Software and Hardware Infrastructure for Visual-Inertial SLAM

by

Mubarik M. Mohamoud

S.B. Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2017

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by .....
Luca Carlone
Associate Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Software and Hardware Infrastructure for Visual-Inertial SLAM

by

Mubarik M. Mohamoud

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

One of the challenges faced by researchers in the field of robot localization and mapping is finding a reliable infrastructure to test their ideas. That infrastructure could be a simulation platform, suitable hardware, or a sensor interface. A useful simulation platform needs to capture the dynamics and the sensor modalities that meet the researchers' needs. A suitable hardware needs to have the capability to navigate, sense the environments, and use onboard computers to run the software it was designed for. A sensor interface allows adapting and testing algorithms on novel sensors.

In this research, we develop an essential hardware and software infrastructure for aiding the development and testing of visual-inertial Simultaneous Localization and Mapping (SLAM) systems. SLAM is a fundamental problem in robot navigation and enables constructing or updating a representation (map) of an environment utilizing sensors on board a robot while concurrently using that representation to localize the robot itself. In visual-inertial SLAM the onboard sensors are cameras (monocular or stereo) and an inertial measurement unit (IMU).

The contribution of this thesis is threefold. First, we develop a hardware platform consisting of a real drone capable of running state-of-art metric-semantic SLAM; this infrastructure allows us to test advanced SLAM algorithms using real sensors and real robot dynamics. Second, we develop a multi-robot simulation platform that includes dynamically accurate, photo-realistic drones; this platform allows extending our tests to multi-robot SLAM systems. Finally, we develop a new sensor interface; in particular, we integrate and test an omnidirectional stereo frontend in Kimera, an open-source visual-inertial SLAM pipeline. The thesis presents the design, implementation, and testing of each contribution.

Thesis Supervisor: Luca Carlone
Title: Associate Professor

Acknowledgments

I would like to express my gratitude to Professor Luca Carlone for the unlimited support that he has shown me through this program. None of this would have been possible without Luca's support and I could not be more thankful for his time, advice, and financial support. I would also like to thank all the researchers in the SPARK Lab including Marcus Abate, Yun Chang, Jingnan Shi, and Samuel Ubellacker who have been helpful throughout this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

The last two decades have seen breathtaking progress in the field of robotics. A fundamental problem to enable robot navigation is Simultaneous Localization and Mapping (SLAM) [20] [4]. SLAM is the computational problem of constructing or updating a representation (map) of an environment utilizing sensors on board a robot while concurrently using that representation to localize the robot itself. Many may consider SLAM as a solved problem, however there are many interesting outstanding challenges including some of the problems discussed in [13], and researchers are continuously coming up with clever improvements for interesting sub-problems. Others, including researchers in SPARK Lab [22] have been putting painstaking effort into pushing the boundary of SLAM towards metric-semantic mapping and high-level 3D scene understanding. The most significant of those efforts include the development of Kimera [48] [47] [49] [17] and 3D Dynamic Scene Graphs (DSGs) [46] [27]. Some of those most recent efforts, including Kimera-Multi [17], enable the extension of the capabilities of SLAM to multi-robot navigation. The general approach is to allow multiple robots navigating the same environment to exchange partial trajectory estimates to cooperatively build a map of the environment.

One of the challenges faced by researchers in this field is finding a reliable robotic infrastructure to test their ideas. The infrastructure includes simulation platforms

that capture the dynamics and the sensor modalities that meet the researchers needs as well as hardware, whether it is a drone, a ground robot or any other robot, that has the capabilities to navigate and capture the environments that the software was designed for.

Even though there are many datasets for SLAM research, including [12] [55] [36], it is common for researchers to spend valuable time developing their own hardware and software to fully test their work. Other times, they have to stop short of thorough testing. In the case of Kimera, the researchers at the SPARK Lab have shown impressive results on datasets including [12], however, they did not have the necessary hardware in place to test Kimera in a real world environment and especially on a flying drone.

To fill this gap, this thesis was dedicated to developing hardware and software infrastructure that can be used to test and further improve the performance of Kimera and other SLAM systems. The infrastructure includes the design of a drone capable of running Kimera on board, and a multi-robot simulation infrastructure. To further enhance the real world capability of Kimera, an omnidirectional stereo camera interface was integrated and tested.

1.2 Thesis Outline

The remainder of this thesis contains four chapters. Chapter 2 will discuss the hardware infrastructure, that is, the drone developed during this research. Chapter 3 will discuss the multi-robot simulation infrastructure. Chapter 4 will discuss the integration and testing of an omnidirectional stereo distortion model in Kimera, and finally Chapter 5 will conclude the thesis by summarizing our findings, lessons learned, and potential future research.

Chapter 2

Drone Design

The first objective of this work was to design and build a drone that has the computational capabilities to run SLAM pipelines like Kimera on board and payload range to carry over 1kg of sensors and a soft gripper [24]. Figure 2-2 shows the platform developed in this thesis and the main components of what we call the SPARK Drone.

2.1 Motivation

Figure 2-1: The Intel Aero Ready to Fly drone components (figure in courtesy of [19]).

This research aims at developing hardware and software infrastructure for Kimera, primarily. The core of Kimera is Kimera-VIO, a Visual-Inertial Odometry (VIO) library. VIO and Visual Inertial Navigation (VIN) consist in estimating the state of a robot using visual and inertial sensors. VIO is well suited for unmanned aerial vehicles (UAVs). This is not only because VIO or VIN are a powerful and mature technology, but also because they rely on lightweight sensor payload which makes them the default option for UAV navigation. Kimera has shown impressive performance on benchmark datasets including, [12]. However, it has not been tested on an actual drone, primarily because there were no drones available off-the-shelf that met the sensing and onboard computation power requirements that were necessary for Kimera.

This presented an opportunity to simultaneously study the hardware and the software that makeup an autonomous aerial vehicle and create a valuable platform for researchers in the SPARK Lab and beyond.

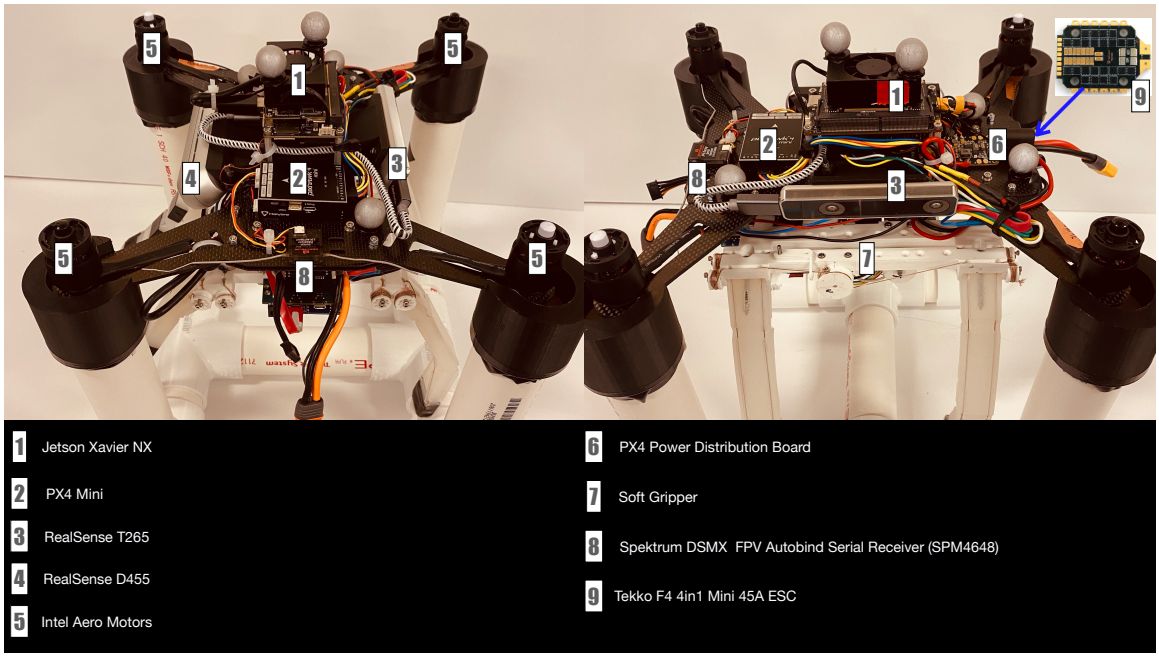


Figure 2-2: This figure shows the final components of the SPARK drone including the frame.

2.2 Related Work

Drone design is a well studied research problem and there has been much research in understanding and analyzing the performance of UAVs. Research publications including [40] and [10] analyze and explain the thrust-to-weight ratio problem for UAVs. The thrust-to-weight ratio is a dimensionless quantity that captures the proportion between the thrust the motors of a drone can produce and how much weight it can carry. This knowledge allows the researcher to adjust the weight and as a consequence endurance of the drone given motor thrust and the propeller efficiency. Other papers such as [38] present a literature review of the current design practices.

Despite the maturity of UAV design, there are no easy-to-use, off-the-shelf and capable drone platforms out there for research. It is common practice that researchers design a drone that is tailored for their experiments. Antonini et al. [3] built their own drone with a Jetson TX2 to collect the Blackbird dataset. This design is similar to the SPARK Drone in terms of on board computation components, but they used a DJI Snail propulsion system. Tian et al. [60] built their own quadcopters to conduct *search and rescue operations in a Forest*. Their design includes an Intel NUC onboard computer. They also used a DJI propulsion system. UAV papers from Kumar's lab including [59] and [28] used a Snapdragon propulsion system, but their 236g quadcopter is much smaller and lighter weight than what was required from the SPARK Drone, which is required to carry multiple sensors and a soft gripper.

Although all of the mentioned works guided the design considerations of the SPARK Drone, the main references used were the Intel Aero Ready to Fly drone documentations [18] and [19]. The SPARK Drone propulsion system, to be discussed as part of the contribution, is a modified Intel Aero Ready to Fly drone, but the computation components and sensors were specifically designed to allow deployment of modern SLAM systems.

2.3 Design Decisions

Scholars in the SPARK Lab have been using the Intel Aero Ready to Fly drone for research and teaching for several years now. One of the more recent publications using the Intel Aero Ready to Fly drone is the award winning paper, Dynamic Grasping with a "soft" Drone: From Theory to Practice by *Fishman et al.* [24]. The goal of the research was to have a drone locate an object in the environment, pick up the object using a soft gripper and drop it off at a known location. The researchers faced two major challenges with the Intel Aero Ready to Fly drone: First, the compute board on the Intel Aero Ready to Fly drone did not have the computational capacity to run a VIO or VIN library like Kimera on board, which means the researchers had to rely on external motion capture system both for locating the target object and for tracking the location of the drone. Second, the drone was heavy because of the attached gripper, which severely limited the flight time.

The new SPARK drones were designed to address those two limitations. It needed to have the computational capabilities necessary to run Kimera on board while being lighter than the Intel Aero Ready to Fly drone.

To solve the computation issue without increasing the weight, extensive research has been done to compile a set of candidate on board computation modules. Table 2.2 shows the specifications under consideration for the final set of candidate compute modules. Among those, the most important parameters are weight, power consumption GPU/CPU, memory, and footprint. The Jetson TX2 beats the Jetson Nano on all measures except the weight, power consumption, and footprint. However, the Jetson Nano is still a powerful computer and has a reduced weight and form-factor. But the clear winner in Table 2.2 is the Jetson Xavier NX which beats the Nano on computation, memory, and storage, but also has the same footprint. The Xavier NX is slightly heavier than the Jetson Nano and the weight difference also increases with the suitable heat-sink. In addition, the Jetson Nano has slightly lower power consumption, but all of these modules are low power. At the end, the Jetson Xavier NX was selected as the main compute board for the SPARK Drone.

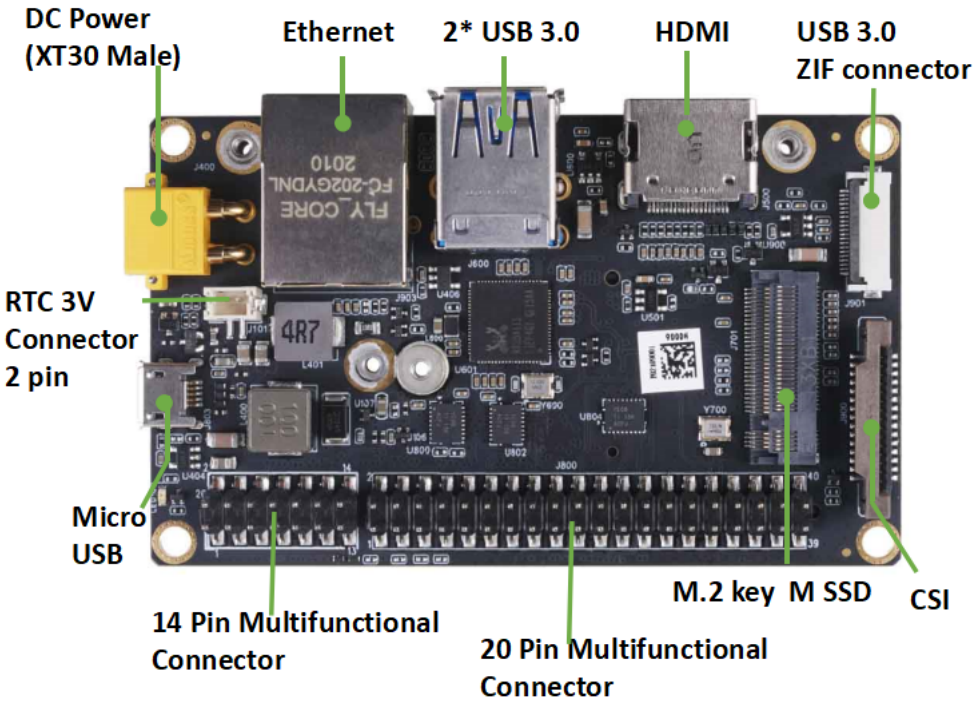


Figure 2-3: The A203 carrier board which was used to replace the Jetson Xavier NX developer Kit (image courtesy of [21]).

The Xavier NX is a powerful supercomputer with the form-factor of a credit card, but it comes with a carrier board that does not meet our weight and form-factor constraints. Again, extensive research was done to compile a list of custom carrier boards to replace the Jetson Xavier NX carrier board which is also known as the Jetson Xavier NX developer Kit. Weight and footprint were very important parameters under considerations for a carrier board, but the available input-output interfaces were also very important, especially the availability of enough serial ports to accommodate two RealSense Cameras and the soft gripper. Table 2.1 shows the important specifications under consideration for candidate carrier boards.

The A203 V2 custom carrier board for the Jetson Nano/NX (see Figure 2-3) was used to replace the NX carrier board. A203 V2 provides two USB 3.0 type-A ports as well as a USB 3.0 ZIF connector. The 2 USB type-A ports were used for the RealSense cameras while the USB ZIF connector was used to connect the Arduino module that is used to control the soft gripper.

spec	NX Dev board	A203	Quark Carrier	EN715
Weight	97g	55g	33g	57g
Serial connectors	4xUSB 3.0 type A	2xUSB 3.0 Type-A and USB 3.0 ZIF	1x USB 3.1 Type-C and USB0 OTG	2x USB 3.0 Type-A
Display	HDMI and DP	HDMI	None	HDMI
Footprint	100mmx80mm	87mmx52mm	82.6mmx58.8mm	87mmx70.6mm

Table 2.1: The important specifications of carrier boards to replace the Jetson Xavier NX Developer Kit carrier board.

To further minimize the weight, the Pixhawk mini flight controller and Tekko32 F4 4in1 mini 45A electronic speed controller were used. The Intel Aero Ready to Fly drone comes with four electronic speed controllers (ESCs) each weighing 17-grams (measured) and with a combined weight of 68-grams. Tekko32 F4 Mini weighs only 8-grams which is a 60-grams reduction. This reduction is more than the entire weight of the A203 carrier board.

The weight was further reduced by re-designing the frame and removing all unnecessary parts including stand-offs (with combined weight of 68-grams), the GPS module, and the on board sensors (including the RealSense R200, front facing RGB camera, and down-ward facing optical flow camera), and bottom plate which are all shown on Figure 2-1. This removed an additional estimated weight of 200-grams which made space for state-of-the-art RealSense T265 and D455 shown on Figure 2-2. With these components on the customized Intel Aero Frame and propulsion system, the initial drone was about 100-grams lighter than the Intel Aero ready to fly. Figure 2-2 shows the major components on board the SPARK Drone.

2.4 Final Product and Future Work

The SPARK Drone prototype is shown in Figure 2-2. We will evaluate its capability in Chapter 4, where we also introduce an interface to use the omnidirectional camera (tested with Realsense T265).

The design can be further improved by replacing the motors with more efficient

Spec	Jetson TX2	Jetson Xavier NX	Jetson Nano
AI Performance	1.3 TFLOPS (FP16)	6 TFLOPS (FP16) 21 TOPS (INT8)	0.5 TFLOPS (FP16)
Power	7.5-15W	10-15 W	5-10W
Memory	8GB 128b LPDDR4 Memory 1866 MHx - 59.7 GB/s	8GB 128-bit LPDDR4x 59.7GB/s	4 GB 64-bit LPDDR4 25.6 GB/s
GPU	256-core NVIDIA Pascal GPU architecture with 256 NVIDIA CUDA cores	384-core NVIDIA Volta™ GPU with 48 Tensor Cores	128-core NVIDIA Maxwell™ GPU
CPU	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM Cortex-A57 MPCore	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6 MB L2 + 4 MB L3	Quad-core ARM A57 @ 1.43 GHz
Storage	32GB eMMC 5.1	16 GB eMMC 5.1	
Weight	85 grams	24 grams	19 grams
Foot print	50 x 87 mm	45 mm x 70mm	45 mm x 70mm

Table 2.2: The specifications for candidate compute modules to replace the Intel Aero compute board.

and powerful motors. There has been more research on studying and improving the electric propulsion of drones. Some of the research that may guide optimization of the thrust-to-weight ratio of the SPARK Drone includes *Soohun Oh et al.* [40] which presented a solution that enables the performance analysis and optimal design of multicopter drones. The work Electric Multirotor Propulsion System Sizing for Performance Prediction and Design Optimization by Bershadsky et al. [10] also proposed a thorough method of propulsion selection for multi-copter UAVs.

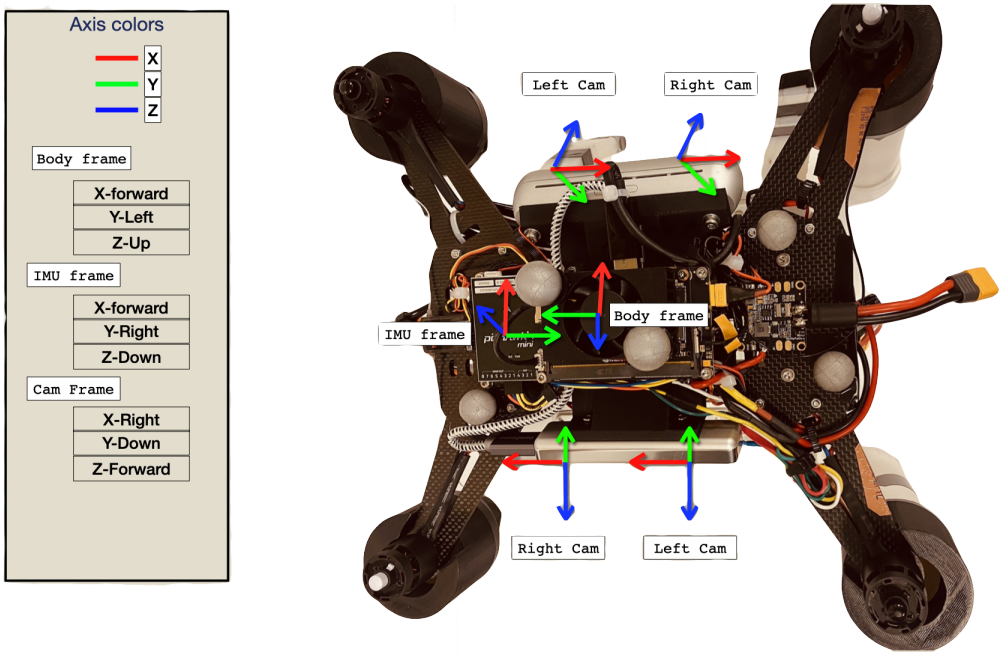


Figure 2-4: This figure shows the coordinate frame definitions for the SPARK Drone body and sensors.

Chapter 3

Multi-Robot Simulation Infrastructure

3.1 Motivation

"Rome wasn't built in a day" is a globally known and self-evident truth; what is equally true is that Rome was not built by one person working alone. As collaboration is paramount for human productivity, it will also be critical for robot effectiveness.

To bring the societal impact of robotics to the next level, researchers in the SPARK Lab and beyond have focused on extending the capability of SLAM to multi-robot systems. The most recent efforts, including Kimera-Multi [17], have shown promising results in enabling communication and awareness among a set of SLAM-enabled robots navigating or working in the same environment by allowing them to share insights they have individually or collectively learned about the environment and build their beliefs based on information consensus.

The continuity of those efforts rely on the availability of reliable and easy-to-use robotic infrastructure. This infrastructure can be hardware or simulated. Chapter 2 of this thesis discussed the development of the SPARK Drone which has the capabilities to cover the hardware needs of those efforts. This chapter will discuss the development of a multi-robot simulation environment.

3.2 Objectives

The goal of the research described in this chapter is to develop a multi-robot, simulated environment that SLAM researchers can use to test their ideas. The environment needs to be photo-realistic and dynamically capable, but also easy to use. To achieve these goals, the simulation environment needed to have the following functionality:

- Enable multiple robots to operate in the same environment.
- Allow the researchers to easily stream the sensor data from all robots as topics and messages in the Robotics Operating System (ROS).
- Create the infrastructure needed to separately and simultaneously control multiple robots.

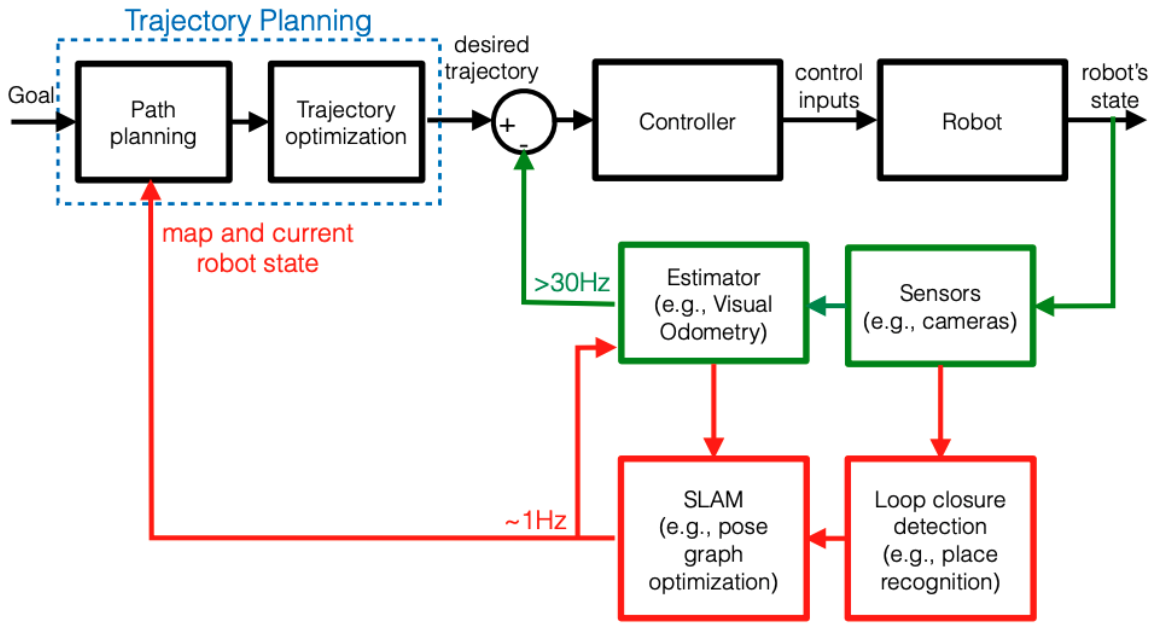


Figure 3-1: A block diagram showing a typical visual navigation pipeline from the course Visual Navigation for Autonomous Vehicles (VNAV) [30] [1].

Figure 3-1 shows the typical high-level architecture of a visual navigation pipeline and how SLAM interacts with the other components of the robot's software stack.

The goal of the work in this chapter is three-fold: First, we implement the blocks on the top row (black boxes) in Figure 3-1 for a multi-robot system. That is the path planning, trajectory optimization, controller, and most importantly simulating the robot itself. Second, we test the entire pipeline shown in Figure 3-1 simultaneously for multiple robots navigating the same environment using multiple instances of Kimera-VIO. Third, we test Kimera-Multi [17] which enables multiple robots navigating the same environment to build a shared map and trajectory estimate by exchanging the optimized pose graph and loop closures. This communication enables the robots to merge their trajectory estimates in a distributed or centralized fashion.

3.3 Related Work

3.3.1 Multi-robot Simulation Platforms

Some of the publications reviewed for this research on multi-robot simulation platforms, including [62] and [41], studied the performance and usability of existing simulation platforms. Ramli et al. [41] studied several simulation platforms and their usability for multi-robot simulations. The platforms they have reviewed include NetLogo, GAMA Platform, Webots, Player/Stage, and V-REP. The researchers suggested NetLogo for multi-robot simulation, but NetLogo is more suitable for a 2D simulation setups. For more photo-realistic 3D based multi-robot environments, Lackele et al. [31] presented SwarmSimX.

Even though the above research provided some inspirations in terms of parallel efforts in multi-robot simulation, the platform used for this research was TESSE [42]. TESSE, which will be discussed further in this chapter, is a Unity based, photo-realistic simulation platform that has been extensively used by the SPARK Lab. Because of that prior know-how and infrastructure, it made most sense to extend that platform to multi-robot operation instead of introducing a new system.

3.4 Implementation

3.4.1 TESSE

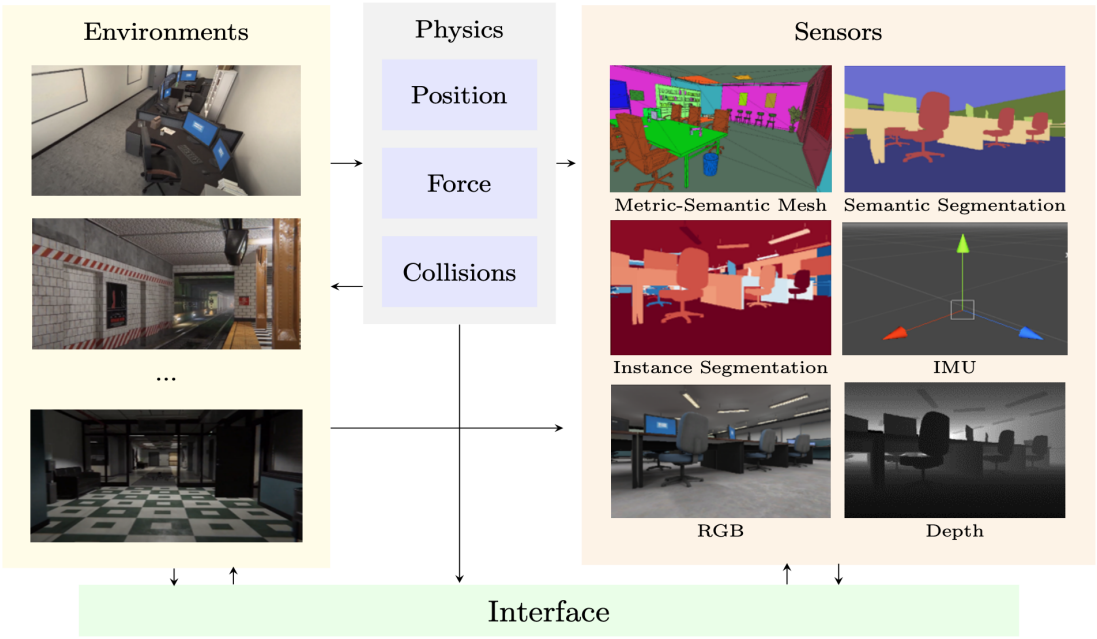


Figure 3-2: The main system diagram of the TESSE simulation which shows the main sensor and communication layers (figure courtesy [42])

The multi-robot simulation developed during this research is an extension of the Task Execution with Semantic Segmentation Environments (TESSE) simulator [42]. TESSE is an open-source Unity plugin that enables users to develop robotic, computer vision, and reinforcement learning technologies. TESSE was selected as the starting point for this work for the following reasons.

- TESSE does not require many dependencies.
- It provides access to different types of sensors, including red-green-blue (RGB) cameras and depth sensors. It also provides a semantic segmentation for each camera image.
- It provides access to physics information for the agent (robot) including position, velocity, and orientation.

- It provides network interfaces to allow sending control commands to Unity and receiving sensor data.

- Researchers in SPARK Lab have been using TESSE for several years including developing their own robotics agents.

The range of cameras available in TESSE and the ability to access physics information supply the complete set of sensor modalities needed for visual-inertial SLAM. The high rate position, velocity, and orientation updates are used to construct virtually noiseless ground-truth odometry and IMU data. The network interfaces available for sending and receiving data over TCP or UDP ports allowed the streaming of sensor and agent state information from Unity and sending control and setup commands to Unity.

Some of the most significant research projects out of the SPARK Lab, including [49], [46], were tested on the single agent version of TESSE, and courses including [1] [30] and Robotics Sciences and Systems (RSS) [2] use TESSE for teaching robotics. Through that multi-year experience, the researchers collected or developed a variety of simulated environments and infrastructure for single robot applications.

3.4.2 TESSE-Multi in Unity

Since TESSE was not designed with multi-robot operation in mind, the bulk of the work went into understanding the code and the architecture of TESSE and Unity to figure out a safe and clean way to add additional robots. TESSE robots live in a dedicated scene called tesse_multiscene. tesse_multiscene is an empty scene that does not contain any physics objects other than the robot itself. This empty scene can additively load other scenes which permits TESSE projects to contain multiple scenes that can be switched between. The most logical way to add new robots was to add them in a tesse_multiscene that already contains another robot. The following steps describe how one may add and integrate a new fully functioning robot to an existing tesse_multiscene.

Adding new agents in Unity: The first step is to drop or copy and paste the

agent into the scene. When this is done, various errors may appear and one needs to fix them one by one. The first error one may encounter is a "multiple audio listeners" error. Unity allows one and only one audio listener for an open scene and one needs to uncheck the audio listener box for all the other robots. You may also see warnings and errors on camera displays. Unity provides 8-camera displays and one needs to map the cameras to those eight different displays such that no two cameras are attempting to use the same display.

Playing and building the scene: After adding a new agent into the scene, attempting to build or play the scene in game mode will encounter two main issues: 1) By default all robots will attempt to initialize the same scenes which will cause instabilities as this process will attempt to spawn multiple collision-enabled objects such as walls in the same physical locations. 2) All robots will be spawned on the same spot which prevents spawning due to collision issues. To stop all robots from loading the scenes, the position interface which handles scene loading, positioning, and scene changes was modified to allow only one robot to load all the scenes. To initialize the scene the robot needs to have a *is_master* flag set to true. Only the robot with that *is_master* flag set will be able to load the scene.

To resolve the issue of multiple robots spawning at the same location, the *base_controller* was modified to allow robots to be spawned at predefined positions with predefined orientations. Since the experiments done for this thesis involved only two robots, the *is_master* flag was used to select the position of the robot. A switch mechanism based on robot id could work for more than two robots.

Enabling Communication: By default, all agents use the same network ports which means connecting to any agent using the TESSE Python Interface will not be possible. It is necessary to give each robot a unique set of communication ports. Table 3.1 shows the ports used for the two robot experiments tested for the current setup.

Port	Agent 1	Agent 2
Pos_listen_port	9000	8000
Met_listen_port	9001	8001
Img_listen_port	9002	8002
Pos_update_port	9003	8003
Step_listen_port	9005	8005
Lidar_listen_port	9006	8006
Pos_send_port	9000	8000
Meta_send_port	9001	8001
Img_send_port	9002	8002
Met_broadcast_port	9004	8004
Lidar_send_port	9006	8006

Table 3.1: Network port mapping for two agents

3.4.3 Multi Agent TESSE ROS Bridge

TESSE ROS Bridge is a ROS package in python that processes the information received by the TESSE Python Interface over the network into ROS messages and then publishes them into appropriate topics. The messages published by this package include image data, IMU, odometry, transformations and sensor information (e.g. camera intrinsics).

Extending this package to multi-agent support, running multiple, properly namespaced instances was not going to work as the design allowed only single connection over the network to a single Unity Instance. To enable multi-agent functionality, the connection setup procedure and the message request and response procedures were separated. This allows us to establish the connection with a single instance of TESSE Interface and have that communicate with Unity by properly mapping requests and commands to the right agent using the communication port enabled for that robot.

3.4.4 Multi-Robot Trajectory Optimization

The primary reason why TESSE was chosen as the starting point for this work was to minimize the time it takes researchers in the SPARK Lab to set up and use the platform. With this platform, researchers will be able to collect data in any Unity scene by dropping their agents in predefined locations and have them fly around on

a trajectory that they defined by selecting a set of way-points in free space.

The trajectory optimization which is based on the implementation of Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments [44] by Richter et al. was first tested for a project for VNAV [1]. The implementation uses MAV Trajectory Generation [37] by Achtelik et al.

Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments [44] uses multi-segment polynomial trajectory optimization and leveraging the differential flatness property of a quadrotor. With the provided package, all the researcher needs to provide is a csv file containing a set of way-points for each robot.

3.4.5 Multi-Robot Controllers

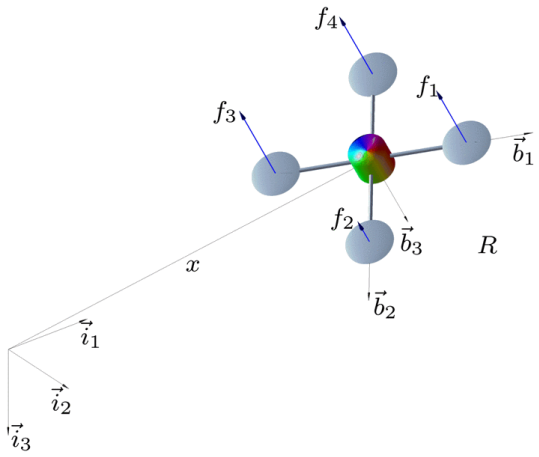


Figure 3-3: Quadrotor model (courtesy of [32])

For the tracking controller of a quadrotor, we used the geometric controller by Lee et al. [32] which we first implemented as part of VNAV [1]. The quadrotor UAV dynamics is expressed globally on the configuration manifold of the Special Euclidean group SE(3) which allows it to avoid singularities and complexities that arise as a result of using local coordinates. Equation (3.1) shows the mapping between the quadrotor propeller forces (also illustrated in Figure 3-3) and the overall thrust and torque of the platform:

$$\begin{bmatrix} f \\ M_1 \\ M_2 \\ M_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -d & 0 & d \\ d & -d & 0 & 0 \\ -c_{\tau f} & -c_{\tau f} & c_{\tau f} & -c_{\tau f} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \tag{3.1}$$

Where f_1,f_2,f_3, and f_4 are the propeller force commands send to Unity, f is the total thrust, and M_1,M_2, and M_3 are the moments in the body fixed frame. The control laws for f and $M = [M_1, M_2, M_3]$ are given in equations (3.2) and (3.3) respectively. (3.3) correspond to a tracking controller on SO(3) that stabilizes the attitude tracking error, while (3.2) corresponds to a tracking controller for the translation dynamics on $\mathbb{R}^3$ [32]. The definitions of all the terms in equations (3.2) and (3.3) are included in Table 3.2 and more extensive documentation can be found in [32]. The portion involving Ω_d $(-J(\hat{\Omega}R^T R_d\Omega_d - R^T R_d\hat{\Omega}_d))$ of Equation (3.3) was ignored in the implementation for this project. Also the ignored terms are not listed in Table 3.2

$$f = -(-k_x e_x - k_v e_v - mge_3 + m\ddot{x}_d).Re_3 \tag{3.2}$$

$$M = -k_R e_R - k_\Omega e_\Omega + \Omega \times J\Omega - J(\hat{\Omega}R^T R_d\Omega_d - R^T R_d\hat{\Omega}_d) \tag{3.3}$$

The TESSE Multi Robot packages include the $C++$ controller discussed with all the documentation and namespacing provided to allow the user to effortlessly integrate new robots. The user will be able to add a new robot to the main launch file by specifying the robot names. The launch file will automatically namespace all messages coming from that robot using the robot name. All the messages concerning that robot, including control commands, are expected to use that namespace as well. for example, if the name of the new robot is $tesse2$, the robot state topic is expected to be $/tesse2/current\_state$.

For tuning the controllers to get a desired behavior, the user will be able to adjust the coefficients of the controllers k_x, k_v, k_R, and k_Ω via a provided yaml file.

Term	Definition
k_x	Distance error
e_x	Distance error coefficient
k_v	Velocity error coefficient
e_v	Velocity error
m	The total mass
g	Gravity constant
e_3	$[0, 0, 1]$
$\ddot{x}_d$	desired acceleration
R	The rotation matrix from the body-fixed frame to the inertial frame
e_R	Rotational error
k_R	Rotational error coefficient
e_Ω	Angular error
k_Ω	Angular error coefficient
Ω	the angular velocity in the body-fixed frame
J	The inertia matrix with respect to the body-fixed frame

Table 3.2: Important terms in equations (3.2) and (3.3).

3.5 Testing

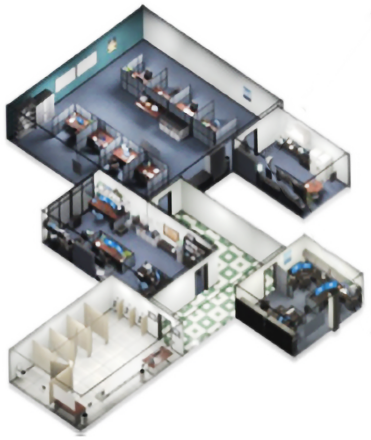
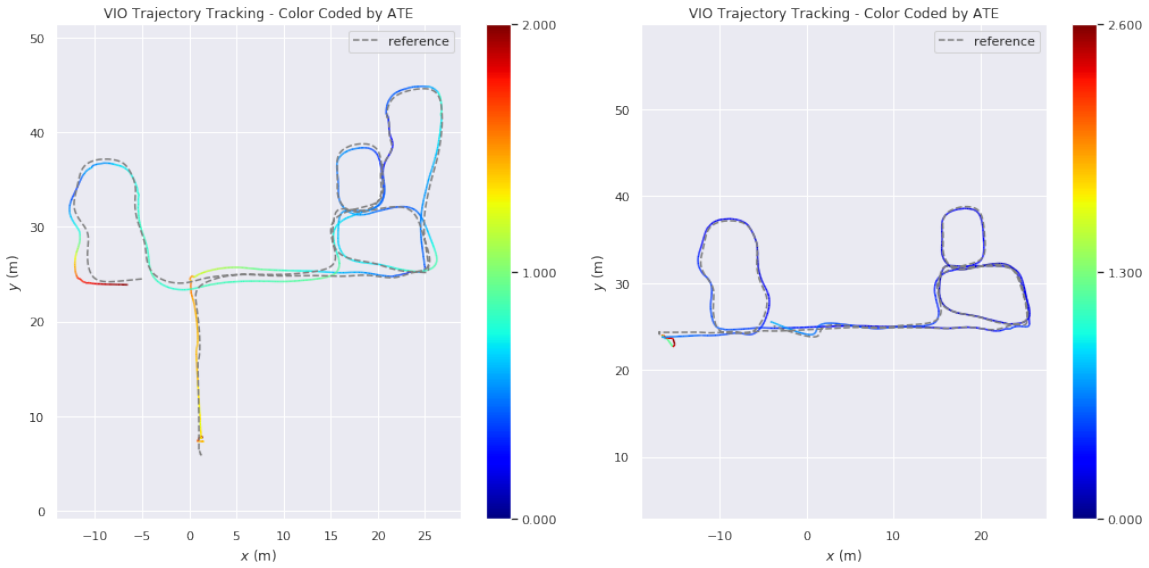


Figure 3-4: A visualization of the *office scene* for this section (figure courtesy of [42]

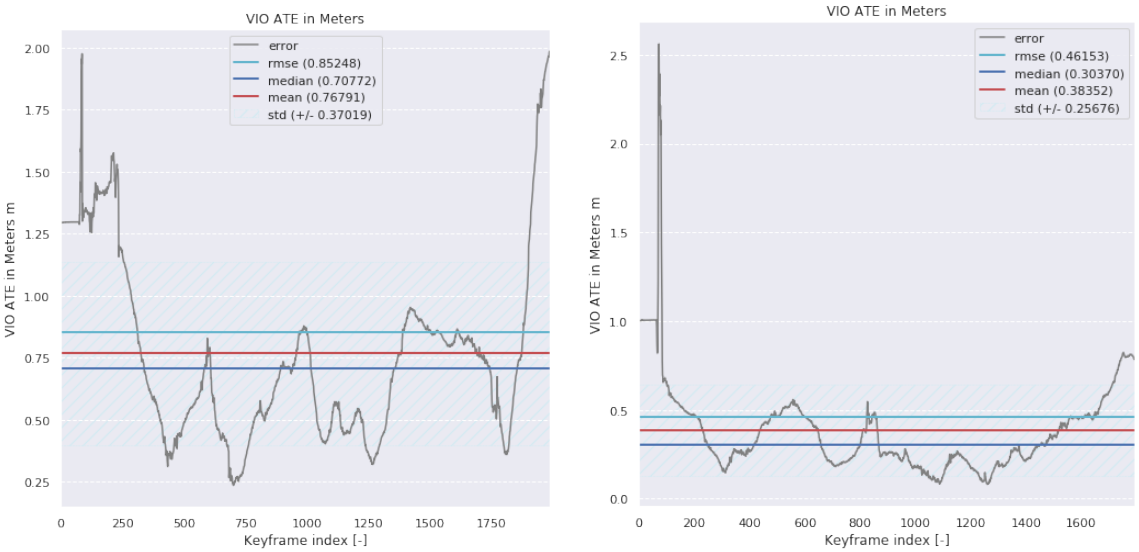
3.5.1 Testing with Multiple Instances of Kimera

Experimental Setup: A ROS bagfile of two drones simultaneously exploring the Office Scene (shown in Figure 3-4) was collected. Then two instances of Kimera-VIO were run on the bag file and the frontend and backend statistics were collected using



(a) robot 1 (b) robot 2

Figure 3-5: The aligned trajectory estimate (colored) of Kimera-VIO and the ground truth odometry (dashed) for the first robot (a) and the second robot (b).



(a) robot 1 (b) robot 2

Figure 3-6: The absolute translation errors after trajectory alignment.

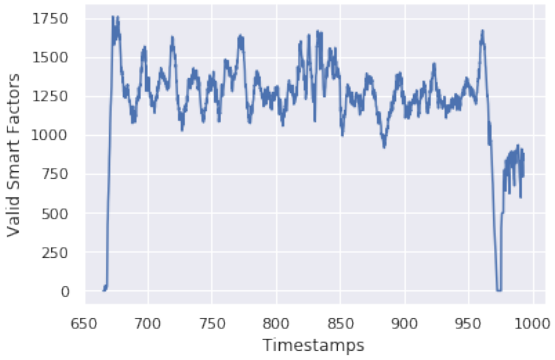
our Kimera-Multi simulator.

Results: Figures 3-5a and 3-5b show the aligned Kimera-VIO estimates (colored) and the ground truth trajectory (dashed) for the two robots. The estimated trajectories can be seen to qualitatively remain close to the ground truth. Over time,

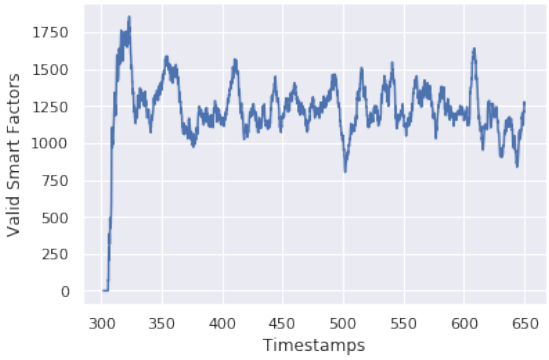
the VIO estimate will gradually drift from the ground truth as expected (we disabled loop closure detection for this experiment). The absolute translation errors after trajectory alignment are shown in Figures 3-6a and 3-6b. The translation error at *keyframe* i is the magnitude of the vector computed as the difference between the ground truth position of the robot at *keyframe* i and the Kimera-VIO estimated position of the robot at *keyframe* i after alignment:

$$t^i_{error} = ||t^i_{gt} - t^i_{kimera}||_2 \tag{3.4}$$

The RMSE of the absolute translation errors remain under 1m for robot 1 and below 0.5m for robot 2. The trajectories are over 100 meters long each.



(a) robot1: Smart factors



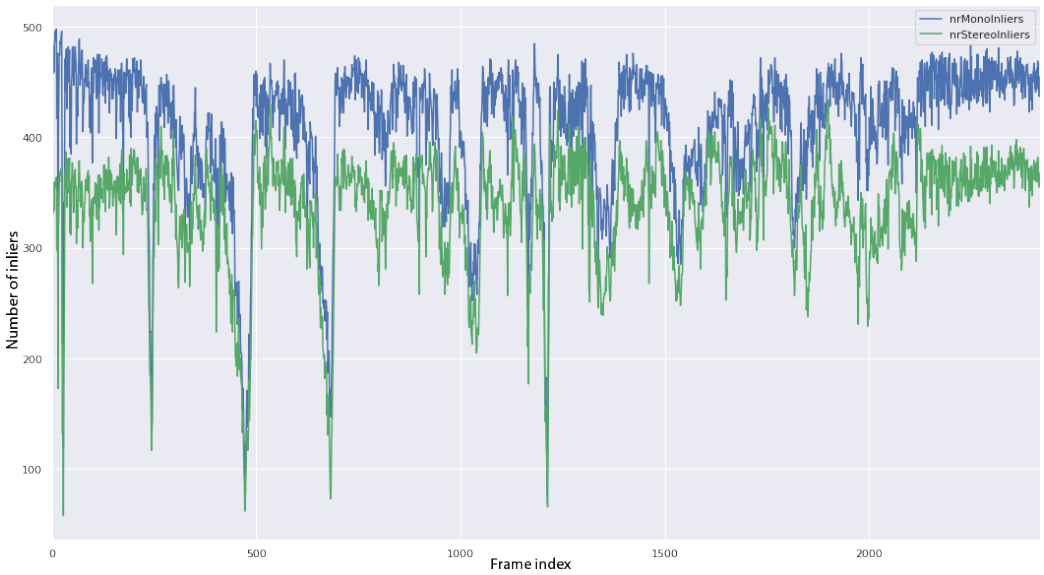
(b) robot2: Smart factors

Figure 3-7: As presented by Carlone et al. [14], Smart Factors is a compact representation of factors of a factor graph by eliminating large sets of support variables without losing the information they contain. Valid smart factors contain visual measurements that are considered correct for the VIO estimator.

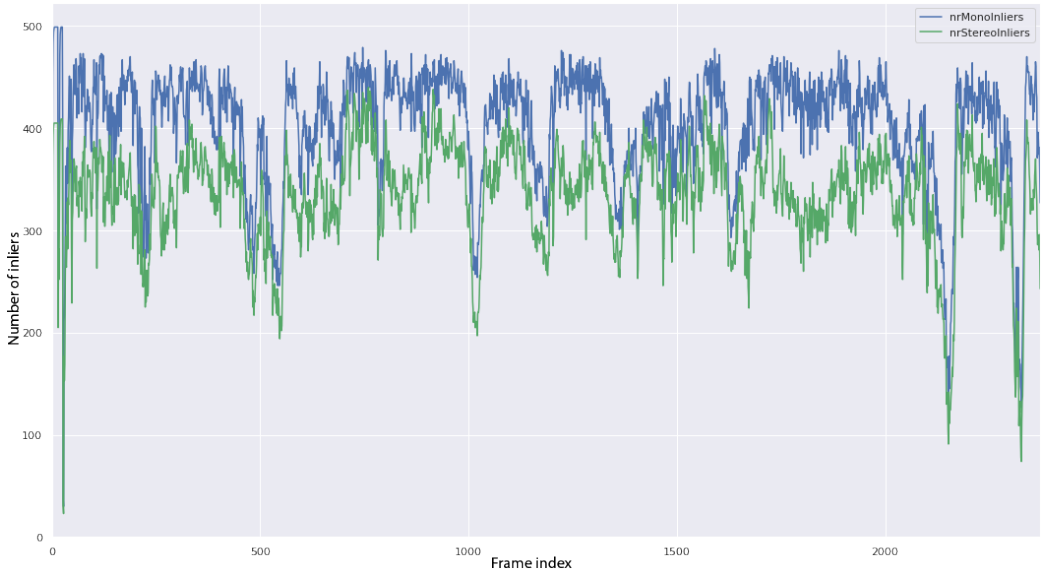
The high number of RANSAC inliers shown in Figures 3-8a and 3-8b and the high smart factors in Figure 3-7 confirm that the visual interface between Unity and Kimera works correctly.

3.5.2 Testing with Kimera-Multi Centralized

Experimental Setup: In the previous experiment, we have shown the feasibility of running two instances of Kimera-VIO in parallel on the sensor data provided by



(a) Robot 1 RANSAC inliers



(b) Robot 2 RANSAC Inliers

Figure 3-8: Random sample consensus [23], or RANSAC, is an iterative method for estimating a mathematical model from a data set that contains outliers. The RANSAC algorithm works by identifying the outliers in a data set and estimating the desired model using data that does not contain outliers. In this case, the data is a set of frame-to-frame key-points for mono ransac and left-to-right keypoints for stereo while the model is relative rotation and translation or a pose. Higher inliers means higher chance that the estimated model is correct.

two simulated drones flying in the same environment. In this experiment, we added a Kimera-Multi-Centralized base-station as shown in figure 3-9. This setup of Kimera-

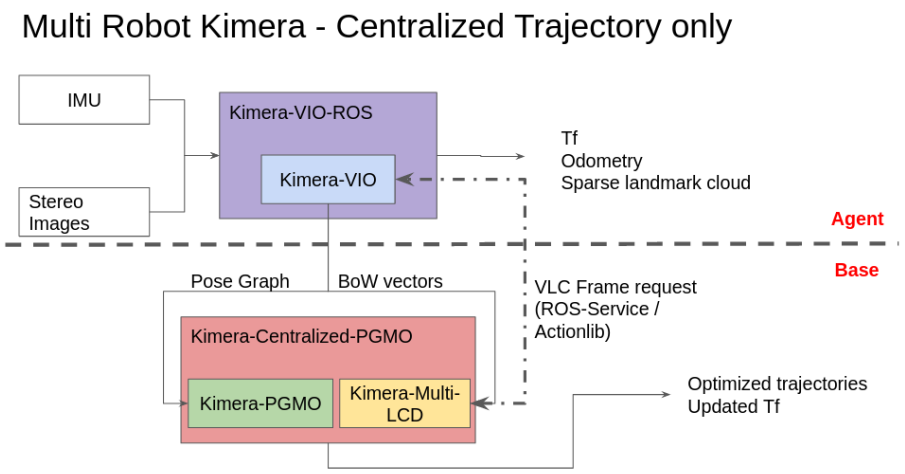


Figure 3-9: Kimera-Multi centralized architecture (courtesy of Yun Chang [17])

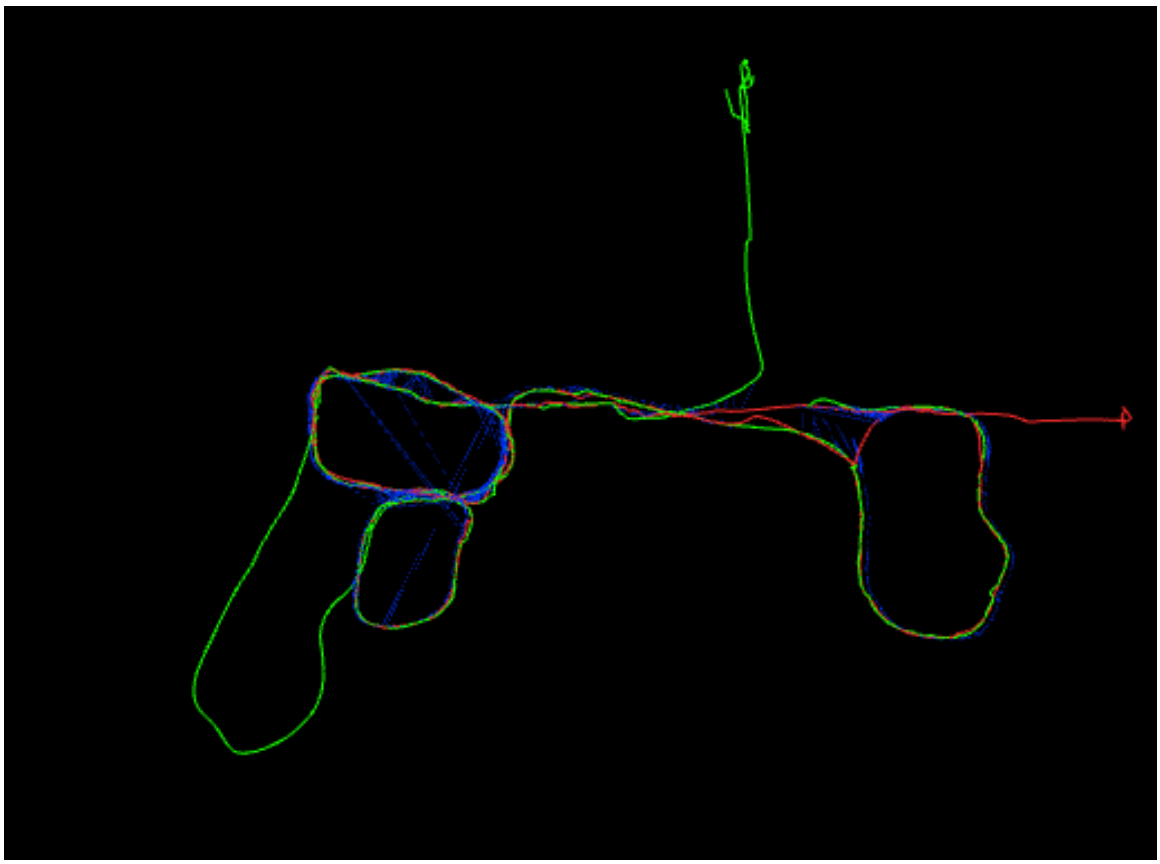


Figure 3-10: Aligned trajectory of the two robots showing loop closure detections.

Multi-Centralized enables detecting loop closures between multiple robots exploring the same environment and as result aligning their trajectories in real-time. Note that only one agent is shown in the architecture in Figure 3-9 connected to the base-station,

but additional agents can be added the same way.

Results: Figure 3-10 shows the aligned trajectories of the two robots. The blue lines show the loop closure detections. Note that some of the loop closures are for a single robot trajectory. The inter-robot loop closures which connect the green and red trajectories are the ones used for trajectory alignment. Figure 3-11 shows the unaligned trajectories when the inter-robot loop closure detection was disabled.

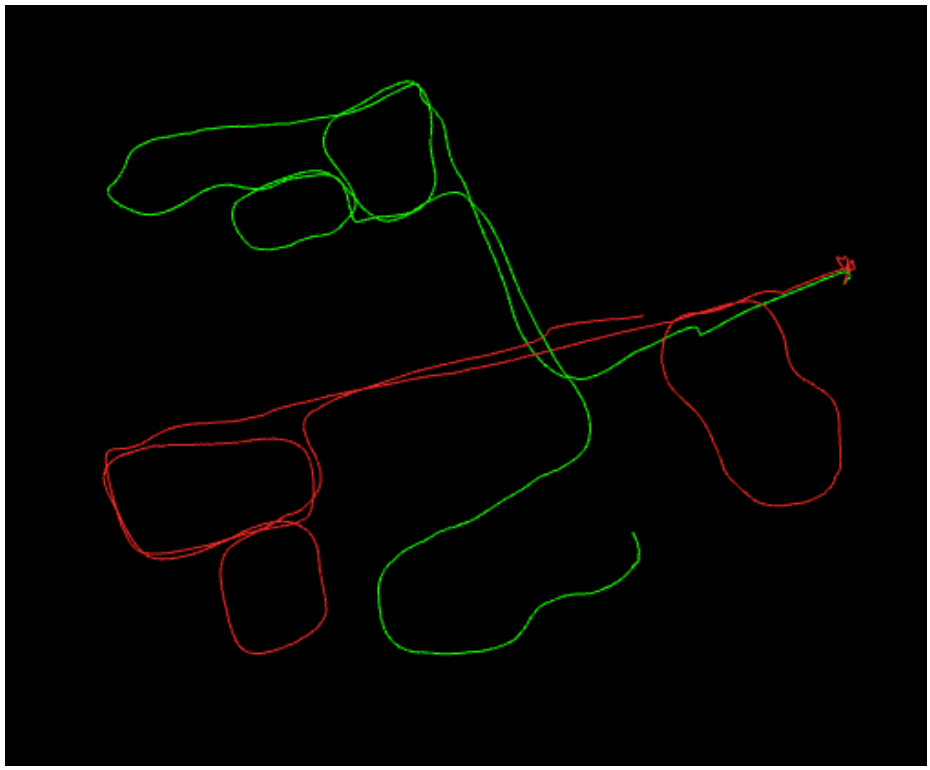


Figure 3-11: Unaligned trajectories

3.6 Conclusion and Future Work

3.6.1 Summary

At the beginning of this chapter, we stated the following three main objectives:

- Enable multiple robots to operate in the same environment.
- Allow the researchers to easily stream the sensor data from all robots as topics and messages in the Robotics Operating System (ROS).

- Create the infrastructure needed to separately and simultaneously control multiple robots.

In Section 3.4.2 we discussed how we enabled multiple robots to operate in the same environment. In Section 3.4.2 and 3.4.3 we discussed how the communication of multiple robots through the TESSE Python Interface and separated communication ports allowed streaming to ROS and sending commands back to TESSE for a multi-robot setup. And finally, in sections 3.4.5 and 3.4.4 we discussed how trajectory optimization and geometric controllers were used to separately and simultaneously control multiple robots in the same Unity environment. That means all the main objectives were met.

To further verify the functionality of the multi-robot system, in Section 3.5 we discussed and presented the performance of Kimera and Kimera-Multi using the multi-robot setup developed.

3.6.2 Improving data transmission efficiency

Sending image data over the network from Unity to Tesse is very slow. With a single robot transmitting stereo images only, the frame rate is around 30Hz. This rate drops to around 15 hz with two robots transmitting twice the image data. Obviously 15 Hz is very slow and needs to be sped up. Some of the approaches could be taken to improve the image transmission rates including rewriting the TESSE interfaces in C++ and fixing compression and gray-scale conversion on the Unity side.

The processing of the image data with Python takes significant bandwidth and may only be usable with a power-full computer. Rewriting the TESSE Interface in C++ would increase the efficiency significantly. And using compression for image transmission on the Unity is currently even less efficient than sending images uncompressed. Sending single channel (gray-scale) images is also less efficient then sending three-channel (RGB) images. Fixing the compression and gray-scale conversion processes are also expected to speed up the data rate significantly.

Chapter 4

Omnidirectional Stereo Interface for Kimera

4.1 Introduction

In this chapter, we will discuss the integration and testing of an omnidirectional stereo camera interface for Kimera-VIO. This work extends an existing implementation of monocular omnidirectional distortion camera model in Kimera-VIO by implementing and testing the stereo matching and rectification. In particular, while the monocular omnidirectional camera model has been implemented by Marcus Abate, a collaborator in the SPARK Lab, the extension to stereo cameras is a contribution of this thesis.

The omnidirectional distortion model is applicable to a broad range of cameras, including both omnidirectional cameras (which have 360 degree field of view) as well as wide field-of-view fisheye cameras. In our case, we tested the model on a panoramic fisheye stereo camera mounted on the SPARK Drone.

4.1.1 Motivation

The majority of open source VIN projects including Kimera are tested for low distortion, narrow field of view cameras. The narrow field of view presents some serious limitations for weight-constrained unmanned aerial vehicles. The first limitation is

the reduced ability to avoid obstacles. Regular pinhole cameras (planar projection cameras) provide a windowed view of the world which limits the observer's ability to see much of the surrounding environment therefore leading to less effective obstacle avoidance. For ground robots, it is possible to put multiple sensors on board as computation and payload are less constrained. However, that is not an option for small battery powered unmanned aerial vehicles (UAVs) due to computation and weight constraints. The maximum flight time of a battery powered UAV is normally around 10-30 minutes and maximum take off weight is typically around 2kgs. This means the on board computation is limited by both power consumption and weight and every extra sensor adds more weight and power requirement.

The windowed view of the world also limits the number of track-able features that are visible to the observer and that are used by VIN to estimate the trajectory of the robot. As the agent moves around, the length of the feature tracks decreases significantly. This makes the VIN system more susceptible to drift and even divergence if the agent maneuvers too quickly.

Those two limitations make planar projection cameras less suitable as a long term solution for Visual Inertial Navigation for UAVs. That's why leading UAV companies such as Skydio use wide field-of-view cameras [58]. This has presented an opportunity for the researcher to integrate omnidirectional stereo vision into Kimera-VIO.

4.1.2 Related Work

Camera Calibration Theory and Techniques

Camera calibration focuses on how to identify a model that describes the relationship between 3D real world coordinates and 2D images coordinates. The most straightforward model is that of the normal pinhole camera which is free of distortion. In the normal pinhole case, $x = \alpha PX$ fully describes the relationship between the 2D image point x and the pictured $3D$ world point X where P is a known projection matrix fully specified by the focal length and camera center. α is a scaling factor which can be resolved with two or more views picturing the same point X.

There has been significant research into modeling omnidirectional and wide field-of-view cameras which are characterized by severe distortion that might not be captured by the standard pinhole model. Some of the research reviewed for this thesis includes the Plenoptic Function and the Elements of Early Vision by Adelson and Bergen [9], which is one of the earliest research describing image formation. Later publications, including Spherical Imaging in Omnidirectional Camera Networks by Tosic and Frossard [61], describe how omnidirectional images captured by different types of mirrors or lenses can be uniquely mapped to spherical images, and present calibration methods that are specific to omnidirectional cameras.

More recently, the research community in computer vision, and robotics has produced a plethora of omnidirectional camera calibration techniques. Most of the techniques, including [15] and [6], use prior knowledge about the environment. Techniques including [26] compute calibration parameters without prior knowledge and work mainly for specific sensor types such as hyperbolic and parabolic mirrors or fisheye lenses. More recent research including Scaramuzza et al. [53] [52] as well as [66] and [7] apply to all kinds of central omnidirectional lenses. Wang et al. [63] discuss stereo calibration and rectification for omnidirectional multi-camera systems and [54] proposes a method for high-quality omnidirectional 3D reconstruction of augmented Manhattan worlds by joint depth optimization.

Omnidirectional SLAM

Rituerto et al. [45] studied monocular omnidirectional vision in Simultaneous Localization and Mapping (SLAM) problems and observed that the omnidirectional camera gives much better orientation accuracy, improving the estimated camera trajectory. Kim and Chung [29] presented a SLAM algorithm for an autonomous mobile robot using an omnidirectional stereo sensor and observed superior robustness to drift. Chang [16] developed a benchmarking suite for visual SLAM to study the feasibility of omnidirectional fisheye cameras for SLAM and compare them to the standard pinhole model. The reported results show improved tracking performance in traditional SLAM systems when using omnidirectional fisheye cameras compared the narrow field

of view pinhole model.

Additionally, Liu et al. [34] extended the monocular ORB-SLAM2 [39] to work with wide-angle fisheye cameras. They reported very high accuracy and improvements over the standard pinhole based monocular ORB-SLAM2 in small indoor environments.

The improvements in orientation accuracy observed by [34] [45] and the robustness mentioned by [29] motivated the implementation in this research. Our experimental analysis, presented later in this chapter, confirms these findings.

Feature Matching for Omnidirectional Cameras

High-distortion lenses create further challenges in stereo matching. The geometry based template matching used in [11] does not apply, and traditional 2D orientation based feature matching techniques struggle as well [43]. Researchers have been attempting to address this challenge by coming up with techniques that improve stereo correspondence finding for spherical or omnidirectional vision. Lee et al. [33] used proximity matrix and an SSD-based similarity matrix to find dominant corresponding feature pairs. Won et al. [64] proposed an end-to-end deep neural network model for omnidirectional depth estimation from a wide-baseline multi-view stereo setup. Resch et al. [43] proposed local image feature matching improvements for omnidirectional camera systems which refine the orientation of the features that traditional feature matching algorithms such as SIFT, SURF or ORB rely on. Their proposed method uses 3D orientations instead.

4.1.3 Chapter Outline

The rest of this chapter will discuss the implementation and integration of an omnidirectional stereo camera model in Kimera [49]. The experimental setups and results will be presented including comparison between the developed Kimera-VIO Omni, and the standard Kimera-VIO system, named Kimera-VIO Pinhole, which uses popular planar projection model. Finally, potential future improvements will be discussed.

4.2 Implementation

4.2.1 Implementation Choices

Sparse Descriptor Matching

Kimera relies on template matching [11] to establish stereo correspondences. Template matching matches image patches between the left and right camera image along the epipolar line. Therefore, it requires undistorted and rectified stereo images. This was not feasible for the omnidirectional model for two reasons: 1) Undistorting the entire image is computationally expensive. 2) Undistorting the entire image leads to information loss and prevents us from taking full advantage of the camera's field of view. To avoid these two issues, a sparse descriptor matching approach was chosen.

The sparse descriptor matching procedure follows three main steps: 1) keypoint detection, 2) descriptor extraction, 3) Descriptor matching. We first introduce our choice of keypoint detector and then describe each step below.

Keypoint Detector

The keypoint detection step attempts to locate the most describable features in the image. After detecting all the keypoints, they can then be ranked based on a feature cornerness score which permits the selection of the strongest keypoints through a procedure known as non maximum suppression.

There are many good feature detection algorithms including Scale Invariant Feature Transform (SIFT) [35] [35], Oriented FAST and Rotated BRIEF (ORB) [50], and Speeded Up Robust Features (SURF) [8] which are included in major Computer Vision libraries including OpenCV. However, one of the challenges is that most of those algorithms do not provide the user with any control over where the detected keypoints are located. It is usually undesirable to have all of the keypoints close to each other as this can hinder the tracking performance, as discussed in [57]. To avoid this issue, we used OpenCV's implementation of Good Features to Track [57], which allows specifying a minimum distance between keypoints to space them apart.

Sparse Stereo Matching Implementation

Keypoint detection: The sparse stereo matcher module expects a stereo frame with the left frame already populated by the tracker module. The tracker module performs optical-flow operations followed by filtering steps to detect a number of trackable keypoints in the left frame. Those keypoints are used for the monocular Kimera and the stereo needs to detect corresponding keypoints on the right frame. However, the keypoint data structure populated on the left frame is OpenCV *point2f*. Essentially, much of the information describing a keypoint including the size, orientation, and response are dropped. This design decision in Kimera was made based on the fact that for each keypoint all of those components change as the camera moves around in subsequent optical-flow steps. Unfortunately, the retained information which is just the 2D pixel location is insufficient for descriptor computation and stereo matching.

The first step was to re-detect the same left keypoints that came with the stereo frame. The tracked left frame keypoints can not be modified so the re-detected keypoints are kept separate in a one-to-one index mapping and they are only used for finding matching keypoints on the right.

We observed that the keypoint detector does not always detect the same keypoints. The first attempted approach was to detect more features than the ones that are tracked and then search the coordinates of each of the tracked keypoints in the re-detected keypoints. When a tracked keypoint is found, it is stored for matching. And if a keypoint is not found, a new keypoint with the coordinates of the tracked keypoint, with size of zero, and with OpenCV default values for orientation, response and other parameters is stored for matching. Note that in the second case, with the size of zero, that keypoint has a very small chance of being matched against the right camera image.

This approach was partially successful, but struggled at times. The second approach was to use OpenCV masks to force the keypoint detector to only detect keypoints in circular areas around the tracked keypoints. The diameter of the circular

region around the keypoint was set to be twice as the minimum distance between keypoints.

Finally, the right keypoints were also detected. To increase the chance of detecting the tracked features on the right frame, a larger number of right keypoints was detected. Detecting twice as many right keypoints as the number of left keypoints gave the best performance.

Descriptor Computation: The OpenCV implementation of ORB is used to compute the descriptors for both the left and right keypoints. Since we maintain the original tracked keypoints on the left frame, it is important that the left keypoints used for descriptor computation map one-to-one to those tracked left keypoints by index. The OpenCV *Feature2d* compute function, which is used to compute the descriptors, sometimes removes keypoints if meaningful descriptors cannot be computed. To maintain the same number of keypoints, additional keypoints are detected and added.

This feature was throwing off the one-to-one mapping of the detected and tracked keypoints which led to wrong associations when populating the right frame. To solve this problem, we search for the left keypoints among the newly detected features and remap them to the original tracked features by creating a hash table that maps the indices of the corresponding keypoints.

Descriptor matching and filtering: K-nearest neighbors approach was used for descriptor matching. For each left descriptor, the top three candidate matches were returned. The left keypoint and the three candidate matches are undistorted and rectified. The undistort and rectification steps are described in section 4.2.2. Finally, a epipolar constraint check was performed to see if any of the candidates pass geometric verification. Only the matching candidates that passed the epipolar check are used to populate the right frame and kept for further processing in the pipeline.

Depth Computation: From the stereo matches, we can compute the depth for each keypoint. Towards this goal, we used the same approach already in Kimera, but the minimum and maximum depth distances are slightly adjusted to account for

inaccuracy in the distortion model.

4.2.2 Calibration

Calibration Technique

For this research, we chose a general calibration model that supports any kind of omnidirectional sensor type to allow future researchers to pick new omnidirectional sensors without reimplementing the distortion model. It was also important to select a model that was shown to work well in practice and has an easy to use implementation. Towards that goal, the model proposed by Scaramuzza et al. [53] [52] was used for this work as it is shown to work well in practice and has a well documented toolbox in Matlab.

Scaramuzza's Omnidirectional Model

Camera calibration consists in identifying a model that describes the relationship between 3D real world coordinates and 2D image coordinates. The most straightforward model is that of the normal pinhole camera which is free of distortion:

$$x = \alpha PX \tag{4.1}$$

In the normal pinhole case, (4.1) fully describes the relationship between the 2D image point x and the pictured $3D$ world point X where P is a known perspective projection matrix. P is fully specified by the focal length and camera center and α is a scaling factor which can be resolved with two or more views picturing the same world point X.

According to Scaramuzza's omnidirectional model, we have a non-linear imaging function g that transforms a 3D half-ray emanating from viewpoint X to pixel point x:

$$\lambda g(x') = \lambda g(Ax + t) = PX, \lambda > 0 \tag{4.2}$$

in the previous equation x' is the projection of X onto the sensor plane (see figure 1

in ([53]) and $x' = Ax + t$ is an affine transformation from image plane to sensor plane. Again, λ is the scale factor which can be resolved with two or more view points. The job of omnidirectional calibration is to estimate A and t and g. $g(x')$ can be written as follows:

$$g(u', v') = (u', v', f(u', v'))^T \tag{4.3}$$

u' and v' are the coordinates of x' on the sensor plane. The final general assumption is that g is rotationally symmetric about the sensor axis. This assumption means that f is only related to x' through $\rho' = \|x'\| = \sqrt{u'^2 + v'^2}$.

Different calibration approaches differ on what form of f to use based on the lens or mirror construction. Scaramuzza's model is designed for general central lens or mirror. The model uses a Taylor series approximation of f, which results in the following polynomial form:

$$f(u', v') = a_0 + a_1\rho' + a_2\rho'^2 ... + a_n\rho'^n \tag{4.4}$$

The coefficients $a_0, .., a_n$, the order of the polynomial n (typically chosen to be either 4 or 5), A and t are what the calibration procedure needs to estimate. This means Equation (4.2) becomes the following expression:

$$\lambda \begin{bmatrix} Ax + t \\ a_0 + a_1\rho' + a_0 + a_2\rho'^2 ... + a_n\rho'^n \end{bmatrix} = PX \tag{4.5}$$

The calibration estimates the polynomial coefficients $a_0, .., a_4$ and the polynomial order n separately from A and t. To recover X given the estimated coefficients and the order, (4.2) is written as follows:

$$\lambda \begin{bmatrix} u' \\ v' \\ a_0 + a_1\rho' + a_0 + a_2\rho'^2 ... + a_n\rho'^n \end{bmatrix} = X \tag{4.6}$$

Note that the projection matrix P is dropped and the sensor coordinates are used. P

is not needed as it only acts as scale which λ accounts for and (u', v') can be recovered from the pixel coordinates (u, v) as follows:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} c & d \\ b & 1 \end{pmatrix} \begin{pmatrix} u' \\ v' \end{pmatrix} + \begin{pmatrix} c_x \\ c_y \end{pmatrix} \tag{4.7}$$

$\begin{pmatrix} c & d \\ b & 1 \end{pmatrix}$ and $\begin{pmatrix} c_x \\ c_y \end{pmatrix}$ are called the stretch matrix and omni center and are both estimated by the calibration procedure. The order of the polynomial f is set to 4 and a_1 is always zero [65].

Calibration Procedure

The initial calibration procedure is straightforward using the Matlab Calibration toolbox. The toolbox takes a small set of calibration images showing a checkerboard as well as the dimensions of the checkerboard. In practice, the calibration procedure always captures the distortion of some regions of the image better than others. It took meticulous refinement to produce satisfactory results. Figure 4-3 shows that after our initial calibration many keypoint matches were discarded since they led to negative depth estimates or failed to meet depth constraints set in Kimera. Those keypoints are mainly concentrated in one region of the image. To improve the calibration and attempt to capture that region better, more images where the calibration target is in that region were taken and then the calibration procedure was repeated.

Another way to visualize the calibration performance is to look at the per image mean reprojection errors and see which images show high reprojection errors. We may then want to take some pictures that are close to the images with higher mean reprojection errors. Figure 4-1 shows the per image mean re-projection errors for multiple calibration refinements. For each step, Figure 4-2 shows the number of stereo matches passing the depth constraints (green), the number of stereo matches that have negative depth (red), and the number of stereo matches that failed minimum or maximum depth constraints.

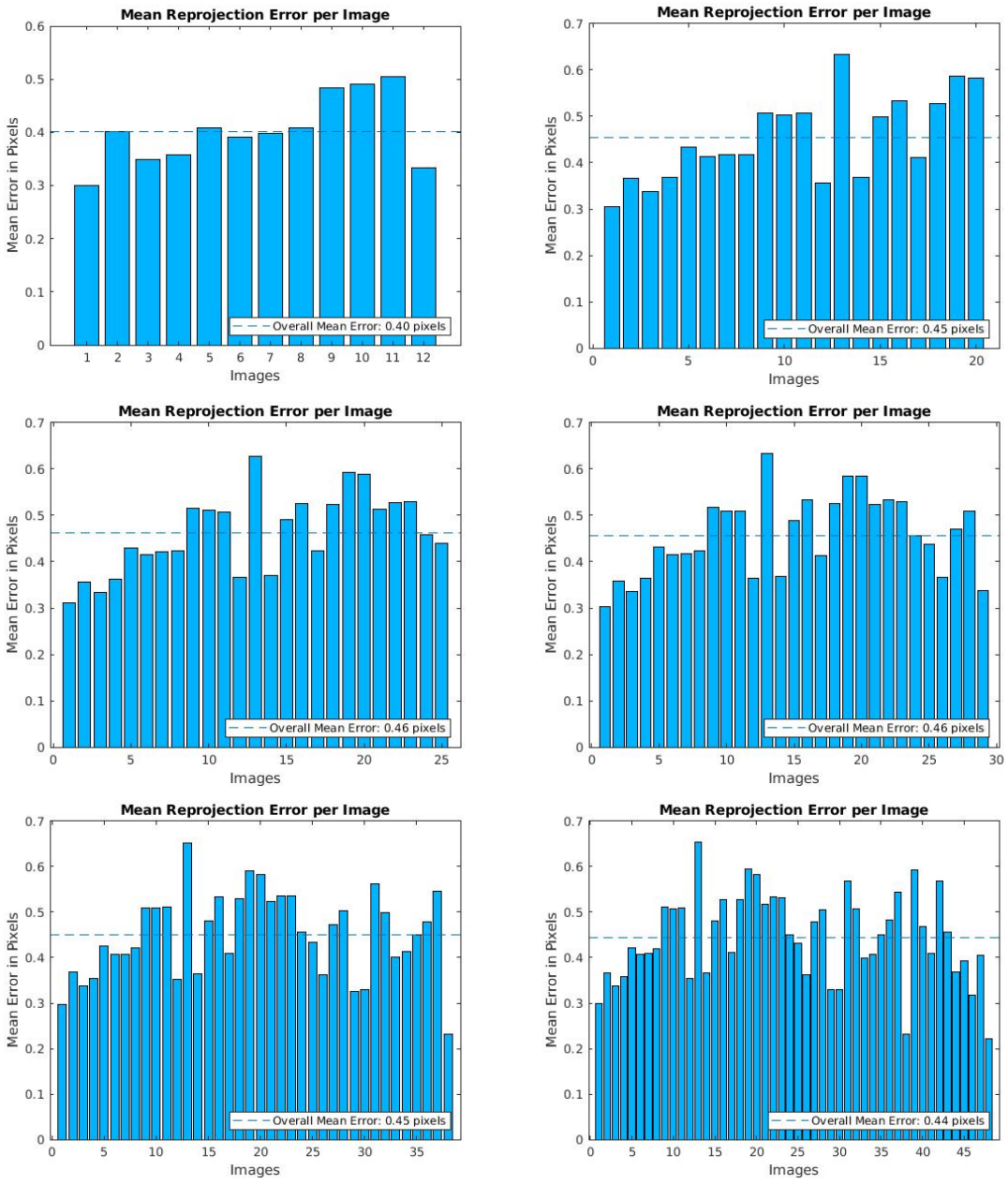


Figure 4-1: Mean reprojection errors per image for multiple calibration refinement

For each calibration iteration, the mean reprojection error per image and the number of valid keypoints that are failing the depth constraints are considered to decide where to add the additional images. Note that the number of matches that are passing the depth constraint checks and the total number of keypoints that are passing the epipolar check are gradually increasing 4-2. The number of valid keypoints that are failing the depth constraint (not shown here) are also decreasing. We remark

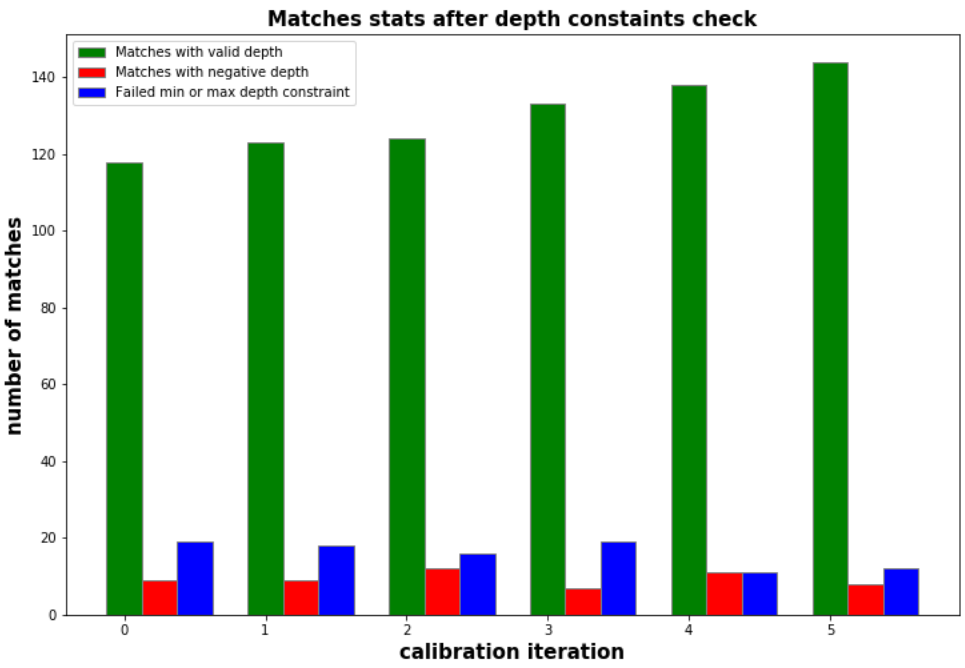


Figure 4-2: Number of stereo matches passing the depth constraint check (green), number of stereo matches with negative depth (red), and number of matches failing depth constraints (blue).

that - while we observe good performance after several calibration refinements - we believe that the calibration process is made more challenging because of the choice of camera. The RealSense T265 uses fisheye lenses which can not be described as central omnidirectional cameras as they do not necessarily satisfy the single view point property [5]. Scaramuzza et al. [53] [52] demonstrated their model works well for central omnidirectional cameras, but made no claims that it may work for non-central omnidirectional cameras.

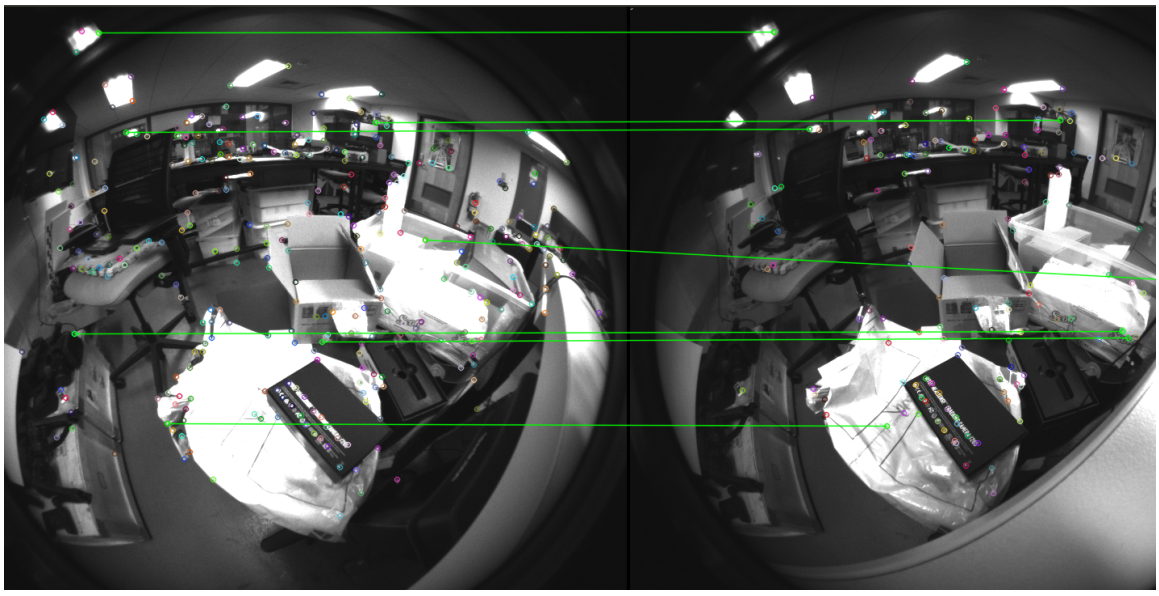


Figure 4-3: This figure shows that after our initial calibration some left to right keypoints failed depth constraints while being valid matches. This issue was mitigated by repeating several calibration iterations. Note that not all of the shown keypoint matches are valid. Some of them are correctly filtered out as wrong matches, but a few of them in one region are actually valid matches that are wrongly filtered.

4.2.3 Parameter Tuning

IMU Noise Parameters

Kimera-VIO worked better with high IMU noise parameters. Two experiments were conducted where in the first one the cameras were hand held and the second the camera was mounted on a flying drone. The noise density and random walk noise parameters were increased by two orders of magnitude for the first experiment and more than four orders of magnitude for the second experiment from the manufacturer provided IMU parameters. Table 4.1 shows the parameters settings for each experiment as well as the default parameters.

IMU noise parameters			
parameter/setting	Default	Hand Held	Flying
Gyro Noise Density	5.148e-6	5.148e-4	5.148e-2
Gyro Random Walk	5.0e-7	5.0e-5	5.0e-3
Accelerometer Noise Density	6.695e-5	6.695e-3	6.695e-1
Accelerometer Random Walk	1.0e-5	1.0e-3	1.0e-2

Table 4.1: IMU noise default parameters and parameter settings for the hand held and flying drone experiments.

Frontend Parameter Turning

The following frontend parameters were tuned to get a better performance from Kimera-VIO Omni.

- **Min distance between features:** The best performance in Kimera-VIO Omni was achieved with the distance between features set to 15.

- **Ransac outlier rejection threshold:** The outlier rejection threshold for Ransac Mono was increased by an order of magnitude from the default value of $10e-6$ to $10e-5$ to account for the inaccuracy in the calibration.

- **Number of tracked features:** The best results were observed when the

maximum number of tracked features were set to 300 with non maximum suppression enabled.

Table A.1 in Appendix shows the complete Kimera-VIO Omni frondend parameter settings and the default (master branch) parameter settings. Complete backend parameter settings for both master and the omnidirectional Kimera-VIO can be found in Table A.2.

4.3 Experimental Setup and Results

This section discusses the experiments conducted to test the performance of Kimera-VIO Omni. The first experiment was to compare the performance of the omnidirectional Kimera-VIO to that of the pinhole model. The second experiment was to compare the omnidirectional Kimera-VIO trajectory estimate to that of a commercial localization system. In particular, we compared Kimera-VIO Omni with the visual-inertial odometry provided by the RealSense T265. The following two subsections discuss the experimental setup for each case and present relevant results.

4.3.1 Comparison with the Realsense T265 Camera Tracking and Vicon

Experimental Setup

As Figure 4-4 shows, the Intel RealSense T265 was mounted on the SPARK Drone with Vicon markers. A two minute flight with aggressive maneuvering was conducted in a Vicon room. The motion capture system can precisely track the position and the orientation of the drone using the motion capture markers to fully determine the translation and the rotation of the drone in global coordinates. The Intel RealSense T265 can also track its position, orientation, and velocity with high accuracy.

During the flight, ROS bag file of the T265 fisheye stereo images, IMU data, and odometry as well as the Vicon determined drone position and orientation data were collected. The Vicon markers and the T265 mounting are shown in Figure 4-4.

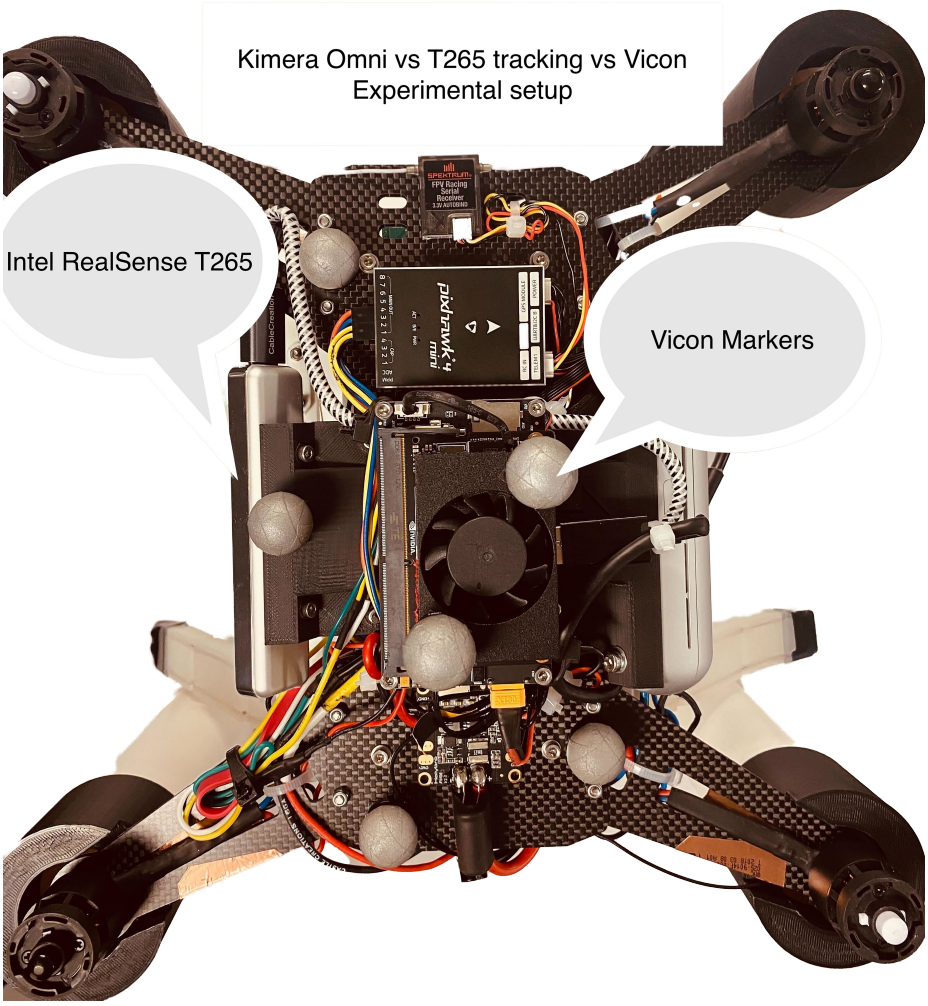


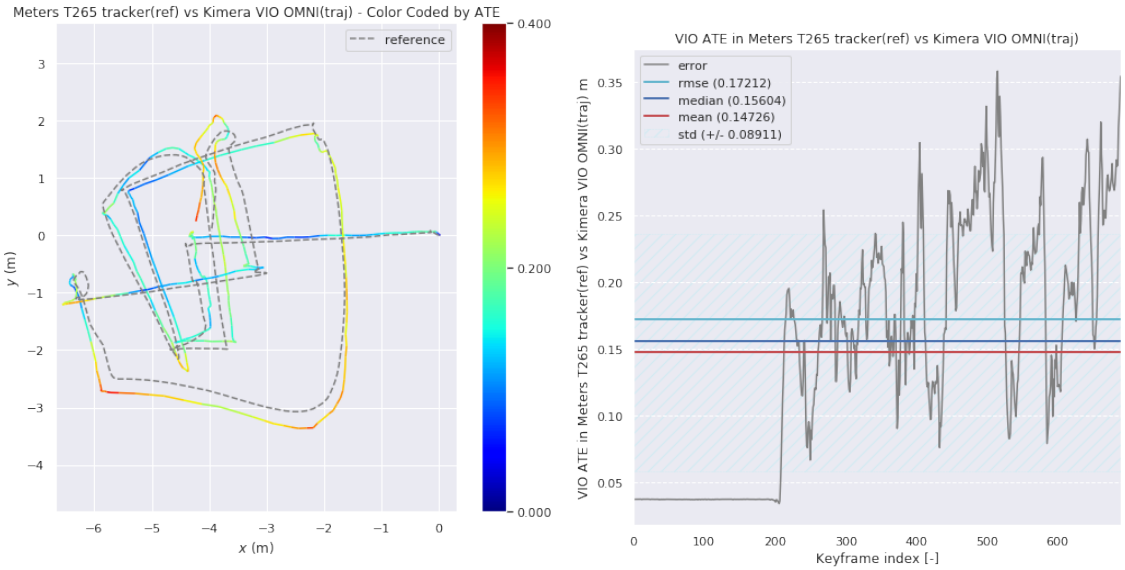
Figure 4-4: Drone setup.

The IMU and image data were fed into Kimera to estimate the trajectory of the drone offline. First the Vicon tracking was compared against the T265 odometry and Kimera-VIO was compared to the T265 odometry. Unfortunately, the Kimera-VIO tracking was not compared to the Vicon tracking due to unresolved alignment issues. The results for both comparisons are presented in the following section.

Results

Figure 4-5 shows the results of the comparison between T265 odometry and Kimera-VIO Omni. Overall, Kimera-VIO shows little drift with root mean squared error (rmse) of 0.172 meters (see Figure 4-5(b)) in aligned absolute translation error. Other

statistics on the translation errors are also shown on Figure 4-5(b). To confirm the stability of the T265 tracking, it was compared to that of an external motion capture. Figure 4-6 shows the comparison of the T265 odometry to the ground truth provided by the Vicon motion capture system. From Figure 4-6, we can see the T265 tracking does not diverge from the ground truth position of the drone, measured by Vicon.



(a) T265 odometry(dashed) vs Kimera Omni Trajectory (color)

(b) Translation Errors

Figure 4-5: Figure (a) shows the trajectory (colored) of Kimera-VIO with the Omni-directional camera model and the odometry of the RealSense T265(dashed). Figure (b) shows the absolute translation error of the aligned Kimera-VIO Omni trajectory with respect to the T265 odometry. This data was collected with a SPARK Drone in a Vicon Room. The accuracy of the T265 was benchmarked against the Vicon position which shown in Figure 4-6

4.3.2 Comparing to Kimera Pinhole

Experimental Setup

As Figure 4-7 shows, the Intel RealSense T265 and the D455 were mounted on a carbon fiber plate. The former is a wide field-of-view camera, while the latter is a standard camera with a narrower field of view. Two experiments with varying complexity were conducted. In both experiments, the researcher walked with the

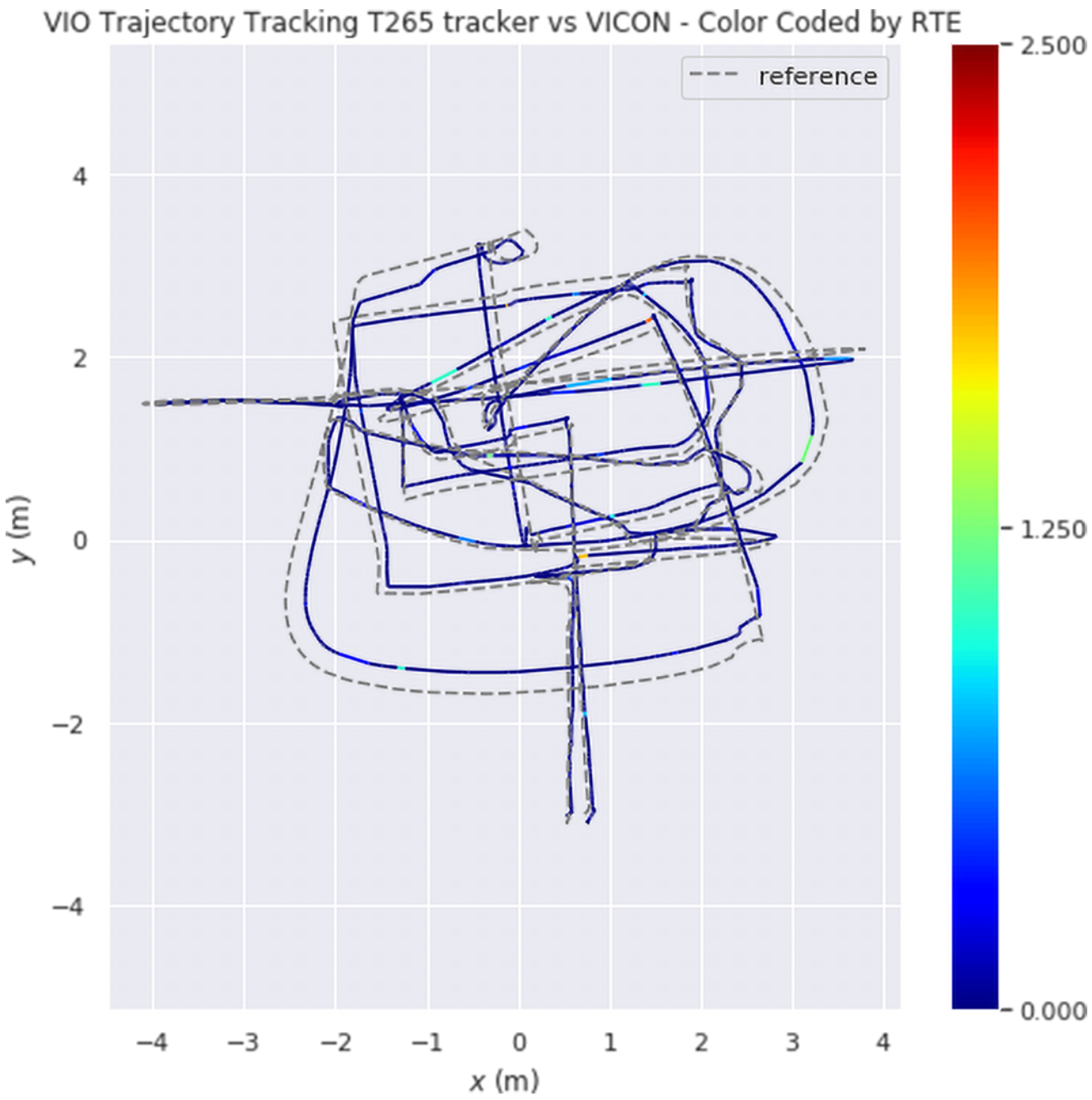


Figure 4-6: This figure shows the trajectory estimate(colored) of the drone by the T265 odometry vs the ground truth (dashed) trajectory of the drone from Vicon.

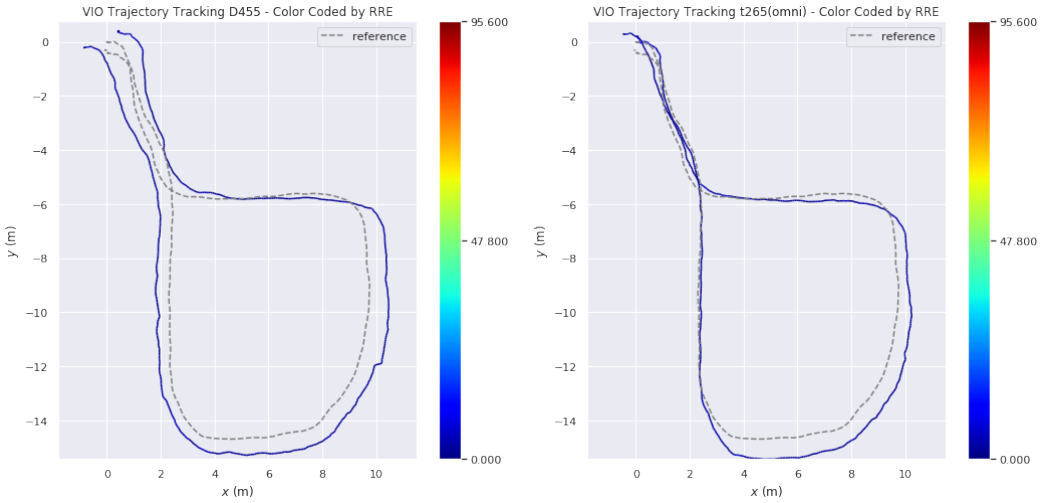
cameras handheld facing forward while collecting the IMU and image data from each camera as well as the odometry estimate from the T265. In both experiments, the Fisheye images and IMU data recorded from of the T265 were fed into Kimera-VIO Omni to estimate the trajectory of the cameras and the planar projection images

Figure 4-7: Hand held experiment setup.

and the IMU data collected from the D455 were fed into Kimera-VIO Pinhole to also determine the trajectories of the cameras.

Finally, the Kimera trajectory estimates were compared to those of the T265 odometry estimate and the relevant results are presented in the following subsection. Figure 4-7 shows the two camera mounting configurations used for this experiment.

Results



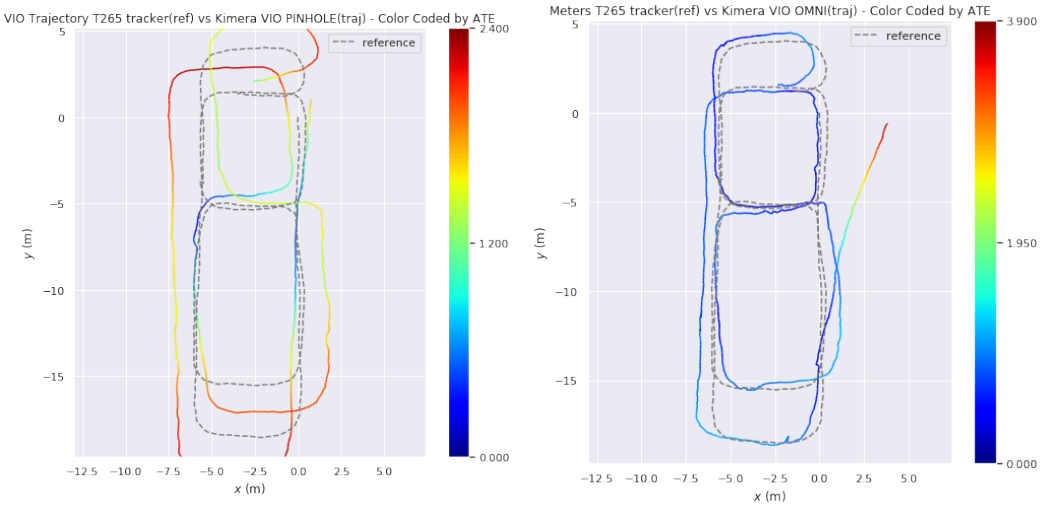
(a) T265 odometry (dashed) vs Kimera-VIO Pinhole Trajectory (color)

(b) T265 odometry(dotted) vs Kimera Omni Trajectory (color)

Figure 4-8: Figure (a) shows the trajectory (color) of Kimera-VIO with the pinhole distortion model versus the Intel RealSense T265 odometry (dashed). Figue (b) shows the trajectory (color) of Kimera-VIO with the omnidirectional distortion model versus the Intel RealSense T265 odometry (dashed). In general the Omni showed less drift than the Pinhole model.

For this comparison, two experiments were conducted. For the first experiment shown in Figure 4-8, data was collected by walking with the cameras from room 31-219 and around the lounge area of the floor 2 of building 31 at MIT in a single loop. Overall, Kimera performs well compared to the ground truth (T265) odometry with Kimera Omni qualitatively showing slightly better performance.

For the second experiment, data was collected by walking with the camera in a larger and more complex space with 20 office cubicles. In this experiment, Kimera Omni again achieves slightly better performance, in particular in terms of translation error. For this experiment, Figures 4-9 shows a side-by-side view of the Kimera trajectory estimates (colored) versus the T265 odometry. On this trajectory, which is over 200-meters long, both versions of Kimera-VIO perform robustly on trajectory tracking with loop closure disabled. However, Kimera-VIO Omni (Figure 4-9(b)) does better than the standard pinhole version. Figure 4-10 shows the plots of the absolute translation errors for Kimera-VIO pinhole (Figure 4-9(a)) and Kimera-VIO



(a) T265 Odometry (dashed) vs Kimera-VIO Pinhole Trajectory (color)

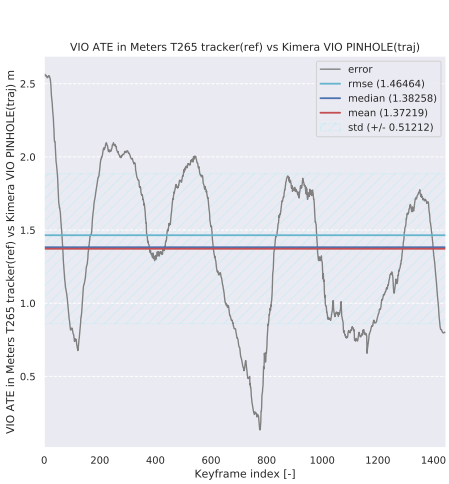
(b) T265 Odometry(dotted) vs Kimera Omni Trajectory (color)

Figure 4-9: Figure (a) shows the trajectory (color) of Kimera-VIO with the pinhole distortion model versus the Intel RealSense T265 odometry (dashed). Figue (b) shows the trajectory (color) of Kimera-VIO with the omnidirectional distortion model versus the Intel RealSense T265 odometry (dashed). The Omni model shows significant drift in the first long hallway but stays close to the reference (T265) odometry for most of the way. On the other hand, Kimera with the Pinhole camera model shows continuous drift through the trajectory.

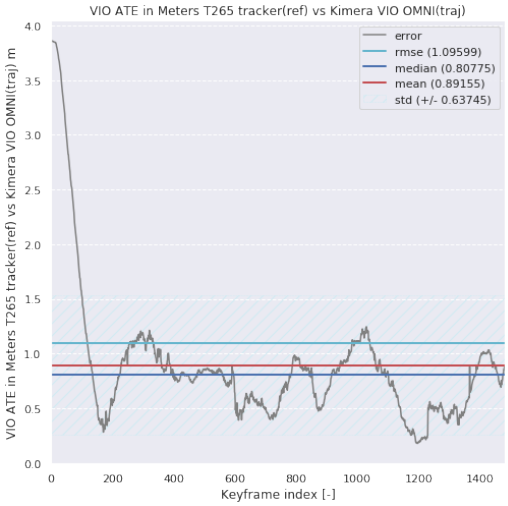
omni (Figure 4-9(b)), and the error statistics for Figure 4-9 are summarized in Table 4.2. The root mean squared error (rmse) of the Kimera-VIO Omni tracking is about 25% lower than that of the standard pinhole version.

Error Metric	Kimera-VIO Pinhole	Kimera-VIO Omni
RMSE	1.465	1.096
Mean	1.383	0.808
Median	1.372	0.892

Table 4.2: Error statistics for Kimera-VIO Pinhole and Kimera-VIO Omni



(a) Pinhole Translation Errors



(b) Omni Translation Errors

Figure 4-10: Figure (a) is showing the translation errors of Kimera-VIO using the low distortion pinhole model with respect to the odometry of the RealSense T265. Figure (b) is showing the translation errors of Kimera-VIO using the high distortion omnidirectional camera model with respect to the odometry of the RealSense T265. Relevant error metrics including residual mean squared error (rmse), median, mean, and standard deviation of errors are displayed within each plot.

4.4 Future work

While the performance of the omnidirectional stereo interface for Kimera-VIO was satisfactory within the goals of this research, there is still room for improvement to fully realize the potential of omnidirectional vision for Kimera and similar SLAM platforms. Those improvements can be done in camera calibration as well as in feature detection and matching. The following two subsections propose potential approaches in improving feature matching and camera calibration.

4.4.1 Improving the Feature Matching

Traditional feature detection and matching algorithms such as SIFT, SUFT, and ORB rely on orientation invariance of the features. As discussed in [43], image distortion and non-monotonic mapping from camera rotations to image rotations makes those traditional orientation based descriptors unsuitable for feature matching for omnidirectional images. Those limitations have also been observed during this work. Figure

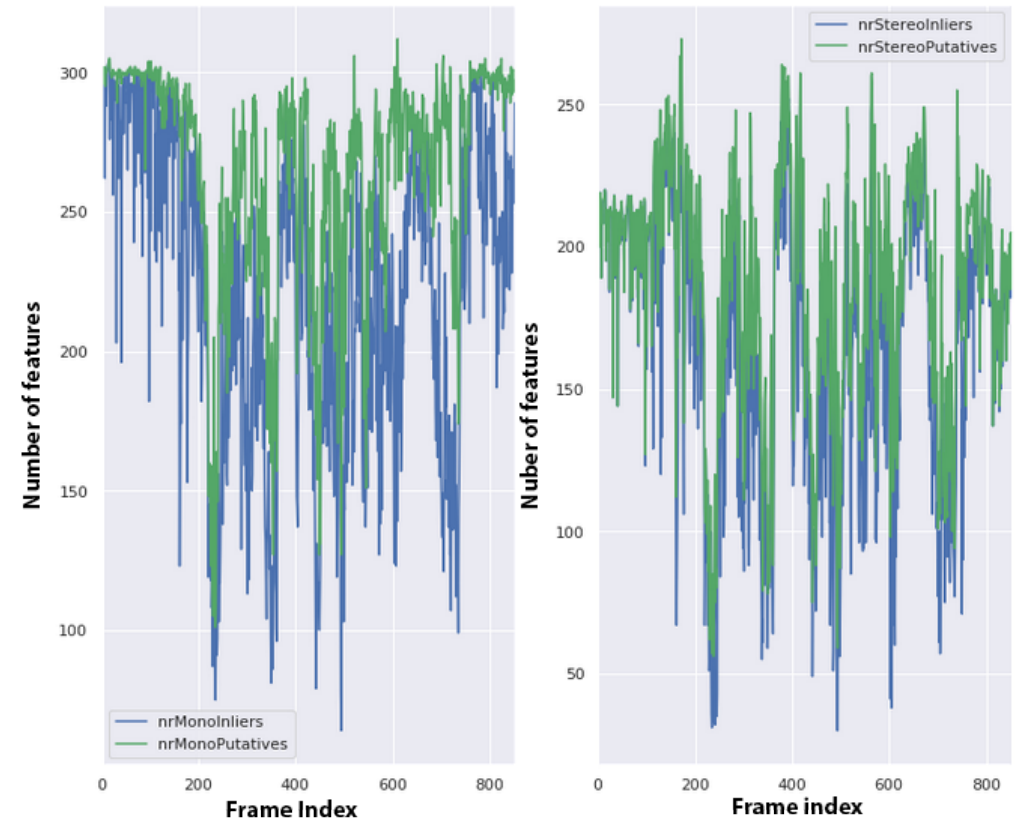
4-11 shows the side by side view of the number of features tracked and the number of stereo matches before and after RANSAC outlier rejection for Kimera-VIO pinhole (4-11a) and Kimera-VIO Omni (4-11b). Both the pinhole and omni model use Good Features to Track [57] for keypoint detection and ORB [50] for feature matching. However, both feature tracking and stereo matching are more successful in the pinhole model case.

There are multiple potential areas of exploration to improve the feature tracking and matching including [43] and similar research, but we believe the most promising approach is using machine learning algorithms.

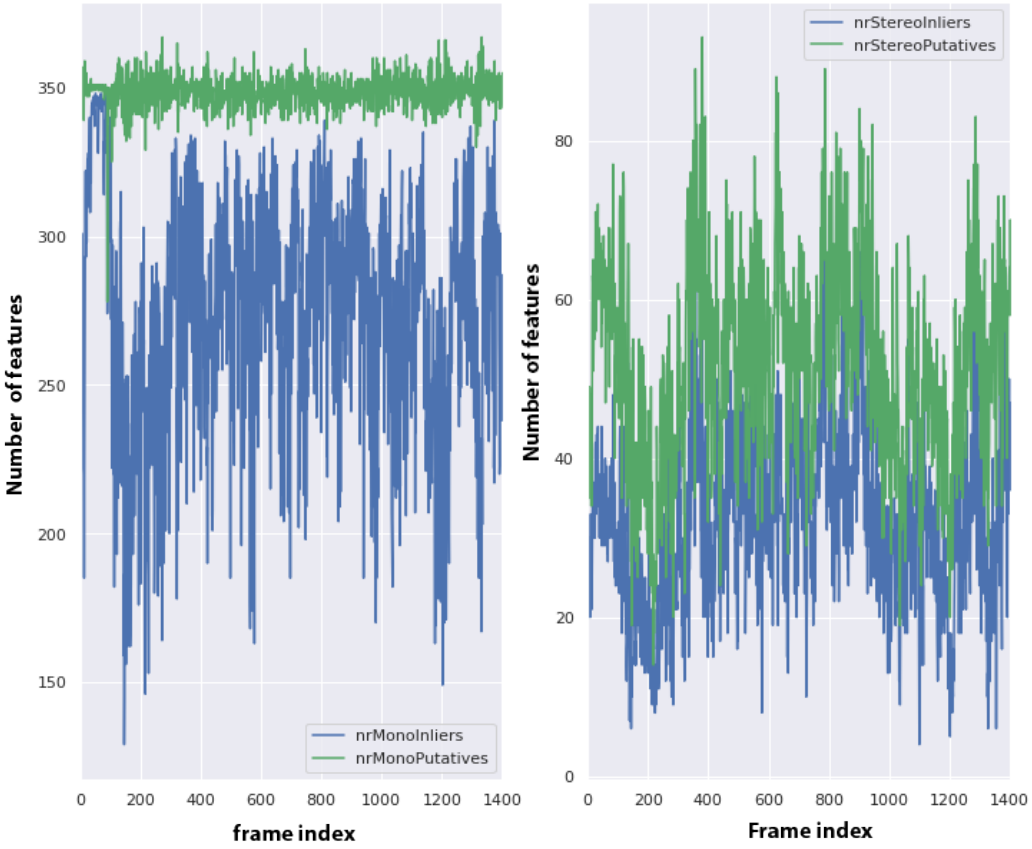
There are successful end-to-end feature detection and matching approaches out there including [56] and [25] that attempt to learn to extract a highly distinguishable set of matching correspondences from pairs of images. There are also joint correspondence optimization based methods including [51] that try to find the optimal set of matching correspondences from sets of pre-detected keypoints.

4.4.2 Improving the Camera Calibration

Scaramuzza's model worked well enough to show the advantages of a wide field-of-view camera for SLAM over the standard pinhole model. Unfortunately, the calibration was never good enough to achieve the stereo matching quality of the pinhole model. Even with our best efforts to refine the calibration, a number of valid stereo correspondences were still producing negative depth due calibration inaccuracy. Improving the calibration, whether it requires adopting a new calibration model and tools or coming up with a better way to refine the estimates from the matlab toolbox, would be an impactful project for future researchers interested in omnidirectional vision for Kimera.



(a) Kimera-VIO pinhole feature tracking and matching



(b) Kimera-VIO omni feature tracking and matching with T265

Figure 4-11: Feature tracking and matching.

Chapter 5

Conclusion and Future Work

5.1 Conclusions

This thesis focused on the design and implementation of simulation and hardware platforms to test simultaneous localization and mapping (SLAM) systems. A useful simulation platform needs to capture the robot dynamics and the sensor modalities typically used for SLAM (in our case, we focus on visual-inertial SLAM). A suitable hardware platform (an unmanned aerial robot in our case) needs to have the capability to navigate, sense the environments, and use onboard computers to run the software it was designed for; such platforms might also need sensor interfaces that allow adapting and testing algorithms on novel sensors.

This thesis includes three contributions: 1) The development of a hardware platform consisting of a real drone capable of running state-of-art metric-semantic SLAM. 2) the development of a multi-robot simulation platform that includes dynamically accurate, photo-realistic drones. 3) the integration of an omnidirectional stereo frontend in Kimera, a state-of-the-art system for visual-inertial localization and mapping.

In particular, Chapter 2 presented the design and prototyping of a new drone platform, the SPARK Drone. The drone has the computational and sensing capabilities needed to run SLAM systems like Kimera on board, and payload range to carry over 1kg of sensors and a soft gripper. Chapter 3 described the development of the multi-robot simulation infrastructure and provided an experimental evaluation of Kimera

and its multi-robot counterpart (Kimera-Multi) in our simulator. The simulator allows multiple robots to operate in the same environment, enables communication between robots through the TESSE Python Interface, and allows streaming sensor data and receiving commands via the Robot Operating System (ROS). The simulator also integrates algorithms for trajectory optimization and geometric control. Chapter 4 described how to interface Kimera with an omnidirectional camera and reports on the implementation of an omnidirectional stereo frontend in Kimera. The chapter also provides an evaluation of Kimera using the omnidirectional frontend, showing its effectiveness in trajectory estimation.

5.2 Future Work

Even though all three contributions have led to platforms and implementations achieving satisfactory performance, there is still room for improvement. For the SPARK Drone, the last section of Chapter 2 discussed how the performance of the propulsion system could be improved to increase the endurance and payload of the drone. For the multi-robot system, the conclusion and future work section of Chapter 3 discussed how the image transmission rate over the TCP/UDP ports could hinder the performance of the simulation platform especially when more than two robots are integrated; the section also suggests computational improvements to speed up the image-processing steps involved in the image transmission. For the omnidirectional camera frontend, in the conclusion and future work section of Chapter 4, we discussed how the stereo matching and feature tracking in the omnidirectional version of Kimera — while leading to more robust trajectory estimates— are performing worse than the standard (pinhole) camera model in Kimera. Potential improvements can be achieved by adopting other types of visual features or using novel learning-based approaches for feature matching to replace the traditional descriptor-based matching used in this thesis.

Appendix A

Omnidirectional Interface

A.1 Frontend Parameters

parameter	Master	Omni
klt_win_size	24	24
klt_max_iter	30	30
klt_max_level	4	4
klt_eps	0.1	0.1
maxFeatureAge	25	25
feature_detector_type	3	3
maxFeaturesPerFrame	300	300
quality_level	0.001	0.001
min_distance	20	20
block_size	3	3
use_harris_detector	0	0
k	0.04	0.04
scale_factor		1.2
nlevels		8
edge_threshold		31
first_level		0

WTA_K		2
score_type		0
patch_sze		31
fast_threshold		20
fast_thresh	10	10
equalizeImage	0	0
nominalBaseline	0.11	0.11
toleranceTemplateMatching	0.15	0.15
templ_cols	101	101
templ_rows	11	11
stripe_extra_rows	0	0
minPointDist	0.5	0.5
maxPointDist	10	10
bidirectionalMatching	0	0
max_nr_keypoints_before_anms	2000	
enable_non_max_suppression	1	1
non_max_suppression_type	6	4
nr_horizontal_bins	7	
nr_vertical_bins	5	
binning_mask	[]	
enable_subpixel_corner_finder	1	1
max_iters	40	40
epsilon_error	0.001	0.001
window_size	10	10
zero_zone	-1	-1
subpixelRefinementStereo	0	0
useSuccessProbabilities	1	1
useRANSAC	1	1

minNrMonoInliers	10	10
minNrStereoInliers	5	5
ransac_threshold_mono	1e-06	1e-4
ransac_threshold_stereo	1	1
ransac_use_1point_stereo	1	1
ransac_use_2point_mono	1	1
ransac_max_iterations	100	100
ransac_probability	0.995	0.995
ransac_randomize	0	0
intra_keyframe_time		0.2
min_intra_keyframe_time	0.2	
max_intra_keyframe_time	5.0	
max_disparity_since_lkf	1000	
minNumberFeatures	0	0
useStereoTracking	1	1
disparityThreshold	0.5	0.5
optical_flow_predictor_type	1	1
use_2d2d_tracking	1	1
2d2d_algorithm	1	1
optimize_2d2d_pose_from_inliers	0	0
use_3d3d_tracking	1	1
optimize_3d3d_pose_from_inliers	0	0
use_pnp_tracking	0	false
pnp_algorithm	3	3
min_pnp_inliers	20	20.0
ransac_threshold_pnp	1.0	1.0
optimize_2d3d_pose_from_inliers	0	0

Table A.1: Complete list of frontend parameters

A.2 Backend Parameters

parameter	Master	Omni
backend_modality	0	0
autoInitialize	0	0
roundOnAutoInitialize	0	0
initialPositionSigma	1e-05	1e-05
initialRollPitchSigma	0.174533	0.174533
initialYawSigma	0.00174533	0.00174533
initialVelocitySigma	0.001	0.001
initialAccBiasSigma	0.1	0.1
initialGyroBiasSigma	0.01	0.01
linearizationMode	0	0
degeneracyMode	1	1
rankTolerance	1	1
landmarkDistanceThreshold	10	10
outlierRejection	3	3
retriangulationThreshold	0.001	0.001
smartNoiseSigma	3.0	3.0
monoNoiseSigma	1.8	1.8
monoNormType	2	2
monoNormParam	4.6851	4.6851
stereoNoiseSigma	1.8	1.8
stereoNormType	2	2
stereoNormParam	4.6851	4.6851
regularityNoiseSigma	0.03	0.03
regularityNormType	1	1
regularityNormParam	0.04	0.04
addBetweenStereoFactors	0	0

betweenRotationPrecision	0	0
betweenTranslationPrecision	100	100
relinearizeThreshold	0.01	0.01
relinearizeSkip	1	1
zero_velocity_precision	1000	1000
no_motion_position_precision	1000	1000
no_motion_rotation_precision	10000	10000
constant_vel_precision	100	100
numOptimize	1	1
nr_states	25	
horizon		6
wildfire_threshold	0.001	0.001
useDogLeg	0	0
pose_guess_source	0	0
mono_translation_scale_factor	0.1	0.1

Table A.2: Complete list of backend parameters

Bibliography

[1] 16.485 - VNAV-Visual Navigation for Autonomous Vehicles (VNAV). *MIT Course*, 2021.

[2] 6.141/16.405 - robotics: Science and systems course (RSS): MIT course. https://github.com/mit-rss, 2021.

[3] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord, and Sertac Karaman. The blackbird dataset: A large-scale dataset for uav perception in aggressive flight, 2018.

[4] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): part ii. *IEEE Robotics Automation Magazine*, 13(3):108–117, 2006.

[5] S. Baker and S.K. Nayar. A theory of catadioptric image formation. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 35–42, 1998.

[6] H. Bakstein and T. Pajdla. Panoramic mosaicing with a 180/spl deg/ field of view lens. In *Proceedings of the IEEE Workshop on Omnidirectional Vision 2002. Held in conjunction with ECCV'02*, pages 60–67, 2002.

[7] J.P. Barreto and H. Araujo. Geometric properties of central catadioptric line images and their application in calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1327–1333, 2005.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[9] James R Bergen and Edward H Adelson. The plenoptic function and the elements of early vision. *Computational models of visual processing*, 1:8, 1991.

[10] Dmitry Bershadsky, Steve Haviland, and Eric N Johnson. Electric multirotor uav propulsion system sizing for performance prediction and design optimization. In *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 0581, 2016.

[11] Roberto Brunelli. *Template matching techniques in computer vision: theory and practice*.

[12] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[13] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[14] Luca Carlone, Zsolt Kira, Chris Beall, Vadim Indelman, and Frank Dellaert. Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4290–4297, 2014.

[15] C. Cauchois, E. Brassart, L. Delahoche, and T. Delhommelle. Reconstruction with the calibrated syclop sensor. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 2, pages 1493–1498 vol.2, 2000.

[16] Raphael M. Chang. Significance of omnidirectional fisheye cameras for feature-based visual slam. https://dspace.mit.edu/handle/1721.1/129885, 2020.

[17] Yun Chang, Yulun Tian, Jonathan P. How, and Luca Carlone. Kimera-multi: a system for distributed multi-robot metric-semantic simultaneous localization and mapping, 2020.

[18] Intel Corporation. Intel® aero compute board hardware features and usage. https://www.intel.com/content/dam/support/us/en/documents/drones/development-drones/intel-aero-compute-board-guide.pdf, 2017.

[19] Intel Corporation. Intel® aero platform for uavs github. https://github.com/intel-aero, 2018.

[20] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[21] Mouser Electronics. Seeed Studio A203 2 Version Carrier Board. https://www.mouser.com/new/seeed-studio/seeed-studio-a203-2-version-board/, 2022.

[22] Luca Carlone et al. Sensing Perception Autonomy and Robot Kinetics (SPARK) Laboratory. https://mit.edu/sparklab/, 2022.

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

[24] Joshua Fishman, Samuel Ubellacker, Nathan Hughes, and Luca Carlone. Dynamic grasping with a" soft" drone: From theory to practice. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4214–4221. IEEE, 2021.

[25] Georgios Georgakis, Srikrishna Karanam, Ziyan Wu, Jan Ernst, and Jana Košecká. End-to-end learning of keypoint detector and descriptor for pose invariant 3d matching. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1965–1973, 2018.

[26] J. Gluckman and S.K. Nayar. Ego-motion and omnidirectional cameras. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 999–1005, 1998.

[27] Nathan Hughes, Yun Chang, and Luca Carlone. Hydra: A real-time spatial perception engine for 3d scene graph construction and optimization, 2022.

[28] Laura Jarin-Lipschitz, Rebecca Li, Ty Nguyen, Vijay Kumar, and Nikolai Matni. Robust, perception based control with quadrotors, 2020.

[29] Jae-Hean Kim and Myung Jin Chung. Slam with omni-directional stereo vision sensor. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, volume 1, pages 442–447 vol.1, 2003.

[30] L. Carlone, K. Khossoussi, V. Tzoumas, G. Habibi, M. Rhyl, R. Talak, J. Shi, and P. Antonette. Visual navigation for autonomous vehicles: An open-source hands-on robotics course at MIT. *2022 In IEEE Intl. Conf. on Integrated STEM Education Conference (ISEC)*, Oct 2022.

[31] Johannes Lächele, Antonio Franchi, Heinrich H Bülthoff, and Paolo Robuffo Giordano. Swarmsimx: Real-time simulation environment for multi-robot systems. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 375–387. Springer, 2012.

[32] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Geometric tracking control of a quadrotor uav on se(3). In *49th IEEE Conference on Decision and Control (CDC)*, pages 5420–5425, 2010.

[33] Young Jin Lee, Do-Yoon Kim, and Myung Jin Chung. Feature matching in omnidirectional images with a large sensor motion for map generation of a mobile robot. *Pattern Recogn. Lett.*, 25(4):413–427, mar 2004.

[34] Shuoyuan Liu, Peng Guo, Lihui Feng, and Aiying Yang. Accurate and robust monocular slam with omnidirectional cameras. *Sensors*, 19(20), 2019.

[35] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[36] András L Majdik, Charles Till, and Davide Scaramuzza. The zurich urban micro aerial vehicle dataset. *The International Journal of Robotics Research*, 36(3):269–273, 2017.

[37] Helen Oleynikova Rik Bähnemann Marija Popović Markus Achtelik, Michael Burri. mav trajectory generation. https://mav-trajectory-generation.readthedocs.io/en/latest/, 2018.

[38] Rico Merkert and James Bushell. Managing the drone revolution: A systematic literature review into the current use of airborne drones and future strategic directions for their effective control. 2020.

[39] Raul Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *CoRR*, abs/1610.06475, 2016.

[40] Soohun Oh, Minwoo Kim, Hyeongseok Kim, Daejin Lim, Kwanjung Yee, and Dongmin Kim. The solution development for performance analysis and optimal design of multicopter-type small drones. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 975–982, 2020.

[41] Nur Raihan Ramli, Sazalinsyah Razali, and Mashanum Osman. An overview of simulation software for non-experts to perform multi-robot experiments. In *2015 International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR)*, pages 77–82, 2015.

[42] Zachary Ravichandran, J Daniel Griffith, Benjamin Smith, and Costas Frost. Bridging scene understanding and task execution with flexible simulation environments. *arXiv preprint arXiv:2011.10452*, 2020.

[43] Benjamin Resch, Jochen Lang, and Hendrik P.A. Lensch. Local image feature matching improvements for omnidirectional camera systems. In *2014 22nd International Conference on Pattern Recognition*, pages 918–923, 2014.

[44] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[45] Alejandro Rituerto, Luis Puig, and J.J. Guerrero. Visual slam with an omnidirectional camera. In *2010 20th International Conference on Pattern Recognition*, pages 348–351, 2010.

[46] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone. 3D dynamic scene graphs: Actionable spatial perception with places, objects, and humans. In *Robotics: Science and Systems (RSS)*, 2020.

[47] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020.

[48] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *The International Journal of Robotics Research*, 40(12-14):1510–1546, 2021.

[49] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: From slam to spatial perception with 3d dynamic scene graphs. *The International Journal of Robotics Research*, 40(12-14):1510–1546, 2021.

[50] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[51] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks, 2019.

[52] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A flexible technique for accurate omnidirectional camera calibration and structure from motion. In *Fourth IEEE International Conference on Computer Vision Systems (ICVS'06)*, pages 45–45. IEEE, 2006.

[53] Davide Scaramuzza, Agostino Martinelli, and Roland Siegwart. A toolbox for easily calibrating omnidirectional cameras. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5695–5701. IEEE, 2006.

[54] Miriam Schönbein and Andreas Geiger. Omnidirectional 3d reconstruction in augmented manhattan worlds. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 716–723. IEEE, 2014.

[55] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jorg Stuckler, and Daniel Cremers. The tum vi benchmark for evaluating visual-inertial odometry. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2018.

[56] Xuelun Shen, Cheng Wang, Xin Li, Zenglei Yu, Jonathan Li, Chenglu Wen, Ming Cheng, and Zijian He. Rf-net: An end-to-end image matching network based on receptive field, 2019.

[57] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[58] Skydio. Inside The Mind Of The Skydio 2: A Revolution In Drone Technology. https://www.skydio.com/blog/inside-the-mind-of-the-skydio-2/, 2019. [Online; December 17 2019].

[59] Dinesh Thakur, Giuseppe Loianno, Laura Jarin-Lipschitz, Alex Zhou, and Vijay Kumar. Autonomous inspection of a containment vessel using a micro aerial

vehicle. In *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–7, 2019.

[60] Yulun Tian, Katherine Liu, Kyel Ok, Loc Tran, Danette Allen, Nicholas Roy, and Jonathan P. How. Search and rescue under the forest canopy using multiple uavs, 2019.

[61] Ivana Tošic and Pascal Frossard. Spherical imaging in omni-directional camera networks. *Multi-Camera Networks, Principles and Applications*, 2009.

[62] Richard Vaughan. Massively multi-robot simulation in stage. *Swarm intelligence*, 2(2):189–208, 2008.

[63] Yanchang Wang, Xiaojin Gong, Ying Lin, and Jilin Liu. Stereo calibration and rectification for omnidirectional multi-camera systems. *International Journal of Advanced Robotic Systems*, 9(4):143, 2012.

[64] Changhee Won, Jongbin Ryu, and Jongwoo Lim. Omnimvs: End-to-end learning for omnidirectional stereo matching. *CoRR*, abs/1908.06257, 2019.

[65] Math Works. Fisheye calibration basics. https://www.mathworks.com/help/vision/ug/fisheye-calibration-basics.html, 2007.

[66] Xianghua Ying and Zhanyi Hu. Catadioptric camera calibration using geometric invariants. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1260–1271, 2004.