph. The full optimization is now formulated as:

$$\begin{aligned} J(\mathbf{x}) = &\sum_{k=1}^{K} ||\mathbf{T}_{maps}\mathbf{x}_k^{new} - \mathbf{T}_{maps}\mathbf{T}_{k-1}^{k}\mathbf{x}_{k-1}^{new}||_{\Sigma_k}^2 \\ &+ \sum_{p=1}^{P} ||\mathbf{x}_p^{old} - \mathbf{T}_{p-1}^{p}\mathbf{x}_{p-1}^{old}||_{\Sigma_p}^2 \\ &+ \sum_{(i,j)\in S} ||\mathbf{x}_j - \mathbf{T}_i^j\mathbf{x}_i||_{\Sigma_{ij}}^2 \end{aligned} \tag{4.7}$$

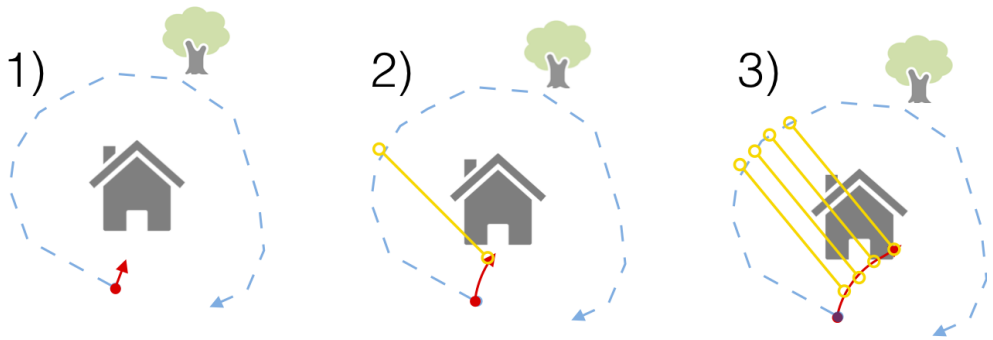
  where $\mathbf{x}$ is a set of all poses from both maps and S is the set of similar image pairs from both maps.

- At this point two trajectories act as a single one, so if subsequent images are similar to images in the old map, loop closure mechanism automatically creates additional links, further improving trajectories alignment. (Figure 4.5 *5)*)

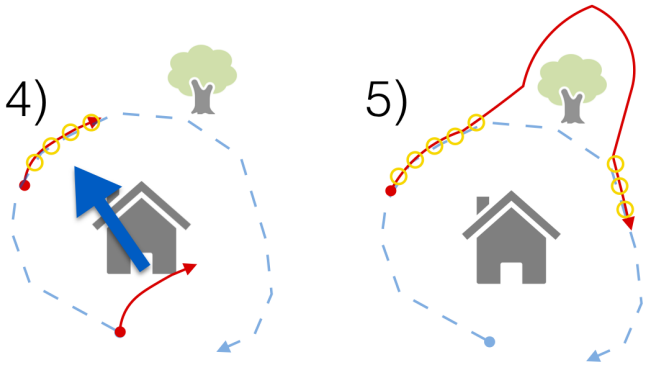
The pseudo-code algorithm is in Figure 4.4.

```
1: procedure TryToRelocalize(frame)      ▷ Called for each new keyframe
2:     match = findDBoWMatch(frame.image, old_map)
3:     if match is not empty then
4:         tf = computeRelativeTransform(frame.image, match.image)
5:         if tf is not empty then
6:             insertToHypothesis(tf)
7:         end if
8:         h = longestHypothesis()
9:         if h.size == 4 then
10:            T_maps = computeRelativeTransform(h)
11:            old_map.appendMap(new_map, T_maps)
12:        end if
13:    end if
14: end procedure
```

Figure 4.4: Relocalization algorithm.



(a) *1)* Initial points, independent from old map. *2)* First image correspondences. *3)* Hypothesis building.



(b) *4)* Translating to old map's coordinate frame. *5)* Merging trajectories and alignment improvements.

Figure 4.5: Stages of relocalization. Blue trajectory is old map ($\mathbf{x}^{old}$). Red trajectory is a new trajectory ($\mathbf{x}^{new}$) that needs to be relocalized within the old map. Yellow shows similar images that build up hypothesis (H_i).

5 Implementation

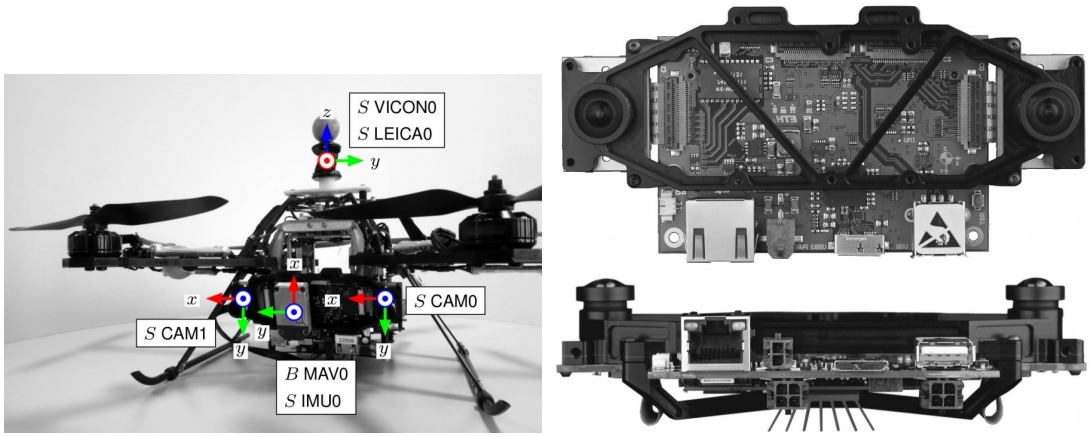
This chapter will describe the hardware and software setup that was assembled to perform visual-inertial odometry and SLAM.

5.1 Hardware

In order to perform visual-inertial odometry, specific hardware is required. First, one or multiple cameras are needed to capture the landmarks and depth. The number of cameras influences the quality of the odometry estimation - one camera brings depth uncertainty, while three cameras and more introduce a lot of complexity in the calibration, synchronization, and computation. Therefore, the setups with two cameras are used in the most recent approaches [LLB+15] [MR07b] . To improve the estimation of the body position in 3D space, the cameras are aided with an IMU that contains an accelerometer and a gyroscope. This way, cameras and the IMU together deliver a better quality of the odometry estimation.
As an example of existing setups, let's have a look at the EuRoC hardware, that was used to record EuRoC MAV dataset [BNG+16] and to evaluate OKVIS:

- Aptina MT9V034 global shutter, WVGA monochrome, 2x20 FPS (Figure 5.1b)
- MEMS IMU (ADIS16448, angular rate and acceleration, 200 Hz)
- Asctec Firefly hex-rotor helicopter to carry the setup (Figure 5.1a)

All components are intrinsically and extrinsically calibrated. The cameras and the IMU are synchronized. Note, that placing the setup on the drone allowed to perform faster and more varied movements compared to the ground robots or manual recording.



(a) Asctec Firefly hex-rotor helicopter used during dataset collection.

(b) Visual-Inertial sensor unit and cameras (carried by the helicopter).

Figure 5.1: EUROC hardware for visual-inertial odometry. Images credit: [BNG+16].

We decided to assemble our own hardware setup in order to be able to conduct loop closure and relocalization experiments. The following components were selected:

- Manta G-046 cameras, operating at 20 fps and 780 x 580 resolution.
- Xsens MTi-G IMU, operating at 200 Hz.

The cameras were attached to a metal plate with the IMU located between them. In order to perform the capture triggering of the cameras by the IMU, we connected the transistor from the SyncOut pin of the IMU to the trigger pin of the cameras. To transfer the captured data, the cameras use GigA technology that allows to use a LAN cable as a transport. To connect together both cameras and a laptop that is used for recordings, a 1 Gbit switch was used. The IMU is connected to the laptop via Rs232 serial USB connector. The power is delivered to the setup from a battery through the specially soldered power cable. This way, the setup is power-independent, allowing to make the recordings in different environments both indoors and outdoors. The whole assembly can be seen at Figure 5.2.
Later during development, we needed a way to start/stop recording while the laptop was in a backpack that was used to carry the setup for recording. To solve this problem we have used *Logitech Wireless Gamepad F710* and a simple Python wrapper (*joystick_ctl.py*) that checks button presses and starts or stops the recording. Also, this way we are able to make checkpoints, i.e. press the specific button every time the person visits some location, after multiple visits we can check how close the checkpoints are in the processed sequence (see Evaluation chapter).

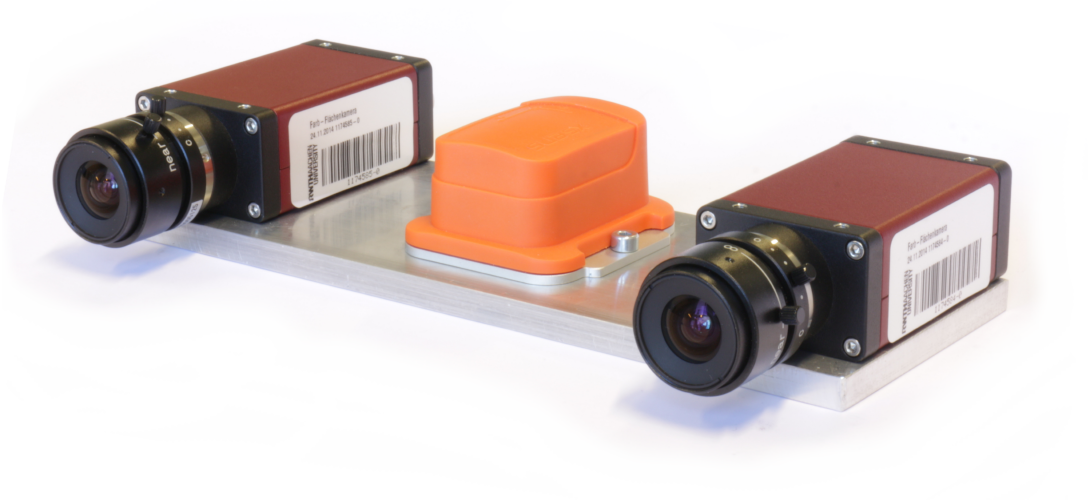


Figure 5.2: Our assembled setup.

5.2 Calibration

A good calibration is one of the main prerequisites for visual-inertial odometry. The following have to be calibrated:

- Camera intrinsics (distortion coefficients, focal length, principal point).
- Camera extrinsics (poses relative to the IMU and to each other).
- IMU intrinsics (noise model, biases).
- Time synchronization between all sensors.

The authors of OKVIS recommend to use Kalibr calibration toolbox [FRS13], so we have followed their advice. For the calibration, the repeated special pattern Aprilgrid was used with the size 0.8 m by 0.8 m.
The calibration process consists of two major parts. First, cameras are calibrated intrinsically and extrinsically. The camera system is fixed and only the calibration target is moved in front of the cameras in the different directions to cover the whole camera image. The recording and subsequent cameras calibration is initiated as following:

```
1 cd kalibr_workspace
2 rosbag record /cam_left/image_raw /cam_right/image_raw -O
    static.bag
3 kalibr_calibrate_cameras --bag static.bag --topics
    /cam_left/image_raw /cam_right/image_raw --models
    pinhole-radtan pinhole-radtan --target grid.yaml
```

Note, that Kalibr outputs pdf and yaml files as a result of calibration. The data from the yaml file has to be copied to the OKVIS/SLAM configuration file.

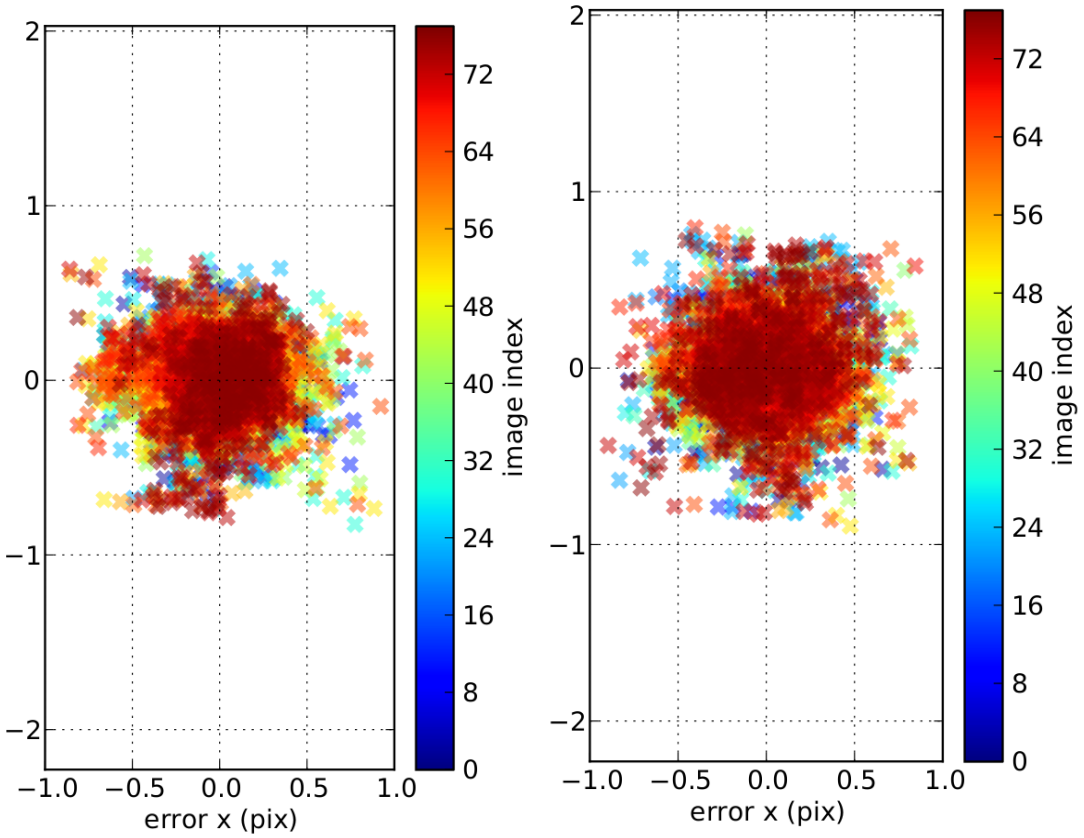
Second, the IMU to cameras calibration. In this case, the calibration target is stationary, while the IMU-cameras system is moved around (having the target in the view port). During the calibration process, it is important to ensure good and even illumination of the calibration target and to keep the camera shutter times low to avoid excessive motion blur.
The recording and subsequent IMU-cameras calibration is initiated as following:

```
1 cd kalibr_workspace
2 rosbag record /cam_left/image_raw /cam_right/image_raw
    /cam_left/exposure /cam_right/exposure /imu/data
    /imu/analog_in1 -O dynamic.bag
3 python bag_adjust.py dynamic.bag dynamic_adj.bag
4 kalibr_calibrate_imu_camera --cam <cameras calibration
    result>.yaml --target grid.yaml --imu imu0.yaml --bag
    dynamic_adj.bag
```

Note, that Kalibr outputs pdf and yaml files as a result of calibration. The data from the yaml file has to be copied to the OKVIS/SLAM configuration file.

5.3 Synchronization

Unfortunately, the time synchronization is not covered by Kalibr. Generally speaking, there are two options. One is a software synchronization, when the timestamps are captured and adjusted by the host system. Another is a hardware synchronization. We decided to perform a hybrid combination of both.
The cameras are running at 20 Hz, while the IMU is running at 200 Hz. We need to be sure that every 10th measurement of the IMU triggers the image capturing in the cameras. This way, a constant measure density is maintained. The IMU has a so called SyncOut pin that can be set to high every few measurements (in our case 10). This signal was connected to the camera's SyncIn pin that triggers the image acquisition process. Since the SyncOut current is not strong enough to power two SyncIn pins, we have used a transistor in between that keeps the signal stable.
However, the cameras and the IMU do not share a single clock, so we can only use the timestamps of the host system (ROS) that represent the time when the measurement was received and processed, but not the real device capture time. On the other hand, we know that inner clocks of the devices are precise enough and that time deltas are 5 ms (δ^{IMU}) and 50 ms (δ^{C}) for the IMU and the cameras respectively. This way we can use the system timestamps with fixed deltas to adjust the IMU and camera timestamps. The important question is how do we



(a) Left camera

(b) Right camera

Figure 5.3: Reprojection errors after cameras calibration

know which IMU measurement has triggered the camera. For this we have created a loop connection from IMU SyncOut to IMU SyncIn, then in the IMU driver we can read the state of SyncIn pin and add it as data to the IMU packages. This way we can find out exactly which IMU measurement is the trigger ($T_k^{IMU\_trig}$) that triggered the camera.
Having relative time deltas and the knowledge of which IMU measurement is a trigger, we can align camera and IMU timestamps. But in practice, the camera starts image acquisition a bit later then the IMU trigger due to trigger latency (l^C), so this latency is added as a constant (28600 ns). Moreover, it makes sense to set camera timestamp not to acquisition beginning, but to the middle of the exposure time to get more accurate results [NRB+14]. For that, we also record the exposure time from the camera (e_k) and add half of the exposure time during adjusting. Summing all above, the formulas for time synchronization are:

$$\begin{aligned} T_i^{IMU} &= T^{start} + i\delta^{IMU} \\ T_k^C &= T_k^{IMU\_trig} + l^C + 0.5e_k \end{aligned} \tag{5.1}$$

where T_i^{IMU} is the ith IMU adjusted timestamp, T^{start} is the system timestamp for the first received measurement, T_k^C is the kth camera adjusted timestamp. $T_k^{IMU\_trig}$ shows the timestamp of the IMU measurement that has triggered kth camera measurement. The time synchronization was implemented as a separate Python script that takes a recorded sequence in ROS bag format and produces its synchronized version.

5.4 Integration with OKVIS

The SLAM layer should be integrated with OKVIS because SLAM uses OKVIS optimization results to further improve the trajectory estimation. Earlier in this thesis development it was decided not to change OKVIS library in any significant way. As a consequence, the bundle of OKVIS with SLAM layer is fully compatible with the original OKVIS library. Only minor additions were made to the source code.
The bundle still runs as a single process or a ROS node. OKVIS itself is multi-threaded via C++11 threads and runs as a single ROS node that can be:

- Synchronous - waits for the visual and inertial data to be fully processed
- Asynchronous - skips the data, if needed, to work in real-time

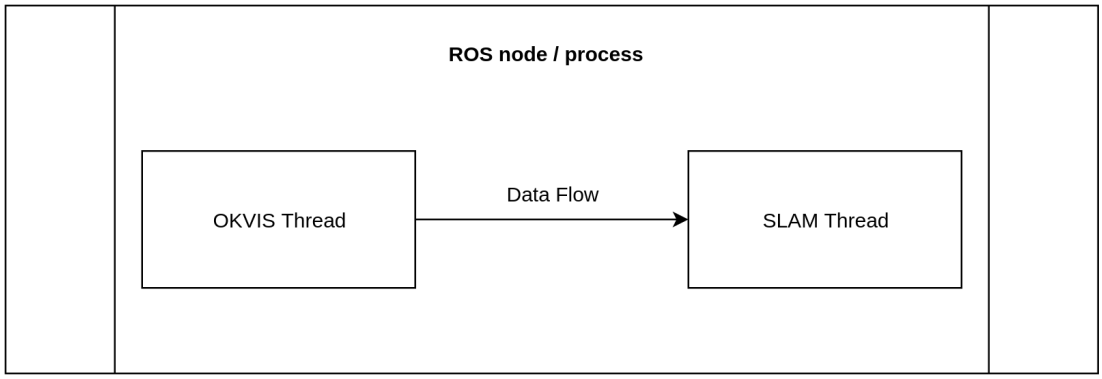


Figure 5.4: The diagram of how threads are related within a process.

We have picked an architecture that leaves the running structure unchanged - the bundle still runs as a single ROS node in one of the two modes. What changes is that SLAM is running in parallel with OKVIS using an extra C++11 thread (see Figure 5.4). To function properly, SLAM requires images from the camera (we use the left one) and the pose that is optimized by OKVIS. Moreover, we are interested in the poses that have been marginalized and moved out of OKVIS' optimization window. This way we are sure that the pose is fully optimized and is not going to change in the future. Such frames are put into a thread-safe queue in the OKVIS thread and are used further in the SLAM thread that blocks and

does nothing until a keyframe arrives in the queue (see Figure 5.5). Note, that SLAM itself doesn't block OKVIS or influence it in any way, the communication is one-way, allowing OKVIS to run at the same speed as before (on a multi-core processor). We wanted to make sure that if OKVIS can run in real-time, then SLAM doesn't change this fact and also can run in real-time.

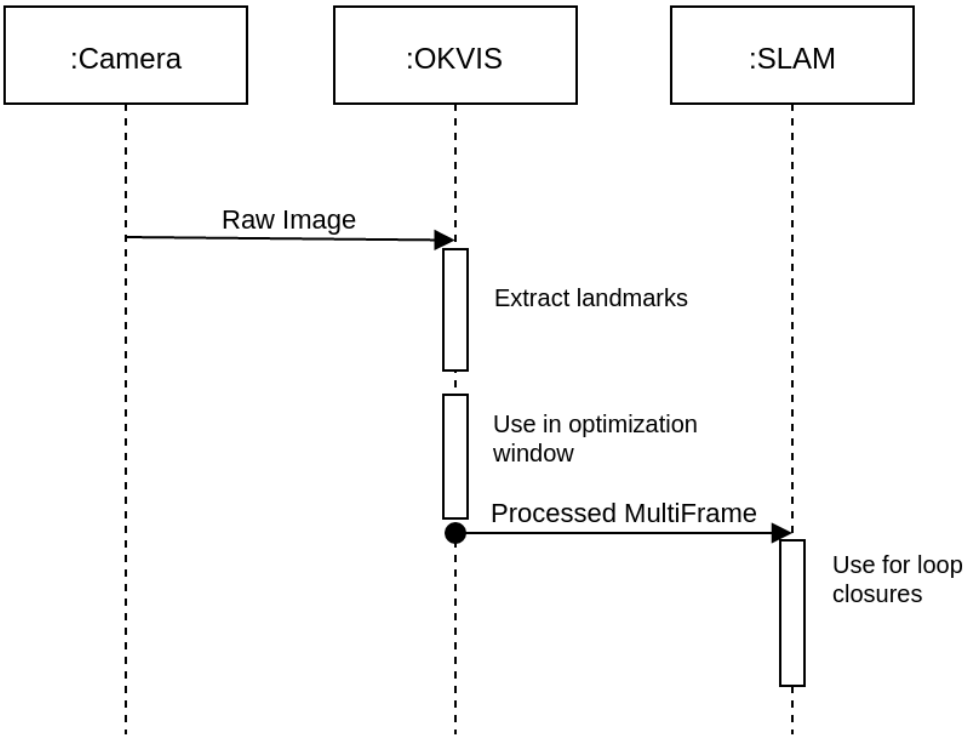


Figure 5.5: The diagram of how the data is passed in the system. Note, that the MultiFrame is a container class from OKVIS that includes both camera images, the landmarks and the associated pose.

As it was mentioned in Section 5.6.7, the start up happens mostly in the same way.
For OKVIS:

```
1   roslaunch okvis_node_synchronous.launch
      bag:=/path/to/dataset_adj.bag
```

For SLAM:

```
1   roslaunch slam_node_synchronous.launch
      bag:=/path/to/dataset_adj.bag
      save_to:=/path/to/save/map/
```

The main difference is that instead of *okvis_node_synchronous.launch* we use *slam_node_synchronous.launch* and the extra parameter *save_to* is added that should point to the folder, where the map will be stored. There is another new

parameter called *load_from* that should point to the map that is used during relocalization. The start code looks as follows:

```
roslaunch slam_node_synchronous.launch
    bag:=/path/to/other_dataset_adj.bag
    save_to:=/path/to/save/map/
    load_from:=/path/to/load/map
```

The relocalization is optional in SLAM as well as *load_from* parameter that can be omitted if SLAM is started without the relocalization feature.

5.5 Output

The main result of OKVIS is the real-time pose of the hardware setup in the world frame. The ROS node is publishing this information as a ROS topic called */okvis_node/okvis_transform*. Each published entry consists of the translation (x, y, z) and the rotation in the form of a quaternion (x, y, z, w). SLAM keeps this topic without any changes and adds a new one called */slam_node/slam_camera_pose*. The new topic has the same structure with the difference that the pose has loop closure optimizations already applied (if any). So, generally speaking, the pose published by SLAM is exactly the same as by OKVIS or has a better precision.

Map files

Apart from publishing the real-time pose, SLAM saves the map after the whole processing is done. The map's files are structured as following:

- *frames/* - the folder contains the images of the keyframes
- *matches/* - the folder contains the loop closure image pairs
- *meta.txt* - the file with the mapping from keyframe ids to frame ids and with the pointer to the matched image (if any)
- *original.txt* - the file with the not optimized (OKVIS) trajectory in g2o format
- *optimized.txt* - the file with the optimized (SLAM) trajectory in g2o format
- *original_eval.txt* - the file with the not optimized (OKVIS) trajectory in evaluation format
- *optimized_eval.txt* - the file with the optimized (SLAM) trajectory in evaluation format

5.5.1 File formats

The g2o file format represents the pose graph and consists of the two kinds of entities: vertices and edges. First, all vertices are listed, each on the separate line with the following syntax:

```
1 VERTEX_SE3:QUAT vertex_id X Y Z QX QY QZ QW
```

where "VERTEX_SE3:QUAT" is a constant string, "vertex_id" is a unique integer and the rest are float numbers. The translation is represented by "X", "Y", "Z" and the rotation quaternion is represented by "QX", "QY", "QZ", "QW".
Second, after all the vertices, there are all the graph edges, each on the separate line with the following syntax:

```
1 EDGE_SE3:QUAT from_id to_id X Y Z QX QY QZ I0 ... I20
```

where "EDGE_SE3:QUAT" is a constant string, "from_id" and "to_id" are the ids of the vertices that edge is connecting. As before, the translation is represented by "X", "Y", "Z" and the rotation quaternion is represented by "QX", "QY", "QZ", "QW". The edge represents the relative transformation from one vertex to another. However, it is important to know, that the translation and the rotation of the edge are provided **in the frame of the vertex where the edge comes from**. This is a big difference to vertices themselves, because vertices transformations are all provided in the world frame. The rest of the line are 21 floats "I0" to "I20" that represent the top right corner of the edge's information 6×6 matrix. This information matrix is the inverse of the covariance matrix that shows how strongly translation and rotation are correlated. In the case of our thesis, this matrix is always a diagonal one.
The images are stored in the gray-scale PNG format having a frame id as a filename. To know what keyframe id corresponds to frame id, the *meta.txt* file can be inspected.

5.5.2 Visualization

As part of the thesis, we have implemented a tool for the visualization of the map. The tool can be used to visually inspect the quality of the algorithm, to compare the trajectories of SLAM and OKVIS, and to check that the loop closure image pairs are properly located in space. The visualizer is implemented using Qt and libQtGlViewer, the 3D graphics is programmed using OpenGL. See Figure 5.6 for the typical usage screenshot.

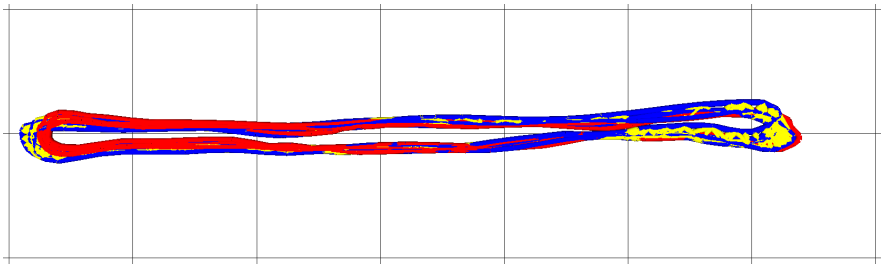


Figure 5.6: The visualization of the map of the loop-walking in the office. The blue trajectory represents OKVIS original version (here the trajectory drifts a little bit). The red trajectory represents SLAM optimized version. The yellow lines link the loop closure keyframes.

5.6 Software documentation

The recording and processing of the datasets is happening in ROS. OKVIS itself is implemented as a ROS node. For storage, ROS bag files are used.

5.6.1 Installation

In order to install SLAM, we first recommend to install original OKVIS from its github page *https://github.com/ethz-asl/okvis_ros*. Most of the libraries required by SLAM are the same as for OKVIS. Once the original OKVIS is compiled successfully, SLAM can be built too.
The only extra library required by SLAM is *opencv* with *non-free* components (for SURF). Usually, *opencv* that is bundled with the OS lacks these components and you have to build full *opencv* manually. You probably will have to update "OpenCV_DIR" and "OpenCV_CONFIG_PATH" variables to point to your build's "share" folder (see example below).
Building SLAM is similar to OKVIS:

```
1  cd slam_workspace
2  catkin_make -DOpenCV_DIR:PATH=
       /home/kasyanov/libs/share/OpenCV/
       -DOpenCV_CONFIG_PATH:FILEPATH=
       /home/kasyanov/libs/share/OpenCV/
```

After building, you have to source the library:

```
1  cd slam_workspace
2  source devel/setup.sh
```

For convenience, this code can be added to */.bashrc* or */.bash_profile* to avoid typing it every time.

5.6.2 Drivers

There are drivers for ROS for both cameras (*prosilica_camera*) and for IMU (*ethzasl_xsens_driver*) that can be installed from Ubuntu repositories or from their github pages. However, we have made some important modifications to existing drivers and bundled the drivers together with thesis: in *ethzasl_xsens_driver* we have fixed a bug with SyncIn (*analogin1*) recording and in *prosilica_camera* we have added the ability to publish camera's exposure. To build new drivers, execute

```
1  cd $DRIVER_WORKSPACE
2  catkin_make
```

in each driver's workspace. After that, the drivers have to be activated using the following command:

```
1  cd $DRIVER_WORKSPACE
2  source devel/setup.sh
```

For convenience, this code can be added to */.bashrc* or */.bash_profile* to avoid typing it every time.

5.6.3 Starting

Once the hardware is calibrated and ready for the recording, we should start ROS first:

```
1  roscore
```

Next, the cameras and the IMU can be started:

```
1  cd run
2  roslaunch start_all.launch
```

To verify that the hardware is powered on and is running, try:

```
1  rosrun image_view image_view
       image:=/cam_left/image_raw
2  rosrun image_view image_view
       image:=/cam_right/image_raw
3  rostopic echo /imu/data
```

Here, "image_view" should display camera images and "echo" should constantly print IMU measurements. If left and right cameras are swapped, then "start_all.launch" file has to be updated. In this case, camera images are black and white, which is due to the fact that "image_raw" topic delivers only such kind of images - this is the expected behavior.

5.6.4 Recording

To record a sequence to be later processed by OKVIS or SLAM, please do the following (after the hardware is running):

```
rosbag record /cam_left/image_raw
   /cam_right/image_raw /cam_left/exposure
   /cam_right/exposure /imu/data
   /imu/analog_in1 –O dataset.bag
```

This will start the recording. After you are done, please press Ctrl+C to stop the recording. The recorded sequence will be saved in the file dataset.bag in the current folder. To see the length of the sequence, you can use:

```
rosbag info dataset.bag
```

After the recording, the sequence's timestamps have to be adjusted (time synchronization is happening here):

```
python bag_adjust.py dataset.bag dataset_adj.bag
```

5.6.5 Configuration

The main config is stored in *slam_workspace/config_ours.yaml*. Most part is the same as OKVIS config. Here is the list of new, SLAM-only options:

- *DBoW2Vocabulary* - path to DBOW2 model
- *DBoW2Threshold* - threshold to assume that images are similar (default: 0.06)
- *factorWeight* - weight of loop closure factors (default: 0.3)
- *displayPath* - display 2D top down view of the trajectory
- *displayImages* - display images that are processed by OKVIS
- *waitFinish* - after the processing is done, wait for user input

If you want to change some options, use *slam_workspace/config_ours.yaml* file. However, if you want to change the calibration, you have to generate a new config. Suppose the output of full kalibr calibration is in file *kalibr_full.yaml*. Launch the following:

```
cd slam_workspace
python okvis_config.py kalibr_full.yaml
```

This will take values from *kalibr_full.yaml* and put them in correct format to *config_ours.yaml*. The template that is used is *config_okvis_templ.yaml*. Note, that OKVIS might complain about *config_ours.yaml* format, then you need to reformat it using some online yaml linter, for example http://www.yamllint.com/.
Do not forget, that the first line of *config_ours.yaml* should be "%YAML:1.0" (yes, it is not valid yaml for python but somehow valid for C++). Comments about OKVIS configuration options can be found in *config_okvis_templ.yaml*.

Indoor/Outdoor IMU configuration

It is important to know, that different IMU settings are required for indoors and outdoors sequences. For indoors, use *sigma_a_c*= 0.002 and *sigma_g_c*= 0.0008. For outdoors, use *sigma_a_c*= 0.02 and *sigma_g_c*= 0.008. The numbers are different, because outdoors the camera is carried by a person or is mounted on the bike. Steps and bumps on the road add more noise to IMU measurements, so these parameters make OKVIS less sensitive to noise.

5.6.6 Running OKVIS

To start OKVIS on the recorded and adjusted sequence, run:

```
1 cd slam_workspace
2 roslaunch okvis_node_synchronous.launch
    bag:=/path/to/dataset_adj.bag
```

During the processing, OKVIS outputs current images as well as a trajectory top-down view.
Note: update the absolute path to config in *okvis_node_synchronous.launch*.

5.6.7 Running SLAM

SLAM is started similar to OKVIS:

```
1 cd slam_workspace
2 roslaunch slam_node_synchronous.launch
    bag:=/path/to/dataset_adj.bag
    save_to:=/path/to/save/map/
```

After the completion, the map will be stored on the hard drive. Moreover, since SLAM supports relocalization, the existing map can be reused:

```
1 roslaunch slam_node_synchronous.launch
    bag:=/path/to/other_dataset_adj.bag
    save_to:=/path/to/save/map/
    load_from:=/path/to/load/map
```

Note: update the absolute path to config in *slam_node_synchronous.launch*.

5.6.8 Real-time SLAM

SLAM (like OKVIS) is real-time capable. It can be started as a ROS node that subscribes to IMU and camera topics and performs the estimation in real time. To save the map, press *Ctrl+C*. The map path is in *slam_node.launch* file. Starting looks like following:

```
1 cd slam_workspace
2 roslaunch slam_node.launch
```

Note: update the absolute path to config in *slam_node.launch*.

5.7 Discussion

In the end we have obtained the working hardware setup to perform visual odometry or SLAM. The cameras are calibrated with respect to each other and to the IMU. The measurements of the IMU trigger the cameras' image capturing process via special hardware synchronisation pins. The whole setup is portable with the laptop and the power bank, and can be placed into a backpack.
We have recorded multiple sequences inside our lab as well as outside, on the street. These sequences were fed to OKVIS to verify that everything is properly calibrated and synchronised. One important finding from these experiments was that dynamic objects in the sequence (e.g. people and cars) degrades quite a lot the quality of the trajectory. This happens because feature points are extracted on dynamic objects, therefore feature points world positions are not constant. It can be easily seen that during OKVIS evaluation by the authors, they have used only EuRoC MAV datasets [LLB+15] without any dynamic objects (empty machine hall, empty room).

6 Evaluation

This chapter will evaluate the SLAM approach that was created during this thesis. In order to do so, we have performed various tests on the EuRoC MAV dataset [BNG+16] and our own recorded sequences.

6.1 DBoW2 vs. FAB-MAP2.0

To pick the image similarity measure, we have decided to evaluate these two approaches. Instead of FAB-MAP 2.0 we have used openfabmap [GMW+12] library that corresponds to the FAB-MAP 2.0 paper [CN11].
As a training dataset we have used 10000 images from SUN RGB-D dataset [SLX15] that consists mostly of indoor images. We have verified that the detectors, trained using this dataset, work fine outdoors too. First notable difference is the speed of training: DBoW2 took around 8 hours, while openfabmap took almost two days.
To use as a test datasets, we have handcrafted small benchmarks that contained image pairs that should be matched as loop closures:

- *office_small* - 6 images from our office, 2 loop closures expected
- *office_big* - 298 images from our office, 3 loop closures expected
- *outdoors* - 10 images from the car camera, 5 loop closures expected

The detection results can be seen in the Table 6.1. There are no false positives using both methods. Few examples of image pairs where DBoW2 works and openfabmap fails can be seen in Figure 6.1.

It can be seen, that DBoW2 constantly delivers better results. Moreover, its execution times are lower too (as stated in the original paper). Having this information we have decided to use DBoW2 as image similarity detector.

Table 6.1: DBoW2 vs FAB-MAP 2.0, Our Comparison

Dataset	openfabmap	DBoW2
office_small	1/2 (50%)	**2/2 (100%)**
office_big	1/3 (33%)	**3/3 (100%)**
outdoors	1/5 (20%)	**3/5 (60%)**

(a) Loop closure pair from *office_small* dataset

(b) Loop closure pair from *outdoors* dataset

Figure 6.1: Examples of loop closures that are detected by DBoW2 but not openfabmap

6.2 EuRoC MAV dataset

6.2.1 General performance

We have started evaluation from running our SLAM system on every EuRoC dataset sequence (Machine Hall and Vicon Room). There are no pre-designed loop closure parts since these sequences were recored to evaluate general performance. However, the same place revisiting happens in most of the sequences leading to the improved quality of the detection.

To quantitatively evaluate the detection, we have used the following metrics [SEE+12]:

- ATE (Absolute Trajectory Error) - represents the average distance between corresponding points of the estimated trajectory and the ground truth. Two points correspond to each other if their timestamps are similar.
- RPE (Relative Position Error) - represents the average error between the parts of the trajectory that are divided by the fixed time deltas (we use 0.5,1,2,5,10 sec).

The dataset includes the ground truth trajectory from the Leica MS50 laser tracker, that can be temporary and spatially aligned with the trajectory from OKVIS or SLAM. The results for stereo setup can be seen in Table 6.2. In the table, the ATE error in meters is displayed for trajectories from OKVIS and SLAM. The datasets MH1-MH5 encode the Machine Hall and V11-V22 encode the Vicon Room.

Table 6.2: The comparison of the SLAM and OKVIS in EuRoC dataset. The lower the better.

ATE [m]	MH1	MH2	MH3	MH4	MH5	V11	V12	V21	V22
OKVIS mono	0.34	0.36	0.30	0.48	0.47	0.12	0.16	0.12	0.22
SLAM mono	0.25	0.18	0.21	0.30	0.35	0.11	0.13	0.12	0.20
OKVIS stereo	0.21	0.14	0.25	0.33	0.36	**0.08**	**0.10**	0.09	**0.16**
SLAM stereo	**0.11**	**0.09**	**0.21**	**0.27**	**0.32**	0.09	**0.10**	**0.08**	**0.16**

It can be seen, that SLAM is better then OKVIS in the Machine Hall sequences. However, the difference is much smaller in Vicon Room. This can be explained through the nature of the sequences: in the Machine Hall, the drone is flying in the big room in straight lines for tenth of meters and the revisiting happens not frequently and gives a good improvement of the trajectory (see Figures 6.2 and 6.3). On the other hand, the Vicon room is pretty small room and camera moves in a chaotic fashion causing a lot of frame overlays in the pretty small area (see Figure 6.4). This leads to just small improvements because there are no real loops closures - the camera never goes further then a few meters from the starting position. From this information we can draw a conclusion that SLAM gives the biggest improvements in the big rooms or areas where the setup moves in long (more then a few meters) shifts, while in the small rooms the SLAM improvement is little to none.

Since OKVIS authors claimed that it can run with the single camera setup, we have decided to evaluate our SLAM approach with a single camera too. The results can be seen in the Table 6.2. The errors for mono are naturally higher then for stereo. However, it can be seen, that SLAM still delivers the improvement over original OKVIS estimation even for a single camera setup.

In order to further evaluate the estimation, we have made plots of the OKVIS and SLAM trajectories compared to ground truth in Figures 6.2, 6.3 and 6.4.

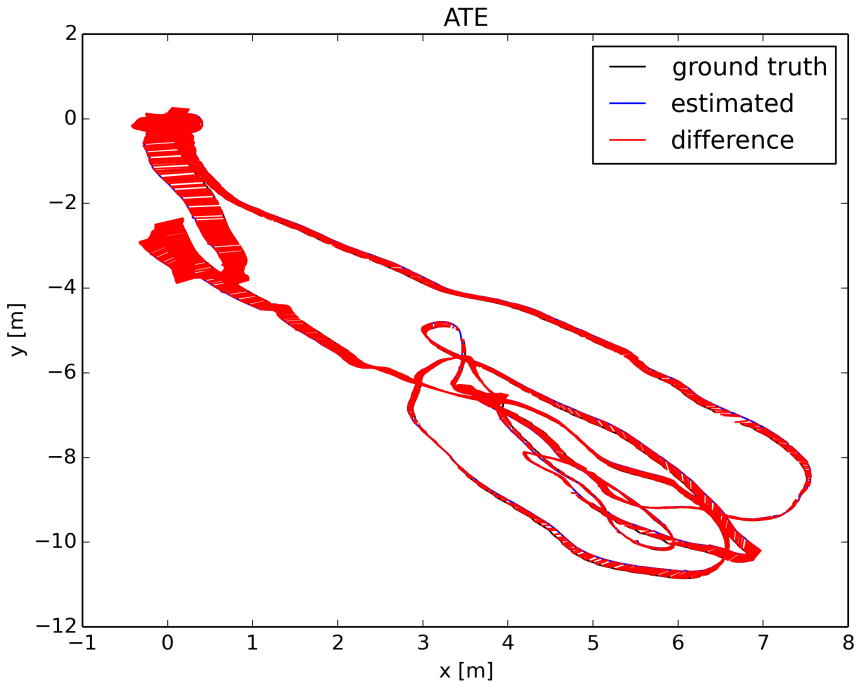
We have mentioned that we have also computed RPE metrics. The evaluation showed that the RPE difference between OKVIS and SLAM is less then 1 cm, so these numbers are not included into the tables. Summing up, SLAM doesn't change much the shape of the trajectory (RPE is similar), but aligns it globally more consistent (APE is smaller).

6.2.2 Relocalization

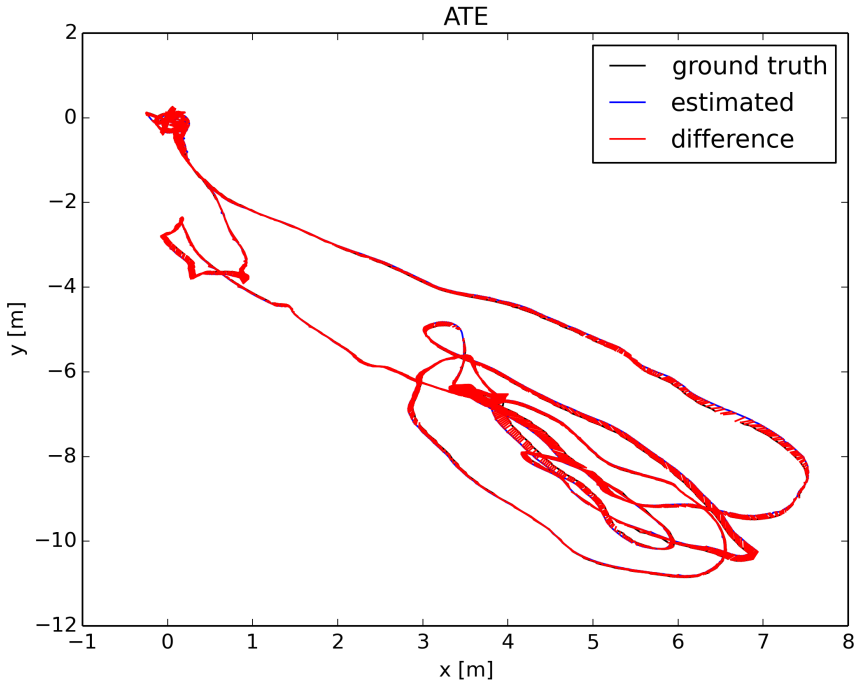
Apart from the trajectory accuracy we wanted to also evaluate the relocalization capabilities. In order to do that, we have performed three tests.

Number of keyframes to relocalize To relocalize in the map, SLAM needs at least 4 subsequent keyframes that match subsequent keyframes in the map, but this number can be higher. We wanted to see how many frames are actually needed in practice. To test this, we have run SLAM on the Vicon Room sequences to relocalize within the same sequence. For example, first we run *V1_01_easy*, save it as a map, and then we start *V1_01_easy* to relocalize in this map. Ideally, relocalization should happen quickly because this is basically the same sequence, i.e. the perfect case. Moreover, we have run relocalization with different time offsets from the beginning of the sequence (10 sec, 20 sec, etc) to get more varied and realistic results (even if we want to relocalize in the same area, it is very unlikely that exactly the same starting location is used). We have built the histogram that can be seen in Figure 6.5. It can be seen that the results are close to expected - most of the runs required exactly 4 keyframes for relocalization to establish and some other required up to 8 keyframes that is still pretty good. The algorithm uses more then 4 keyframes when there are no reliable subsequent correspondences between the keyframes. For example, when 8 keyframes are used that means that relocalization happened after 8 keyframes and 4 out of 8 had a reliable matches in the old map.

Relocalization error We wanted to see how does incremental relocalization and alignment improve the ATE error in relation to the ground truth. It is expected that right after the relocalization is established, the error can be high, but after some time, the new factors that tie together the new sequence and the map it is relocalized within, should decrease the error. The sequences were run in the same map as in previous example with different time offsets. The result can be seen in Figure 6.6. The curve goes down pretty quickly as we have expected and becomes flat after 10 frames on average. Having cameras working at 20 Hz, we can say that the relocalization becomes stable after 0.5 *sec* after establishing the initial association between the sequence and the map.

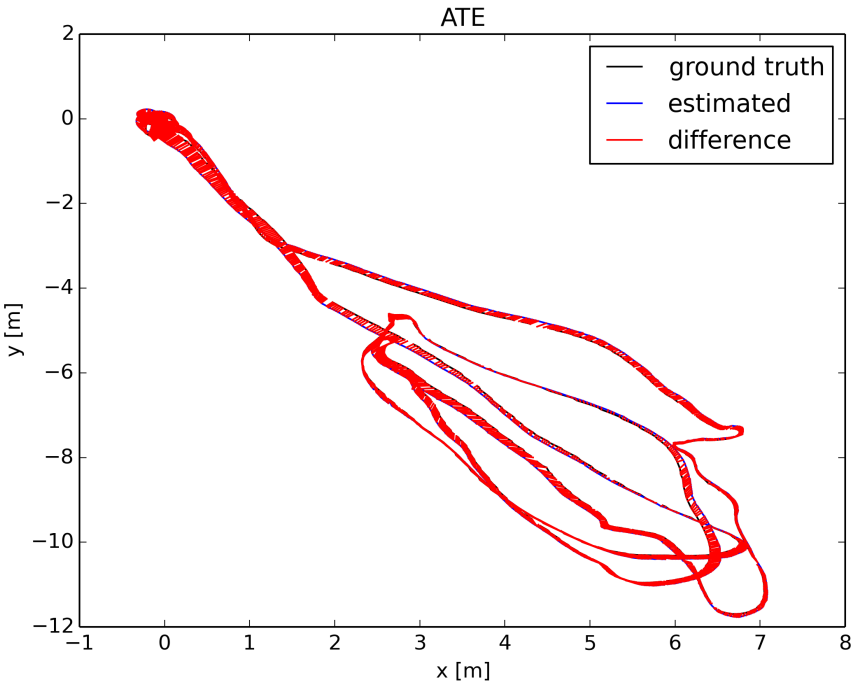


(a) OKVIS trajectory estimate.

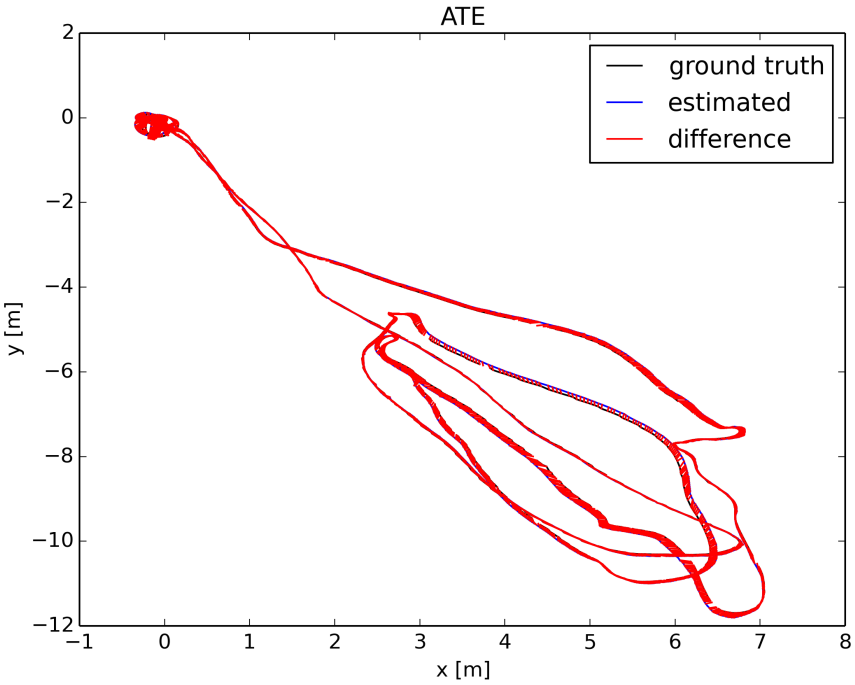


(b) SLAM trajectory estimate.

Figure 6.2: Estimates on the *MH_01_easy* sequence. The red lines represent the difference between the ground truth and the estimated trajectory. The smaller the red gaps, the better. We can see well in the top left corner that SLAM trajectory has smaller deviation from the ground truth.

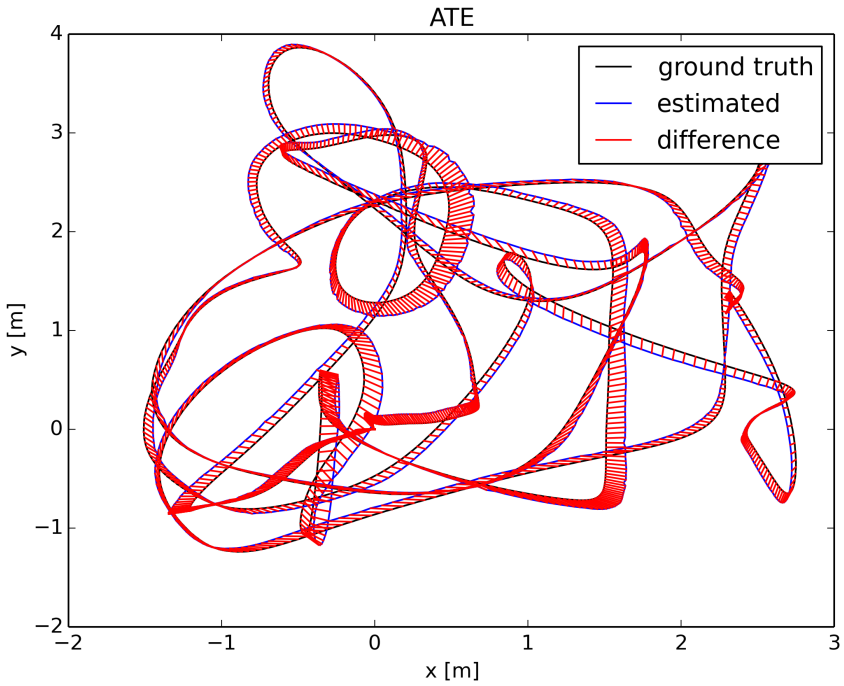


(a) OKVIS trajectory estimate.

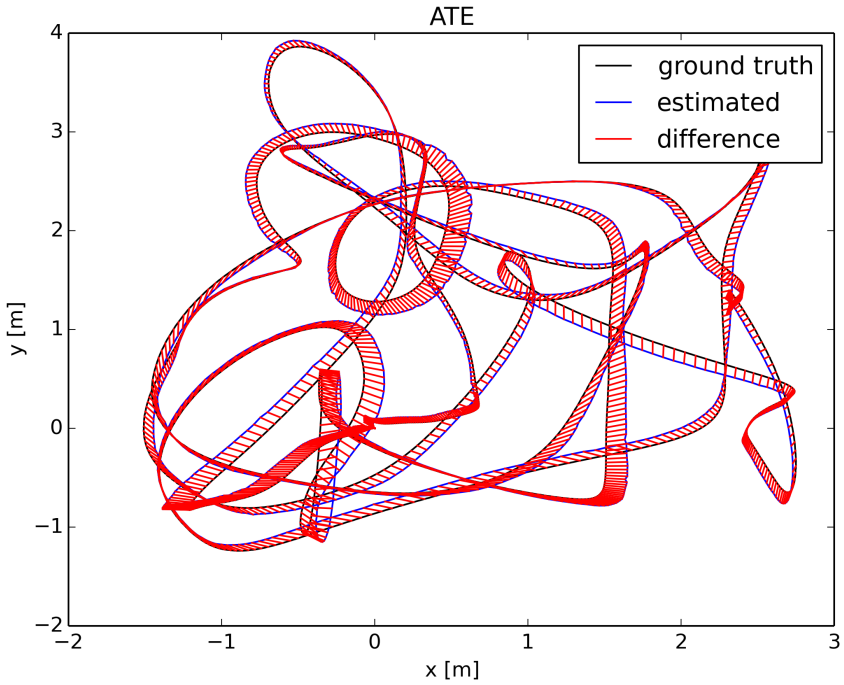


(b) SLAM trajectory estimate.

Figure 6.3: Estimates on the *MH_02_easy* sequence. The red lines represent the difference between the ground truth and the estimated trajectory. The smaller the red gaps, the better. We can see well in the top left corner that SLAM trajectory has smaller deviation from the ground truth.



(a) OKVIS trajectory estimate.



(b) SLAM trajectory estimate.

Figure 6.4: Estimates on the *V1_02_medium* sequence. The red lines represent the difference between the ground truth and the estimated trajectory. The smaller the red gaps, the better. We can see that there is little to no difference between OKVIS and SLAM which corresponds to equal ATE values.

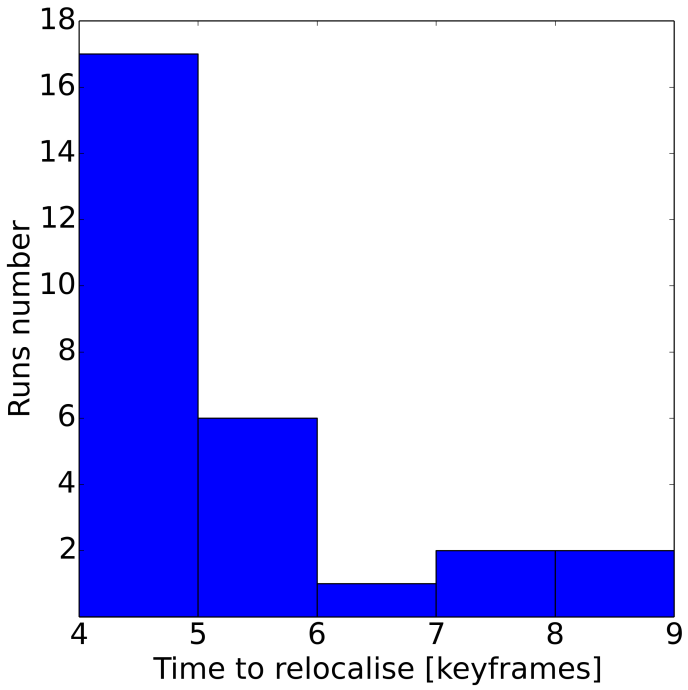


Figure 6.5: The amount of keyframes needed to relocalize within the same sequence (EuRoC sequences).

Relocalization in different map The Vicon Room sequences are all recorded in the same, relatively small room. We have decided to try to relocalize these sequences within each other. For example, sequence *V1_01_easy* can use *V1_02_medium* as a map. After relocalizing within the different sequence, we have computed the ATE error of the sequence trajectory compared to the ground truth. The results can be seen in the Table 6.3. We see that for some combinations the error goes up, but it still remains quite low. The increasing of the error can be explained by the bias to the map sequence. The new sequence is aligned with the map sequence, that in turn, is not perfect and has some ATE error compared to its ground truth. These errors, naturally, accumulate. However, the provided test shows that relocalization works well when we use different sequence as a map with just minor image overlays.

Table 6.3: ATE error after the sequence was relocalized with respect to other sequence. The lower the better.

ATE [m]	V11 in V12	V12 in V11	V21 in V22	V22 in V21
Solo	**0.09**	**0.10**	**0.08**	0.16
Relocalized	0.11	0.13	**0.08**	**0.13**

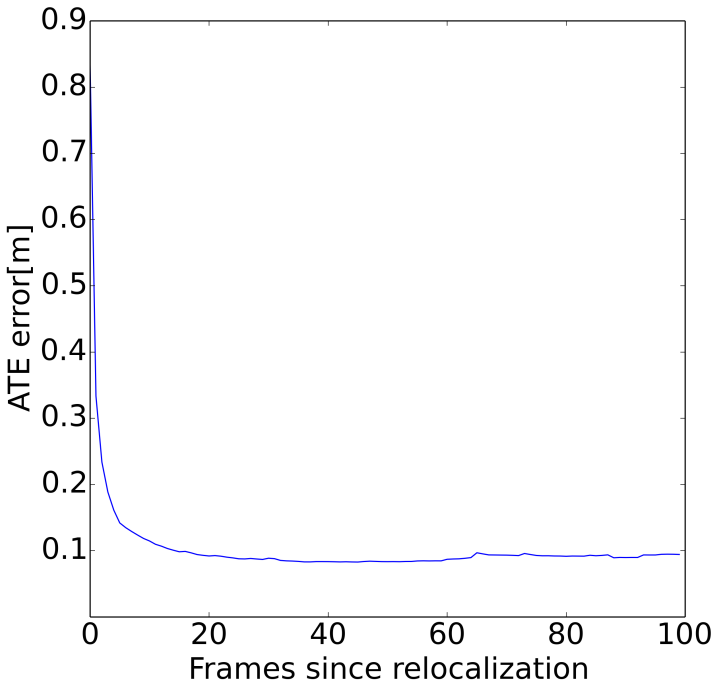


Figure 6.6: The ATE error progression after relocalization.

6.2.3 3D loops closure evaluation

One important part of our system is detecting of the similar images. More precisely, the images that show the same scene, possibly from a bit different positions. We can use such images to determine that the cameras revisit the same location. It was mentioned before that we use the combination of DBoW2 (Section 3.3.2) and RANSAC 3D-2D relative pose estimation [KF14]. After the system was in place, we have decided to quantitatively evaluate the image matches quality. To do so, we have started by taking a 3D point cloud from the Vicon Room sequences that is a room 3D-scanned by Leica MS50 in EuRoC MAV dataset (see Figure 6.7).

Using it as a ground truth, we were able to project 3D points into the camera frame that follows the ground truth trajectory. If a pair of images have at least 50% of shared 3D points in the projection, then this pair is treated as a ground truth match. After processing, we have calculated how many of these matches were detected by our SLAM system. The results can be seen in Table 6.4. We can see that the percentages are quite low. However, after we have looked at the ground truth matches that were not detected by SLAM system, it became apparent that this method of evaluation is over-performing. I.e. view port changes up to 90 degrees are not a problem for 3D point cloud and occlusion is not accounted

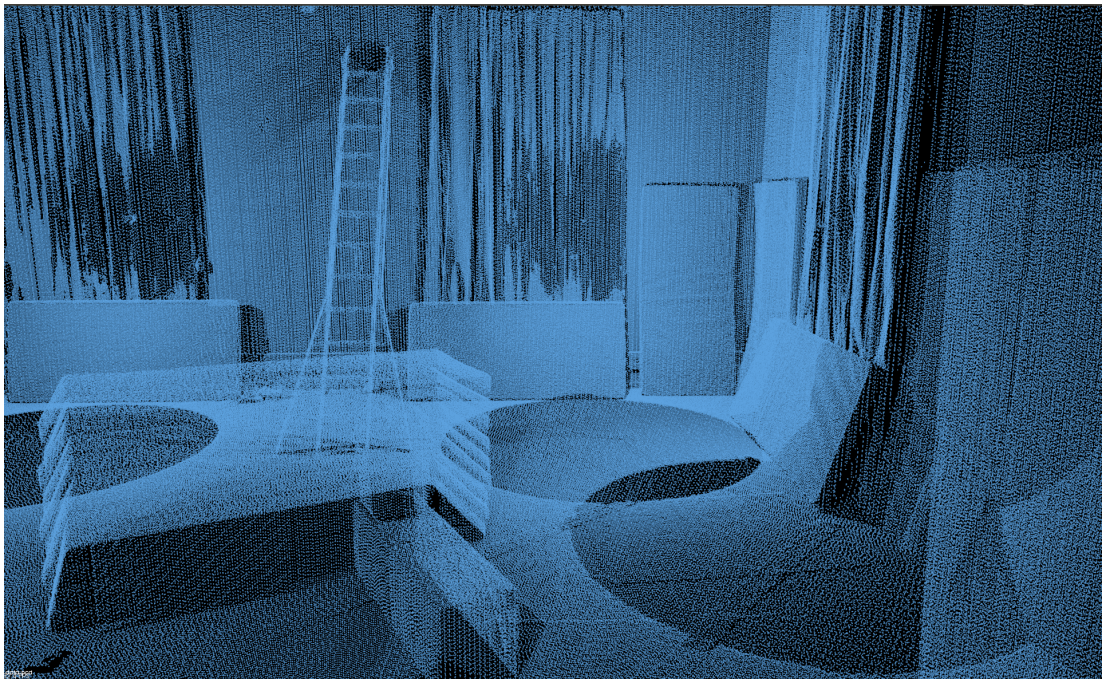


Figure 6.7: The 3D point cloud from Vicon Room dataset that was used to determine ground truth image matches.

for. There are some example of false negatives in Figure 6.9. It can be seen, that such detections are very challenging even for the human eye. Motion blur also makes it harder to compute and compare distinct features while 3D cloud is unaffected by it. We have also analyzed how the loop closure performance changes with the ground truth threshold and it can be seen that it is quite stable (see Figure 6.8).

Table 6.4: Image matches detection of our SLAM system compared to the 3D cloud ground truth.

	V11	V12	V21	V22
# detected	18/54	19/34	10/41	30/75
% detected	33.3	55.8	24.3	40.0

6.3 Our dataset

We have assembled and calibrated our own hardware setup that works with OKVIS and SLAM system. In order to evaluate the setup and SLAM system we have recorded various sequences using our setup.

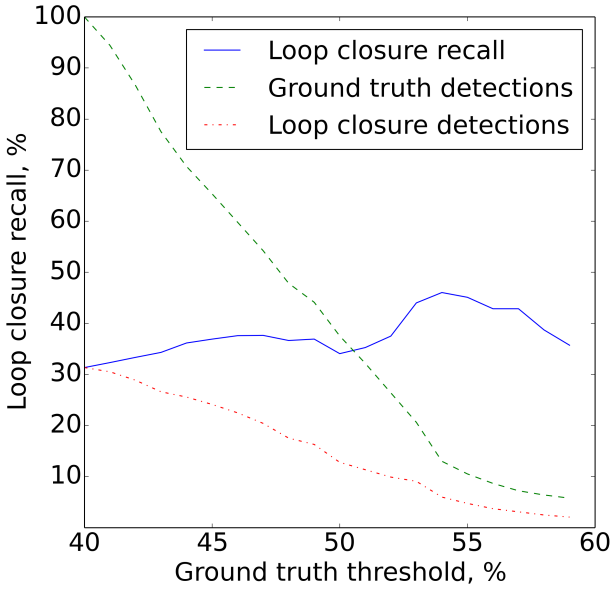


Figure 6.8: This plot shows how SLAM accuracy and the number of detected loop closures change with ground truth threshold.

6.3.1 Overview

In this section we are going to evaluate the SLAM system using the following sequences:

- *E1* - small sequence (135 m) inside the long office room. The setup is positioned on a rolling chair and is moved in multiple long loops.
- *E2* - small sequence (177 m) inside the university building. The setup is held in hands of a walking person. The person walks through three floors.
- *E3* - medium (875 m) outdoors sequence. The setup is held in hands of a walking person. The person makes a circle around the campus.
- *E4*, *E5* - big outdoors sequences (1341 and 1366 m). The setup is held in hands of a walking person. The person makes a circle around the campus with few loops around different buildings. We have made two similar sequences here to see how consistent is the system at such scale.
- *E6* - the longest (2891 m) outdoor sequence. The setup is mounted on a bicycle steering wheel.

Images examples from the sequences can be found in Figures 6.10, 6.11 and 6.12.

6.3.2 E1

In this experiment the camera on a rolling chair was moved in loops. The length is 135 m. The results were good (see Figure 6.13) - in stereo setup, OKVIS had only minor drifts that were corrected by SLAM. Loop closures happened along the way as expected. In mono setup, OKVIS drifted quite a lot, however SLAM corrected it very well, resulting in aligned loops, like in stereo case.

Figure 6.9: The image pairs that show the same scene according to the 3D point cloud, but not to SLAM system. Such cases are very challenging for computer vision detections.

Figure 6.10: Example images from *E1* sequence, where the camera is positioned on the rolling chair in the office.

6.3.3 E2

During the university building traverse, OKVIS showed pretty good estimate, but SLAM results were much worse. We have investigated into that, and the reason for poor SLAM estimation is that the camera sees staircase parts at different levels that look very similar. Therefore, SLAM fires a loop closure while these images were actually taken at different floors. You can see the trajectory and staircase image examples in Figure 6.14.

6.3.4 E3

This outdoors sequence was processed well by both OKVIS and SLAM. It was a first outdoors sequence with the trajectory few hundred meters long (875 m). We have overlayed the trajectory with the Google Map of the neighborhood, see Figure 6.15. It is interesting, that with mono configuration, the OKVIS trajectory is not much worse then stereo.

Figure 6.11: Example images from *E2* sequence, where the person with the camera is walking around the building, visiting multiple floors.

Table 6.5: Average checkpoint error

error[m]	E4	E5
OKVIS	10.9	40.7
SLAM	**0.89**	**0.93**

Table 6.6: Average checkpoint error. We can see that SLAM improved trajectory is much better then original OKVIS estimation.

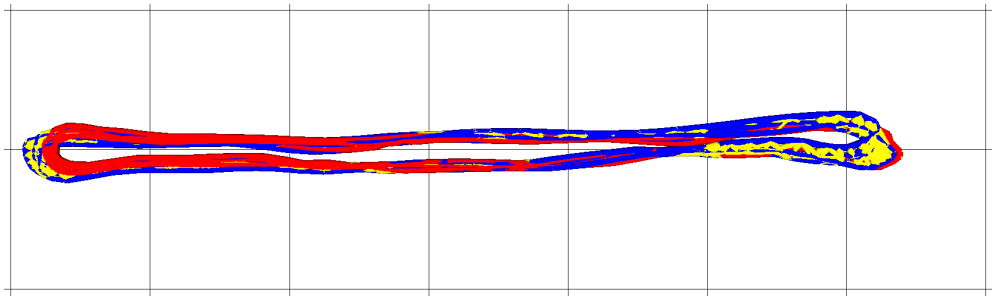
6.3.5 E4

This outdoors sequence is longer then *E3* (1366 m) and it also has two big loops in the middle (around the building and around the parking lot). The results can be seen in Figure 6.16. We can see that the final position is pretty close to the initial position (less then 2 meters for SLAM optimized trajectory) which means a good estimation.

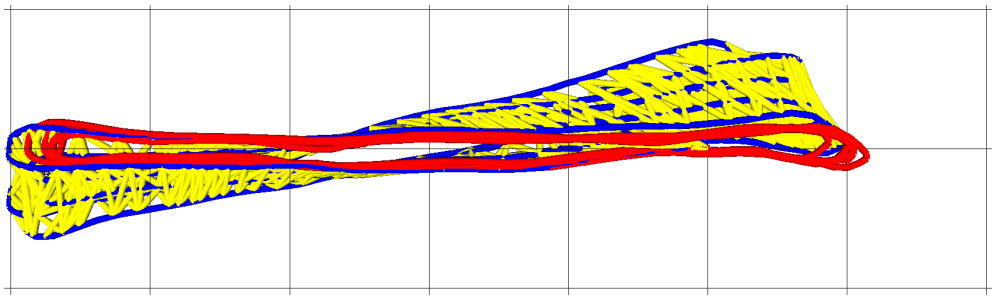
Figure 6.12: Example images from *E3* sequence, where the person with the camera is walking around the university campus on a sunny day.

6.3.6 E5

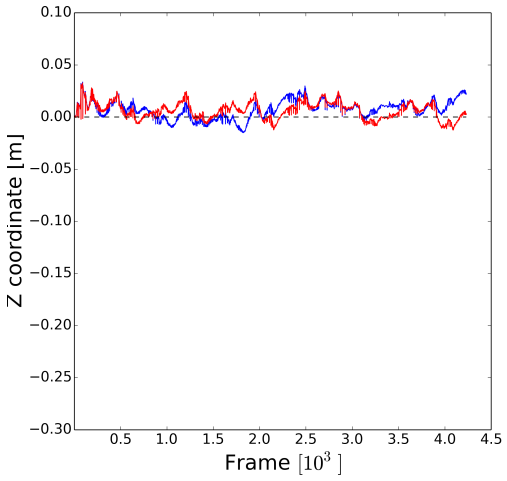
This sequence is very similar to *E4*, but during recording under the afternoon sun, the laptop was overheated, throttled and few camera frames were dropped. Such a situation is an interesting challenge for both OKVIS and SLAM. We are providing stereo only result here in Figure 6.17. It can be seen, that OKVIS trajectory is quite bad, it even finishes out of bound of the satellite map. Fortunately, additional loops provided enough loop closure constrains to SLAM, so it delivers a very good trajectory, that is almost indistinguishable from the results of *E4* (that had no issues during recording).
To have some quantitative estimation of how good the trajectory is, we have placed checkpoints (via gamepad buttons) in roughly same place before and after each loop, 7 checkpoints in total. Later we have calculated average checkpoint error that shows the distance within the three groups (2,2,3) of checkpoints. Each group is showed as a green circle in the trajectory images. Ideally, the error should be close to zero. The results for sequences *E4* and *E5* are shown in Table 6.5.
Moreover, we have verified that relocalization is working on our sequences. The



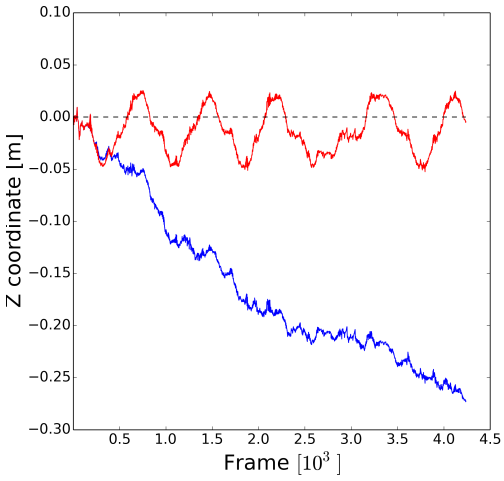
(a) Stereo case, top view.



(b) Mono case, top view.



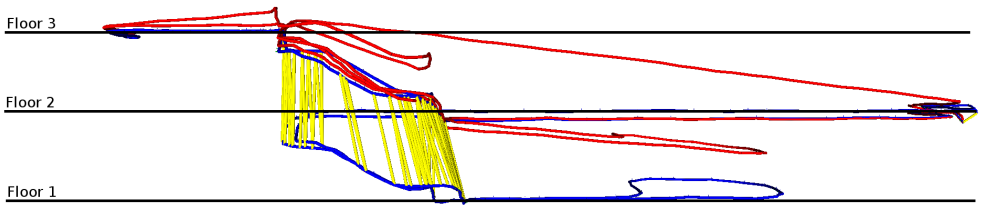
(c) Stereo case, Z position over time.



(d) Mono case, Z position over time.

Figure 6.13: *E1* OKVIS and SLAM trajectory estimates. Blue is OKVIS estimated, red is SLAM estimated, yellow lines show the loop closure constraints.

amount of keyframes to relocalize (see Figure 6.18) is close to EuRoC results (Figure 6.5) and is mostly 4 keyframes (which is a minimal required amount). During relocalization evaluation, we have also measured the performance of RANSAC



(a) Side view, black lines show actual floors. SLAM trajectory is much worse then OKVIS because of the wrong staircase matches.

(b) Example of false positive match. Note that posters on the wall are actually different because these images were taken at different floors.

Figure 6.14: *E2* OKVIS and SLAM trajectory estimates. Blue is OKVIS estimated, red is SLAM estimated, yellow lines show the loop closure constraints.

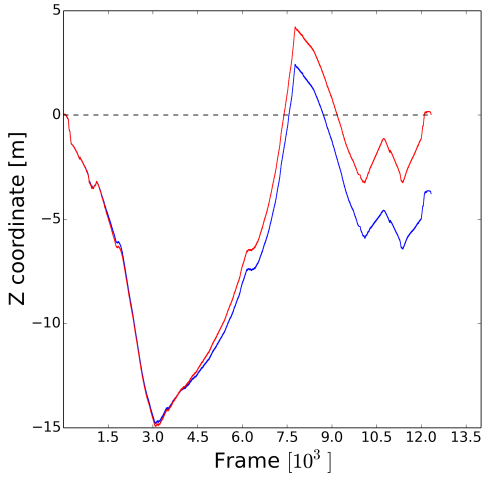
3D-2D relative pose computation. Since the relocalization was happening in exactly the same sequences, very similar images are matched. Therefore, relative pose offset should be close to zero. The results are displayed in Figure 6.20. It can be seen that most of the relative poses' offsets are close to zero, as expected. The progression of ATE error with respect to frames since relocalization can be seen in Figure 6.19. Here, the ATE error is computed with respect to saved SLAM trajectory. The shape is similar to EuRoC evaluation.

6.3.7 E6

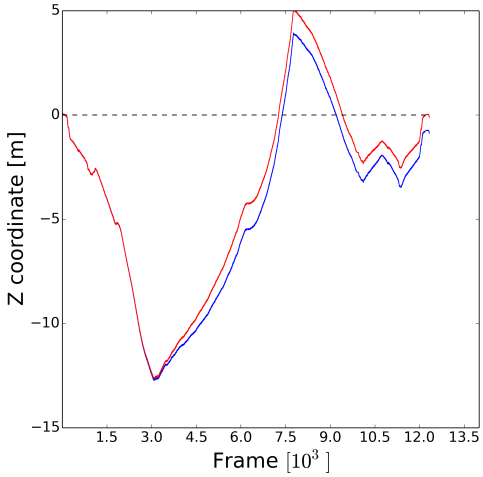
This sequence is the longest that we have recorded (2.8 km). The camera was mounted on the bicycle steering wheel. We have had a ride around the residential area next to the campus. The sequence is challenging because of the movement of other cars and people, the speed of the camera and the IMU shocks. The IMU we use has pretty low thresholds for gyro and accelerometer saturation, so small bumps on the road and sharp turns lead to saturation in OKVIS. We think this is

(a) Stereo case.

(b) Mono case.



(c) Stereo case, Z position over time.



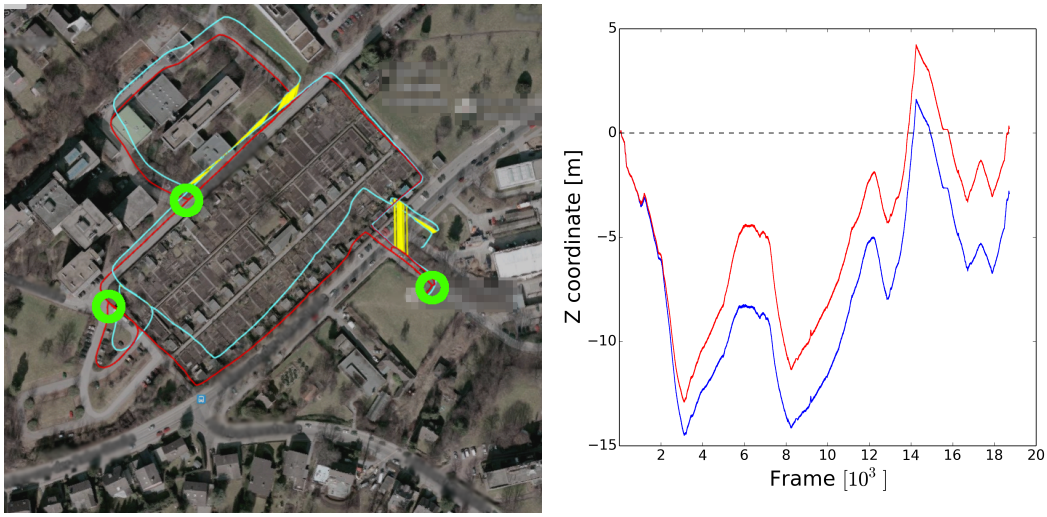
(d) Mono case, Z position over time.

Figure 6.15: *E3* OKVIS and SLAM trajectory estimates. Blue is OKVIS estimated, red is SLAM estimated, yellow lines show the loop closure constraints.

the reason why OKVIS estimation deviates a lot for actual trajectory. Moreover, it can be seen that the biggest drift happens on the way back right after a sharp 270 degree turn. On the other hand, SLAM improves the OKVIS trajectory a lot, delivering an estimation that is close to the actual trajectory (see Figure 6.21).

6.4 Discussion

We have evaluated our SLAM approach using both EuRoC MAV dataset and the sequences that we have recorded using our hardware setup. For every experiment we have used laser-scanned ground truth trajectory (if available) and OKVIS



(a) Trajectory with map.

(b) Z position over time.

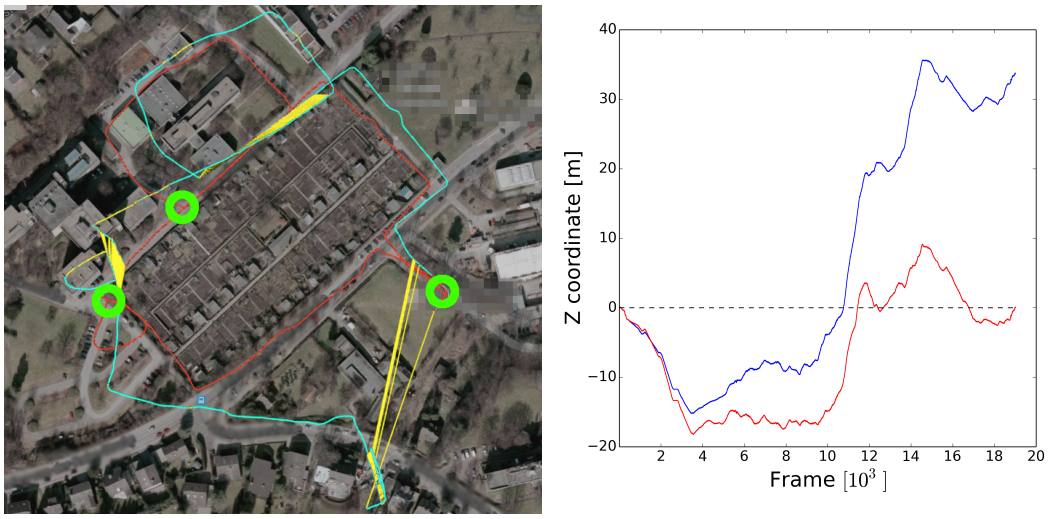
Figure 6.16: *E4* OKVIS and SLAM trajectory estimates. Blue is OKVIS estimated, red is SLAM estimated, yellow lines show the loop closure constraints. Green circles show the areas with checkpoints.

estimation as a baseline. Summing the experiments, we can say that SLAM outperforms OKVIS in all sequences with loop closures (*Machine Hall*, *E3-E6*) and in most sequences without specific loop closures (*Vicon Room*). We have found out that the longer the sequence, the bigger are the gains from using SLAM over OKVIS - for this we have recorded sequences outdoors with length up to 3 km.

As for relocalization we are able to reliably perform the relocalization in the same sequences with expected errors behavior. Relocalization within different sequence was also tested and yielded good results, compared to the ground truth trajectory.

Evaluation of the image similarity algorithms DBoW2 and FAB-MAP showed that DBoW2 provides better matches and had better testing and training times, so we have selected it as our primary image similarity estimator. RANSAC with 3D-2D matches geometrically verifies the estimation and provides a relative pose between images.

We have tried to evaluate loop closure detection using 3D point cloud from the ground truth. However, this method showed that detecting loop closures from 3D point cloud is superior to 2D images comparison and it not a very good measure in our case.



(a) Trajectory with map.

(b) Z position over time.

Figure 6.17: *E5* We have lost few frames during recording due to overheating. Because of that, OKVIS estimated pretty bad trajectory, that finishes out of bounds of the given map piece. Nevertheless, SLAM have used loops in the trajectory to improve the trajectory dramatically. Green circles show the areas with checkpoints.

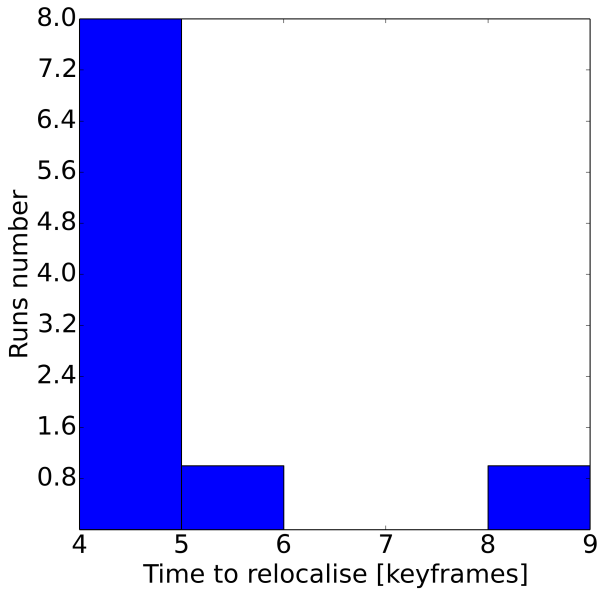


Figure 6.18: The amount of keyframes needed to relocalize within the same sequence (sequences *E3*, *E4*).

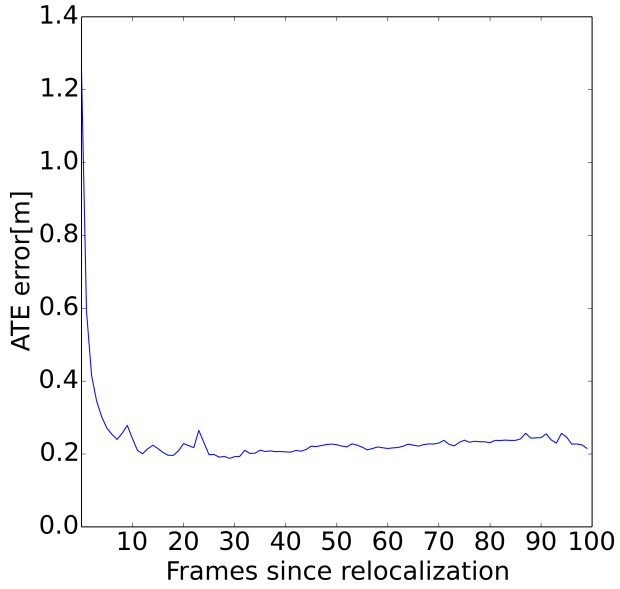
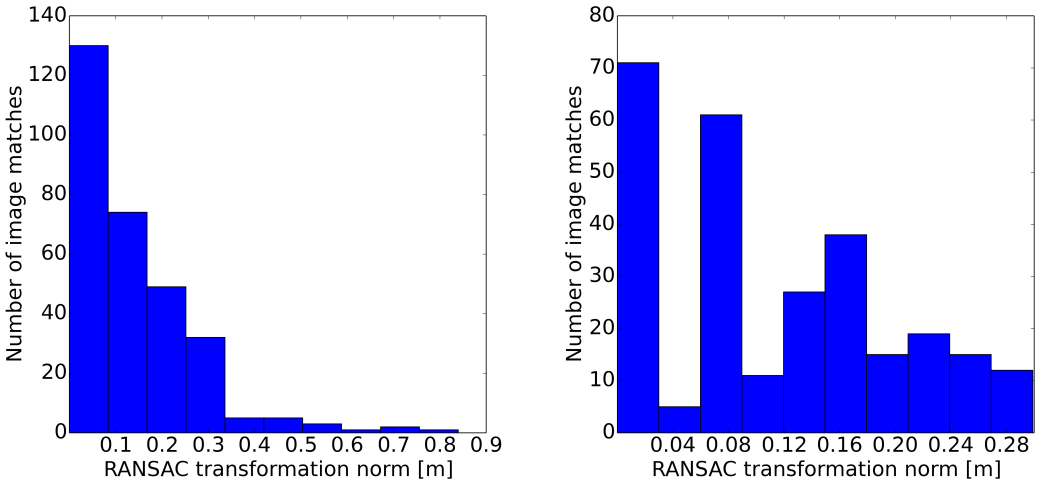


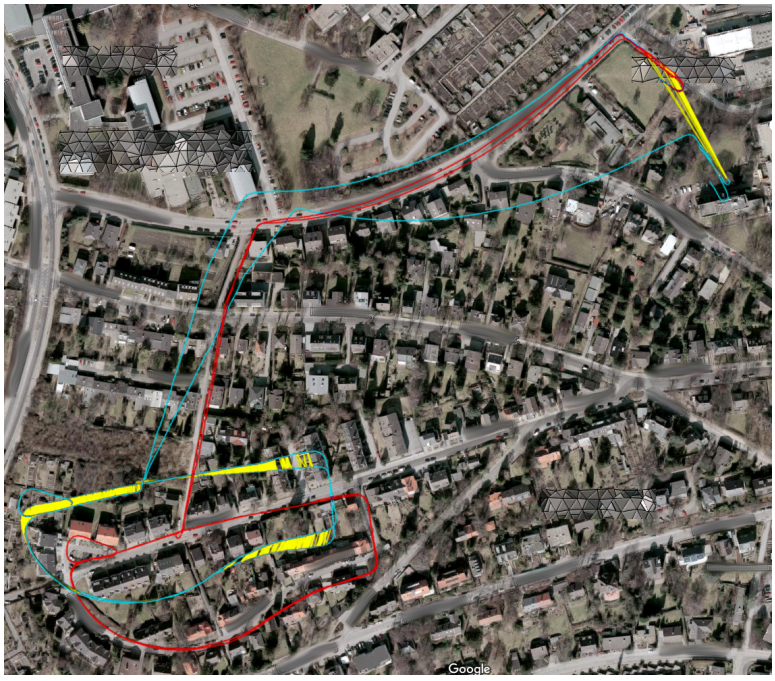
Figure 6.19: The progression of ATE error when relocalizing within the same sequence (sequences *E3*, *E4*). The curve gets flat quickly, as expected.



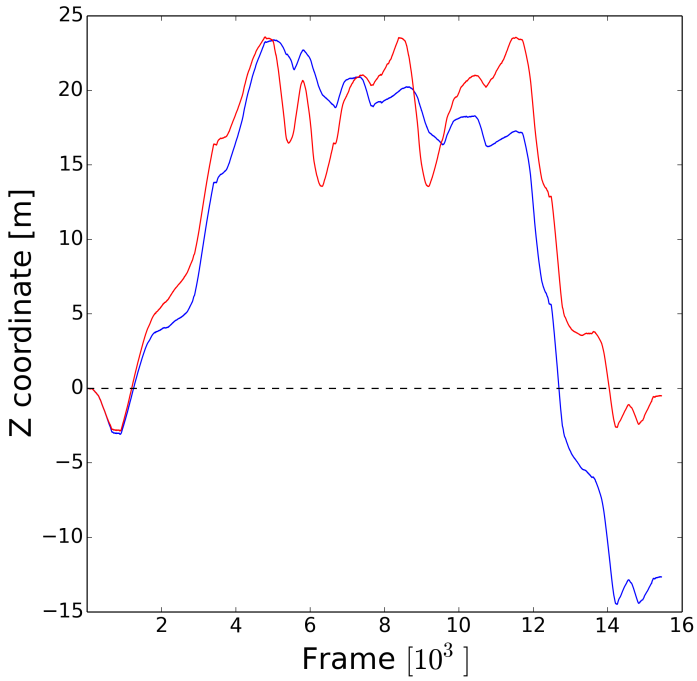
(a) Full scale.

(b) Details about the dense area.

Figure 6.20: The length of the RANSAC relative pose using very similar images. Best when close to zero.



(a) Trajectory with map.



(b) Z position over time.

Figure 6.21: *E6* OKVIS and SLAM trajectory estimates. Blue is OKVIS estimated, red is SLAM estimated, yellow lines show the loop closure constraints

7 Conclusion

In this thesis we have presented a Simultaneous Localization and Mapping (SLAM) layer on top of the existing visual-inertial odometry library OKVIS [LLB+15]. Our method uses a global pose graph to improve the trajectory estimation, while OKVIS delivers only local temporal consistency. The estimated trajectory can be saved as a map in which other sequences can relocalize. After relocalization, SLAM continues to work with respect to both new and old images, and trajectories.

Global constraints are provided by a loop closure mechanism. Namely we detect when the camera revisits the same area and add corresponding constraints to the pose graph. This is done by estimating image similarity via DBoW2 [GLT12] and RANSAC 3D-2D relative pose [KF14] calculation between two images. The graph built from OKVIS constraints and loop closure constraints are optimized by gtsam library. There is the ability to save the map to disk and relocalize in the map during subsequent runs. Having a global map with the ability to relocalize within this map can be useful for autonomous robots that operate in the unknown location.

The resulting software is bundled together as a ROS node and is capable of real-time processing. Apart from the software solution, we have assembled the hardware setup from two cameras and IMU. These components are calibrated with respect to each other (including time calibration). This way we are able to record our own sequences to better debug and evaluate SLAM. Furthermore, this setup can be used for other Computer Vision tasks solely or mounted to some robot.

Quantitative evaluation on EuRoC MAV dataset showed that the SLAM layer delivers a good performance boost compared to original OKVIS. We have also performed qualitative evaluation using our recorded sequences indoors and outdoors. The results show that the SLAM layer improves the OKVIS estimation when loops are present. Relocalization evaluation showed that it works consistently and allows to perform relocalization within different sequences.

7.1 Discussion

Overall, our SLAM system performance is better then OKVIS. The improvement is the highest when loop closures are present in the sequence. The disadvantage of the proposed method is that when there are no loop closures in the sequence and same areas are never revisited, the trajectory is indistinguishable from OKVIS estimation, while small running time overhead is added. We can recommend to use our SLAM system in cases when loop closures are probable in the sequences. Relocalization, on the other hand, works well even in the maps without any loop closures. Of course, having loops closures either in the map or in the sequence that is relocalized, improves the estimated trajectory.

Another advantage is that the SLAM system runs in real time, so it can be used by robots for navigation. The downside is that since SLAM uses only frames that were marginalized out by OKVIS, there is a 5 keyframes delay between the actual robot movement and the moment it is processed by SLAM.

We have found that OKVIS is very sensitive to moving objects in the sequence. For example, passing people or cars can decrease the quality of the estimation. Moreover, the lightning conditions are also important, especially outdoors. In one case loop closure was not detected, because the sun has moved behind the building and the shadows looked different. Another outdoors problem can arise when relocalization is happening few days after the map was recorded. Things like parking cars or falling leaves can drastically change the location appearance, making it harder for DBoW2 to correctly identify the loop closures. Very similar appearance patters, on the other hand, can introduce the wrong loop closure constraints, like in *E2* staircase example.

7.2 Future work

As for the future work, the following points can be improved:

- **OKVIS factors** Instead of OKVIS optimized poses, one can try to use raw IMU and reprojection factors that OKVIS uses in optimization. Combined with loop closure constraints, it might further improve the estimation.
- **IMU stability** We have found out that the IMU we used is not as precise as the one used by Leutenegger et al. Replacing IMU in the hardware setup can improve the performance of our hardware.
- **Optimization** Even though SLAM can run in real-time, it is still possible to speed it up. The slowest part currently is graph optimizer, so replacing *gtsam* with faster library or creating one can speed up the system
- **Dynamic objects** The dynamic objects in the sequence (e.g. people, cars etc) bring down the OKVIS estimation quality. Using semantic segmenta-

tion to detect such image regions and ignoring them for landmark estimation can reduce the negative impact.

- **Better loop closures** As we have seen in *E2* dataset, using only image similarity sometimes is not enough to perform reliable loop closures, because different areas can look almost perfectly similar while they are not the correct case for the loop closure.

Bibliography

[BFF11] Timothy Barfoot, James R Forbes, and Paul T Furgale. Pose estimation using linearized rotations and quaternion algebra. *Acta Astronautica*, 2011.

[BNG+16] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, 2006.

[CL68] C Chow and C Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 1968.

[CN08] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 2008.

[CN11] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 2011.

[Dav03] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings. Ninth IEEE International Conference on Computer Vision*, 2003.

[Del12] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. 2012.

[ESC14] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*. 2014.

[FCDS15] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *RSS*, 2015.

[FRS13] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *IEEE International Conference on Intelligent Robots and Systems*, 2013.

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

[GMW+12] Arren Glover, William Maddern, Michael Warren, Stephanie Reid, Michael Milford, and Gordon Wyeth. Openfabmap: An open source toolbox for appearance-based loop closure detection. In *IEEE International Conference on Robotics and Automation*, 2012.

[GSCI15] Ben Glocker, Jamie Shotton, Antonio Criminisi, and Shahram Izadi. Real-time RGB-D camera relocalization via randomized ferns for keyframe encoding. *IEEE Trans. Vis. Comput. Graph.*, 2015.

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[KF14] Laurent Kneip and Paul Furgale. Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *IEEE International Conference on Robotics and Automation*, 2014.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *International Conference on Computer Vision*, 2011.

[LLB+15] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 2015.

[LSB+15] Simon Lynen, Torsten Sattler, Michael Bosse, Joel Hesch, Marc Pollefeys, and Roland Siegwart. Get out of my lab: Large-scale, real-time visual-inertial localization. In *RSS*, 2015.

[LSCU12] Hyon Lim, Sudipta N. Sinha, Michael F. Cohen, and Matthew Uyttendaele. Real-time image-based 6-dof localization in large-scale environments. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[MAMT15] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 2015.

[MR07a] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics and Automation*, 2007.

[MR07b] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics and Automation*, 2007.

[MSUK14] Sven Middelberg, Torsten Sattler, Ole Untzelmann, and Leif Kobbelt. *Scalable 6-DOF Localization on Mobile Devices*. 2014.

[MTK+02] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, 2002.

[NRB+14] Janosch Nikolic, Joern Rehder, Michael Burri, Pascal Gohl, Stefan Leutenegger, Paul T Furgale, and Roland Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *IEEE International Conference on Robotics and Automation*, 2014.

[SEE+12] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[SGZ+13] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2013.

[SLX15] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[UESC16] Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. Direct visual-inertial odometry with stereo cameras. 2016.

[VH12] Jonathan Ventura and Tobias Höllerer. Wide-area scene mapping for mobile visual tracking. In *ISMAR*, 2012.