ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

V4RL VISION FOR ROBOTICS LAB
Vision for Robotics Lab
Prof. Dr. Margarita Chli

Semester Thesis

High Accuracy Visual Inertial SLAM for Autonomous Navigation of small UAVs

Spring Term 2018

Supervised by:
Marco Karrer
Patrik Schmuck
Prof. Dr. Margarita Chli

Author:
Thomas Ziegler

Declaration of Originality

I hereby declare that the written work I have submitted entitled

High Accuracy Visual Inertial SLAM for Autonomous Navigation of small UAVs

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Thomas	Ziegler

Student supervisor(s)

Marco	Karrer
Patrik	Schmuck

Supervising lecturer

Margarita	Chli

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Abstract

Unmanned Aerial Vehicles (UAVs) require accurate pose estimates with low latency in order to achieve a robust and stable flight behavior. However, due to the power and payload restrictions of aerial platforms, state estimation algorithms must provide these qualities under computational constraints. Monocular Visual Inertial Odometry (VIO) systems, consisting of a camera and an Inertial Measurement Unit (IMU), can satisfy these constraints and form the minimum sensor suite for metric 6 Degree of Freedom (DoF) pose estimation.

In this work, we propose an accurate keyframe based monocular VIO pipeline for onboard state estimation of UAVs. The publicly available Visual Inertial (VI) Simultaneous Localization and Mapping (SLAM) system VINS-Mono is therefore adapted. The performance of the VIO pipeline is evaluated for different parameter settings. Based on these evaluations, we propose a VIO pipeline running in real-time while only exhibiting a little loss in accuracy compared to the default implementation of VINS-Mono.

The proposed VIO pipeline is extensively tested on all sequences of the EuRoC dataset. Furthermore, the real-time applicability is confirmed by deploying and running the VIO pipeline on a real UAV.

Acronyms

Acronyms and Abbreviations

ASL Autonomous System Lab

BA Bundle Adjustment

EKF Extended Kalman Filter

DoF Degree of Freedom

IMU Inertial Measurement Unit

MSF Multi Sensor Fusion

ROS Robot Operating System

SfM Structure from Motion

SLAM Simultaneous Localization and Mapping

UAV Unmanned Aerial Vehicle

VI Visual Inertial

VIO Visual Inertial Odometry

VO Visual Odometry

Chapter 1

Introduction

Simultaneous Localization and Mapping (SLAM) is the task of navigating in a previously unknown environment by building a map of the environment which the robot is in and simultaneously estimating its position within this map.

SLAM systems can usually be divided into two basic components, the front-end and the back-end. The front-end builds a local map and estimates the robot's pose incrementally within this local map. Small deviations are accumulated over time which results in a drift of the estimated pose. If a previously visited location is detected, the back-end can correct for the drift with the help of loop closures and pose graph optimization. Although, a globally topologically consistent map can be achieved this way, the accumulated error cannot be removed completely. This means, the accuracy of the front-end is fundamental for the overall performance of a SLAM system.

Cameras are now easily found in many consumer electronic products. This makes systems using a single camera very appealing due to their small size and low power consumption. Furthermore, cameras can operate under different light conditions, both indoors and outdoors. Hence, monocular front-end systems have become very popular in recent years. However, it is known that a visual only front-end, implemented as monocular Visual Odometry (VO) system, cannot retrieve the scale of a scene. A popular solution for this problem is the integration of an IMU. The rich representation of a scene captured with a camera, together with the accurate short-term movement estimates by the IMU have been acknowledged to complement each other well.

The performance of VIO system is often evaluated using UAVs. Due to their fast dynamics and 6 DoF movements, they represent the most challenging type of robot. However, UAVs normally have high power and payload constraints and the state estimation must work on embedded hardware with limited computational resources.

Monocular SLAM or VIO solutions are either filter-based (e.g. using an Extended Kalman Filter (EKF)) or optimization based using keyframes. It has been shown [1] that keyframe based methods outperform filter-based ones, given enough computational power. Hence, most new releases of monocular SLAM systems are keyframe based. Keyframes provide the additional advantage that they can be used by simultaneously running algorithms at the same time. Keyframes can for example be exchanged between two robots to perform collaborative SLAM [2], [3] or keyframes can be used for path planning tasks [4]. Running other tasks simultaneously beside the SLAM system increases the computational payload significantly, reducing the

available computational resources dedicated for the SLAM task.

This work proposes an accurate keyframe based monocular VIO system for computationally restricted platforms. The systems builds on an existing open-source VIO implementation VINS-Mono [5]. The impact of the different parameter settings in the initial implementation of VINS-Mono are analyzed on a UP Board, a single board computer. Based on this evaluation an adaption is proposed in order to achieve an implementation with real-time performance. The resulting system is extensively tested on all sequences of the EuRoC micro aerial vehicle dataset [6]. Furthermore, the system is deployed on a real UAV to ensure its functionality.

Chapter 2

Related Work

There exists extensive research on vision-based odometry/SLAM systems. However, in this chapter we skip a full discussion on the available literature and focus only on work that this semester project relies on. In particular, a benchmark comparison of different VIO systems and a brief discussion on the publicly available VIO and VI-SLAM systems. The chapter ends with a short comparison of the suitability of these systems regarding the goals of this work.

2.1 A Benchmark Comparison

In [7], an extensive comparison of publicly available VIO pipelines (MSCKF, OKVIS, ROVIO, VINS-Mono, SVO+MSF, and SVO+GTSAM) is performed with respect to the per-frame processing time, CPU and memory load. The algorithms are evaluated on all sequences of the EuRoC datasets while processed on different single board computer systems. The results are presented as a benchmark.

2.2 MSCKF

The Multi-State constraint Kalman Filter (MSCKF) [8] is a popular EKF based VIO system. It maintains several previous camera poses in the state vector and uses geometric constraint between all the poses that observe a particular feature as update. There exist a publicly available implementations of this algorithm[1].

2.3 OKVIS

Open Keyframe-based Visual-Inertial SLAM (OKVIS) [9] is a state-of-the-art SLAM system that works with monocular and stereo cameras. However, it should be noted that it is not optimized for monocular VIO and its performance is superior in the stereo setup. It utilizes non-linear Bundle Adjustment (BA) optimization with a sliding window of keyframe poses. Keyframes older than the sliding window are marginalized out in order to retrieve the constraints. The source code of a Robot Operating System (ROS) implementation has been made publicly available[2].

[1] https://github.com/daniilidis-group/msckf_mono
[2] https://github.com/ethz-asl/okvis_ros

2.4 ROVIO

Robust Visual Inertial Odometry (ROVIO) [10] is a visual-inertial estimator based on an EKF. ThisVIO system uses the photometric error of tracked image patches to find the optimal state in the update step. The framework is specifically designed for monocular systems and does not require any special initialization procedure. The source code of a ROS implementation has been made publicly available[3].

2.5 SVO+GTSAM

Semi-Direct Monocular Visual Odometry (SVO) [11] is a computationally lightweight VO algorithm which employs sparse image alignment to estimate motion. It tracks features by minimizing the photometric error of image patches around the features between subsequent frames. The tracking is tightly coupled with an online factor graph optimization based on iSAM2 [12] for the state estimation of selected keyframes. The IMU measurements between two consecutive keyframes are pre-integrated as described in [13] in order to be used in the optimization. Both components SVO[4], and iSAM2 implemented in the GTSAM 4.0 optimization toolbox[5] [14], are publicly available. However, the integration of these two systems is not published and SVO is only provided as binaries.

2.6 VINS-Mono

VINS-Mono, presented in [5] is a complete VI-SLAM system. It uses non-linear BA optimization over a sliding window for the pose estimation. Robust corner features are tracked over consecutive frames. Similar to OKVIS, the optimization is performed over keyframes, which are marginalized out when they leave the window. Similar to SVO, the IMU measurements between consecutive keyframes are pre-integrated to reduce the computational demand. VINS-Mono also provides a tightly integrated relocalization. Together with the 4 DoF pose graph optimization it builds the back-end of the SLAM system. The source code is available as a ROS compatible implementation[6].

2.7 Summary

As stated in the introduction, in chapter 1, the goal of this work is to implement a low demanding monocular keyframe based VIO system. From the above mentioned systems MSCKF and ROVIO are both filter based and do not have a keyframe selection. SVO+GTSAM requires the source code to implement the tightly coupled system. The source code of SVO is only available in the initial version, which does not support front-looking cameras. The remaining systems are OKVIS and VINS-Mono. OKVIS is optimized for stereo cameras and has to re-integrate the IMU measurements in every optimization step, leading to computational overhead. VINS-Mono, on the other hand, is designed for a monocular setup and does support pre-integration of the IMU measurements in order to avoid repeated IMU re-integration. We therefore decided to use VINS-Mono as basis for this work.

[3] https://github.com/ethz-asl/rovio
[4] http://rpg.ifi.uzh.ch/svo2.html
[5] https://bitbucket.org/gtborg/gtsam/
[6] https://github.com/HKUST-Aerial-Robotics/VINS-Mono

Chapter 3

System overview

This chapter provides an overview of the VINS-Mono SLAM system proposed by [5] as visualized in Figure 3.1. The focus lies on the parts of the system which are crucial for this work whereas the rest is only described briefly. For a detailed description of the system we refer to [5].

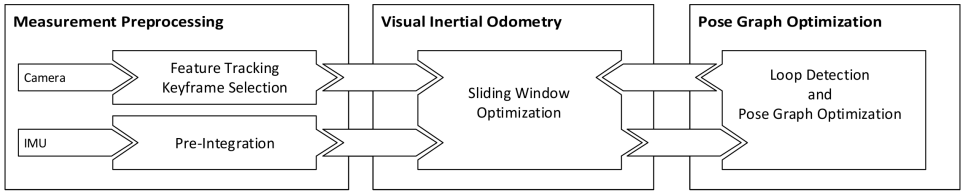


Figure 3.1: VINS-Mono SLAM system overview. Shown are the preprocessing module on the left side, the front-end in the middle and the back-end on the right side.

3.1 Measurement Preprocessing

In the preprocessing step (left side in Figure 3.1), features in the frames are extracted and tracked over consecutive frames using the KLT sparse optical flow algorithm [15]. To maintain a minimum number of features in each image, additional corner features are detected. Keyframes are also selected in the preprocessing step. The current frame is selected as keyframe if one of the following two criteria is fulfilled:

1. The average parallax apart from the previous keyframe is beyond a certain threshold.

2. The number of tracked features goes below a certain threshold.

The IMU measurements between two consecutive frames are pre-integrated including bias correction as proposed in [13].

3.2 Visual Inertial Odometry

For a robust and accurate state estimation a tightly-coupled monocular VIO pipeline is used (middle part in Figure 3.1). It performs a non-linear BA optimization followed by a marginalization.

3.2.1 Optimization

In order to bound the computational complexity, the optimization is performed in a sliding window manner. The optimization is a visual-inertial BA. The sum of prior and the Mahalonobis norm of all measurement residuals is minimized over the full state vector $\boldsymbol{\mathcal{X}}$ to obtain a maximum posteriori estimation:

$$\min_{\boldsymbol{\mathcal{X}}} \left\{ \|\mathbf{r}_p - \mathbf{H}_p \boldsymbol{\mathcal{X}}\|^2 + \sum_{k \in \mathcal{B}} \left\| \mathbf{r}_{\mathcal{B}} \left(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \boldsymbol{\mathcal{X}} \right) \right\|_{\mathbf{P}_{b_{k+1}}^{b_k}}^2 + \sum_{(l,j) \in \mathcal{C}} \rho \left(\left\| \mathbf{r}_{\mathcal{C}} \left(\hat{\mathbf{z}}_l^{c_j}, \boldsymbol{\mathcal{X}} \right) \right\|_{\mathbf{P}_l^{c_j}}^2 \right) \right\}, \tag{3.1}$$

where ρ is the Huber norm. $\mathbf{r}_{\mathcal{B}} \left(\hat{\mathbf{z}}_{b_{k+1}}^{b_k}, \boldsymbol{\mathcal{X}} \right)$ and $\mathbf{r}_{\mathcal{C}} \left(\hat{\mathbf{z}}_l^{c_j}, \boldsymbol{\mathcal{X}} \right)$ are the residuals for the IMU and the visual measurements, respectively. The IMU residual consists of constraints on the poses, the velocity, and the orientation of the IMU between two consecutive (key-)frames in the sliding window. The IMU residual also contains the acceleration and gyroscope bias. The visual measurement residual consist of constraints on the poses and the orientation of the (key-)frames and the features observed in these frames. A detailed description of the residuals can be found in [5]. $\{\mathbf{r}_p, \mathbf{H}_p\}$ is the prior information retrieved from the marginalization described in subsection 3.2.2. It contains constraints on the pre-integrated IMU factor between the marginalized and the subsequent keyframe and on the features observed in the marginalized keyframe.

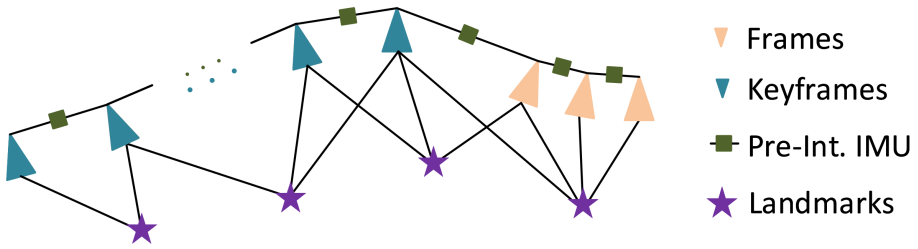
The BA optimization is performed over a sliding window, containing the latest two frames, a new arriving frame, and the eight previous keyframes. An illustration of the sliding window is shown in Figure 3.2, with the corresponding graph representation in Figure 3.3a.

Figure 3.2: Representation of the sliding window. The eight previous keyframes are marked in blue with the oldest one most left. The latest two frames in the window and a new arriving frame, illustrated on the right most side, are marked in orange.

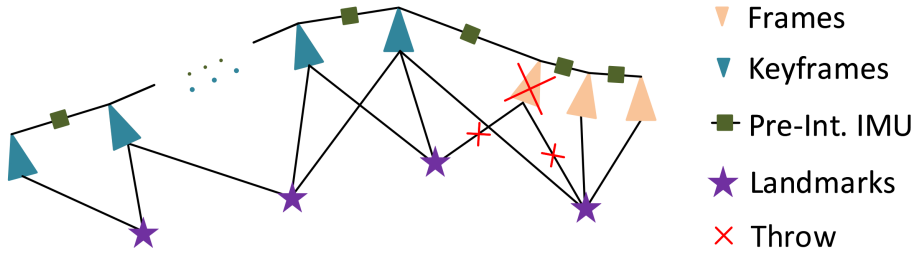
3.2.2 Marginalization

There are two possible cases after the BA optimization. If the second latest frame is not a keyframe, only the IMU measurements are kept and the visual measurements are dropped, as shown in Figure 3.3. In case the second latest frame is a keyframe, it will stay in the window and the oldest keyframe is marginalized, as shown in Figure 3.4. The marginalization is performed in order to preserve constraints between the IMU state and the features observed in the oldest keyframe.

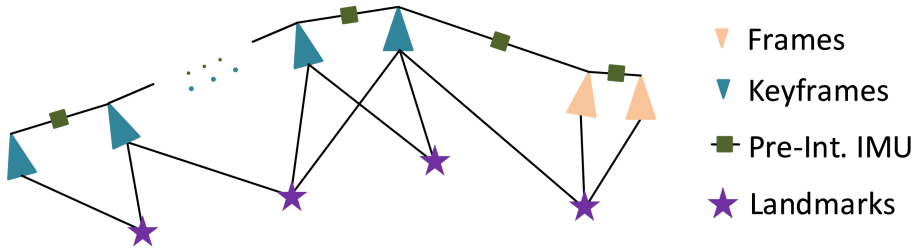
The marginalization is carried out using the Schur complement [16]. A new prior related to the removed state is constructed and added to the existing prior in Equation (3.1).



(a) Graph of the state variables and measurements the BA optimization is performed on.



(b) Graph of the state variables and measurements after the BA optimization. If the second latest frame is not a keyframe, it is removed together with its corresponding visual measurements. The pre-integrated IMU measurements are combined into one measurement·



(c) Graph of the state variables and measurements, after the BA optimization and marginalization, before a new frame arrives and the BA optimization is performed.

Figure 3.3: An illustration of the marginalization strategy (a)-(c) if the second latest frame is not a keyframe. Blue triangles mark the keyframe poses and orange triangles mark the latest two frame poses and the pose of the new arriving frame. The stars represent landmarks that are observed within the (key-)frames. The green squares represent the pre-integrated IMU measurements.

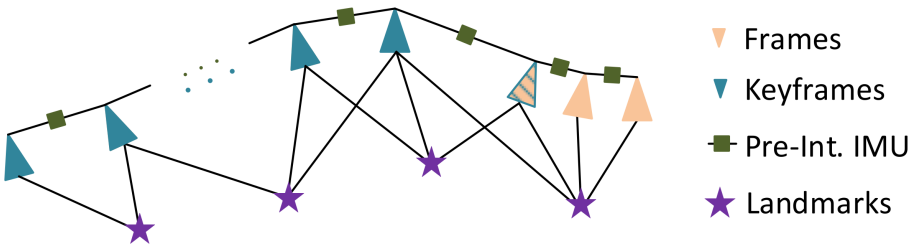
Schur Complement

Marginalization out parameters of a linear system matrix is equivalent to applying the Schur complement on this linear system matrix, as explained in [16]. For example, given the system

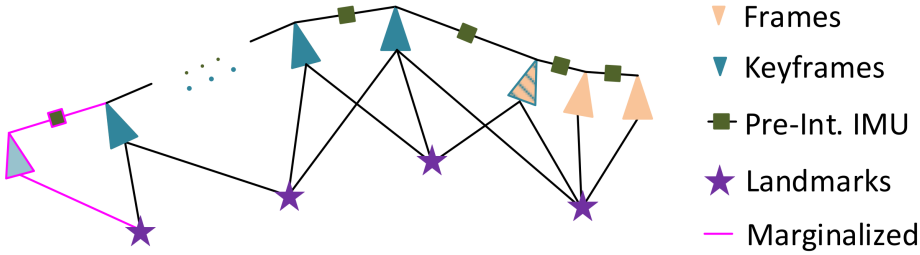
$$\begin{bmatrix} \mathbf{\Lambda}_a & \mathbf{\Lambda}_b \\ \mathbf{\Lambda}_b^\top & \mathbf{\Lambda}_c \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}_a \\ \delta\mathbf{x}_b \end{bmatrix} = \begin{bmatrix} \mathbf{g}_a \\ \mathbf{g}_b \end{bmatrix}. \tag{3.2}$$

reducing the parameters $\mathbf{x}_a$ onto the parameters $\mathbf{x}_b$ leads to

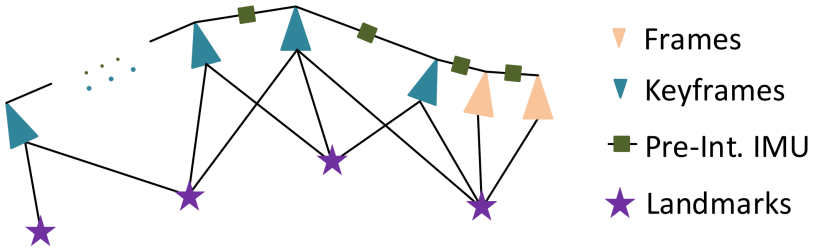
$$\begin{bmatrix} \mathbf{\Lambda}_a & \mathbf{\Lambda}_b \\ 0 & \mathbf{\Lambda}_c - \mathbf{\Lambda}_b^\top \mathbf{\Lambda}_a^{-1} \mathbf{\Lambda}_b \end{bmatrix} \begin{bmatrix} \delta\mathbf{x}_a \\ \delta\mathbf{x}_b \end{bmatrix} = \begin{bmatrix} \mathbf{g}_a \\ \mathbf{g}_b - \mathbf{\Lambda}_b^\top \mathbf{\Lambda}_a^{-1} \mathbf{g}_a \end{bmatrix}. \tag{3.3}$$



(a) Graph of the state variables and measurements the BA optimization is performed on.



(b) Graph of the state variables and measurements after the BA optimization. If the second latest frame is a keyframe, it is kept in the window, and the oldest keyframe and its corresponding visual and IMU measurements are marginalized out. Marginalized measurements are turned into a prior.



(c) Graph of the state variables and measurements, after the BA optimization and marginalization, before a new frame arrives and the BA optimization is performed.

Figure 3.4: An illustration of the marginalization strategy (a)-(c)if the second latest frame is a keyframe. Blue triangles mark the keyframe poses and orange triangles mark the latest two frame poses and the pose of the new arriving frame. The stars represent landmarks that are observed within the (key-)frames. The green squares represent the pre-integrated IMU measurements.

After this forward substitution step, the smaller lower-right system is independent of $\mathbf{x}_a$ and can be solved to update $\mathbf{x}_b$:

$$\left[\mathbf{\Lambda}_c - \mathbf{\Lambda}_b^\top \mathbf{\Lambda}_a^{-1} \mathbf{\Lambda}_b\right] [\delta \mathbf{x}_b] = \left[\mathbf{g}_b - \mathbf{\Lambda}_b^\top \mathbf{\Lambda}_a^{-1} \mathbf{g}_a\right]. \tag{3.4}$$

In our case, the pose estimation problem can be described as a 2×2 system of equations (for simplicity the IMU states are included in the pose states):

$$\begin{bmatrix} \mathbf{\Lambda}_m & \mathbf{\Lambda}_{mp} \\ \mathbf{\Lambda}_{mp}^\top & \mathbf{\Lambda}_p \end{bmatrix} \begin{bmatrix} \delta \mathbf{x}_m \\ \delta \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{g}_m \\ \mathbf{g}_p \end{bmatrix}. \tag{3.5}$$

Where $\mathbf{x}_m$ and $\mathbf{x}_p$ are composed of the observed 3D landmarks and the robot poses, respectively. The system matrix contains the "pose block" $\mathbf{\Lambda}_p$, the "map block"

$\mathbf{\Lambda}_m$, and the "observation block" $\mathbf{\Lambda}_{mp}$. On the right-hand side of Equation (3.5), $\mathbf{g}_m$ and $\mathbf{g}_p$ are the vectors corresponding to the robot path and the map, respectively.

This system has the same structure as Equation (3.2) and therefore marginalization can be performed via Schur complement.

3.3 Relocalization and Pose Graph Optimization

The backend of the SLAM system (right side in Figure 3.1) consist of a loop closure and pose graph optimization. Thanks to the addition of IMU, drift only occurs in 4 DoF (the global 3D position (x, y, z) and the rotation around the gravity direction), as described in [5]. To eliminate these drifts, VINS-Mono provides a tightly-coupled relocalization module which is seamlessly integrated with the monocular VIO.

After relocalization, the local sliding window is aligned with the past poses. The pose graph optimization module uses these relocalization results to perform a 4 DoF optimization to ensure the set of past poses is registered into a globally consistent configuration.

In order to reduce the computational demand of the overall system, the loop closure and pose graph optimization is deactivated for this work.

3.4 Implementation

The optimization Equation (3.1) is implemented using Google's *ceres solver* [17]. The default settings regarding the parameters of the feature tracker and the optimization are the following:

- Maximum number of tracked features throughout the keyframes in the sliding window: 150 features.
- Maximum number of solver iterations: 8 iterations.
- Maximal solving time: 40 milliseconds.

The two break conditions of the optimization ensure real-time performance of the optimization step irrespective of convergence. However, the followed marginalization step is not guaranteed to be performed before the next frame arrives, prohibiting real-time performance. Hence, if the overall optimization time (optimization and marginalization) between two consecutive frames has to be reduced, one needs to focus on the marginalization.

Chapter 4

Evaluation

In this chapter, different adaptions of the implementation of Equation (3.1) are made. Their effect on time and accuracy is analyzed on the MH3 sequence of the EuRoC micro aerial vehicle dataset [6]. This dataset provides stereo WVGA monochrome images at 20Hz and temporally synchronized IMU measurements at 200Hz. A ground truth trajectory is given by a Leica MS50 laser tracker. Only the left camera is used in this evaluation. The adaptions are analyzed on a UP Board with an Intel Atom x5-Z8350 CPU running at up to 1.92 GHz and compared to the default VINS-Mono settings running on a Laptop with an Intel i7-6600U CPU running at up to 3.4 GHz. If not stated differently, the default parameters as described in section 3.4 are used. The evaluation focuses on two metrics, the accuracy and the per-frame optimization time.

4.1 Metrics

4.1.1 Accuracy

The accuracy is measured with the translation error. We are most interested in the local accuracy of the VIO. Therefore, the accuracy is measured by averaging the drift over short trajectory segments of different length. The first 25 poses of the estimated trajectory are aligned with the corresponding ground truth poses using the *sim3* trajectory alignment proposed in [18]. The translation error is then measured as the Euclidean norm at the last pose of the trajectory segment. The initial pose is then moved by five poses and the alignment and error calculation is repeated. This metric is performed for trajectory segments of length 1m, 2m, 5m, 10m, 15m, 20m, 25m, and 30m. The rotation error around the gravity axis is not measured separately. Drift in the yaw-axis will implicitly create drift in the x and y axis which hence is covered with the translation error.

4.1.2 Per-Frame Optimization Time

The time spent for the optimization is crucial to ensure real-time performance. As described in section 3.4 VINS-Mono works with a refresh rate of 10Hz. This implies that the per-frame optimization should be performed within 100 milliseconds. Hence, the time used for the whole optimization including marginalization is measured. The marginalization has the most influence on the overall time, since the solving time of the non-linear BA optimization is limited as described in section 3.4.

4.2 Implementation of the Schur Complement

All the evaluations discussed below are performed after an important change in the implementation of the Schur complement. Meticulous timing analysis on the UP Board has shown that the following two lines of C++ code required abnormally high computation time.

```
A = A_rr - A_rm * A_mm_inv * A_mr;
b = b_rr - A_rm * A_mm_inv * b_mm;
```

Where "A's" are matrices and "b's" are vectors corresponding to the left-hand and right-hand side of Equation (3.4). The redundant matrix multiplication was moved into a temporary variable, leading to the following code:

```
A_tmp = A_rm * A_mm_inv;
A = A_rr - A_tmp * A_mr;
b = b_rr - A_tmp * b_mm;
```

This reduced the computation time of this operation to a reasonable value. However, this was not further investigated since this phenomenon was limited to the UP Board and the goal of this work is not specifically designed for the UP Board.

4.3 Reducing the Number of Tracked Features

As described in section 3.1 the preprocessing performs a feature tracking over consecutive frames. The default number of tracked features is 150. In this section, the influence of the number of tracked features is analyzed. Reducing the number of tracked features results in a lower dimension of the system matrix in Equation (3.5). This leads to a lower number of features to be marginalized. The reduced optimization time can be seen in Figure 4.1. In Figure 4.2 one can see how the accuracy decreases noticeable with less tracked features, as shown in. Hence, simply reducing the tracked feature until real-time performance is reached is not a satisfactory solution.

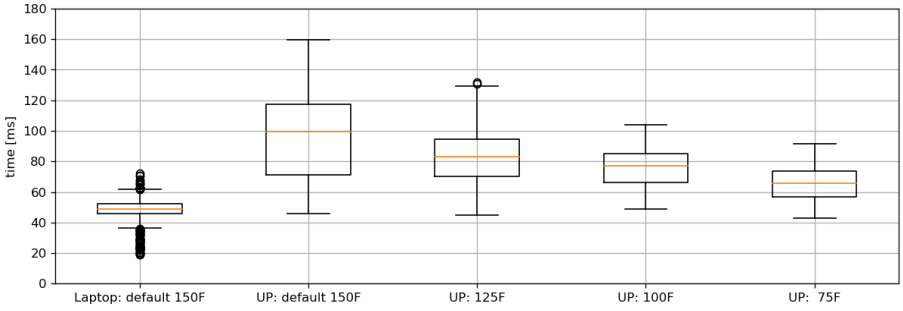


Figure 4.1: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline when changing the number of tracked features over consecutive frames. The analyzed number of tracked features are {150, 125, 100, 75}. The first number is the default setting and is evaluated on a Laptop and on a UP Board, all other numbers are evaluated on a UP Board. The average numbers of solver iterations for the five different cases are $\{6.00, 2.06, 2.25, 2.74, 3.05\}$ *iterations.*

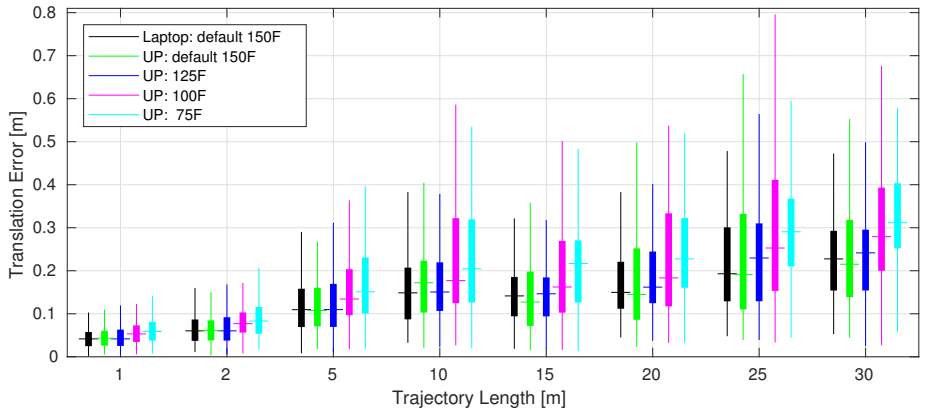


Figure 4.2: Boxplot summarizing the translation error statistic for the VIO pipeline when changing the number of tracked features over consecutive frames. The analyzed number of tracked features are {150, 125, 100, 75}. The first number is the default setting and is evaluated on a Laptop and on a UP Board, all other numbers are evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length {1,2,5,10,15,20,25,30} m.

4.4 Changing the Sliding Window Size

The non-linear BA is performed over a sliding window containing the latest two frames and a number of previous keyframes. The default window size is 10, which means that the number of previous keyframes is eight. In this section the impact of the window size is analyzed.

The reduction in the number of keyframes has influence on the optimization and the marginalization. With less keyframes, the number of poses in the "pose block" $\mathbf{\Lambda}_p$ in Equation (3.5) is reduced, as every keyframe leads to a pose. Also the number of corresponding landmark observations in the keyframes is potentially reduced. For an alternative explanation we can take a closer look at the dimensions of the system matrix in Equation (3.5). With a reduced number of poses, the dimensions of the symmetric "pose block" $\mathbf{\Lambda}_p$ is reduced. Therefore, the number of columns of the "observation block" $\mathbf{\Lambda}_{mp}$ has to be reduced as well. This also results in a reduced system for marginalization. The resulting decreased optimization times are shown in Figure 4.3.

Furthermore, a smaller window size reduces the freely adjustable parameters in the optimization. The solver can then perform more iterations in the given limited solving time. The increased number of iterations results in a higher convergence rate, which counteracts the decreased accuracy due to the smaller window size. Because of this, the translation error does not increase much along the different window sizes as seen in Figure 4.4.

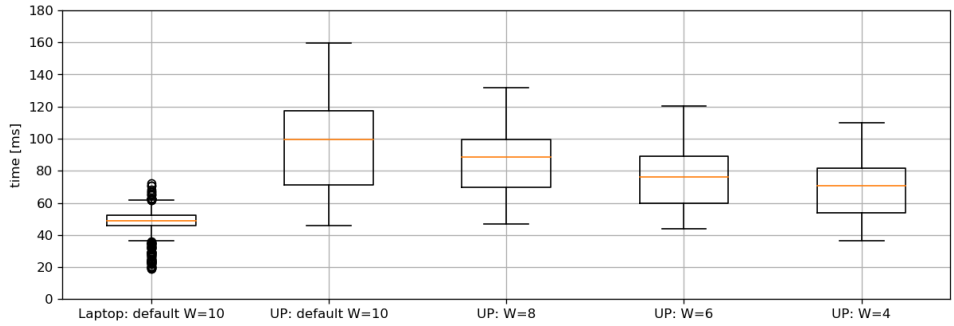


Figure 4.3: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline when changing number of keyframes in the sliding window. The analyzed number of keyframes in the window are $\{8, 6, 4, 2\}$*, which results together with the latest two frames in a window size of* $\{10, 8, 6, 4\}$*. The first number is the default setting and is evaluated on a Laptop and on a UP Board, all other numbers are evaluated on a UP Board. The average numbers of solver iterations for the five different cases are* $\{6.00, 2.06, 2.69, 3.21, 4.73\}$ *iterations.*

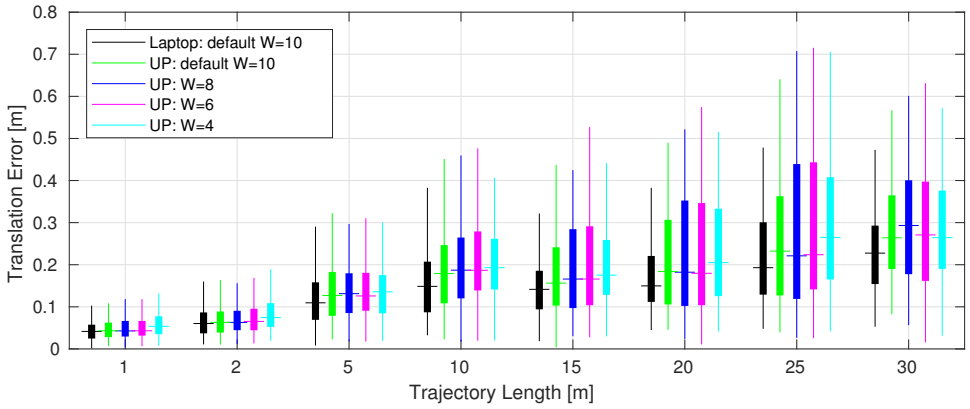


Figure 4.4: Boxplot summarizing the translation error statistic for the VIO pipeline when changing number of keyframes in the sliding window. The tested number of keyframes in the window are $\{8, 6, 4, 2\}$*, which results together with the latest two frames in a window size of* $\{10, 8, 6, 4\}$*. The first number is the default setting and is evaluated on a Laptop and on a UP Board, all other numbers are evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length* $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

4.5 Setting Features Constant in Optimization

Another method to reduce the number of freely adjustable variables in the optimization is to set the constraints of landmarks observed in the oldest keyframes in the window to constants. The idea behind this approach is, that the features corresponding to these landmarks have already been optimized several times. The constraints of these features are added as residuals, but will not change during the optimization. The resulting changes in per-frame optimization time and accuracy can be seen in Figure 4.5 and Figure 4.6, respectively. This approach has almost no effect on the overall optimization time. The entries in the system matrix stay the same and hence, the number of factors in the marginalization does not change either. One can see a significant decrease in accuracy for this approach. The reason

is assumed to be the method how VINS-Mono represents the features. They are not represented as a position in the 3D map but as a single parameter, the inverse depth. This means, setting the inverse depth constant restricts much more than setting the feature position constant.

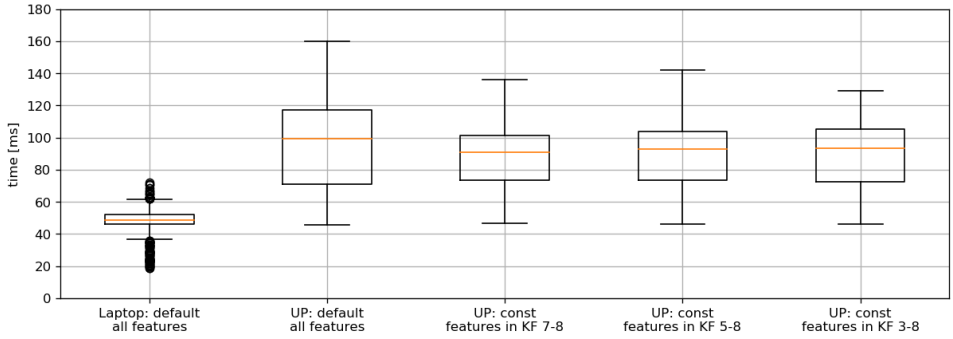


Figure 4.5: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline when setting the feature constraints as constants in the BA optimization for a number of oldest keyframes in the sliding window. The analyzed numbers of oldest keyframes are $\{0, 2, 4, 6\}$ *corresponding to the keyframe position {none, KF* $7-8$*, KF* $5-8$*, KF* $3-8$*}. The default setting is evaluated on a Laptop and a UP Board. The different number of keyframes, where the feature constraints are constant, is evaluated on a UP Board. The average numbers of solver iterations for the five different cases are* $\{6.00, 2.06, 2.56, 2.66, 2.73\}$ *iterations.*

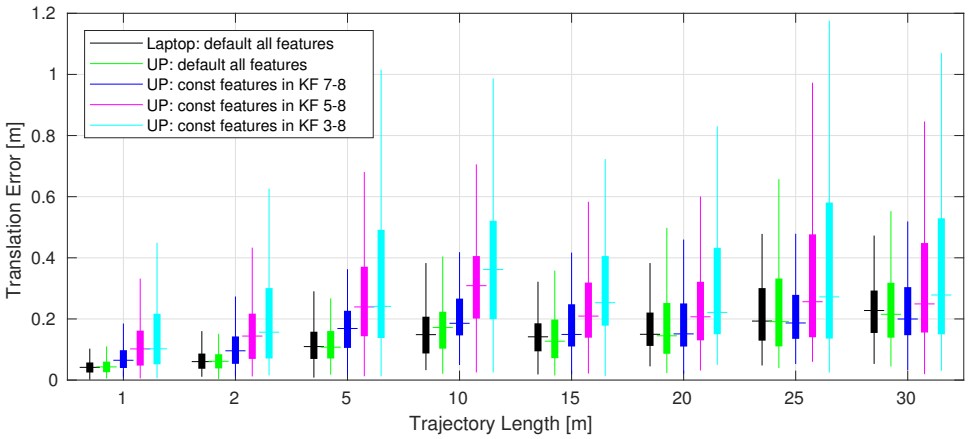


Figure 4.6: Boxplot summarizing the translation error statistic for the VIO pipeline when setting the constraints as constants in the BA from the features observed in the oldest keyframes in the sliding window. The analyzed numbers of oldest keyframes are $\{0, 2, 4, 6\}$ *corresponding to the keyframe position {none, KF* $7-8$*, KF* $5-8$*, KF* $3-8$*}. The default setting is evaluated on a Laptop and a UP Board. The different number of keyframes, where the feature constraints are constant, is evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length* $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

4.6 Not adding Features of the oldest Keyframes to the Bundle Adjustment

A similar adaptation as changing the window size is as follows. For a number of old keyframes in the window we do not add any feature constraints. This is equal to reducing the optimization to a pose graph optimization for these keyframes.
Due to the implementation of VINS-Mono, the marginalization is not affected by this change since the system matrix in Equation (3.5) is created independently of the parameters the optimization is performed on. As shown in Figure 4.7 the per-frame optimization time decreases only slightly. The translation error increases only barely, as shown in Figure 4.8. This implies that the constraints from features do not have as much influence in the older keyframes as they have in the newer (key-)frames. The reason is, that the older keyframes have already been optimized several times when they reach the end of the sliding window.

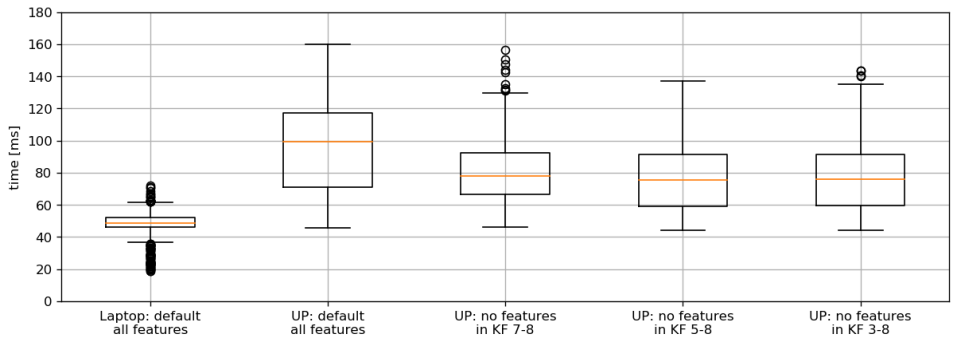


Figure 4.7: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline when ignoring the feature constraints in the BA optimization in a number of oldest keyframes in the sliding window. The analyzed numbers of oldest keyframes in which no feature constraints are added are $\{0, 2, 4, 6\}$ *corresponding to the keyframe position* {*none, KF* 7−8*, KF* 5−8*, KF* 3−8}*. The default setting is evaluated on a Laptop and a UP Board. The different number of keyframes where the feature constraints are ignored are evaluated on a UP Board. The average numbers of solver iterations for the five different cases are* $\{6.00, 2.06, 2.17, 3.09, 3.10\}$ *iterations.*

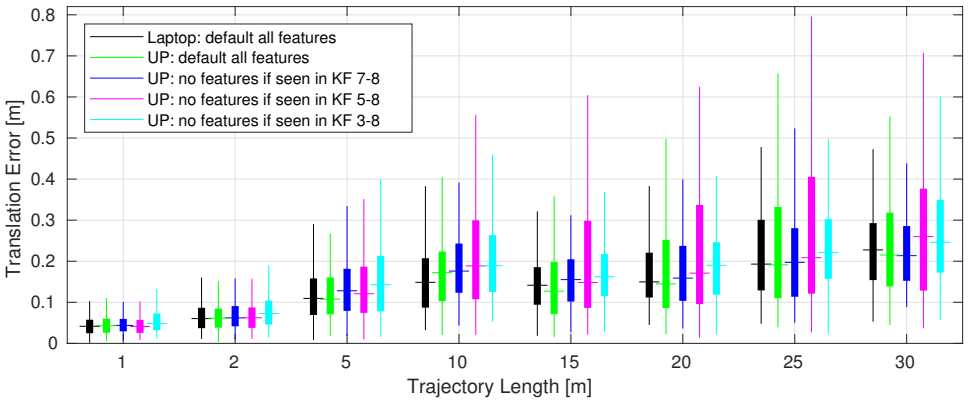


Figure 4.8: Boxplot summarizing the translation error statistic for the VIO pipeline when ignoring the feature constraints in the BA optimization in a number of oldest keyframes in the sliding window. The analyzed numbers of oldest keyframes in which no feature constraints are added are $\{0, 2, 4, 6\}$ *corresponding to the keyframe position* {*none, KF* 7−8*, KF* 5−8*, KF* 3−8}*. The default setting is evaluated on a Laptop and a UP Board. The different number of keyframes where the feature constraints are ignored are evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length* $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

4.7 Removing Features from the Bundle Adjustment

We looked for other ways to reduce the number of parameters the optimization is performed on. A similar but more radical approach as the one described before is ignoring landmarks, which are observed in at least one of the oldest keyframes in the sliding window, in all (key-)frames in the BA optimization. This allows the solver to perform more iterations in the given solving time similar to the approach of reducing the sliding window size. As in the previous approach, the marginalization is not affected by this change. The effect on the optimization time is almost negligible, as shown in Figure 4.9. However, if too many landmarks are ignored, the translation error increases noticeably, as shown in Figure 4.10. Otherwise, the accuracy is affected only slightly.

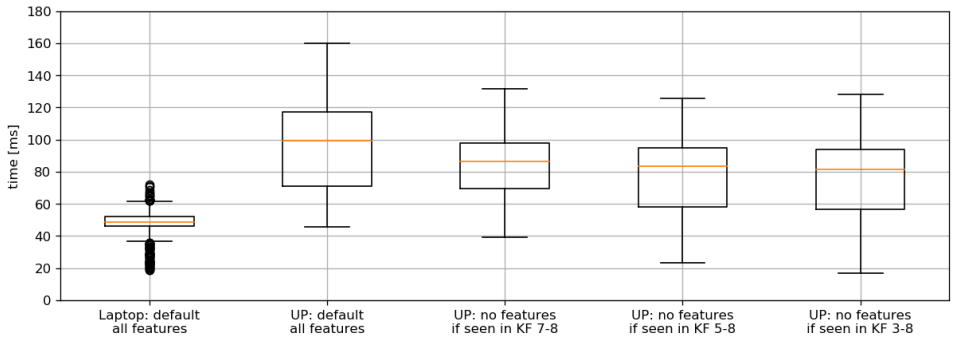


Figure 4.9: Boxplot summarizing the whole optimization time for the VIO pipeline when ignoring the feature constraints in the BA optimization in all (key-)frames if the feature has been observed in one of a number of oldest keyframes in the sliding window. The analyzed numbers of oldest keyframes, in which no feature constraints are added, are $\{0, 2, 4, 6\}$ *corresponding to the keyframe position* $\{none,\ KF\ 7-8,\ KF\ 5-8,\ KF\ 3-8\}$*. The default setting is evaluated on a Laptop and a UP Board. The different number of old keyframes, which decide if a feature constraints, is ignored in all (key-)frames is evaluated on a UP Board. The average numbers of solver iterations for the five different cases are* $\{6.00, 2.06, 4.72, 5.37, 6.78\}$ *iterations.*

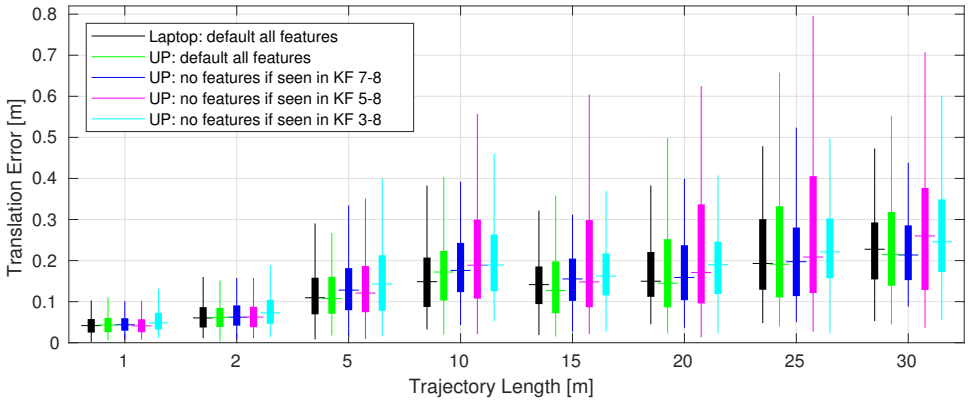


Figure 4.10: Boxplot summarizing the translation error statistic for the VIO pipeline when ignoring the feature constraints in the BA optimization in all (key-)frames if the feature has been observed in one of a number of oldest keyframes in the sliding window. The analyzed numbers of oldest keyframes, in which no feature constraints are added, are $\{0, 2, 4, 6\}$ *corresponding to the keyframe position* $\{none,\ KF\ 7-8,\ KF\ 5-8,\ KF\ 3-8\}$*. The default setting is evaluated on a Laptop and a UP Board. The different number of old keyframes, which decide if a feature constraints, is ignored in all (key-)frames is evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length* $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

4.8 Skipping Marginalization

This evaluations focuses on the marginalization. The effect of ignoring the marginalization fully can be seen in Figure 4.11 and Figure 4.12. We can see that the overall optimization time is drastically reduced, confirming that the marginalization is responsible for the increased timing on the UP Board. However, we can also see that the accuracy is reduced drastically, implying that ignoring the marginalization is not an option.

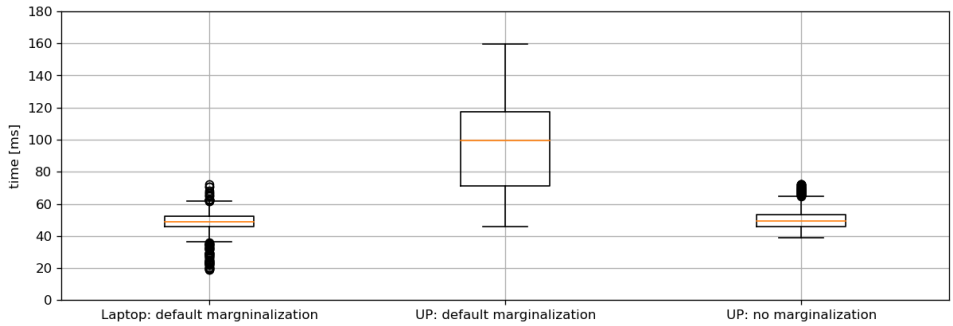


Figure 4.11: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline when marginalization is performed compared to no marginalization. The default marginalization is evaluated on a Laptop and on a UP Board and no marginalization is evaluated on a UP Board. The average numbers of solver iterations for the three different cases are $\{6.00, 2.06, 2.30\}$ *iterations.*

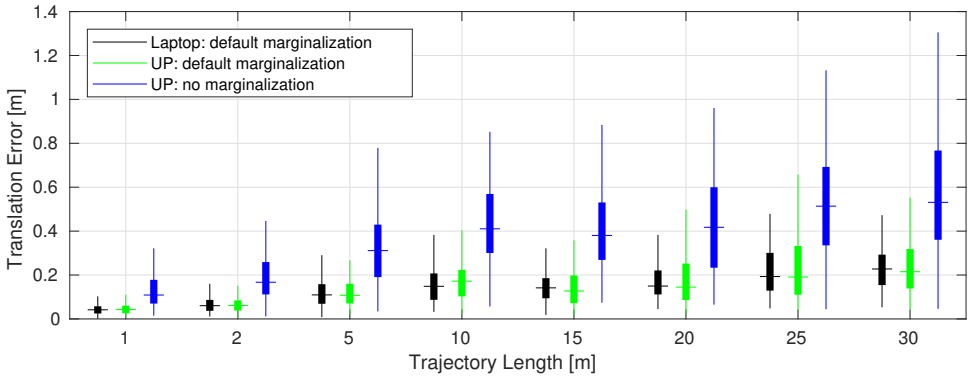


Figure 4.12: Boxplot summarizing the translation error statistic for the VIO pipeline when marginalization is performed compared to no marginalization. The default marginalization is evaluated on a Laptop and on a UP Board and no marginalization is evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

4.9 Removing Features from Marginalization

To reduce the number of parameters in the marginalization the following approach was analyzed. Features are only marginalized if their corresponding landmarks have been observed in more than a certain number of keyframes. This reduces the dimensions of the system matrix in Equation (3.5), leading to a reduced marginalization time as shown in Figure 4.13.

Reducing the parameters in the marginalization has also impact on the prior in the non-linear BA. A prior is only added for features that have been marginalized, which implicitly reduces the freely adjustable parameters in the optimization. This again allows for more iterations in the same solving time. If only landmarks observed in more than two keyframes are marginalized the translation error decreases. However, if too many landmarks are ignored in the marginalization the absence of the corresponding prior in the BA optimization leads to increased translation errors. These effects are shown in Figure 4.14.

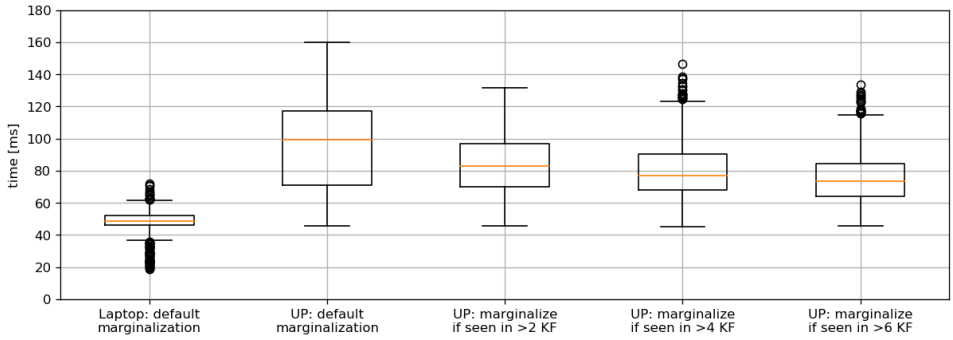


Figure 4.13: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline when only marginalize features if their corresponding landmarks have been observed in a minimum number of (key-)frames in the sliding window. The analyzed numbers of (key-)frames are $\{0, 2, 4, 6\}$*. The default setting is evaluated on a Laptop and a UP Board. The minimum number of (key-)frames a landmark has to be observed in is evaluated on a UP Board. The average numbers of solver iterations for the five different cases are* $\{6.00, 2.06, 2.15, 2.15, 2.14\}$ *iterations.*

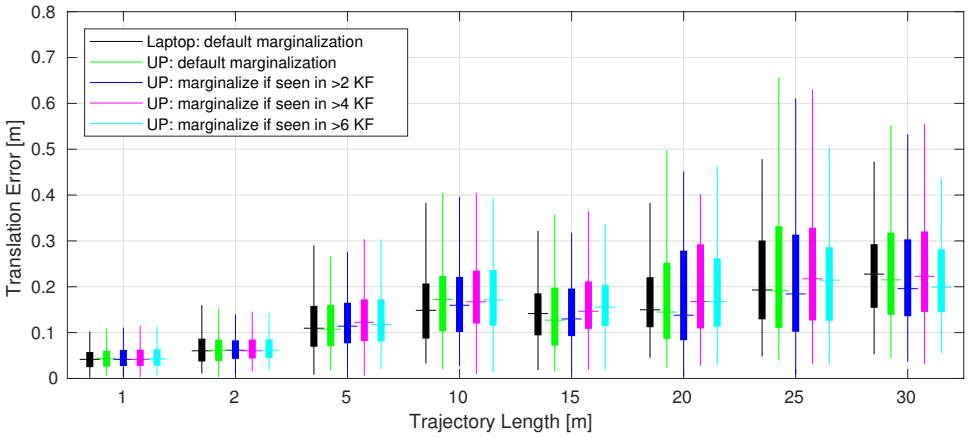


Figure 4.14: Boxplot summarizing the translation error statistic for the VIO pipeline when only marginalize features if their corresponding landmarks have been observed in a minimum number of (key-)frames in the sliding window. The analyzed numbers of (key-)frames are $\{0, 2, 4, 6\}$*. The default setting is evaluated on a Laptop and a UP Board. The minimum number of (key-)frames a landmark has to be observed in is evaluated on a UP Board. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length* $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

4.10 Conclusion

As mentioned in the beginning of this chapter, the overall optimization time consists of the time to solve the BA optimization and the time needed for the marginalization.
A first setting to reduce the overall optimization time is to limit the number of tracked features.
The maximal time the solver can spend for the BA optimization is fixed and in most of the cases the termination criteria of the optimization. Therefore, reducing the number of parameters the optimization is performed on does not decrease the optimization time but increases the number of iterations the optimizer can perform in the given solving time, which results in an increased accuracy.
The best way to decrease the number of parameters, the BA optimization has to optimize, while still achieving a reasonable accuracy is the approach presented in section 4.7.
The best trade off between time and accuracy for marginalization achieved the approach in which only landmarks are marginalized which were observed in more than a certain number of keyframes described in section 4.9.
Based on these insights from the evaluation, we build our proposed approach, presented in the next chapter.

Chapter 5

Approach

As described in the evaluation, the overall optimization time consists of the time to solve the BA optimization and the time needed for the marginalization. The runtime for the BA optimization is limited, but the accuracy can be affected depending on the optimization settings. The marginalization time, on the other hand, depends on the number of features to be marginalized.

To achieve real-time performance for our VIO pipeline, the total optimization time has to be below 100 milliseconds as our system runs with a refresh rate of 10Hz. Based on the evaluation in chapter 4, we propose an approach which fulfills this real-time requirement.

Our approach reduces the number of tracked features $n_{\text{features\_tracked}}$ and combines the optimization- and marginalization-settings which lead to the best trade off between time and accuracy. In the optimization we ignore landmarks which were observed in at least one of a certain number of old keyframes $n_{\text{oldest\_keyframes}}$. In the marginalization we do not marginalize landmarks which are observed in less than a certain number of (key-)frames $n_{\text{observed\_keyframes}}$.

In summary, the proposed changes are:

- Reduce the number of tracked features from 150 down to $n_{\text{features\_tracked}}$.
- Ignoring feature constraints in the BA optimization, if the corresponding landmark has been observed in at least one of the $n_{\text{oldest\_keyframes}}$ oldest keyframes.
- Only marginalize out features that have been observed in more than $n_{\text{observed\_keyframes}}$ (key-)frames.

Chapter 6

Experimental Results

In this chapter we evaluate the performance of the proposed approach, described in chapter 4. Two experiments are performed: In the first experiment, we compare our approach running on three different platforms, two of them computationally restricted. The platforms are a laptop computer, a UP-Board, and the onboard computer of an AscTec Hummingbird[1]. Additionally, we also run the default implementation of VINS-Mono on the laptop. This first experiment was performed on all sequences of the EuRoC dataset [6]. In the second experiment our approach is deployed on a AscTec Neo[2], a real UAV.

6.1 Experimental Settings

6.1.1 First Experiment

Hardware Platforms

The tests are performed on a Laptop with an Intel i7-6600U CPU running at up to 3.4 GHz, on a UP Board with an Intel Atom x5-Z8350 CPU running at up to 1.92 GHz, and on the onboard computer of an AscTec Hummingbird with an Intel Atom E3845 CPU running at 1.92GHz.

Metrics

We measure the same two metrics (accuracy and per-frame optimization time) as described in the evaluation section 4.1. However, the metrics are measured over all the sequences combined. In addition for each individual sequence, the whole trajectory is aligned to the ground truth using a *sim3* trajectory alignment according to the method proposed in [18]. Then the RMSE position error over the aligned trajectory is calculated. We also highlight the maximal per-frame optimization time for each sequence to ensure that real-time performance is fulfilled.

6.1.2 Second Experiment

Hardware Platforms

In the second experiment, the proposed pipeline has been deployed on an AscTec Neo, equipped with an Intel NUC which has much more computational power than required. Due to time limits the pipeline could not be deployed on a UAV with a more restricted onboard computer. However the results in subsection 6.3.1 show

[1] http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-hummingbird/
[2] http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-neo/

that a similar accuracy on a less powerful platform can be assumed.

The VIO pose estimation uses camera and IMU measurements obtained by the VI-Sensor described in [19]. Instead of the front-looking camera an additional down-looking camera has been used. The calibration between the additional camera and the IMU was performed using the *kalibr* toolbox [20] from Autonomous System Lab (ASL)[3]. In addition, the proposed VIO pose estimation has been fused with the onboard IMU of the UAV using an EKF Multi Sensor Fusion (MSF) framework from ASL[4] described in [21]. *kalibr* toolbox was again used for the calibration between the onboard IMU and the camera frame.

6.2 Experiments

Both experiments are evaluated with the parameters of our approach set to the following:

- $n_{\text{features\_tracked}} = 120$ features.
- $n_{\text{oldest\_keyframes}} = 4$ keyframes.
- $n_{\text{observed\_keyframes}} = 6$ (key-)frames.

6.2.1 First Experiment

We evaluate our proposed approach using the EuRoC micro aerial vehicle dataset [6]. This dataset provides stereo WVGA monochrome images at 20Hz and temporally synchronized IMU measurements at 200Hz from a micro-aerial vehicle manually piloted around three different indoor environments. Within each environment three qualitative difficulties are provided. For example, Machine Hall 01 is "easy" containing rather slow motions, while Machine Hall 05 is much more challenging, introducing fast motions, poor illuminations, etc. Each sequence provides a ground truth trajectory given by a Leica MS50 laser tracker for the Machine Hall environment and by a Vicon motion capture system for the other two environments. Only the images from the left camera are used for the evaluation.
All sequences of the dataset are used to show the performance of our approach in as many different scenarios as possible. As comparison the default VINS-Mono implementation running on the Laptop is also evaluated.

6.2.2 Second Experiment

Due to time limits the test were limited to hover at the current position. The pose estimations of the MSF were recorded together with ground truth provided by a Vicon motion capture system.

6.3 Results

6.3.1 First Experiment

First and most important, with our approach the optimization time is below 100 milliseconds in all sequences, compared to the default VINS-Mono running on the UP-Board. The results are listed in Table 6.2. We can see that the accuracy of

[3] https://github.com/ethz-asl/kalibr
[4] https://github.com/ethz-asl/ethzasl_msf

our approach has decreased compared to the default version of VINS-Mono running on the laptop, best noticeable in the median in the box plot in Figure 6.2. Furthermore, one can see that the accuracy of our approach on the Laptop and the UP-Board is very similar. The only difference on the UP-Board is the increase in computational time, as shown in Figure 6.1. This indicates the robustness of our algorithm, meaning the algorithm does not run at the limit on the UP Board. On the AscTec Hummingbird the accuracy is slightly worse. We assume the reason is the older CPU generation which lacks of the Intel Burst function for short-time overclocking of the CPU. The RMSE in Table 6.1 shows that our approach even outperforms the default VINS-Mono in the "easy" sequences of the Machine Hall and V1 environment. Whereas the default VINS-Mono implementation running on the laptop achieves higher accuracy in the difficult sequences. During the fast movements of these sequences the feature tracker is not as accurate as during slow movements. We assume that during fast movements, more constraints from the observed features improve the accuracy. During slow movements the feature tracking is very accurate and running the BA optimization with only a reduced number of feature constraints is sufficient enough, allowing for more iteration in the limited solving time. This results in a higher accuracy.

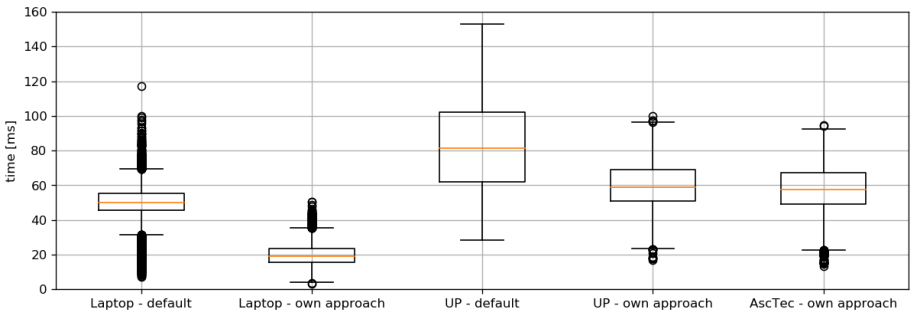


Figure 6.1: Boxplot summarizing the whole optimization time (optimization and marginalization) for the VIO pipeline over all sequences of the EuRoC dataset. The default VINS-Mono implementation running on a laptop and on a UP Board is compared with our own approach running on a laptop, a UP-Board and on the onboard computer of an AscTec Hummingbird UAV.

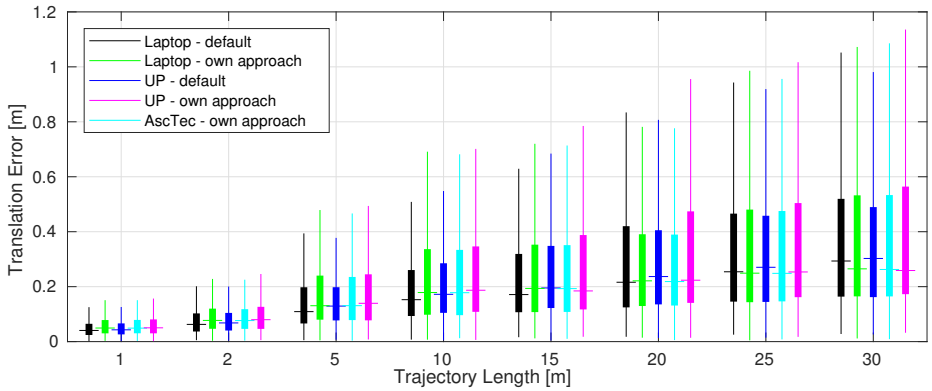


Figure 6.2: Boxplot summarizing the translation error statistic for the VIO pipeline over all sequences of the EuRoC dataset. The default VINS-Mono implementation running on a laptop and on a UP Board is compared with our own approach running on a laptop, a UP-Board and on the onboard computer of an AscTec Hummingbird UAV. Errors were computed using the metric described in subsection 4.1.1 for trajectory segments of length $\{1, 2, 5, 10, 15, 20, 25, 30\}$ *m.*

	Laptop default VINS-Mono	UP Board default VINS-Mono	Laptop	UP Board	AscTec Hummingbird
MH 01 - easy	0.427	0.284	0.139	0.140	0.174
MH 02 - easy	0.238	0.375	0.179	0.180	0.149
MH 03 - medium	0.211	0.246	0.175	0.177	0.144
MH 04 - difficult	0.223	0.365	0.276	0.262	0.232
MH 05 - difficult	0.351	0.361	0.564	0.548	0.557
V1 01 - easy	0.142	0.116	0.066	0.066	0.095
V1 02 - medium	0.063	0.062	0.126	0.110	0.085
V1 03 - difficult	0.091	0.147	0.126	0.127	0.110
V2 01 - easy	0.066	0.087	0.067	0.067	0.074
V2 02 - medium	0.090	0.087	0.093	0.094	0.095
V2 03 - difficult	0.123	0.187	0.172	0.172	0.167

Table 6.1: Absolute translation errors (RMSE) in meters for all sequences of the EuRoC dataset. The default VINS-Mono implementation running on a laptop is compared with our own approach running on a laptop, a UP-Board and on the onboard computer of an AscTec Hummingbird UAV. Errors have been computed after the estimated trajectories were aligned with the ground truth trajectory using the method proposed in [18].

	Laptop default VINS-Mono	UP Board default VINS-Mono	Laptop	UP Board	AscTec Hummingbird
MH 01 - easy	86.1	148.2	35.9	97.3	94.6
MH 02 - easy	99.9	148.8	40.9	95.9	94.0
MH 03 - medium	70.3	145.8	41.2	99.7	89.6
MH 04 - difficult	97.9	137.4	47.7	96.0	91.0
MH 05 - difficult	80.6	135.2	46.2	88.6	89.6
V1 01 - easy	117.2	152.9	45.4	95.5	91.5
V1 02 - medium	65.9	125.6	42.8	95.6	80.8
V1 03 - difficult	66.7	135.7	44.2	87.6	88.5
V2 01 - easy	71.2	135.5	38.9	88.9	91.6
V2 02 - medium	66.7	118.8	39.7	97.4	87.6
V2 03 - difficult	65.1	129.5	50.7	87.1	80.4

Table 6.2: Maximal optimization time in milliseconds for all sequences of the EuRoC dataset. The default VINS-Mono implementation running on a laptop is compared with our own approach running on a laptop, a UP-Board and on the onboard computer of an AscTec Hummingbird UAV.

6.3.2 Second Experiment

Since the UAV is supposed to hover, the relative pose should stay at zero for x, y, and z. We can see that in all axis, there exist a noticeable error in the state estimate of the MSF filter, shown in Figure 6.3, Figure 6.4, and Figure 6.5. We assume that the main reason for this behavior is that the filter parameters were not optimized. The test was performed with default filter values and due to time restrictions, no parameter tuning for VINS-Mono and/or our approach could be done. The calibration between the down-looking camera and the on-board IMU has also some potential for improvement.

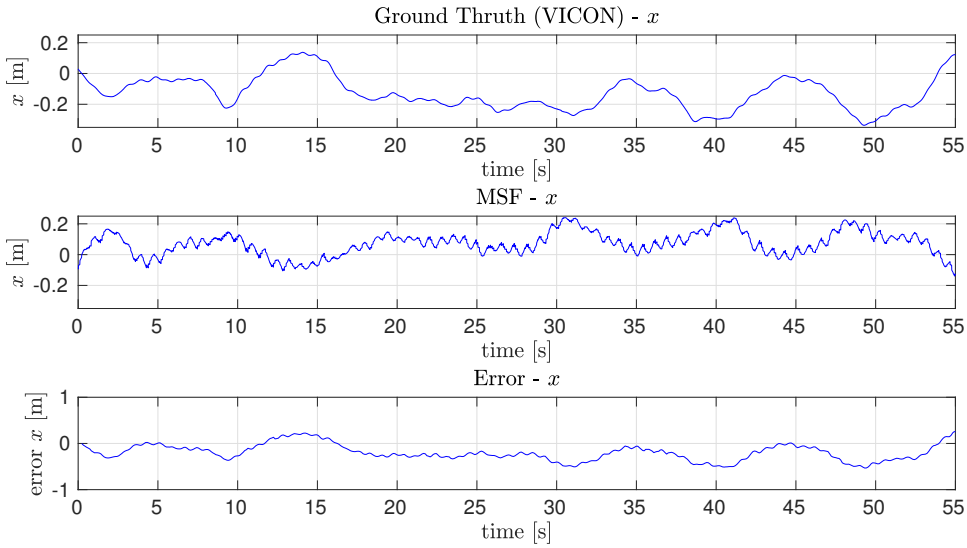


Figure 6.3: Relative pose estimation, while the UAV is hovering, in the x axis of the MSF filter and the Vicon ground truth, and the difference between them.

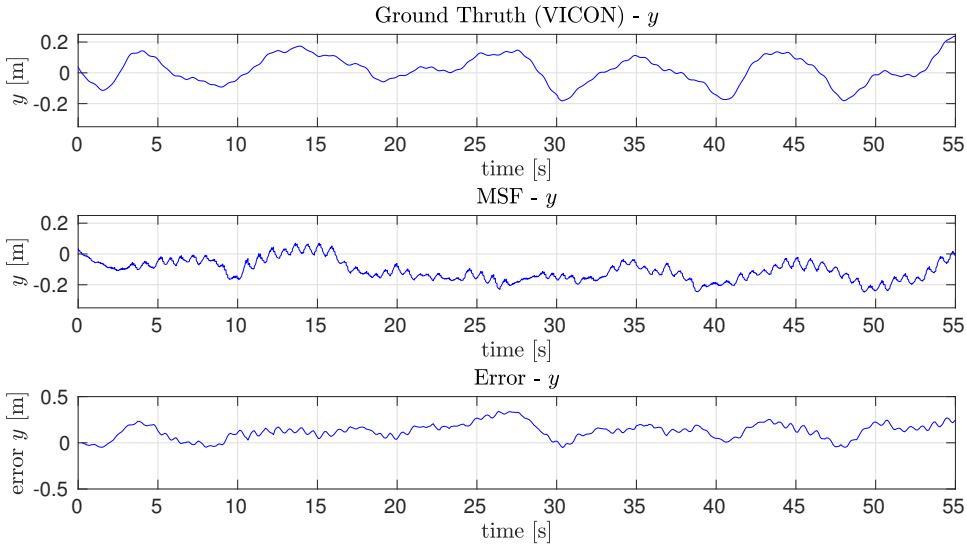


Figure 6.4: Relative pose estimation, while the UAV is hovering, in the y axis of the MSF filter and the Vicon ground truth, and the difference between them.

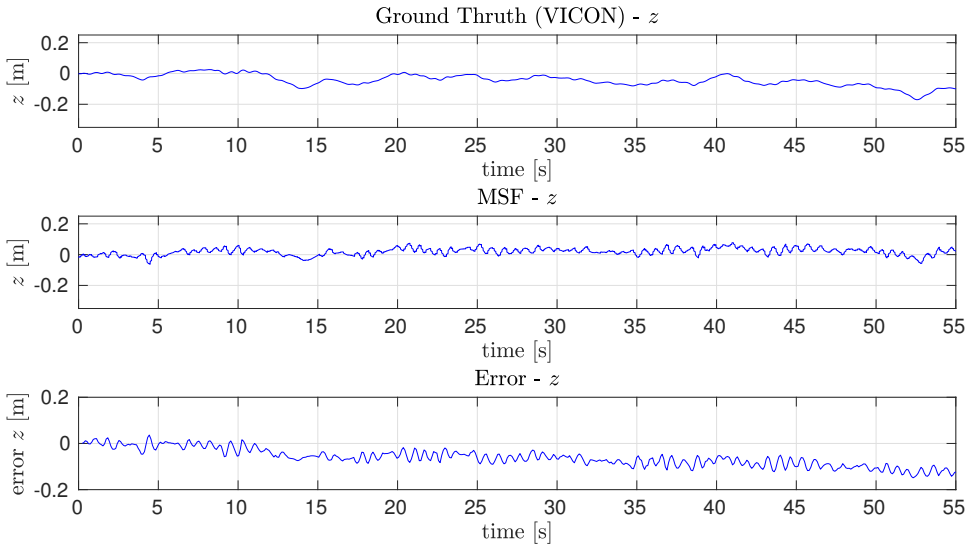


Figure 6.5: Relative pose estimation, while the UAV is hovering, in the z axis of the MSF filter and the Vicon ground truth, and the difference between them.

Chapter 7

Conclusion and Outlook

7.1 Conclusion

In this semester project, an existing open source monocular keyframe based VIO pipeline was adapted to achieve accurate pose estimation, while having as little computational demand as possible. Our approach is based on VINS-Mono [5], which was chosen after a comparison of the publicly available monocular VIO pipelines.

An extensive evaluation of the default VINS-Mono pipeline parameters was performed to determine their influence on the accuracy and the per-frame optimization time. Based on these evaluation an approach was proposed, which ignores any features in the BA optimization, if the corresponding landmark has been observed in one of the oldest four keyframes. This means, the oldest four keyframes do not provide any feature constraints. This reduces the number of freely adjustable parameters in the optimization. In addition, only features that have been observed in at least six keyframes in the window are marginalized out. Finally, in order to have a save margin in the optimization time, the number of tracked features was reduced to 120.

The proposed adaption were then extensively tested on all eleven sequences of the EuRoC dataset. The results show that our approach is less accurate than the default implementation of VINS-Mono, as expected, but is very consistent over the three tested platforms. This includes two computational limited single board computers and a much stronger Laptop. The similar results proof the robustness of our algorithm. Furthermore, the test shows that even the outliers in the per-frame optimization time do not exceed the 100 millisecond, ensuring real-time applicability. The RMSE errors show that our approach is most suitable for slower movements. Whereas with faster movements, the reduced number of features in the optimization becomes noticeable. This suggests, that as one might expect, there is no free lunch in visual state estimation.

We also tested the proposed VIO pipeline with a MSF framework on a UAV, which succeeded to hover. However, there was noticeable incertitude in the movement. Due to time limits, this effects could not be further investigated. However we assume, that most of this incertitude can be eliminated by properly tuning the MSF parameters and performing a more accurate calibration.

We want to note that in our first approach we only ignored the feature constraints in the oldest four keyframes of the sliding window, as described in section 4.6. Which

is equivalent to perform only a pose graph optimization for the oldest keyframes. However, we had to reduce the number of tracked features down to 80 to achieve real-time performance in all of the tested sequences described in subsection 6.2.1. Otherwise, there were always a few outliers in the total per-frame optimization time above 100 milliseconds. Reducing the tracked feature that much reduced also the accuracy significantly. This led us to the approach of ignoring all feature constraints in the BA optimization if a feature has been observed in one of the oldest keyframes.

7.2 Outlook

The evaluation is done on the different summands of the BA optimization, but this does not cover all of the adjustable parameters. Some of the parameters that have not been analyzed in this work are the following:

- Minimum distance between two features.
- Keyframe selection threshold (parallax and minimum detected features).
- Solver types in Google's *ceres solver*[17].

These parameters might have a significant influence on the accuracy or the per-frame optimization time.

The ROS implementation of the default VINS-Mono and our adaption contains buffers. However, when using as real-time VIO pipeline, it would be better to drop frames if the optimization can't catch up with the feature tracker. Otherwise, the BA optimization may be performed on old data. Hence, one could create a "live" version of our VIO pipeline, which does not use buffers.

Ignoring features seen in the oldest keyframes is a quite radical approach for reducing the number of feature constraints. A more sophisticated heuristic for reducing the number of feature constraints may lead to an even better time accuracy trade off.

Bibliography

[1] H. Strasdat, J. M. M. Montiel, and A. Davison, "Scale Drift-Aware Large Scale Monocular {SLAM}," in *Robotics: Science and Systems {VI}*. Robotics: Science and Systems Foundation, jun 2010.

[2] P. Schmuck and M. Chli, "Multi-UAV collaborative monocular SLAM," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3863–3870.

[3] M. Karrer and M. Chli, "Towards Globally Consistent Visual-Inertial Collaborative SLAM," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[4] I. Alzugaray, L. Teixeira, and M. Chli, *Short-term UAV Path-Planning with Monocular-Inertial SLAM in the Loop*. ETH-Zürich, 2017.

[5] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *CoRR*, vol. abs/1708.0, pp. 1–17, 2017.

[6] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[7] J. Delmerico and D. Scaramuzza, "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[8] A. I. Mourikis and S. I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, apr 2007, pp. 3565–3572.

[9] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual – inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[10] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct EKF-based approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, sep 2015, pp. 298–304.

[11] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semi-Direct Visual Odometry for Monocular , Wide-angle, and Muti-Camera Systems," *Nccr*, pp. 1–17, 2016.

[12] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2014.

[13] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2017.

[14] F. Dellaert, "Factor Graphs and GTSAM: A Hands-on Introduction," GT RIM, Tech. Rep. GT-RIM-CP&R-2012-002, sep 2012.

[15] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[16] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding window filter with application to planetary landing," *J. Field Robotics*, vol. 27, pp. 587–608, 2010.

[17] S. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

[18] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, apr 1991.

[19] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, "A synchronized visual-inertial sensor system with FPGA preprocessing for accurate real-time SLAM," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 431–437.

[20] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, nov 2013, pp. 1280–1286.

[21] S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation," in *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2013.