CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS

Jan Bednář

Self-Localization of Unmanned Aerial Vehicles Using Visual Inertial Odometry

Department of Cybernetics

Project supervisor: **Ing. Matěj Petrlík**

January 2020

Declaration of authorship

I hereby declare that I wrote the presented thesis on my own and that I cited all the used information sources in compliance with the Methodical instructions about the ethical principles for writing an academic thesis.

Prague, date............................ ....

signature

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Bednář Jan**	Personal ID number:	**420192**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Control Engineering**		
Study program:	**Cybernetics and Robotics**		
Branch of study:	**Cybernetics and Robotics**		

II. Master’s thesis details

Master’s thesis title in English:

Self-Localization of Unmanned Aerial Vehicles Using Visual Inertial Odometry

Master’s thesis title in Czech:

Sebelokalizace bezpilotní helikoptéry pomocí vizuálně-inerciální odometrie

Guidelines:

The focus of this thesis is to analyze the performance of available state-of-the-art visual-inertial odometry (VIO) [1] algorithms and to evaluate the suitability for integration into the control system of unmanned aerial vehicles (UAVs). A trajectory-shaping filter that imposes constraints on the motion of the UAV based on the properties of the chosen VIO algorithm should be designed. The following tasks will be solved:
• Perform a survey of available VIO algorithms, especially SVO [2] and MSCKF [3, 4].
• Become familiar with stereo cameras, particularly Intel RealSense cameras, and their integration into the ROS middleware, Gazebo simulator and MRS group UAV platform.
• Prepare a dataset from a real-world UAV flight with precise GPS, and compare the performance of the algorithms on this dataset.
• Integrate the algorithms into the Gazebo simulator and examine how the mounting position of the camera on the UAV affects the performance of the algorithms.
• Prepare the algorithms for integration into the position feedback control loop of the UAV and test its feasibility in the Gazebo simulator.
• Design and implement a trajectory-shaping filter that improves the localization precision of the algorithms.

Bibliography / sources:

[1] J. Delmerico and D. Scaramuzza, "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots," 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, 2018, pp. 2502-2509. doi: 10.1109/ICRA.2018.8460664
[2] C. Forster, M. Pizzoli, and D. Scaramuzza, “SVO: Fast semi-direct monocular visual odometry,” in IEEE International Conference on Robotics and Automation (ICRA), 2014.
[3] 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, April 2007, pp. 3565–3572.
[4] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, “Robust stereo visual inertial odometry for fast autonomous flight,” CoRR, vol. abs/1712.00036, 2017. [Online]. Available: http://arxiv.org/abs/1712.00036.

Name and workplace of master’s thesis supervisor:

Ing. Matěj Petrlík, Multi-robot Systems, FEE

Name and workplace of second master’s thesis supervisor or consultant:

Ing. Martin Saska, Dr. rer. nat., Multi-robot Systems, FEE

Date of master’s thesis assignment: **13.09.2019** Deadline for master's thesis submission: **07.01.2020**

Assignment valid until:
by the end of summer semester 2020/2021

Ing. Matěj Petrlík Supervisor’s signature	prof. Ing. Michael Šebek, DrSc. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Acknowledgements

I would like to thank my thesis adviser, Ing. Matěj Petrlík, for his outstanding leadership, endless patience, and the immense amount of helpful advice that helped me with this thesis. Next, I would like to thank all members of the MRS group who backed me up when necessary.

I would like to thank also to my family for astonishing support during the whole study period. Without their support, it would not be possible to manage that.

Finally, I would like to express my most excellent thanks to my girlfriend Dominika, who always stood behind my back and supported me all the time.

Abstract

This thesis is concerned with visual-inertial odometry (VIO) algorithms and their usability and suitability for integration into the feedback loop of the unmanned aerial vehicle (UAV) control system. The main part of the thesis is focused on the comparison of chosen VIO algorithms in terms of pose estimation precision for chosen camera mounting orientations, camera frame rate, UAV velocity and the feedback suitability. According to prior survey of VIO algorithms precision, availability and fitness on UAV deployment, three VIO algorithms are chosen for this thesis, namely S-MSCKF, SVO, and VINS-Fusion. The VIO algorithms performance is evaluated in both the simulation environment and the real environment to prove the suitability for feedback loop integration. The trajectory-shaping filter was implemented to smooth the trajectory by constraining the accelerations according to the UAV dynamics. Such filter improves not only the precision of VIO pose estimation but also the similarity of the control reference generated from the tracked trajectory. Lastly, the feedback integration for tested algorithms is presented for all used VIO algorithms in the simulator and partially in the real deployment.

Keywords: unmanned aerial vehicle, visual-inertial odometry, inertial measurement unit, cameras, trajectory shaping, pose estimation, camera calibration, camera orientation, feedback control

Abstrakt

Tato práce se zabývá vhodností použití algoritmů vizuálně-inerciální odometrie (VIO) pro integraci do zpětnovazebního řízení bezpilotního letounu (UAV). Hlavní část této práce je zaměřena na porovnání vybraných VIO algoritmů z hlediska přesnosti estimace polohy UAV v závislosti na použité orientaci kamery, snímkové frekvenci použité kamery, rychlosti UAV a vhodnosti použití ve zpětnovazebním řízení. Dle prvotního průzkumu na základě přesnosti, dostupnosti a vhodnosti VIO algoritmů pro použití na UAV byly vybrány tři algoritmy, a to S-MSCKF, SVO a VINS-Fusion. Výkonnost VIO algoritmů pro zpětnovazební řízení je testována jak v simulačním, tak v reálném prostředí. Dále je implementováno vyhlazování trajektorie, které vyhladí trajektorii použitím akceleračních omezení dynamiky UAV. Takový filtr zlepšuje nejen přesnost odhadu pozice z VIO, ale také podobnost řídící reference vygenerované ze sledované trajektorie. V poslední části jsou použité VIO algoritmy testovány ve zpětnovazebním řízení UAV jak v simulačním, tak v reálném prostředí.

Klíčová slova: bezpilotní letoun, vizuálně-inerciální odometrie, inerciální měřící jednotka, kamery, vyhlazování trajektorie, estimace pozice, kalibrace kamery, orientace kamery, řízení ve zpětné vazbě

Contents

List of Figures **v**

List of Tables **vii**

1 Introduction **1**

1.0.1 Related work . . . 2
1.0.2 Preliminaries . . . 4
1.0.3 Contribution . . . 8
1.0.4 Outline . . . 9

2 Hardware setup **11**

2.1 Intel RealSense cameras . . . 11
2.1.1 Depth camera - D435i . . . 11
2.1.2 Tracking camera - T265 . . . 12
2.1.3 IMU . . . 13
2.1.4 Camera installation . . . 13
2.2 Calibration tools . . . 14
2.2.1 Kalibr . . . 14
2.2.2 Intel RealSense IMU Calibration tool . . . 17
2.2.3 IMU Noise Model . . . 17
2.3 UAV platform . . . 17

3 Vision Algorithms **19**

3.1 Stereo Multi-State Constraint Kalman Filter (S-MSCKF) . . . 19
3.1.1 Implementation . . . 22
3.2 Semi-Direct Visual Odometry (SVO) . . . 23
3.2.1 Implementation . . . 27
3.3 VINS-Mono . . . 28
3.3.1 Implementation . . . 36

4 Practical evaluation **39**

4.1 Evaluation metrics . . . 39

4.2 Trajectory planning . . . 41

4.3 Trajectory shaping . . . 44

4.4 Gazebo simulator . . . 46

4.4.1 S-MSCKF . . . 47

4.4.2 SVO . . . 48

4.4.3 VINS-Fusion . . . 48

4.5 Algorithm compatibility with MRS system . . . 48

4.6 Testing scenarios . . . 49

4.7 Simulator experiments . . . 50

4.7.1 Camera orientation test . . . 51

4.7.2 High altitude test . . . 52

4.7.3 High FPS rate . . . 54

4.7.4 Feedback control . . . 55

4.8 Real experiments . . . 58

4.8.1 Dataset acquisition . . . 58

4.8.2 Dataset algorithms comparison . . . 60

4.8.3 Feedback control with the real UAV . . . 63

5 Conclusion **67**

5.0.1 Future work . . . 68

Bibliography **69**

Appendices **75**

.1 List of abbreviations . . . 77

List of Figures

1.1 Examples of various UAV applications 2
1.2 Coordinate frames system of the UAV 6
1.3 Control pipeline diagram of the UAV 7

2.1 Intel RealSense D435i camera 12
2.2 Intel RealSense T265 camera 13
2.3 The Kalibr calibration output figure of the reprojection error 16
2.4 The UAV platform 18

3.1 Comparison of S-MSCKF precision and performance 21
3.2 SVO frames notation 24
3.3 SVO feature alignment strategies 25
3.4 Depth estimation for SVO 26
3.5 SVO interpretation of stereo camera system 26
3.6 VINS-Mono visual-inertial alignment 29
3.7 Marginalization process for VINS-Mono 31
3.8 Relocalization and pose graph optimization for VINS-Mono 32
3.9 Illustration of the pose graph for VINS-Mono 33
3.10 Indoor experiment results for VINS-Mono 34
3.11 Illustration of temporal misalignment 35
3.12 Temporal calibration influence on VINS-Mono 36

4.1 Example of ATE 40
4.2 Example of RPE 41

4.3 Trajectory samples for constant velocity assumption 42
4.4 Velocity and acceleration profile for constant velocity assumption 43
4.5 Example application of acceleration constraints on the trajectory 44
4.6 Variable velocity trajectory example 45
4.7 Simulation camera attachment on the UAV 47
4.8 Example of a successful trial of the camera orientation test for S-MSCKF . . 53
4.9 Figure of failed trial of camera orientation test for VINS-Fusion 53
4.10 Example of precise pose estimation of VINS-Fusion algorithm in feedback control in the simulation environment 57
4.11 Example of drifting pose estimation of SVO algorithm in feedback control in the simulation environment 58
4.12 Real camera attachment on the UAV 59
4.13 Pattern projection onto D435i camera 62
4.14 S-MSCKF feature detection on corrupted image 62
4.15 VINS-Fusion feature detection on corrupted image 62
4.16 Feedback loop control with S-MSCKF algorithm on the real UAV 63
4.17 Feedback loop control with SVO algorithm on the real UAV 64
4.18 S-MSCKF feedback loop test with fixed UAV orientation 65

List of Tables

2.1 List of supported resolution/frame rate combinations for RealSense D435i infra module 12

4.1 Camera orientation test in simulator environment 51
4.2 High altitude test in the simulation environment 54
4.3 High FPS rate test in the simulation environment 55
4.4 Results of feedback control test in the simulation environment 56
4.5 Results of feedback control test in longer duration test in the simulation environment 57
4.6 Algorithm comparison on real datasets 61

1 CD Content 77
2 Lists of abbreviations 78

Chapter 1

Introduction

The recent growth in the availability of various unmanned aerial vehicles (UAVs) for both personal usage and commercial applications increased general interest in these devices. Thanks to the rising utility of UAV, even more industrial applications [1] are planned for the future. Investments into the development of more computationally powerful devices, cheaper and more precise sensors as well as investments into the research are becoming more appealing. Some of the current research includes e.g., optimal trajectory tracking for UAVs using model predictive controller (MPC) together with non-linear state feedback [2], onboard relative localization method based on ultraviolet light (UVDAR), used for realtime control of a leader-follower formation of UAVs [3] and data collection planning approach based on Dubins Orienteering Problem with Neighborhoods (DOPN) to determine the sequence of visits to the most rewarding subsets of target locations [4]. Finally, the development in these fields allowed to accomplish ambitious scenarios such as agile autonomous landing on a car moving at $15\,\mathrm{km\,h^{-1}}$ [5], autonomous cooperative localization, grasping and delivering of colored ferrous objects [6, 7], autonomous realtime monitoring of large areas with Micro Aerial Vehicles (MAVs) with minimal sensory and computational resources [8]. An autonomous Search And Rescue (SAR) operation within a constrained and hostile environment [9] is an example of a more practical UAV deployment based on previous research. The UAVs also helps with condition monitoring of interiors of historical buildings[1] [10] such as churches, castles, etc. Some of previously mentioned applications are shown in Figure 1.1.

All of these scenarios are a step closer to include the UAVs in wide-spread applications. A research project [11] includes the UAVs as effective monitoring, diagnostic, and reporting unit in the city infrastructure. The UAVs can improve the safety and efficiency of building sites, where they can be deployed in various operations, e.g., pre-investigation processes of the suitable future site, continuous monitoring of individual operations, and precise measurements during site surveys. Also, the UAVs are well suited to handle utility infrastructure inspection in the road tunnels, where otherwise, a human worker has to perform the task, which can be risky in case of failure or accident.

[1]`https://dronument.cz/`

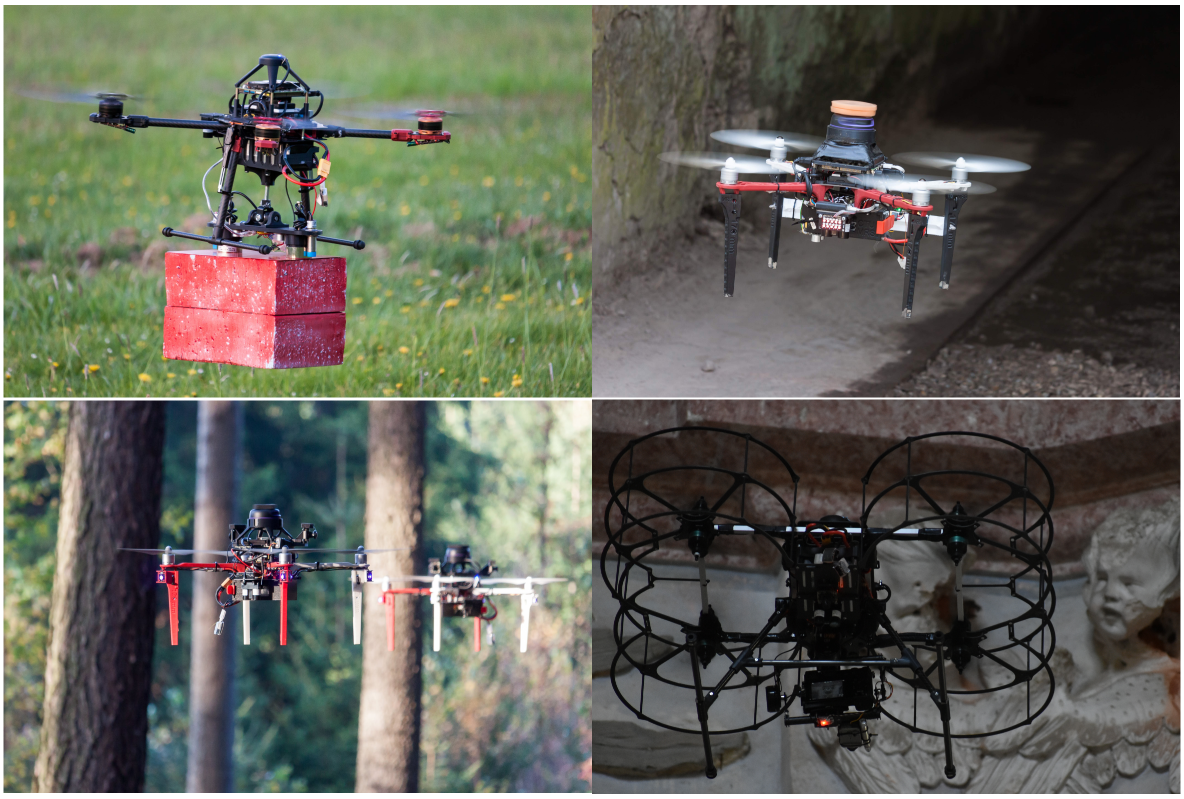


Figure 1.1: Examples of various UAV applications. The photo at the top left corner shows the magnetic brick holder attached to the UAV - scenario for MBZIRC 2020. At the top right corner is a UAV during the DARPA challenge in August 2019 [9]. At the bottom left corner shows the swarm of two UAVs in the forest during experiments with UVDAR [3]. In the bottom right corner is the UAV during church inspection [10].

1.0.1 Related work

UAV localization is an essential aspect of possible autonomous behavior making. The autonomy allows to perform the designed task accurately, which can involve obstacle avoiding, object transportation, or SAR scenarios. In most cases, the Global Navigation Satellite System (GNSS) is used for localization due to its full availability and easy-to-use approach. However, the GNSS system is providing a stable and reliable position only in the outdoor environment, where no high obstacles cover the sky around the UAV. The GNSS system can reach the best precision within meters, which might be not sufficient for precise localization that is required by some applications. The GNSS outdoor-only availability limits the usage of the GNSS in all kinds of applications that can cover the mentioned tasks as SAR, infrastructure inspection, or any indoor scenario. The precision of GNSS can be improved with the usage of Differential GNSS (DGNSS) or Real-Time Kinematic (RTK), which gives the centimeter precision. However, these solutions require specific equipment i.e., a base station that achieves accuracy improvement of the GNSS system by sending position corrections to the UAV. It is quite impractical for commercial deployment. Hence such an option is mostly useful only in experimental UAV deployments.

Barring GNSS, there are other forms of possible onboard sensors gathering knowledge of UAV surroundings suitable for localization. One of the essential motivations is the independence on GNSS, since, apart from being imprecise and unavailable, GNSS limits the UAV deployment to large open areas with direct satellite visibility. One example of different localization methods is the LiDAR-based localization. LiDAR technology is a laser-based time-of-flight method for measuring the distance from obstacles. The advantage of LiDAR is the independence on weather conditions such as bright sunlight. Next, LiDAR can also be used during the night or under unfavorable lighting conditions. Moreover, LiDAR can obtain the data even from textrureless objects. However, LiDAR technology is dependent on the reflectivity of the surface. Also, the wide-open areas can be problematic because data scans might seem to be similar for a different position, hence resulting position estimation might not be accurate. Another issue for using LiDAR is the price (especially for multi-planes version) and sensor weight, which is crucial for the UAV. An example of LiDAR-based localization is the Hector SLAM [12] for single-plane LiDARs and [13] for multiple-plane LiDARs. Another type is vision-based localization methods. The vision-based methods use a camera unit to obtain information from the surroundings. Cameras have the main disadvantage in strong dependence on weather conditions, especially the lighting conditions. For sudden light changes, the camera must react with exposure adjustment, which might be a limitation for some applications. However, cameras have several significant advantages over other methods. First, cameras are a much cheaper alternative to GNSS than LiDAR, which is crucial for practical applications. Cameras are also a lightweight and low-cost way to localize the UAV, contrary to more massive and much more expensive LiDAR sensors. These are the reasons why the thesis is focused only on visual-based methods.

Precise visual state and motion estimation for the UAV has been under intensive development in recent years. Various approaches are combining a monocular or stereo camera and an IMU (Inertial Measurement Unit) in different ways. Formerly, a single camera was used to estimate the state, especially the monocular variant due to its low price, smaller size, and easy setup. The camera state (motion) estimation is referred to as Visual Odometry (VO). However, the monocular setup cannot correctly recover the metric scale and feature depth. This limitation can be partially balanced using a low-cost IMU. It gives the system the ability to retrieve the metric scale as well as roll and pitch angles. Additionally, the IMU can effectively substitute the camera system in hostile visual conditions such as motion blur, textureless area, and rapid illumination changes. For a short time, the IMU can be fused either as loosely coupled where the camera and IMU data are treated separately, i.e., the IMU assists in the visual structure or as tightly coupled where IMU data are precalculated and then used to improve the vision part. The camera-IMU setup refers to Visual-Inertial Odometry (VIO). Nowadays, onboard CPUs become much faster and able to handle the more demanding stereo camera systems. An immense advantage of the stereo system is the precise capability of feature depth estimation thanks to the camera baseline increasing the pose estimation accuracy in various environments and support rapid motion and rotation. Together with the IMU, it delivers robust state estimation. Further on, only stereo-camera supported algorithms are assumed.

VIO algorithms can be divided into optimization-based and filter-based approaches. The filter-based methods generally rely on an Extended Kalman Filter (EKF), where the IMU is used for the state propagation, and the vision poses are used for the update step. Optimization-based methods rely on jointly optimizing the residuals of image and IMU data measurements to obtain the state estimates. However, to ensure the consistent processing time per frame, the bounded-size sliding window has to be implemented along with marginalization to reduce the number of optimization states. Further on, the image extraction part can be divided into feature-based and direct methods. Feature-based methods extract the features in the image and track them in the following images by minimizing the reprojection error. Direct methods are working directly on pixel intensity to minimize the photometric error. Most of the computational time for the feature-based methods is used on feature extraction, which causes a high constant computational cost per frame. In contrast, direct methods require a good initial guess. Feature-based methods are often used in real-world engineering deployment due to robustness, while direct methods can be easily extended to dense mapping.

Currently, there are several benchmark studies for visual state estimation. A recent benchmark comparison regarding monocular VIOs for UAVs is [14]. The authors take into account many state-of-the-art VIOs and compare them in terms of computational requirements as well as in terms of translation errors and absolute translation error on the EuRoC dataset [15]. Next, there are other visual odometry comparisons as [16, 17], but it is related only to non-inertial methods or non-6DoF variants [18]. However, all of the localization methods are tested only on publicly available benchmarks, which are usually based on custom sensor set. Because of that, several VIO algorithms are chosen and compared regarding camera orientation in this thesis. Furthermore, the mentioned benchmarks are assuming a specific camera (camera-IMU) sets which are not off-the-shelf. As mentioned in [14], there is a plentiful variety of VIO implementations suitable for a stereo setup. However, not all of them are publicly available, as [19, 20, 21]. The selection process from the public ones follows. An example of filter-based VIOs is S-MSCKF algorithm [22] or ROVIO algorithm [23, 24]. Both algorithms are based on EKF. However, the ROVIO algorithm for the stereo-camera setup is not recommended due to implementation issues. Hence it will not be compared in this thesis. For optimization-based approaches, there are several variants as SVO [25], VINS-Fusion [26] or OKVIS [27]. VINS-Fusion algorithm compares the position estimation w.r.t. OKVIS algorithm. As seemed from results published in [26], VINS-Fusion pose estimation is more precise than OKVIS. Both VINS-Fusion and OKVIS are optimization-based algorithms with generally similar approaches. Hence only VINS-Fusion is assumed in this thesis together with different optimization approach in SVO.

1.0.2 Preliminaries

Before continuing with the detailed description of the vision algorithms, it is fitting to introduce the components of the system used for the evaluation of the algorithms. Notably, the basics of the Robot Operating System (ROS) system [28] are briefly described as well as the used MRS system, especially the MPC tracker [29] and SO3 controller [30].

Robot Operating System (ROS)

ROS is an open-source middleware defining and providing a structured communication layer above the host operating system. The authors have defined the goals of the ROS as follows:

- **Peer-to-peer communication** ensures connectivity between ROS processes and the number of different hosts to the peer-to-peer topology in realtime.

- **Tools-based** approach allows creating a microkernel module with a large number of small tools to run various ROS components. These tools allow to perform various tasks with the ROS, e.g., navigate through source code, visualize the topology, measure bandwidth utilization, and so on.
- **Multi-lingual** form offers to code modules in ROS with different programming languages (C++, Python,..). Because the ROS specification is at the communication layer, modules can be coded in different languages and work together. To further improve the cross-platform independence, the interface definition language (IDL) is introduced to define the messages between modules.
- **Thin** property proposes to place the libraries in separate packages supporting the modularity. The libraries are not dependent on the ROS, and hence they can be reused in other projects out of the ROS. Contrarily, other open source projects libraries can be reused in ROS as Player project [31] or OpenCV [32].
- **Free and Open-Source**. ROS is publicly available[2] and distributed under BSD license.

This thesis is using ROS version Melodic[3] installed on Ubuntu 18.04[4] operating system. The basic ROS glossary used in the thesis is enumerated.

- Node - A computation process in ROS corresponding to a software module. Nodes are communicating with other nodes using messages and services.
- Message - A data structure with a fixed format for communication between nodes. Each message is published on a topic identified by a string. The main messages used in this thesis are *Odometry*[5], *PoseWithCovariance*[6], *IMU*[7] and *Image*[8].
- Topic - Topic is an interface between the nodes and the ROS network. A node can either subscribe or publish to the topic. A single node can publish or subscribe to multiple topics. Furthermore, several nodes can be publishing or subscribing to a single topic.
- Service - A service is a request/reply type of communication between the nodes. The service is used for synchronous communication in contrary to the aforementioned topic subscribe-publish scheme.
- Rosbag - A ROS file format for storing the ROS messages. It allows to record messages during the experiment for later playback.

Another attractive feature of ROS is the handling of transformations between coordinate frames. Usually, the robotic unit consists of several different sensors with their own coordinate frames, which are related by a translation and rotation. In the thesis, the frames are the

[2]`http://wiki.ros.org`
[3]`http://wiki.ros.org/melodic`
[4]`http://releases.ubuntu.com/bionic/`
[5]`http://docs.ros.org/melodic/api/nav_msgs/html/msg/Odometry.html`
[6]`http://docs.ros.org/melodic/api/geometry_msgs/html/msg/PoseWithCovariance.html`
[7]`http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Imu.html`
[8]`http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Image.html`

world frame, the UAV frame, the IMU frame, and the camera frame. The world frame is the frame where the UAV is flying in. The world frame is essential for the control mechanism to determine the UAV position, eventually orientation change. In the simulator, the UAV starts at the origin of the world frame. In real experiments, the origin of the world frame can be set appropriately to GPS coordinates if GPS is available. The UAV frame has its origin in the center of the flight control unit (FCU) of the UAV. The IMU frame is either the same as the UAV frame in case of simulation or the same as the camera frame in the real deployment. Each of them has its own coordinate system oriented differently. The *tf* ROS package constructs a dynamic transformation tree that relates the frames and offers a convenient way to seamlessly transform data between these frames, including interpolation between available transform messages.

UAV state

The state vector of the UAV is defined as

$$\mathbf{x} = (\mathbf{r}, \dot{\mathbf{r}}, \ddot{\mathbf{r}}, \mathbf{R}, \dot{\mathbf{R}}), \tag{1.1}$$

where $\mathbf{r} = [x, y, z]^T$ is the UAV position and $\mathbf{R}(\phi, \theta, \psi)$ is the orientation in the world frame w. The state variables linear acceleration $\ddot{\mathbf{r}}$, orientation $\mathbf{R}$ and angular rate $\dot{\mathbf{R}}$ are obtained directly from the Flight Control Unit IMU. The coordinate frames notation is shown in Figure 1.2. The $\mathbf{r}$ and $\dot{\mathbf{r}}$ has to be estimated, more about that in Chapter 1.0.2.

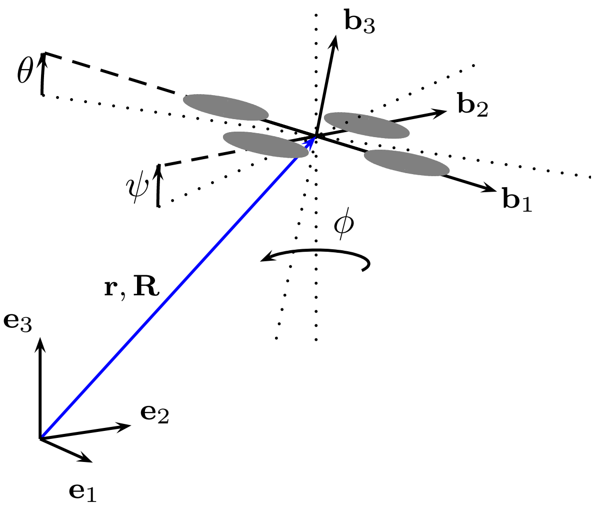


Figure 1.2: The translation $\mathbf{r} = [x, y, z]^T$ and orientation $\mathbf{R}(\phi, \theta, \psi)$ of the UAV expresses the relation between the world frame $w = \{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ and the body frame $b = \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3\}$. Courtesy of [9]

Control pipeline

The MRS system is composed of a control pipeline with state estimation shown in Figure 1.3. The system description from [29] is as follows. The pipeline for evaluation of algorithm precision consists of several parts. The first part is the mission planner, which is mostly the square follower in this thesis. The mission planner defines the required position $\mathbf{r}_D$ and orientation ϕ_D of the UAV which is handed over to the MPC tracker [29] to obtain the command $\mathbf{x}_D$, $\dot{\mathbf{x}}_D$, $\ddot{\mathbf{x}}_D$,ϕ_D,$\dot{\phi}_D$ and $\ddot{x}_D$ for the SO(3) controller. The MPC tracker repeatedly solves the optimization problem

$$\min_{\mathbf{u}_t,\mathbf{v}_t} V(\mathbf{x},\mathbf{u}) = \frac{1}{2}\sum_{i=1}^{m-1}(\mathbf{e}_i^T\mathbf{Q}\mathbf{e}_i + \mathbf{u}_i^T\mathbf{P}\mathbf{u}_i), \tag{1.2}$$

$$\mathbf{x}_{t+1} = \mathbf{A}\mathbf{x}_t + \mathbf{B}\mathbf{u}_t, \qquad \forall t \in \{0,\dots,m-1\} \tag{1.3}$$

$$\mathbf{x}_t \le \mathbf{x\_max}_t, \qquad \forall t \in \{1,\dots,m\} \tag{1.4}$$

$$\mathbf{x}_t \ge \mathbf{x\_min}_t, \qquad \forall t \in \{1,\dots,m\}, \tag{1.5}$$

where $\mathbf{x}$ is the UAV state vector, the $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ is the control error where $\hat{\mathbf{x}}$ is the trajectory reference and $\mathbf{u}$ is the action input. The quadratic cost function from Equation (1.2) penalizes the control error $\mathbf{e}$ and action input $\mathbf{u}$ over a horizon of $m \in \mathbb{Z}^+$ in length. Penalization matrices $\mathbf{Q}$ and $\mathbf{P}$ are positive semi-definite. Constraint in Equation (1.3) forces the states to follow the LTI model. Constraints in Equation (1.4) and 1.5 bound the states to a box to limit the maximal acceleration and velocity.

The output of the MPC tracker creates the command values for the SO(3) controller [30]. The controller produces the reference thrust and orientation reference for the embedded attitude controller. The attitude controller commands the Electronic Speed Controllers (ESC) of the UAV directly.

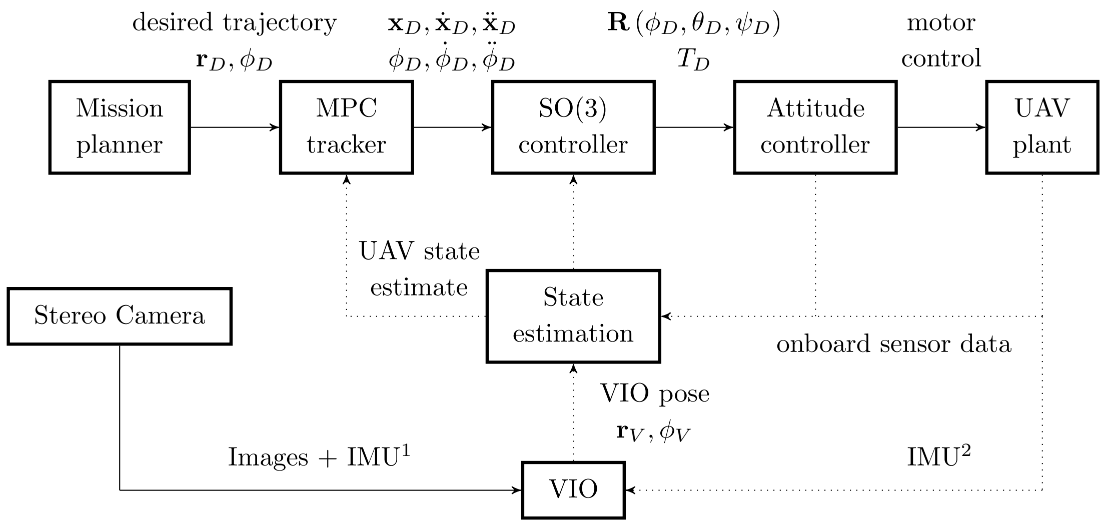


Figure 1.3: Diagram of the control pipeline. The reference trajectory $\mathbf{r}_D, \phi_D$ serves as a setpoint for the MPC in the MPC tracker, which outputs a command $\mathbf{x}_D, \dot{\mathbf{x}}_D, \ddot{\mathbf{x}}_D$, $\phi_D, \dot{\phi}_D, \ddot{\phi}_D$ for the non-linear $SO(3)$ controller. The non-linear controller produces the orientation and thrust reference for the embedded attitude controller. (1) symbolizes the variant, when the IMU data comes from the same source as images, e.g. RealSense D435i camera. (2) symbolizes the variant, when the IMU data are from the attitude controller (built-in IMU), e.g. Gazebo simulator.

Pose estimation

Pose estimation description is taken from [9]. The components of the UAV position $\mathbf{r}$ are independent, hence they can be processed separately. The z-component is obtained from rangefinder-barometer fusion in a linear Kalman filter scheme. The filters for each x and y are identical. Thus only x-axis is described. The state vector for x-axis is defined as

$$\mathbf{x} = (x, \dot{x}, \ddot{x}, \ddot{x}^{(u)}, \ddot{x}^{(d)}, \theta), \tag{1.6}$$

where $\ddot{x}^{(u)}$ is the acceleration resulting from the control input, $\ddot{x}^{(d)}$ is the acceleration from the external disturbing forces (wind, propellers,...) and θ is the rotation along y-axis of the world frame w.

Integration realized by the model obtains the velocity and the position, and as such, they are prone to drift, so a position or velocity correction is necessary to stabilize the system. The state of the UAV is estimated from the most suitable localization source, thanks to a bank of Kalman filters for every localization method. A possible localization methods currently available at the MRS system are GPS, GPS RTK, OpticFlow [33] and HectorSLAM [9]. Propeller-induced vibrations degrade direct measurements of acceleration, so it is not used to correct the acceleration state variable. In this thesis, the output of the VIO algorithm is used as a position correction for the UAV model.

1.0.3 Contribution

In this thesis, the VIO algorithms have been integrated into the feedback loop of the UAV control system. Three algorithm have been chosen, according to the prior survey described in Chapter 1.0.1. Firstly, the filter-based S-MSCKF algorithm [22] designed explicitly for UAVs. Hence it covers fast motion, lower computational power, and sudden changes. Next, the semi-direct VO algorithm called SVO [25] is introduced, which combines feature-based and direct methods with robust probabilistic depth estimation, including motion priors. Lastly, the optimization-based VINS-Fusion algorithm is presented, supporting the fusion of different sensor suites based on VINS-Mono VIO. The work compares the algorithms on off-the-shelf camera-IMU sensor sets, which are widely used in robotics.

The main contributions should be noted. Most of the VIO implementations are compared against results on available public datasets. However, the VIO authors focus on achieving the best result on the given dataset, while in practice, the precision might be much different in other conditions. This thesis focuses more on the possible camera orientations during the flight, which are crucial, especially for outdoor deployment due to changing lighting conditions. Next, the UAV trajectory smoothing is presented to improve UAV precision further. The trajectory smoothing takes the originally planned trajectory and adjusts it to fulfill the UAV acceleration constraints.

Lastly, the algorithms are deployed to the feedback loop of the control system of a real UAV. Usually, the authors are proposing a new algorithm that is verified and compared against other algorithms only on the available datasets or in the specific hand-carried scenarios. Theoretically, the simulation experiments should be sufficient to prove the functionality. However, the real deployment usually faces more complicated events that cannot be simply modeled in the simulation. As this thesis shows, the simulation environment can give much better estimations that cannot be equally achieved in a real deployment.

1.0.4 Outline

The rest of the thesis is outlined as follows. Firstly, the used hardware is described in Chapter 2, including the Intel RealSense cameras, calibration process, and UAV platform Next, the algorithms implementation details are described in Chapter 3 along with required adjustments of algorithms for usage in this thesis. Lastly, the details regarding pre-test requirements as metric evaluation definition, testing scenarios explanation, and finally, simulation and real experiments together with the results are described in Chapter 4. The content of the CD and list of abbreviations are situated in Appendix.

Chapter 2

Hardware setup

In this chapter, the required steps for setting up the camera and UAV platform for the real experiments are covered. The used calibration tools are described along with how to proceed during the calibration.

2.1 Intel RealSense cameras

The continuous support along with easy setup, small form factor and reasonable price were the reasons for choosing Intel RealSense cameras[1] for the experimental evaluation of the VIO algorithms in this thesis. Intel supports the development of ROS-based Wrapper[2] for Intel RealSense cameras which facilitates integration into the existing system of MRS group. Nowadays, Intel offers depth and tracking cameras.

2.1.1 Depth camera - D435i

Intel RealSense D435i[3], shown in Figure 2.1, was chosen from the available depth camera sensors. D435i has an RGB sensor with $69.4°$ x $42.5°$ x $77°$ ($+/-$ $3°$) (Horizontal x Vertical x Diagonal) Field of View (FOV), 1920x1080 maximal resolution and maximal 30 FPS rate for given resolution. The camera resolution is expressed in pixels throughout the thesis. Next, the D435i camera consists of infrared stereo camera pair with 50 mm baseline and infrared projector to assist with depth estimation in low light and textureless environments. Further on, only an infrared camera pair is considered when the D435i camera is mentioned. The camera also contains an IMU unit. The specialty of this model is the precise time synchronization of IMU and camera images. The ROS Wrapper launch files allow setting the base camera parameters such as resolution and FPS rate. However, more advanced parameters as exposure time control, color corrections, etc. are set via configuration file. The D435i camera firmware

[1] https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html

[2] https://github.com/IntelRealSense/realsense-ros

[3] https://www.intelrealsense.com/depth-camera-d435i/

Figure 2.1: Intel RealSense D435i camera

was updated to version 05.11.06.250[4]. The list of all supported combinations of resolutions and frame rates for the infra camera module is in Table 2.1. All possible combinations are available with command *rs-enumerate-devices* with plugged-in device.

Camera Resolution [px]	**6 Hz**	**15 Hz**	**25 Hz**	**30 Hz**	**60 Hz**	**90 Hz**
1280x800		✓	✓	✓		
1280x720	✓	✓		✓		
848x480	✓	✓		✓	✓	✓
640x480	✓	✓		✓	✓	✓
640x400		✓	✓			
640x360	✓	✓		✓	✓	✓
480x270	✓	✓		✓	✓	✓
424x240	✓	✓		✓	✓	✓

Table 2.1: List of supported resolution/frame rate combinations for RealSense D435i infra module

2.1.2 Tracking camera - T265

The only available model in this category is T265 camera[5], shown in Figure 2.2. The camera has a fisheye camera stereo pair and the IMU unit available. The device has a built-in V-SLAM algorithm running directly on the device, which tracks the pose of the camera in the world to provide odometry data. The camera does not allow to change any camera nor IMU parameters due to its interconnection with V-SLAM. The camera runs with 848x480 resolution and 30 FPS frame rate. The resolution and frame rate are strictly set and cannot be changed. The built-in V-SLAM was tested during the real experiments, but the enormous drift was noticed during the UAV flight. The vibration from the UAV rotors probably results

[4]`https://downloadcenter.intel.com/download/29255/Latest-Firmware-for-Intel-RealSense-D400-Product-Family?product=128255`
[5]`https://www.intelrealsense.com/tracking-camera-t265/`

in high noise in acceleration measured by the IMU. It means that the built-in V-SLAM in its current form cannot be used for controlling the UAV without hardware modifications that would dampen the vibrations.

Figure 2.2: Intel RealSense T265 camera

2.1.3 IMU

Both D435i and T265 cameras have the Bosch BMI055[6] IMU unit. This IMU should be sufficient to track the UAV orientation and movement. However, the IMU precision is deeply influenced during the UAV flight, by high-frequency vibrations coming from the UAV propellers. These high-frequency vibrations degrade the measurements from both accelerometer and gyroscope, which in turn is detrimental to the VIO performance.

2.1.4 Camera installation

Intel provides a GitHub repository[7] which contains *librealsense* library to work with the camera on various supported platforms. Users can follow the instructions to install all necessary packages for the system. All repositories were installed and tested on the Ubuntu 18.04, kernel version: 4.18.0, *librealsense* version 2.25.0.

Another necessary software for accessing the camera from ROS nodes is ROS wrapper for Intel RealSense[8]. This ROS wrapper extends the support of *librealsense* library to the ROS platform.

The built-in camera IMU produces two separate streams published at different rates. One stream contains linear acceleration from an accelerometer, and the second consists of angular velocity from the gyroscope. That is inconvenient for localization algorithms because they require an IMU message containing both accelerometer and gyroscope data. The Intel

[6] `https://www.bosch-sensortec.com/products/motion-sensors/imus/bmi055.html`
[7] `https://github.com/IntelRealSense/librealsense`
[8] `https://github.com/IntelRealSense/realsense-ros`

RealSense ROS wrapper has the functionality to merge IMU data into one message. It offers two methods of merging. One is a unite method, which creates a new topic containing the latest angular velocity and linear acceleration data. Here comes a problem with the different sampling frequency of the sensors. The different sensor rates cause the replication of slower sensor data in the merged message. This behavior is inappropriate because it seems that one data source indicates change while the other not. Hence this method is not well usable. The second method is linear interpolation. Every new IMU message consists of the interpolated value from the previous value based on the timestamp. It ensures the values are not repeating in the published message. While using the linear interpolation option, the linear acceleration in the resting stage is exceeding the average value of standard gravity $9.81\,\mathrm{m\,s^{-2}}$, so it is also inconvenient.

A custom message merging ROS package had to be created as part of this thesis using *message_filters* package, specifically *Approximate Time* synchronization policy that uses an adaptive algorithm to match messages based on their timestamp[9]. It ensures that the values of both sensors inside IMU are used just once (with the frequency of the slower one) or not at all.

The ROS Wrapper officially does not support Ubuntu version 18.04 and ROS Melodic used by the MRS system. However, no struggle with the newer version of ROS and Ubuntu has been experienced.

Eventually, a problem with the value of angular velocity timestamp appeared. The timestamp was twice the linear acceleration timestamp value. Because of that, it was impossible to merge the messages into one with *Approximate Time* synchronization policy. To fix the issue, the *librealsense* libraries had to be compiled on the PC (instead of installing the prebuilt ones), with the specific version of Ubuntu kernel (4.15.0.47). Fortunately, this issue has been fixed recently, so now the newer Ubuntu kernel and precompiled binaries could be used.

Another problem was with the cycling timestamp of the accelerometer sensor. The accelerometer timestamps were repeating in a 4 seconds cycle after the launch. Hence it could not be merged with the gyroscope values due to the timestamp mismatch. This issue has been discovered earlier by other developers, so ROS wrapper developers fixed it in the next release.

2.2 Calibration tools

In this section, the calibration tools for the camera and IMU are described.

2.2.1 Kalibr

The ROS wrapper for RealSense camera also publishes the *CameraInfo* message, which contains the calibration parameters, but only the camera intrinsic parameters are filled out from the camera. The Kalibr toolbox[10] allows obtaining intrinsic and extrinsic parameters

[9] `http://wiki.ros.org/message_filters/ApproximateTime`
[10] `https://github.com/ethz-asl/kalibr`

of the camera system. It also calibrates the camera-IMU system. While using our own ROS package to merge the IMU sensor messages, the Kalibr calibration can detect whether there is a time shift between the camera and IMU messages. Kalibr can obtain calibration parameters for the following problems:

1. Multi-camera calibration - intrinsic and extrinsic calibration parameters of camera systems
2. Visual-inertial calibration - spatial and temporal calibration parameters of an IMU w.r.t. a camera system
3. Rolling Shutter Camera calibration - full intrinsic calibration (projection, distortion and shutter parameters) of rolling shutter cameras

Both of the cameras have a global shutter, so just the first two calibration processes are necessary. Officially, Kalibr does not support Ubuntu version 18.04 and ROS Melodic. However, it is possible to build and launch Kalibr on such a system. The tool does not require building from the source, but it is the preferable way of installation. Calibration approaches used in Kalibr are based on [34], [35] and [36].

Multi-camera calibration

The tool estimates the intrinsic and extrinsic calibrations for the multi-camera system. The neighboring cameras must have overlapping fields of view. The tool supports calibration from the ROS bag with raw images. The process goes through the images and picks images that contain the full grid of the calibration target. The tool allows combining various projection and distortion models of cameras.

The calibration process consists of the following steps:

1. Collect images - Create a ROS bag containing the raw images of a calibration target. The toolbox gives calibration targets to download and print. It is necessary to measure the important sizes of the printed target and optionally adjust the target configuration file.
2. Run the calibration - Execute the script with required parameters.
3. Output files - The calibration process outputs a PDF report with plots of calibration results for documentation. It also contains a text file summary and calibration results and transformation in the YAML format.

An example of calibration output plot for 640x360 resolution for D435i camera is shown in Figure 2.3.

The April grid 6x6 calibration target[11] was used for the calibration for both D435i and T265 cameras. The Kalibr contains a calibration validator tool to validate the calibration result.

[11] `https://github.com/ethz-asl/kalibr/wiki/downloads`

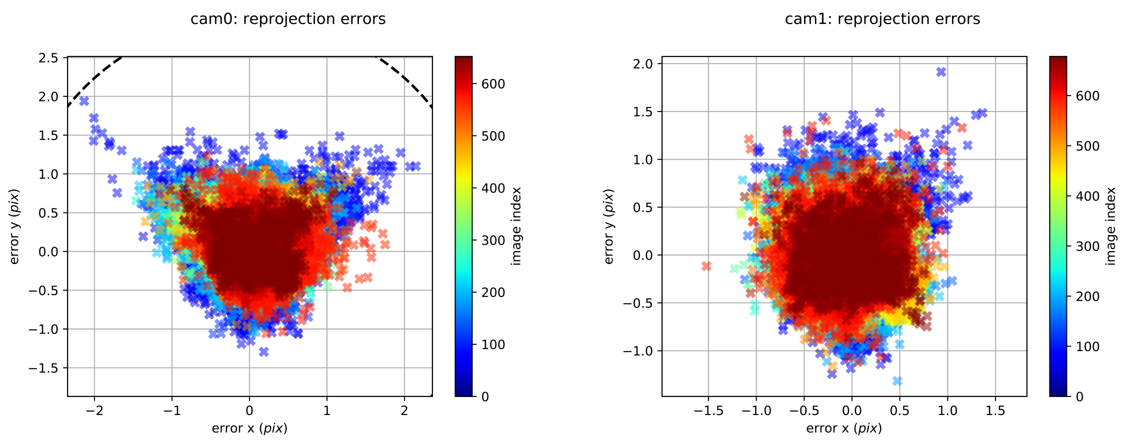


Figure 2.3: The Kalibr calibration output figure showing the reprojection error of D435i camera for 640x360 camera resolution.

Camera-IMU calibration

The Camera-IMU calibration tool estimates the spatial and temporal parameters of a camera system concerning an intrinsically calibrated IMU. Again the ROS bag must contain raw images and sensor data. Camera-IMU calibration process consists of the following steps or prerequisites:

1. Requires IMU intrinsic parameters in YAML file before the calibration. Details regarding obtaining these data are in Chapter 2.2.3.
2. Create a ROS bag containing raw images of the calibration target and IMU messages. It is important to excite all the IMU axes with proper IMU movement to have precise calibration.
3. Run the calibration with camera system parameters and IMU intrinsic parameters.
4. The results contain a PDF file with plots of calibration result, the calibration summary in a text file, and a calibration output in YAML format based on input camera system parameters. The S-MSCKF and SVO algorithms calibration files are based on the Kalibr YAML output format.

The T265 camera IMU consists of an acceleration sensor sending data at 62.5 Hz frequency and the gyroscope at 200 Hz frequency. The D453i camera has two frequency options for both accelerometer and gyroscope. The IMU accelerometer can publish data at 62.5/250 Hz and gyroscope at 200/400 Hz.

The T265 camera has only a single combination of camera resolution and IMU frequencies, hence only the D435i camera has more variants. Multiple camera resolutions have been calibrated to find a suitable resolution in further algorithm tests on the UAV. The 1280x720, 848x480, 640x480 and 640x360 camera resolutions were chosen for calibration. For the IMU, the highest frequencies available were picked, specifically 250 Hz for accelerometer and 400 Hz for the gyroscope.

Intel guarantees that the D435i camera has synchronized IMU messages with the captured images, but the temporal calibration indicates that it is not exactly right. The higher the resolution, the generally higher the time shift between camera images and IMU measurements. The 1280x720 camera resolution has time shift equal to t=0.007 s. The 640x360 camera resolution has time shift equal to t=0.003 s. These values are still small enough for 30 FPS camera rates when the time difference between individual images is equal to t=0.03 s. Also, the Kalibr result might be influenced by improper IMU movements during dataset capturing. Still, the time shift seems to be small enough for the algorithms to work well.

2.2.2 Intel RealSense IMU Calibration tool

Intel provides its own calibration tool for the IMU[12]. This tool enables to calibrate intrinsic parameters of the IMU, such as scale factor, bias, and off-axis terms.

The calibration tool requires to record a few seconds of the static position of the camera in 6 different positions to excite all axes properly. The calibration results are available during the calibration on the screen. The results can be written directly to the camera EEPROM.

2.2.3 IMU Noise Model

After the proper IMU calibration, it is necessary to obtain the IMU noise model for the appropriate setting of both algorithms. For the Gazebo simulator and real experiment purposes, the IMU noise model[13] is characterized by noise density (white noise) and random walk (bias instability). The noise density represents the fast variation of the sensor data with a zero-mean Gaussian white noise with a standard deviation σ. The higher the σ, the more noisy data are. The random walk represents the integration of the "white noise" of the noise strength σ_b.

The calibration tool[14] is calculating the gyroscope and accelerometer and bias instability parameter via analysis of the Allan variance method [37], [38]. For using the method, it is necessary to record a ROS bag containing the IMU data of at least 2 hours length. The camera (IMU) has to be in a stable position during the time of measurement. The tool uses the recorded ROS bag to calculate the desired parameters.

2.3 UAV platform

The UAV platform is based on DJI F550 hexaframe equipped with E310 DJI motors[15]. The low-level control is handled by PixHawk 4 FCU[16]. The PixHawk 4 has a built-in IMU with accelerometers, gyroscopes, and magnetometers that are used for the UAV state estimation

[12]`https://www.intelrealsense.com/wp-content/uploads/2019/07/Intel_RealSense_Depth_D435i_IMU_Calibration.pdf`

[13]`https://github.com/ethz-asl/kalibr/wiki/IMU-Noise-Model`

[14]`https://github.com/gaowenliang/imu_utils`

[15]`https://www.dji.com/cz/e310`

[16]`https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html`

and the UAV attitude stabilization. The PixHawk is extended by a GPS module simplifying the manual control of the UAV by GPS mode in the outdoor environment. The GPS module can also be used in the feedback loop of the UAV control system. The main CPU onboard is the Intel NUC Kit 8i7BEH computer[17] with i7 processor available having four physical cores and 8 threads thanks to hyperthreading option. It has a few connectivity options, namely USB 3.0 ports, WiFi, and RJ-45 connector, to communicate with other onboard or offboard units. Next, the altitude is estimated by fusion of laser rangefinder Garmin LIDAR-Lite v3[18], barometer and accelerometer from FCU. This precise altitude estimation enables to manually fly with the UAV in altitude mode, which is comfortable for the UAV pilot. Next, the UAV is equipped with a RealSense D435i camera for several camera-related scenarios. The image of the UAV platform used is shown in Figure 2.4.

Lastly, the UAV has an Real Time Kinematics (RTK) GPS receiver[19] for 1 centimeter precise position estimate. The RTK localization system consists of two RTK receivers. One receiver, which is located on a fixed-location tripod serves as the base station, while the second one is mounted on the UAV. The base station broadcasts its position along with the carrier measurements for all visible satellites. The UAV uses this information to resolve the carrier phase ambiguity and resolve the position relative to the base station. If the solution is exact, the highest possible precision of 1 cm is accomplished. It is called RTK FIX mode. If the exact solution cannot be achieved, the solution gives slightly worse precision but still better than GPS. This mode is called RTK FLOAT.

Additionally, the UAV can be equipped with a LIDAR sensor, the Bluefox camera, or different types of the gripper for specific scenarios.

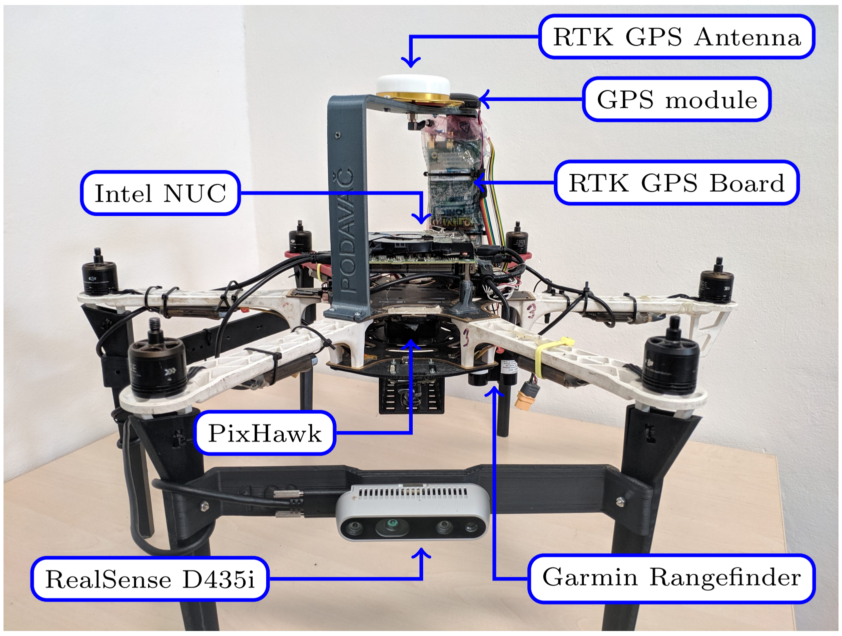


Figure 2.4: The UAV platform used in real experiments

[17]https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc8i7beh.html
[18]https://buy.garmin.com/en-US/US/p/557294
[19]https://www.tersus-gnss.com/product/bx305-oem-board

Chapter 3

Vision Algorithms

In this chapter, the analyzed VIO (VO) algorithms are described. Firstly, the filter-based S-MSCKF algorithm. Next, the semi-direct VO algorithm called SVO, which combines feature-based and direct methods with robust probabilistic depth estimation, including motion priors. Lastly, the optimization-based VINS-Fusion algorithm is described supporting the fusion of different sensor suites based on VINS-Mono VIO.

3.1 Stereo Multi-State Constraint Kalman Filter (S-MSCKF)

The S-MSCKF is a filter-based stereo VIO algorithm that uses the Multi-State Constraint Kalman Filter (MSCKF). The authors based the work on the state-of-art MSCKF [39], [40] and [41] algorithm because of its accuracy and consistency.

The notation of the world (global) frame G is denoted as ${}^{G}(\cdot)$. The authors have defined the IMU frame to be set as the body frame, which is denoted as I. However, the algorithm calibration file allows setting the transformation between the body frame and the IMU frame arbitrarily. Transformation between the global frame G and the IMU (body) frame I is ${}^{I}_{G}(\cdot)$. The estimate of the x value is denoted as $\hat{x}$.

The IMU state for the filter is defined as

$$\boldsymbol{x}_I = ({}^{I}_{G}\boldsymbol{q}^T \quad \boldsymbol{b}_g^T \quad {}^{G}\boldsymbol{v}_I^T \quad \boldsymbol{b}_a^T \quad {}^{G}\boldsymbol{p}_I^T \quad {}^{I}_{C}\boldsymbol{q}^T \quad {}^{I}\boldsymbol{p}_C^T)^T, \tag{3.1}$$

where ${}^{I}_{G}\boldsymbol{q}$ is the rotation from the global frame to the IMU frame. The velocity and position of the IMU frame in the global frame are represented as ${}^{G}\boldsymbol{v}_I$ and ${}^{G}\boldsymbol{p}_I$. Vectors $\boldsymbol{b}_g$, $\boldsymbol{b}_a$ are the biases of the measured angular velocity and linear acceleration from the IMU. The ${}^{I}_{C}\boldsymbol{q}$ represents the rotation between the camera frame and the IMU frame. The ${}^{I}\boldsymbol{p}_C$ is the position of the camera frame in the global frame. Usage of actual IMU state might cause singularities in calculated covariance matrices, due to the unit constraint on the quaternions. Hence the error IMU state is used instead as

$$\tilde{\boldsymbol{x}}_I = ({}^{I}_{G}\tilde{\boldsymbol{\theta}}^T \quad \tilde{\boldsymbol{b}}_g^T \quad {}^{G}\tilde{\boldsymbol{v}}_I^T \quad \tilde{\boldsymbol{b}}_a^T \quad {}^{G}\tilde{\boldsymbol{p}}_I^T \quad {}^{I}_{C}\tilde{\boldsymbol{\theta}}^T \quad {}^{I}\tilde{\boldsymbol{p}}_C^T)^T, \tag{3.2}$$

where the standard additive error is used for position, velocity and biases (e.g. the standard additive error in the estimate ${}^G\hat{\boldsymbol{v}}_I$ of a quantity ${}^G\boldsymbol{v}_I$ is defined as ${}^G\tilde{\boldsymbol{v}}_I = {}^G\boldsymbol{v}_I - {}^G\hat{\boldsymbol{v}}_I$), For quaternions an error quaternion $\delta\boldsymbol{q} = \boldsymbol{q} \otimes \hat{\boldsymbol{q}}^{-1}$ is used, where $\otimes$ symbolizes the quaternion multiplication. The error quaternion $\delta\boldsymbol{q}$ is related to the error state orientations as

$$\delta\boldsymbol{q} \approx (\frac{1}{2}{}^G_I\tilde{\boldsymbol{\theta}}^T \quad 1)^T, \tag{3.3}$$

where ${}^G_I\tilde{\boldsymbol{\theta}} \in \mathbb{R}^3$ describes the correspondence of the true and the estimated rotation. A total of N camera states ($i = 1..N$) are considered together in the state vector, which is

$$\begin{aligned} \tilde{\boldsymbol{x}} &= (\tilde{\boldsymbol{x}}_I^T \quad \tilde{\boldsymbol{x}}_{C_1}^T \quad ... \quad \tilde{\boldsymbol{x}}_{C_N}^T)^T, \\ \tilde{\boldsymbol{x}}_{C_i} &= ({}^{C_i}_G\tilde{\boldsymbol{\theta}}^T \quad {}^G\tilde{\boldsymbol{p}}_{C_i}^T)^T, \end{aligned} \tag{3.4}$$

where $\tilde{\boldsymbol{x}}_{C_i}$ is the camera error state and $\tilde{\boldsymbol{x}}_I^T$ is the error IMU state from Equation (3.2).

The linearized continuous dynamics for the IMU error state is:

$$\dot{\tilde{\boldsymbol{x}}}_I = \boldsymbol{F}\tilde{\boldsymbol{x}}_I + \boldsymbol{G}\boldsymbol{n}_I, \tag{3.5}$$

where $\boldsymbol{n}_I^T = (\boldsymbol{n}_g^T \; \boldsymbol{n}_{wg}^T \; \boldsymbol{n}_a^T \; \boldsymbol{n}_{wa}^T)^T$. The $\boldsymbol{n}_g$ and $\boldsymbol{n}_a$ represent the Gaussian noise of the gyroscope and accelerometer measurement. The $\boldsymbol{n}_{wg}$ and $\boldsymbol{n}_{wa}$ are the random walk rate of the gyroscope and accelerometer measurement biases. The system matrices $\boldsymbol{F}$, $\boldsymbol{G}$ are shown in [22].

The feature position in the world frame ${}^G\boldsymbol{p}_j$, where $j = 1...L$ is the j-th feature from total L features, is computed using the least square method as shown in [39] based on the current pose estimate. The residual of the measurement can be approximated as

$$\boldsymbol{r}_i^j = \boldsymbol{z}_i^j - \hat{\boldsymbol{z}}_i^j = \boldsymbol{H}_{C_i}^j\tilde{\boldsymbol{x}}_{C_i} + \boldsymbol{H}_{f_i}^j \, {}^G\tilde{\boldsymbol{p}}_j + \boldsymbol{n}_i^j, \tag{3.6}$$

where $\boldsymbol{n}_i^j$ is the measurement noise. The $\boldsymbol{z}_i^j$ is the stereo measurement of the j-th feature in the i-th camera state. The Jacobian $\boldsymbol{H}_{C_i}^j$, $\boldsymbol{H}_{f_i}^j$ and stereo measurement $\boldsymbol{z}_i^j$ are detailed in [22]. Stacking multiple observations of the same feature f_j gives

$$\boldsymbol{r}^j = \boldsymbol{z}^j - \hat{\boldsymbol{z}}^j = \boldsymbol{H}_{\boldsymbol{x}}^j\tilde{\boldsymbol{x}} + \boldsymbol{H}_f^j \, {}^G\tilde{\boldsymbol{p}}_j + \boldsymbol{n}^j. \tag{3.7}$$

The ${}^G\boldsymbol{p}_j$ is computed using camera poses, hence the uncertainty of ${}^G\boldsymbol{p}_j$ is correlated with the camera poses in the state. To ensure the uncertainty does not affect the residual and the update step of the EKF can be processed in a standard way, the residual is projected onto null space $\boldsymbol{V}$ of $\boldsymbol{H}_f^j$ hence

$$\boldsymbol{r}_o^j = \boldsymbol{V}^T\boldsymbol{r}^j = \boldsymbol{V}^T\boldsymbol{V}_{\boldsymbol{x}}^j\tilde{\boldsymbol{x}} + \boldsymbol{V}^T\boldsymbol{n}^j = \boldsymbol{H}_{\boldsymbol{x},o}^j\tilde{\boldsymbol{x}} + \boldsymbol{n}_o^j. \tag{3.8}$$

The filter model inherited the update mechanism from [39] with modification for real-time execution. The authors have proposed to remove two camera states every other update step. The two-camera states are chosen according to the relative motion between the second latest camera state and its previous one. Either the second latest or the oldest camera state is selected for removal. This procedure is done twice to remove two states. The latest camera state is always kept due to new feature observations.

The FAST feature detector [42] processes the images and extracts the features from the image. Existing features are tracked using the KLT optical flow algorithm [43]. There are two types of outlier rejection procedures implemented. A 2-point RANSAC algorithm is implemented to remove outliers in temporal tracking. To further remove outliers generated in the feature tracking and stereo matching steps, a circular stereo matching algorithm similar to [44].

The S-MSCKF algorithm was compared to the state-of-art methods as OKVIS [27], ROVIO [23], and also VINS-mono algorithm, that was also included for comparison in Chapter 3.3. The VINS-mono algorithm includes a loop closure module which was disabled to compare just odometry of different algorithms. Algorithms were compared on the EuRoC datasets [15] in terms of Root Mean Square Error (RMSE) and the average CPU load, as shown in Figure 3.1.

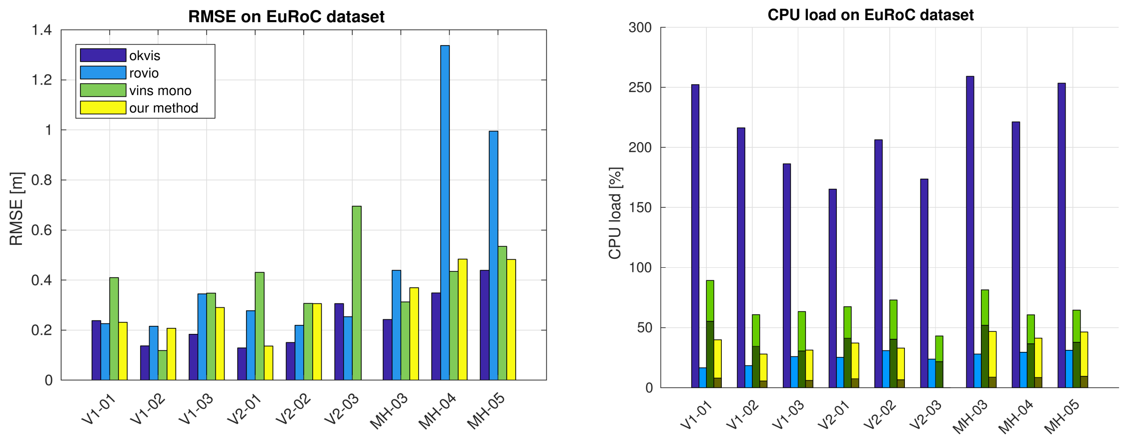


Figure 3.1: RMSE and CPU comparision of OKVIS, ROVIO, VINS-Mono and S-MSCKF algorithm on EuRoC dataset. Courtesy of [22]

Results show that filter-based (mono/stereo) methods are more efficient in terms of CPU usage than optimization-based methods (OKVIS, VINS-Mono). OKVIS has higher CPU usage than VINS-Mono mainly due to Harris corner detector [45] and BRISK descriptor [46] for both temporal, and stereo matching and also the OKVIS backend runs at the fastest possible rate, but VINS-Mono runs on 10 Hz fixed rate. The VINS-Mono rate is possible to set to a higher rate. However, this is the default rate set by the authors.

The authors have also created their own fast flight dataset with top velocities of $5\,\mathrm{m\,s^{-1}}$, $10\,\mathrm{m\,s^{-1}}$, $15\,\mathrm{m\,s^{-1}}$ and $17.5\,\mathrm{m\,s^{-1}}$. The ROVIO algorithm was omitted in these datasets due to a massive drift in the scale, resulting in low accuracy in comparison with other methods. The S-MSCKF method has slightly worse RMSE than the rest of the algorithms, but the lowest CPU load, despite the longer time spent on the image processing due to faster movements. All details can be found in [22]. It showed that S-MCSKF achieves robustness with a modest computational budget for aggressive movements, high velocity, and indoor/outdoor transition.

3.1.1 Implementation

The authors implemented and published S-MSCKF on GitHub[1], provided a ROS package with installation and usage instructions. There are also configuration files available for camera types used in the examples. Calibration files containing intrinsic and extrinsic camera parameters plus transformation matrix between IMU and each camera. These parameters were obtained with the Kalibr tool, as described in Chapter 2.2.1. The S-MSCKF uses a similar format of the calibration file format as the output YAML file from the Kalibr.

The implementation consists of two ROS nodes. First is the image processor node, which processes the images and publishes detected image features. The image processor node allows setting the parameters to tune the image processing such as:

1. The number of columns and rows the image is divided into
2. The minimal and maximal number of features in every cell
3. The FAST feature detector parameters such as image pyramid levels, patch size, FAST threshold, etc.

The second is the VIO node, which processes the IMU messages and tracked features in consecutive images. The VIO node has several parameters to set, such as:

1. The maximum number of camera states to maintain
2. The keyframe selection thresholds according to rotation and translation change
3. The standard deviations of the IMU inputs and feature observations
4. The initial conditions of the MSCKF filter

The node publishes the position and velocity in the ROS *Odometry* message. Precise parameter settings, as well as extrinsic calibrations, are essential to initialize the filter properly.

To be able to compare the calculated output, the transformation of the odometry message has to be applied. It was necessary to create a transformation publisher between the *local_origin* frame in which the ground truth is published, and the *msckf_origin* frame where the algorithm odometry is published since these two coordinate frames do not correspond. The *local_origin* frame is defined by GPS coordinates with the x-axis facing magnitude north, while the *msckf_origin* is defined by the camera pose at the start of the algorithm. Then the odometry message itself has to be transformed by the transformation, so it is comparable with the ground truth values.

[1]`https://github.com/KumarRobotics/msckf_vio`

3.2 Semi-Direct Visual Odometry (SVO)

SVO [25] is a visual odometry algorithm that uses a direct method to track and triangulate pixels while using feature-based methods for joint optimization of the structure from motion (SfM). Using a robust probabilistic depth estimation algorithm enables us to track pixels on weak corners and edges efficiently. The algorithm could be easily extended to multiple cameras, to include motion priors from motion sensors or using a very wide field of view cameras. The version that is described here is the extended version of the previous work of authors [47]. Additionally, the presented version includes the generalization for multiple-camera systems, motion priors assumption, and edge features use. Because of the motion priors coverage, the SVO is treated as a VIO algorithm.

The authors have designed a separated thread system containing Motion Estimation thread and Mapping thread. The motion estimation thread task is to perform the semi-direct approach to motion estimation, additional feature alignment to eliminate drift and final refinement.

The camera C at time k provides an intensity image I_k^c. A 3D point $\boldsymbol{\rho} \in \mathbb{R}^3$ is mapped to the image coordinates $\boldsymbol{u} \in \mathbb{R}^2$ through the camera projection model

$$\boldsymbol{u} = \pi(\boldsymbol{\rho}). \tag{3.9}$$

The camera projection model is known from the calibration process as mentioned in Chapter 2.2.1. The 2D image point (pixel) $\boldsymbol{u} \in \mathcal{R}_k^C$ with the inverse depth $\rho > 0$ can be back-projected to the 3D point with

$$\boldsymbol{\rho} = \pi_\rho^{-1}(\boldsymbol{u}), \tag{3.10}$$

where $\mathcal{R}_k^C$ are pixels with known depth at time k in camera C. The $() \cdot ()$ symbolizes the scalar multiplication between two elements.

The authors proposed a sparse image alignment model based on the image to model alignment, which estimates the incremental camera motion by minimizing the intensity difference (photometric error) of pixels observing the same 3D point.

To allow generalization for multiple cameras, a body frame B is introduced. This frame is rigidly attached to camera frame C with known extrinsic calibration transformation. This relations are described in Figure 3.2.

The goal is to estimate the body frame incremental motion $T_{k,k-1}$ that minimizes the photometric error between the patches in the frames:

$$T_{k,k-1}^* = \arg\min_{T_{k,k-1}} \sum_{\boldsymbol{u} \in \bar{\boldsymbol{R}}_{k-1}^C} \frac{1}{2} ||\boldsymbol{r}_{I_{\boldsymbol{u}}^C}(T_{k,k-1})||^2_{\sum_I}, \tag{3.11}$$

where $\boldsymbol{r}_{I_{\boldsymbol{u}}^C}$ is the photometric residual and it is defined by the intensity difference of pixels in subsequent images I_k^c and I_{k-1}^c observing the same 3D point $\boldsymbol{\rho_u}$

$$\boldsymbol{r}_{I_{\boldsymbol{u}}^C}(T_{k,k-1}) \doteq I_k^c\big(\pi(T_{CB}T_{k,k-1}\ \boldsymbol{\rho_u})\big) - I_{k-1}^c\big(\pi(T_{CB}\ \boldsymbol{\rho_u})\big). \tag{3.12}$$

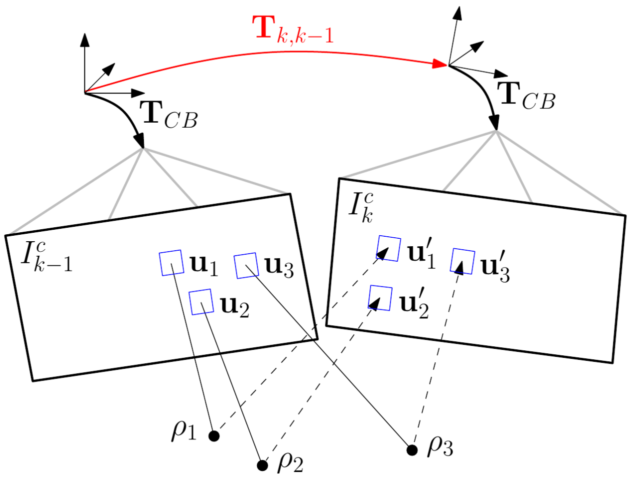


Figure 3.2: The change of the relative pose $T_{k,k-1}$ between the current and the previous frame implicitly moves the position of the reprojected points $\boldsymbol{u}'_i$ in the new image. Courtesy of [25].

The optimization in Equation (3.11) includes only those pixels, for which the backprojected points are also visible in the I^c_k. The authors have proposed a sparse image alignment approach that assumes known depth just for corners and features lying on intensity edges. To increase the approach robustness, the small patch photometric cost aggregation was proposed with the assumption of the same depth for the neighboring pixels. It is solved with a standard iterative non-linear least-squares method, such as Levenberg-Marquardt.

Next, the authors proposed relaxing the geometric constraints given by the reprojection of 3D points and perform 2D alignment of corresponding feature patches. However, this alignment generates a reprojection error between the projected 3D point and the aligned feature. Therefore a bundle adjustment process has to be implemented to optimize 3D positions and camera poses, minimizing the reprojection error. Predicted corner feature position $\boldsymbol{u}'$, where

$$\boldsymbol{u}' = \pi\Big(T_{CB}\ T_{kr}\ T_{BC}\ \pi_\rho^{-1}(\boldsymbol{u})\Big) \tag{3.13}$$

has to be corrected by a correction $\delta\boldsymbol{u}^* \in \mathbb{R}^2$ with respect to the newest frame k minimizing the photometric error, where

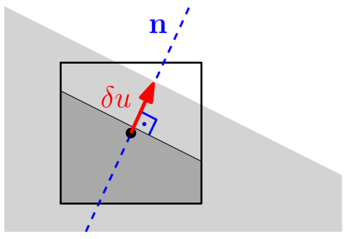
$$\delta\boldsymbol{u}^* = \arg\min_{\delta\boldsymbol{u}} \sum_{\Delta\boldsymbol{u}\in\mathcal{P}} \frac{1}{2}\Big|\Big|I^c_k(\boldsymbol{u}' + \delta\boldsymbol{u} + \Delta\boldsymbol{u}) - I^C_r(\boldsymbol{u} + \boldsymbol{A}\Delta\boldsymbol{u})\Big|\Big|^2. \tag{3.14}$$

where $\Delta\boldsymbol{u}$ is the iterator variable calculating sum over patch $\mathcal{P}$, r is the reference frame of the first observation of the feature. Affine warping $\boldsymbol{A}$ is applied to the reference patch, which is computed from the estimated T_{kr} between the reference frame r and current frame k. The corrected feature position $\boldsymbol{u}'^*$ is then

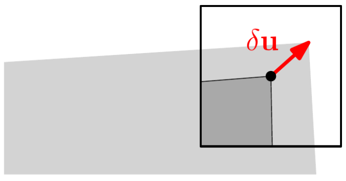
$$\boldsymbol{u}'^* = \boldsymbol{u}' + \delta\boldsymbol{u}^*, \tag{3.15}$$

Edge features are facing the so-called aperture problem. Hence the alignment freedom is limited to the normal direction of the edge. Edge features are optimized with the scalar correction $\delta u^* \in \mathbb{R}$ in the direction of the edge normal $\boldsymbol{n}$, giving equation

$$\delta u^* = arg\min_{\delta u} \sum_{\Delta\boldsymbol{u}\in\mathcal{P}} \frac{1}{2}\Big|\Big|I^c_k(\boldsymbol{u}' + \delta u\cdot\boldsymbol{n} + \Delta\boldsymbol{u}) - I^C_r(\boldsymbol{u} + \boldsymbol{A}\Delta\boldsymbol{u})\Big|\Big|^2. \tag{3.16}$$



(a) Edge alignment.



(b) Corner alignment.

Figure 3.3: Example of the described feature alignment strategies. Courtesy of [25].

To obtain feature position $\boldsymbol{u}'^*$ in the newest frame, the predicted positon $\boldsymbol{u}'$ from Equation (3.13) is corrected with scalar correction δu^* as

$$\boldsymbol{u}'^* = \boldsymbol{u}' + \delta u^* \cdot \boldsymbol{n}. \tag{3.17}$$

The example of alignment strategies is shown in Figure 3.3.

Finally, the refining of the camera poses and landmark positions $\mathcal{X} = \{T_{kw}, \boldsymbol{\rho}_i\}$ has to be performed by minimizing the squared sum of reprojection errors

$$\mathcal{X}^* = arg \min_{\mathcal{X}} \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{L}_k^C} \frac{1}{2} ||\boldsymbol{u}_i'^* - \pi(T_{CB} T_{kw} \boldsymbol{\rho}_i)||^2 + \sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{L}_k^E} \frac{1}{2} ||\boldsymbol{n} \boldsymbol{u}_i'^* - \pi(T_{CB} T_{kw} \boldsymbol{\rho}_i)||^2, \tag{3.18}$$

where $\mathcal{K}$ is the set of all keyframes in the map, $\mathcal{L}_k^C$ is the set of all landmarks corresponding to corner features, $\mathcal{L}_k^E$ is the set of all landmarks corresponding to edge features observed in the k-th camera frame.

The rigid transformation T_{kw} represents the world frame orientation and position concerning the k-th camera frame. A point expressed in the world frame can be transformed to the k-th camera frame as

$$\boldsymbol{\rho}_k = T_{kw} \boldsymbol{\rho}_w \tag{3.19}$$

Previously, the depth was assumed to be known. Now the depth estimation process is described. Single-pixel depth is estimated from multiple observations utilizing a recursive Bayesian depth filter. The depth filter is modeled with a two-dimensional distribution. The illustration of depth estimation is shown in Figure 3.4.

The first dimension is the inverse depth ρ, and the second dimension is the inlier probability of γ. Hence a measurement $\tilde{\rho}_i^k$ is modeled with Gaussian distribution around the true inverse depth ρ_i and Uniform distribution for outlier measurement that arises from interval $[\rho_i^{min}, \rho_i^{max}]$mixture model distribution

$$p(\tilde{\rho}_i^k | \rho_i, \gamma_i) = \gamma_i \mathcal{N}(\tilde{\rho}_i^k | \rho_i, \tau_i^2) + (1 - \gamma_i) \mathcal{U}(\tilde{\rho}_i^k | \rho_i^{min}, \rho_i^{max}). \tag{3.20}$$

The described algorithm has been proposed only for the monocular camera. However, due to the generalization of the motion estimation algorithm, it can be extended to the multi-camera system. Then, the cost function contains additional summation for all $c \in C$ for all

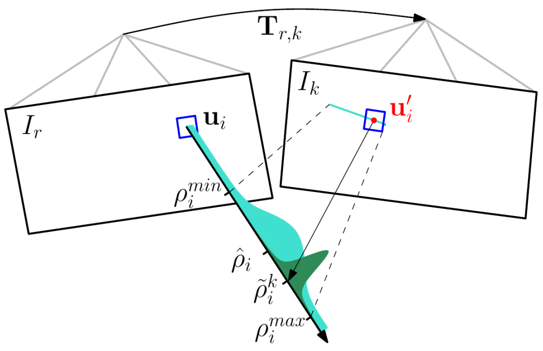


Figure 3.4: Probabilistic depth estimate $\hat{\rho}_i$ for feature i in the reference frame r. The depth estimate $\hat{\rho}_i$ is updated with the triangulated depth $\tilde{\rho}_i^k$ computed from the point $\boldsymbol{u}'_i$ of highest correlation with the reference patch. Courtesy of [25]

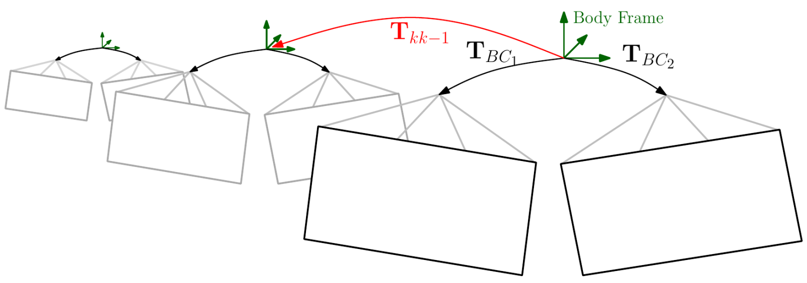


Figure 3.5: Example of interpretation of stereo camera system in SVO. The T_{BC_1} and T_{BC_2} are known from extrinsic calibration as described in Chapter 2.2.1. Courtesy of [25].

used cameras.

$$T^*_{k,k-1} = arg \min_{T_{k,k-1}} \sum_{c \in C} \sum_{\boldsymbol{u} \in \bar{\boldsymbol{R}}^C_{k-1}} \frac{1}{2} ||\boldsymbol{r}_{I^C_{\boldsymbol{u}}}(T_{k,k-1})||^2_{\sum_I} \tag{3.21}$$

The same modification in required for bundle adjustment step in Equation (3.18) to sum up the reprojection errors from all cameras. The example of stereo camera system interpretation in SVO is shown in Figure 3.5.

Next, the motion priors are added to the sparse image alignment process resulting in the following cost:

$$\begin{aligned} T^*_{k,k-1} = \arg \min_{T_{k,k-1}} \sum_{c \in C} \sum_{\boldsymbol{u} \in \bar{\boldsymbol{R}}^C_{k-1}} \frac{1}{2} ||\boldsymbol{r}_{I^C_{\boldsymbol{u}}}(T_{k,k-1})||^2_{\sum_I} \\ + \frac{1}{2} ||\boldsymbol{p}_{k,k-1} - \tilde{\boldsymbol{p}}_{k,k-1}||^2_{\sum_p} \\ + \frac{1}{2} || \log(\tilde{R}^T_{k,k-1} R_{k,k-1})||^2_{\sum_R}, \end{aligned} \tag{3.22}$$

where $\tilde{\boldsymbol{p}}_{k,k-1}$ is the relative translation prior, $\tilde{R}^T_{k,k-1}$ is the relative rotation prior. The covariances $\sum_p$ and $\sum_R$ are set according to uncertainty of the motion prior and the variables $(\boldsymbol{p}_{k,k-1}, R_{k,k-1}) \doteq T_{k,k-1}$ are the current estimate of the relative position and orientation.

The sparse approach does not reach the same convergence radius in terms of distance to the reference image. Thus SVO uses sparse image alignment only to align the current image to the previous image, not to the last keyframe.

The SVO was compared with DSO, ORB-SLAM, and LSD-SLAM. Due to the complexity of the SVO, it has been tested in various configurations (monocular/stereo, with FAST corners only, with edgelets, with motion priors, etc.) against the other algorithms. SVO extracts features just for selected keyframes. Hence it prospers from high frame-rate cameras due to not fixed cost per frame because the proposed sparse image alignment step is initialized closer to the final solution resulting in faster convergence. Then higher frame-rate reduces the computational cost per frame. SVO is the most robust while running with high frame-rate (40 to 80 frames per second). Another benefit of SVO is the direct start of optimization, allowing the integration of multiple camera measurements as well as motion priors.

3.2.1 Implementation

The algorithm binaries are available upon request on the developer's website as mentioned in their GitHub account[2]. Due to the binary version, a specific installation process in a separate ROS workspace is required. Authors provided example calibration files for EuRoC datasets [15] as well as parameter files to set up the SVO algorithm. Kalibr tool, described in Chapter 2.2.1, can provide the required calibration files. The authors prepared a script to convert the Kalibr calibration output to the SVO format.

The SVO parameter file allows adjusting the algorithm configuration. SVO was originally only a monocular VO. However, the authors have implemented a stereo camera option and also included IMU as motion priors. The parameters have to be set accordingly to the used camera resolution and model, IMU, and computational power available. The main configurable parameters are:

1. The sensor suite setup - Mono or stereo camera, the IMU used or not
2. Feature related - grid size, number of features, keyframe selection criterions, pose optimization, etc.
3. IMU related - Gyroscope prior in pose optimization and image alignment, velocity assumption in translation

The estimated position is published in a ROS *PoseWithCovariance* message as a point, instead of *Odometry* message. The SVO publishes the estimated position of IMU and stereo cameras in individual frames. The frame can be attached to the *local_origin* frame, which is used for the drone navigation using a static transform built from the ground truth pose of the start of the algorithm. Then the position estimated by the algorithm can be compared with the ground truth value.

[2]`https://github.com/uzh-rpg/rpg_svo_example`

3.3 VINS-Mono

VINS-Mono [48] is a monocular visual-inertial state estimator using pre-integrated IMU factors. The authors propose a tightly coupled non-linear optimization-based method to obtain highly accurate visual-inertial odometry by fusing pre-integrated IMU measurements and feature observations. Furthermore, the authors presented the following features:

1. The robust initialization procedure enables to bootstrap the system from unknown initial states, including the online camera-IMU extrinsic calibration module.
2. Loop detection module with relocalization and four degrees-of-freedom (DoF) global pose graph optimization to enforce global consistency.
3. Pose graph reuse enables to save, load, and merge multiple local pose graphs.

The first step of the VINS-Mono algorithm is measurement preprocessing, in which the images and IMU measurements are processed. Corner features are tracked in successive frames with KLT sparse optical flow algorithm [43], and new features are detected in the latest frame with [49]. Outlier rejection is performed using the RANSAC algorithm.

Keyframe selection has two criteria. The first one is the average parallax caused by translation from the previous keyframe. The parallax can also be caused by rotation. However, the features cannot be triangulated in the rotation-only motion. To prevent the influence of rotation on the parallax calculation, a short-term gyroscope measurement integration is used to compensate the rotation. This rotation is compensated just for keyframe selection, and it is not used in the VIO calculation process itself. If the average parallax of tracked features between the current frame and the latest keyframe is above a certain threshold, it is considered as a new keyframe. Another criterion is tracking quality. If the number of tracked features goes below a certain threshold, this frame is treated as a new keyframe. This criterion is present to avoid complete loss of feature tracked.

The authors have defined the notations for the world frame as $(\cdot)^w$, where gravity is aligned with the z-axis of the world frame. The body frame is denoted as $(\cdot)^b$, which is also the IMU frame. The camera frame is denoted as $(\cdot)^c$. The transformation from the body frame to the world frame is denoted as $(\cdot)^w_b$. The noisy measurements or satisfied quantity estimation is denoted as $\hat{(\cdot)}$.

The IMU measurements are pre-integrated between two consecutive frames. At first, the IMU noise and bias have to be handled. The IMU measurements, which are measured in the body frame, combine the gravity force with the platform dynamics and are affected by the acceleration bias $\boldsymbol{b}_a$, the gyroscope bias $\boldsymbol{b}_w$ and the additive noise $\boldsymbol{n}_a$. Accelerometer $\hat{\boldsymbol{a}}$ and gyroscope $\hat{\boldsymbol{\omega}}$ raw measurements are given by

$$\hat{\boldsymbol{a}}_t = \boldsymbol{a}_t + \boldsymbol{b}_{a_t} + \boldsymbol{R}^t_w \boldsymbol{g}^w + \boldsymbol{n}_a, \tag{3.23}$$

$$\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}_t + \boldsymbol{b}_{w_t} + \boldsymbol{n}_w, \tag{3.24}$$

assuming the additive noise in acceleration and gyroscope measurements is Gaussian white noise, $\boldsymbol{n}_a \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\sigma}^2_a)$, $\boldsymbol{n}_w \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\sigma}^2_w)$. The acceleration and the gyroscope biases are

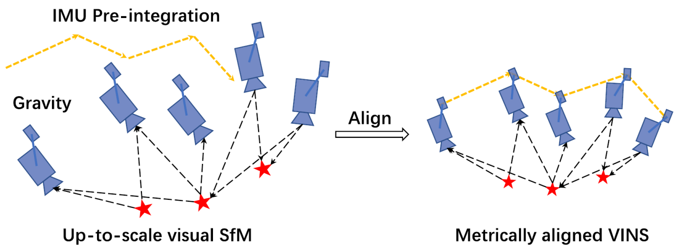


Figure 3.6: Illustration of the visual-inertial alignment for VINS-Mono. Courtesy of [48].

modeled as random walk, whose derivatives are Gaussian white noise, $\boldsymbol{n}_{b_a} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\sigma}^2_{b_a})$, $\boldsymbol{n}_{b_w} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\sigma}^2_{b_w})$

$$\dot{\boldsymbol{b}}_{a_t} = \boldsymbol{n}_{b_a} \tag{3.25}$$

$$\dot{\boldsymbol{b}}_{w_t} = \boldsymbol{n}_{b_w} \tag{3.26}$$

For two time consecutive frames b_k and b_{k+1}, there exist several inertial measurements in time interval $[t_k, t_{k+1}]$. Given the bias estimate, the measurements are integrated into the local frame b_k. The integration process can be found in [48].

VINS-Mono requires a precise initial guess, which is obtained by loosely aligning IMU pre-integration with the vision-only structure. The procedure is illustrated in Figure 3.6, starting with visual structure from motion (SfM) that estimates a graph of camera poses and feature positions. Computational complexity is bounded by keeping a fixed number of frames in the sliding window. Next, feature correspondence between the latest frame and all previous frames in the sliding window is checked. If the stable amount of features (more than 30) can be found and there is sufficient parallax (more than 20 pixels) between the latest frame and any other frame in the sliding window, it recovers the relative rotation and up-to-scale translation between these two frames using five-point algorithm [50]. Then arbitrarily chooses the scale and triangulates all observed features. Based on these points, a perspective n-point (PnP) method [51] is applied to estimate the poses of all other frames in the window, and a global bundle adjustment is applied to minimize the total reprojection error of all observed features. While there is not a piece of knowledge about the world frame yet, the first camera frame is marked as a reference for SfM. Hence all frame poses $(\bar{\boldsymbol{p}}^{c_0}_{c_k}, \boldsymbol{q}^{c_0}_{c_k})$ and feature positions are represented with respect to the first camera frame c_0. Knowing extrinsic calibration between camera and IMU $(\boldsymbol{p}^b_c, \boldsymbol{q}^b_c)$, the camera frame can be expressed in the IMU (body) frame

$$\boldsymbol{q}^{c_0}_{b_k} = \boldsymbol{q}^{c_0}_{c_k} \otimes (\boldsymbol{q}^b_c)^{-1}, \tag{3.27}$$

$$s\bar{\boldsymbol{p}}^{c_0}_{b_k} = s\bar{\boldsymbol{p}}^{c_0}_{c_k} - \boldsymbol{R}^{c_0}_{b_k}\boldsymbol{p}^b_c, \tag{3.28}$$

where s is the unknown scaling parameter. The $\otimes$ symbol denotes quaternion multiplication. The next step is to align the IMU pre-integration with the up-to-scale visual structure. VINS linearizes the IMU pre-integration concerning gyroscope bias minimizing appropriate cost function resulting in the initial calibration of gyroscope bias for VIO. Next, other essential states have to be initialized, specifically the velocity, the gravity vector, and the metric scale.

$$\mathcal{X}_I = \left[\boldsymbol{v}^{b_0}_{b_0}, \boldsymbol{v}^{b_1}_{b_1}, ..., \boldsymbol{v}^{b_n}_{b_n}, \boldsymbol{g}^{c_0}, s\right], \tag{3.29}$$

where $\boldsymbol{v}^{b_k}_{b_k}$ is the body frame velocity for the k^{th} image, $\boldsymbol{g}^{c_0}$ is the gravity vector in the c_0 frame and s is the scaling parameter scaling SfM to metric units. The $\mathcal{X}_I$ values are found by minimizing an appropriate cost function. The gravity vector g can be further refined, constraining the magnitude. The magnitude is more or less known, which decreases the problem dimension to 2-DoF. Thus the gravity vector is perturbed with two variables on its tangential space. The gravity vector g is perturbed by $\delta\boldsymbol{g}$ as follows

$$g(\overrightarrow{\hat{\boldsymbol{g}}} + \delta\boldsymbol{g}), \delta\boldsymbol{g} = w_1\boldsymbol{b}_1 + w_2\boldsymbol{b}_2, \tag{3.30}$$

where g is the known magnitude of the gravity, $\overrightarrow{\hat{\boldsymbol{g}}}$ is a unit vector of the gravity direction and $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ are two orthogonal basis spanning the tangential plane. The $g(\overrightarrow{\hat{\boldsymbol{g}}} + \delta\boldsymbol{g})$ is then substituted into the $\mathcal{X}_I$ and solved with other state variables until it converges. Gravity refinement gives the rotation $\boldsymbol{q}^w_{c_0}$ between the world frame and the camera frame c_0 by rotating the gravity to the z-axis. All variables from body frame b and camera frame c_0 are rotated to the world frame w, body frame velocities are also rotated to the world frame, and translation components from the visual SfM are scaled to metric units. Detailed information regarding the cost functions and equation can be found in the [48].

The tightly coupled VIO is then formulated with the full state vector in the sliding window as

$$\begin{aligned}
\mathcal{X} &= [\boldsymbol{x}_0, \boldsymbol{x}_1, ...\boldsymbol{x}_n, \boldsymbol{x}^b_c, \lambda_0, \lambda_1, ...\lambda_m], \\
\boldsymbol{x}_k &= [\boldsymbol{p}^w_{b_k}, \boldsymbol{v}^w_{b_k}, \boldsymbol{q}^w_{b_k}, \boldsymbol{b}_a, \boldsymbol{b}_g], k \in [0, n], \\
\boldsymbol{x}^b_c &= [\boldsymbol{p}^b_c, \boldsymbol{q}^b_c],
\end{aligned} \tag{3.31}$$

where $\boldsymbol{x}_k$ is IMU state at the time when the k-th image is captured. Position $\boldsymbol{p}^w_{b_k}$, velocity $\boldsymbol{v}^w_{b_k}$ and orientation $\boldsymbol{q}^w_{b_k}$ of the IMU are expressed in the world frame and acceleration bias $\boldsymbol{b}_a$ and gyroscope bias $\boldsymbol{b}_g$ are expressed in the body frame, which is the IMU frame. The total number of keyframes is denoted by n, and m is the total number of features in the sliding window. The inverse distance of the l-th feature from its first observation is marked as λ_l. The authors use a visual-inertial bundle adjustment formulation and minimize the sum of the prior and the Mahalanobis form of all measurement residuals to obtain a maximum posterior estimation

$$\min_{\mathcal{X}} \left\{ ||\boldsymbol{r}_p - \boldsymbol{H}_p\mathcal{X}||^2 + \sum_{k\in\mathcal{B}} ||\boldsymbol{r}_{\mathcal{B}}(\hat{\boldsymbol{z}}^{b_k}_{b_{k+1}}, \mathcal{X})||^2_{\boldsymbol{P}^{b_k}_{b_{k+1}}} + \sum_{(l,j)\in\mathcal{C}} \rho(||\boldsymbol{r}_{\mathcal{C}}(\hat{\boldsymbol{z}}^{c_j}_l, \mathcal{X})||^2_{\boldsymbol{P}^{c_j}_l}) \right\}, \tag{3.32}$$

where $\boldsymbol{r}_{\mathcal{B}}(\hat{\boldsymbol{z}}^{b_k}_{b_{k+1}}, \mathcal{X})$ and $\boldsymbol{r}_{\mathcal{C}}(\hat{\boldsymbol{z}}^{c_j}_l, \mathcal{X})$ are residuals for IMU and visual measurements, $\mathcal{B}$ is the set of all IMU measurements, $\mathcal{C}$ is the set of features that have been spotted at least twice in the sliding window and $\{\boldsymbol{e}_p, \boldsymbol{H}_p\}$ is the prior information from prior knowledge and marginalization. The Huber norm ρ [52] is defined as

$$\rho(s) = \begin{cases} s & s \leq 1 \\ 2\sqrt{s} - 1 & s > 1. \end{cases} \tag{3.33}$$

This non-linear optimization problem is solved with Ceres solver [53]. The details of IMU and visual measurement residuals minimization can be found in [48].

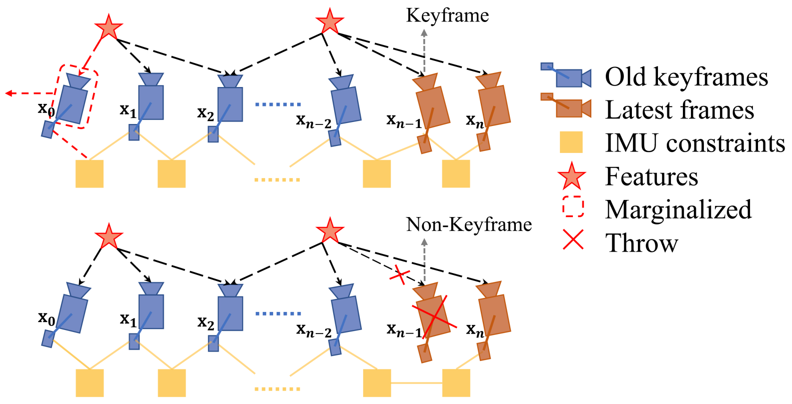


Figure 3.7: Illustration of the marginalization process for VINS-Mono. Courtesy of [48].

The optimization complexity is bounded by marginalization. The IMU states $\boldsymbol{x}_k$ and features λ_l are marginalized out from the sliding window, and measurements corresponding to marginalized states are converted into a prior. The new prior based on all marginalized measurements related to the removed state is added to the existing one. If the second latest frame is a keyframe, it stays in the window, and the oldest state is marginalized out with its corresponding measurements. Otherwise, if the second latest frame is a nonkeyframe, it discards visual measurements and keeps IMU measurements that connect to the nonkeyframe, and the pre-integration process continues to the next frame. The marginalization scheme aims to keep spatially separated keyframes in the window. This process ensures sufficient parallax for feature triangulation and maximizes the probability of maintaining accelerometer measurements with significant excitation. The marginalization process is shown in Figure 3.7

Bounded complexity also brings accumulating drift to the system. An elimination process has to be proposed. Authors came with tightly coupled relocalization that is smoothly integrated with the monocular VIO. Relocalization starts with a loop detection module, identifying already visited places. If there exists a candidate for loop closure, feature-level connections are established. These connections are tightly integrated into the VIO module, resulting in a driftless state estimate with minimal computation power.

The authors use the DBoW2 [54] bag-of-words place recognition approach for loop detection. The corner feature detector [49] is used to detect 500 more features for loop detection. The features are described by the BRIEF descriptor [55]. Resulting BRIEF descriptions are used as the visual word to query the visual database DBoW2. DBoW2 returns loop-closure candidates after temporal and geometrical consistency checks. All BRIEF descriptors are kept for retrieving features, and raw images are discarded to reduce memory consumption.

The relocalization process has to be finalized with the alignment of the current sliding window to the past poses. The nonlinear cost function from Equation (3.32) is modified by adding the following term

$$\sum_{(l,v)\in\mathcal{L}} \rho(||\boldsymbol{r}_{\mathcal{C}}(\hat{\boldsymbol{z}}_l^v, \mathcal{X}, \hat{\boldsymbol{q}}_v^w, \hat{\boldsymbol{p}}_v^w)||^2_{\boldsymbol{P}_l^{c_v}}), \tag{3.34}$$

where $\mathcal{L}$ is the set of observations of retrieved features in the loop-closure frames, (l, v) is l-th feature observed in the loop-closure frame v. Despite the cost function extension, the dimension of the states to be solved is the same, because loop-closure frames poses are treated as constants.

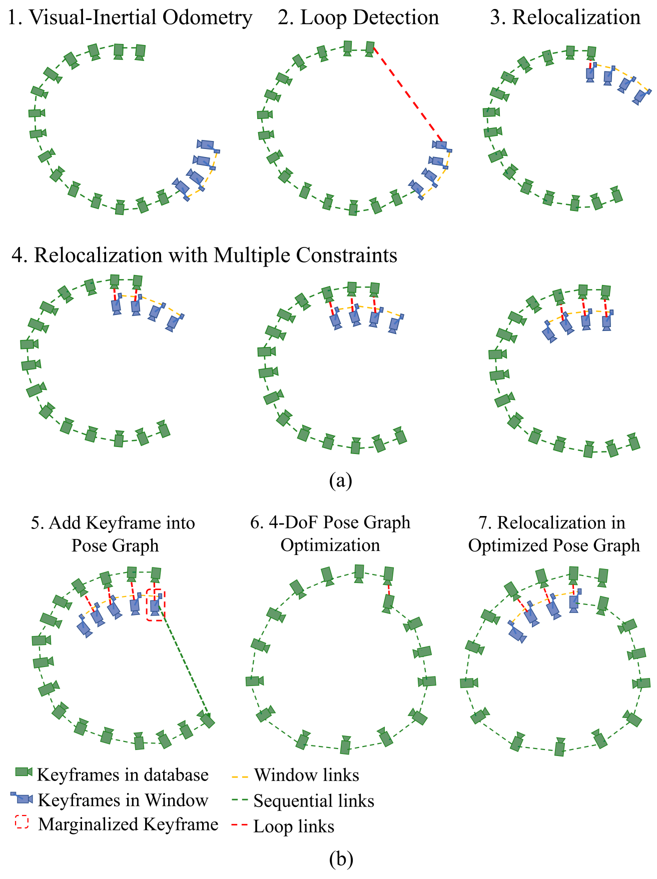

1. Visual-Inertial Odometry
2. Loop Detection
3. Relocalization
4. Relocalization with Multiple Constraints
(a)
5. Add Keyframe into Pose Graph
6. 4-DoF Pose Graph Optimization
7. Relocalization in Optimized Pose Graph
Keyframes in database
Keyframes in Window
Marginalized Keyframe
Window links
Sequential links
Loop links
(b)

Figure 3.8: Diagram illustrates the relocalization (a) and pose graph optimization (b) process. While the VIO is estimating the pose, the loop detection (red line) occurs between the current keyframe (blue) and past keyframe(green). Thanks to feature-level correspondences, the relocalization process handles the loop-closure constraints from several past keyframes. The keyframe is added to pose-graph after marginalization. If the loop detection finds the loop-closure candidate from the past keyframes, the 4-DoF relative transformation is also added to the pose graph. Courtesy of [48].

The relocalization ensures the finding of previously visited spots. However, the relocalized image frames have to be optimized to ensure global consistency and eliminate the drift. Thanks to the observability of the horizontal plane (roll and pitch angle) in the system by the gravity vectors, the problem reduces to 4-DoF. The keyframes are added to the pose graph as vertices, and they are connected to other vertexes (keyframes) by either a sequential edge or a loop edge. Sequential edge is drawn from a keyframe to several of its previous keyframes, as shown in Figure 3.9. It represents the relative transformation between two keyframes.

The edge is described by

$$\begin{aligned}\hat{\boldsymbol{p}}^i_{ij} &= \hat{\boldsymbol{R}}^{w\,-1}_i(\hat{\boldsymbol{p}}^w_j - \hat{\boldsymbol{p}}^w_i),\\ \hat{\psi}_{ij} &= \hat{\psi}_j - \hat{\psi}_i,\end{aligned} \tag{3.35}$$

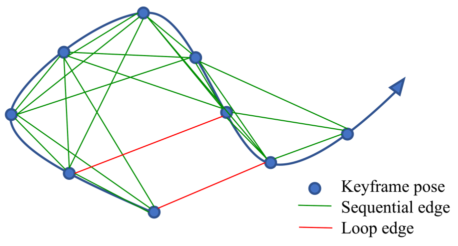


Figure 3.9: Illustration of the pose graph for VINS-Mono. Courtesy of [48].

where i is the current keyframe and j is one of the previous keyframes, $\hat{\boldsymbol{p}}_{ij}^{i}$ is the relative position and $\hat{\psi}_{ij}$ is the yaw angle. The $\hat{\boldsymbol{R}}_{i}^{w-1}$ denotes the rotation between the world frame w and the i^{th} keyframe The loop edge is composed of the same components as shown in Equation (3.35). The values are obtained from relocalization results.

The whole graph of sequential edges and loop edges is optimized by minimizing the following cost function:

$$\min_{\boldsymbol{p},\psi}\left\{\sum_{(i,j)\in\mathcal{S}}||\boldsymbol{r}_{i,j}||^2+\sum_{(i,j)\in\mathcal{L}}\rho(||\boldsymbol{r}_{i,j}||^2)\right\}, \tag{3.36}$$

where $\mathcal{S}$ is the set of all sequential edges, $\mathcal{L}$ is the set of all loop-closure edges. The Huber norm ρ is used to reduce the possible wrong loop detections further. The residual $\boldsymbol{r}_{i,j}$ is shown in [48]. To distribute the load and speed up the computing, the VIO and pose graph optimization module run concurrently in separate threads. The illustration of marginalization together with pose graph optimization is shown in Figure 3.8.

The authors also proposed a pose graph merging method. It allows us to merge pose graphs from different launch instances with the loop connections. Pose graph can be saved into a file with the following structure for i-th keyframe:

$$[i,\ \hat{\boldsymbol{p}}_{i}^{w},\ \hat{\boldsymbol{q}}_{i}^{w},\ v,\ \hat{\boldsymbol{p}}_{i_v}^{i},\ \hat{\psi}_{iv},\ \boldsymbol{D}(u,v,des)], \tag{3.37}$$

where $\hat{\boldsymbol{p}}_{i}^{w}, \hat{\boldsymbol{q}}_{i}^{w}$ are position and orientation, respectively, of the i-th frame. If the frame has a loop-closure frame, the v is the loop-closure frame index. The relative position and yaw angle between the i-th and v-th frame are denoted as $\hat{\boldsymbol{p}}_{i_v}^{i}$, $\hat{\psi}_{iv}$. The feature set, where every feature contains the 2D location, and its BRIEF descriptor is expressed as $\boldsymbol{D}(u,v,des)$.

The VINS-Mono performance has been tested against the state-of-the-art OKVIS [27] algorithm on EuRoC datasets [15]. The authors used images from the left camera for the VINS. The comparison was made for both VINS-Mono with relocalization and loop closure as well as VINS-Mono without these enhancements. The full VINS-Mono outperformed the OKVIS in most of the experiments. The VINS-Mono without the relocalization and loop closure performed in most of the tests slightly better than OKVIS, but worse than full VINS-Mono.

Another test was performed to demonstrate the map merging feature. The authors used all of the EuRoC datasets one by one. New sequences were merged onto the previous map,

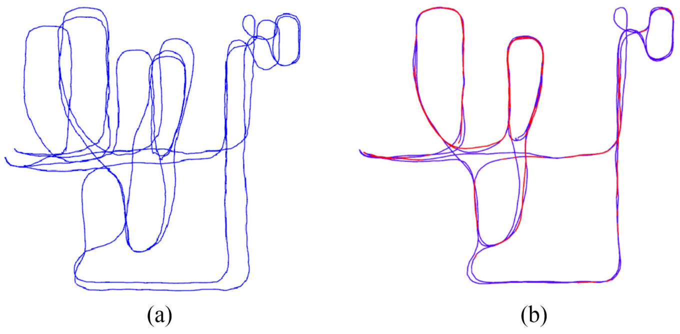


Figure 3.10: Indoor experiment comparison with OKVIS. The (a) figure is output from OKVIS, where noticable drift occurs especially during the circle movements. The (b) figure shows the VINS-Mono output with loop closure detection (red lines) showing successfully elimination of the drift. Courtesy of [48].

which was growing in size with each new dataset. The whole trajectory was compared to the ground truth resulting in ATE of 0.21 m.

At first, real-world experiments were performed by walking with the sensor suite indoors. Again, the results were compared against OKVIS as shown in Figure 3.10.

The results show that VIO drifts significantly in all axes (x, y, z) and yaw angle. Thanks to relocalization and loop-closure, VINS efficiently eliminates these drifts. The second experiment was held outside with walking again around the university (HKUST) campus. The total path length is 5.62 km. The estimated trajectory was projected to Google maps showing an almost drift-free trajectory in this long-duration test. Lastly, VINS was tested in the feedback control of an aerial robot showing the applicability of the algorithm. In this experiment, the loop closure module was switched off. Total trajectory length was 61.97 m. The final drift in every axis was [0.08, 0.09, 0.13]m resulting in 0.29 % positional drift.

The originally posted VINS-Mono algorithm was described. It is available on GitHub[3]. Additionally, this GitHub version contains online temporal calibration between the camera and IMU [56].

Most of the VIO algorithms assume precise temporal calibration between the camera and IMU. However, most of the low-cost devices or self-assembled devices violate this assumption. In real systems, the camera and IMU typically suffer from triggering and transmission delays resulting in temporal time offset among samples. This temporal offset is usually an unknown constant value and dramatically influences the performance of proper data fusion. Sometimes, due to different clocks in the sensors, the time offset might be evolving in time. Those kinds of sensors are inappropriate for fusion. Several papers have been proposed to estimate the time offset as [57], [58] or Kalibr. In this thesis, Kalibr is used for the estimation of camera intrinsic and extrinsic parameters, which is detailed in Chapter 2.2.1. Kalibr also provides a precise estimation of the time offset between camera and IMU. However, Kalibr runs offline on images of specific planar pattern with readily determined features (such as a chessboard). The authors define a constant unknown time offset t_d as

$$t_{IMU} = t_{cam} + t_d. \tag{3.38}$$

[3] `https://github.com/HKUST-Aerial-Robotics/VINS-Mono`

The time offset t_d defines the time value that should be added to the camera timestamp so that the camera and IMU streams become temporarily synchronized as shown in Figure 3.11. The algorithm shifts observations of features in the timeline instead of the whole camera frames or IMU sequence.

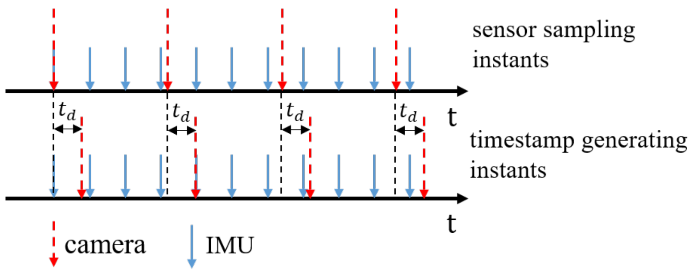


Figure 3.11: An illustration of temporal misalignment between camera and IMU streams. The generated timestamp is not equal to the actual asmpling time due to triggering delay, transmission delay resulting in a temporal misalignment. Courtesy of [56].

The authors modified the classical (re)projection error approach with a new variable, the time offset. The features are parametrized, usually in two ways. The first is the parametrization of a feature as its 3D position in the global frame, and the second is the parametrization of a feature as depth or inverse depth to a specific image frame. The authors formulated the new projection error including the t_d as

$$\boldsymbol{e}_l^k = \boldsymbol{z}_l^k(t_d) - \pi(\boldsymbol{R}_{c_k}^{w^T}(\boldsymbol{P}_l - \boldsymbol{p}_{c_k}^w)), \tag{3.39}$$

$$\boldsymbol{z}_l^k(t_d) = \begin{bmatrix} u_l^k & v_l^k \end{bmatrix}^T + t_d \boldsymbol{V}_l^k, \tag{3.40}$$

where $\boldsymbol{P}_l = [x_l, y_l, z_l]^T$ is the feature parametrized as 3D position, $\boldsymbol{z}_l^k$ is the observation of feature l in frame k, $(\boldsymbol{R}_{c_k}^w, \boldsymbol{p}_{c_k}^w)$ is the camera pose transforming feature P_l from global frame to local camera frame, $\pi(.)$ is the camera projection model, $\boldsymbol{V}_l^k$ is the speed of the feature on the image plane, t_d is the unknown constant time offset. The formulation of the new reprojection error including t_d is

$$\boldsymbol{e}_l^j = \boldsymbol{z}_l^j(t_d) - \pi(\boldsymbol{R}_{c_j}^{w^T}(\boldsymbol{R}_{c_i}^w \lambda_i \begin{bmatrix} \boldsymbol{z}_l^i(t_d) \\ 1 \end{bmatrix} + \boldsymbol{p}_{c_i}^w - \boldsymbol{p}_{c_j}^w)), \tag{3.41}$$

$$\boldsymbol{z}_l^i = \begin{bmatrix} u_l^i & v_l^i \end{bmatrix}^T + t_d \boldsymbol{V}_l^i, \; \boldsymbol{z}_l^j = \begin{bmatrix} u_l^j & v_l^j \end{bmatrix}^T + t_d \boldsymbol{V}_l^j, \tag{3.42}$$

where λ_i is a depth in the image i.

The state variables are augmented with a time offset, which is defined as:

$$\mathcal{X} = [\boldsymbol{x}_0, \boldsymbol{x}_1, ...\boldsymbol{x}_n, \boldsymbol{P}_0, \boldsymbol{P}_1, ...\boldsymbol{P}_l, t_d], \tag{3.43}$$

$$\boldsymbol{x}_k = [\boldsymbol{p}_k^w, \boldsymbol{v}_k^w, \boldsymbol{R}_k^w, \boldsymbol{b}_a, \boldsymbol{b}_g], k \in [0, n], \tag{3.44}$$

where the k-th IMU state contains the position $\boldsymbol{p}_k^w$, velocity $\boldsymbol{v}_k^w$, orientation $\boldsymbol{R}_k^w$ in the global frame and IMU bias $\boldsymbol{b}_a, \boldsymbol{b}_g$ in the local body frame. Either 3D position or depth parametrization parametrize the feature $\boldsymbol{P}_l$.

The temporal calibration problem is then formulated in one cost function containing IMU propagation factor, reprojection error, prior factor, and the vision factor as follows:

$$\min_{\mathcal{X}} \left\{ ||\boldsymbol{e}_p - \boldsymbol{H}_p \mathcal{X}||^2 + \sum_{k \in \mathcal{B}} ||\boldsymbol{e}_{\mathcal{B}}(\boldsymbol{z}^k_{k+1}, \mathcal{X})||^2_{\boldsymbol{P}^k_{k+1}} + \sum_{(l,j) \in \mathcal{C}} ||\boldsymbol{e}_{\mathcal{C}}(\boldsymbol{z}^j_l, \mathcal{X})||^2_{\boldsymbol{P}^j_l} \right\}, \tag{3.45}$$

where $\boldsymbol{e}_{\mathcal{B}}(\boldsymbol{z}^k_{k+1}, \mathcal{X})$ is the error term from IMU pre-integration, $\mathcal{B}$ is the set of all IMU measurements, $\boldsymbol{e}_{\mathcal{C}}(\boldsymbol{z}^j_l, \mathcal{X})$ is the proposed vision factor, $\mathcal{C}$ is the set ot features which have been detected at least twice in the image frame, $\{\boldsymbol{e}_p, \boldsymbol{H}_p\}$ is the prior information from prior knowledge and marginalization. Marginalization is applied to bound the computational complexity. The marginalized out states are converted into prior. The non-linear cost function is optimized using Gauss-Newton methods.

The temporal calibration process was integrated into the VINS-Mono code. The authors compared the VINS-Mono RMSE without the time offset calibration and with it showing the noticeable difference when the time offset is present, as shown in Figure 3.12. It depicts that time offset influences odometry precision.

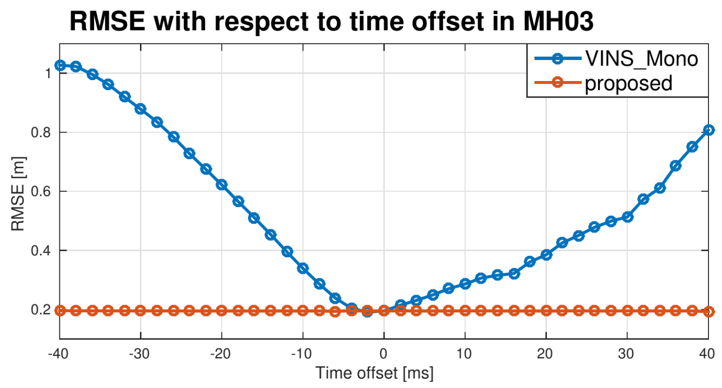


Figure 3.12: The comparison of original VINS-Mono and the VINS-Mono with temporal alignment integrated. The x-axis shows the chosen time offsets and the y-axis the RMSE. Courtesy of [56].

Next, they compared VINS-Mono with the time offset calibration against the OKVIS algorithm on EuRoC datasets. However, EuRoC datasets are perfectly time-synchronized. Hence authors purposely shifted the timestamp of the dataset IMU of 5 ms and 30 ms for both algorithms. VINS-Mono with the temporal calibration has stable RMSE at both time shifts, but OKVIS RMSE is increasing with a higher time offset.

3.3.1 Implementation

On January 2019, the authors have released an improved version of VINS-Mono called VINS-Fusion [26]. This version comes with improved functionality. It brings the possibility to choose from different sensor suites (stereo camera/monocular camera, using IMU or not) and to swap them if necessary easily. According to the used suite, the algorithm fuses the appropriate sensors. Despite these changes, the system still uses the global pose graph optimization, relocalization as well as loop closure. Additionally, authors have also provided global data fusion [59]. It assumes local accuracy of VIO and allows to fuse data from other sensors such as GPS, barometer, etc. to achieve a global scale driftless precision.

Authors have published their code as open-source[4] along with example calibration and configurations for various sensor suites. Calibrations are provided for EuRoC [15] suite, KITTI dataset [60] suite and even Realsense D435i model. The implementation also supports omnidirectional or fisheye cameras, which is convenient for monocular setup thanks to the fact that large FoV prevents losing tracked features during significant inter-frame motion. The code contains a calibration utility to obtain the intrinsic and extrinsic calibration for all supported models. However, the Kalibr calibration outputs, detailed in Chapter 2.2.1, were used for the evaluation of the algorithm. The algorithm publishes the message in ROS *Odometry* format. VINS-Fusion algorithm can be tuned for following parameters

1. The feature tracking parameters such as the number of features, the minimal distance between features and RANSAC threshold and optimization parameters for Ceres solver [53] as max time, max iteration and keyframe parallax threshold
2. On/Off online temporal calibration of the time offset between IMU and camera and on/off the online camera-IMU extrinsic calibration

[4]`https://github.com/HKUST-Aerial-Robotics/VINS-Fusion`

Chapter 4

Practical evaluation

Practical experiments and comparison of VIO algorithms presented in this chapter are one of the key contributions of this work. Used evaluation metrics, as well as testing scenarios, are described. The algorithms were firstly tested in the Gazebo simulator to find well-performing configurations of the algorithms, and to solve any potential issues since experiments with actual hardware are much more demanding in terms of human and time resources. After testing the possible variants in the simulator, the tested were performed also on the real UAV platform. Let us first introduce the evaluation metrics in this thesis and also describe the testing scenarios.

4.1 Evaluation metrics

A critical aspect when assessing the performance of any localization technique is evaluating the drift, i.e., the deviation of estimated position from ground truth, over a period of time. The drift can be divided into translation and rotation components. However, the translation error comparison is sufficient because the rotation error is propagated into the translation error while moving. For the altitude, the current MRS system uses the rangefinder-barometer linear Kalman fusion to obtain the UAV altitude. The current UAV altitude estimation is sufficient. Hence the z-component of the position estimated by the VIO algorithm is not used for the control of the UAV. The following experiments focus primarily on the evaluation of lateral error, i.e., the x and y components of the UAV position.

The ATE (Absolute Trajectory Error)[61] metric is used as the primary evaluation metric. ATE compares the whole trajectory with the reference ground truth one. Thus it covers both the translation and rotation error. The evaluation requires precise ground truth values. In the simulation environment, the GPS is used, which is the exact UAV position with no error. Hence the comparison is exact. In real flights, the RTK GPS is used. More details regarding the real deployment and RTK GPS are in Chapter 4.8. The VIO odometry poses in time are defined as a series of points $\boldsymbol{P}_1 ... \boldsymbol{P}_n \in SE(3)$. Similarly, the ground truth poses are defined as points $\boldsymbol{Q}_1 ... \boldsymbol{Q}_n \in SE(3)$. Since VIO odometry and RTK are usually in the different inertial systems, the trajectories must be aligned first. The Horn method [62] is used to finds the rigid-body transformation $\boldsymbol{S}$. The transformation $\boldsymbol{S}$ is used to align the VIO trajectory $\boldsymbol{P}_i$ to the ground truth trajectory $\boldsymbol{Q}_i$ giving absolute trajectory error $\boldsymbol{F}_i$ at time step i as

$$\boldsymbol{F}_i := \boldsymbol{Q}_i^{-1}\boldsymbol{S}\boldsymbol{P}_i. \tag{4.1}$$

Then the RMSE error is given as

$$RMSE(\boldsymbol{F}_{1:n}) := \sqrt{\Big(\frac{1}{n}\sum_{i=1}^{n}||trans(\boldsymbol{F}_i)||^2\Big)}, \tag{4.2}$$

where *trans* symbolizes the translation error between same time indexes between the ground truth and the VIO poses. The result of Equation (4.2) is used if ATE is calculated. In this thesis, the ATE is always expressed in meters. The components of ATE calculations can be showed in Figure 4.1. The global trajectory consistency metric has been shown, but the VIO

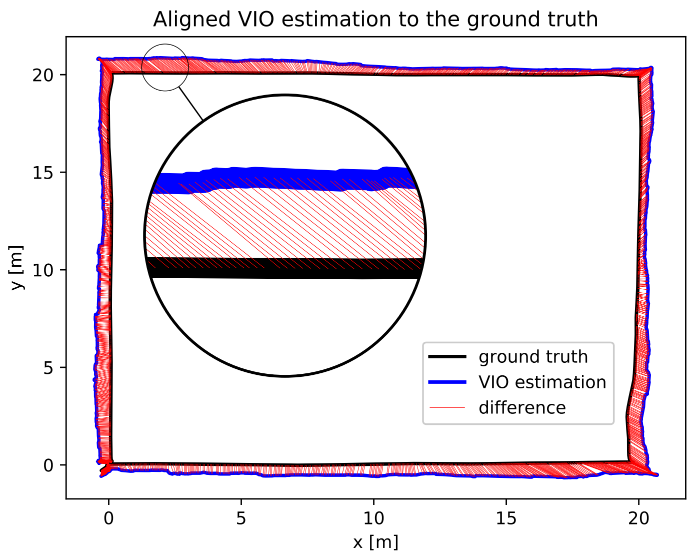


Figure 4.1: Figure shows the difference between aligned estimated odometry and the ground truth. Every red line represents the distance difference between the same time samples. The differences are then used in Equation (4.2).

algorithms might have a local drift in the set of IMU and camera measurements. This type of error is reflected in the relative pose error (RPE). The RPE takes two samples of the ground truth trajectory in the interval Δ and compares it with the same time samples from VIO trajectory. The error is calculated as

$$\boldsymbol{E}_i := \big(\boldsymbol{Q}_i^{-1}\boldsymbol{Q}_{i+\Delta}\big)^{-1}\big(\boldsymbol{P}_i^{-1}\boldsymbol{P}_{i+\Delta}\big). \tag{4.3}$$

Hence the total amount of sequences to compare are equal to $m = n - \Delta$ and again a RMSE is calculated over errors of all intervals as

$$RMSE(\boldsymbol{E}_{1:n}, \Delta) := \sqrt{\Big(\frac{1}{m}\sum_{i=1}^{m}||trans(\boldsymbol{E}_i)||^2\Big)}. \tag{4.4}$$

In this thesis, the RPE is always expressed in meters. The authors have implemented several types of the interval Δ. The Δ can be specified as the length in meters, as time duration in seconds, or as the number of frames (odometry messages). If not specified differently, this thesis uses the Δ as time in seconds. In such a way, the outcome is the root mean squared drift within one second. Another approach might be the translation error for every frame captured or the translation error within one meter of the ground truth. These variants give several possibilities on how to compare the VIO algorithms. The result of Equation (4.4) with $\Delta = 1\,\text{s}$ is used for RPE is caculations in this thesis. An example of translation error is shown in Figure 4.2.

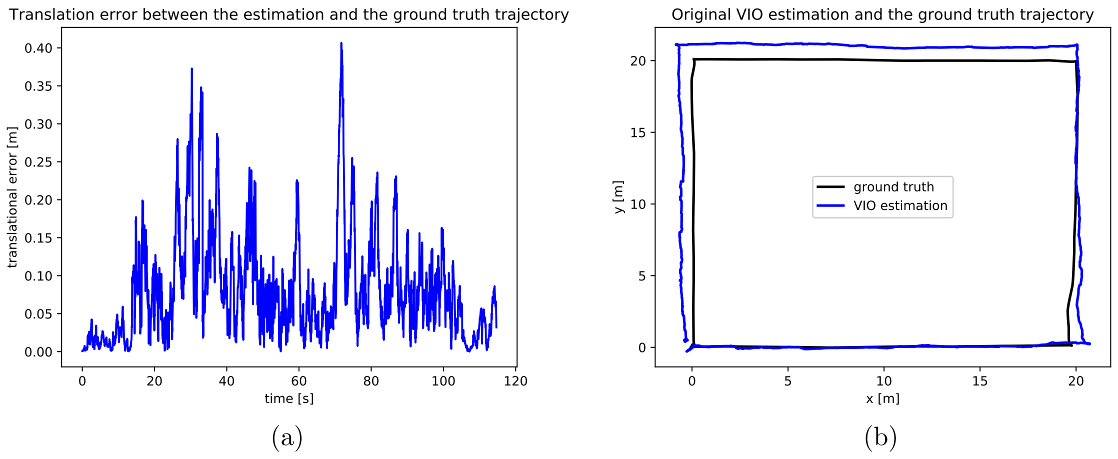


Figure 4.2: The figure (a) shows the translation error between the not aligned trajectories (b) used for RPE calculations. The trajectory error is calculated from the measured initially trajectories. The interval Δ between for RPE calculations is set to 1 second.

It is worth mentioning that both ATE and RPE are highly correlated, but each of them measures a different kind of error. Thus both of them are used for evaluation to highlight differences between the tested algorithms.

4.2 Trajectory planning

As mentioned in Chapter 4.6, the VIO algorithms are susceptible to fast movements and rapid changes in illumination, motion blur, etc. However, the primary purpose of this part is to improve the planned trajectory to suit the VIO more. Hence simple square trajectory planner was implemented assuming constant velocity along the trajectory. The UAV can have a different rotation in the yaw axis during the flight.

1. Forward orientation - The UAV can always face forward in the direction of the flight.
2. Constant orientation - The UAV always faces constant direction during the flight.
3. Center orientation - The UAV always faces in the center of the trajectory shape, specifically square. Nevertheless, this orientation is infeasible in another situation where the center of the flown trajectory is not easy to identify, or even the trajectory is not closed-loop.

The MRS system allows loading a predefined trajectory using a trajectory tracker. The trajectory has to be sampled with 5 Hz frequency, because the tracker reads the samples with this rate. The MPC tracker, mentioned in Chapter 1.0.2, forms a feasible control reference for the SO(3) controller so that it satisfies constraints imposed by the UAV dynamics.

The simple trajectory to follow is the constant velocity trajectory show in Figure 4.3. The constant velocity profile of this trajectory, shown in Figure 4.4, is problematic due to the infinite magnitude of acceleration while starting from zero initial velocity. The example has trajectory of total length $s_{total} = 12\,\mathrm{m}$ and constant velocity $v = 1\,\mathrm{m\,s^{-1}}$. The acceleration is not infinite at the beginning due to the sampling frequency. The velocity change between the first two samples is $\delta v = 1\,\mathrm{m\,s^{-1}}$ and the time to reach it is $t = 0.2\,\mathrm{s}$, hence acceleration between the first and second sample is $a = \frac{v}{t} = \frac{1}{0.2} = 5\,\mathrm{m\,s^{-2}}$. As shown in Figure 4.4, the acceleration constraints ($a_{max} = 0.4\,\mathrm{m\,s^{-2}}$) are violated. The figure shows the velocity and acceleration in each axis separately. The reason is that the points are sampled, assuming constant velocity. Thus the acceleration is violated just at the beginning before reaching the max velocity. In every corner of the trajectory, where the velocity is changed from x-axis to y-axis or vice versa, the acceleration reaches $\pm 5\,\mathrm{m\,s^{-2}}$. The MPC tracker prevents outputting a control reference that would violate the acceleration constraints. The accelerations from the constant velocity trajectory are bounded to the acceleration limit which results in a feasible control reference at the cost of precision of trajectory tracking, which can be seen in Figure 4.3, where the control reference cannot reach the corner of the square due to the acceleration constraints.

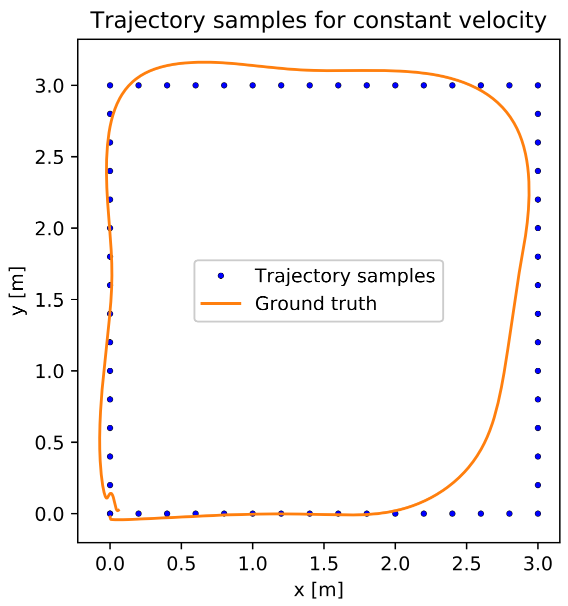


Figure 4.3: Constant velocity trajectory example showing violation of acceleration constraints during the UAV flight. The ground truth trajectory represents the flown trajectory with the MPC tracker. The trajectory following for MPC tracker is not precise, because the MPC prevents the control reference from violating the acceleration constraints. The example has trajectory of total length $s_{total} = 12\,\mathrm{m}$ and constant velocity $v = 1\,\mathrm{m\,s^{-1}}$.

The difference between the desired trajectory and the constrained control reference can, however, endanger success of some missions, where precise trajectory tracking is critical, such

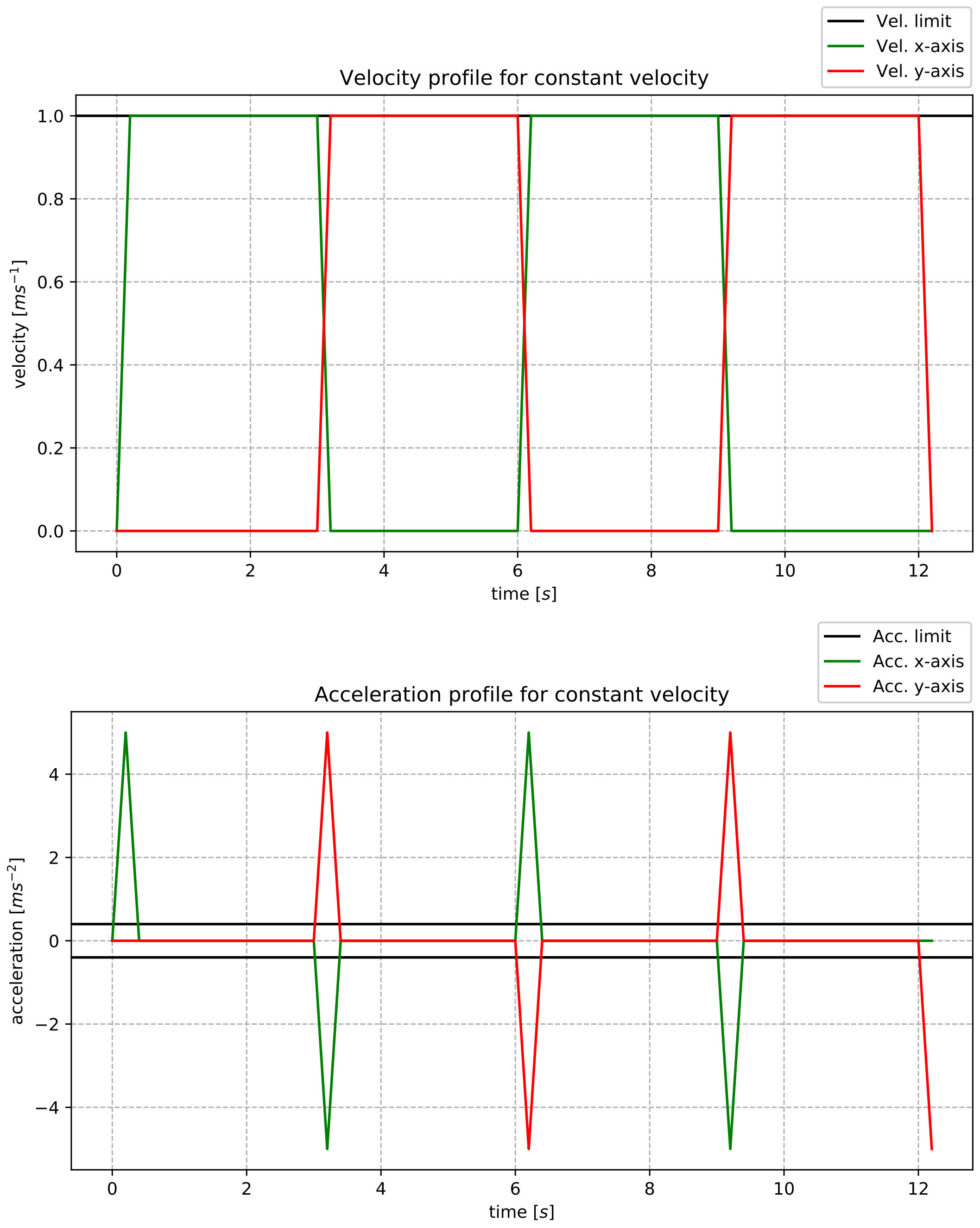


Figure 4.4: Velocity and acceleration profile for constant velocity ($v = 1\,\mathrm{m\,s^{-1}}$) assumption showing violating of acceleration constraints on edges of the square-shaped trajectory.

as flying through narrow passages, or multi-robot surveillance, such as [63], where deviation from the original trajectory could result in collision of agents.

4.3 Trajectory shaping

To keep the original trajectory shape, trajectory shaping, which modifies the velocity profile, has to be applied to constrain the acceleration. A trajectory shaping takes a trajectory with a constant velocity profile and limits the velocity in the critical points of the trajectory where acceleration limitation would be violated otherwise. As seen in Figure 4.4, the acceleration is exceeded three times, which is caused by changing the direction of the flight at the corners. That is why the corner trajectory points have to be resampled, specifically three samples at the corners, as shown in Figure 4.5. The yellow samples have to be added into the trajectory, which limits the acceleration to fulfill the acceleration constraint. The acceleration limit shown in left Figure 4.5 is equal to $a_{max} = 2\,\mathrm{m\,s^{-1}}$. The acceleration constraint is

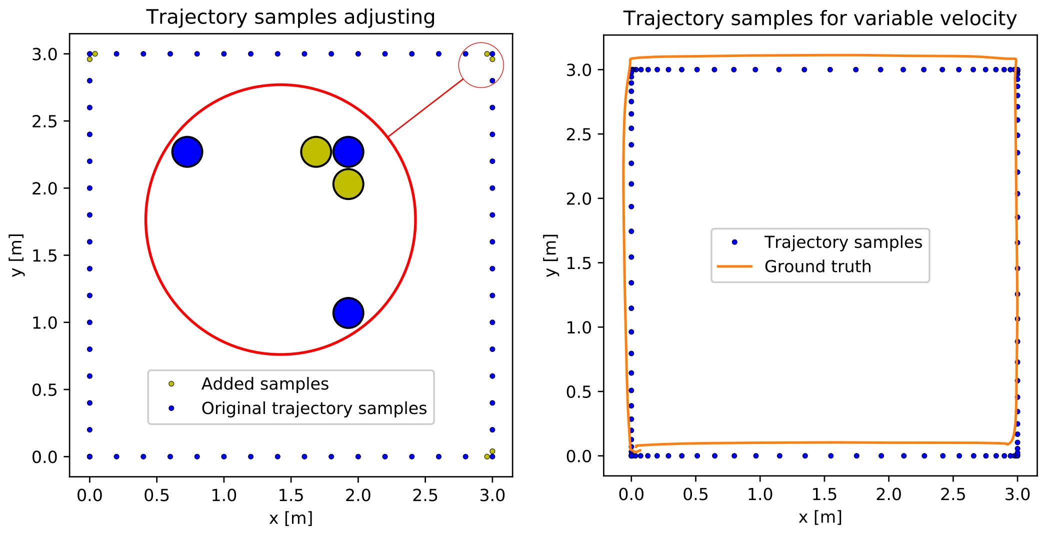


Figure 4.5: Example application of acceleration constraints on constant velocity trajectory with $v = 1\,\mathrm{m\,s^{-1}}$ UAV velocity. The figure on the left has acceleration constraints equal to $a_{max} = 2\,\mathrm{m\,s^{-1}}$. The samples that are added by the trajectory shaping algorithm to satisfy the acceleration constraints are depicted in yellow. The figure on the right is an example of a smoothed trajectory with acceleration constrained to $a_{max} = 0.4\,\mathrm{m\,s^{-1}}$ together with the ground truth trajectory representing the flown trajectory with the MPC tracker. The trajectory tracking his example is much more precise than in Figure 4.3 due to acceleration constraints application. The error in the right figure is caused by imprecise control of the UAV.

set in practice lower to about $a_{max} = 0.4\,\mathrm{m\,s^{-1}}$ as shown in right Figure 4.5. The resulting velocity and acceleration profiles of the smoothed trajectory in right Figure 4.6 are shown in Figure 4.6. The acceleration profile shows some limits exceeding, which is caused by imprecise trajectory resampling, which is negligible due to trajectory preprocessing by MPC tracker.

The main goal of the trajectory shaping was to limit the acceleration and adapt the trajectory according to the UAV dynamics. However, if the maximal acceleration is set too high, the instant change in acceleration to the maximal value is also not physically possible. A change of acceleration in time is defined as the jerk

$$j = \frac{da}{dt}. \tag{4.5}$$

Including the jerk into the trajectory planning is possible. However, the MPC tracker has its constraints on the jerk. Hence the trajectory acceleration is limited by the MPC tracker.

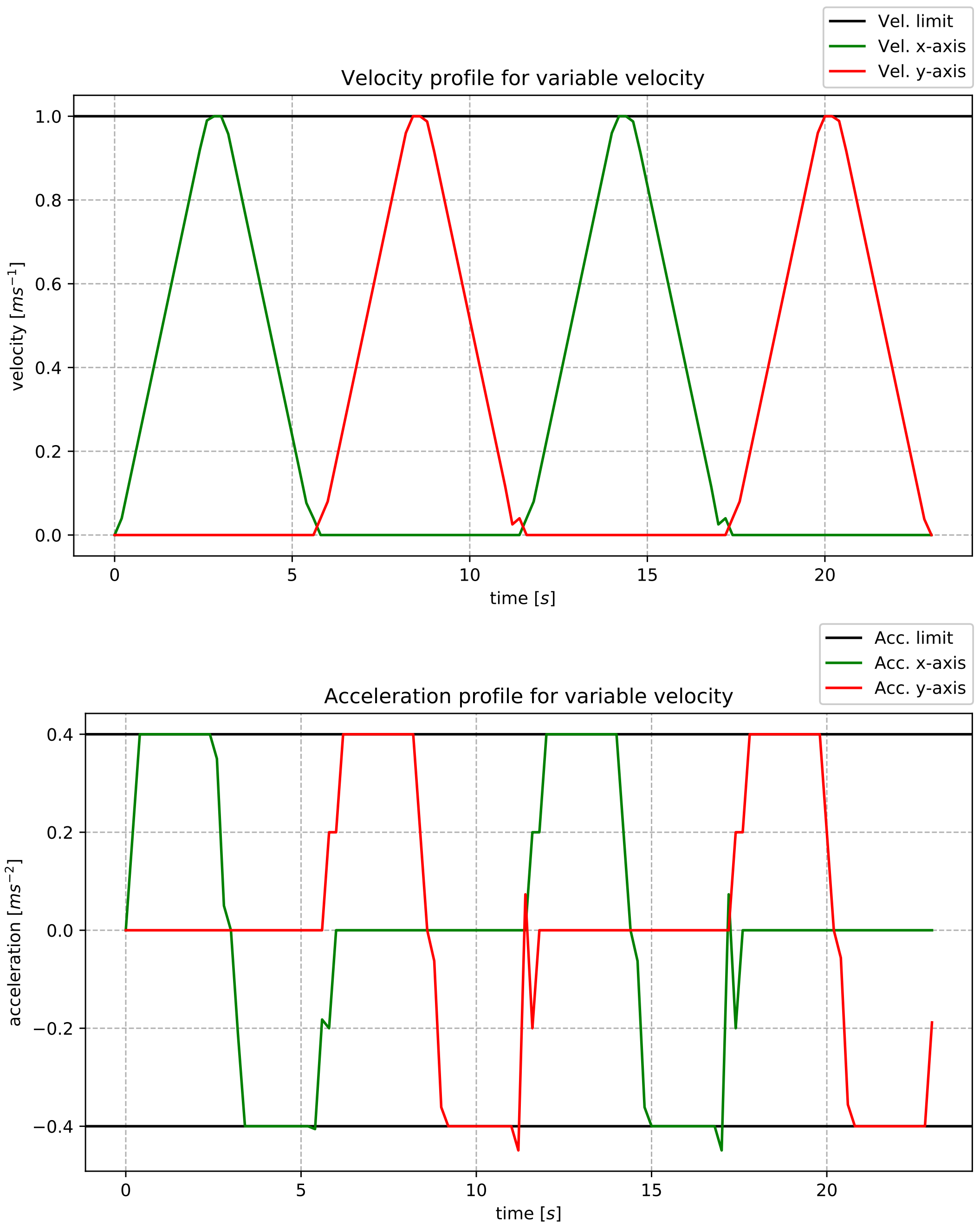


Figure 4.6: Variable velocity trajectory example after resampling the original constant velocity trajectory.

Furthermore, the MPC tracker also has constraints on acceleration and velocity. It is essential to set the MPC constraints according to the maximal velocity and maximal acceleration of the trajectory. The MPC then keeps the trajectory shape as it was planned. If the constraints for MPC tracker velocity or acceleration is lower than the trajectory assumes, the MPC shortcuts the trajectory, as shown in Figure 4.3.

4.4 Gazebo simulator

The UAV platform, along with the whole sensor suite, can be simulated in the realistic Gazebo simulator [64]. One of the simulated sensors is the RealSense device. It allows simulating the camera device with all the options that the real camera has. It has a color camera, depth camera, and stereo cameras. The only difference is the absence of the IMU unit directly in the RealSense camera. However, the IMU unit is already included in the used Pixhawk [65] flight controller.

The Pixhawk controller contains the ADIS16448 IMU model available from as a PX4 Gazebo simulator plugin[1]. The model allows parametrizing the IMU in terms of gyroscope and accelerometer noise density and random walk, which is detailed in Chapter 2.2.3. The default values of all parameters were used.

The RealSense camera parameters as camera resolution and frame rate are configurable via the Gazebo model configuration file. It also allows configuring the noise parameters to demonstrate further the real camera conditions as well as the minimal and maximal distance that the camera captures. Because simulating a lens distortion is a computationally expensive operation, which would, especially with multiple-camera setup, limit the real-time performance of the simulator. Therefore the simulated camera does not have any distortions of disturbances as the real camera does. Hence the usage of Kalibr software to obtain the camera parameters is unneeded. The intrinsic parameters are readable from the *camera_info* ROS message and the camera-IMU transformation with the build-in ROS command-line tool *tf_echo*. A slight noise was added to the recorded images to simulate the real condition of the D435i camera. Also, the maximal visible distance of the camera module was set to 80 m.

Despite these settings, proper initialization of the algorithms was not possible at first. The S-MSCKF algorithm covariance matrix values were rising quickly, resulting in enormous position error. The SVO was unable to initialize the features in the image, and VINS-Fusion poses were rapidly jumping. Since all algorithms are well tested on the proven EuRoC datasets [15], the issue had to be in the simulator. The idea was to import the calibration target from the Kalibr tool into the simulator and run the calibration process to obtain the camera parameters. The results showed that the camera baseline is not 5 cm horizontal, as set in the parameters, but 1 cm vertical. That means the camera model baseline is incorrect. Hence the algorithms cannot work properly. The image captured by the cameras of the stereo pair has been taken from the same position, so the algorithms could not estimate the depth of the features to initialize the localization process correctly.

After fixing the bug, the next Kalibr measurement showed the correct camera position as it should be. Unfortunately, the Kalibr cannot accurately measure the correct translation between cameras in the simulator. It was caused by measuring the calibration data during the flight. Hence there were too fast movements, degrading the calibration process. Therefore the system transformation, as mention above, can be used directly without any modifications.

The Gazebo simulator struggles with precise timing and timestamps in the published messages. Occasionally Gazebo publishes the same image message several times, and that can cause an error in the feature detection process. Also, the same problem is related to the IMU

[1] `https://github.com/PX4/sitl_gazebo`

messages. A message filter has to be applied to filter out the repetitive messages for both sensors.

The simulation requires an appropriate world that simulates the real conditions for the UAV flight. The simulation world consisting of grass plain texture was created, giving available features to track. However, the real-world does not have just a simple grass plain. Hence the world includes several trees forming a forest, which adds additional features above the horizon.

The UAV trajectory is planned in the center of the forest, where there is a space for the planned trajectory to fly. Detailed info regarding trajectory planning is in Chapter 4.2 and 4.3. The tested camera mounting orientations on the UAV in the simulation environment are shown in Figure 4.7.

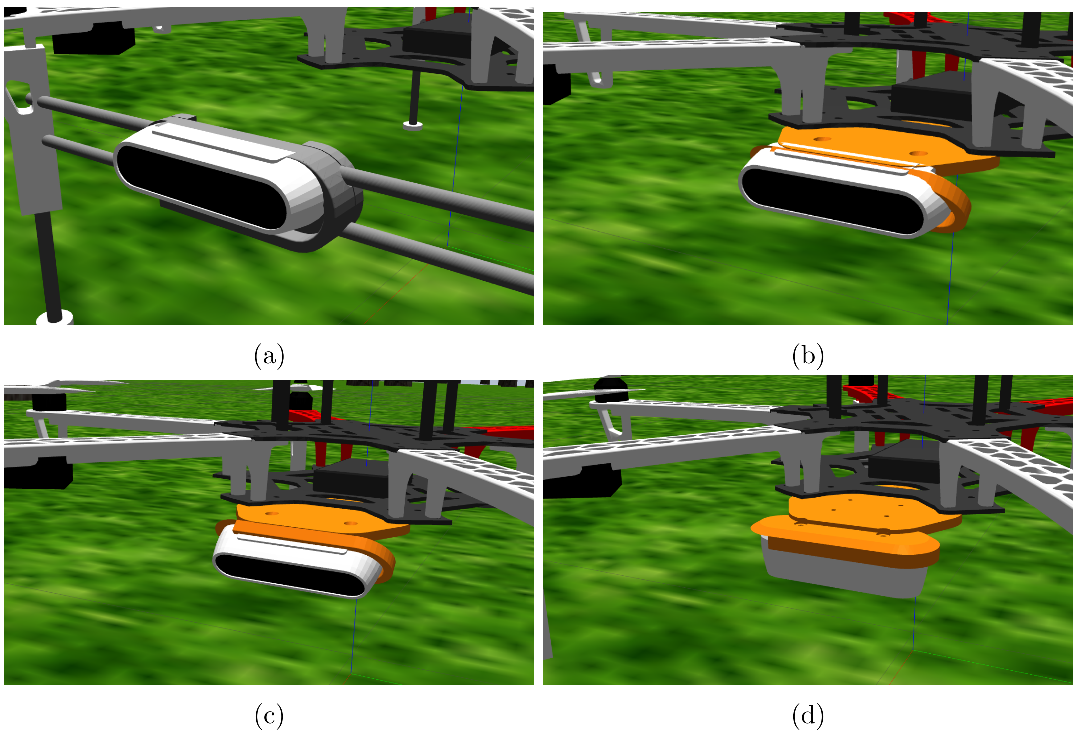


Figure 4.7: Images of simulation camera attachment on the UAV for front camera orientation (a), 10° pitched camera orientation (b), 30° pitched camera orientation (c) and down camera orientation (d).

All algorithms require specific changes in order to work well with the Gazebo environment. The common requirement for all algorithms is the unique calibration file for each of the four possible camera orientations.

4.4.1 S-MSCKF

The algorithm requires to start from the stationary position to properly initialize the EKF covariances. Hence it is preferred to launch the S-MSCKF right before the flight or in a stable flight position. The noise values, representing the standard deviations of IMU measurements and bias random walk, has to be changed in order to increase the filter robustness and stability. The IMU noise values are increased, and the feature noise is decreased w.r.t. the default values from S-MSCKF example files. The feature noise decrease is paired with an increase of feature amount in the image processing part.

4.4.2 SVO

SVO can initialize quickly before the flight, so it does not require to start from a stationary position. SVO has been proposed for forward and down camera orientations, hence the SVO parameters have to be modified while changing camera orientations. Otherwise, the higher number of features is kept to make sure the SVO is stable during the flight. As mentioned, the IMU is used only to adjust the feature observations. However, proper settings of the IMU priors are crucial to handle angular motions of the UAV.

4.4.3 VINS-Fusion

VINS-Fusion requires proper initialization at the beginning, similarly to S-MSCKF. VINS temporal calibration feature and extrinsic calibration estimation are switched off due to precise extrinsic calibration in the simulator and zero time offset between the camera images and IMU measurements. The default IMU parameters as the standard deviation of measurements and bias random walk for both algorithms have to be increased appropriately to the same values as used in S-MSCKF. The Ceres maximal solver time has to be adjusted for different scenarios, as described in Chapter 4.7.

4.5 Algorithm compatibility with MRS system

Before any further scenario description, the compatibility of algorithms with D435i and T265 camera has to be tested. A discussion of the capability of algorithms to be used with the MRS system, not just in the gazebo simulator follows.

The algorithms were tested on similar hardware setup, as mentioned in Chapter 2.3. The IMU on camera was set to the highest rate possible, which was 400 Hz for gyroscope and 250 Hz for the accelerometer. The highest possible IMU rate guarantees accurate movement detection in the flight.

All algorithms were compatible with the D435i camera. Unfortunately, the T265 camera was unable to be launched with the SVO algorithm. The feature matching from stereo fisheye cameras is not currently possible, as authors have confirmed. It is related to imprecise undistortion of fisheye images. The T265 camera has unpleasant limitations as missing the Gazebo simulator model, only static camera parameter settings, and unable to launch SVO in a stereo configuration. Due to these limitations, the T265 camera is not considered in the next experiments at all.

The first experiments on the UAV have been with Intel NUC PC Intel NUC 7i5BNK[2] which has four cores on a processor unit. At first, the compressed camera images and IMU

[2]`https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits/nuc7i5bnk.html`

data were recorded without the algorithms running. The test aim to verify if the camera message publishe rate and dataset recording work well together. Camera resolution was set to 1280x720 with a 30 FPS rate. The images were compressed, to achieve the smaller dataset size. Unfortunately, the image compression process consumes significant CPU resources, so the algorithms cannot be launched on the UAV with this configuration. Assuming real deployment of the UAV with the VIO algorithms running, the images will not be recorded. Hence it is sufficient to record just the algorithms published trajectory, thus lower the CPU load. It appeared that camera resolution has to be decreased significantly despite the turned off image recording to be able to guarantee real-time processing. The algorithms publish rate was about 5 Hz, which is not sufficient for feedback control. Next, the lower camera resolutions were tested, namely 848x480, 640x480 and 640x360. The position estimation publish rate has increased to the camera frame rate (30 FPS), but more importantly, the CPU load decreased. Due to the future presence of other running processes, the 640x360 resolution is used in the next experiments because of the lowest requirements in the image processing part. Another improvement came with mounting the newest model Intel NUC Kit 8i7BEH on the UAV platform. It has the i7 processor with four physical cores and eight with hyperthreading, which almost doubled the previous configuration and thus increased the CPU resources and performance. After these changes, the CPU load on all cores has decreased to 30 %. Hence the algorithms perform well with the rest of the flight essential software.

4.6 Testing scenarios

The algorithms are compared not just in metric precision using evaluation metrics introduced in Chapter 4.1 but also in suitability for feedback control of the MRS system. The good results, according to the metric, does not imply better feedback control. According to brief analysis in Chapter 4.5 only 630x360 resolution is used.

Next, the camera attachment position to the drone is also essential. Camera orientations can be forward-looking, down looking, and pitched camera under different angles. Usually, the forward orientation is assumed for most of the algorithms. The great advantage of the forward orientation is always available features in the straight movement. However, it might suffer from rapid illumination changes, especially in the outdoor environment in the sunny weather. Besides that, fast rotation in the yaw angle might decrease the algorithm precision and increase the drift. Furthermore, the landscape horizon is visible in the middle of the image. Assuming the forests and hilly regions, about 1/3 of the image contains sky. Features detected in the sky are very unreliable and unstable, hence detecting these features should be avoided. This is why the pitched camera might become handy. The pitched camera successfully avoids the sky features. Hence it observes more ground features and becomes more stable in terms of feature detection. Also, during fast rotations, more ground features are successfully tracked between frames than with the forward-looking camera. Moreover, pitching the camera offers the option of using the down-looking camera. The down-looking approach gives a considerable advantage in stable feature detection during various weather/lighting conditions, and the fast rotation becomes no problem for feature tracking. However, the fast movements in low altitude are challenging due to rapid scene changes, making the feature detection and matching more difficult. Also, fast accelerations and decelerations of the drone make the UAV pitch angle unstable, and then the feature detection is even more complicated. Furthermore, taking off is a challenge for the down-looking camera due to features being too close, but if the UAV is taking off from the feature-rich ground, this problem is not significant. To summarize, the camera positions have several variants, while each has its advantage and disadvantages. All these aspects are important to consider while choosing the most suitable solution for the environment. In this thesis four camera orientations are compared: forward orientation (indicated as 0°), 10 degrees pitch (indicated as 10°), 30 degrees pitch (indicated as 30°) and downlooking orientation (indicated as 90°).

Another important aspect is the frame rate. Generally, the higher the frame rate, the higher the computational time. However, the higher rate might bring the advantage of slower feature movement between frames, which can be used mainly during fast movements. Again, the optimal trade-off has to be found. Three frame rates are considered, given by Table 2.1 specifically 30, 60 and 90 frame per second.

The UAV velocity has been mentioned as an essential aspect of VIO robustness in this chapter. Hence several UAV velocities are used to generate the UAV trajectory, specifically $1\,\mathrm{m\,s^{-1}}$, $2\,\mathrm{m\,s^{-1}}$ and $5\,\mathrm{m\,s^{-1}}$. The shape of the trajectory is a square with a total length of 80 m. While none of the algorithms have loop-closure in odometry (VINS-Mono, mentioned in Chapter 3.3 publishes loop-closure odometry in the different topic than original one), there is no need to design trajectories with multiple loop-closing possibilities.

4.7 Simulator experiments

During some of the experiments, algorithms cannot properly estimate the position from available data resulting in pose estimation jump or divergence of the algorithm. Any of the used algorithms cannot correctly recover from such a state, and they have to be restarted/reinitialized. In a real deployment, these situations have to be minimized. These experiments are treated as outliers and test marked as failed and not included in algorithm evaluation.

The simulation tests are divided into four parts. In the first part, the 640x360 resolution with 30 FPS is set, and all algorithms are tested with all combinations of velocity and camera orientations. The reason is lower data bandwidth and lower processing time than higher FPS rates and resolutions. The second part follows up with experiments in higher altitude for 90° camera orientation. In the third part, the lower UAV altitude is analyzes along with higher FPS rates. Lastly, the algorithm performance is verified in feedback control on the simulation UAV.

The simulation experiments are using a gazebo simulation GPS for the feedback control of the UAV control system. The gazebo simulator generates the ground truth used, hence it is the exact UAV position with no disturbances.

During the first simulation experiments, no relation between UAV orientation during the flight and VIO precision has been found. To assume most generality, the forward orientation is used in further simulation experiments. It brings the most options in the various scenarios with the UAV, where specific UAV orientation is desired.

4.7.1 Camera orientation test

Because the EuRoC dataset [15] uses the 752x480 camera resolution, the 640x360 camera resolution is used as similar resolution size. The camera plugin can simulate any resolution, but the limitation is the D435i camera, which has limited resolutions, as shown in Table 2.1. Next, the camera frame rate is set to 30 FPS. This combination of camera resolution (640x360) and frame rate (30 FPS) is presumed to be the stable and well working one. The camera resolution is reasonable in terms of computational time and a sufficient amount of features. The frame rate should be satisfactory for the velocity of planned scenarios. As mentioned in Chapter 1.0.2, the altitude of the UAV is determined by fusion of the laser rangefinder and the barometer sensor, hence the altitude is constant and equal to $z = 3\,\mathrm{m}$. Table 4.1 summarizes the algorithms results.

Camera	**UAV velocity**	**S-MSCKF**		**SVO**		**VINS-Fusion**	
Orientation	$[\mathrm{m\,s^{-1}}]$	ATE	RPE	ATE	RPE	ATE	RPE
0°	1	0.3946	0.0981	**0.1932**	0.0718	**0.2625**	0.0473
0°	2	**0.2747**	0.0959	0.4987	0.1007	0.4770	0.0561
0°	5	0.4089	0.1208	0.8866	0.1433	0.4306	0.1415
10°	1	**0.2816**	0.0758	**0.2821**	0.1837	0.2274^{*}	0.0329
10°	2	0.2678^{*}	0.0777	0.5334	0.2534	0.3625	0.0440
10°	5	0.3754	0.1019	0.5809	0.2897	**0.3166**	0.0833
30°	1	**0.1825**	0.0417	0.4475	0.0469	0.0737^{*}_{4}	0.0577
30°	2	0.2798	0.0683	**0.4280**	0.0648	0.1185^{*}	0.0681
30°	5	0.6106_{4}	0.0844	0.4375	0.0996	0.117^{*}_{3}	0.1384
90°	1	0.3195^{**}_{3}	0.0784	0.2566	0.0862	0.0475^{*}_{2}	0.0387
90°	2	0.6068^{**}_{4}	0.1174	0.4378	0.1114	0.0764^{*}_{4}	0.0534
90°	5	1.4281^{**}_{2}	0.1974	2.2822	0.0912	9.2172^{*}_{1}	0.6869

Table 4.1: Each scenario has been repeated 5 times on the same dataset to check the robustness and precision. The value in the table is the mean from these five trials. The best scenario result accomplished for all trials for each algorithm, and every camera orientation is in bold. The number $()_n, n \in \{1, .., 5\}$, if present, indicates the count of successful test repetitions giving the calculated ATE. If not present, all repetitions were successful. The tests marked with $()^{*}$ required initial dataset time postpone. The tests marked with $()^{**}$ required both initial dataset start time postpone and earlier end of the dataset.

Regarding the velocity, the results demonstrated that most feasible is the scenario with UAV velocity equal to $1\,\mathrm{m\,s^{-1}}$ for all algorithms, because of the UAV stability and lack of rapid UAV movements during the flight. Generally, the higher the velocity, the lower robustness, and stability. It is caused by the increase of rapid movements, faster scene changes resulting in longer image processing time, and generally more challenging optimization and filtering of position.

Regarding the camera orientation, the down-facing camera orientation had the worst results. One of the reasons is the closeness of the grass texture during takeoff. The used grass texture might not have sufficient resolution, and the proximity of the camera challenges the image processing for all algorithms. The time was postponed to approximately the 15-th second of the dataset, where the UAV reached the maximal altitude. The second reason is the low altitude of the flight. During the fast velocity movement, the features were changing too fast to handle it.

Generally, S-MSCKF requires initial stable position to initialize accelerometer bias, gyroscope bias and UAV orientation, however some scenarios cannot fully guarantee stable initial position such as 90° camera orientation scenarios. The reason is the feature absence at the beginning and at the end of the flight, which required the modification of start and end time of the 90° scenario dataset. The 10° camera orientation and the UAV $2\,\mathrm{m\,s^{-1}}$ velocity scenario required only slight time adjustment. The S-MSCKF does not perform well during a longer duration of hovering or stationary pose, resulting in an unstable filter state and an increase in the filter covariance. Figure 4.8 shows the example of very precise estimation of S-MSCKF algorithm for 30° camera orientation with $1\,\mathrm{m\,s^{-1}}$ UAV velocity.

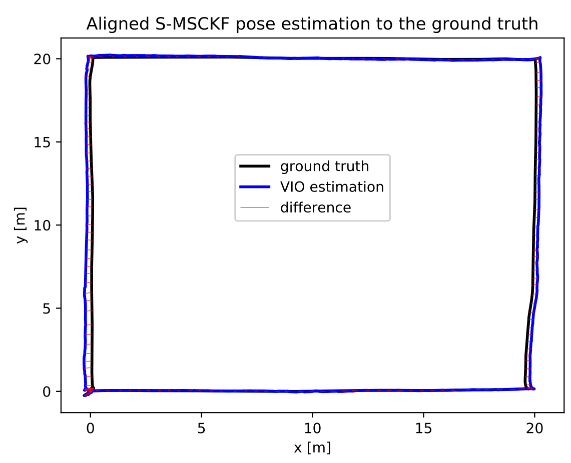
The SVO algorithm proved to be the most robust in terms of repeatability. It has accomplished all the scenarios and the repetitions without any failed trial and any time adjustment of the dataset. The SVO does not initialize until successfully triangulating the features from the image. This is the reason why 90° camera works because the initialization starts in the air, and SVO does not require IMU initialization. However, the precision of the SVO is not the best one in most of the scenarios. This might be induced by insufficient tuning of the algorithm, although the ATE value is still pretty impressive on the global scale.

The VINS-Fusion algorithm outperformed the other tested algorithms in most of the tests in terms of precision. Nevertheless, the algorithm encountered issues with robustness, especially with a 90° camera in the high velocity. The problem is caused by low altitude and fast feature changes, especially during the higher UAV velocity scenarios. Furthermore, the Ceres solver [53] maximal optimization time parameter raises the processing time of the incoming messages. It was necessary to decrease the maximal optimization time so that the algorithm can run in the realtime, but the robustness of the solution might be lower then. Further, the initialization process is crucial for the VINS-Fusion, as mentioned in Chapter 3.3, which becomes challenging for scenarios as denoted by $()^*$. Thus the time adjustment increased the repeatability. Figure 4.9 shows an example of the failed pose estimation trial for 90° camera orientation with $1\,\mathrm{m\,s^{-1}}$ UAV velocity resulting in big ATE and RPE error.

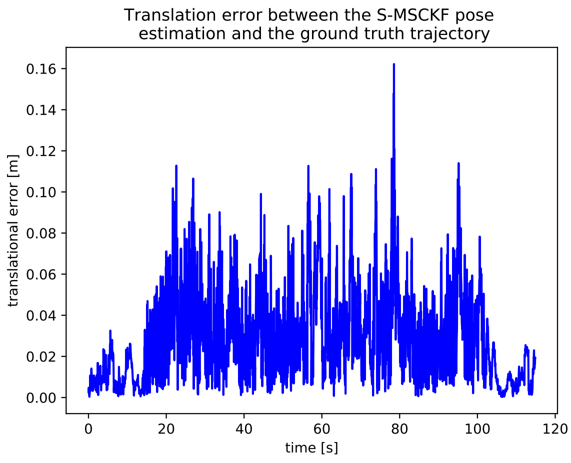
The camera orientation results in Table 4.1 shows that all camera orientation scenarios tested are comparable except the 90° camera orientation, which performed successfully for all trials only in lower velocities for SVO. However, that assumption is misleading, because the altitude of the UAV during the flight was too low for 90° camera orientation, which is shown in Chapter 4.7.2.

4.7.2 High altitude test

In previous test, the constant altitude of the UAV was presumed for all scenarios. However, the 90° camera orientation appeared to be inappropriate for low altitudes and higher velocities. In this test, the higher altitude for 90° camera orientation is tested. The resolution and frame rate is the same for all tests, which is 640x360 and 30 FPS, respectively. The other camera orientations are not tested, because the higher altitudes do not offer enough feature possibility for these orientations in the simulation world. The test summary is shown in Table 4.2.

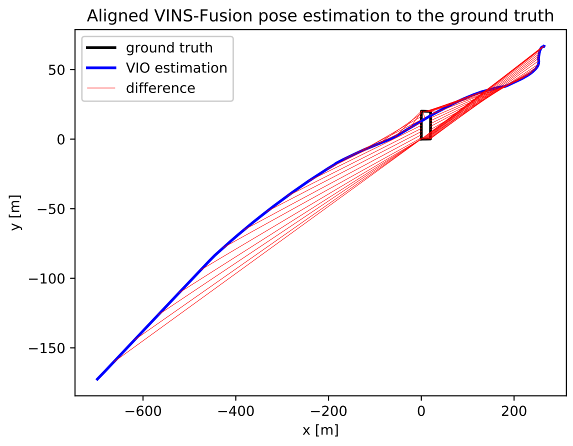


(a)

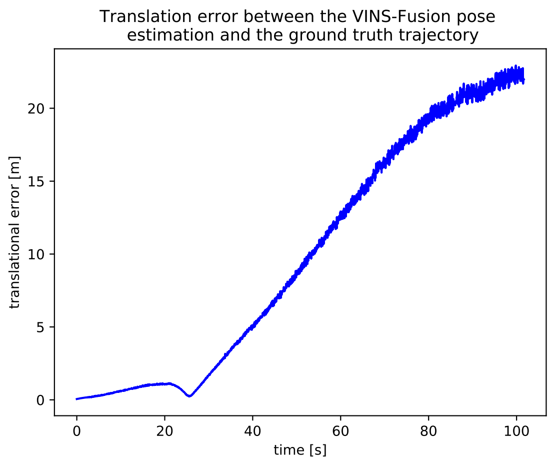


(b)

Figure 4.8: Figure shows results of one of the successful trial for scenario with 30° camera orientation and $1\,\mathrm{m\,s^{-1}}$ UAV velocity set. Aligned S-MSCKF pose estimation trajectory to the ground truth is shown in (a) giving ATE = 0.1926 m. The difference (red line) is shown only for every 10-th sample for clarity. The trajectory error (b) is calculated from the originally measured trajectories (not aligned) and give RPE = 0.0341 m.



(a)



(b)

Figure 4.9: Figure shows results of one of the failed trial for scenario with 90° camera orientation and $1\,\mathrm{m\,s^{-1}}$ UAV velocity set. Aligned VINS-Fusion pose estimation trajectory to the ground truth is shown in (a) giving ATE = 289.7062 m. The difference (red line) is shown only for every 40-th sample for clarity. The trajectory error (b) is calculated from the original measured trajectories (not aligned) giving RPE= 13.0215 m.

Both S-MSCKF and VINS-Fusion algorithms required dataset start and end time adjustment for the same reasons, as mentioned in the previous test in Chapter 4.7.1. However, the higher altitude improved both algorithms' robustness. The reason is more extended feature "life" even during the faster velocities. Same as in Chapter 4.7.1, the VINS-Fusion required a decrease in the maximal Ceres solver time to track the UAV position in real-time.

The simulator world comprises trees of about 6 meters high. The tests with UAV altitude equal to 6 meters are about that altitude, so the camera observes just the grass. However, the 10 meters UAV altitude leads to the observation of the top of the trees, which might

UAV Altitude [m]	**UAV velocity** [$\mathrm{m\,s^{-1}}$]	**S-MSCKF** ATE	**S-MSCKF** RPE	**SVO** ATE	**SVO** RPE	**VINS-Fusion** ATE	**VINS-Fusion** RPE
3	2	0.6068_4	0.1174	**0.4378**	0.1114	0.0764_4	0.0534
3	5	1.4281_2	0.1974	2.2822	0.0912	9.2172_1	0.6869
6	2	**0.1073**	0.0737	**0.3987**	0.2342	0.1012	0.0979
6	5	0.2134_3	0.1670	0.5239	0.2717	**0.0852**	0.0942
10	2	0.3422	0.1684	**3.8580**	0.6880	0.1753_4	0.1134
10	5	**0.2480**	0.1915	10.5773_3	1.3361	0.2315_3	0.2055

Table 4.2: Each scenario has been repeated 5 times on the same dataset to check the robustness and precision. The value in the table is the mean from these five trials. The best scenario accomplished for all trials for each algorithm, and every altitude is in bold. The number $()_n, n \in \{1, .., 5\}$, if present, indicates the count of successful test repetitions giving the calculated ATE. If not present, all repetitions were successful.

be misleading for the algorithm image processing. The fast movements of close distance tree features are not suitable, especially for the SVO algorithm, which fails in the 10-meter altitude.

To summarize, the higher the altitude, the higher the velocity that can be achieved. Nevertheless, the limitations are fast-changing features in front of the camera, which influence the maximal reachable altitude. Hence the best results were achieved for all algorithms in the 6 m altitude.

4.7.3 High FPS rate

As shown in the previous test in Chapter 4.7.1, the 30 FPS camera rate seems to be sufficient to handle even the high velocities. However, the 90° camera orientation struggled with the rapid feature changes during the low altitude flight. In this test, the higher FPS rates are tested for 90° camera orientation to improve the feature tracking precision potentially. The camera resolution used is 640x360. The UAV altitude is kept on 3 m. Table 4.3 summarizes the results.

The 1 $\mathrm{m\,s^{-1}}$ and 2 $\mathrm{m\,s^{-1}}$ UAV velocities are tested with higher frame rate just for clarity. Both velocities are already low enough to be tracked with a 30 FPS camera rate.

The SVO results proved that the authors note regarding the higher camera frame rate. The higher camera frame rate reduces the computational cost per frame, which is useful especially in higher velocities as seen in Table 4.3 for scenarios with 5 $\mathrm{m\,s^{-1}}$ UAV velocity. The higher FPS rates improved the precision of the SVO result.

Both S-MSCKF and VINS-Fusion results have not benefited from the higher frame rate in this test The higher frame rate did not improve the ability of feature tracking in all tested velocities. Additionally, the VINS-Fusion cannot process the higher frame rate in the realtime despite the parameter tuning. The tuning of calculation time results in an unstable position estimate.

To summarize, the VINS-Fusion is not robust with higher frame rates. The SVO approved to be stable and well-working with high frame rates. The S-MSCKF algorithm also struggles with fast feature changes, and the higher frame rate just degrades the filtering process.

Camera FPS	UAV velocity $[\mathrm{m\,s^{-1}}]$	S-MSCKF		SVO		VINS-Fusion	
		ATE	RPE	ATE	RPE	ATE	RPE
30	1	0.3195_3	0.0784	**0.2566**	0.0862	0.0475_2	0.0387
30	2	0.6068_4	0.1174	0.4378	0.1114	0.0764_4	0.0534
30	5	1.4281_2	0.1974	2.2822	0.0912	9.2172_1	0.6869
60	1	0.5986_3	0.1147	**0.1843**	0.0429	0.0355^*_4	0.0233
60	2	0.3758_4	0.2837	0.2579	0.0645	0.0294^*_2	0.0134
60	5	0.3141_1	0.1534	0.4384	0.1048	0.0567^*	0.0712
90	5	1.0513_1	0.9427	**0.4095**	0.0912	x	x

Table 4.3: Each scenario has been repeated 5 times on the same dataset to check the robustness and precision. The value in the table is the mean from these five trials. The best scenario accomplished for all trials for each algorithm, and every FPS rate is in bold. The number $()_n, n \in \{1,..,5\}$, if present, indicates the count of successful test repetitions giving the calculated ATE. If not present, all repetitions were successful. The (x) symbol means none of the tests were successful. The $()^*$ symbol means that VINS-Fusion did not publish the odometry message in the realtime.

4.7.4 Feedback control

In this test, the VIO is tested in the feedback of the UAV control system as described in Chapter 1.0.2. The output of each algorithm is transformed as mentioned in Chapter 3.1.1, 3.2.1 and 3.3.1 so that VIO positions are in the same frame of reference as the ground truth. S-MSCKF and VINS-Fusion are directly publishing the odometry message, also containing velocity in addition to the position and orientation. SVO publishes just the position and orientation, so the velocity has to be obtained by fusion of the VIO position output with the accelerations in the UAV state estimation module in Figure 1.3.

Each of the algorithms publishes the position in a different frame. Hence the corresponding transformation has to be taken into account to have the message in the *local_origin* frame. Previously, the algorithms could be tested on the same dataset. To test the feedback control, every run might be slightly different. Hence the result is not as precise as before.

As the simulation test showed, there is not a vastly different result between the 0° camera orientation and 10° or 30° camera orientations. The reason is probably the constant light conditions in the simulator, which cannot precisely simulate the sudden light changes while rotating due to fast exposure changes. In other words, the simulation camera is a precise model of the real camera giving too good results. This is why only 0° and 90° orientations are tested in the feedback for simulation. Also, the fast frame rate did not perform well with the S-MSCKF and VINS-Fusion algorithms. Hence only 30 FPS rate is used. The SVO has proved its robustness in a higher FPS rate, so 60 FPS rate is used for it. Furthermore, the altitude is set to 5 m for the 90° camera orientation and 3 m for the 0° camera orientation. The reason for the lower altitude is the most stable flight than in higher altitudes, which has been noticed while simulating the feedback control. The results are summarized in Table 4.4.

All algorithms required specific changes in the control system of the UAV. However, the changes are also related to the camera orientation. The 0° camera orientation worked well with the default settings of the SO(3) controller for the UAV in the simulation. The

Camera Orientation	**UAV velocity** [$\mathrm{m\,s^{-1}}$]	**S-MSCKF** ATE	**S-MSCKF** RPE	**SVO** ATE	**SVO** RPE	**VINS-Fusion** ATE	**VINS-Fusion** RPE
0°	1	0.6473	0.0871	**0.6182**	0.0610	**0.3369**	0.0451
0°	2	**0.3202**	0.0967	0.7480	0.0781	0.4145	0.0944
0°	5	0.4518	0.1390	1.2714	0.0994	0.3961	0.0773
90°	1	**0.0647**	0.0504	0.3470	0.1041	0.0402	0.0294
90°	2	0.1185	0.0630	**0.2481**	0.0973	**0.0378**	0.0371
90°	5	0.2437	0.1426	0.3894	0.1131	0.1078	0.0631

Table 4.4: Each scenario has been repeated 3 times to check the algorithm robustness and precision. The value in the table is the mean from these three trials. The best scenario accomplished for all trials for each algorithm and every camera orientation is in bold.

state feedback constants k_p for the position, and k_v for velocity had to be decreased to reduce aggressive motions of the UAV, hence smoothing the final trajectory. This approach might bring a small control error into the system, but more importantly, if it increases the VIO stability and tracking precision for this orientation. The 0° camera orientation worked well with the default constants. Although during the high velocity ($5\,\mathrm{m\,s^{-1}}$) scenarios the tracking precision was slightly improved with lowering k_p and k_v.

Generally, the best precision was achieved with 90° camera orientation with lower velocities. Again, the UAV cannot take off with 90° camera orientation with the VIO in the feedback, due to lack of the features while staying on the ground. The UAV has to take off with another odometry sources, e.g., GPS. The VINS-Fusion achieved an almost driftless trajectory with 90° camera orientation, which places it as one of the top candidates for the real deployment. The biggest issue for VINS-Fusion is precise initialization. Even small movements of the UAV while hovering propagates error into the tracking precision. Also, VINS-Fusion might struggle with optimizing in real-time. Once VINS-Fusion computations are delayed several times in a row, it cannot recover successfully. This is the reason why it is better to fly with lower velocity. The S-MSCKF results show that it also performs well with the 90° camera orientation. Same as VINS-Fusion, the initialization is critical for S-MSCKF, but otherwise, there are no requirements to adjust the parameters for different UAV velocities. The SVO precision for 90° camera orientation is better than the results for 0° camera orientation. The reason is the feature tracking stability for 90° camera orientation, which is essential for SVO. The higher FPS rate guarantees better precision in higher velocity.

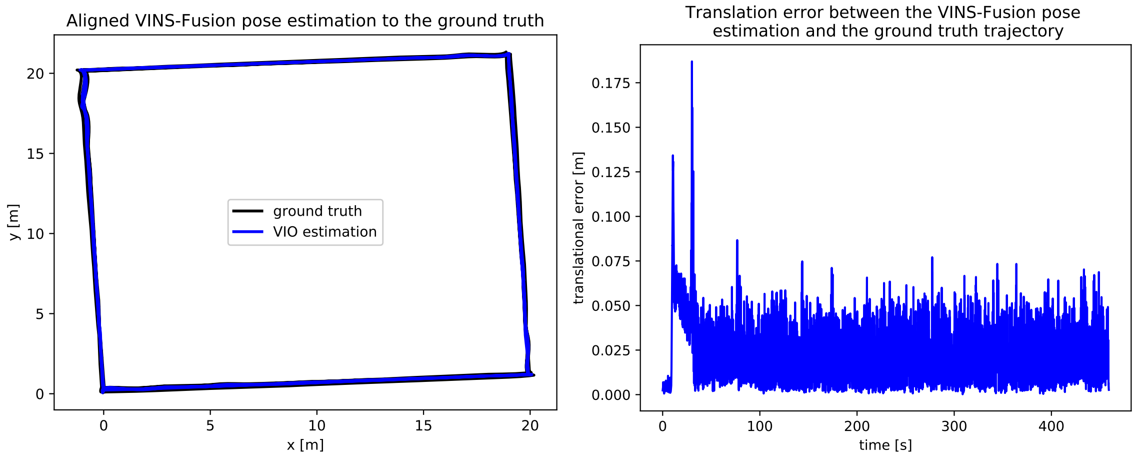
The 0° camera orientation results are slightly worse than the 90° camera orientation. The reason is the longer feature visibility for 90° camera orientation in fast straight movements and fast rotations at the trajectory corners. The S-MSCKF stability is satisfactory among all velocities for 0° camera orientation. The EKF based approach on the fusion of features and IMU improves the stability during rapid movement changes, such as acceleration and decelerations when features are changing faster. Similarly, the VINS-Fusion optimization-based equally on visible features, and IMU measurements assure the stability. Contrary, the SVO relies basically just on the features. The IMU measurements are just for motion corrections, hence just improves the result. During the rapid movement changes, the number of features might decrease rapidly, resulting in positional drift.

As mentioned in Chapter 4.1, ATE, and RPE values are highly correlated, which can be seen in the evaluations. Hence only ATE is shown further for clarity. The feedback control test, presented in Figure 4.5, shows an evaluation of precision during longer flights. Every scenario has been tested on the same square trajectory, but the trajectory was looped five times to ensure the consistency of trajectory following. The parameters are set for this test the same as for the feedback test shown in Table 4.4. For S-MSCKF and VINS-Fusion algorithms, the 30 FPS rate is set for all scenarios. For the SVO algorithm, the 60 FPS rate is set for all scenarios. Furthermore, the altitude is set to 5 m for the 90° camera orientation and 3 m for the 0° camera orientation.

Camera Orientation	**UAV velocity** $[\mathrm{m\,s^{-1}}]$	**S-MSCKF**	**SVO**	**VINS-Fusion**
0°	1	0.8458	**1.8530**	**0.4443**
0°	2	**0.6235**	2.7066	0.5608
0°	5	0.7597	2.7415	0.7083
90°	1	**0.1545**	0.4550	**0.0876**
90°	2	0.2274	**0.4441**	0.2655
90°	5	0.7278	1.1572	0.8544

Table 4.5: ATE results for 5 times looped square trajectory. The best scenario for each algorithm and every camera orientation is in bold.

The 90° camera orientation resulted in accurate trajectory tracking. Mainly S-MSCKF and VINS-Fusion algorithms have achieved low position error over the whole trajectory. The precision of VINS-Fusion algorithm is shown in Figure 4.10. SVO algorithm drifts over time, which then results in high ATE. Figure 4.11 shows the drift of SVO algorithm in time.



(a) (b)

Figure 4.10: Figure shows the result of 5 times looped square experiment for VINS-Fusion feedback control with 90° camera orientation and $1\,\mathrm{m\,s^{-1}}$ UAV velocity. Figure (a) shows the aligned VINS-Fusion trajectory to the ground truth giving ATE= 0.0876 m. For clarity purposes, the difference between the time samples is not displayed. The trajectory error (b) is calculated from the original measured trajectories (not aligned) giving RPE= 0.0277 m.

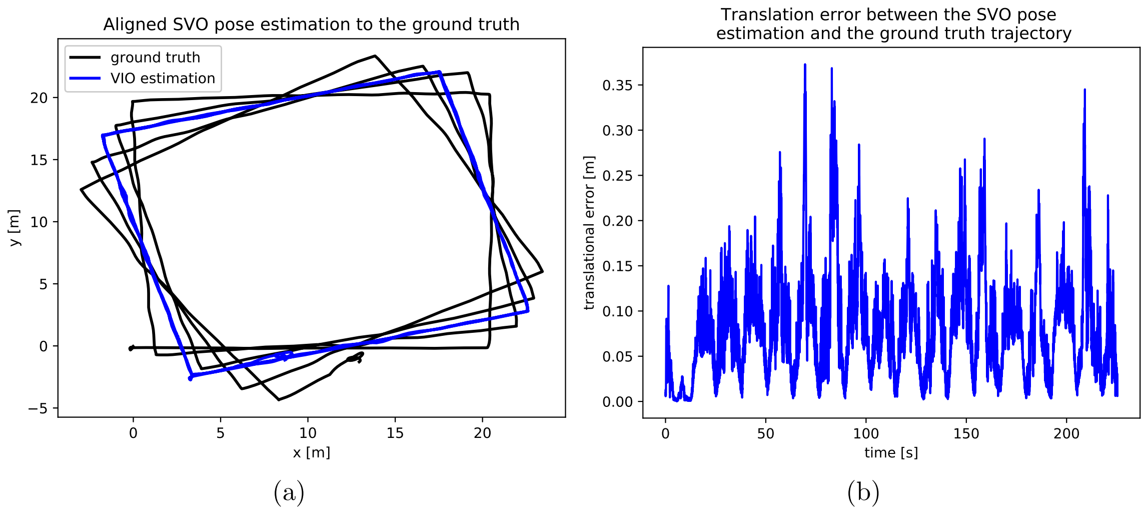


Figure 4.11: Figure shows the result of 5 times looped square experiment for SVO feedback control with 0° camera orientation and $2\,\mathrm{m\,s^{-1}}$ UAV velocity. Figure (a) shows the aligned VINS-Fusion trajectory to the ground truth giving ATE= 2.7066 m. For clarity purposes, the difference between the time samples is not displayed. The trajectory error (b) is calculated from the original measured trajectories (not aligned) giving RPE= 0.0951 m.

4.8 Real experiments

In this chapter, the acquisition of reference datasets is described as well as algorithm comparison on these datasets. Finally, the suitable algorithms are tested in the feedback control of the real UAV.

4.8.1 Dataset acquisition

The experiments were held during the middle of October 2019. At that time, the trajectory planning with acceleration constraints, as presented in Chapter 4.2, has not been implemented yet. The constant velocity trajectory limits the ability of the real UAV to smoothly follow the trajectory, hence only lower velocities are considered, specifically $0.5\,\mathrm{m\,s^{-1}}$, $1\,\mathrm{m\,s^{-1}}$ and $2\,\mathrm{m\,s^{-1}}$. The higher the velocity, the less precise trajectory following is due to the MPC tracker that tries to minimize the control error and control input while respecting constraints imposed on the UAV dynamics. When the planned trajectory is not feasible (i.e. does not respect the dynamics of the real UAV), the tracker can still generate a feasible reference for the controller. However, the trajectory shape might get modified and the trajectory following precision might suffer. As mentioned in Chapter 1.0.2, the world frame is essential for UAV control. The world frame is tied to the GPS coordinate system, and the origin of the world frame is set according to the GPS coordinates of the experiment area approximate center. This arrangement ensures to recognize the position when the UAV crosses the border of the experiment area and does not allow to set a reference for the controller, which would have the UAV outside of the experiment area. It is handy for the cases when a new feature is tested, so in case of the algorithm unpredicted behavior and remote controller signal loss the UAV can safely land. Also, it allows to visually compare the ground truth and VIO estimated position

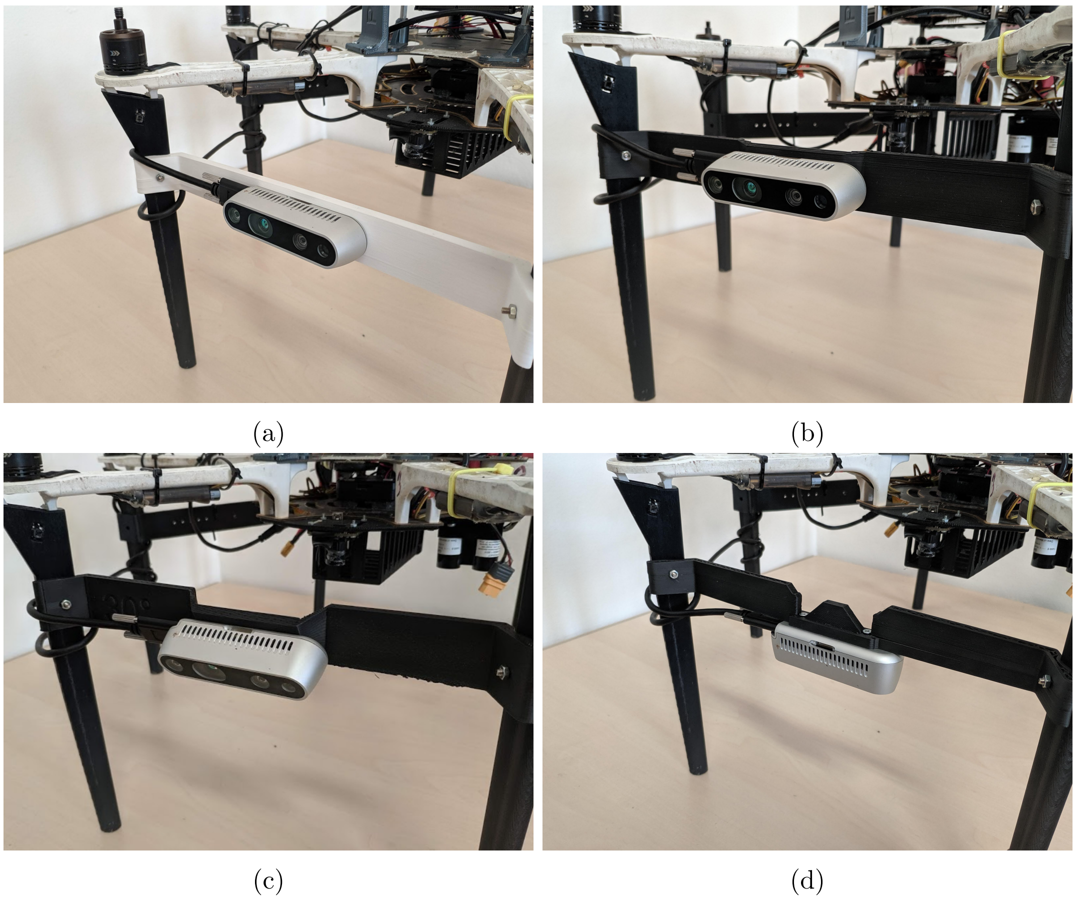


Figure 4.12: Images of D435i camera attachment on the UAV for 0° camera orientation (a), 10° camera orientation (b), 30° camera orientation (c) and 90° camera orientation (d).

in the RVIZ (3D visualization tool for ROS) during the flight for the VIO feedback control. In the dataset acquisition part, the RTK GPS is used for the feedback control of the UAV control system and also as the ground truth for comparison.

All datasets are recorded with a 30 FPS rate. The higher frame rate is impractical due to the growing size of the recorded datasets. As simulation experiments showed, the higher frame rate is not such an improvement in pose estimation precision. Only the SVO algorithm profits from the higher frame rate, but mostly in terms of processing time, which is not so crucial in the current UAV available computational load. The default altitude is set to 3 m for all datasets except the 90° camera orientation with the $1\,\mathrm{m\,s^{-1}}$ UAV velocity, where the altitude is set to 5 m to facilitate feature tracking during maneuvers involving aggressive tilting or high velocity.

During the real experiments, the sunlight significantly influences the camera. Hence the different UAV orientation during the flight, as mentioned in Chapter 4.2, becomes handy in feedback control with the real UAV, Chapter 4.8.3.

4.8.2 Dataset algorithms comparison

The main difference against the simulation environment is the presence of the IMU unit in the D435i camera. All algorithms presume three-camera frames: a camera frame, an IMU frame, and a body frame. The images captured from the camera are positioned in the camera frame. The IMU measurements are placed in the IMU frame. Finally, the estimated position is published in the body frame. However, VINS-Fusion and SVO assume the body frame the same as the IMU frame. That option worked pretty well for the simulation, but it would be more practical to set the transformation between the IMU frame and body frame arbitrarily. Next, for the pitched real camera, the estimated position is also pitched with the same angle and orientation as the camera is. Hence the transformation of the published position estimation w.r.t. UAV orientation has to be applied. The S-MSCKF algorithm allows setting the relation between the UAV (body) frame and the IMU frame. Hence the published message can represent the UAV position and orientation correctly. For SVO and VINS-Fusion presuming the body frame as the IMU frame, the correction is more complicated. The UAV position for SVO and VINS-Fusion is correct, but every message orientation has to be adjusted individually for the camera orientation angle, to match the UAV orientation.

Again, certain modifications have to be applied to the parameters of the algorithms. The IMU measurements are heavily influenced by the UAV propellers vibration during the flight. However, that is something that cannot be easily eliminated, so the algorithm parameters have to be adjusted. S-MSCKF algorithm noise parameter for the accelerometer had to be increased. Otherwise, the filter is more susceptible to diverge. VINS-Fusion algorithm needs as same as S-MSCKF to increase the noise parameters, but the best-found values minimizing the ATE are slightly higher than with S-MSCKF. Only accelerometer measurement noise has to be increased. VINS-Fusion assumes perfect calibration between the camera and the IMU. Unfortunately, the extrinsic calibration used does not work well without adjusting it first. The extrinsic calibration estimation has to be turned on together with the temporal calibration feature. SVO does not heavily depend on the IMU. Hence only minor adjustment had to be applied regarding IMU priors.

The results in Table 4.6 indicates the algorithm suitability for deployment on the real UAV. The overall best results were achieved with the S-MSCKF algorithm. It outperformed the other two algorithms in all tests, except the 30° camera orientation, which is explained later. The explanation for the good results is the well-balanced noise parameters for IMU and features. VINS-Fusion which was the best candidate from the simulator experiments, however, worked well in all trials except the 90° camera orientation. The only difference against the simulation environment is much better initial conditions for 90° camera. In the beginning, the camera is about 12 cm distant from the ground, which is sufficient enough to initiate the algorithms on the ground. Still, the proximity of all visible features seems to misalign the scale of VINS-Fusion at startup, giving the big ATE result. The postponed the start of the algorithm, compared to the simulation environment experiments, did not work well due to the strong propeller-induced vibrations. SVO performance is for the 90° camera orientation similarly overscaled because both VINS-Fusion and SVO depend more on the features than IMU. S-MSCKF feature noise can be set in the EKF. Hence it can handle the 90° camera orientation much better.

SVO more or less failed in all tests. The problem seems to be low quality of camera images. SVO can detect enough features. However, the trajectory scale is not correct.

Camera Orientation	UAV velocity [$\mathrm{m\,s^{-1}}$]	S-MSCKF	SVO	VINS-Fusion
0°	0.5	0.5533	**1.6090**	1.7885
0°	1	0.6878	2.0226	1.8431
0°	2	**0.3553**	2.5472	**0.6605**
10°	0.5	**0.2797**	2.7256	**0.4704**
10°	1	0.6952	**2.4049**	0.9081
30°	0.5	x	5.5443	0.5744
30°	1	x	4.8295	1.2863
30°	manual	**0.4175**	**3.9779**	**0.4983**
90°	0.5	0.7793	9.1213	3.0201_1
90°	1	**0.4793**	**6.0972**	5.3223_1

Table 4.6: ATE results (in meters) for 3 times repeated real UAV dataset. The number $()_n, n \in \{1,..,3\}$, if present, indicates the count of successful test repetitions giving the calculated ATE. If not present, all repetitions were successful. The x symbolizes none of the repetitions were successful. The manual UAV velocity symbolizes the remote reference position set for the UAV from the remote controller. The best scenario accomplished for all trials for each algorithm and every camera orientation is in bold.

The scale is different for every dataset, hence some further parameter adjustment might increase the performance. However, all attempts to improve the precision failed. The loosely couples IMU configuration cannot improve the estimation performance.

Suprisingly, the 30° camera orientation, specifically the $0.5\,\mathrm{m\,s^{-1}}$ and $1\,\mathrm{m\,s^{-1}}$ UAV velocities fails for S-MSCKF and decrease the performance for both SVO and VINS-Fusion. The D435i camera suffers from occasional flickering, especially under outdoor sunlight, when using the auto exposure mode. Unfortunately, due to inconsistent light conditions during the whole flight, the exposure cannot be set to a single value. This issue has already been reported and fixed in the latest production firmware release[3] for the D435i camera. It seemed that this error has not excessively influenced the algorithm estimation. However, in practice, this flickering can be frequent, especially in the outdoor environment, which can completely corrupt the feature detection from captured images. However, another issue showed up during the dataset recording. Sometimes, the D435i right infrared camera projects a pattern into the captured image. An example of such a projection is shown in Figure 4.13. It is the reason why S-MSCKF and partially VINS-Fusion performance is worse than with other camera configurations. S-MSCKF cannot correctly match the features between the cameras, which affects this algorithm. The visible and matched features for such an corrupted image for S-MSCKF algorithm is shown in Figure 4.14 VINS-Fusion feature detection has similar pattern as S-MSCKF has as shown in Figure 4.15. Nonetheless, the visible features are sufficient to estimate the position for the $0.5\,\mathrm{m\,s^{-1}}$ flight. The $1\,\mathrm{m\,s^{-1}}$ flight is slightly worse due to higher velocity. The pattern has also been spotted in some of the 0° camera orientation datasets. However, it did not influence performance significantly. Only a single square-shaped dataset for 30° camera orientation has not been impacted by the pattern. It is why the evaluation consists of manual flight. The manual flight proved the trend of S-MSCKF and VINS-Fusion providing a more robust estimation compared to SVO over all the datasets.

[3]`https://downloadmirror.intel.com/29255/eng/RealSense-D400-Series-Spec-Update.pdf`

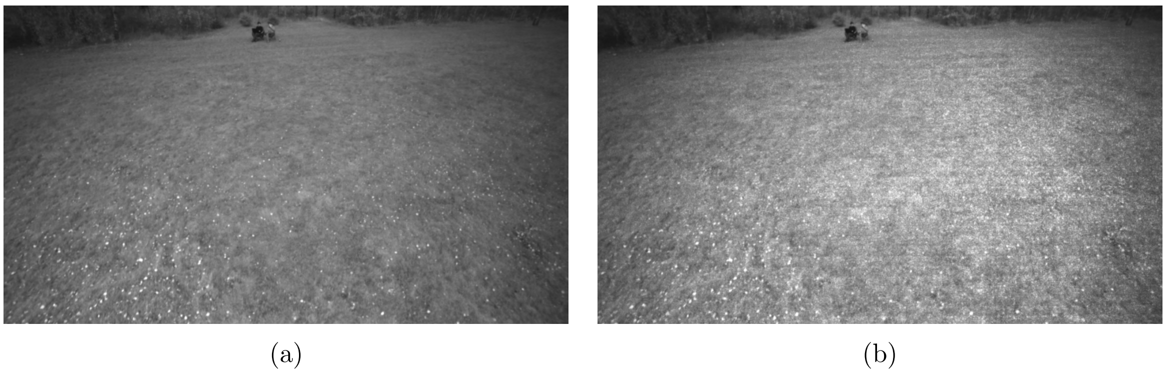


Figure 4.13: The image (a) is the correct image from the left infra camera. The image (b) is affected by the projected pattern creating a noisy image.

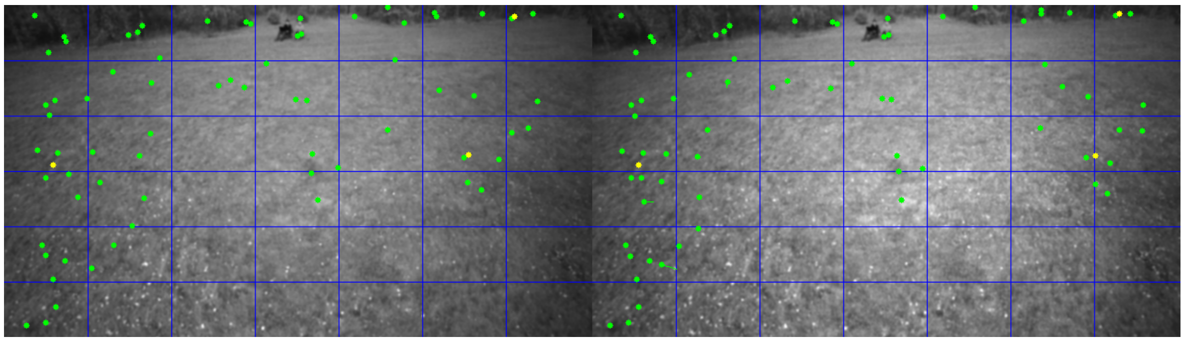


Figure 4.14: An example of S-MSCKF detected features for the captured images influenced by the pattern. The left image represents the not affected camera, and the right image represents the affected image. The center part is affected by the pattern the most as there are almost no features detected. The blue grid represents the areas where the algorithm searches for features. Every cell has a defined maximum amount of features to be detected.

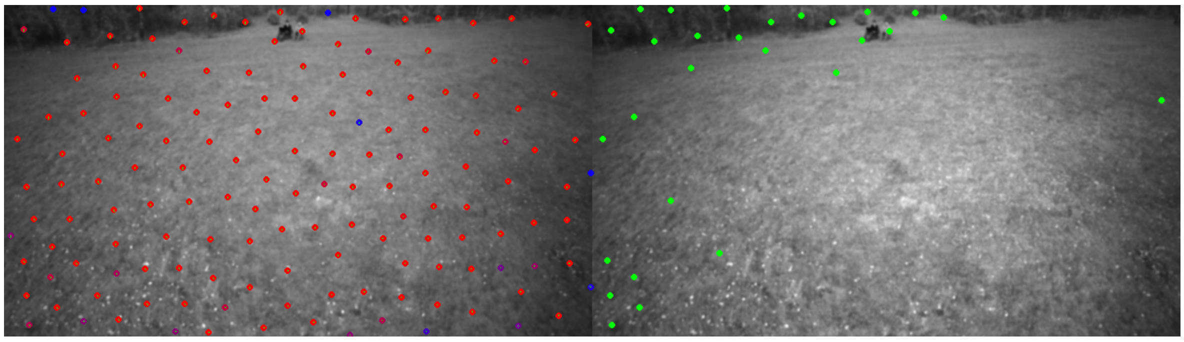
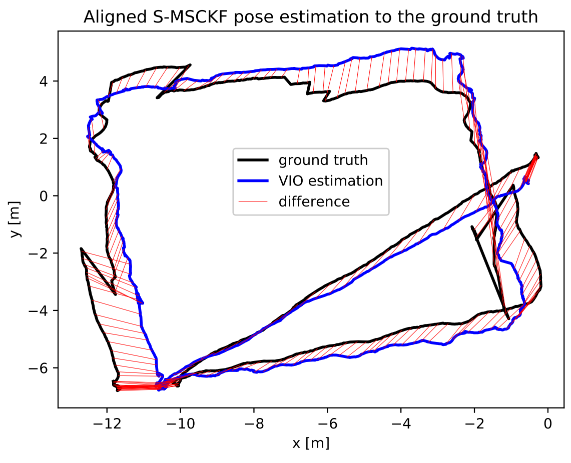
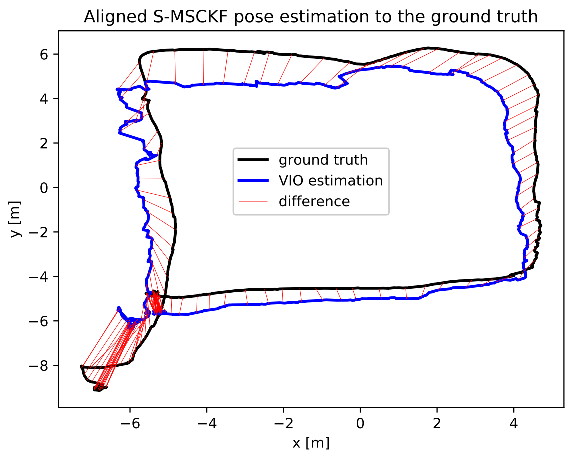


Figure 4.15: An example of VINS-Fusion detected features for the pattern influenced captured images. The left image contains all features detected in the left unaffected image. The blue points represent the newest features. The red points represent the oldest features. The right image shows the matched features between the left and right images used for estimation. As seen the center part has no matched features due to pattern in the right image.

Generally, all camera orientations worked well on the real UAV datasets. As discussed in Chapter 4.6, the advantage of pitched camera 10° might be the absence of direct sunlight on the camera while keeping most of the features in front of the UAV as assumed by the



(a)



(b)

Figure 4.16: The figure (a) represents the aligned trajectories of ground truth and S-MSCKF algorithm for $0°$ camera orientation and $0.5\,\mathrm{m\,s^{-1}}$ UAV velocity, ATE=1.0898 m. The figure (b) represents the aligned trajectories of ground truth and S-MSCKF algorithm for $0°$ camera orientation and $1\,\mathrm{m\,s^{-1}}$ UAV velocity, ATE=1.4819 m. For both figures, the difference between samples (red line) is plotted only for every 10-th sample.

algorithms.

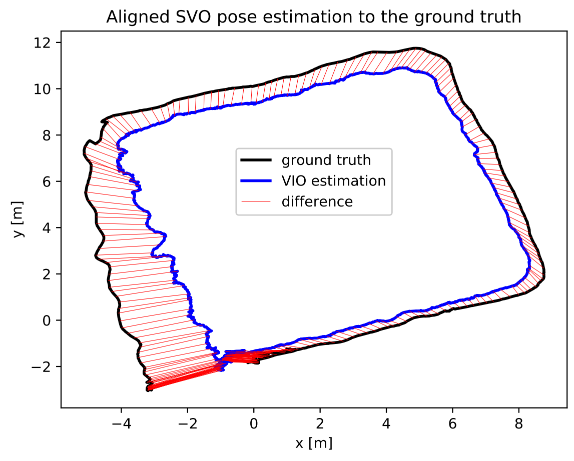
4.8.3 Feedback control with the real UAV

During the outdoor experiments, the S-MSCKF algorithm has already been tuned enough to be able to test it in the feedback loop of the UAV control system. SVO algorithm was not as successful, but it was stable enough to test its performance. Unfortunately, VINS-Fusion was not tuned enough at that time to test it safely in the feedback loop. Hence only S-MSCKF and partly SVO results are presented.

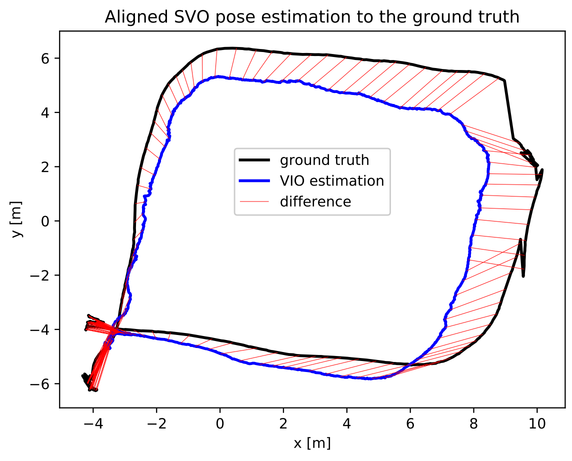
The ground truth values are obtained from RTK station, which should have precision within centimeters. However, the GPS RTK ground truth values can cause sudden jumps when changing from RTK FIX to RTK FLOAT and vice versa. These jumps can be seen in the trajectory alignment figures and significantly influence the ATE result. Because every experiment is unique, the results are presented only in figures.

Figure 4.16 shows that S-MSCKF is suitable to be used in the feedback loop of the UAV control system. The slightly higher ATE values are mostly caused by RTK jumps, which can be seen, especially in Figure 4.16 (a). Also, Figure 4.16 (b) shows that in certain parts of the trajectory, the S-MSCKF estimation might induce some oscillations into the flight. It shows that during the turns at the corners of the trajectory, the light exposure changes due to the sunlight might cause wiggle in S-MSCKF pose estimation. Generally, the S-MSCKF worked well during the experiments.

Despite the volatile SVO performance with the real dataset experiments, SVO pose estimation was usable in the feedback loop, as shown in Figure 4.17. However, SVO did not perform well in the rest of the experiments. The metric scale was the issue causing a huge positional error, and the individual experiments had to be terminated.



(a)

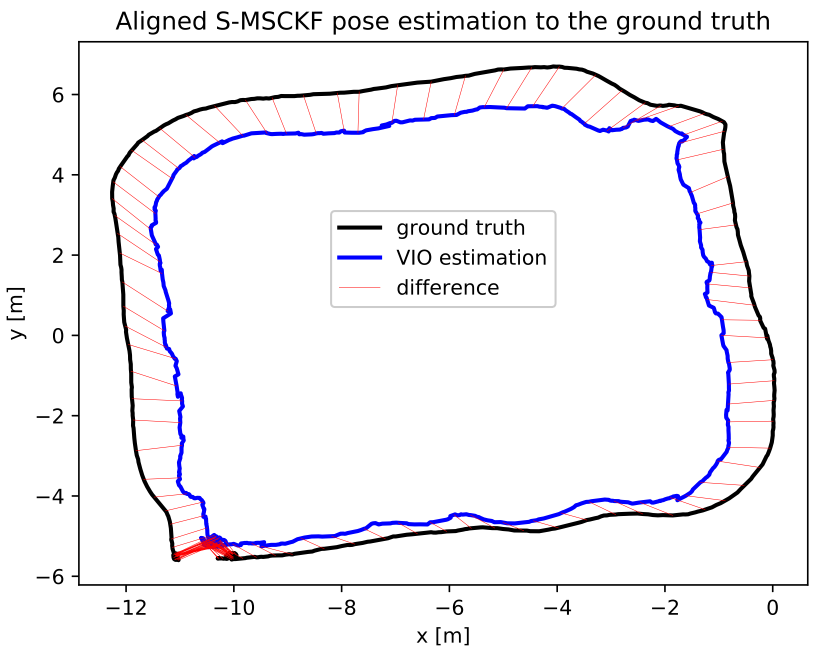


(b)

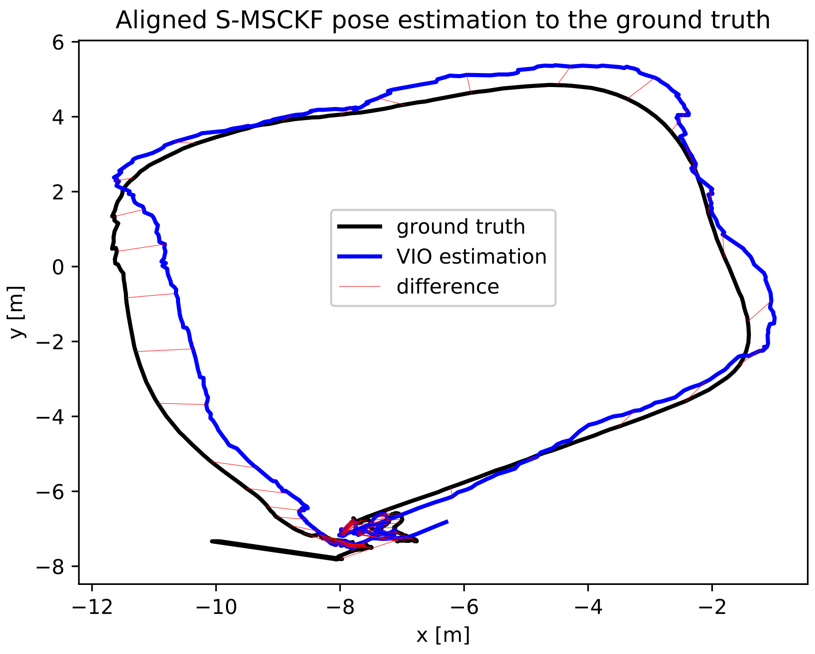
Figure 4.17: The figure (a) represents the aligned trajectories of SVO and ground truth for 0° camera orientation and 0.5 m s^{-1} UAV velocity, ATE=1.3395 m. The figure (b) represents the aligned trajectories of SVO and ground truth for 0° camera orientation and 1 m s^{-1} UAV velocity, ATE=1.3038 m. For both figures, the difference between samples (red line) is plotted only for every 10-th sample.

Until now, the UAV was always facing towards the direction of the flight. Hence the camera was in front of the UAV. However, several tests showed up that constant UAV orientation during the flight might increase the performance in the real environment, as shown in Figure 4.18. The reason is the more or less constant light conditions during the whole flight and minimized camera rapid movements/rotations giving a stable amount of features during the whole flight. Suprisingly, the 3 m high velocity flight show in Figure 4.18 was really successful with minimal drift, despite the fast velocity changes. That shows that S-MSCKF might be strong on fast flights, as authors also tested on their camera setup. Next, the 0° camera orientation flight is shown in Figure 4.18 just proves that constant camera orientation giving constant light condition increases the estimation precision. However, the constant UAV orientation is impractical for the real deployment of the UAV. The UAV consists of a different set of sensors and end effectors such as grippers, nozzels of probes for interaction with the environment for a given task or scenario. The UAV scenario presumes particular UAV orientation, which usually cannot be constant during the flight.

Unfortunately, the tests with 10° and 30° camera orientation were not performed due to limited time available to perform the real experiments.



(a)



(b)

Figure 4.18: The figure (a) represents the aligned trajectories of ground truth and S-MSCKF algorithm for 0° camera orientation, $1\,\mathrm{m\,s^{-1}}$ UAV velocity and fixed UAV orientation giving ATE=0.7574 m. The figure (b) represents the aligned trajectories of ground truth and S-MSCKF algorithm for 90° camera orientation, $3\,\mathrm{m\,s^{-1}}$ UAV velocity and fixed UAV orientation giving ATE=0.4617 m. For both figures, the difference between samples (red line) is plotted only for every 10-th sample.

Chapter 5

Conclusion

In this thesis, the suitability of VIO algorithms has been tested within the control system of the UAV. According to the prior survey, three algorithms have been chosen as the suitable ones, namely filter-based S-MSCKF, semi-direct VO called SVO, and optimization-based VINS-Fusion. From the available hardware cameras from Intel RealSense, the D435i camera was chosen for real experiments. The trajectory was adjusted to fulfill the UAV acceleration limits and to smooth the UAV movements during the flight. For all algorithms, the estimation performance has been tested on a predefined square-shaped trajectory in the simulator environment. Several camera orientations have been used to evaluate the best performance together with several UAV velocities. All algorithms proved the ability to estimate the UAV position from the recorded GPS-controlled flight datasets in the gazebo simulation environment for all used camera orientations and UAV velocities. The feedback loop integration into the simulated UAV model was tested for all algorithms with all used camera orientations and several UAV velocities. Next, the real datasets were captured with the D435i camera attached to the real UAV for all assumed orientations and velocities. In these flights, the RTK GPS was used for the UAV control and later as the ground truth for comparison. Real datasets recordings were influenced mainly by two factors. First, the camera exposure has to react to changing sunlight due to the orientation changes during the flight. This can be partially removed by using the pitched camera, dismissing the direct sunlight influence on the camera. Secondly, the UAV propellers influence the IMU measurements with noisy vibrations degrading the IMU usability. Despite the mentioned issues, S-MSCKF and SVO algorithms were successfully integrated into the feedback loop of the real UAV.

To summarize the result, all of the tested algorithms were capable of feedback loop flights within the simulation environment UAV control system for all camera orientations. The 90° camera orientation had the best result according to the used metric within the simulation environment. As expected, the higher the UAV velocity, the slightly worse results, but the reasonable UAV velocity is between $1\,\mathrm{m\,s^{-1}}$ and $2\,\mathrm{m\,s^{-1}}$. The results show an expected decrease in performance when the algorithms are run on real datasets. In real environment, the camera faces different conditions in terms of exposure changes due to the sunlight. Another encountered issue is the propeller-induced vibrations, which influences the noise of IMU measurements. Unfortunately, the VINS-Fusion algorithm was not tested directly in the feedback loop of the real UAV, but it seems to be promising for further investigation. S-MSCKF algorithm handles these conditions as the best from tested algorithms, and it is suitable for further use in the feedback loop of the UAV control system.

The following assigned tasks have been accomplished in this thesis:

- A survey of available VIO algorithms, especially SVO and MSCKF was performed.
- The author became familiar with stereo cameras, particularly Intel RealSense cameras, and their integration into the ROS middleware, Gazebo simulator, and MRS group UAV platform.
- A dataset from a real-world UAV flight with precise GPS was prepared and the performance of the algorithms on this dataset were compared.
- The algorithms were integrated into the Gazebo simulator and the mounting position of the camera on the UAV affecting the performance of the algorithms was examined.
- The algorithms were prepared for the integration into the position feedback control loop of the UAV and its feasibility was tested in the Gazebo simulator.
- A trajectory-shaping filter improving the localization precision of the algorithms was designed and implemented.

Multimedia material of this work is available on the website `http://mrs.felk.cvut.cz/bednar2020thesis`

5.0.1 Future work

This work showed several ideas about how to improve the VIO estimation precision for chosen algorithms in the MRS system. The used RealSense D435i camera does not guarantee the perfect time synchronization of captured images and IMU measurements. An improvement might appear with the usage of the hand made camera-IMU unit with hardware time synchronization between camera and IMU, providing minimal time delay. Also, the temporal calibration process advertised in Chapter 3.3 can be extended into the other algorithms to further improve the time synchronization between camera images and IMU measurements.

The D435i camera was held by a firm holder, as shown in Figure 4.12. However, this type of attachment transmits the propeller-induced vibrations into the IMU measurements. Hence a dampened holder for the camera might help to increase the IMU measurement reliability. Also, the 90° camera orientation struggles with the start on the ground, but it is more stable in the air due to constant light conditions. The holder can consist of a DC-motor to change the camera angle and use the advantage of individual camera orientations in specific scenarios.

The implemented trajectory shaping assumes the acceleration constraint of the UAV. The MPC tracker, however, has its constraints also for the jerk, which were set accordingly to have a smooth trajectory but to fly most of the time with the desired velocity. Generating the trajectory with the jerk constraint considered could improve UAV behavior, especially for higher acceleration constraints, when the smoothness is more desired.

Bibliography

[1] B. Canis, "Unmanned aircraft systems (uas): Commercial outlook for a new industry," 2015.

[2] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[3] V. Walter, N. Staub, A. Franchi, and M. Saska, "UVDAR System for Visual Relative Localization with application to Leader-Follower Formations of Multirotor UAVs," *IEEE RAL - online first*, 2019.

[4] R. Pěnička, J. Faigl, M. Saska, and P. Váňa, "Data collection planning with non-zero sensing distance for a budget and curvature constrained unmanned aerial vehicle," *Autonomous Robots*, 2019.

[5] T. Baca, P. Stepan, V. Spurny, M. Saska, J. Thomas, G. Loianno, and V. Kumar, "Autonomous landing on a moving vehicle with an unmanned aerial vehicle," *Journal of Field Robotics - online first*, 2019.

[6] V. Spurny, T. Baca, M. Saska, R. Penicka, T. Krajnik, J. Thomas, D. Thakur, G. Loianno, and V. Kumar, "Cooperative Autonomous Search, Grasping and Delivering in a Treasure Hunt Scenario by a Team of UAVs," *Journal of Field Robotics*, vol. 36, no. 1, pp. 125–148, 2019.

[7] G. Loianno, V. Spurny, T. Baca, J. Thomas, D. Thakur, T. Krajnik, A. Zhou, A. Cho, M. Saska, and V. Kumar, "Localization, grasping, and transportation of magnetic objects by a team of mavs in challenging desert like environments," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1576–1583, 2018.

[8] P. Ješke, Š. Klouček, and M. Saska, "Autonomous compact monitoring of large areas using micro aerial vehicles with limited sensory information and computational resources," in *Lecture Notes in Computer Science, vol 11472.* Springer International Publishing, 2019.

[9] M. Petrlík, T. Báča, D. Heřt, M. Vrba, T. Krajník, and M. Saska, "A robust uav system for operations in a constrained environment," in *IEEE ICRA*, 2020, under review.

[10] M. Saska, V. Kratky, V. Spurny, and T. Baca, “Documentation of dark areas of large historical buildings by a formation of unmanned aerial vehicles using model predictive control,” in *IEEE ETFA*, 2017.

[11] R. Richardson, R. Fuentes, T. Chapman, M. Cook, J. Scanlan, Z. Li, and D. Flynn, “Robotic and autonomous systems for resilient infrastructure,” 2017.

[12] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf, “A flexible and scalable slam system with full 3d motion estimation,” in *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011.

[13] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time.” in *Robotics: Science and Systems*, vol. 2, 2014, p. 9.

[14] J. Delmerico and D. Scaramuzza, “A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2502–2509.

[15] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, “The euroc micro aerial vehicle datasets,” *The International Journal of Robotics Research*, vol. 35, 01 2016.

[16] A. Q. Li, A. Coskun, S. M. Doherty, S. Ghasemlou, A. S. Jagtap, M. Modasshir, S. Rahman, A. Singh, M. Xanthidis, J. M. O’Kane *et al.*, “Experimental comparison of open source vision-based state estimation algorithms,” in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 775–786.

[17] J. Engel, V. Usenko, and D. Cremers, “A photometrically calibrated benchmark for monocular visual odometry,” *arXiv preprint arXiv:1607.02555*, 2016.

[18] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[19] M. K. Paul, K. Wu, J. A. Hesch, E. D. Nerurkar, and S. I. Roumeliotis, “A comparative analysis of tightly-coupled monocular, binocular, and stereo vins,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 165–172.

[20] S. Houben, J. Quenzel, N. Krombach, and S. Behnke, “Efficient multi-camera visual-inertial slam for micro aerial vehicles,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1616–1622.

[21] Y. Song, S. Nuske, and S. Scherer, “A multi-sensor fusion mav state estimation from long-range stereo, imu, gps and barometric sensors,” *Sensors*, vol. 17, no. 1, p. 11, 2017.

[22] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, “Robust stereo visual inertial odometry for fast autonomous flight,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[23] 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)*. IEEE, 2015, pp. 298–304.

[24] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, “Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback,” *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, 2017.

[25] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, “Svo: Semidirect visual odometry for monocular and multicamera systems,” *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2016.

[26] T. Qin, J. Pan, S. Cao, and S. Shen, “A general optimization-based framework for local odometry estimation with multiple sensors,” *arXiv preprint arXiv:1901.03638*, 2019.

[27] 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, 02 2014.

[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[29] T. Baca, D. Hert, G. Loianno, M. Saska, and V. Kumar, “Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2018, pp. 1–8.

[30] T. Lee, M. Leok, and N. H. McClamroch, “Geometric tracking control of a quadrotor uav on se (3),” in *49th IEEE conference on decision and control (CDC).* IEEE, 2010, pp. 5420–5425.

[31] B. P. Gerkey and R. T. Vaughan, “Really reusable robot code and the player/stage project,” *Software Engineering for Experimental Robotics, Springer Tracts on Advanced Robotics*, 2007.

[32] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library.* ” O’Reilly Media, Inc.”, 2008.

[33] V. Walter, T. Novák, and M. Saska, “Self-localization of unmanned aerial vehicles based on optical flow in onboard camera images,” in *Lecture Notes in Computer Science, vol 10756.* Cham: Springer International Publishing, 2018.

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

[35] P. Furgale, C. H. Tong, T. D. Barfoot, and G. Sibley, “Continuous-time batch trajectory estimation using temporal basis functions,” *The International Journal of Robotics Research*, vol. 34, no. 14, pp. 1688–1710, 2015.

[36] J. Maye, P. T. Furgale, and R. Siegwart, “Self-supervised calibration for robotic systems,” *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 473–480, 2013.

[37] V. A. A. Variance, “Noise analysis for gyroscopes. 2015,” *Freescale Semiconductor Application Note AN5087.*

[38] O. J. Woodman, "An introduction to inertial navigation," University of Cambridge, Computer Laboratory, Tech. Rep., 2007.

[39] 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.* IEEE, 2007, pp. 3565–3572.

[40] M. Li and A. I. Mourikis, "High-precision, consistent ekf-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 690–711, 2013.

[41] ——, "Optimization-based estimator design for vision-aided inertial navigation," in *Robotics: Science and Systems*, 2013, pp. 241–248.

[42] M. Trajković and M. Hedley, "Fast corner detection," *Image and vision computing*, vol. 16, no. 2, pp. 75–87, 1998.

[43] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[44] B. Kitt, A. Geiger, and H. Lategahn, "Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme," in *2010 ieee intelligent vehicles symposium.* IEEE, 2010, pp. 486–492.

[45] C. G. Harris, M. Stephens *et al.*, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[46] S. Leutenegger, M. Chli, and R. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 IEEE international conference on computer vision (ICCV).* Ieee, 2011, pp. 2548–2555.

[47] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA).* IEEE, 2014, pp. 15–22.

[48] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[49] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition.* IEEE, 1994, pp. 593–600.

[50] D. Nister, "An efficient solution to the five-point relative pose problem," in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, vol. 2. IEEE, 2003, pp. II–195.

[51] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o (n) solution to the pnp problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[52] P. J. Huber, "Robust estimation of a location parameter," in *Breakthroughs in statistics.* Springer, 1992, pp. 492–518.

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

[54] D. Gálvez-López and J. D. Tardos, “Bags of binary words for fast place recognition in image sequences,” *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[55] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in *European conference on computer vision.* Springer, 2010, pp. 778–792.

[56] T. Qin and S. Shen, “Online temporal calibration for monocular visual-inertial systems,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2018, pp. 3662–3669.

[57] E. Mair, M. Fleps, M. Suppa, and D. Burschka, “Spatio-temporal initialization for imu to camera registration,” in *2011 IEEE International Conference on Robotics and Biomimetics.* IEEE, 2011, pp. 557–564.

[58] J. Kelly and G. S. Sukhatme, “A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors,” in *Experimental Robotics.* Springer, 2014, pp. 195–209.

[59] T. Qin, S. Cao, J. Pan, and S. Shen, “A general optimization-based framework for global pose estimation with multiple sensors,” *arXiv preprint arXiv:1901.03642*, 2019.

[60] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition.* IEEE, 2012, pp. 3354–3361.

[61] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of rgb-d slam systems,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2012, pp. 573–580.

[62] B. K. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *Josa a*, vol. 4, no. 4, pp. 629–642, 1987.

[63] M. Petrlík, V. Vonásek, and M. Saska, “Coverage optimization in the cooperative surveillance task using multiple micro aerial vehicles,” in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8914330

[64] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, Sep 2004, pp. 2149–2154.

[65] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, “Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision,” *Autonomous Robots*, vol. 33, pp. 21–39, 2012.

Appendices

CD Content

In Table 1 are listed names of all root directories on CD.

Directory name	**Description**
thesis	the thesis in pdf format
kalibr	the Kalibr example output
realsense_to_imu	the ROS package to merge IMU data from RealSense camera
s_msckf	configuration files of S-MSCKF algorithm
scripts	additional scripts for rosbags processing
simulation_filter	the ROS package for simulator message filtering
svo	configuration files for SVO algorithm
trajectory_shaping	trajectory shaping filter source code
vins	configuration files of VINS-Fusion algorithm
README.txt	additional info

Table 1: CD Content

List of abbreviations

In Table 2 are listed abbreviations used in this thesis.

Abbreviation	**Meaning**
UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
DOPN	Dubins Orienteering Problem with Neighborhoods
SAR	Search And Rescue
UVDAR	Ultra Violet Direction and Ranging
DARPA	Defense Advanced Research Projects Agency
GPS	Global Positioning System
GNSS	Global Navigation Satellite System
DGNSS	Differential Global Navigation Satellite System
RTK	Real-Time Kinematics
LIDAR	Light Detection and Ranging
SLAM	Simultaneous Localization and Mapping
V-SLAM	Vision-based SLAM
IMU	Inertial Measurement Unit
VO	Visual Odometry
VIO	Visual-Inertial Odometry
S-MSCKF	Stereo Multi-State Constraint Kalman Filter
KLT	Kanade-Lucas-Tomasi
RANSAC	Random sample consensus
BRISK	Binary Robust Invariant Scalable Keypoints
ROVIO	Robust Visual-Inertial Odometry
VINS	Visual-Inertial System
OKVIS	Open Keyframe-Based Visual-Inertial SLAM
SVO	Semi-Direct Visual Odometry
PC	Personal Computer
CPU	Central Processing Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
MPC	Model Predictive Control
EKF	Extended Kalman Filter
MSCKF	Multi-State Constraint Kalman Filter

Abbreviation	Meaning
MBZIRC	Mohamed Bin Zayed International Robotics Challenge
MSCKF	Multi-State Constraint Kalman Filter
LTI	Linear Time-Invariant
DoF	Degrees of Freedom
IDL	Interface Definition Language
ROS	Robotic Operating System
MRS	Multi-robot Systems
BSD	Berkeley Software Distribution
FCU	Flight Control Unit
ESC	Electronic Speed Control
FoV	Filed of View
FPS	Frames Per Second
PDF	Portable Document Format
RMSE	Root Mean Square Error
EuRoC	European Robotics Challenge
SfM	Structure from Motion
DSO	Direct Sparse Odometry
LSD-SLAM	Large-Scale Direct Monocular SLAM
PnP	Perspective-n-Point
HKUST	Hong Kong University of Science and Technology
ATE	Absolute Trajectory Error
RPE	Relative Pose Error

Table 2: Lists of abbreviations