RGB-D SLAM : AN IMPLEMENTATION FRAMEWORK BASED ON THE JOINT EVALUATION OF SPATIAL VELOCITIES

by

Hugo Herman Godelieve Coppejans

Submitted in partial fulfillment of the requirements for the degree
Master of Engineering (Computer Engineering)

in the

Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

October 2017

SUMMARY

RGB-D SLAM : AN IMPLEMENTATION FRAMEWORK BASED ON THE JOINT EVALUATION OF SPATIAL VELOCITIES

by

Hugo Herman Godelieve Coppejans

Supervisor(s):	Dr. H.C. Myburgh
Department:	Electrical, Electronic and Computer Engineering
University:	University of Pretoria
Degree:	Master of Engineering (Computer Engineering)
Keywords:	SLAM, MAV, RGB-D SLAM, Kinect, quadcopter, spatial velocity, visual odometry, frame rate, translational error, rotational error

In pursuit of creating a fully automated navigation system that is capable of operating in dynamic environments, a large amount of research is being devoted to systems that use visual odometry assisted methods to estimate the position of a platform with regards to the environment surrounding it. This includes systems that do and do not know the environment *a priori*, as both rely on the same methods for localisation. For the combined problem of localisation and mapping, Simultaneous Localisation and Mapping (SLAM) is the *de facto* choice, and in recent years with the advent of color and depth (RGB-D) sensors, RGB-D SLAM has become a hot topic for research.

Most research being performed is on improving the overall system accuracy or more specifically the performance with regards to the overall trajectory error. While this approach quantifies the performance of the system as a whole, the individual frame-to-frame performance is often not mentioned or explored properly. While this will directly tie in to the overall performance, the level of scene cohesion experienced between two successive observations can vary greatly over a single dataset of observations.

The focus of this dissertation will be the relevant levels of translational and rotational velocities experienced by the sensor between two successive observations and the effect on the final accuracy of the SLAM implementation. The frame rate will specifically be used to alter and evaluate the different spatial velocities experienced over multiple datasets of RGB-D data.

Two systems were developed to illustrate and evaluate the potential of various approaches to RGB-D SLAM. The first system is a real-world implementation where SLAM is used to localise and map the environment surrounding a quadcopter platform. A Microsoft Kinect is directly mounted to the quadcopter and is used to provide a RGB-D datastream to a remote processing terminal. This terminal runs a SLAM implementation that can alternate between different visual odometry methods. The remote terminal acts as the position controller for the quadcopter, replacing the need for a direct human operator. A semi-automated system is implemented, that allows a human operator to designate waypoints within the environment that the quadcopter moves to.

The second system uses a series of publicly available RGB-D datasets with their accompanying ground-truth readings to simulate a real RGB-D datasteam. This is used to evaluate the performance of the various RGB-D SLAM approaches to visual odometry. For each of the datasets, the accompanying translational and angular velocity on a frame-to-frame basis can be calculated. This can, in turn, be used to evaluate the frame-to-frame accuracy of the SLAM implementation, where the spatial velocity can be manually altered by occluding frames within the sequence. Thus, an accurate relationship can be calculated between the frame rate, the spatial velocity and the performance of the SLAM implementation.

Three image processing techniques were used to implement the visual odometry for RGB-D SLAM. SIFT, SURF and ORB were compared across eight of the TUM database datasets. SIFT had the best performance, with a 30 % increase over SURF and doubling the performance of ORB. By implementing SIFT using CUDA, the feature detection and description process only takes 18 ms, negating the disadvantage that SIFT has compared to SURF and ORB. The RGB-D SLAM implementation was compared to four prominent research papers, and showed comparable results. The effect of rotation and translation was evaluated, based on the effect of each rotation and translation axis. It was found that the z-axis (scale) and the roll-axis (scene orientation) have a lower effect on the average RPE error in a frame-to-frame basis. It was found that rotation has a much greater impact on the performance, when evaluating rotation and translation separately. On average, a rotation of 1° resulted in a 4 mm

translation error and a 20% rotation error , where a translation of 10 mm resulted in a rotation error of 0.2° and a translation error of 45%. The combined effect of rotation and translation had a multiplicative effect on the error metric.

The quadcopter platform designed to work with the SLAM implementation did not function ideally, but it was sufficient for the purpose. The quadcopter is able to self stabilise within the environment, given a spacious area. For smaller, enclosed areas the backdraft generated by the quadcopter motors lead to some instability in the system. A frame-to-frame error of 40.34 mm and 1.93° was estimated for the quadcopter system.

LIST OF ABBREVIATIONS

ATE	Absolute trajectory error
BRIEF	Binary robust independent elementary features
CPU	Central processing unit
CUDA	Compute unified device architecture
DoF	Degrees of freedom
DoG	Difference of gaussian
EMM	Environmental measurement model
FAST	Features from accelerated segment test
FOV	Field of view
fps	Frames per second
GLC	Global loop closure
GPU	Graphics processing unit
g^2o	General graph optimization
ICP	Iterative closest point
IPC	Inter-process communication
IMU	Inertial measurement unit
LLC	Local loop closure
MAV	Micro air vehicle
NNMMT	Nearest neighbour minimum matching threshold
OpenCV	Open source computer vision library
ORB	Oriented FAST and rotated BRIEF
PCL	Point cloud library
PID	Proportional integral derivative
PWM	Pulse width modulation
RANSAC	Random sample consensus
REST	Representational state transfer
RGB	Red green blue
RGB-D	Red green blue depth
RPE	Relative pose error
SIFT	Scale-invariant feature transform
SLAM	Simultaneous localization and mapping

SURF	Speeded up robust features
S-NNMMT	Second nearest neighbour minimum matching threshold
UAV	Unmanned aerial vehicle
VO	Visual odometry

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PROBLEM STATEMENT

1.1.1 Context of the problem

In recent years the use of unmanned aerial vehicles (UAVs) has become more prominent in both the private and public sectors. Historically the public sector, particularly the military, was the only entity that focused on developing and producing UAVs. While UAVs have been around before the 21st century, they were typically fixed-wing drones used for military or geographical purposes. It was not until the last decade had it become possible to create what is called a micro air vehicle, or MAV.

While there are no formal specifications for what type of vehicle constitutes an MAV, they are usually represented by a small device that is capable of unassisted flight and has some size restriction. Since the mid 2000s the price associated with building an MAV declined significantly, allowing much more exposure to the field. This created a significant increase in the number of researchers designing and creating MAVs for a multitude of purposes.

A very popular application is waypoint based automated flight. This allows the MAV to fly autonomously, based on the Global Positioning System (GPS) readings and a predetermined set of locations. While this allows for an inexpensive and easy solution to automation, it also opens up the MAV to the potential pitfalls of GPS. The MAV will be unable to fly in GPS denied areas, or if the GPS signal is particularly weak, it can lead to a potentially dangerous situation where the location data is very inaccurate.

Because of the small dimensions of MAVs, they are capable of flying inside enclosed environments such as buildings or tunnels. This opens up the problem of autonomous flight in GPS denied areas, where a small error in the positioning data can lead to a collision. For an MAV to be capable of true autonomous flight, it should have the ability to localise and navigate inside any enclosed environment. Localisation refers to the ability of estimate the distance relation between the MAV and the surrounding objects. Navigation refers to the ability of remembering the previously seen locations and mapping the current position onto it, allowing the MAV to self-navigate the environment. This process is commonly referred to as Simultaneous Localisation and Mapping, or SLAM. The performance of any given SLAM implementation can be measured by the error between its perceived location and its actual location with regards to the environment. The overall goal of SLAM is to reduce this error metric to near zero, thus reflecting the position of the MAV near-perfectly.

1.1.2 Research gap

A significant body of research is available that focus on comparing and evaluating different types of SLAM implementations. Comparatively, very little research has been performed on the effect that the quality and quantity of data has on performance, especially with consideration to the spatial relationship between observations. This dissertation will specifically focus on the effect that data reduction through frame rate reduction has on the performance of various SLAM implementations with regards to a real-world approach. By reducing the number of frames available per second, the effective change in the field of view (FOV) between two successive observations is increased. This can be used to compare the accuracy of a SLAM implementation based on the relative translational and angular velocity the sensor undergoes.

The real-world application refers to a physical implementation for a SLAM based system that can be used to demonstrate the importance of speed versus accuracy. This system will also provide a suitable demonstration of the capabilities that a SLAM implementation can deliver. Multiple approaches to visual odometry for SLAM will be compared in a real-time situation over varying degrees of frame rate reduction.

Typically, when developers set out to create a SLAM based system, they have the tendency to maximise the performance of the system with regards to the budget. This dissertation intends to give a clear

guideline of the data frame rate required to operate within a certain error margin, given a predetermined translational and angular velocity. This, in turn, can be used to design the system hardware in order to comply with a predefined performance level.

1.2 RESEARCH OBJECTIVE AND QUESTIONS

Students and researchers that start working on a new topic tend to over-design the specifications for their projects, which could lead to budgetary problems or a simple waste of resources. This holds true with regards to SLAM and real-world applications, partly because of the developers' mindset and partly because of a lack of literature when it comes to the subject of optimisation.

For a large portion of SLAM implementation, the most basic concept that is typically neglected in the literature, is the level of scene coherence required for smooth operation. This scene coherence is directly related to the datastream density, and refers to the quality and quantity of the raw data being supplied to the system. This, in turn, directly relates to the level of spatial variability experienced between two successive frames of data. The quality and quantity of the data being used can be evaluated on different levels:

- Frame rate - The number of times per second that an observation is made and the data is delivered also referred to as frames per second (fps).
- Data density - The resolution of the data. This can be defined in pixels per inch (PPI).
- Data accuracy - How accurately the environment is represented in the digital data. This factor is hard to evaluate using standard means.
- Field of View (FOV) - Which viewing angle does the sensor in question has. In comparison humans have a 114° three dimensional FOV.

A few other minor qualities exist, but these four are the main indicators of data quality.

Comparatively more literature involving the data accuracy and the data density exists than the frame rate and FOV. Since the FOV for a specified sensor is predetermined by the hardware used, it cannot be changed easily. It also is very difficult to directly compare the performance between two separate

cameras the various datasets will differ. This dissertation will therefore focus on the frame rate and the resulting angular and translational velocity accompanying each frame rate.

The following research questions are posed and evaluated during the course of this dissertation. All the questions posed are directly related to SLAM.

- What effect does a reduction in the frame rate have on the accuracy of a SLAM-based navigation system?
- How does a reduction in frame rate relate to the angular and translation velocities experienced?
- What effective frame-to-frame error rate can be expected for various levels of rotation and translation?
- What effect does an increased latency have on a real-world SLAM-based navigation system and what factors contribute the most to the full loop latency for such a system?

Each of these questions will be analysed and evaluated in both a controlled simulated environment and a real-world environment, where SLAM will be used in a real-world setting to approximate the location of a moving platform.

1.3 APPROACH

For the posed research questions two separate systems need to be created that can work in conjunction with each other. The first system that should be created, is a robust testing environment, or a physical platform that can be used to test a real-world approach. This is to illustrate the effectiveness and potential advantages that a SLAM-based system has when working towards an automated system. Owing to limitations, this system will only be able to operate inside an enclosed and controlled environment. This platform should have the ability to be controlled remotely and have the ability to transverse inside the enclosed environment. Additionally, it should provide a suitable mount and datastream for a camera or sensor that can be used for a SLAM-based approach to automation.

The second system that needs to be created, is a SLAM implementation that can be used to localise and map the environment from a given datastream. This implementation should also serve as a robust simulation platform to evaluate the accuracy of various SLAM approaches over a varying degree of data

reduction. This system should be able to use community driven datasets that can be used to effectively evaluate the performance of the SLAM implementations against peer-reviewed results.

To achieve this, the system needs to be able to accept public datasets that are freely available to other researchers, as well as to have a correspondent metric to evaluate the results achieved. This system will have to simulate a real-world situation, and certain parameters such as frame rate and data density needs to be adjustable. The system should be able to calculate the effective angular and translational speeds for varying levels of frame rate, as well as compile the associated error metrics. This data will then be used to evaluate the effect that varying degrees of frame rate reduction have on the accuracy of a few prominent SLAM implementations. These results can then be directly compared to the results obtained by other researchers, using the same datasets.

1.4 RESEARCH GOALS AND CONTRIBUTION

The goal of this dissertation is to create a base of reference that can be used when starting with a new project. A clear indication will be given about what the expected performance of a given SLAM implementation will be, given a known translational and angular velocity. This will provide individuals that are new to the field a good point of reference when starting with a project, or provide more established researchers a broader sense on the limitations of each approach. This information will improve the chances that a new SLAM implementation will work smoothly without major alterations after having designed the system.

A very limited amount of research that delves into the performance of various SLAM algorithms with regards to the spatial relationship between frames is available and thus this research has a valuable role to play in progressing the overall field. During the implementation of SLAM, the amount of processing power can be a major challenge to overcome. Without the use of GPUs, it can become almost impossible to run a large scale SLAM implementation. This is especially true when working with onboard processing for smaller platforms such as quadcopters or small land roving cars.

Thus, this research will provide a sense about the level of data quality and quantity of data required to achieve a desired accuracy. This effect will also be measured on a publicly available database so that it

can be compared to other corresponding research papers, and will allow an accurate measurement of performance.

1.5 RESEARCH OUTPUTS

A single article titled ”A primer on autonomous aerial vehicle design” has been published that detail the process followed in creating a potentially autonomous quadcopter [1]. The article investigates potential hardware configurations and different sensors that can be used for SLAM. Multiple SLAM implementations were identified and presented as possible solutions for automating a quadcopter platform. The trade-offs between online and offline processing is also evaluated.

Two additional articles are in preparation, with one focusing on the limitations and problems faced when creating a quadcopter that can operate in an enclosed environment. The second article will focus on the relative frame-to-frame error rate determined for various levels of spatial velocity, and the effect that translation and rotation had on that error. It is expected that these articles will be published before the examination date.

1.6 OVERVIEW OF DISSERTATION

The next chapter will give a full literature review of the topics relevant to this dissertation. The quadcopter platform will be discussed briefly, as well as the controllers required to operate the quadcopter. Theoretical SLAM will be presented and discussed, as well as the approximations used to implement SLAM in a real-world system. RGB-D SLAM is chosen as the implementation of choice for this dissertation. All the relevant systems for RGB-D SLAM is discussed in a theoretical sense. Various visual odometry methods are presented and evaluated based on a few key parameters.

Chapter 3 will discuss the creation and implementation of the quadcopter and the software applications for this dissertation. The interconnectivity between the applications are discussed, and the methods used for the SLAM implementation is presented. Two error metrics are defined that will be used to evaluate the performance of the SLAM implementation.

Chapter 4 presents and discusses the results obtained. The first section is dedicated to the SLAM implementation and the effective accuracy achieved by each of the systems in the front-end and back-end. The overall accuracy is calculated and compared to four other implementations. The next section presents an investigation on the effects that the spatial velocity has on the frame-to-frame error rate. These effects are presented and discussed with regards to the relative rotational and translational velocities. Finally the performance of the quadcopter platform is estimated based on the results obtained.

The final chapter provides an overview of the dissertation and the results obtained. Potential future alterations and improvements are discussed.

CHAPTER 2 LITERATURE STUDY

2.1 CHAPTER OVERVIEW

To fully comprehend what will be required to achieve the research objectives, the full dissertation scope must be discussed. This dissertation will describe two systems working in conjunction with each other, but each will still function independently. These two systems will be a physical platform that will be used to demonstrate the effectiveness and utility of SLAM, and the second will be the actual implementation of a SLAM system.

The goal of the physical platform is to create a stable testing platform that can be used for multiple purposes. The platform should provide a stable environment that can be manipulated through a wireless connection from a remote station, allowing it to be controlled by either a human operator or some automated process. The platform should include a 3D sensor that is capable of providing a datasteam that the SLAM implementation can utilise.

For the SLAM implementation, it should be able to connect to a datastream provided either by the physical platform or a data manipulation application. The SLAM implementation should be able to operate in realtime with the data provided by the physical platform. Ideally this should allow the platform to be controlled autonomously in future, stabilising itself within the environment by correcting for any drift, and allowing it to move to designated waypoints.

An alternative application should also be created that can alter and manipulate the datasteam being sent to the SLAM implementation. This will allow for controlled manipulation of the frame rate, effectively altering the size of the translation and rotation experienced between observations. This application should also be able to use a datastream that has been recorded, allowing the use of public datasets that

have verified positioning data. These databases will provide the ability to track the accuracy of each SLAM implementation, allowing for an accurate assessment of the error metric with regards to varying degrees of translational and rotational speeds.

By combining these two systems into one, the platform will be able to localise itself within the environment while building a dense 3D map of the surroundings.

This chapter will describe the relevant literature associated with both the physical platform and SLAM in general. Section 2.2 will focus on the physical platform required for this dissertation. In Section 2.3 focus will be placed on SLAM in general as well as the possible derivations of SLAM. Section 2.4 will focus on RGB-D SLAM and a few base approaches to RGB-D SLAM that will be used for this dissertation. Section 2.5 explains some of the feature detector and descriptor pairs that will be important for this dissertation.

2.2 PLATFORM REQUIREMENTS AND LITERATURE

2.2.1 Platform choice

For this dissertation, the physical platform that will be used is a quadrotor helicopter, more commonly known as a quadcopter. Quadcopters fall under the classification of MAV, but they can vary in size, some being able to fit into the palm of your hand and others large enough to carry humans. A quadcopter was chosen because of its availability, as the field has recently seen a spike in availability and use. It will provide a novel platform for such an implementation and will also align with the the current high level of interest in the field of automation.

From a practical viewpoint a quadcopter platform using SLAM is ideal for mapping enclosed areas, such as houses and offices, and can be used to explore areas that are potentially hazardous for humans to enter, such as cave-ins or hazardous material storage. Alternatively, these systems can be combined with facial recognition software to track and match people moving beneath the quadcopter.

One major advantage that a quadcopter platform will provide over its more common ground-based brethren is elevation. Because most land-based robots are fairly low to the ground, the field of view is

rather constricted due to objects such as tables and chairs blocking the field of view. A quadcopter, however, can fly at the average human height, providing an image that is much easier to relate to from a human perspective.

2.2.2 Creating a quadcopter capable of localisation and mapping

Because quadcopters are considered to be highly unstable and non-linear systems, the choice of sensors, controllers and software is intricate. It is fairly easy to create an ideal model of a quadcopter system, but to design such a system around a set of constraints and specifications is fairly complex. This section will explain and present the various components related to creating a quadcopter capable of accurate indoor flight.

The largest problems that MAVs face generally, are the stabilization and control in six degrees of freedom (DoF), more commonly referred to as attitude and position control. The attitude control can be solved using a simple proportional-derivative (PD) controller. Other techniques can also be applied, such as “sliding mode” and “backstepping” [2, 3].

The second problem of position control is much more complex and will be the main focus of this dissertation. Without position control on a quadcopter, it will be prone to drift because of the constant corrections the attitude controller has to make. The quadcopter will also be unable to localise itself within a known environment. This problem can easily be solved using GPS [4] and has been the preferred method for UAV designs in the past. For MAVs, particularly for quadcopters, GPS is not a reliable service as it is prone to lose accuracy and connection in urban canyons and indoor areas. Because of this, an alternative solution must be found for quadcopters intended for indoor use.

Some of the earliest methods involve the use of externally-mounted cameras (*i.e.* off-board) to track the movement of the platform. The platform is mounted with reflective tags, picked up by the cameras and by knowing the exact position of the cameras, the platform’s position can be inferred [5, 6]. This method has numerous disadvantages: The quadcopter is limited to areas that are covered by the cameras, and the cameras need manual installation and calibration. Thus, this method is limited to areas that are accessible by humans. The only alternative to this method is to use vision sensors that allow the platform to capture the environment surrounding it. This allows the platform to freely explore

without a constraint on the area in which it can operate. The difficulty arises when the quadcopter needs to be able to build a map of its surroundings and then localise itself within that map. A method called simultaneous localisation and mapping (SLAM), coined by Durrant, Rye & Nebot in [7], is the preferred method for solving this problem.

Thus, the attitude and position controllers are responsible for keeping the MAV, in this case the quadcopter, in the air and moving towards its goal. Figure 2.1 shows these two controllers' interaction with each other and the relevant components.

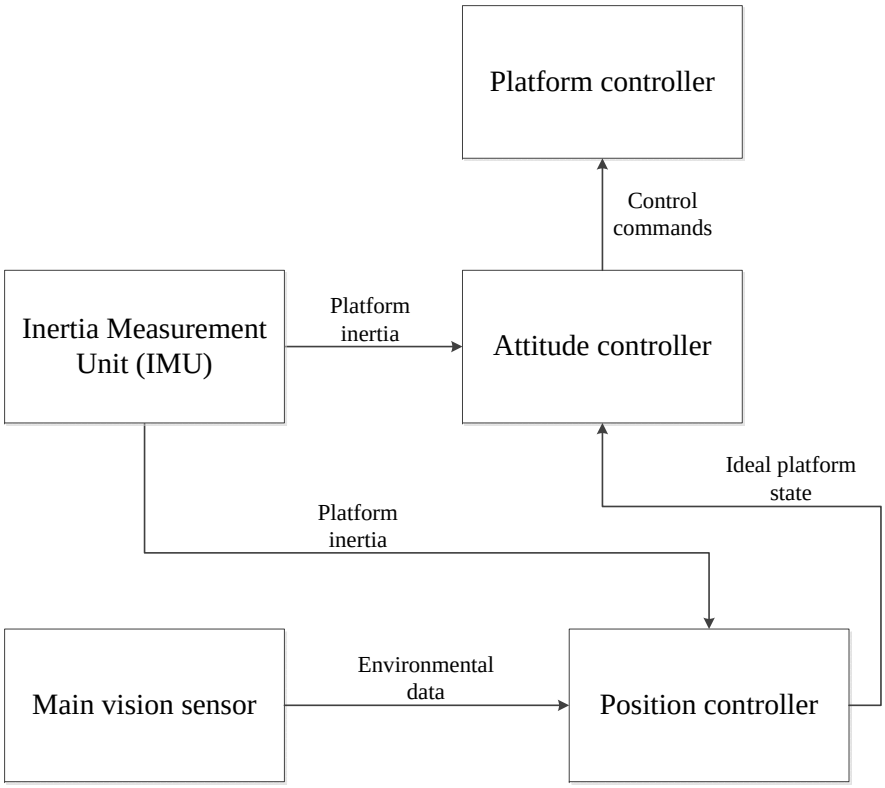


Figure 2.1. This shows the basic block diagram for an automated MAV. The vision sensor provides the position controller with data to stabalise the MAV within the environment, while the attitude controller uses the IMU data to stabalise the quadcopter without any regards to the environement. The position controller in turn gives the attitude controller the ability to move to desired locations.

2.2.2.1 Platform Controller

The platform controller does not require any type of intelligence or complex algorithms to operate. This controller receives a set of instructions from the attitude controller before adjusting the power

distribution between the motors of the platform to perform the received command. Figure 2.2 shows the possible states that a quadcopter can adopt. Each of these states has a certain outcome, indicated by the blue arrow at the top left of each subfigure. The commands can be hard-coded into the platform controller, and even two or more states can be combined at one time interval. The attitude controller also provides the desired power output for the quadcopter. The power to each motor is then scaled according to this power output, which allows the quadcopter to change altitude.

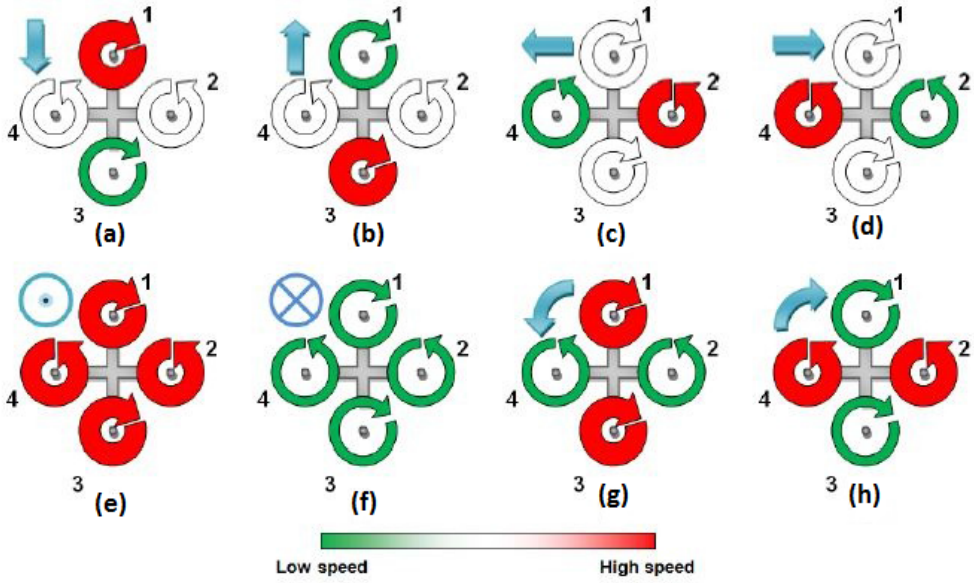


Figure 2.2. Quadcopter states when viewed from above. (a) Forward motion (b) Backwards motion (c) Movement left (d) Movement right (e) Increase altitude (f) Decrease altitude (g) Counter-clockwise rotation (h) Clockwise rotation

2.2.2.2 The attitude controller

Attitude control stabilises the quadcopter and keeps it in the desired state or position. The attitude controller compensates for environmental factors that cannot be estimated, such as wind turbulance [8]. It also compensates for small errors in power distribution, caused by non-ideal hardware, which would otherwise lead to the instability of the quadcopter. The attitude controller receives inertia data from the inertia measurement unit (IMU) on the platform, allowing the controller to infer the platform state in free space from the data. The platform's current state is compared to the ideal platform state that is defined by the position controller. The attitude controller calculates the appropriate changes that need to be made to the platform to achieve the ideal state. While this method is sound in theory, the platform

will never be able to achieve the ideal state, but will oscillate around the ideal state instead [8]. The performance of the attitude controller can be quantified by the size of the oscillation.

2.2.2.3 The position controller

In a standard human-controlled quadcopter, the position controller is replaced by commands given through a controller device that is operated by a human, such as a remote control. For the purpose of this dissertation, the human controller is replaced by a controller system that attempts to reach a goal state. This goal state is either defined beforehand or is changed based on the information gathered during operation. The goal state of the quadcopter will vary with each new application, so it cannot be explicitly defined. As an example, the goal state of an MAV that is tasked to create a map of the environment will be to move the platform to unknown areas in the map. As each area is scanned, the goal will dynamically change. The position controller calculates the correct roll, pitch and yaw so that the platform will start to move towards the goal state at a certain speed. The position controller uses data from both the IMU and the vision sensor to build a map of the environment and to localise the quadcopter within that map. The data from the IMU is used as a dead-reckoning technique [9], where SLAM is used with the data from the vision sensor. With regard to a UAV or MAV that operates outdoors, the position controller will most likely consist of a GPS. The position controller does not directly send control commands to the MAV, but rather updates the ideal state of the attitude controller.

2.2.3 Quadcopter structure and design

Even though building a custom quadcopter is no trivial task, there is very little academic purpose in evaluating and explaining all the relevant design decisions that were presented and will not ultimately contribute to the research objectives of this dissertation. An article published by the author has already been created to cover the topic of creating an automated quadcopter [1]. The article covers the topic of building a quadcopter that can be used for autonomous flight. All relevant design decisions are evaluated and discussed in full.

2.3 SIMULTANEOUS LOCALISATION AND MAPPING

SLAM is the problem of placing a mobile robot in an unknown environment (no prior knowledge), wherein the robot must then be able to incrementally build a consistent map of this environment while simultaneously determining its own location with regard to the map that is being built [10]. This problem is one of the cornerstones of creating autonomous vehicles, whether they be land-based, aerial or underwater [11]. The theoretical SLAM problem has already been defined, but real-world implementations are almost always much more difficult to solve. The concepts of solving SLAM have been developed and simulated by researchers, but the practical realisation of these concepts is still an ongoing field of research.

Recently, great strides have been made in creating MAVs that are able to operate in unknown, cluttered and GPS-denied environments using SLAM [12–15]. Altough the problem has been solved to a point where it can be used in a practical application, research is continuing in order to create alternative solutions or to make improvements. More advanced problems, such as autonomous exploration, swarm navigation and large trajectory planning, are also being researched [14, 16, 17].

2.3.1 Why use SLAM

In this work SLAM will play a key role in the position controller of the quadcopter. The position controller can be summarised as a controller that takes over the role of a human operator. Thus, it has the very simple task of informing the MAV where to go and of keeping it stable with regards to its environment. For a human, this task is very rudimentary, but to implement it using artificial means is very difficult. Thus some type of *awareness* must be given to the system that allows it to know its own position with regards to the surroundings. This might sound easy, but it is a far more complex problem than the layman might expect.

The best way to capture the surroundings into a digital container is to use a type of sensor that replicates the environment into digital data, basically a camera. Ideally a 3D camera would provide the best usable data for 3D mapping. The camera captures the environment within its field of view (FOV) at a specific frame rate and data density. An implementation that is able to track the spatial relationship is required, i.e. translation (x,y,z movement in space) and rotation (orientation in space), between two

successive frames. It also needs the ability to remember locations it has previously visited, commonly referred to as loop-closure.

To date, SLAM is the most advanced and successful method for solving this problem. SLAM allows the user to quantify the environment currently within the quadcopter's FOV into landmarks, and then match those landmarks to previously recorded landmarks. Combining this with 3D data, it allows the system to be aware of its own position and orientation with regards to its surroundings, while tracking its own movement within that environment.

Thus, the position and orientation data that the SLAM implementation will return, will be used for:

1. stabilising the quadcopter with regards to its surrounding environment,
2. building a systematic 3D map of the environment,
3. localising itself within that 3D map, allowing for possible automated navigation.

2.3.2 SLAM within the position controller

Figure 2.3 shows a breakdown of the position controller into its basic functional units.

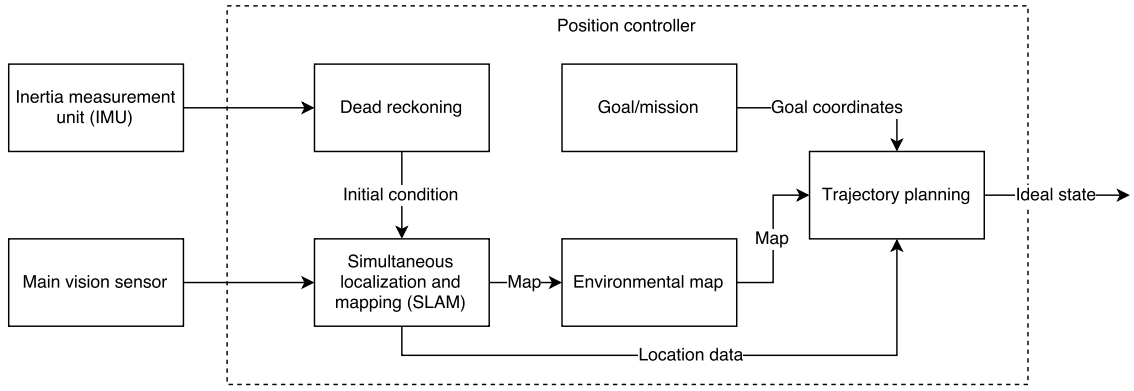


Figure 2.3. Functional blocks of the position controller. This system is responsible for potentially automating an MAV.

The position controller uses data from both the IMU and the vision sensor. The data from the IMU is only used for a single purpose, but it is a very important one. The data obtained from the IMU is used

to estimate the movement that the platform has undergone in 3D space between the current instance and the previous one. This method is called dead-reckoning and provides an initial estimate of the quadcopter's relative movement in 3D space. Dead-reckoning is not a very accurate or trustworthy method and on its own does not allow for a very good position estimate, but it allows the SLAM system to have an initial estimate as to the spatial movement undergone between observations.

The SLAM system receives periodical data from the vision sensor mounted on the quadcopter. The type of data will depend on the sensor being used. A large variety of different sensors and accompaning implementations can be used. The SLAM system must be tailored to work with the type of sensor that has been selected. For instance, when a Microsoft Kinect is used, the data would be comprised of a depth image and a red, green and blue (RGB) colour image. If a standard stereo camera is used, the type of SLAM system that would be required, differs considerably from one using a Kinect [18]. The different possibilities and trade-offs are discussed in the related article published by the author [1]. For this dissertation the Microsoft Kinect will be used as the main sensor for the platform, due to its availability at the time.

The SLAM system returns two pieces of information. The first piece of information is a map of the surrounding area, that is constructed and updated continuously with each iteration of the algorithm. This map can either be a 2D or 3D map, showing all the objects and structures that were observed. This map can be known *a priori*, but the idea of an autonomous MAV is to function in an unknown area; thus, the map needs to be created on the fly. This map will be on scale with the environment around the platform, and the quality will depend on the type of sensor used. Because the SLAM implementation is so time-consuming, some implementations purposefully reduce the resolution of the map to increase the frames per second of the SLAM system [19, 20]. The second piece of information that the SLAM implementation returns, is the location of the platform with respect to the map it has created up until that point. This allows the system to "know" where it is relative to the objects and structures in its environment. This is a vital part of the MAV's automation process. This allows the MAV to stabilise itself with regards to objects surrounding it, and also provides the trajectory planning subunit with a current position.

The position controller contains a goal or mission that it must complete. This goal is very difficult to define, as it will be different for each unique application. One common thread that will be shared between all applications is that the MAV must reach a certain location on the map created by the

SLAM system. For this dissertation the goal state will be represented by a position that the quadcopter needs to be at, at that specific instance. The quadcopter will attempt to move to this position until a set of predefined conditions have been met. This topic will be discussed in more detail later in this chapter.

2.3.3 Probabilistic SLAM

SLAM originated by applying estimation theory methods to mapping and localization problems [21]. For a complete discussion of SLAM, the probabilistic model of SLAM should be understood. This suggests that a set of different vectors is defined, each describing a specific characteristic of the SLAM problem. Figure 2.4 shows the SLAM model used to describe these vectors.

One important fact that must be stated is that the locations of landmarks are never physically measured in the real world and provided to the SLAM problem.

Figure 2.4 shows a model of the SLAM problem for a robot moving through an environment. For each time instant k (typically, this will be for each observation made), the following vectors are defined.

- $\mathbf{x_k}$: This is the state vector that describes each state at each time instant. The location, orientation and velocity of the robot can be used to give more information for each state.
- $\mathbf{u_k}$: This is the control vector that contains the commands issued to move the robot from the position at time instant $k-1$ to the current position at time instant k.
- $\mathbf{m_i}$: This is a vector that contains the position of all of the landmarks found in the environment up until time instant k.
- $\mathbf{z_{ik}}$: This contains the data from an observation taken of landmark i at time instant k.

Additionally, the following datasets are also defined:

- $\mathbf{X_{0:k}} = \{\mathbf{x_0}, \mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_k}\}$: the history of vehicle states,
- $\mathbf{U_{0:k}} = \{\mathbf{u_0}, \mathbf{u_1}, \mathbf{u_2}, ..., \mathbf{u_k}\}$: the history of command inputs,
- $\mathbf{m} = \{\mathbf{m_0}, \mathbf{m_1}, \mathbf{m_2}, ..., \mathbf{m_k}\}$: the position of all of the landmarks in the known environment,

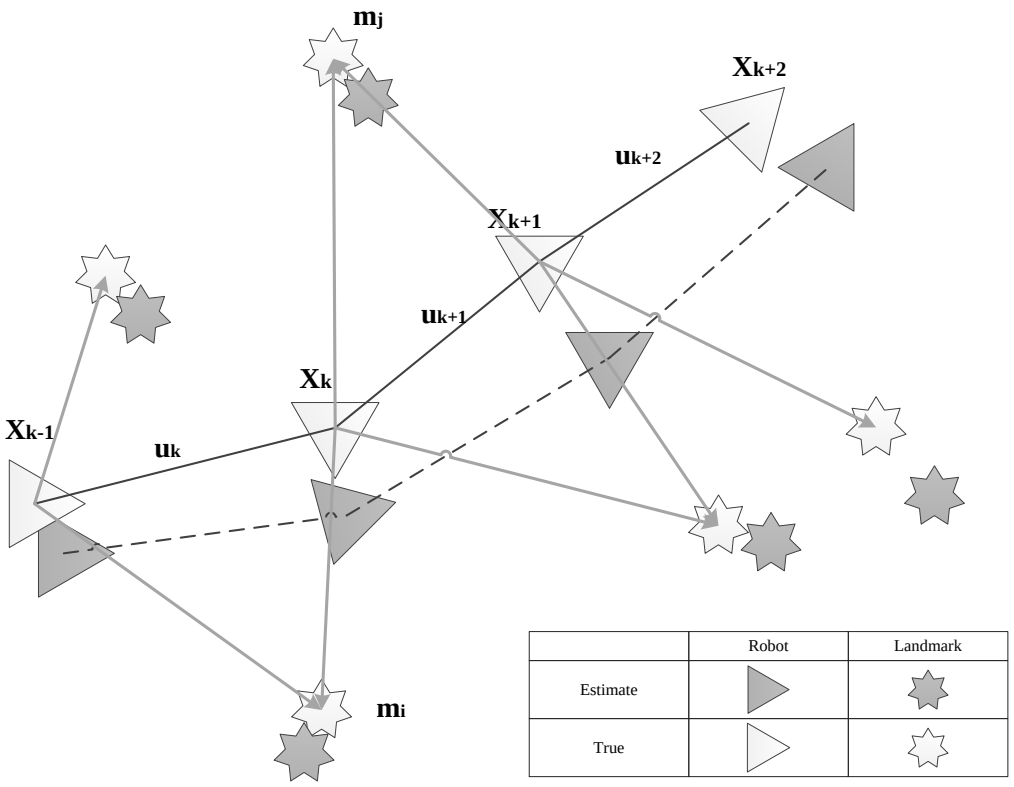


Figure 2.4. The SLAM problem quantified into different vectors. Each individual landmark is represented by $\mathbf{m}$, while $\mathbf{X_k}$ and $\mathbf{u_k}$ shows the vehicle state and input commands respectably.

- $\mathbf{Z_{0:k}} = \{\mathbf{z_0}, \mathbf{z_1}, \mathbf{z_2}, ..., \mathbf{z_k}\}$: the set of all landmark observations,

where Position 0 is to be the initial starting location/landmark.

Using this model, SLAM can be defined as a probabilistic problem that can be solved using Bayes' equation. For all of the time instants k, the probability distribution of:

$$P(\mathbf{x_k}, \mathbf{m} | \mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}) \tag{2.1}$$

must be calculated. This will produce the probability distribution that describes the joint posterior density of the landmark locations and vehicle states at the given time instant k. The distribution depends on all of the observations and command inputs from the initial position up until time instant k.

An iterative process is used to compute the probability distribution in (2.1) at each time instant k, depending on all past time instants $k-t$, where $t = 0, 1, 2..., k-1$. The probability distribution can be

calculated using Bayes' equation

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}. \tag{2.2}$$

Thus, from (2.2) the following probability needs to be derived:

$$P(\mathbf{x_{k-1}}, \mathbf{m}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k-1}}). \tag{2.3}$$

This will give an estimate for the distribution at the time $k-1$, after a control command $\mathbf{u_k}$ and observation $\mathbf{z_k}$ have occurred. Two new models, the state transition model and the observation model, are defined to do this calculation.

The observation model describes the probability that an observation $\mathbf{z_k}$ is made (at time instant k), given the vehicle locations and landmark locations. This probability is defined as

$$P(\mathbf{z_k}|\mathbf{x_k}, \mathbf{m}). \tag{2.4}$$

Considering a typical SLAM situation, it can safely be assumed that once the location of the robot and the map is defined, observations are conditionally independent of each other.

The second model that is required is the motion model. The motion model for the robot is defined as

$$P(\mathbf{x_k}|\mathbf{x_{k-1}}, \mathbf{u_k}). \tag{2.5}$$

It is therefore assumed that the state transitions (or movement) of the robot can be modelled as a Markov process, implying that each state $\mathbf{x_k}$ is only dependent on the previous state $\mathbf{x_{k-1}}$ and the control command $\mathbf{u_k}$. It is assumed therefore that there are no remaining kinetic energy (such as momentum) on the robot from any states prior to $\mathbf{x_{k-1}}$.

Given the observation and motion models, the SLAM algorithm (and in general, Bayesian filtering) can be implemented in a two-step recursive fashion. The time update and measurement update are done recursively for each time instant. The time update is given as:

$$P(\mathbf{x_k}, \mathbf{m}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}}, \mathbf{x_0}) = \int [P(\mathbf{x_k}|\mathbf{x_{k-1}}, \mathbf{u_k}) P(\mathbf{x_{k-1}}, \mathbf{U_{0:k}}, \mathbf{x_0})] \mathrm{d}\mathbf{x_{k-1}} \tag{2.6}$$

and the measurement update is given as:

$$P(\mathbf{x_k}, \mathbf{m}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}) = \frac{P(\mathbf{z_k}|\mathbf{x_k}, \mathbf{m}) P(\mathbf{x_k}, \mathbf{m}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}}, \mathbf{x_0})}{P(\mathbf{z_k}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}})}. \tag{2.7}$$

From (2.6) and (2.7) a recursive solution can be given for calculating the joint posterior probability

$$P(\mathbf{x_k}, \mathbf{m} | \mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{x_0}), \tag{2.8}$$

given the robot states $\mathbf{x_k}$, the map (or landmarks) $\mathbf{m}$ for all of the observations from zero to k.

For the next part of the probabilistic SLAM model, the concepts that are presented can be quite complex and confusing. To allow for an easier explanation of the concepts, (2.1) is reduced to

$$P(\mathbf{x_k}, \mathbf{m} | \mathbf{x_k}), \tag{2.9}$$

removing all conditions, except on the locations of the robot. Conceptually it can be reasoned that there is a dependency between the observations of the robot location and the landmark locations, as they are both directly correlated to each other. Thus, it would follow that we cannot use:

$$P(\mathbf{x_k}, \mathbf{m} | \mathbf{z_k}) = P(\mathbf{x_k} | \mathbf{z_k}) P(\mathbf{m} | \mathbf{z_k}) \tag{2.10}$$

to calculate (2.9). This fact has been verified multiple times by actual implementation [21, 22].

Figure 2.4 not only shows the actual position of the platform compared to the landmarks, but also the estimated location of the platform together with the landmarks. Because the current location of the robot is highly reliant on the previous location, it can be seen that the error between the actual location and the estimated location propagates from each instance to the next. This occurs because the location probability is biased incorrectly, and then propagates to the next location in sequence. The position of the robot and the position of the landmarks are therefore highly correlated. Because this error propagates with each time instant, the distance between two landmarks or robot positions may be very accurate, even if the actual position is not very accurate. In probabilistic terms, this would suggest that the joint probability distribution of both landmarks $P(\mathbf{m_i}, \mathbf{m_j})$ has very concentrated peaks, while the marginal probability distributions $P(\mathbf{m_i})$ and $P(\mathbf{m_j})$ are more evenly distributed. This was one of the greatest breakthroughs in SLAM research. It can therefore be deduced that the correlations between landmark estimates increase monotonically with the size of the observation dataset. For this reason, a map with more sampled data will have greater knowledge about the relative landmark locations.

From Figure 2.4, a time instant k can be related to the robot position at $\mathbf{X_k}$ and three observer landmarks. When the robot moves to position $\mathbf{X_{k+1}}$ during time instant $k+1$, the robot loses vision of the landmark

$\mathbf{m_i}$. However, since the landmarks are highly correlated, the extra observation of $\mathbf{m_j}$ will not only increase the estimation of $\mathbf{m_j}$, but also of $\mathbf{m_i}$, because of the dependency between the current and previous observations. Thus, every observation of a landmark will effect all other landmark estimations. This links all landmarks as shown in Figure 2.5.

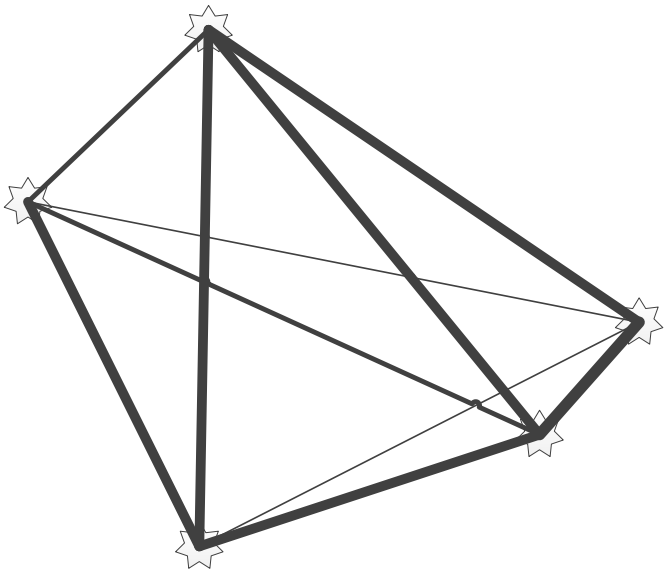


Figure 2.5. Landmark correlation. Each landmark position is not equally dependant on all others.

Although every landmark is dependant to all the others, the observation of one landmarks might have a more profound effect on some landmarks, while a very small effect on others. This occurs when landmarks are observed during the same observation, thus indicating that they are more dependent on each other, compared to landmarks that are observed separately in different time instances. To illustrate this effect, Figure 2.5 depicts the effect that the estimation of each landmark has on the others by the width of the connecting line. Comparing Figures 2.4 and 2.5, it can be seen that each landmark that is observed during the same time instant has a high dependency, while those that are not observed will have a smaller dependency. The effect will deteriorate as the robot moves further away from a given landmark, thus reducing the correlation.

2.3.4 Approximating probabilistic SLAM using filters

The parameters of the SLAM model need to be defined for a particular application. This is achieved by defining an appropriate representation for the observation model and the motion model. These representations must allow for an efficient and consistent computation of the prior and posterior probability distributions. There are two commonly used Bayesian filtering algorithms that a large number of implementations build upon, namely the extended Kalman filter (EKF) method [23] and the Rao-Blackwellised particle filter method [24]. By applying both of these methods to the probabilistic SLAM equations, the EKF-SLAM and FastSLAM techniques are realised. The former method is by far the most popular of the two, but the latter approach, also known as FastSLAM, has received considerable attention in recent years.

EKF-SLAM was the original solution to the SLAM problem and has been the most widely used, but that does not mean that it is the most suitable solution. Comparatively EKF-SLAM does perform the best under supervised conditions, but it lags behind FastSLAM when it comes to feature-rich (also called landmarks) real-world environments. EKF-SLAM calculations scale exponentially as the number of landmarks increase, while FastSLAM scales logarithmically. EKF-SLAM also lacks the ability to handle posterior probability distributions with multiple peaks, where the native linearization with extended Kalman filters will induce large errors. FastSLAM, however, poses its own set of problems. The FastSLAM system tends to degenerate with time, thus becoming less accurate the longer it operates [23].

Great care must be taken when implementing any type of SLAM system with regards to optimisation. Because SLAM operates by correlating landmarks that are similar, the more landmarks there are, the more processing power will be required. Thus, any implementation must be optimised with regards to data, as well as programming techniques. Where possible, redundant and unnecessary processes need to be avoided. There are numerous preprocessing steps that can be applied to the data that can reduce the overall redundant data. The amount of time it takes to complete a single iteration of SLAM can greatly affect the final accuracy.

These two methods encompass some of the current-day working systems, but there are other methods, such as graph-based SLAM [25], and hybrids like occupancy grid SLAM [26] that are becoming more prominent in state of the art implementations. Both EKF-SLAM and FastSLAM themselves are very

basic and would not function properly on the quadcopter system without extensive modifications. For the proposed system the most popular approach is graph-based SLAM.

2.3.5 Graph-based SLAM

Graph-based SLAM was proposed in 1997 by Lu & Milios in [25], but took a very long time to find traction within the research community. The main difference between filter-based SLAM and graph-based SLAM is the way they handle past observations. Filter-based SLAM works on the principle of Markov's assumption : "The future is independent of the past, given the present". Thus based on the information of the current state, it can be interpreted that the current and previous states are independent. This assumption can be seen in the motion model in (2.4). Thus, once a new state is approximated, the previous observation and control sequences will not directly be used again. While this approach is acceptable under some approaches, it is not ideal for implementations where the environment is not known *a priori*. For the envisioned implementation the 3D map of the environment will be built incrementally as the quadcopter moves around. Thus, it needs to be able to remember the spatial relation between each observation, allowing the opportunity to refine those estimates as more information becomes available.

Another problem that filter-based SLAM strives to solve is the problem of the "kidnapped robot". The kidnapped robot problem refers to a robot being placed inside a known environment, but at an arbitrary location. For the envisioned implementation this is not a problem that needs to be solved and thus will not be considered.

Filter-based SLAM is predominantly used when the environmental data is 2D in nature or the environment is mapped in advance [25, 27]. This will typically be from 2D range finders or LiDAR devices. Graph base SLAM has been receiving much more attention with the advent of RGB-D style sensors, such as the Kinect.

2.3.5.1 Graph-based SLAM theory

Filter-based SLAM works by calculating the probability of the platforms location, while graph-based SLAM calculates the relative motion, also referred to as the egomotion estimation, of the

platform between each observation. The system then remembers the calculated transition between each observation, with the hopes of refining those transitions as more information is obtained in subsequent observations of the environment. Thus, filter-based SLAM only estimates the current pose of the platform, where graph-based SLAM calculates and remembers the complete trajectory history of the platform. Graph-based SLAM is also known as a least squares approach to the problem. Thus, it tries to minimise the sum of the squared errors between each successive observations landmarks.

Graph-based SLAM typically gets separated into two sections. The graph-construction (also known as the front-end) and the graph-optimisation (also know as the back-end) [28]. The interaction between the two sections are shown in Figure 2.6

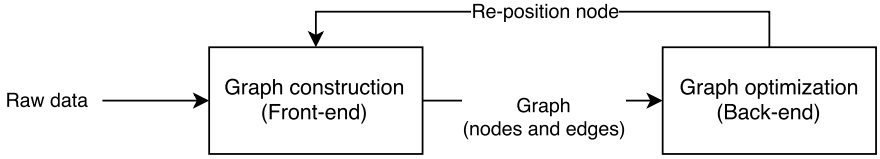


Figure 2.6. Graph-based SLAM showing the front-end and back-end.

The front-end is responsible for building the graph-based representation of the platform's trajectory. This is done by recording the "pose" of the platform at each observation as shown in Figure 2.7. This graph can also be referred to as the pose-graph.

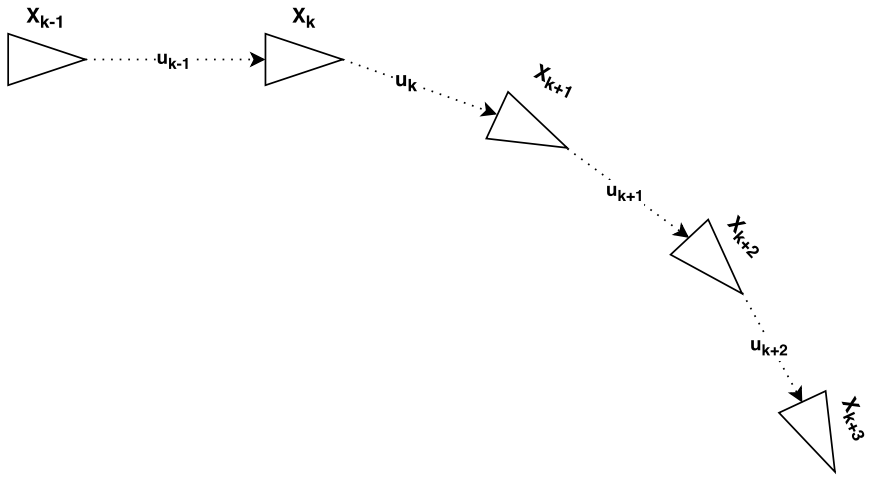


Figure 2.7. Pose graph of five successive observations.

Each pose $\mathbf{X_k}$ is represented by the location and orientation of the platform with regards to a coordinate frame. This position can be represented in 2D or 3D space. Each pose consist of the Euclidean

coordinates, as well as the orientation. The orientation can be given by a rotation in either Euclidean space or using quaternions, typically along the three rotation axis: roll, pitch and yaw. Both Euclidean space and quaternions can be used to represent the same rotation, but Euler rotational matrices can fall subject to gimbal lock [29]. Each pose in the pose-graph is connected by an edge $\mathbf{u_k}$, representing the spatial constraint between them. Each edge thus represents the translation and rotation the platform underwent between two successive observations.

Using the same vectors and datasets defined for Figure 2.4, the front-end's goal is to estimate the pose $\mathbf{X_k}$ so that it best explains the observations made in $\mathbf{Z_k}$. To do this a function $\mathbf{F}$ is defined, that maps the landmarks observed in $\mathbf{X_k}$ to the landmarks observed in $\mathbf{X_{k-1}}$. This is usually done by using image processing algorithms to extract landmarks from both observations, and match them based on mathematical descriptors computed at each landmark location [30, 31].

$$\mathbf{F}(\mathbf{Z_k}, \mathbf{Z_{k-1}}) = \mathbf{Z_k} \rightarrow \mathbf{Z_{k-1}} \tag{2.11}$$

Considering that both $\mathbf{Z_k}$ and $\mathbf{Z_{k-1}}$ consist of a vector with lengths n, the function $\mathbf{F}(\mathbf{Z_k}, \mathbf{Z_{k-1}})$ will return a matrix of size $n_k \times n_{k-1}$. Considering that the number of landmarks in each observation can become excessively large, this approach alone is very inefficient. By calculating descriptors for each landmark, the number of matches between landmarks can be reduced drastically. Descriptors are typically mathematical representations of the immediate area surrounding the landmark, although some approaches use object recognition for landmarks [32, 33]. By using descriptors to filter out the majority of matches between landmarks, the size of the matrix $\mathbf{F}(\mathbf{Z_k}, \mathbf{Z_{k-1}})$ is reduced drastically. At this point most landmarks in observation $\mathbf{Z_k}$ is matched to a small subset of other landmarks in observation $\mathbf{Z_{k-1}}$. It is, however, not uncommon for some landmarks not to have any matches within the reference frame.

Ideally the goal of the front-end is to find the combination of landmarks between observation $\mathbf{X_k}$ and $\mathbf{X_{k-1}}$ that will perfectly match the two observations. Owing to noisy measurements in both observations, there is no perfect match between the two. This will also lead to some landmarks in observation $\mathbf{X_k}$ being matched to multiple other landmarks in observation $\mathbf{X_{k-1}}$.

Because of the noisy measurements and the fact that a 1:1 correspondence between the landmarks is not detected, the solution is to minimise the error between matched landmarks in Euclidean space.

$$\mathbf{u_k} = \mathbf{argmin} \sum_{x,y,z} (\mathbf{F}(\mathbf{Z_k}, \mathbf{Z_{k-1}})) \tag{2.12}$$

Equation (2.12) gives the translation and rotation of observation $\mathbf{X_k}$ so that the detected landmarks $\mathbf{Z_k}$ form matched pairs with landmarks from $\mathbf{Z_{k-1}}$ so that the resulting error metric is minimised. This translation and rotation is the calculated spatial motion the platform underwent between observations, also represented by the edges in the pose-graph. Thus, by taking the pose of the platform at $\mathbf{X_{k-1}}$ and adding the calculated motion $\mathbf{u_k}$, the new position of the platform is estimated.

Multiple factors will contribute to the final error in each pose estimate. There could be discrepancies in the distance measured due to hardware limitations, or the landmark pairs matched were incorrect. These factors will be discussed in greater detail at a later stage, and techniques will be discussed on how to mitigate the size of the error. As more and more observations are made, the error made in previous observations transfers to the next, resulting in an accumulating error that is called drift. The size of the drift is linear with the number of observations made; thus over a large sequence of observations the error can become comically large, resulting in a map that is disproportionate to the real environment. While the platform is operating in a new, previously unseen location, there is no way to correct the drift resulting from previous pose estimations. However, when the platform moves to a position where an observation is made of the same location than that of a previous pose, the potential exists to calculate the accumulated drift and correct for it. This is the responsibility of the back-end in graph-based SLAM and is called loop-closure.

Loop-closure occurs when the platform re-observes a location in the environment. Thus, the platform has captured an indeterminate number or observation and returned to a previously seen location. Once this has occurred there is an opportunity for loop-closure. Figure 2.8 represents this situation.

After an indeterminate number of observations t, the estimated pose is given by the shaded object $\mathbf{Y_{k+t}}$, while the real position is given by $\mathbf{X_{k+t}}$. Thus, the accumulated drift can be represented by the rotation and translation difference between the estimated pose and the real pose.

To implement loop-closure, graph-based SLAM remembers all the landmarks observed during the entire operation time. This dataset $\mathbf{m} = \mathbf{m_0}, \mathbf{m_1}, \mathbf{m_2}, \ddots, \mathbf{m_k}$ contains the observed landmarks for each pose in the pose graph. For each new observation $\mathbf{X_k}$ that the platform makes, the front-end calculates

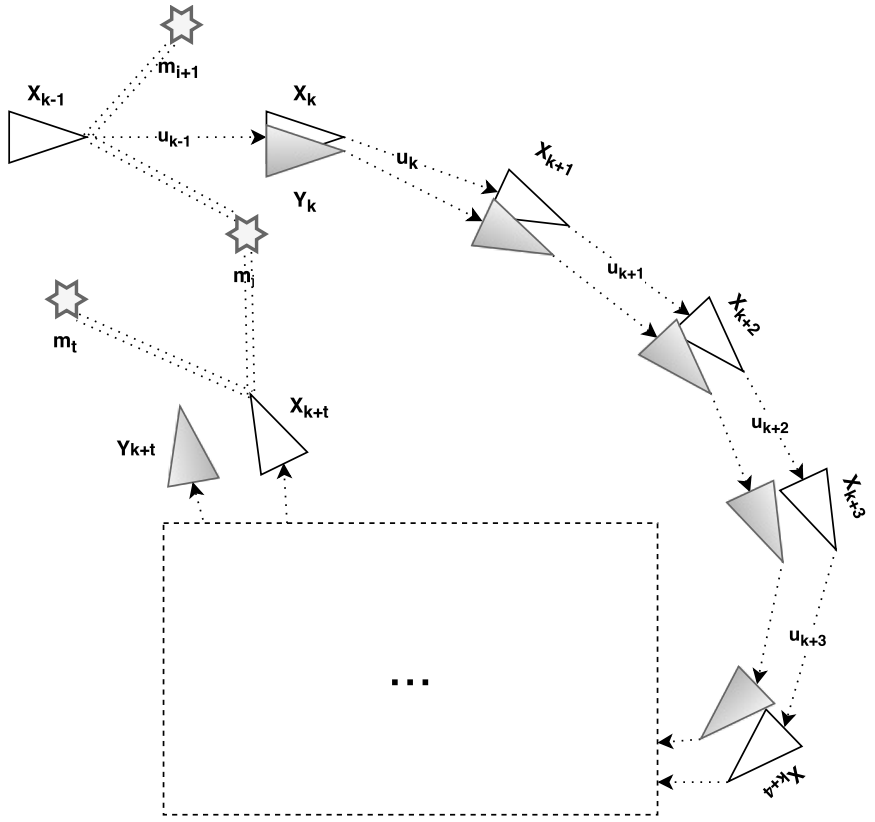


Figure 2.8. Pose graph representing loop-closure. The starting location $\mathbf{X_{k-1}}$ is revisited after a known number of observation. The dashed rectangle represents an indeterminate number of observations.

the relative motion between $\mathbf{X_k}$ and $\mathbf{X_{k+1}}$,and the back-end compares all observed landmarks $\mathbf{Z_k}$ to all the landmarks known in the environment.

If every observed landmark for each pose is added to the dataset $\mathbf{m}$, the spatial complexity will increase linearly, as will the time complexity to compare the current set of observed landmarks $\mathbf{Z_k}$ to $\mathbf{m}$. This poses a major problem, as the number of observations can vary greatly, and could climb high into the thousands. For the moment this problem will be ignored, but it is explained and possible solutions are given later in this section.

At pose $\mathbf{X_{k+t}}$ an observation of a specific landmark $\mathbf{m_i}$ is made. The front-end uses this information to calculate the estimated pose $\mathbf{Y_{k+t}}$. The back-end will then compare landmark $\mathbf{m_i}$ to the dataset of landmarks $\mathbf{m}$. Once the back-end identifies a landmark that it has observed previously, the system can refine the pose estimation of $\mathbf{Y_{k+t}}$ with regards to the previous observation of that landmark, namely pose $\mathbf{X_{k-1}}$. This new estimation will be much closer to the real-world location because it is able to

remove the accumulated drift from all the observations between $\mathbf{X_{k-1}}$ and $\mathbf{X_{k+t}}$. Note that not all the landmarks observed at $\mathbf{X_{k+t}}$ need to match all the landmarks observed at $\mathbf{X_{k-1}}$ for a loop-closure to be relevant.

Even though the discrepancy between $\mathbf{Y_{k+t}}$ and $\mathbf{X_{k+t}}$ has been found and corrected, the error still persists in all the observations between $\mathbf{X_{k-1}}$ and $\mathbf{X_{k+t}}$. This will still lead to a distorted map representation of the environment. To solve this problem, the error is propagated back through each of the poses that accumulated drift. This is done by altering the edge between each pose (egomotion estimation) by an amount proportional to the size of the edge and the size of the drift. Using this method will remove the drift from the system, and the only discrepancy will be the error induced when calculating the position of $\mathbf{X_{k+t}}$ using $\mathbf{m_i}$. The most popular approach to loop-closure detection is fast image matching [15, 34], which will be discussed in detail later in this section.

Graph-based SLAM is a fairly popular approach to the SLAM problem because it has a unique approach to building and localising the robot within an unknown environment. As with the standard probabilistic SLAM, there are various ways of implementing graph-based SLAM in a real-world application. These implementations will vary depending on the platform being used, the sensor being used and the environment it is designed to operate in.

In the implementation described in the dissertation, the Microsoft Kinect sensor will be used, which will limit the desired platform, a quadcopter, from operating in areas where the sensor will not function properly. The Kinect is known as an RGB-D camera, or a sensor that returns two unique images of the environment inside its FOV. The first image is a standard RGB image, and the second is a depth image that gives the distance between the sensor and a grid of points inside the FOV. This type of data is known as RGB-D data and is often used for graph-based SLAM, thus leading to the term RGB-D SLAM. RGB-D SLAM falls in the category of Visual SLAM and can be implemented across any type of platform that has a RGB camera and a range camera that can return 3D data that corresponds with the RGB camera. These two cameras do not necessarily have to be packaged together, but creating a RGB-D setup by oneself can be laborious and require a large amount of fine-tuning.

2.4 RGB-D SLAM

By using an RGB-D style camera for SLAM, a few important drawbacks are introduced to the system with regards to 3D mapping. These sensors are only capable of returning a depth map with a limited distance (typically between $5-7$ m) and a relatively small field of view ($60°$) compared to specialized LiDAR cameras that can return images with a FOV of $180°$ or more. Thus, compared to other graph-based SLAM implementations, the data available is much more constrained; thus the final accuracy could suffer.

Even when utilising the best available hardware, there can still be challenges running any graph-based SLAM implementation in real-time. This is because the sensors return 3D data with corresponding RGB pixel values. Thus, there will potentially be a large number of landmarks in each observation. Given a typical RGB-D sensor running at 30 fps, no SLAM implementation will be able to keep up with the large numbers of frames and landmarks without additional pruning and optimisation.

As previously explained, graph-based SLAM is split into two subsystems: the front-end and back-end. These two subsystems work in tandem to provide the overall SLAM implementation, but they do not necessarily work in serial with each other. Most of the recent real-world implementations run the front-end and back-end systems in parallel. This is because the computation speed per observation is quite different across the two subsystems. Typically the front-end can finish computation in a relatively short time, but the back-end has the tendency to take a significantly longer time because the time complexity is different.

The real-time importance of both systems is also very different. For the front-end to lag behind the real-world platform could lead to a disastrous outcome, especially if the platform is constantly in motion. If the system that defines the location of the platform to the immediate environment surrounding it lags behind, the platform can easily crash into objects.

For the back-end, the end results are refining the position of each previous observation with regards to the global environment; thus it will not affect the frame-to-frame operation of the system that much. Thus, for a real-world RGB-D SLAM implementation the front-end system needs to operate in real-time, while the back-end can lag behind a few seconds without any major ramifications. If there is

significant overlap between successive observations, the back-end can even drop a few observations between each iteration of the system.

2.4.1 Egomotion estimation (front-end)

For RGB-D SLAM three popular methods can be used to implement the front-end egomotion estimation. These three methods are the most prominently used in almost all the current and historical implementations. The three approaches are: inertial odometry, iterative closet point (ICP) and visual odometry (VO).

Inertial odometry is a fairly old approach to the problem of egomotion estimation. Most of the implementations solely using this approach are severely lacking in accuracy and robustness [35]. Implementations using this approach are fairly outdated, and was created when the research field was still young. This, however, does not mean that the method is obsolete, as it can add another dimension of information to a system.

Iterative closest point [36], better known as ICP, was used for one of the first ground-breaking implementations of RGB-D SLAM [37,38]. It is a brute-force approach that can deliver egomotion estimation very accurately, but only under a very limited set of conditions. It is a much more computationally complex solution than inertial odometry, but has proven to be more accurate overall.

Visual odometry is the most acclaimed and widely used method for current implementations [33,38,39]. Although it also has to function under a limited set of conditions, these conditions are much more relaxed than with ICP. It has proven to be the most reliable method for egomotion estimation in systems that are dynamic and not limited to a predetermined set of rules. It is far more versatile than ICP and inertial odometry and allows for much larger datasets to be used over larger areas.

Each of the three methods can operate individually, but they can be used in conjunction with one another to achieve greater success than any one can on its own [39]. Many implementations utilize two or all three of the methods described above to solve the front-end system for RGB-D SLAM to varying levels of success.

2.4.1.1 Inertial odometry

Inertial odometry is the method of using hardware sensors to measure the perceived motion of the platform. Two of the most prominent methods are using wheel-encoders for land-based platforms and IMUs for a more general solution. IMUs are particularly useful because they can tract the velocity, acceleration and rotation of a body in motion [40]. While this might sound like an end-all solution to the problem, multiple problems arise with this solution.

IMU's are prone to fairly substantial amounts of error and drift during normal applications. Even in combination with probabilistic models and Monte-Carlo techniques, the system still returns fairly large levels of uncertainty and drift. Another problem that this approach faces, is that it does not return any information about the environment it is operating in. Thus, this approach can only be used in an area that has already been mapped and where the starting location of the platform is known.

2.4.1.2 Iterative closest point

Basic ICP is a method used to calculate the rigid-body transformation between two sets of points that allow for the best spatial matching of the two sets. The best match would refer to the minimisation of the difference between the two sets of points in 3D space. For RGB-D SLAM the points would be the depth map data, also called a point cloud. This point cloud has 3D data; thus the error metric used is the Euclidean distance between each set of matching points in the respective point clouds.

ICP functions in four algorithmic steps:

- For each point of data in the source point cloud A, find the closest matching point in the reference point cloud B.
- Calculate the best translation and rotation that will minimise the RMSE of the point-to-point distance between each matched point.
- Transform the source point cloud B, using the translation and rotation obtained in the previous step.
- Iterate this process until a set a criteria has been matched.

Although ICP can be seen as a very useful solution to the problem, it suffers from major pitfalls when it is used to calculate the egomotion estimation.

ICP is at its core a brute-force solution to match two sets of data. Thus, this solution has a very large space and time complexity to it. As the size of either of the point clouds go up, the time complexity will grow exponentially. This is a very real problem for any implementation that has to run in real-time, having only a few milliseconds to complete the operation on each frame of data could lead to poor results.

ICP is an iterative solution to the problem; thus the error metric defined is reduced in each iteration. This allows the system to converge to a perceived minimum error with regards to the difference between the two point clouds in 3D space. This minimum error, however, could potentially not be the global minimum, but rather a local minimum perceived as the global minimum. This would provide an egomotion estimation that is different to the real-world motion. This is a problem that persists over many different topics of work, with no perfect solutions. Given an infinite amount of time or computational resources, this could be solved using complete enumeration, but ICP in itself is already a very time-consuming process.

Multiple methods can be used to minimise this problem, but none can solve it completely. Currently the best solution is to provide an initial estimate of the motion that the platform underwent and then align the source point cloud to the reference point cloud using this initial estimate. This would result in ICP only being used to refine the initial estimate. This approach suffers from one major drawback, namely the accuracy of the initial estimation. If the estimation has a large level of uncertainty, there could be the serious error of alignment between the two point clouds.

Another potential problem for most ICP implementations is the level of similarity between the two point clouds. For two sets of data with very little spatial structure difference between the two (thus a high level of correlation between the two scenes presented), ICP should be able to match the two without any problems. In a situation where there is only moderate to minimal correlation, ICP will most likely fail because it is busy matching **all** the points in both point clouds. This could be solved by removing points in both point clouds that are perceived as out of scope for the other, but this is an arbitrary solution, as this would assume the egomotion estimation is already known.

Multiple variations to the ICP algorithm have been proposed and tested over the last few years. The overall robustness of ICP on 3D data has been improved extensively by using point-to-plane techniques or incorporating colour values [41, 42]. Although quite a few have managed to improve the accuracy and reliability of ICP with regards to RGB-D SLAM, none have solved the inherit problems of ICP. All implementations still rely on a suitable initial egomotion estimation and have a fairly large time complexity. Significant research has been done on improving ICP using RANSAC and RE-RANSAC, a process that combines visual odometry with ICP [34].

2.4.1.3 Visual odometry

Visual odometry refers to a method that uses the visual aspects of the data obtained to calculate the egomotion estimation. Visual aspects refer to some type of image processing algorithms that extract information about the environment from the data. This is typically called features in image processing. Features are areas or pixels in an image that are prominent and unique; thus, their colour or intensity separates them from their neighbouring pixels. The full process for basic visual odometry is shown in Figure 2.9.

Finding the location of these features in an image is called feature detection and forms the back-bone of most RGB-D SLAM implementations. The detected features can have a tendency to be prominet to an observer, but most algorithms proposed for feature detection can detect features that are difficult for a casual observer to notice. These features can be used as the landmarks thus far referred to during the SLAM explanations. Not all implementations use the features themselves as the landmarks, as some implementations propose detecting objects and surfaces themselves as the landmarks, only using feature detection as a way of identifying and segmenting objects in the environment [43]. This method has shown significant promise and can provide an extra level of accuracy to most implementations. As most RGB-D style cameras return both the RGB colour image as well as the depth map, both of these images can be used during feature extraction, as they each offer a different source of information.

Feature detection on its own will not be able to provide any type of egomotion estimation between two captured frames, as the features between two images still need to be matched. This process requires what is called a feature descriptor. A feature descriptor is a mathematical representation of that specific feature that has been detected. Thus, it represents the visual effects that distinguish that feature from

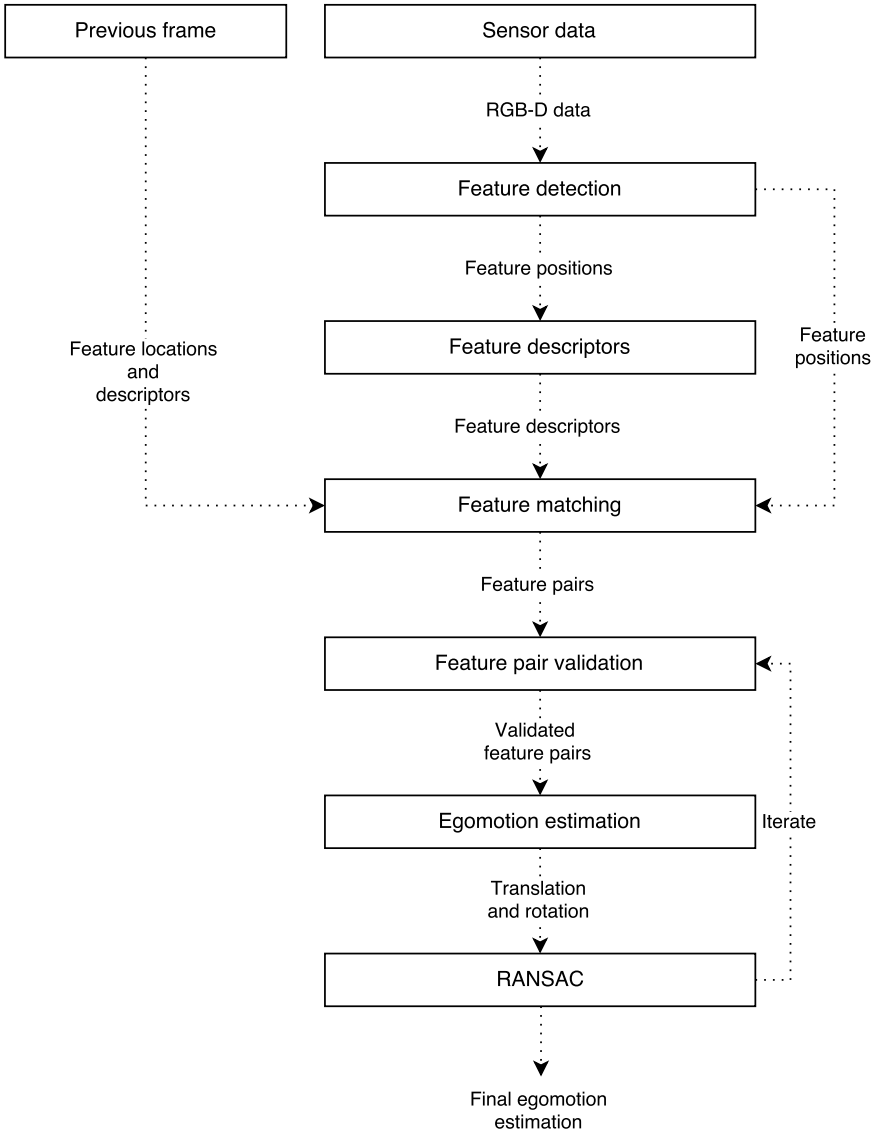


Figure 2.9. Visual odometry procedure. Each image is processed for visual features, and descriptors defining each feature is calculated. There features are validated and matched between the two images. RANSAC is used to estimate the accuracy of the transformation detected between the two images.

its surroundings in a mathematical format. By quantifying the feature's uniqueness, a comparison can be made between two features, calculating the difference between the two features in descriptor space. The descriptor space can vary greatly, depending on the type of descriptor being used.

After each feature descriptor has been calculated, the resulting feature pairs can be identified between the current frame and the previous data frame. This is performed by calculating the distance between each descriptor in descriptor space. Descriptor space is the mathematical representation of each

descriptor, and can be a very high dimensional space. Descriptor space is typically representated by multiple histograms that depict the change in pixel intensity surrounding the feature location. The difference between two descriptors is calculated by taking the difference between the two within that high dimensional descriptor space. There are various ways of defining the difference in descriptor space and it is dependent on the descriptor type itself [44].

Once pairs of features in subsequent frames have been found, the implementation needs to evaluate each pair of features to determine how likely it is that those two features correspond to the same area in the environment. This is achieved, once again, by evaluating the distance between the two descriptors and pruning out any pairs where the discrepancy between the two descriptors is too large. This discrepancy limit will typically fall below one standard deviation, but will also depend on the implementation and data source, as well as the acceptable false-positive ratio. Thus, most of the falsely associated pairs will by removed from the pool of matched features. False associations occur because the best match is determined for every feature detected; thus, every feature will have a match even if the features between observations are not shared. By putting a hard-limit in place for the acceptable distance between descriptors, a large portion of false associations can be removed.

Once an acceptable dataset of pairs is available, projective geometry is used to calculate the rigid-body rotation and translation to match the selected features best. This can be achieved by using the five-point relative pose solution [45], where five feature pairs are used to estimate the relative camera pose between each observation. There are other approaches that only use the three closest matching feature pairs to calculate the relative camera poses, and they show reliable accuracy, provided the feature pair selected were accurate [46]. After the egomotion estimation has been made, it is applied to all features detected so that the two datasets are overlaid against each other in 3D space.

Once the first estimation has been completed, an evaluation of this estimation can be made using RANSAC [47]. RANSAC is typically used to solve data association problems, where a set of falsely associated points is found in a dataset. Thus, RANSAC will be able to further ignore falsely associated pairs and gives an indication of how close the two sets of features match each other in 3D space, as shown in various implementations [48, 49]. RANSAC will provide the number of features that fall within a limited distance from each other, referred to as the number of inliers. The number of inliers will give an indication of how well the two images matched each other. A large percentage of feature pairs that accounts as inliers will indicate a very good match, and will most likely result in a very

good egomotion estimation. If the number of inliers are very low, it does not automatically indicate a bad estimation. This could indicate that the overlap between the two observations is low, or it could indicate that the area is very sparse in either feature space or descriptor space. If RANSAC returns an unsatisfactory result, the process can be repeated with a different set of feature pairs to calculate the rigid-body transformation. This will, however, not always provide a better match, but a few iterations can be attempted before the system decides the two frames cannot be better matched.

2.4.2 Graph-optimization (back-end)

Although the front-end system for RGB-D SLAM is heavily dependent on the sensor being used, the back-end can typically be described as a system that relies on the abstract representation of the environment obtained by the front-end. The graph-optimisation process is a typical non-linear least squares problem. The non-linear egomotion estimation between poses is assumed to be affected by Gaussian noise, which leads to the accumulated error known as drift. The drift can only be detected and eliminated by realising a loop-closure procedure that can estimate the drift error.

Thus, the back-end system consists of two subsystems working in conjunction with each other. The first subsystem scans for loop-closures and calculates the accumulated drift associated with each loop-closure. The second system takes the drift and corrects all the edges in the pose-graph associated with that specific drift. Figure 2.10 shows the configuration of the back-end:

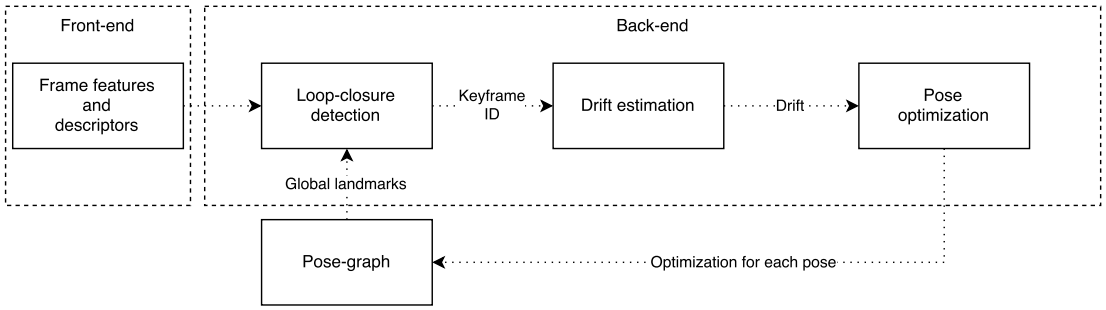


Figure 2.10. Back-end procedure for RGB-D SLAM. The front-end provides the back-end with features, descriptors and pose information at each interval.

2.4.2.1 Loop-closure detection and drift estimation

The loop closure detection and drift estimation subsystem is very similar in nature to the front-end of RGB-D SLAM. The general purpose of the subsystem is to detect when the platform re-observes a previously seen area, and then determine the error associated with all the egomotion estimations made between when the area was first seen and the current observation. This process is shown in Figure 2.11.

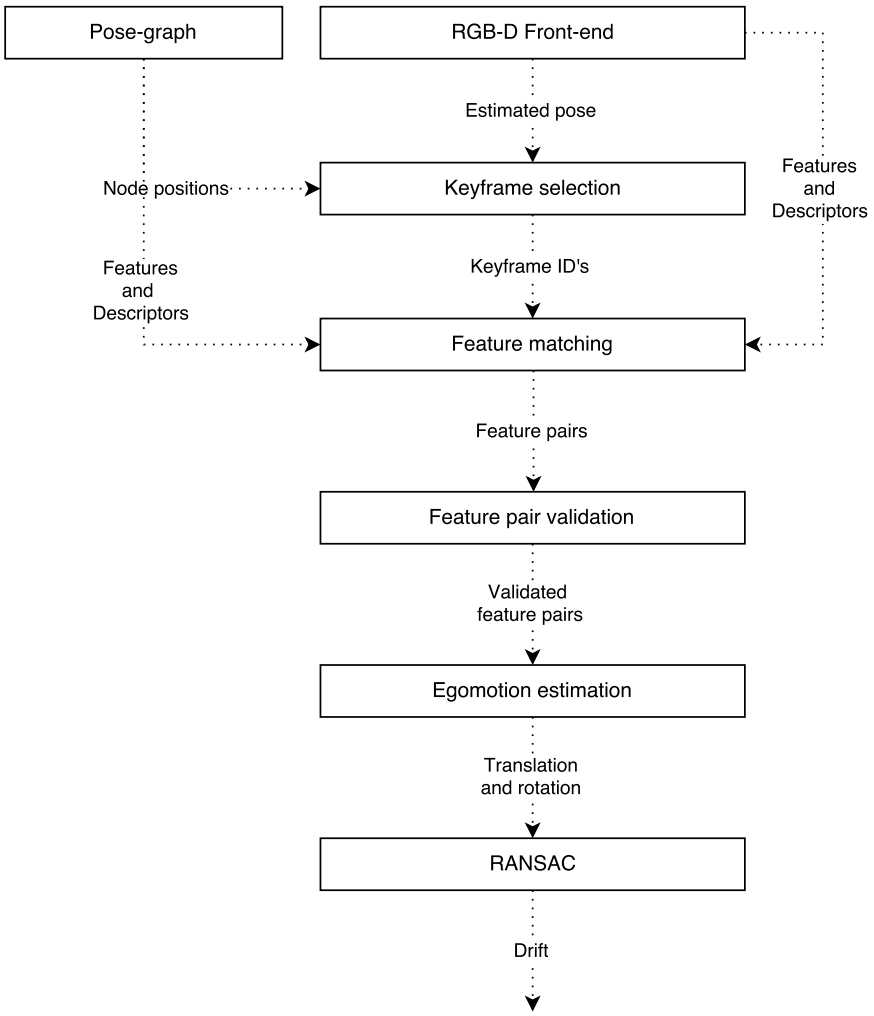


Figure 2.11. Loop closure and drift estimation. This process searches for correlation between the current scene and previously seen locations, thus attempting to "recognise" a location it has already visited

After the front end system has finalised the estimated pose of the current frame in the pose-graph, the frame's feature information and pose estimation is conveyed to the back-end. The subsystem then has

to compare the current frame with previous observations in the pose-graph to determine if the system is currently observing an area that has already been observed. It would be near impossible to compare the current frame with all previous observations in the pose-graph; thus, a method of keyframes are used.

Keyframes are defined as an observation of the environment that do not overlap with previous observations in the pose-graph. Thus, the sequence of keyframes can be seen as a filtered version of the pose-graph, where all the poses that share the same FOV with the previous keyframe, are removed. This significantly reduces the number of observations that are used to detect a loop-closure, while still accurately representing the environment being mapped [50]. The criteria for choosing keyframes can differ from implementation to implementation.

One of the best ways to improve the computational performance of the back-end is to use a probabilistic approach in selecting the order that keyframes are drawn upon. By using the front-end pose estimation as a basic guideline, the order that keyframes are used for loop-closure detection can be rearranged so that keyframes more likely to have matching features, are prioritised. This is achieved by taking the relative orientation and position of the platform into consideration. Thus, keyframes that are closest to the current estimated position of the platform are compared first, while those that are much further away are compared last. The orientation can also be used to load keyframes that are more likely to be in the current field of view. The effectiveness of this process is dependent on how susceptible the overall system is to drift, as well as the distance the platform has moved since the previous loop closure. For a map where loop-closure only occurs after a long time, the number of keyframes in the system will be very large and the drift can become excessively inflated.

The process used to compare the current observation with keyframes is very similar to the process used in the front-end. The features and descriptors from the current observation are compared in a frame-to-frame feature matching process. The features and descriptors are saved for each keyframe and the same process is used to match the descriptors.

It is very likely that an observation detected as a loop-closure will correspond to two or more keyframes, as the platform will not make the observation from the same pose as the one used to create the keyframe. To account for this, once a potential match has been found, the two adjacent keyframes in the chain will also be incorporated to improve and validate the potential loop-closure.

Once a positive loop-closure has been detected, the system follows the same process used in the front-end to calculate the egomotion estimation. Bad feature pairs are pruned away and RANSAC is used to optimise the rigid-body transformation. The rigid-body transformation will then represent the estimated drift the platform accumulated between the keyframe and the current observation. The estimated drift is then used for pose-graph optimisation.

2.4.2.2 Graph optimisation

The previous subsystem detects a loop-closure and estimates the accumulated drift of the platform over the loop-period. This process, however, only corrects the current pose in the pose-graph, while all the previous poses associated with the loop still have the accumulating effect of drift. To allow for a globally consistent map of the environment, the system has to backtrack through each of the associated poses and correct the egomotion estimate between each pose.

The egomotion estimations and the drift are represented by the spatial relationship between two observations, or the change in translation and rotation that the sensor has experienced. The pose of the platform for each observation is represented by (2.13)

$$\mathbf{X_k} = \begin{cases} p_k \in \mathcal{R}^3, & \text{position of pose } k \\ \theta_k \in SO(3), & \text{orientation of pose } k, \end{cases} \tag{2.13}$$

where $SO(3)$ donates a special orthogonal rotation group in three-dimensional Euclidean space.

The edge between each pose in the pose-graph is represented by $\mathbf{u_k}$, as shown in (2.14),

$$\mathbf{u_k} = \begin{cases} \mathbf{D_{k+1}} = \mathbf{R}(\theta_k)^T(p_{k+1} - p_k) + noise, & \text{translation edge} \\ \delta_{k+1} = < \theta_{k+1} - \theta_k + noise >_{2\pi}, & \text{orientation edge} \end{cases} \tag{2.14}$$

where the noise is the drift introduced at each pose estimation and $\mathbf{R}$ is the rotation matrix for each edge. δ and θ is the corrected orientation and detected orientations respectably, and $< \ldots >_{2\pi}$ refers to a 2π modulus operation, bounding the rotation to a $(-\pi, +\pi]$ dimension.

The pose-graph optimisation problem for RGB-D SLAM is a problem of global translation estimation and global orientation estimation. The translation of the platform is in Euclidean space, but the orientation is usually represented using rotation matrices or quaternions. The orientation is usually represented in an over-parametrised way to avoid gimbal-lock (also called singularities), that occur at a few unique rotations where two of the rotation axes align.

Because of the nature of the data, the error induced in each pose estimation is assumed to be Gaussian in nature; thus, the problem can be solved using a least squares approach. This would suggest an iterative optimisation technique that tries to locate the global minimum of the error function. Equation (2.15) shows how the maximum likelihood estimate can be derived by minimising the weighted sum of the drift using the Mahalanobis distance.

$$(\mathbf{p}^{\star}, \theta^{\star}) = argmin \sum_{(k,k+1)\in\varepsilon} ||\mathbf{R_k^T}(\mathbf{p_{k+1}} - \mathbf{p_k}) - \mathbf{D_{k+1}}||^2_{\mathbf{P}^{-1}_{\mathbf{D_{k+1}}}} + || < \theta_{k+1} - \theta_k - \delta_{k+1} >_{2\pi} ||^2_{\mathbf{P}^{-1}_{\delta_{\mathbf{k+1}}}} \quad (2.15)$$

The combined problem of solving orientation and translation, as shown in (2.15), ends up being a hard non-convex optimisation problem. However, if the rotation of each pose in the least squares problem is assumed as correct, the problem is reduced to a simple convex problem that can be solved using a normal least squares approach. Thus, (2.15) can be reduced to

$$\mathbf{p}^{\star} = argmin \sum_{(k,k+1)\in\varepsilon} ||\mathbf{R_k^T}(\mathbf{p_{k+1}} - \mathbf{p_k}) - \mathbf{D_{k+1}}||^2_{\mathbf{P}^{-1}_{\mathbf{D_{k+1}}}}, \quad (2.16)$$

where $\mathbf{R_k^T}$ is known. This, in turn, is solved in closed form using standard least squares.

A standard approach is to solve the global orientation estimation first, and use the resulting pose orientation to solve the full pose optimisation. So, solving the orientation alone becomes (2.17)

$$\theta^{\star} = argmin \sum_{(k,k+1)\in\varepsilon} || < \theta_{k+1} - \theta_k - \delta_{k61} >_{2\pi} ||^2_{\mathbf{P}^{-1}_{\delta_{\mathbf{k+1}}}}. \quad (2.17)$$

Unfortunately, this is not a problem that can easily be solved. It is well-known that optimising poses with regards to orientation is a typical NP-hard problem that is currently receiving a lot of attention in state of the art research. Current implementations use an initial guess as to the orientation change required, and then approximating the problem so that it represents a convex problem by re-parametrisation and iteration.

This field of research is very advanced and beyond the scope of this dissertation; thus, focus will be placed on implementations already available and proven to solve the graph optimization problem with an accuracy that is acceptable for an RGB-D SLAM implementation.

A general approach to solving the problem is using bundle adjustment [51], and can be used to optimise the pose-graph in a map that consists of sparse features. More advanced bundle adjustment algorithms are available that can exploit the sparse connectivity in a pose-graph, also called SBA [39,52,53].

Another valuable contribution to pose-optimisation is the general graph optimization framework, known as **g^{2}o** [54]. This is an opensource implementation that can be utilised for various pose-graph optimisation problems. It was created with graph-based SLAM as one of the intended benefactors, and is openly available for use.

2.5 FEATURE DETECTORS AND DESCRIPTORS

Thus far features and landmarks have been referred to in a conceptual sense, without any discussion on how to detect and describe these features from observations. This section will detail various methods for implementing feature detectors and descriptors. This topic coincides with the field of image processing, as most of the proposed techniques are derivations or complete implementations of image processing algorithms.

As previously stated, RGB-D data is split into two frames of data that coincide with each other. The first is an RGB colour image and the second is a depth map that can be mapped directly to the RGB image. As image processing is done primarily on normal images and not depth images, most of the implementations work on the RGB data and only incorporate the depth data for added information depth.

Currently a plethora of feature detection methods is available, to such an extent that it would be infeasible to list them all. For each method there is also a large number of derivations that are tailored to specific datasets or purposes. Because of this fact, this dissertation will only cover the three most prominently used feature detectors and descriptors implemented in RGB-D SLAM systems. It should also be noted that the detectors and descriptors paired together are not mutually exclusive. Any

descriptor can be used with any detector, but typically the detector and descriptor pair is tailored to each other, working substantially better when used together.

2.5.1 Ideal features

Before discussing the various feature detectors and descriptors, it is important to define the qualities that make a specific detector and descriptor pair suitable for use in RGB-D SLAM.

2.5.1.1 Number of features detected

The number of features detected within a single observation will be subjected to large amounts of variation. Some criteria such as the illumination, number of defining objects and image quality can alter the number of features detected between two images. Most feature detection algorithms are capable of changing the estimated number of features that will be detected by altering the parameters used to find features.

Ideally the process should be able to choose the number of features that should be detected in each observation so that an effective calculation can be performed on the time and space complexity of the RGB-D SLAM implementation.

2.5.1.2 Scale invariant

Scale invariance refers to the ability to match a single location in the environment successfully over two or more observations, independent of the distance of the sensor from that location. Thus, if one observation is 1 m from the detected feature and the second is 2 m away, the two locations should be matched to each other. Thus, the feature detection and descriptor are invariant with regards to the size of objects in each observation.

Scale invariance is highly reliant on the image quality and the parameters that are used to initiate the feature detection. Ideally the implementation should be completely scale invariant, but that is a very unrealistic expectation, given the nature of image processing and the limitations imposed.

2.5.1.3 Orientation invariant

Similar to scale invariance, orientation invariance is the ability to successfully detect and match a single location in the environment over two or more observations, independent of the angle of the sensor to the location. Thus, if the same location is seen from two different perspectives, the detector and descriptor should be able to detect and match that location.

Ideally the feature detector and descriptor pair should be completely orientation invariant, irrespective of how large of an angular difference there is. This is practically impossible, since angular change results in new information surrounding the feature to be seen. Thus, the neighbouring pixels differ depending on the size of the angular difference. This could result in a very different descriptor for the same feature. There are ways of improving this process, but none will be able to work perfectly.

2.5.1.4 Repeatability

The ability to reproduce the same features and descriptors is highly dependent on the two previous properties. The ability to be scale and orientation invariant plays the biggest part in how well a single feature can be tracked across multiple observations. The ability to track a feature over multiple observations is the backbone that RGB-D SLAM is built upon. If the system is unable to correctly track features between observations, the front-end and back-end of pose-graph SLAM will be unable to operate at all. From this it can be deduced that the final performance of the RGB-D SLAM system will be highly dependent on how well the system is able to match features.

2.5.1.5 Time complexity to compute

As the amount of available computational resources increase, SLAM can strive more to achieve compete enumeration. Thus, the more time and computational power is available, the better the solution will be. Unfortunately, both of these are limited with any implementation of SLAM. The time it takes to compute the features and descriptors will dictate the number of frames the system will be able to process per second. As more frames become available, more data will be available for finding and matching features. The effective change in perspective will also be decreased as the frame rate

increases, hence careful consideration must be given to the time complexity when choosing a detector and descriptor pair.

2.5.1.6 Illumination changes and non-rigid transformations

Although the ultimate goal of SLAM research is to create a system that can operate in any environment under any conditions, two conditions can potentially hamper or disable the system. These are illumination changes and non-rigid transformations in the environment. Illumination changes refers to the illumination of the environment changing drastically, for instance a light source being turned off. The second refers to the alteration of shapes or object locations throughout the environment during operation. This would typically refer to moving objects such as human beings. If the location of an object changes during operation, the system is incapable of detecting that the object itself has changed position. This could lead to potential false loop-closures and incorrect egomotion estimations that could cause the system failing in localising itself. To prevent this a controlled environment will be used to test the system. Consequently, during operation the illumination will not change and there will be no movement in the environment.

2.5.2 SIFT

Scale Invariant Feature Transforms (SIFT) is a method for extracting and describing distinctive scale-invariant features from grey-scale images. SIFT was proposed by David G. Lowe [55] as a scale and orientation invariant feature detector and descriptor, proving to be one of the best detector and descriptor pairs available. The SIFT approach transforms the given image into a large collection of local feature vectors. This is the property that makes SIFT one of the leading feature detectors, because the vectors used allow the features to be fairly resistant to scale and orientation invariance. SIFT utilises a four stage filtering approach to detect the features and to create descriptors for each feature.

- Scale-space extrema detection - The first step is used to identify locations in the image that could be detected independently of scale or orientation. This step is performed in scale-space, allowing SIFT its ability to be scale invariant.

- Feature localisation - The second step optimises the location and eliminates any features that have poor positioning, such as points located along edges in the image.
- Orientation assignment - The third step assigns a consistent orientation to each location, based on the intensity of the points surrounding that location. This gives SIFT its orientation invariance property.
- Feature descriptor - The final step computes a histogram descriptor using the local pixel data.

The first two steps are known as the SIFT detector and the final two the SIFT descriptor. SIFT by itself is not meant to be used on RGB-D data, as it is unable to utilise the depth image provided. It only operates on the RGB image of each observation.

2.5.2.1 SIFT detector

The SIFT detector is responsible for finding the locations of potential features in the RGB image. Typically for RGB images, the image is first converted to a greyscale image, as it has been proven that using a proper descriptor negates most of the advantages that RGB has over greyscale with regards to feature detection and matching. The goal is to find features that are not only unique, but will also be recognisable from different viewpoints and scales. This is achieved by operating in the scale space of the image. The scale space is a convolution of the image with a Gaussian kernel with a scale of σ, as shown in (2.18).

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{2.18}$$

where L is the scale space image, G is the Gaussian kernel and I is the image used.

This process is also called "applying a blob filter to the image". This process allows SIFT to be scale invariant, because the maxima of a Laplacian-of-Gaussian (LoG) have proven experimentally to be the best representation of scale.

Unfortunately, the LoG calculations is very expensive; thus, it is approximated using a Difference-of-Gaussians (DoG). Therefore to detect the location of stable scale-invariant features the DoG is used to find the scale-space extrema between images with varying scale k.

Equation (2.19) shows the process of creating a single DoG scale-space representation of the image.

$$D(x,y,\sigma_D) = L(x,y,k*\sigma) - L(x,y,\sigma_D) \tag{2.19}$$

where D is the DoG image.

For each level of k, the images are grouped into octaves, where each octave represents a doubling in the value of σ. The DoG process is easier to compute than the LoG, as it only requires one image to be subtracted from the other. Once all the required DoG images are created, the features are detected by comparing each pixel in the DoG images to its immediate neighbours in the same image, as well as the nine pixels in the neighbouring images within the same octave. If that pixel is the minimum or maximum between the surrounding pixels in all the selected neighbourhoods, it is selected as a maxima or minima and thus a potential feature.

The positions of each feature is refined by using the quadratic Taylor expansion on each DoG image with the original defined location as the center. This calculates the interpolated location of the selected extrema, which has proven to improve the positions of features.

This process, however, returns too many features that are unstable and very susceptible to noise, especially features that are improperly located on edges or that have a low contrast. Features that have a low contrast can easily be discarded by applying a hard limit to the Taylor expansion around the feature. By using a DoG to filter the images, it will make SIFT having a strong response around the edges. To eliminate features that are poorly placed along edges, any feature that has a high edge response is removed. This will remove features that are susceptible to small amounts of noise.

2.5.2.2 SIFT descriptor

Once all the appropriate features features in the image have been found, a descriptor has to be made to represent the feature, using a mathematical model. This descriptor will be used to finally find matching features between two observations. Before the descriptor itself can be created, the orientation assignment process is used to make the feature as orientation invariant as possible. This is achieved by calculating the orientation of each feature, based on the gradient or surface normals surrounding it. These normals are based on the intensity of the pixels, and not the 3D data points associated with them.

Each feature can have multiple orientations assigned to it. For example, a corner feature can have three or more different normal surfaces neighbouring it.

The DoG filtered image $L(x,y,\sigma_D)$ chosen as the maxima or minima for this feature is used for calculations. For each feature the feature neighbourhood donated by $L(x,y)$ is used to calculate the gradient and orientation. Equation (2.20) is used to calculate the gradient magnitude

$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2} \tag{2.20}$$

and (2.21) is used to calculate the orientation.

$$\theta = atan2(L(x,y+1) - L(x,y-1), L(x+1,y) - L(x-1,y)) \tag{2.21}$$

This process is performed for each pixel in the feature neighbourhood. This provides an orientation histogram of the area surrounding the feature. The most prominent orientations are then assigned to the feature.

The descriptor is created using the same local gradient calculations to find the orientation. For each feature a descriptor is created by aligning the surface normals with the prominent gradient detected in the previous step and then weighted by a Gaussian function. A set of histograms spanning a window of 4×4 around the feature is created to represent the surface surrounding it. Thus, a set of 16 histograms is created for each feature. This does not however mean that the window size is fixed, as it can be changed if the descriptor needs to be more accurate. This does, however, significantly increase both the space and time complexity of the SIFT method. It has been shown that the dimensionality of the descriptor directly affects the matching performance [55].

2.5.3 SURF

Speeded up robust features (SURF) [56] is a derivation of the SIFT algorithm, and as the name would suggest, its aim is to improve the time complexity of SIFT. The trade-off is a detector and descriptor pair that is not as well-adjusted for scale and orientation invariance. Comparative analysis has shown

that SURF can compute up to three times faster than SIFT [57], while still producing results that can compete with SIFT under certain conditions [58].

SURF uses the same processes that SIFT employs to detect and describe features. The most prominent difference is the manner in which SURF approximates the LoG operation for the first step. The orientation assignment and descriptors also have alterations to improve calculation speeds. SURF does not employ the Keypoint Localisation step that SIFT does, as it does not have the same edge response problem.

2.5.3.1 SURF detector

SIFT uses a DoG approach approximating the LoG operation, where SURF takes it a step further by approximating the LoG using a Monge-Ampere operator, also called a determinant of the Hessian blob detector. This process can be applied to the target image by calculating three integer operations, but a much faster approach is to calculate the sum of the original image with a precompiled integral image. By using the precompiled image, the speed of the feature detector is increased significantly. The blob filter used is a square-shaped box, allowing the use of the integral image for most implementations.

Similar to SIFT, points of interest are found at different scales, where the scale is represented by the standard deviation of the approximated Gaussian. The scale of the image can be changed by increasing the size of the box filter, resulting in the standard deviation of the approximated Gaussian to be increased. Once again, the scale space is subdivided into octaves, where a single octave contains all the images associated with doubling the standard deviation.

Points of interest are found using the determinant of the Hessian matrix around the point of interest. This is used to measure the level of change in the area surrounding the point of interest, and the feature is chosen at the point where the determinant is maximised.

2.5.3.2 SURF descriptor

Similar to SIFT, before a descriptor can be created for each feature, the orientation is calculated to provide a level of rotational invariance. In order to calculate the dominant orientation of the

neighbouring pixels, wavelet responses are used. By calculating the Haar wavelet response in a circular area around the feature position, the dominant scale-representative orientation can be found. A sliding window approach is used to determine the vector that represents the local orientation. The size of this window can alter the effectiveness of the overall descriptor, so careful consideration must be given to its size.

After the orientation of each feature has been calculated, a descriptor is created by orientating each feature with the dominant orientation calculated above. A variable sized window is projected over each feature where the wavelet responses are, once again, calculated and weighted with a Gaussian. The wavelet responses are taken in subregions within the window, and a histogram is created to act as the descriptor. Once again, the size and number of bins used for the histogram can increase the discrimination of each descriptor, but it will lose robustness and the time and space complexity will increase significantly.

2.5.4 FAST

Features from accelerated segment test (FAST) [59] is vastly different from the two previous feature detectors. FAST is a corner detection method that is used to find the perceived corners of objects within the image. The main advantage that this feature detector has, is its time complexity. FAST features can be extracted from an image in a fraction of the time that it takes SIFT to complete, but it lacks in its ability to be scale and orientation invariant. FAST has comparable performance to SIFT and SURF when it comes to repeatability, under the correct conditions [60].

2.5.4.1 FAST detector

FAST is a typical corner detection algorithm that defines a location as a corner by comparing the pixel intensity values to its neighbouring pixels. FAST operates on the sixteen pixels in a Bresenham circle around the point of interest. The sixteen pixels are systematically chosen and stored in a circular chain structure. FAST compares each of the pixels in the chain sequentially with the point of interest, and if a contiguous set of N pixels are found to either have a higher or lower intensity, the point is defined as a corner. The number of pixels N requires before a set is completed, can be changed to alter the

number of features detected. The intensity threshold can also be changed to reduce or increase the number of features.

Various methods exist for increasing the processing speed of FAST. This includes the High-speed test where only four of the sixteen pixels are first evaluated to judge if the point of interest should be evaluated further. Machine learning can also be used to train the detector on data extracted from the same type of images used as the source data.

FAST is very susceptible to orientation and scale changes, as the size of the Bresenham circle is usually fixed for each implementation. Thus, the repeatability of FAST features are low when compared to SIFT and SURF in a situation with a large scale or orientation difference.

2.5.5 ORB

Oriented FAST and rotated BRIEF (ORB) [61] feature detection and descriptor is a derivation of FAST keypoint detection and Binary Robust Independent Elementary Features (BRIEF) descriptors [62]. It was first proposed in 2011 by OpenCV labs and is described as being an efficient alternative to both SIFT of SURF. The main attraction of ORB is that it is free to use, compared to SIFT and SURF which are patented for commercial use.

Both FAST and BRIEF are known for being very susceptible to orientation changes, but ORB utilised a few extra techniques to make both FAST features and BRIEF descriptors more orientation invariant. Although ORB is not nearly as effective as SIFT or SURF in being orientation and scale invariant, it is much faster to compute. This makes ORB an ideal choice for implementations where the computational power is limited, such as mobile devices or small development boards.

2.5.5.1 ORB detector

Based on FAST feature detection, ORB includes an orientation assignment step for each feature. Using the intensity value of all the pixels surrounding the point of interest, a weighted centroid is positioned with a corner on the feature location. A vector describing the orientation is then taken as the direction from this corner to the centroid's center.

2.5.5.2 ORB descriptor

ORB uses the BRIEF descriptor type, first presented as a stand-alone descriptor [62]. BRIEF is, however, also known to perform poorly under rotation and orientation changes. This problem is addressed by only computing the descriptor using the pixels in the direction of the feature orientation calculated in the previous step. This will allow for a consistent descriptor, given that the orientation stays the same every time.

Because ORB still struggles with orientation changes, the repeatability of the detector and descriptor pair lags behind most other methods. The time complexity, however, is significantly reduced, allowing for a higher frame rates. ORB would be perfect for a system that operates at a high data rate, or alternatively has a low level of computational power.

2.5.6 3D-SIFT

All the previous feature detectors and descriptors utilise either the RGB colour image, or the greyscale image obtained from the range data. None of the methods mentioned use the inherit 3D property of the depth data to detect and describe features.

By applying the standard image processing algorithms to the greyscale image obtained from the depth map, it will not yield very consistent results. This is because all the algorithms rely on the fact that the intensity of a point is not dependent on the position of the sensor with regards to the point, but solely dependent on the point in the environment itself. This does not hold true for the depth map, where the intensity is determined based on the distance of the sensor from the point. Thus, the pixel intensity will change as the sensor changes position.

To account for this, a 3D-SIFT detector and descriptor was proposed that takes advantage of the inhered 3D data of the range image. There are many other types of 3D detectors and descriptors, but the majority have proven to be extremely costly to compute, and will not be evaluated in this disseration [63–65].

2.5.6.1 3D-SIFT detector

The detector still uses the same guidelines that normal SIFT would implement, but instead of using the pixel intensity to define the location, the principal curvature of the point cloud at each point is used. This approach is proposed to work on dense point cloud data, similar to RGB-D data [66]. This approach requires that a surface normal is computed for each point in the point cloud. Thus, after the curvature for each point has been calculated, the standard scale-space DoG approach is used to find the maxima and minima, and potential features are filtered out based on low-contras areas, such as image borders.

The number of detected features can be reduced by occluding features that are too close one another in Euclidean space, as well as by removing features that have a small number of neighbours within a spherical distance.

2.5.6.2 3D-SIFT descriptor

There are two types of 3D descriptors that are recommended for use with 3D-SIFT: Point Feature Histograms (PFH) [67] descriptors and Signature of Histograms of Orientations (SHOT) [68]. PFH uses a nearest neighbourhood search to extract a set of nearest neighbour points at each feature location. Between each neighbour a vector and rotation matrix is calculated to represent the relationship between the two points. These values are added to a histogram that represents the area surrounding the feature. SHOT uses a more complex procedure that creates a covariance matrix for all the neighbouring pixels and then computes the eigenvectors that can best define the local gradient. This produces a more stable local coordinate system for each feature, allowing a better orientation invariance.

3D-SIFT has a very good level of repeatability, if the 3D data has a small amount of noise associated with it. If the noise of the 3D sensor is too high, the 3D-SIFT descriptors will fail to match correctly because of their high dimensionality and reliance on surface normals [69].

2.6 CHAPTER SUMMARY

While there are many other derivations of SLAM that were not discussed in this chapter, they are not required to understand the SLAM implementation used for this dissertation. The majority of SLAM derivations are tailored to specific sensors and tasks, but all of them still operate on the same base approach discussed. For the feature detector and descriptor pairs, only the most prominently used ones were discussed. There is an abundance of them to choose from, especially for 3D features, but given the nature and time limitations, only some of them will be viable solutions.

The next chapter will focus on the implementations of all the systems associated with this dissertation, as well as the metrics used to evaluate performance.

CHAPTER 3 METHOD

3.1 CHAPTER OVERVIEW

This chapter will be divided into four sections, each relevant to the actualisation of the dissertation scope specified. The four sections will be devoted to the following topics:

- Quadcopter assembly - Section 3.2 will focus on the realisation of the hardware platform in the form of a quadcopter. This section will be devoted to the actual construction of the quadcopter, as well as the embedded software implementations.
- Software applications - Section 3.3 will describe the interconnectivity and utility of all applications used for this dissertation.
- SLAM implementation - In Section 3.4 the details of the RGB-D SLAM implementations is discussed. This section will consist of relevant design decisions and methods used to implement RGB-D SLAM.
- RGB-D SLAM evaluation - Section 3.5 will consist of the methods used to obtain and validate the accuracy of the RGB-D SLAM system.

For this dissertation it was decided to use a remote terminal for the SLAM processing. This is due to two reasons: The first is that RGB-D SLAM's time and space complexity is very large, independent of which implementation is used. To be able to operate within the desired time constraint, the system needs access to a large amount of processing power. The second reason is the cost. To purchase a development board that might be able to process SLAM in real-time with the added weight and size constraints of the quadcopter, would turn out to be a very costly expense that could not be justified for this dissertation.

3.2 QUADCOPTER ASSEMBLY

Creating a quadcopter is not a trivial task, but it has become a lot easier than a few years ago. This section will contain all the specific hardware designs used to create the quadcopter system. This quadcopter was designed to operate within an enclosed environment, to receive control commands from a remote source and to provide detailed RGB-D data that can be used for RGB-D SLAM.

To discuss the creation of such a platform, the design procedure for the quadcopter is divided into three subunits:

- Quadcopter base - The actual quadcopter on which all the required hardware is mounted. This quadcopter base was not built from first principles for this dissertation, but an off-the-shelf solution was purchased that allows for a high level of customisation.
- Quadcopter control - The methods and software used to control the quadcopter from a remote source.
- RGB-D Camera - The hardware and software used to provide an RGB-D datafeed for the remote terminal, that can be used as the datastream for RGB-D SLAM.

3.2.1 Quadcopter platform

For this dissertation the quadcopter platform will be bought as a stand-alone product and modified to suit the desired needs. When constructing a quadcopter for the envisioned requirements, one of the main design criteria that must be considered is the total weight of the quadcopter. The maximum weight of the quadcopter is determined by the thrust-to-weight ratio it is capable of. The thrust-to-weight ratio refers to the maximum amount of weight that the motors can hold in suspension. A thrust-to-weight ratio of 2:1 would indicate that the quadcopter is capable of holding double its own weight while stationary. The ratio required is determined by the application for which the quadcopter is being built. Typically, a quadcopter that will be used for slow, meticulous scanning would need a ratio of 2:1. A quadcopter that needs to perform high speed manoeuvres in mid-air would need a ratio between 4:1 and 8:1.

3.2.1.1 Quadcopter base

A DJI F450 base was chosen for this dissertation. This off-the-shelf product consist of the following components:

1. Quadcopter frame - The frame is created from a very sturdy plastic polymer that only weighs 250 g. The wheelbase diameter of the frame is 450 mm, ensuring that the quadcopter is free to move within an enclosed environment. With the propellers attached, the vertical diameter is just under 550 mm.
2. Brushless motors - Brushless motors are used because they can produce more torque than a brushed motor with the same weight and power usage. The motors provided have a power rating of 960 KV. For the power supply used, this translated to a thrust of 800 g per motor. Thus, the total thrust of the quadcopter will roughly be 3.2 kg, assuring a thrust-to-weight ratio of 2:1 for a platform weighing in at 1.6 kg.
3. Electronic speed controller (ESC) - Because brushless motors operate from an AC source, specialised speed controllers are used for each motor. The ESCs provided are capable of 30 A continuous current, and need to be calibrated for each motor.
4. Power distribution system - The platform includes a skilfully created power distribution system that makes the system easy to install and to modify.
5. Propellers - A set of propellers with a diameter of 10 inches and a pitch of 45° are also included.

The fully constructed frame is shown in Figure 3.1.

3.2.1.2 Battery pack

For the platform two battery packs were used. The main battery is used to power the quadcopter itself, and the second is used to power all the development boards and sensors installed on the quadcopter. Two separate batteries were used because the main battery pack is strained heavily during use, and a voltage drop of $20-25\%$ can be seen during power spikes. To prevent this power spike from affecting the micro controllers, the secondary battery is used. This also reduced the overall strain on the main battery pack, extending the potential operating time for the quadcopter.

Figure 3.1. DJI 450 frame

A four cell lithium polymer battery (LiPo) is used as the main battery. Thus, a fully charged battery produces 16.8 V before load. This equates to a 800 g thrust for each motor attached. Two main battery packs were used during construction and testing. A 4200 mAh battery is used for development to extend the potential operating time of the quadcopter, making the development process easier as the platform can operate up to 10 minutes before the batteries need to be charged. This battery does, however, make the final platform overweight and, therefore, a smaller 3000 mAh battery was used for the final testing. This stabilises the quadcopter better, but has a reduced operating time of just over 7 minutes.

The secondary battery mounted on the platform is a three cell 2200 mAh LiPo, used to power the two development boards and the main vision sensor. This battery can supply power for just under an hour before it requires a recharge. During most of the development time the development boards needed to be powered, but not the quadcopter itself. A switching power system was created where the user can switch between the battery pack or an external power source, allowing for easier development and testing.

3.2.2 Quadcopter control

The two main logic units that the quadcopter needed, are the attitude and position controller, as mentioned in Chapter 2. Both these systems were implemented using development boards purchased and mounted to the quadcopter.

3.2.2.1 Attitude and platform controller

For both the attitude and platform controller an off-the-shelf solution was used. 3D Robotics provides a module called the APM 2.6 that is capable of performing the tasks of both the attitude and platform controller.

The platform controller constantly updates the ESC controllers with the desired power levels for each motor. This information is provided through a Pulse Width Modulated (PWM) signal. The APM 2.6 computes the correct PWM signal to send to each ESC from the respective input controls.

The attitude controller is a more complex system and utilises the embedded IMU sensor on the APM 2.6. This IMU is used for accurate real-time stabilisation of the quadcopter. This property allows the quadcopter to function, and is integral to stable flight. The APM 2.6 allows the user to calibrate the attitude controller so that it can function for a variety of different sized frames and frame orientations.

The APM 2.6 is intended to be controlled with a radio transmitter. A radio transmitter and receiver pair operating on 2.4 GHz is the typical position controller for a quadcopter, but for this dissertation the position controller is replaced by a computer system. This system has to interact with the APM 2.6 to control the quadcopter.

3.2.2.2 Position controller

The substitute used for the position controller is the SLAM implementation's mapping function and user commands. For the quadcopter to be able to receive control commands a wireless solution is used. A small wireless board that can connect to a wireless network, was placed on the quadcopter and

runs a specialised program that can receive commands via the network. The development board then simulates the radio receiver's PWM commands to control the quadcopter.

An Arduino Uno Wifi development board was used to fill this position, shown in Figure 3.2.

Figure 3.2. Arduino Uno Wifi Rev 1.0

The Uno Wifi is a relatively new board that adds wireless capabilities to the popular Arduino Uno. This development board was chosen because of its versatile design and the ability to generate PWM signals for the APM 2.6 system. Unfortunately the system was not capable of TCP/IP connections at the time, thus a small web server was created to run on the Uno Wifi. This web server receives URL requests with the relevant control commands embedded in the URL link. The web server processes each URL into the required roll, pitch and yaw that the quadcopter need to adopt. This approach works well, as the web server can handle up to fifty requests per second, far outpacing the possible observations per second that the camera system can make.

One possible problem that was encountered with this implementation was that the web server sometimes drops a request and then the client making the request has to time-out before another can be made. To counter this problem, multiple web clients are initiated to send URL requests in a rotating pattern. Thus, if one client is stuck while waiting for a time-out, the others will still be sending the latest control commands to the quadcopter.

3.2.3 RGB-D Camera

For this dissertation a Microsoft Kinect will be used as the mounted sensor that is capable of capturing RGB-D data. The Kinect's original purpose is to track the movement of human limbs in real-time using an infrared sensor that captures 3D images of the environment in front of it. The release of this device sparked a significant amount of interest from the research community because of the potential that it could have in other applications.

Since the release of the Kinect, a large amount of work has already been done, using this device. Microsoft has released a software development kit (SDK) to support and encourage development using a Kinect. Because the device is readily available and has a large amount of documentation and datasets, it is preferred over alternative devices that can perform the same functions. An opensource library called OpenNI and corresponding Kinect drivers can be used to interface the Kinect with an Ubuntu 12.04 OS.

The Kinect is an affordable sensor that has the capability of producing high-quality images. However, there are a few problems with the Kinect. The Kinect itself is rather heavy and has to be modified before it can be used on the quadcopter. It also has a physical limitation on the distance at which the sensors can operate. Any object within 0.7 m will not be captured at all and could potentially affect the reliability of the quadcopter's operation. Objects that are further away than 7 m will be prone to a very large signal-to-noise ratio (SNR), which might also cause detrimental effects on the performance of the SLAM implementation. The Kinect is also sensitive to sunlight, due to the infrared light from the sun interfering with the infrared sensor on the Kinect, which will completely distort the depth image that the Kinect returns.

The Kinect can return data with a resolution of 640×480, for both the depth map and RGB image. The Kinect can capture data at a maximum frame rate of 30 fps. This could potentially allow the position controller to update the ideal position once every 34 ms. This property, however, poses another problem. Each scan that the Kinect produces is 1.536 Mbytes in size. At a full frame rate of 30 fps, the Kinect produces 46 Mbytes/s, which is a prohibitive amount of data for most types of wireless communications to handle.

The cost of the Kinect is relatively inexpensive for its functionality. The Kinect might not be able

to capture environmental data at long distances, but it is perfect for an indoor environment. The limitations will, however, prevent the Kinect from operating outside or near any objects that radiate infrared electromagnetic waves. Considering that the Kinect can also be stripped down to a device that weighs 115 g, as shown in Figure 3.3, it would not add much unnecessary weight.

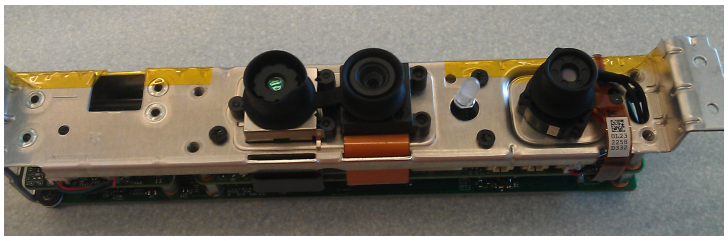


Figure 3.3. Stripped Microsoft Kinect.

Currently, alternatives to the Microsoft Kinect are available. The ASUS Xtion and PrimeSense Carmine provide much of the same capabilities as the Kinect, but in smaller packages. While these alternatives were originally much smaller, they are comparable in size to the stripped-down Kinect. The Kinect was used for this dissertation because one was readily available, being used in similar implementations before.

3.2.4 RGB-D datastream

The final piece of the quadcopter platform that needs to be discussed, is the RGB-D datasteam that has to be captured from the Kinect sensor. The Kinect requires a USB 2.0 connection to either a Windows or Linux OS. Because of the size and weight limitations of the quadcopter, a small development board is the only choice. A standard Pandaboard is used as the interface for the Kinect, as the board is capable of capturing the datastream from a Kinect, as well as providing wireless communication.

3.2.4.1 Pandaboard

Even though the Pandaboard has been available for a long time, it is still an effective development board. It operates using a 1.2 GHz ARM processor and has 1 GB RAM available. An onboard wireless module is available, but the performance is very poor compared to present-day wireless networks.

The Pandaboard was first intended to also function as the relay between the remote terminal and the APM 2.6 system, but this approach was replaced by the Arduino Uno Wifi because of the latency induced due to large amounts of data being sent over the connection. By separating the two systems the latency induced due to high traffic is contained to the Pandaboard, while Uno Wifi can operate at ideal latencies.

The Pandaboard only has one objective, namely to provide a robust wireless medium for a remote terminal to connect to, as well as to transfer the RGB-D data. This process turned out to be more difficult than expected, as the Pandaboard has some limitations. A copy of Ububntu 12.04 Server for ARM is used as the OS for the Pandaboard. The server edition is used because the GUI of the normal Ubuntu distribution will not be used during operation, and it slows down the system considerably. Using the server distribution, the Pandaboard is able to poll the Kinect data at a near 30 fps. However, this operating system reached its end of life a few years ago, so the driver support is very limited. There are very few other options when it comes to operating systems, because of the ARM CPU and Kinect drivers. An old version of OpenNI and Primesence Kinect drivers are used to interface the Kinect with the Pandaboard.

Because the system will be mounted to a quadcopter, it would be infeasible to use an ethernet connection to transfer data to the remote terminal. While the Pandaboard has an onboard wireless connection, the connection is extremely slow, running at a maximum of 8 Mbps. To counter this problem a USB wireless access device is used. A 300 Mbps TP-LINK wireless USB module is used to send the Kinect data wirelessly. This posed additional problems as there are no supporting drivers for this device (or any similar devices) that can run on Ububntu 12.04 ARM. Through advice from online Linux enthusiasts the current drivers for the device were adapted to function for Ububntu 12.04 ARM. Owing to the USB bus speed and wireless limitations the best possible speed that can be achieved using this device is 120 Mbps (15 MBytes/s), which is a far cry from the 370 Mbps required to send a full 30 FPS. Thus, this system is only able to send RGB-D data at roughly 8 frames per second.

3.2.4.2 Datastream

The datastream containing the Kinect RGB-D data can be accessed through a TCP/IP connection. Two TCP/IP sockets are available, one for the RGB image and one for the depth map. The RGB image

consist of three bytes representing the red, green and blue values of each pixel. The depth map consists of a 16 bit unsigned integer for each depth pixel. Two separate connections are available to allow the user to potentially only use one datastream, thus almost doubling the other stream's frames per second. Another approach that can be employed is to capture multiple RGB images for each depth image. This is because RGB-D SLAM operates mainly on the RGB data. Each observation of both RGB and depth data has an accompanying time stamp to match the frames with each other later, as they are not captured at precisely the same time.

3.2.5 Full quadcopter platform

The full system is shown in Figure 3.4.

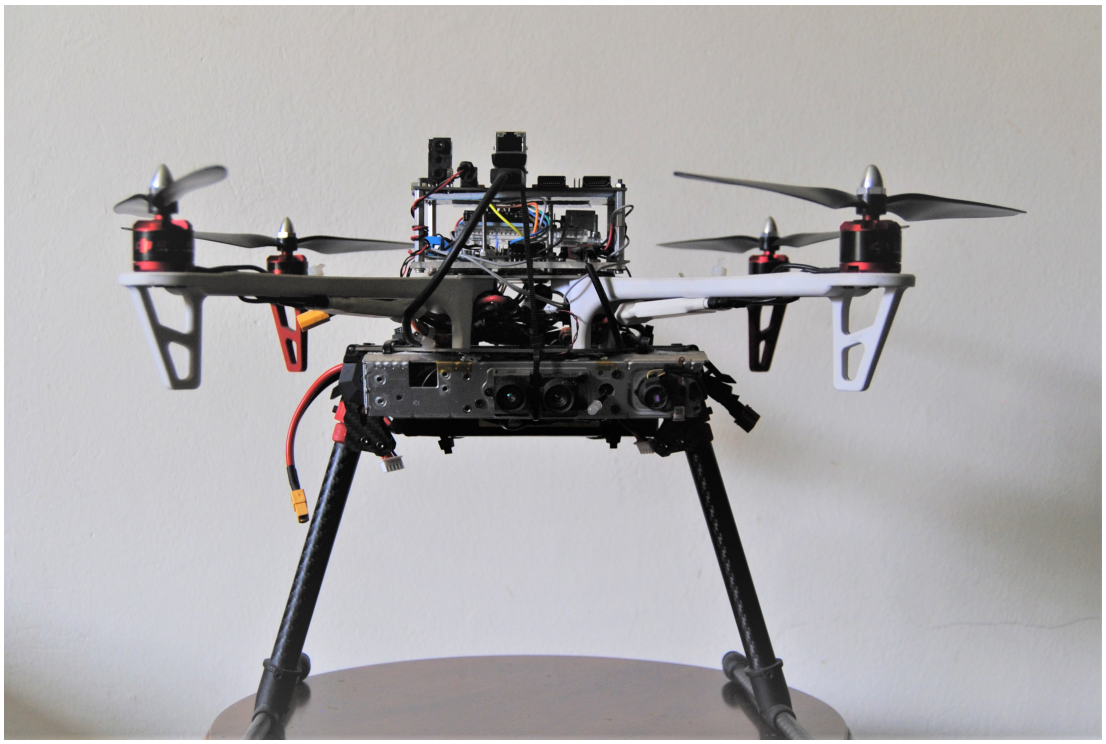


Figure 3.4. Full quadcopter system.

The system is capable of flight for just over 10 minutes before the voltage of the LiPo batteries drop too low for stable flight. This puts the LiPo cell voltage at 3.4 V, still well above the minimum of 3 V per cell. Two voltage regulators, a 12 V and 5 V regulator are also used to power the Pandaboard, Arduino and Kinect sensor. Small pads of velcro are used to isolate all the electronics from micro vibrations in the fame, reducing the noise experienced by the IMU and Kinect.

The complete system using the 4200 mAh LiPo weighs just over 1.9 kg, putting the system over the 2:1 thrust-to-weight ratio. By using the smaller 3000 mAh battery the system weighs 1.65 kg, putting it very close to the ideal 2:1 ratio.

3.3 SOFTWARE APPLICATIONS

For this experimental setup, two applications are running on the quadcopter platform, one for each of the development boards mounted to the quadcopter, as well as four applications that cover a range of tasks on the computer terminal used for SLAM.

The quadcopter software comprises of two applications, one application for the Arduino and one for the Pandaboard.

- Arduino - This is a small application that functions as the wireless medium between the remote terminal running the SLAM implementation and the quadcopter.
- Pandaboard - This application captures the Kinect data, registers the RGB and depth image to each other and sends the resulting data frame over a wireless connection.

The remote terminal has four applications designed to run on it, but not all four applications are required to run at the same time. Two applications serve the purpose of creating the datasteam that is used for the third, the SLAM implementation. The final application is a visualiser that renders a specified point cloud in real-time, allowing the user to see the 3D perspective or 3D map being constructed in real-time.

- Quadcopter control - This application was designed to work with the quadcopter platform created for this dissertation. The application connects to the Pandaboard and retrieves the depth and RGB images, while also connecting to the Arduino to send the most recent control commands. The RGB-D data obtained is shared with the SLAM implementation to receive up-to-date location data.
- Offline datastream - This application was created to fulfill the role of performance evaluation for this dissertation. This application accepts RGB-D datasets created by the research community and creates an RGB-D datastream that mimics the datastream from the quadcopter. This

application connects to the SLAM implementation using the same process as the Quadcopter control application.

- SLAM implementation - This application is the main SLAM implementation. This application is created as a generic black box system, where it receives an RGB-D datastream and returns the location data and pose-graph for the current datastream. The system also saves the pose-graph created during operation to allow for 3D map reconstruction.
- 3D visualiser - This application is created to display the relevant 3D data in real time. This application connects to the SLAM implementation and periodically receives various point clouds to display.

Figure 3.5 shows a diagram depicting the relationship between each application.

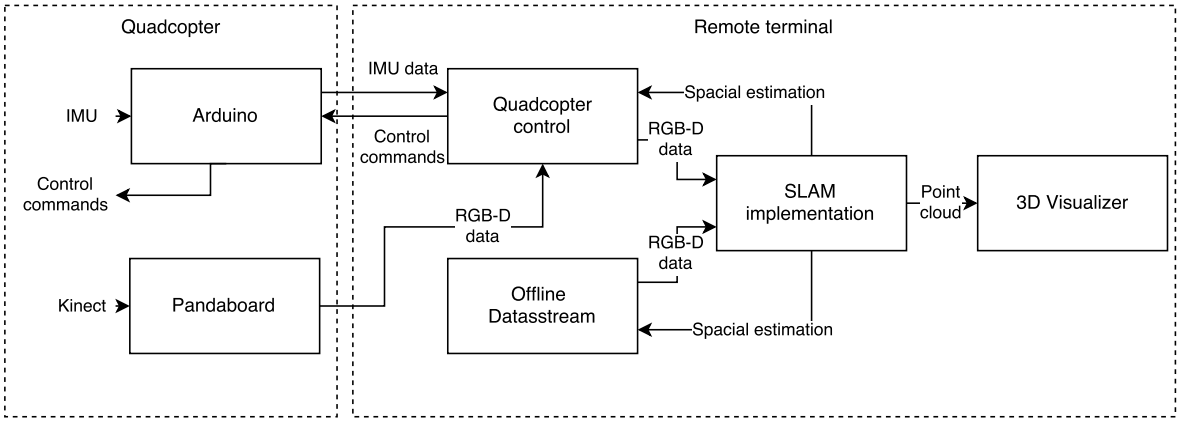


Figure 3.5. The full experimental setup, showing the interaction between the quadcopter platform and the remote terminal.

3.3.1 Arduino embedded application

The Arduino hosts a representational state transfer (REST) webserver designed to be queried, using the standard HTTP request-response protocol. This application is designed to receive the quadcopter control commands over a wireless connection, and return the IMU inertial measurements. The Arduino is capable of connecting to any WiFi network, accepting PUT methods from any source on the network. These PUT commands are used to update variables that are maintained on the REST webserver, and these variables in turn represent the roll, pitch, yaw and throttle of the quadcopter. The Arduino constantly translates these roll, pitch, yaw and throttle variables to PWM signals that are sent to the appropriate pins connected to the APM 2.6 board.

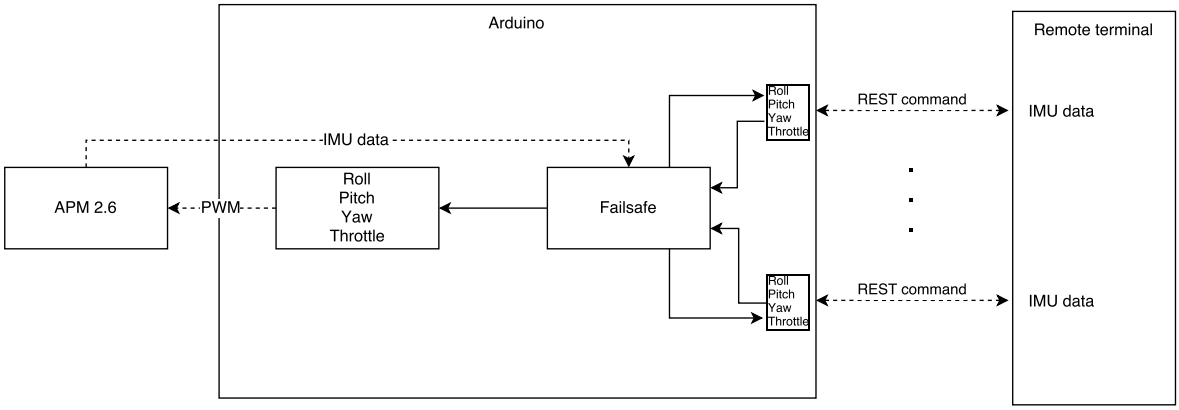


Figure 3.6. Arduino application diagram showing the interconnectivity between the APM board, Arduino and the remote terminal.

Thus, a simple PUT command sent to the REST server is used to control the quadcopter. The system has a simple design where the current IMU measurements are returned for every command sent. The system does not differentiate between clients, and accepts all connections as long as they adhere to the predefined command structure. The system is capable of handling 50 requests per second, allowing for an updated control command every 25 ms.

A hardlimit is placed on the variables representing the roll, pitch, yaw and throttle of the quadcopter. This is to prevent the system from jerking the quadcopter due to a problem in the position controller. Thus, only minor levels of movement are allowed, slowing the quadcopter down to increase safety. If the Arduino detects that no new positioning data is being received, the system quickly reduces the throttle to zero, forcing the quadcopter into landing. This is done to minimise any potential damage that the quadcopter might sustain. This can occur when the position controller software has crashed on the remote terminal, or the quadcopter has moved outside WiFi connectivity range.

3.3.2 Pandaboard application

The Pandaboard application is written in C++, and can be set to automatically begin on system start-up. The system continuously searches for a TCP/IP server to start on a predefined IP address. This is the IP address of the remote terminal running the SLAM implementation. Once a connection has been made, two sockets are created to send the RGB and depth data respectively.

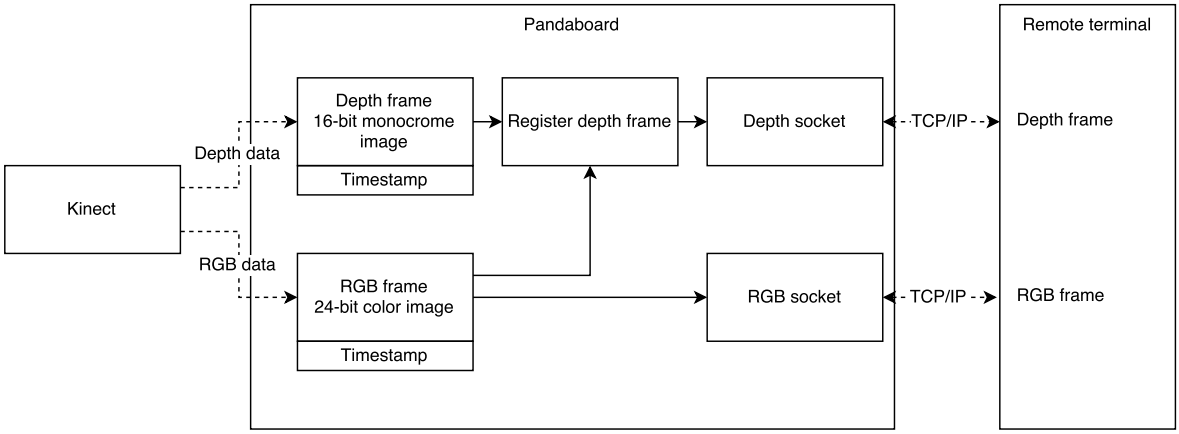


Figure 3.7. Pandaboard application diagram showing the connectivity between it and the remote terminal.

The RGB and depth datastreams are disconnected from each other to allow for each to be requested individually. This separation is made because the wireless connection used is limited, only allowing for a portion of the frames captured each second to be sent to the remote terminal. Thus, not every RGB frame must have an accompanying depth frame, allowing for a larger number of RGB data frames per second.

For each frame an accompanying timestamp is provided, partly to help synchronize the depth and RGB data; partly to link the IMU data obtained from the Arduino to each frame. This is required because the IMU data is obtained through a separate medium; thus, it will not be synchronised with the data from the Pandaboard.

An XML script is used to configure the OpenNI software used to capture the Kinect data. Because of the Kinect hardware itself, the Kinect was mounted upside-down on the quadcopter. This was done because there were no appropriate mounting points on the topside of the stripped-down Kinect, and a heatsink prevented the Kinect from being mounted straight. Using the XML script, the data obtained from the Kinect has already been corrected, without any code required.

Because the RGB and depth images are not perfectly aligned, to use the data in combination with each other, the images need to be registered to each other. The OpenNI drivers allow for the depth image to be reprojected onto the RGB image, but it adds to the overall preprocessing that has to be done, reducing the number of frames that can be processed each second.

Potential data compression and reduction techniques were investigated to increase the frame rate. All the potential lossless and lossy compression techniques were evaluated and tested in a paper published by the author [1]. Unfortunately, because of the limited processing power available on the Pandaboard, the compression techniques take up too much time, making the approach infeasible to implement.

3.3.3 Quad control

This application is the central hub of information between the quadcopter, the SLAM implementation and the user. The application is created using C# and WPF was used for the GUI. This application serves four purposes:

- Quadcopter communications - Connect to the software running on the quadcopter and manage the internal synchronisation between data.
- SLAM interface - Interact with the SLAM implementation, sending the RGB-D data and receiving information pertinent to the quadcopter.
- User interface - Provide a user interface that can be used to understand and evaluate the current quadcopter state. This interface also provides the option to operate the quadcopter within the environment.
- Quadcopter pose - Calculate the correct position the quadcopter has to assume to reach a desired goal state.

Because of the purpose and design of the Quad control software, the application is heavily threaded to ensure that a single system not working properly, will not pull down the entire chain of functions. The full application diagram is shown in Figure 3.8.

3.3.3.1 Quadcopter communication

This portion of the application connects to the Arduino and Pandaboard, as shown in the previous two sections. Two threads are used to connect to the Arduino, to prevent a single thread that is timing-out from halting all position control commands. The actual calculations for the correct roll, pitch, yaw and throttle are discussed in Section 3.3.3.4. Thus, for each thread a single iteration reads the latest roll,

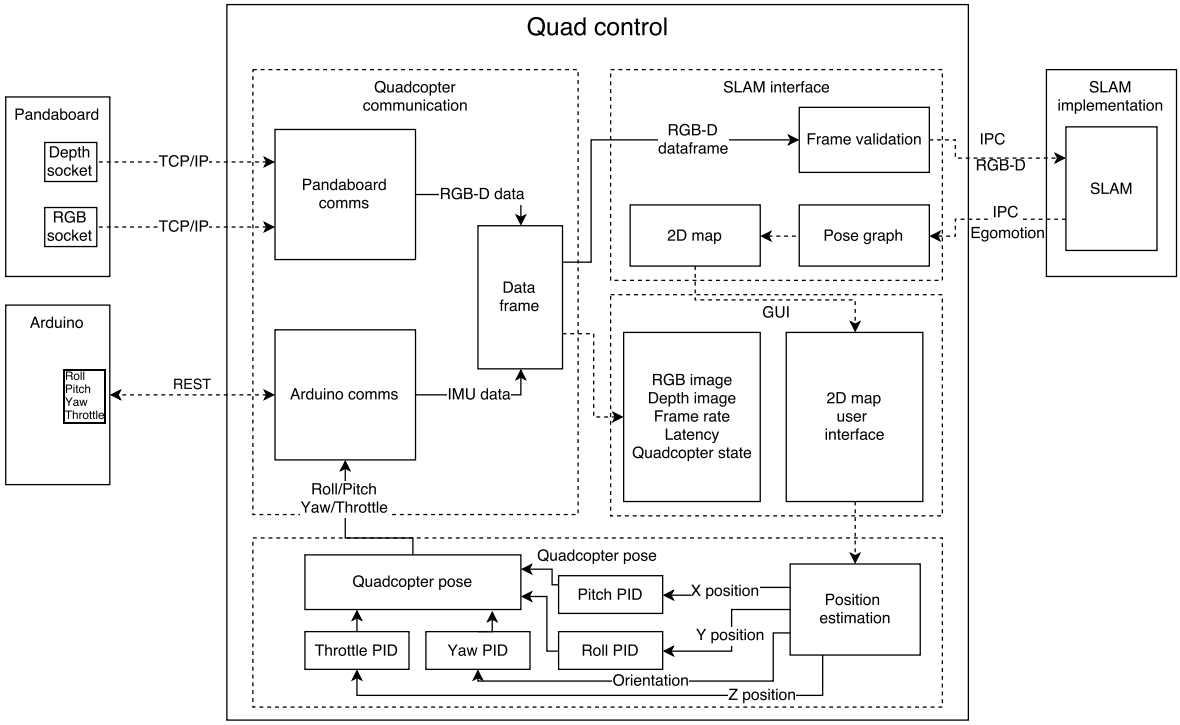


Figure 3.8. Quad control application diagram showing the fully connected system that will control the quadcopter.

pitch, yaw and throttle values and creates an HTTP PUT command to send to the Arduino. The URL is sent to the Arduino and the thread waits for the Arduino to respond with the IMU readings at that time. Once the IMU values are received, the IMU data is saved with a timestamp to be matched with data coming from the Pandaboard.

The application also uses two threads to retrieve data from the Pandaboard. The first thread is responsible for retrieving the depth data with the corresponding timestamp. The second thread is responsible for retrieving the RGB data with the corresponding timestamp. This C++ class is created in such a way that the Quad control application can request RGB and depth frames separately. For each frame received, the class does a simple validation test to check if the data obtained (both RGB or depth) is valid. This is required because some frames captured by the Kinect can be completely blank, most likely resulting from the older drivers being used on the Pandaboard.

3.3.3.2 SLAM interface

Once a stable connection has been created to the Quadcopter, the user can launch the SLAM implementation from the Quad control application. This will automatically inform the SLAM implementation that the quadcopter platform will be supplying the RGB-D datastream.

The Quad control application and the SLAM implementation communicate with each other using inter process communication (IPC). There are many ways to implement IPC, and three methods were tested. TCP/IP using the loopback address, named pipes and shared memory were all three evaluated, based on speed and latency. For all three methods results were obtained that fell well within the requirements, thus all three could be used. For this dissertation shared memory will be used for all IPC between applications running on the remote terminal.

The Quad control application initialises the SLAM implementation with the pixel size of the data that will be used. For the Kinect data this is 640×480 for both the depth and RGB image. A memory block of size 1 536 000 bytes is allocated for IPC, with an additional 64×24 byte region for the pose-graph and 2D map. Each time the application received a data frame from the quadcopter, the data frame is written to this memory block and the appropriate flag is triggered to inform the SLAM implementation that a new data frame has arrived. The application does not wait for the SLAM implementation to return the current position estimation, but rather starts retrieving the next data frame. If the position estimation is received before the next frame has been received from the Pandaboard, the application stalls until a full frame is available again. If the SLAM implementation has not finished operations before the next frame is available, that frame will be dropped and a new frame will be requested. Thus, only the latest data frame will be presented to the SLAM implementation.

3.3.3.3 User interface

The user interface is created using C# WPF. The interface is designed to give the user all the information available with regards to the quadcopter's state and position. A direct feed for both the RGB and 16-bit greyscale depth image is provided, as well as the relevant information with regards to the connection with the quadcopter. The latency for both the Arduino and Pandaboard is shown in real-time, and an accurate frame to frame FPS counter is implemented.

The user interface offers the option to arm the quadcopter, sending the signal to the APM 2.6 board that the quadcopter is intending to take off and fly. Once the system has been armed, the user can directly control the quadcopter using a standard computer mouse and keyboard. The mouse is used for roll and pitch changes, while the keyboard is used for throttle and yaw control.

Once the IPC connection has been made to the SLAM implementation, a secondary control option is available. The SLAM implementation returns the current location of the quadcopter, as well as a 2D top-down view of the known area surrounding it. This 2D map shows the surface area of what the SLAM implementation perceives as the ground. Without any other instructions, the quadcopter will try to maintain a height of 1.7 m in the exact location of take-off. A series of PID controllers are used to achieve this. The user can alternatively specify a goal location within the 2D map, and the quadcopter will first change orientation to match the desired location, and then adjust the desired x and Z coordinates for the PID controllers. The quadcopter will steadily make its way to the desired location.

Once the user has been satisfied and wishes to terminate operation, a signal can be sent to the quadcopter to land. This is simply achieved by steadily reducing the throttle of the quadcopter over a short time period, and once the system has estimated that it is close to the ground, the system stalls all four the motors. The throttle cannot simply be reduced to zero over time, as this has proven to be dangerous. There is a large area of throttle where the rotors are still spinning, but the quadcopter is not capable of flight. Thus, if the quadcopter does touch down at a slight angle, this typically leads to it tipping over, damaging the rotor blades and injuring any individual nearby.

Figure 3.9 shows a screen shot of the Quad Control application, while Figure 3.10 shows a screen shot of the 2D map that is used to input waypoint directions for the quadcopter.

3.3.3.4 Quadcopter pose

To assume the role of the position controller for the quadcopter, the Quad control application creates a goal state that the quadcopter needs to assume. This goal state is represented by a set of Euclidean coordinates and a desired orientation. The initial starting position is taken as the goal state, if no

Figure 3.9. This image shows a screenshot of the Quad control application. Both the depth and RGB images are displayed, as well as all the relevant information pertaining to the current Quadcopter state.

other user designated waypoints are present. Waypoints can be provided by selecting a position the quadcopter needs to be at on the 2D floor map of the area.

A series of PID controllers takes the current x,y,z position and the relative 2D orientation from the SLAM implementation, and calculates the respective roll, pitch, throttle and yaw to move the platform to the goal position. A single PID controller is used for each of the principal axes and one is used for the 2D orientation. The 2D orientation is used because a 3D orientation will be directly dependent on the the immediate movement of the quadcopter, interfering with the other three PID controllers. Thus, only the orientation on the yaw-axis is used for the PID controller.

It should also be noted that the PID controller for the yaw, or orientation, should not be affected by the current position of the quadcopter in Euclidean space. This would only lead to the platform attempting to rotate 180° if the goal position is behind it. This is not required, since the quadcopter can move on all axes; thus, capable of full 6 DoF manoeuvrability.

The PID controllers were created using the notation in (3.1).

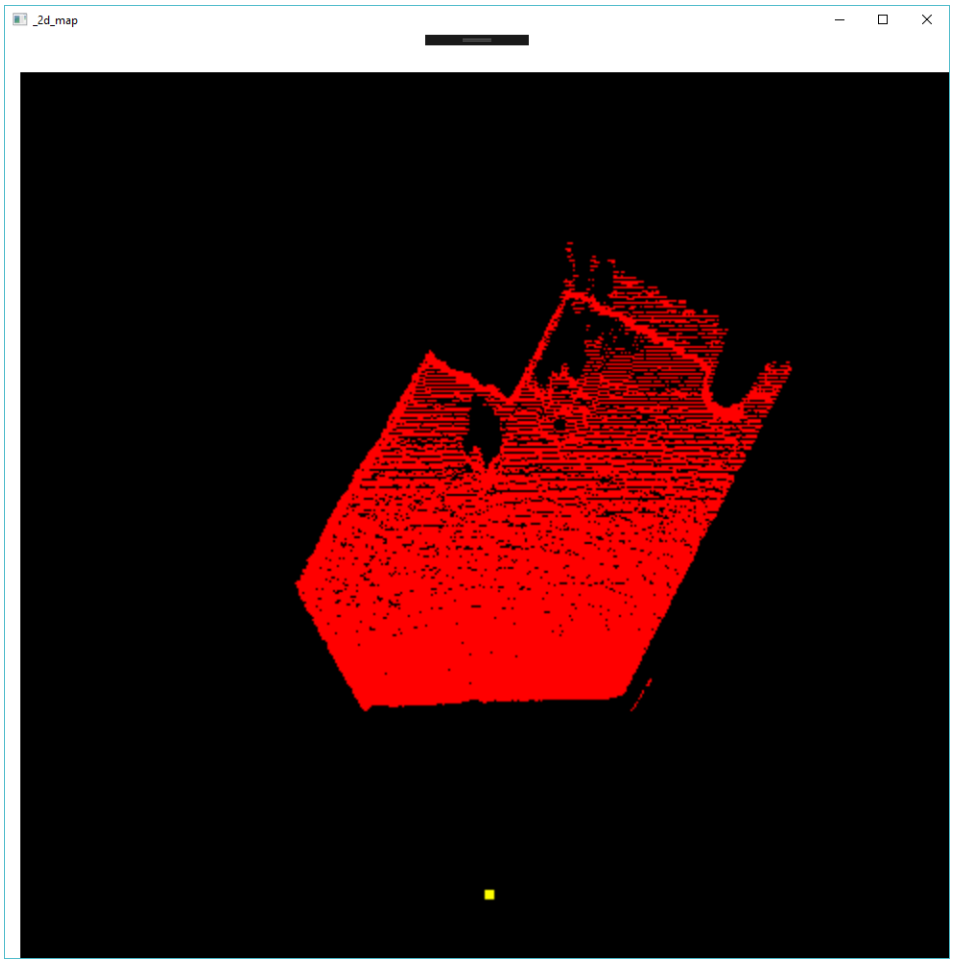


Figure 3.10. An example of how the 2D map is shown to the user. By clicking within the red designated area the quadcopter will attempt to move to that location.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{3.1}$$

The term e represents the error function, and K_p,K_i and K_d respectively donates the proportional, integral and derivative coefficients. The PID coefficients were manually calibrated for the platform, and are shown in Table 3.1. The error metric used, is the distance between the current point and the desired position on each specific axis. This distance is measured in mm, as the Kinect range data is also given in m.

Because the system is fairly unstable around a single point, the proportional term of the PID controllers are set relatively high, so that the main driving force will be the actual error between the current

Table 3.1. PID controller coefficients.

PID Controller	K_p	K_i	K_d
Roll PID	0.015	0	0.01
Pitch PID	0.015	0	0.01
Yaw PID	0.06	0	0.02
Throttle PID	0.025	0.013	0.02

position and the desired position. A maximum limit is set on the effect that the PID controller can have on the roll, pitch, yaw and throttle. This is to prevent the system from moving the quadcopter at an unreasonable pace. The integral term is set to a near zero level for the roll, pitch and yaw, but at a high level for the throttle. The large throttle value is required due to the effect of a depleting battery. The throttle level is very dependent on the battery voltage level, and thus it has to dynamically adjust for the battery drain. The derivative term for all four of the PID controllers is set near the same level as the proportional term, as the system should rather slow down a lot once it nears the goal, than overshoot and having to adjust against the momentum of the quadcopter.

3.3.4 Offline datastream

The Offline datastream and the Quad control applications are mutually exclusive, as they both serve the same function with a different goal. The Quad control application is intended for use with the quadcopter itself, where the Offline datasteam application is intended to utilise datasets already captured to evaluate the SLAM implementations performance, relative to other researchers. Figure 3.11 shows the software diagram for this application. This application has been created using C# WPF.

The application queries the user for the directory containing the datasets in question, and then the user can select the dataset that should be used. The datasets themselves and the techniques used to evaluate performance, will be discussed in Section 3.5. After the user has selected a dataset, the desired frame rate is chosen as well.

For each frame in the dataset, a corresponding timestamp is provided. To simulate a real system with the selected frame rate, the application uses the timestamp to sample the dataset at the correct

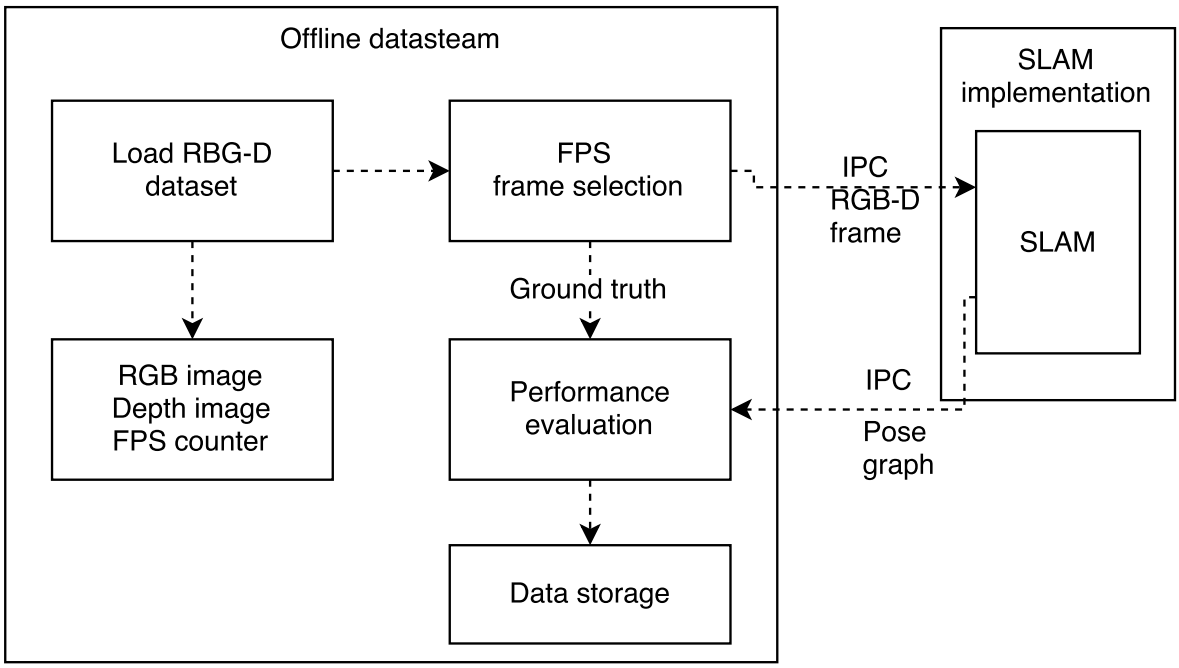


Figure 3.11. Offline datastream software diagram showing the internal processes and the connectivity to the SLAM implementation

framerate. This frame rate is also matched in the delay between each frame being sent to the SLAM implementation to simulate a real-world system. The application displays each RGB and depth image being sent to the SLAM implementation.

For each frame sent to the SLAM implementation, a range of data is returned. This includes the frame-to-frame egomotion that the front-end calculated, as well as the current pose-graph before and after graph optimisation. This data is saved so that it can be evaluated offline, using various metrics. A screen shot of the application is shown in Figure 3.12;

3.3.5 SLAM implementation

This subsection will only discuss the SLAM implementation with regards to its interaction with the other application. Section 3.4 discusses the SLAM implementation itself. The SLAM implementation application is created in a C++ environment, and uses 3rd party libraries OpenCV and CUDA. The software diagram is given in Figure 3.13.

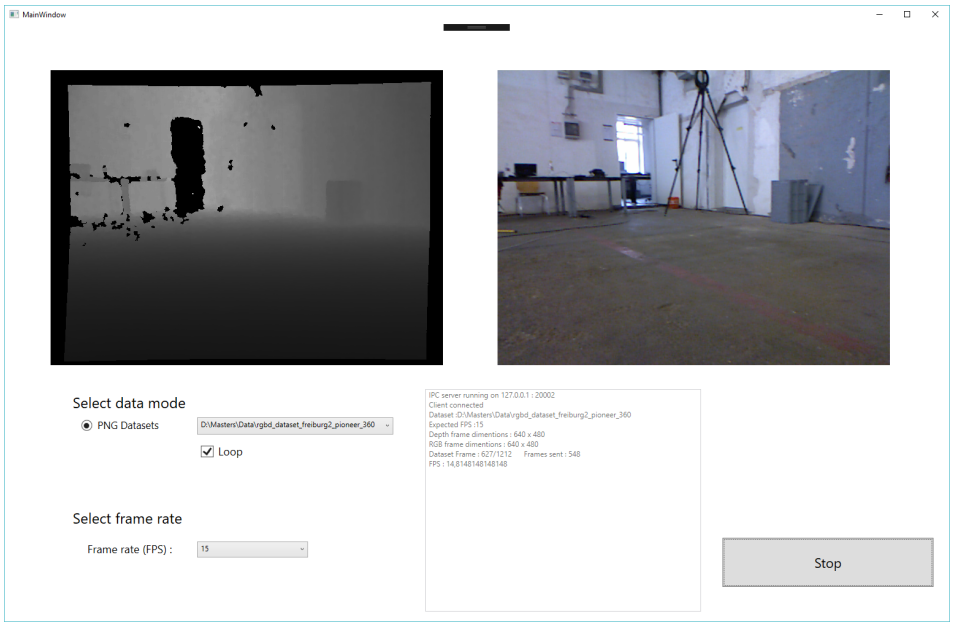


Figure 3.12. The offline datastream application where the dataset and frame rate can be selected. After a full dataset is completed the relevant raw data is saved for offline evaluation.

The application establishes an IPC connection with either the Quad control application or the Offline datastream application. Once this has been established, the application reads the RGB-D data for each frame sent. The frame is processed through the front-end of the SLAM implementation. The relevant frame-to-frame egomotion estimation is saved, and the relevant data is sent to the back-end of the application. The back-end checks for a loop-closure and corrects the drift, using graph optimisation. The resulting pose-graph with the corrected egomotion estimation is sent to the RGB-D application connected to the SLAM implementation.

Both the front-end and back-end application utilise their own thread. This is to allow the SLAM implementation to run in real-time. The front-end operates on each and every frame received, while the back-end only operates on the latest available data once each iteration has ended. Alternatively, the application can be set to always wait for the back-end to finish computing before accepting a new frame, and can be used with the Offline datastream as a real-world platform is not reliant on its time constraints.

The RGB-D data is converted into a point cloud and sent to the 3D visualiser application, using IPC. This process is performed in this application because the GPU can be used to create the point cloud from the RGB and depth data. A large amount (roughly 20%) of the point cloud generated will be

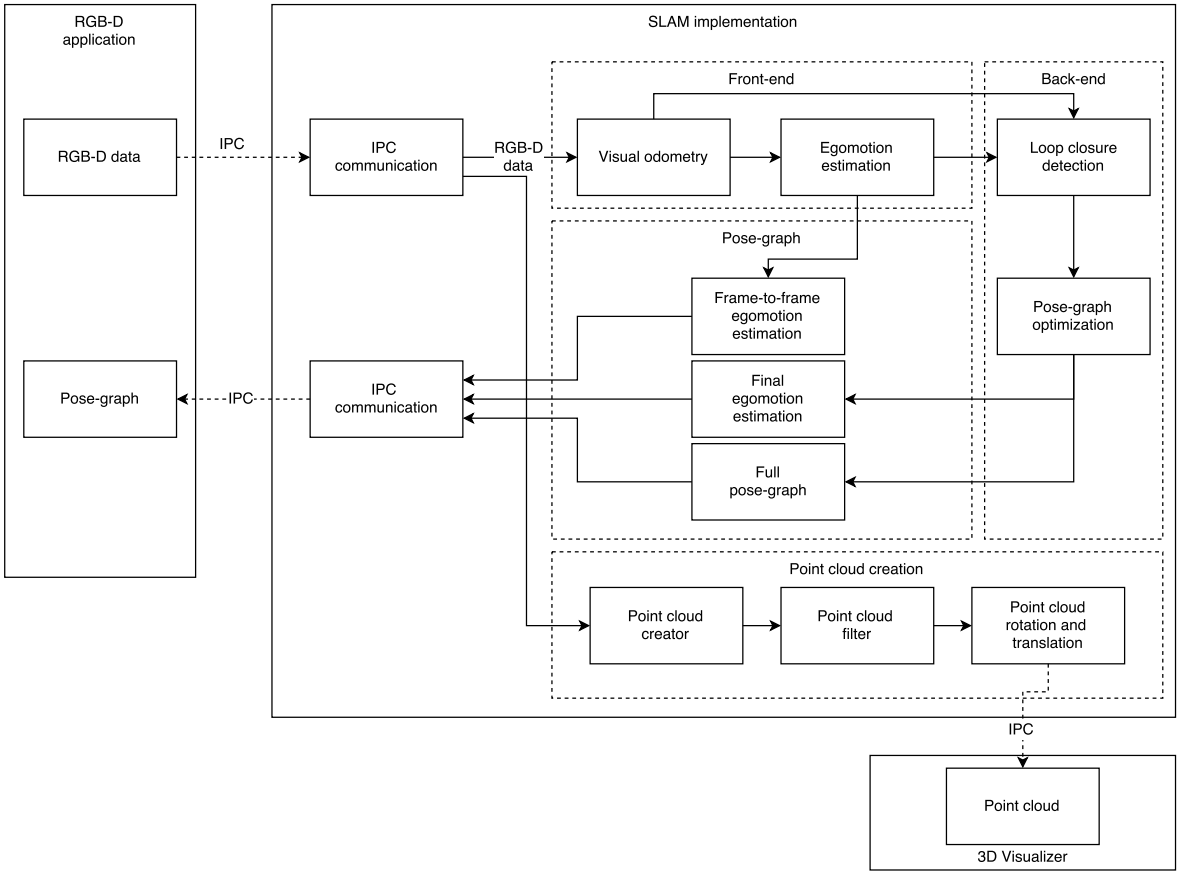


Figure 3.13. SLAM implementation software diagram showing all the internal processes

invalid points, because of the registration process between the RGB image and the depth image. The point cloud removes all the points without an accompanying depth value, and sends the point cloud for visualisation. Additionally, if the SLAM implementation is operating in conjunction with the Quad control application, the system extracts a sub-sampled set of points from the ground level. These points are combined, using the egomotion estimation to give a 2D top down view of the floor. This, in turn, is sent to the Quad control application to give the user information on the surroundings.

The ground level is determined on start-up. The system is designed for the quadcopter to be started on an elevated position with a clear view of the floor surface. The SLAM implementation has to find the correct rotation to align the floor with the quadcopter, since the Kinect is mounted at an angle and the landing gear of the quadcopter is not perfectly rigid.

3.3.5.1 OpenCV

Open Source Computer Vision Library (OpenCV) is an open source public library that offers a range of image processing techniques and data structures. A large number of data structures implemented in OpenCV is used throughout the application, and some image processing techniques from the library are also used.

Whenever this library is being used, it will be mentioned specifically, as some of the functionality available in OpenCV has been implemented from first principles to give the author better control over the processes implemented.

3.3.5.2 CUDA

CUDA is a public library created by NVIDIA and available for C++. CUDA allows developers to utilise the GPU of a computer for parallel computing. A computer GPU is capable of computing a large amount of threads in parallel, and is invaluable for speeding up image processing. For the SLAM implementation a large number of the techniques implemented utilises the GPU to speed up the system, allowing it to complete in a fraction of the time a standard CPU would have taken. The use of CUDA is discussed in detail in Section 3.4.

3.3.6 3D Visualiser

The 3D Visualizer application is used to display a point cloud in a 3D environment, where the user can interact with the point cloud. This application uses Point Cloud Library (PCL) to display the point clouds. The application is automatically launched from the SLAM implementation, and does not interact with the SLAM implementation itself. This application was split from the SLAM implementation due to the excessively long compilation time PCL requires, slowing down development.

Even though this application is appropriate for viewing small point clouds, it suffers greatly when the SLAM implementation operates on a large dataset. Thus, only the most recent point clouds are combined and displayed simultaneously.

3.4 SLAM IMPLEMENTATION

As discussed in Section 3.3.5, the SLAM implementation is designed to work as a black-box system, where RGB-D data is supplied and the estimated pose-graph is returned. This implementation does not evaluate performance or calculate the required roll, pitch or yaw of the quadcopter, but rather provides the information required for other applications to do so.

Figure 3.14 shows the full flow diagram of the SLAM implementation.

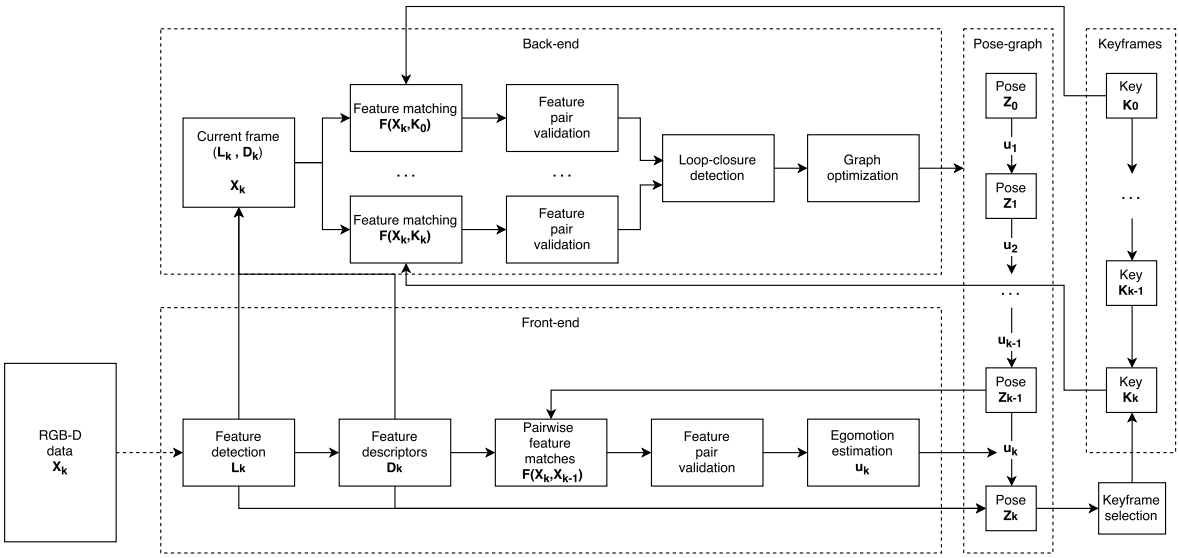


Figure 3.14. SLAM implementation flow diagram. The front end calculates the features and descriptors for each new image, while matching them to the previous image. The back-end utilizes the detected features and descriptors from the front-end to search for loop closures and updates the pose-graph accordingly.

3.4.1 Threads

The SLAM implementation is heavily threaded to improve performance and insure that the application is able to operate with a real-time system. The application relies heavy on the GPU to parallelise the image processing sections of the implementation.

3.4.1.1 Main process threads

For the front-end and back-end implementations two main CPU threads are used. The purpose of splitting these two processes into two individual, unsynchronised, threads was to allow the two to operate at different frame rates. This is because the back-end implementation tends to operate much slower than the front-end, and at times are unable to keep up with the full frame rate.

Thus, while the front-end always operates on each and every frame delivered, the back-end only operates on the latest frame each time the iteration has been completed. Consequently, if the front-end captures and processes two frames before the back-end has been completed, the back-end will not utilize one of the frames.

3.4.1.2 CUDA threading

For some of the image processing techniques implemented, CUDA is used to speed up the large number of repetitive tasks needed to be performed. The remote terminal (PC) used to compute the results for this dissertation, has a GeForce GTX 1060 graphics cards. This card uses the new Pascal architecture and is part of the first series of cards that utilise this architecture.

This particular card has 10 multiprocessor units, each with 128 CUDA cores. Therefore, a total of 1280 CUDA cores are available in parallel. A total of 6144 Mbytes of global memory is available as well. CUDA uses a process of grouping threads into thread blocks, with a maximum of 1024 threads per thread block. Two layers of memory are available for use on the GPU. The global memory can be accessed by any thread at any time, but it has a relatively slow access speed. The second type of memory present is the shared memory. Shared memory is a small memory block that is dedicated to each specific thread block. The access speed for the shared memory is several orders of magnitude faster than the global memory. Thus, a task that operates on the same memory block multiple times will benefit greatly by using the shared memory. The only problem is that each thread block only has 49152 bytes of shared memory allocated to it. It should also be noted that there is inter-thread communication between threads in the same thread block, but no communication is possible between threads in different thread blocks. This makes it impractical to run processes that depend on each other

over multiple thread blocks, as one thread block will have to wait for the other to finish computing. This process will waste valuable resources and time.

3.4.2 Feature detectors and descriptors

The first process applied to newly arrived RGB-D data is to extract the relevant features from the RGB image. This process is performed by using the feature detection methods discussed in Chapter 2. For this dissertation three main feature detectors and descriptors are implemented, namely SIFT, SURF and ORB. These feature detector and descriptor pairs are interchangeable; thus, the performance of each can directly be compared to the others. The feature location's schematics between each feature detector are similar, as this is represented by the 6D vector used to represent the feature (x,y,z and r,g,b values). The main difference between the descriptors is the size of the descriptor and the representation. SIFT and SURF are similar, but ORB uses a bit representation for the descriptors.

3.4.2.1 SIFT feature detection

SIFT feature detection has been implemented from first principles for this dissertation. This is the only feature detection method implemented from first principles because previous studies have proven that SIFT outperforms all the other methods with regards to accuracy.

The first step performed once a new RGB-D dataframe has been received, is to convert the RGB image to a grey scale image by calculating the intensity for each pixel. This is a simple process and does not take much time, as each pixel can be calculated in parallel with the rest, using CUDA. The next step is to create the Gaussian pyramid for the given RGB image. Thus, the RGB image is blurred over a range of σ values, with each of the resulting images grouped into an octave. The images are blurred by convolving the RGB image with the specified Gaussian function. The size of the Gaussian pyramid can be set by specifying the starting σ value, the number of images per octave and the number of octaves that need to be computed.

Once the Gaussian pyramid has been completed, the accompanying Difference of Gaussians (DoG) can be computed by subtracting the images generated from one another. The local extrema are found by comparing each pixel of interest with its neighbouring pixels, in both scale and space. Accordingly,

the pixel in question is compared to the pixels surrounding it in the same image, using a specified window size. This window is typically chosen as the 8 pixels directly neighbouring the current pixel, but can also be increased to the first two pixels in each direction. If the pixel is found to be an extrema, the pixel is compared to the corresponding pixels in the other DoG images in the octave. The point is chosen as a feature by saving the location and specific scale σ, where the point was maximised or minimised.

For each point of interest found, the location is refined by interpolating the neighbouring data, using a Taylor expansion. If it is found that a neighbouring pixel is a better candidate, the location of the feature is changed. Additionally, during this process any features having a low contrast are discarded without further consideration.

To eliminate edge responses that typically deliver weak descriptors, the ratio between the eigenvalues of the Hessian matrix is used. The Hessian matrix is computed at the location and scale where the feature has been found.

Additionally, to give an effective method to limit the number of features detected, each feature is scored in a ranking system, where a feature with a low local contrast difference is scored 1 and a very high contrast difference scores a 10. By discarding features that fall below a certain ranking, the number of features can be controlled to a certain degree.

3.4.2.2 SIFT feature descriptor

The SIFT feature descriptor is also implemented from first principles. Given the corresponding scale σ of each feature in question, the local gradient of the feature is computed over a window size dependent on the scale. An orientation histogram is created to represent all possible orientations that the feature can assume. This histogram is populated by the gradient vectors of the surrounding pixels. The dominant peak of the histogram is used to orient the grid used to compute the descriptor. It is possible for a single feature to have multiple dominant orientations, resulting in multiple descriptors with different orientations for a single feature.

For each dominant peak in the histogram, a descriptor is created by placing a 4×4 grid around the

feature location, oriented based on the histogram peak chosen. For each point in this grid (16 points) another 8-bin histogram is created to represent the local gradient direction of the pixels at that specific scale. Points closer to the feature are weighted to increase their respective effect. Thus, for each point in the 4×4 grid an 8-bin histogram is created, resulting in a 128 dimensional descriptor.

3.4.2.3 SIFT CUDA implementation

For both the detector and descriptor the SIFT implementation was parallelised to speed up computation. The image was subdivided into 320 blocks of 960 pixels each (two columns of pixels per block). Each block of pixels was given to a thread block that can have a maximum of 1024 threads. For each thread block the portion of the image that will be required is loaded into the shared memory of that thread block. Each thread block then computes the required number of Gaussian blurred images, depending on the parameters provided. Each portion of the relevant images are saved to a global memory block dedicated to each Gaussian image. Once this has been done, the required DoG can be computed, where each pixel is assigned its own thread. To detect the local maxima and minima, the images are once again subdivided into thread blocks, where each thread is responsible for a single pixel location, spread across the DoG images within each octave. Each thread compares its respective pixel with those surrounding it and across the other relevant DoG images. Once each thread block has completed finding the possible features, the Tylor and Hessian matrices are computed over a number of threads for each feature, optimising and filtering out any undesired features.

For the descriptor, each gradient is computed in its own thread, and each histogram has its own thread block. Thus, once again, the section of the scale image required to calculate the descriptor is loaded into the shared memory of the thread block, and then each gradient is calculated in its own thread. A maximum of two descriptors are allowed per feature, and are finally stored in the global memory of the GPU.

Because of the nature of the implementation, this approach will not be ideal, or will not even function correctly, for all types of graphics cards. This is because the size of the shared memory and the maximum threads in a thread block is not always the same across different GPUs.

3.4.2.4 SURF and SURF GPU

OpenCV was used to compute the SURF features and descriptors for any given image. The OpenCV implementation allows the user to choose the number of octaves to compute, the number of layers in each octave and also the descriptor length. A threshold can also be set to remove features below a certain contrast value, using a Hessian matrix. This process returns feature locations and descriptors very similar to SIFT, and very few alterations has to be done to interchange SURF with a SIFT implementation.

OpenCV also provides a CUDA implementation for SURF feature extraction. This process uses a native approach to CUDA, and is not tailored to any specific type of graphics card.

3.4.2.5 ORB feature detector and descriptor

Similar to SURF, OpenCV was used to add ORB features to this implementation. OpenCV, again, provides a number of parameters to customise the feature detection process. The user can set the maximum number of features, the pyramid scale factor, number of levels in the pyramid and the edge threshold. No GPU implementation for ORB is available, but since it is already much faster than SIFT and SURF, this did not not pose a major problem.

3.4.3 Pairwise feature matching

For the feature matching process a brute-force approach was used. For each new observation, all the features detected in the image are matched to all the features in the previous image. This approach is chosen because the purpose of this dissertation is to measure the effect that scene changes have on the accuracy of SLAM; thus, the level of variation between observations will change, based on the dataset. This will not allow the use of any method to exclude certain features, based on distance.

Each feature in the new observation is allocated a thread block on the GPU. Each thread is responsible for comparing the source descriptor to the target descriptor. The descriptors are compared by calculating the Euclidean distance between the descriptors in the high dimensional descriptor space. Any two feature descriptors that are more than half the distance between the source descriptor and the origin

from each other, are rejected outrightly. Each thread saves its Euclidean distance in the shared memory of the thread block, in ascending order.

The two best matches for each feature in the new observation are saved and used for the next process. By limiting each feature to a single thread block, limits the number of features that can be evaluated per cycle. If a single image has more than 1024 features, the remaining features are compared and evaluated after the first 1024 features have been completed.

3.4.4 Feature pair validation

To further eliminate false-positive matches, a method proposed by Lowe is utilised [55]. By not only evaluating the distance between the descriptor and its nearest neighbour, but also evaluating the distance between the nearest and second nearest neighbour, the number of falsely associated feature pairs can be reduced. This is because good feature pairs will ideally have a single other feature that is close to it in descriptor space, while the second closest will be much further away. Thus, the uniqueness of that particular feature is very high, and it is robust against false associations.

By only allowing features that have a single close neighbour, the number of feature pairs are decreased significantly. This will not only provide a better metric for matching two observations, but also reduce the strain that a large number of feature pairs will have on the following processes.

3.4.5 Pairwise transformation

The pairwise transformation is used to calculate the egomotion estimation. Given the feature pairs in the previous section, a small subset of these feature pairs are used to estimate the transformation the sensor underwent between two observations. Only a small subset of points are used because the processing time increases exponentially as more points are added to the problem. By only choosing the feature pairs that are perceived as being very accurate, i.e. ideally very close in descriptor space to only one other feature, the estimated transformation will be more accurate than a larger set of points where the average error is higher.

The pairwise transformation calculation is done in a three step process:

1. Take a small subset of paired features, calculate the 3D point location from the depth data and find the transformation that aligns them by using a leasts squares approach.
2. Estimate the accuracy of the transformation by calculating the number of inlier points, using RANSAC.
3. Repeat the process with a larger or different subset of points if the number of inliers are not sufficient.

3.4.5.1 Transformation matrix

To calculate the affine transformation the four feature pairs closest in descriptor space are chosen. For each of these feature pairs the depth value must be known. Each depth value is converted to its respective 3D point, and given these four pairs of 3D points, the method proposed by S. Umeyama [70] is used to estimate the transformation between the two sets of points.

This method uses a least squares approach to minimise the sum of distances between each pair. This method returns a translation matrix of size 1×3 and a rotation matrix of size 3×3.

3.4.5.2 RANSAC

RANSAC is implemented to estimate the accuracy of the affine transformation. For all the feature pairs detected between the two observations, the corresponding 3D points are calculated. Each of the 3D points in the target observation is transformed, using the calculated transformation matrix. Owing to scene coherence, this should align the full set of feature pairs with one another. This processes will typically place falsely associated feature pairs far from one another in 3D space. RANSAC is used to compute the number of inliers, or rather the number of feature pairs that fall within a short distance from one another after the transformation.

The number of inliers are compared to the number of feature pairs and provide the only metric to how well the two sets of points match each other. It should be noted that a small number of inliers do not prove that it is a bad transformation. If there is very little scene coherence then there should be a small number of inliers, but the SLAM implementation has no way of knowing that.

3.4.5.3 Repetition

If the number of inliers are low, the process can be repeated using a different subset of points. This might provide a different transformation that can, once again, be evaluated. If, however, the number of inliers do not improve over a few iterations, the transformation that yielded the most inliers is chosen and a small scene coherence is assumed.

3.4.6 Loop-closure detection

To detect and assess possible loop-closures, the system compares the current observation to a subset of poses called the keyframes. This system uses the same process as the front-end to compare features between two images, and then uses RANSAC to compute the number of inliers detected to determine if a loop-closure has occurred.

On implementation the system displayed a tendency to match the current observation to the last few keyframes added to the keyframe sequence. This is because the sensor is still in the same relative scene as the last keyframe, and due to the keyframe selection process the moment the current observation is not matched properly to the last keyframe, a new keyframe is introduced. Through this process the system will always either match the current observation to the previous keyframe, or classify the current observation as a new keyframe. Although this system is capable of reducing the drift introduced over a small subset of poses, it will not allow the system to identify full loop-closures very well. Thus, the system does not classify the current observation as a full loop-closure if it is only matched to the latest few keyframes.

To solve this problem the system ignores the fact that the current observation did or did not match the last keyframe in the sequence, but continues to attempt to match the current observation to the other keyframes present on the system. Through this process the system uses the latest keyframe to marginally reduce the drift between the current observation and the last keyframe, and then allows for a full system loop-closure as well.

3.4.6.1 Keyframe selection

Two different keyframe selection processes were implemented. As common with most older implementations, the first is to choose a keyframe after a set amount of frames are captured. This process works well for a system where the frame rate is constant over all iterations, but for this dissertation this was modified so that a set number of keyframes are generated every second. Thus, a set of three keyframes are chosen each second, independent of the frame rate. These frames are merely selected based on the time that has passed since the previous keyframe, and are not determined based on the number of features. This process produces an excessive amount of keyframes, and is not well-suited for a practical implementation.

The second approach implemented is to add a new keyframe based on the number of inliers the RANSAC process found between the current observation and a small set of previous keyframes. The current observation is matched to the three most recent keyframes introduced to the system. Only the last three keyframes are used because this process is only a small portion of the full back-end, and the system needs to maintain a relative fast iteration speed. Thus, if RANSAC estimates that there is a large enough scene change compared to the last three keyframes detected, the current observation is classified as a new keyframe and added to the sequence. This process produces keyframes that are roughly based on the level of scene coherence, not some arbitrary observation number. This approach was used to capture all the results presented in the next chapter.

To classify the current observation as a keyframe or not, the system first compares the number of feature pairs found between the current observation and each of the three latest keyframes. If the number of feature pairs between two observations are below 15% of the total number of features in the current observation, the system automatically assumes a bad coherence and a new keyframe is created. If the number of feature pairs exceed this margin, RANSAC must additionally return at least 45% of the feature pairs as inliers for the system to assume a scene coherence that is acceptable.

3.4.6.2 Local loop-closure

Matching the current observation to the last three keyframes in the sequence is named as the local loop closure (LLC). This process attempts to optimise the current observation by assuming that a small

level of scene variance has occurred in the last few observations. This is particularly effective for scale changes, where the sensor moves perpendicular to the sensor (forwards and backwards). Thus, the system estimates the level of scene variance experienced and determines if the current observation should be optimised using one of the previous three keyframes. If the current observation does not qualify as a new keyframe, the observation egomotion estimate is optimised, using the keyframe that return the largest number of inliers.

Although this process can improve on the overall accuracy for each single observation, it can have a detrimental effect if the error introduced by matching the observation with a keyframe is larger than the potential drift experienced since that keyframe has been taken. To counteract this the effect that the local-loop closure can have on the current observations pose is limited to one quarter of the edge associated with that pose. Therefore, the transformation estimation between the current frame and the previous frame is used to temper the potential detrimental effect that the last keyframe can have on the edge. This approach makes the assumption that the level of drift experienced between each frame is no more than 25% of the egomotion experienced. Even though this assumption might not be true for all observations, this limits the effect that a single false match between observation and keyframe can have on the system.

3.4.6.3 Global loop-closure

Once the system has refined the current observation using the local loop-closure, the same process is used to check for a potential global loop-closure. Even though most of the application is created to operate on a GPU, this process can take up a fair amount of time. This occurs because the process runs in parallel with the front-end, and both utilise the same GPU for processing. The system is designed in such a way that the front-end will always have priority over the GPU threads that are available. Thus, if a sequence of tasks are lined up for GPU processing, the front-end related tasks will always be prioritised over the back-end. Given this prioritisation and the fact that the number of keyframes can become excessively large, it is not feasible to match the current observation with all the keyframes.

Thus, a process of systematic comparison is used. For each iteration of the global loop-closure procedure, only seven keyframes are used from the full sequence (excluding the latest three). These

seven keyframes are sequentially chosen so that an even distribution is taken with regards to time. Hence the full range of keyframes are subdivided into seven equally spaced blocks, and one keyframe is sequentially selected from each block for each iteration.

Once the seven keyframes have been selected, the current observation is matched to each of those. Similar to the local loop-closure process, the number of feature pairs identified between the two images are used as the first metric to evaluate a potential match. At least 20% of the features found in the current observation must be matched to the target keyframe. Once this margin has been passed, the number of RANSAC inliers must fall above 60% for the system to identify a match. A much higher number of inliers are required to qualify for a global loop-closure because the overall effect that this process can have on the system is tremendous. If a false-positive global loop-closure is made, the pose-graph can potentially be irreversibly altered, resulting in the SLAM implementation failing in localising the sensor correctly.

If the current observation is matched to one of the keyframes presented, the system attempts to refine the keyframe used by also evaluating the two keyframes directly neighbouring the one of interest. Between these three keyframes the one with the largest number of inliers detected is selected, and the global loop-closure process is accepted. The keyframe that was found to be the best match, is placed at the end of the keyframe sequence; thus, this keyframe will be one of the three used in the local loop-closure for the following observations.

3.4.7 Graph optimization

For the full pose-graph optimisation only the global loop-closure is considered. This is because the local loop-closure will not affect a large number of observations, and will only make a marginal difference. The main impact of pose-graph optimisation occurs for full loop-closures on a global scale. Once a successful global loop-closure has been found, the transformation between the keyframe and the current observation is estimated. The drift is then computed by calculating the difference between the pose dictated by the front-end and the pose calculated by the loop-closure. This constitutes the estimated drift accumulated between the keyframe and the current pose. To allow for a globally consistent map, this drift needs to be minimised over all the poses inside the loop-closure.

For this implementation the **g^{2}o** framework is used. This is an open-source implementation provided for free, and is implemented in C++ [54]. This framework can be incorporated directly into the SLAM implementation, and offers three potential methods for solving the graph optimisation problem. The **g^{2}o** implementation specifies that the error function in (3.2) be used

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j\rangle \in C} e(x_i, x_j, z_{ij})^T \Omega_{ij.} e(x_i, x_j, z_{ij}) \tag{3.2}$$

The error over the trajectory can then be minimised to give a globally consistent trajectory according to

$$\mathbf{X}^\star = argmin_{\mathbf{X}} \mathbf{F}(\mathbf{x}). \tag{3.3}$$

The vector $\mathbf{x} = (x_1^T, \ddots, x_n^T)$ represents all the relevant poses included in the loop-closure. Ω_{ij} represents the information matrix between the poses $\langle i, j\rangle$. This is the inverse of the covariance matrix between the two poses, and can be calculated for each edge in the pose-graph at any time. The term z_{ij} is the mean of constraints between the two poses, or more specifically the egomotion estimation, the front-end calculated. The error function $e(x_i, x_j, z_{ij})$ is the vector error function between each of the poses, and determines how well the two poses compare to their ideal location.

The **g^{2}o** framework provides three possible linear solver techniques that can be used. CHOLMOD, CSparse and PCG can be interchanged at any time, but each provides a different level of accuracy and speed. PCG was used for this implementation, as both the CHOLMOD and CSparce methods can take excessively long to complete for some pose-graphs. PCG's completion time is dependent on the size of the error that has to be minimised. Thus the smaller the drift between optimisation points, the faster the optimisation will be. The trade-off is that PCG underperformed compared to both CHOLMOD and CSparce, but the difference is minimal for this implementation.

3.5 RGB-D SLAM EVALUATION

For this study to be scientifically accurate, some metric must be used to evaluate the performance of the SLAM implementation. This would suggest a system that is able to compare the estimated position of the platform to its real-world position. Thus, the system needs the ability to track the ground-truth position of the platform. This is, however, a very daunting task and will add an excessive amount of work to the dissertation, as a controlled and fully-observed environment must be created to capture empirical evidence of the platform location. Consequently, for this dissertation it will not be possible to track the ground truth of the quadcopter itself, but an alternative will be used to evaluate the SLAM implementation.

To accurately evaluate the performance of the SLAM implemenation, a publicly available benchmark will be used. The TUM RGB-D benchmark [71] was created in 2012 by Sturm *et al.* and provides an accurate way of evaluating the performance of RGB-D SLAM systems. This benchmark provides a large set of RGB-D datasets that are publicly available and can be used to simulate a real-world SLAM scenario. A Microsoft Kinect was used to capture a range of datasets at a full 30 fps, and the RGB and depth images are provided with the accompanying timestamps for each. Along with this RGB-D data a ground-truth measurement of the sensor is also given. These ground-truth readings were captured using a motion-capture system that accurately tracks the pose of the sensor at 100 Hz. These ground-truth measurements are estimated to be within submillimeter accuracy, more than enough for the purposes of this dissertation. The datasets are captured over a range of environments, from a cluttered office space to a sparse industrial area. The ground-truth readings are given in a Euclidean coordinate space for the translational position and a unit quaternion for the orientation. From this data an error metric can be devised that represents the disparity between the true sensor pose and the estimated sensor pose.

Each dataset is unique with regards to a few criteria. These criteria will ultimately determine the accuracy that can be achieved for each individual dataset. These criteria are:

- Environment type - The actual quality of all the data frames will be the same over all the datasets, but the environment captured will differ greatly. Scenes that offer very few opportunities to extract unique visual features will prove difficult to localise in, and scenes with overly large amounts of visual features can slow down the SLAM implementation.

- Translation velocity - The translation velocity will dictate the size of the scene change present between two successive observations. The amount of scale variation experienced between observations will be linked directly to the translation velocity. While still relevant, the translation velocity will have a smaller impact on the level of scene variation than the rotational velocity.
- Rotational velocity - The rotational velocity of the sensor will greatly impact the level of scene variation experienced between successive observations. This metric is the main focus of this dissertation, and will be evaluated in full to estimate the impact the rotational velocity has on a SLAM implementation.
- Loop-closure - The presence of a potential loop-closure in the dataset will directly affect the final accuracy of the system. Some of the datasets include a loop-closure after a lengthy duration, while other do not include any loop-closures.

3.5.1 Sensor pose representation

Before defining the possible error metrics that can be used to estimate the accuracy of the SLAM implementation, the notation used to donate the pose of the system at each time interval must be discussed. The transition between two poses in the pose-graph is donated by a translation and a rotation. Both the translation and rotation must be given in a 3D plane to accurately represent the pose of the sensor. Because the data captured by the sensor is a structured representation (i.e. the position of each pixel with regards to all the other pixels are defined) the data obtained is seen as a rigid body. Thus, the relation between each pixel is known and should not be altered in any way.

The most common approach to depict a pose is by giving the displacement between the current observation and a reference frame or pose. The reference pose is typically chosen as the first observation in the sequence. Thus, the first observation will be seen as the center of the 3D space that all other observations will be referenced to. Accordingly, given a reference pose $\mathbf{X_0}$, the pose of a specific observation $\mathbf{X_t}$ will be given as the rigid body transition between $\mathbf{X_0}$ and $\mathbf{X_t}$. This is given by calculating the homogeneous transformation matrix between $\mathbf{X_0}$ and $\mathbf{X_t}$.

3.5.1.1 Homogeneous transformation matrix

The homogeneous transformation matrix $\mathbf{T_{4\times4}}$ is given in (3.4)

$$\mathbf{T}_{4\times4} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \mathbf{0}_{1\times3} & \mathbf{1} \end{bmatrix}. \tag{3.4}$$

The matrix $\mathbf{R_{3\times3}}$ is the rotation matrix that represents the orientation change that occurred between the reference pose and the current pose. The matrix $\mathbf{t_{3\times1}}$ is the translation experienced between poses in the x, y and z plane. Thus, the transformation is classed as a type of rigid body kinematic in the special Euclidean group $SE(3)$. The transformation matrix $\mathbf{T_{4\times4}}$ uses a homogeneous coordinates system SE(3) and not the standard Euclidean space $\mathcal{R}^3$. This makes the process of transforming a point using the transformation matrix much easier than it would be using a normal rotation and translation vector.

3.5.2 Error metrics

Two error metrics are proposed by Sturm *et al.* [71] that can be used to evaluate the performance of any SLAM implementation using their data. Given a sequence of poses from the pose-graph that forms the estimated trajectory $\mathbf{P_1}, ..., \mathbf{P_n} \in SE(3)$ and the ground-truth measurements $\mathbf{Q_1}, ..., \mathbf{Q_n} \in SE(3)$, and that al the given poses are matched across the two sequences, the relative pose error (RPE) and the absolute trajectory error (ATE) can be computed.

For both the proposed error metrics an open source and online evaluation tool are provided that will calculate the error automatically, given a sequence of poses. This tool was used to calculate the ATE error to compare the global consistency of this dissertation's SLAM implementation with other similar implementations.

3.5.2.1 Relative pose error (RPE)

The RPE error is intended to measure the local accuracy of the SLAM implementation over a fixed time interval Δ. By altering the time interval Δ, the RPE error can be used to represent the frame-to-frame error calculated by the front-end and back-end. This error metric can also be used to calculate the average error over a set of poses. The relative pose error at time interval i is given in (3.5)

$$\mathbf{E_i} = (\mathbf{Q_i^{-1}}\mathbf{Q}_{i+\Delta})^{-1}(\mathbf{P_i}^{-1}\mathbf{P}_{i+\Delta}). \tag{3.5}$$

For any number of poses m the error $\mathbf{E_{1:m}}$ can be computed. The root mean squared error (RMSE) for both the translation and rotation can be computed from this sequence of errors.

$$RMSE_T(\mathbf{E}_{1:m}, \Delta) = \left(\frac{1}{m} \sum_{i=1}^{m} ||trans(\mathbf{E_i})||^2 \right)^{1/2} \tag{3.6}$$

$$RMSE_R(\mathbf{E}_{1:m}, \Delta) = \left(\frac{1}{m} \sum_{i=1}^{m} ||rot(\mathbf{E_i})||^2 \right)^{1/2} \tag{3.7}$$

Equation (3.6) computes the RMSE for the translation error and (3.7) calculates the RMSE for the rotation error. The window's size is set as $\Delta = 1$ so that a frame-to-frame reference is used. This method should not be used with a larger window sizes as this has been shown to penalise the rotational errors made earlier on in the sequence compared to those at the end.

3.5.2.2 Absolute trajectory error (ATE)

Since the RPE calculates the error between successive frames, it only gives an indication of the average drift between two observations. The RPE error metric will not give a fair indication of how accurate the overall mapping of the environment was. The ATE error metric estimates the global consistency of the system as a whole. This will give a clearer indication of how accurately the final mapping of the environment was compared to the RPE error.

The ATE error is computed after the two trajectories' paths have been aligned with each other using a least squares approach. After the sequence $\mathbf{P}_{1:m}$ and $\mathbf{Q}_{1:m}$ have been aligned with each other, the absolute trajectory error can be computed from (3.8). This process is similar to the RPE error, but encompasses the entire sequence

$$\mathbf{F_i} = \mathbf{Q_i^{-1}}\mathbf{S}\mathbf{P_i}, \tag{3.8}$$

where **S** is the rigid-body transformation calculated during the alignment.

Once again the RMSE of the sequence is calculated to find the average translation and rotation error.

$$RMSE_T(\mathbf{F}_{1:m}) = \left(\frac{1}{m} \sum_{i=1}^{m} ||trans(\mathbf{E_i})||^2 \right)^{1/2} \tag{3.9}$$

$$RMSE_R(\mathbf{F}_{1:m}) = \left(\frac{1}{m} \sum_{i=1}^{m} ||rot(\mathbf{E_i})||^2 \right)^{1/2} \tag{3.10}$$

It is noted by Sturm *et al.* that the rotational errors present in the system will lead to a larger translation error in successive frames; thus, the RPE error metric combines both the translation and rotation error into (3.6) for window sizes larger than 1. Because the ATE error aligns the two sequences with each other before calculating the error, both (3.10) and (3.9) only contain the error for their respective parameters. Typically, this will present in the form where the RPE error is slightly larger than the ATE error.

3.5.3 Dataset manipulation

To increase the level of data available, the nature of the datasets themselves can be used to generate additional downsampled datasets that will provide a much better platform to compare results to.

3.5.3.1 Frame rate manipulation

Because timestamps are given with the RGB-D data and the ground-truth measurements, it is possible to synthetically reduce the frame rate of the datastream utilising the datasets. The effect of this will be that the transformation between successive frames will be increased, effectively creating a new dataset of the same environment with higher average rotation and translation velocity.

Thus, for any given dataset, it can be downsampled multiple times, resulting in a much larger data pool available. This could also help with the cross validation of data. Comparing two datasets that were

Table 3.2. Downsampling ratio with the accompanying frame rate

	Ratio							
	$r = \frac{1}{1}$	$r = \frac{1}{2}$	$r = \frac{1}{3}$	$r = \frac{1}{4}$	$r = \frac{1}{5}$	$r = \frac{1}{6}$	$r = \frac{1}{7}$	$r = \frac{1}{8}$
Effective frame rate	30	15	10	7.5	6	5	4.29	3.75

taken of different scenes is valid, but considering the inherit differences between the two it will not be a perfect comparison. By downsampling the same dataset multiple times each of those can be directly compared to each other, without the effect of different environments coming into play.

Although this approach will give a different dataset for each frame rate, in effect it will be dropping frames at a given rate. Thus, for a frame rate of 29 Hz, 1 out of every 30 frames will be dropped. This does not increase the transformation between all 29 frames, but rather the transformation stays the same across 28 of them and one single frame will have almost double the normal transformation. To compensate for this the datasets are downsampled at a rate in line with the base frame rate. Thus, for each frame-to-frame transformation in the dataset to have an increased transformation, every second, third, etc. frame is sampled. The effective frame rate for this process is shown in Table 3.2.

The datasets can be downsampled even more, as long as the number of frames being discarded between each observation in the new dataset stay the same. For each level of r it should effectively increase the average rotation and translation of the dataset by the same factor. There are, however, limitations to this process. The number of times a dataset can be downsampled will depend on the initial rotation and translation velocities, as increasing the effective angular velocity beyond a certain point will make RGB-D SLAM impossible.

Additionally, another factor that will not be included in the downsampled datasets will be the effect of motion blur. The level of blur accompanying each observation will depend on the spatial velocity of the sensor during that instance, and thus the dataset will retain the same level of blur during downsampling.

3.5.3.2 Segmenting the axis of motion

Additionally, the datasets can be evaluated with regards to the main axis of motion for each observation. This would suggest that the motion, be it rotation or translation, is evaluated over each of the three respective axes individually. If it is found that a particular frame-to-frame motion is bound predominantly to a single axis, the relative error between the prediction and the ground truth can be evaluated as resulting from movement on that axis. Thus, for translation the effect of movement on the x,y and z axis can be evaluated. Similarly the effect of rotation around the roll, pitch and yaw axis can be evaluated individually.

This is accomplished by evaluating each and every successive frame in all the datasets. If it is found that more than 85% of the effective motion (separating translation and rotation) resulted from a single axis, the resulting egomotion and error are saved into a histogram for each axis of motion. It was found that the majority of the frames selected for this process came from the four benchmark datasets, fr1/xyz-rpy, and fr2/xyz-rpy. These datasets are discussed in the next section.

By combining this with the process of downsampling described in the previous section, a large number of frames for each combination of rotation and translation can be found. Using these samples, a direct evaluation can be done over a range of rotation and translation combinations. For instance, all translations below 3 mm are accepted, a total of 13 682 frames can be used to compute the effect of roll, pitch and yaw. This will provide more than enough samples to produce statistically valid results.

3.5.4 Datasets

The TUM RGB-D benchmark database contains over 50 RGB-D datasets that can be used for a range of purposes, from RGB-D SLAM to 3D object recognition and dynamic object tracking. In this work eight of these datasets will be used for a range of discrete evaluations. Of these eight datasets only four will be discussed in detail, as they represent the key features that will affect the performance of the SLAM implementation. These eight datasets are split between two main environmental types. One is a feature-rich office environment, with emphasis being put on the desks and the objects placed on the desks. The second environment is an industrial hall that presents a scene that is not as feature rich, but has a larger spatial element to it. Thus, the datasets are a combination of feature-rich, short range

environments and some large, monotone environments. Some of the datasets focus around a certain point of interest; thus, the sensor is moved around the point while always orientated towards the point. Other datasets were captured with a robot SLAM implementation in mind, where a very large loop is made before returning to the origin.

Table 3.3 gives the technical specifications for each relevant dataset. Only four of the datasets will be discussed in detail, as they will be the main focus during the results and discussion. Note that the values given in Table 3.3 do not perfectly depict the values as they will be for the SLAM implementation, as there is no perfect correlation between RGB, depth and ground truth readings.

Table 3.3. Selected datasets from TUM RGB-D SLAM database

Sequence name	**Duration** [s]	**Total trans** [m]	**Avg tans vel** [m/s]	**Avg rot vel** [deg/s]	**Number of frames**
fr1/xyz	30.09	7.112	0.244	8.920	798
fr1/rpy	27.67	1.664	0.062	50.147	723
fr2/xyz	122.74	7.029	0.058	1.716	3669
fr2/rpy	109.97	1.506	0.014	5.774	3290
fr1/desk	23.4	9.263	0.413	23.327	613
fr1/room	48.90	15.989	0.334	29.882	1362
fr2/pioneer_360	72.75	16.118	0.225	12.053	1225
fr2/pioneer_slam	155.72	40.380	0.261	13.379	2921

From the eight datasets presented, only the following four will be referenced within the discussion of the results. This is because among these four datasets the majority of parameters are evaluated, and adding the additional four will not add substantive material for discussion. The additional datasets are, however, used to compare to other SLAM implementations; thus, they have been added to the sequences tested. For each of the datasets evaluated below a histogram is given of the translation and rotation experienced during each. A top down view is also given of the trajectory the sensor followed.

3.5.4.1 Dataset : fr2/xyz

Duration	:	112.74 s
Total translation distance	:	7.029 m
Average translation velocity	:	0.058 m/s
Average rotation velocity	:	1.716 deg/s
Number of frames	:	3669 frames
Environment type	:	Cluttered office
Motion blur	:	None
Large loop-closure	:	None

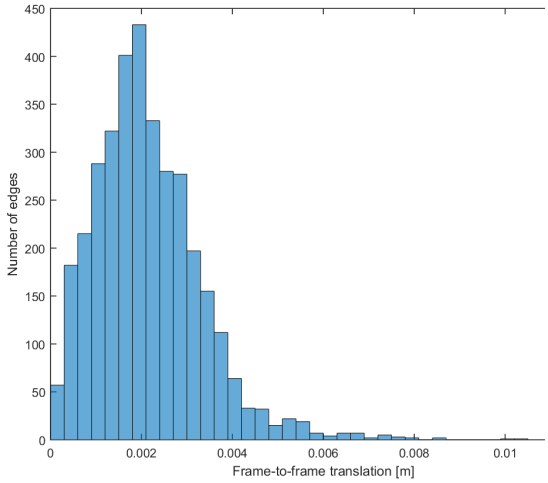
The fr2/xyz dataset is intended as a benchmark and debugging dataset for RGB-D SLAM processes. This dataset is very similar in nature to the fr1/xyz dataset, except that this one's original frame-to-frame motion is a lot slower and deliberate. This reduces the level of motion blur that the images might have. This sequence is also more than four times longer, giving a much larger dataset that can be used for evaluation.

The fr2/xyz dataset is captured in a typical office environment with the main focus being a cluttered office desk. For this dataset care was given to only allow translation between frames, and as little rotation as possible was added. This can be seen in the low average rotation speed present over the dataset. Through this dataset the orientation differences between observations will be minimised; thus, the error obtained will reflect the translation property of sensor transformation to a greater extent. Because the dataset is very large and the relative transformation between two successive observations is low, the frame rate of this dataset can be reduced drastically to provide a datastream with a wide range of different translation velocities. A large level of downsampling is shown in Table 3.4.

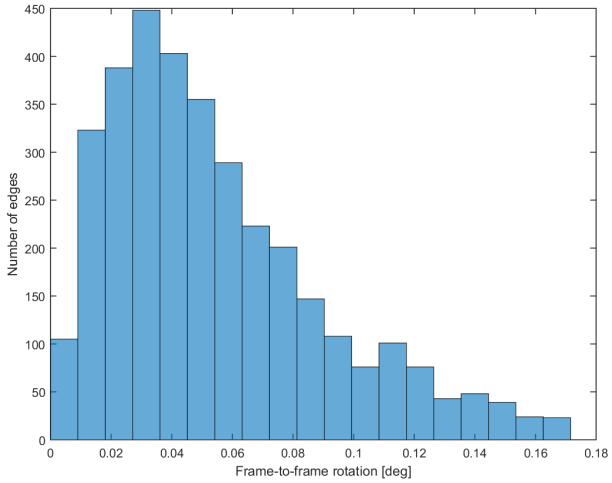
Table 3.4. The relevant parameters for all the various levels of downsampling on the fr2-xyz dataset

Frame rate	Duration s	Frames	Trans. Vel. m/s	Rot. Vel. deg/s
30.00 fps	122.17	3,479	0.06	1.69
15.00 fps	61.08	1,833	0.11	3.42
10.00 fps	40.72	1,222	0.17	5.09
7.50 fps	30.54	917	0.23	6.97
6.00 fps	24.43	733	0.29	8.56
5.00 fps	20.36	611	0.34	10.2
4.29 fps	17.45	524	0.4	11.95
3.75 fps	15.27	458	0.47	13.99
3.33 fps	13.57	361	0.53	15.42
3.00 fps	12.22	247	0.58	17.34
2.73 fps	11.11	187	0.64	19.06
2.50 fps	10.18	154	0.7	20.21
2.31 fps	9.4	144	0.76	22.28
2.14 fps	8.73	132	0.8	23.81
2.00 fps	8.14	122	0.88	25.48
1.88 fps	7.64	116	0.91	27.53
1.76 fps	7.19	109	1.01	29.18
1.67 fps	6.79	103	1.05	30.52

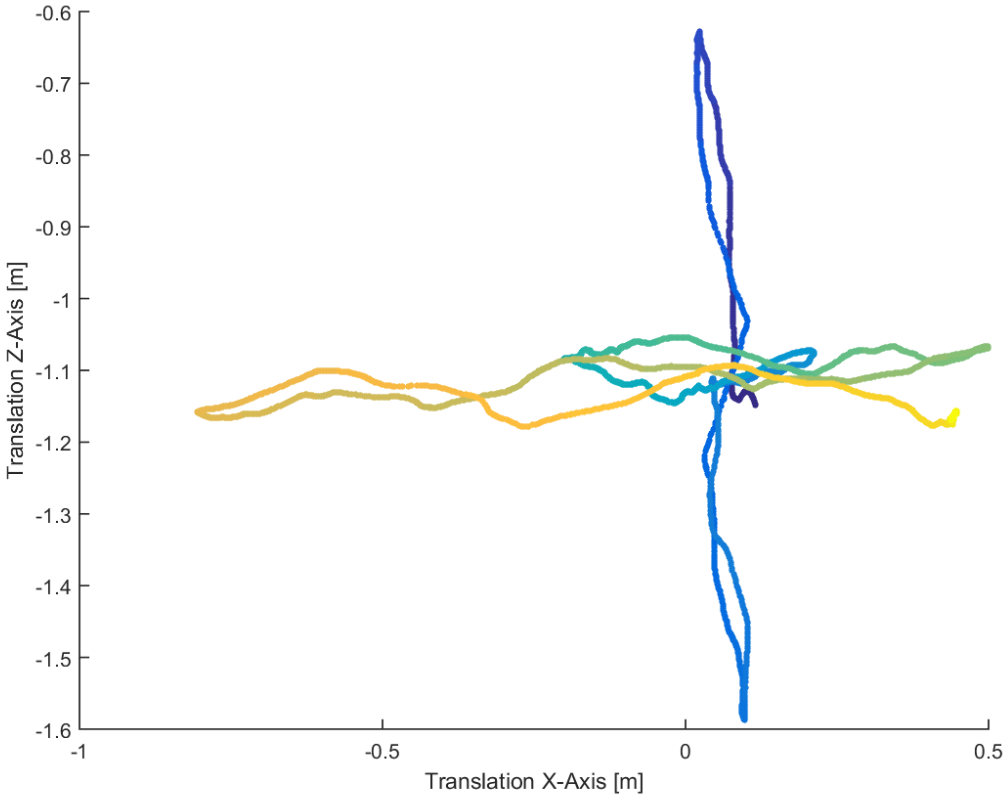
For this dataset the three principle axes of translation are also segmented. Throughout the dataset the sensor motion is primarily on one of the axes, with a very slight movement on the other two axes. Thus, this dataset will provide the majority of the data used to compare the difference between the three axes of translational movement. This dataset typically represents an ideal system with an abundance of data, where the amount of data far outweighs the level of scene transformation, and is ideal for the purposes of this dissertation. Figure 3.15 shows the translation and rotation histograms of this dataset, as well as the top down trajectory that the sensor followed.



(a) Translation histogram.



(b) Rotation histogram.



(c) Global trajectory.

Figure 3.15. This figure shows a a histogram of the translation (a) and rotation (b) experienced between two successive frames for the fr2-xyz dataset. Image (c) shows the top down trajectory of the sensor during operation. The trajectorey is colored based on the starting location (blue) and end (yellow

3.5.4.2 Dataset : fr2/rpy

Duration	:	109.97 seconds
Total translation distance	:	1.506 meters
Average translation velocity	:	0.014 m/s
Average rotation velocity	:	5.774 deg/s
Number of frames	:	3290 frames
Environment type	:	Cluttered office
Motion blur	:	None
Large loop-closure	:	None

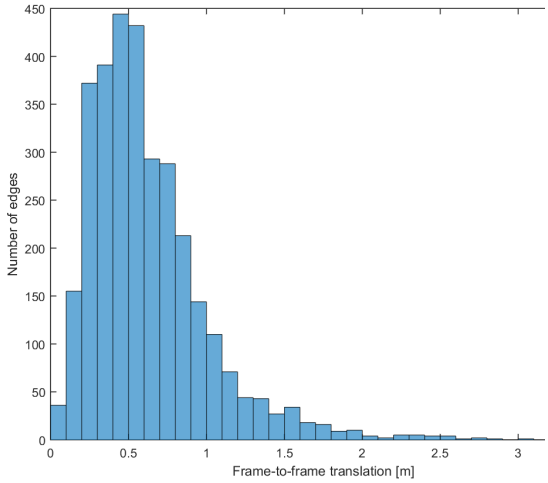
The fr2/rpy is the rotational version of the fr2/xyz dataset. This dataset is also intended as a benchmark and debugging dataset for RGB-D SLAM processes. Where the fr2/xyz dataset minimised the rotation during the dataset, this dataset minimises the translation, while showing large degrees of movement for the sensor orientation. Compared to the fr1/rpy dataset, this one is very similar, except that the average level of rotation is lower, resulting in a much clearer image. The images in dataset fr1/rpy are heavily beset by motion blur, and will thus affect the performance of the visual odometry system.

The fr2/rpy dataset is captured in a typical office environment with the main focus being a cluttered office desk. Care was taken to isolate the rotation of the sensor to a single rotational axis at any given moment, and the average velocity is low enough to prevent motion blur and rolling stutter effects. Because the dataset is very large and the relative transformation between two successive observations is very low, the frame rate of this dataset can be reduced drastically to provide a datastream with a wide range of different rotational velocities. Table 3.5 shows the effective values for various levels of downsampling on this dataset.

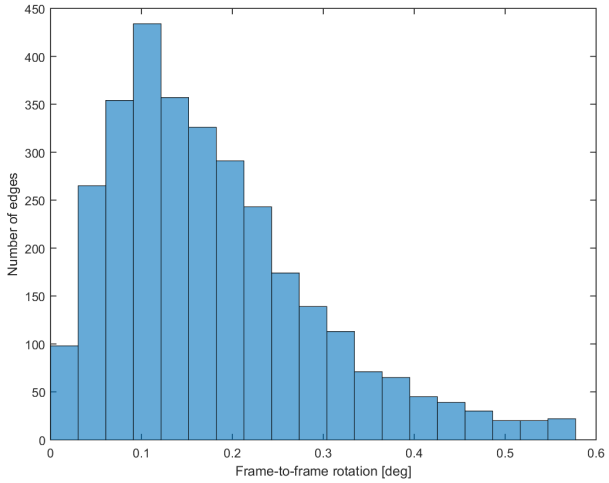
Table 3.5. The relevant parameters for all the various levels of downsampling on the fr2-rpy dataset

Frame rate	Duration s	Frames	Trans. Vel. m/s	Rot. Vel. deg/s
30.00 fps	109.53	3,182	0.01	5.82
15.00 fps	54.77	1,644	0.03	11.63
10.00 fps	36.51	1,096	0.04	17.56
7.50 fps	27.38	822	0.06	22.56
6.00 fps	21.91	657	0.07	28.85
5.00 fps	18.26	548	0.08	34.49
4.29 fps	15.65	470	0.1	38.85
3.75 fps	13.69	411	0.11	46.03
3.33 fps	12.17	326	0.13	52.23
3.00 fps	10.95	202	0.14	57.21
2.73 fps	9.96	161	0.16	63.99
2.50 fps	9.13	139	0.17	67.37
2.31 fps	8.43	131	0.18	75.26
2.14 fps	7.82	119	0.2	80.93
2.00 fps	7.3	110	0.21	86.94
1.88 fps	6.85	103	0.22	91.27
1.76 fps	6.44	97	0.24	96.64
1.67 fps	6.09	92	0.25	102.75

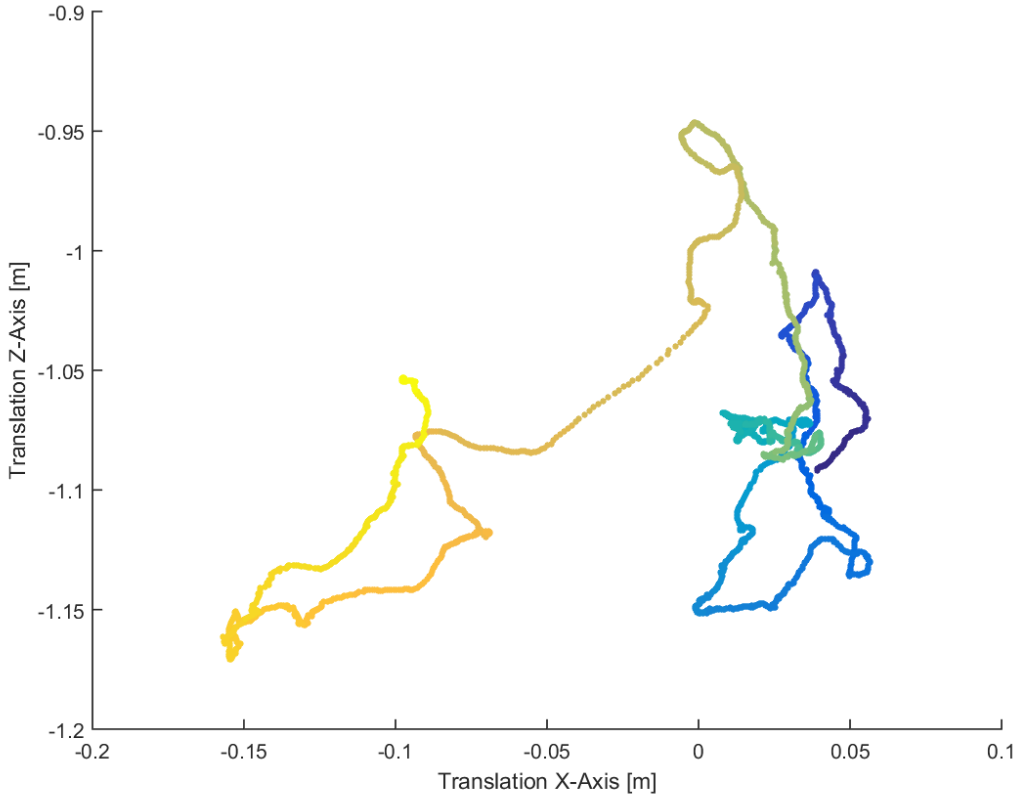
The top down trajectory shows how this dataset has a very low translation average across the entire dataset. Compared to all the others this one showed very low movement.



(a) Translation histogram.



(b) Rotation histogram.



(c) Global trajectory.

Figure 3.16. This figure shows a a histogram of the translation (a) and rotation (b) experienced between two successive frames for the fr2-rpy dataset. Image (c) shows the top down trajectory of the sensor during operation. The trajectorey is colored based on the starting location (blue) and end (yellow)

3.5.4.3 Dataset : fr1/room

Duration	:	48.90 seconds
Total translation distance	:	15.989 meters
Average translation velocity	:	0.334 m/s
Average rotation velocity	:	29.882 deg/s
Number of frames	:	1362 frames
Environment type	:	Cluttered office
Motion blur	:	Some - Minimal
Large loop-closure	:	Several

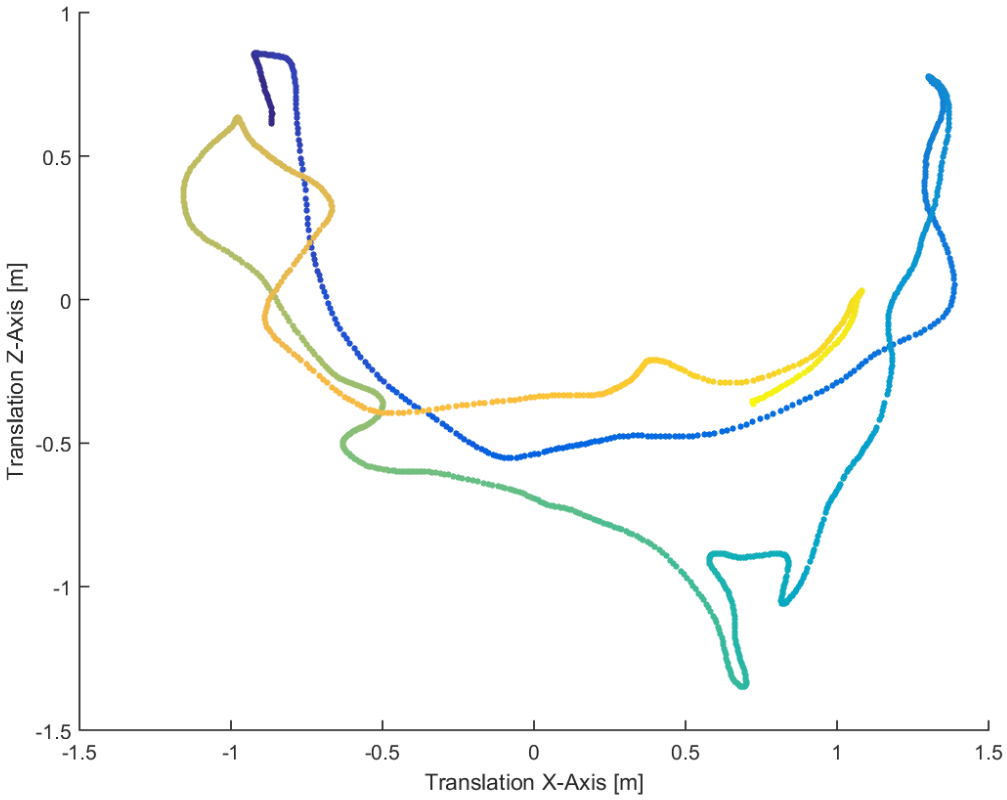
This dataset is captured using a handheld Kinect sensor and follows a trajectory through a small office environment. A single, sizeable office room is captured where the Kinect captures the entire office from multiple different orientations. The dataset contains a large number of small to large scale loop-closures, that will trigger the global loop-closure process. This dataset was created with the intent of evaluating how well a SLAM implementation is able to handle loop-closures.

This dataset contains motions that are fairly large compared to others. The rotation and translation for most of the egomotions are combined over multiple axes, and the average velocity is stable across the entire dataset. The transformation between each frame is rather large; thus, it does induce a small amount of motion blur for some of the frames into the datasets. The scenes themselves vary from extremely cluttered images that are feature rich to walls and floors that are relatively featureless. The sections where only a sparse scene is given will most likely be the portions that give the highest error rate.

Table 3.6 shows the effective values for various levels of downsampling on this dataset. Because of the large original spatial velocity, this dataset can not be downsampled multiple times.

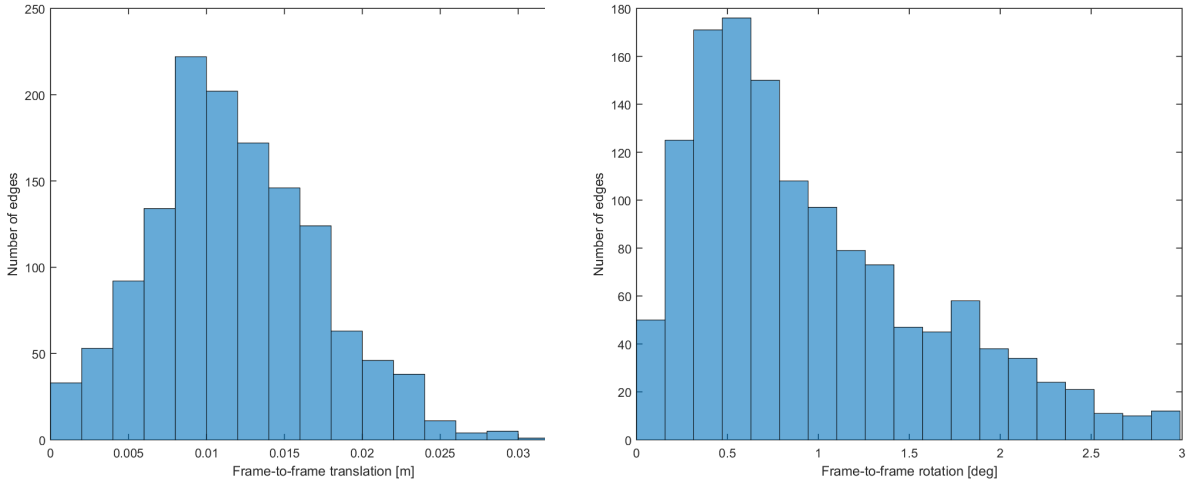
Table 3.6. The relevant parameters for all the various levels of downsampling on the fr1-room dataset

Frame rate	Duration s	Frames	Trans. Vel. m/s	Rot. Vel. deg/s
30.00 fps	45.26	1,348	0.34	29.76
15.00 fps	22.63	680	0.68	60.9
10.00 fps	15.09	453	1	89.33



(a) Global trajectory.

Figure 3.17. This figure shows the top down trajectory of the sensor during operation of the fr1/room dataset.



(a) Translation histogram.

(b) Rotation histogram.

Figure 3.18. This figure shows a a histogram of the translation (a) and rotation (b) experienced between two successive frames for the fr1-room dataset. Image (c) shows the top down trajectory of the sensor during operation. The trajectorey is colored based on the starting location (blue) and end (yellow

3.5.4.4 Dataset : fr2/pioneer_slam

Duration	:	155.72 seconds
Total translation distance	:	40.380 meters
Average translation velocity	:	0.261 m/s
Average rotation velocity	:	13.379 deg/s
Number of frames	:	2921 frames
Environment type	:	Factory corridors and tables
Motion blur	:	Minimal - None
Large loop-closure	:	Several

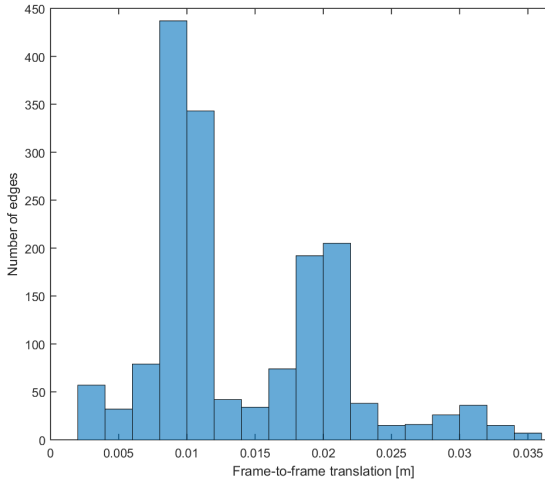
For all the pioneer datasets, the Kinect was mounted to a Pioneer land-based robot, and this setup was used to capture the resulting datasets. While the Pioneer robot is a bit unsteady, most of the rotation experienced throughout the dataset is contained to the yaw axis. Any rotations on the roll and pitch axes are due to irregularities on the floor, resulting in the Pioneer robot vibrating on those axes. The translation is also very steady across most of the frames in the dataset.

There are, however, two major difficulties associated with this dataset. The first is the fact that the frame rate drops down extensively during some parts of the sequence. While this does not invalidate the dataset, it makes it harder to localise during those frame drops. The second difficulty is the environment. The area in which this dataset was capture, is a large industrial area with some objects placed in key positions. Even though the objects are few and far between, the area lacks defining features. This makes the scene represented by the dataset very monotone and much more difficult to find defining features. Because of the relatively large translation speed the dataset can only be downsampled a few times, as shown in Table 3.7.

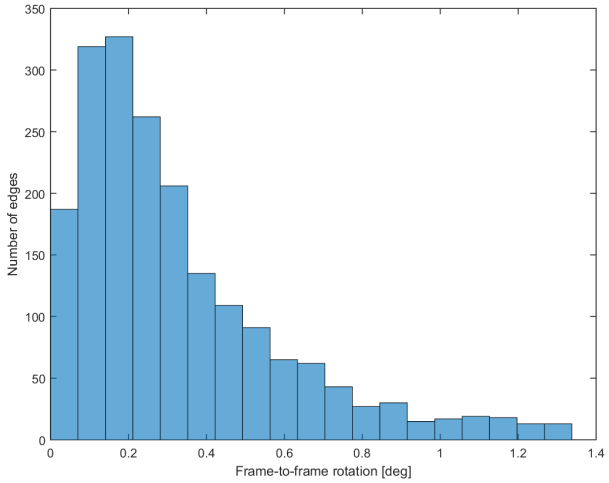
Table 3.7. The relevant parameters for all the various levels of downsampling on the fr2-pioneer dataset

Frame rate	Duration s	Frames	Trans. Vel. m/s	Rot. Vel. deg/s
30.00 fps	153.27	2,081	0.48	24.25
15.00 fps	76.63	2,008	0.52	26.94
10.00 fps	51.09	1,494	0.78	40.32
7.50 fps	38.32	1,120	1.04	53.7
6.00 fps	30.65	899	1.26	67.29

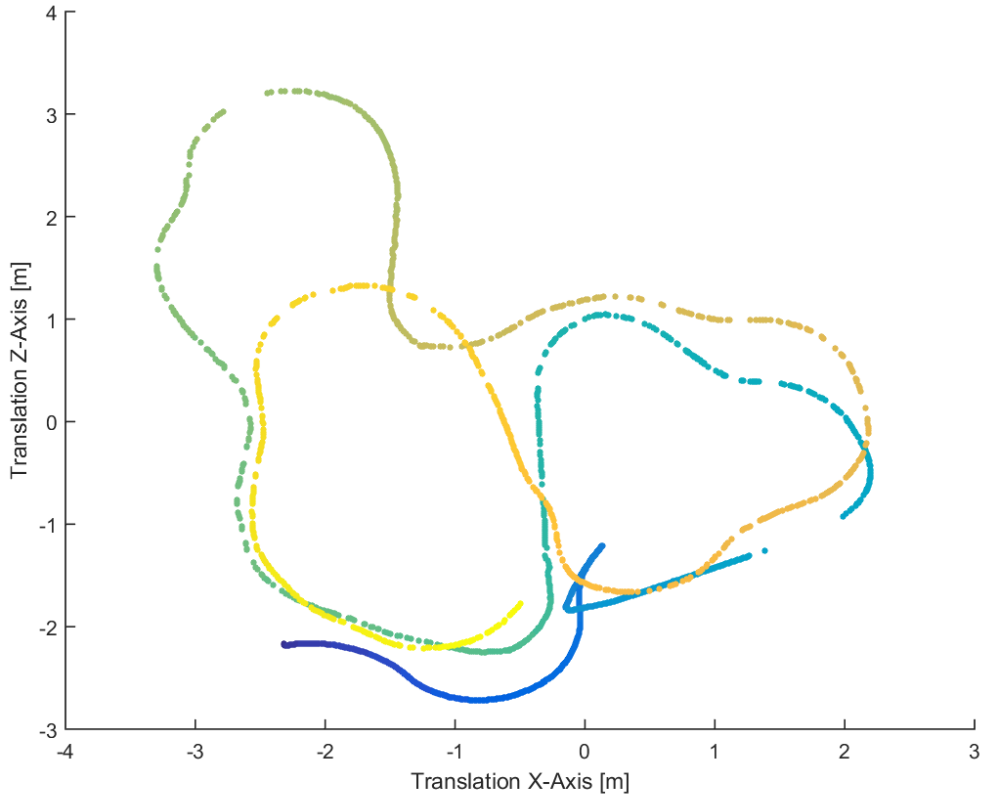
Because the pioneer robot criss-crosses a maze type layout, there are some small portions of the trajectory that is not captured by the ground truth. This can be seen in the top down trajectory plot for this dataset in Figure 3.19. Between this and the frame rate drops experienced, this dataset will push the SLAM implementation to a very large error rate.



(a) Translation histogram.



(b) Rotation histogram.



(c) Global trajectory.

Figure 3.19. This figure shows a a histogram of the translation (a) and rotation (b) experienced between two successive frames for the fr2-pioneer-slam dataset. Image (c) shows the top down trajectory of the sensor during operation. The trajectorey is colored based on the starting location (blue) and end (yellow)

3.6 CHAPTER SUMMARY

This chapter discussed the particulars of creating the various system involved with this dissertation. The quadcopter platform was created to house a Kinect RGB-D sensor, while providing a method of controlling and steering the quadcopter based on the relevant data obtained from the SLAM implementation. The interaction between all the various software applications were discussed as well as the design and implementation of SLAM. The two error metrics that will be used to evaluate the SLAM implementation was presented, alongside a detailed discussion on four of the datasets used.

The next chapter will present the results obtained for this dissertation as well as a detailed discussion on the effects that various spatial velocities have on the relative error.

CHAPTER 4 RESULTS

4.1 CHAPTER OVERVIEW

This chapter will focus on the scientific evaluation of both systems developed for this dissertation. First, in Section 4.2, the overall SLAM implementation will be evaluated in accordance with the error metrics designated and explained in the previous chapter. This will produce results that are directly comparable with other implementations that used the same error metrics and datasets.

Secondly, in Section 4.3, the relative error will be evaluated on a frame-to-frame basis over a range of different spatial velocities. Thus, the separate effect of both the translation and rotation will be evaluated, as well as the estimated performance for various levels of velocity and frame rate. These results will, in turn, be associated with each other, indicating what level of cumulative effect the translation and rotation has on each other.

The third evaluation, Section 4.4, will be on the quadcopter platform that was built to work in conjunction with the SLAM implementation. As no ground-truth readings are available for the quadcopter, it is impossible to accurately estimate the performance of this quadcopter itself, but it can be inferred from the data computed in the other sections. Thus, the platform performance will be discussed in general, and not in quantifiable terms.

4.2 SLAM IMPLEMENTATION

As mentioned in the literature study, a large number of image processing techniques are available for feature extraction and matching. This is the one aspect that varies most between normal RGB-D

implementations. Thus, three of the most prominent image processing techniques are used for visual odometry. SIFT, SURF and ORB are used to realise three SLAM implementations. For each of the implementations the parameters used to extract and match features can be changed, resulting in a variety of different features that are extracted in the image. The feature matching process can also be altered to either allow more feature pairs or fewer feature pairs. This also holds true for keyframe detection, loop-closure and graph optimisation. Through altering all these parameters, a few different RGB-D SLAM implementations can be realised.

This section will be devoted to evaluating each subsection of the SLAM implementation separately, and then evaluate the overall system against a few prominent configurations.

4.2.1 Image processing

This section will focus on the results produced by the various image processing techniques that were implemented to extract features and calculate descriptors for them.

For the SIFT CUDA implementation and the two OpenCV implementations, SURF and ORB, a variety of parameters were iterated over to find the best perceived combination for each of the three methods. It should be noted that ideally a single image should not produce more than 1024 features, as each image is allocated 1024 threads during the feature matching process, and any more will result in doubling the processing time. It should also be noted that the CUDA implementation of SIFT was verified by direct comparison with the OpenCV implementation of SIFT.

For each set of parameters used, a group of test images were used to measure the effect these parameters had on the feature extraction process. Two images from each of the datasets were used to test the combination of parameters with regards to the number of features extracted, number of feature pairs found and the processing time.

Table 4.1 shows the relevant parameters that were used for the three implementations, and Table 4.2 shows the relevant performance for each.

Table 4.1. Image processing parameters for SIFT, SURF and ORB. These parameters were chosen to best match the data that was used (with regards to contrast, size and environment). An in-depth evaluation will not be presented, as this is a topic that has already been thoroughly discussed over the last decade.

SIFT	
Number of octaves	6
Scale per octave	3
First octave blur	$\sigma = 0.8$
Contrast threshold	0.035
Edge threshold	15

SURF	
Number of octaves	4
Scale per octave	3
Descriptor length	128 bins
Hessian threshold	550

ORB	
Number of features	1024
Scale factor	2
Scale per octave	6
Edge threshold	31
Patch size	31

For SIFT the number of scales per octave and the number of octaves used impacted the processing time linearly, increasing the processing time by a slight margin for each increase in both parameter. The number of features were also directly associated with the size of both, although the number of additional features are reduced significantly for each new level in the the pyramid. The standard deviation used for the first level in the Gaussian pyramid has a major impact on the number of features detected. A smaller number, $\sigma = 0.5$, will result in a much larger number of features, but these additional features will be very small in scale, as the majority of features detected end up being in the lowest octave for the pyramid. These smaller features will not typically be used as feature pairs, as they are not very discriminative, and tend to easily create incorrect pairs. The contrast and edge threshold was manipulated until a level of feature filtering was reached that associated with the given datasets.

For both SURF and ORB, the number of parameters that can be changed, are specified by the OpenCV implementation. For both SURF and ORB the parameters were manipulated until a satisfactory level

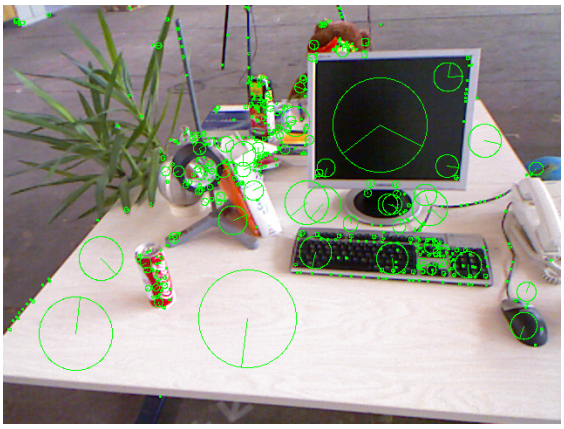
Table 4.2. Performance evaluation for each of the image processing techniques. This data was obtained using image pairs from each of the datasets that had a substantial transformation between them. The number of features detected for each process are dependent on their own parameters, but the feature matching process was similar for all three. Note that the time indicated only applies to feature detection and descriptors, and does not include feature matching.

Method	Number of features Avg. $\pm$ Std. Dev.	Feature pairs Avg. $\pm$ Std. Dev.	Processing time Avg. $\pm$ Std. Dev.
SIFT	824 ± 312	88 ± 23	18 ms ± 3
SURF	1489 ± 163	98 ± 43	9 ms ± 2
ORB	1024 ± 0	61 ± 59	16 ms ± 1

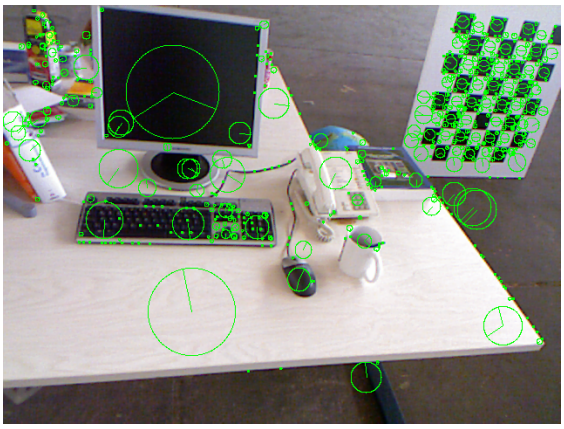
of feature pairs were established between each image pair.

Table 4.2 shows the average number of features detected between each image for each of the image processing methods. Through the SIFT parameterisation, the number of features that are detected fall roughly around 800 features per image. It was noted by Felix *et al.* [15] that the accuracy of their SLAM implementation did not benefit from more than 700 features per image; thus, 800 features per image will be more than sufficient for the SLAM implementation. SURF detected more features than SIFT on a per image basis because it was found that the number of feature pairs established were lower than that of SIFT with the same number of features. Thus, to compensate for this, the number of SIFT features were increased to roughly 1500 per image. As the OpenCV implementation of ORB can place a limit on the number of features, each image returns the best 1024 features available. As both the SIFT and SURF implementations are based on CUDA, the processing time is significantly reduced compared to normal CPU implementations. As for ORB, this extraction technique has already been one order of magnitude faster than normal SIFT and SURF; thus, its normal CPU processing time is also very low.

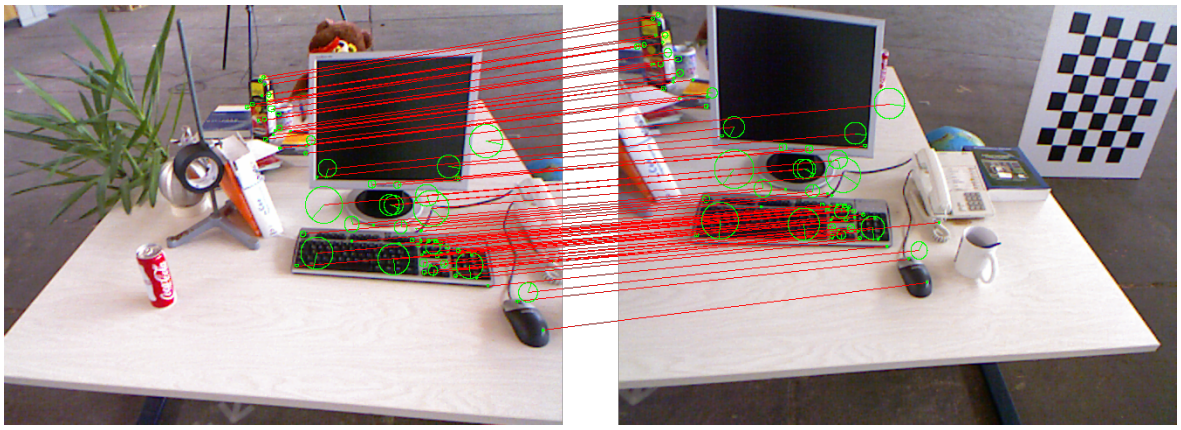
Figure 4.1 shows one of the image pairs that were used for the tests. These image pairs were chosen to represent a large transition between frame; thus, the number of feature pairs between images will be considerably lower than usual.



(a) First image



(b) Second image



(c) Feature pairs found using SIFT and the defined configuration parameters

Figure 4.1. This figure shows a single pair of images in the dataset fr2/rpy that are matched to each other. Image (a) and Image (b) show the two images with all the features found, as well as the direction of the descriptors. Image (c) shows the features pairs established between the two images, using SIFT.

4.2.2 Effect of feature pair rejection due to nearest neighbours

For feature pair rejection, a two step process is used. The first step is to reject feature pairs solely based on the distance between the two features descriptors. Thus, for each feature in the target frame, the single best matching feature is found, and the pair is accepted or rejected based on the distance between the descriptors in descriptor space. This processes is not very effective, as discussed by Lowe [55] in the original publication of SIFT.

The approach proposed by Lowe, and in turn adopted for this implementation, is to use the distance comparison between the best feature descriptor and the second best feature descriptor. Accordingly for each feature pair that passed the original descriptor space test, a comparison is made between the, distance of the feature pair in question and the second closest neighbour in descriptor space. A ratio of 0.6 is used; thus, a feature pair is only established if the fraction between the two is less than 0.6, as shown in (4.1).

$$S-NNMMT = 0.6 < \frac{distance_{NN}}{distance_{S-NN}} \tag{4.1}$$

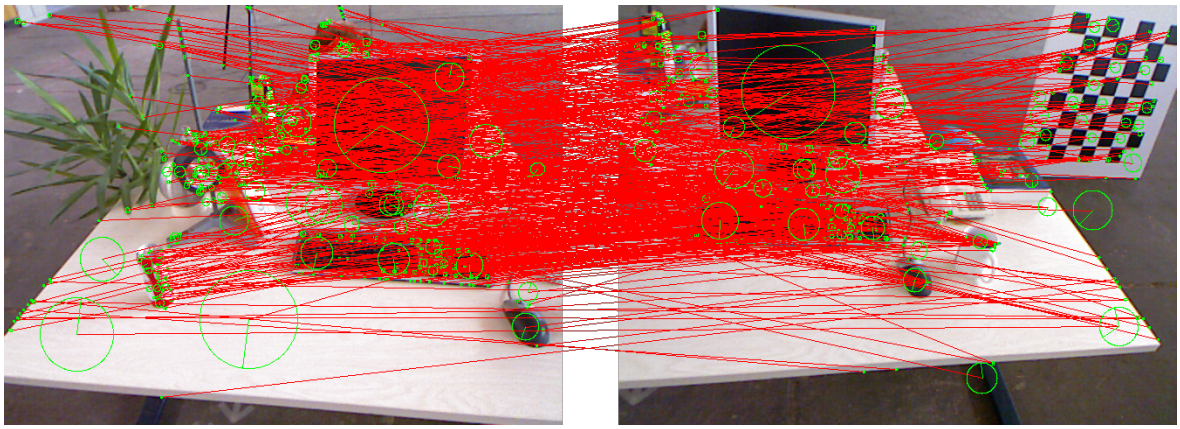
The effect of this process is shown in Figure 4.2.

Both the nearest neighbour minimum matching threshold (NNMMT) and the second nearest neighbour minimum matching threshold (S-NNMMT) have a direct correlation with the number of features matched between images. For the NNMMT the rejection ratio cannot be increased too much, as this will force the matching process to be too discriminative with regards to invariance to rotation and scale. Thus, the NNMMT is chosen to be relatively relaxed, but the S-NNMMT is used to only accept feature pairs that are fairly unique. Through this two step process the matching between two images returns feature pairs that are invariant to rotation and scale to a high degree, but still very accurate.

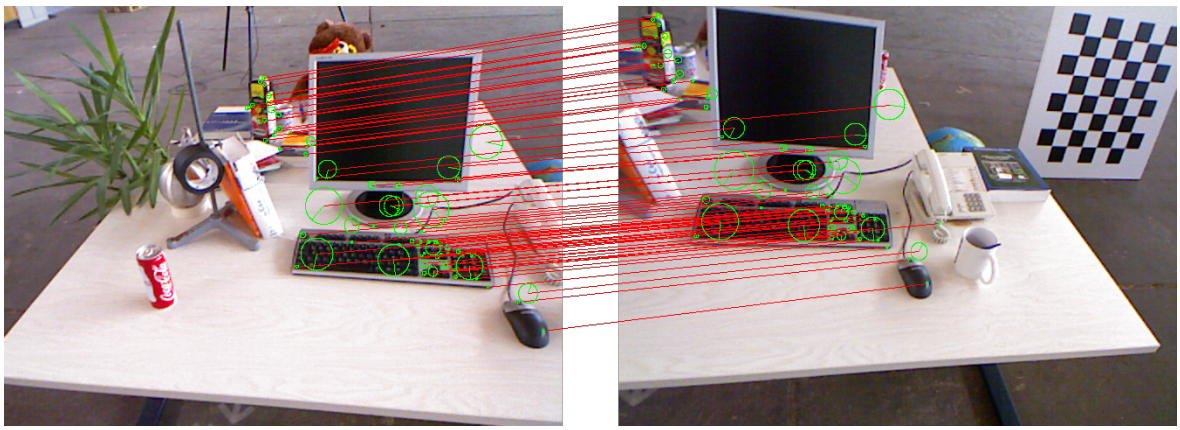
Table 4.3 shows the number of feature pairs established for a varying level of S-NNMMT, using image pairs that have an average translation of 0.19 mm and average rotation of 0.05° per frame.

Table 4.3. Feature pairs established using image pairs that are very close to each other. This indicates the average number of features pairs established for four different levels of S-NNMMT rejection ratios. The image pairs used to generate these results were very close to each other, representing a typical small transition.

Method	S-NNMMT (Small transition)			
	ratio = 0.1	*ratio* = 0.4	*ratio* =0.6	*ratio* =0.9
SIFT	74 ± 9	219 ± 27	267 ± 26	391 ± 52
SURF	82 ± 11	247 ± 43	280 ± 41	434 ± 102
ORB	62± 32	176± 41	193 ± 53	328± 111



(a) Feature matching based only on the nearest neighbour



(b) Feature matching based on the second nearest neighbour

Figure 4.2. From Figure (a) it can be seen that only using the nearest neighbour produces a large number of false associations, and each feature in the target frame can be matched to multiple other features. Figure (b) shows the overall effect of applying a second nearest neighbour approach with a minimum matching threshold of 0.6. This process significantly reduces the number of matches, and produces feature pairs that are much more likely to be true pairs between the images.

For a given image pair that has a small transition between them the number of features pairs established is highly dependent on the rejection ratio. For a ratio of 0.1, less than 10% of the features between two images will be paired. Even though these feature pairs will be highly accurate, such a high rejection rate for a small transition will be very detrimental to the SLAM implementation. At a ratio of 0.6, roughly 30% of the SIFT features are paired. This would produce a large number of points for the RANSAC validation.

Table 4.4 shows the same information as Table 4.3, except that the image pairs used have a much greater translation and rotation between them. For this table the average translation is 0.18 m and average rotation of 23.87° per frame.

Table 4.4. Feature pairs established using image pairs that are relatively far from each other. This table indicates the average number of features pairs established for four different levels of S-NNMMT rejection ratios.

Method	S-NNMMT (Large transition)			
	$ratio = 0.1$	$ratio = 0.4$	$ratio = 0.6$	$ratio = 0.9$
SIFT	3 ± 2	53 ± 7	90 ± 11	234 ± 27
SURF	4 ± 2	52 ± 11	102 ± 16	266 ± 35
ORB	1 ± 1	37 ± 19	67 ± 34	204 ± 64

From the data above it can be seen why a rejection ratio lower than 0.6 cannot be used. Although the transformation between the image pairs used to calculate the results above is fairly high, it does not represent the largest rotations and translations that will be experienced during testing. Given a rejection ratio of 0.6, the number of feature pairs have already dropped to just above 10%, giving RANSAC a much smaller set of points to validate the transformation.

Thus, using a ratio of 0.6 for the S-NNMMT the feature matching process finds a balance between the number of feature pairs established and the rotational and translational invariance of the algorithm. The number of false-positive feature pairs detected using this ratio are fairly low for SIFT and SURF, but has shown to be slightly higher for ORB.

4.2.3 Front-end egomotion estimation

Once an acceptable set of feature pairs have been formed, the rigid body transformation matrix is calculated. A minimum of three points can be used to calculate this transformation, but more can also be used to improve the accuracy, but in turn also increase the processing time. For this dissertation the four features that are the closest to their respective pairs in descriptor space are used to calculate the transformation. RANSAC is used to verify the estimation. The number of inliers that RANSAC returns,

will be highly dependent on the effective transformation between the two images; thus, RANSAC calculates the percentage of feature pairs that are inliers. If RANSAC returns less than 60% of the feature pairs as inliers with a rejection distance of 3 cm, a poor transformation is assumed and another is attempted, using a new pair of 3D points. This process is repeated up to 10 times, and the transformation with the highest level of inliers is finally chosen.

The process of feature detection, description and egomotion estimation constitutes the full front-end of the SLAM implementation. Table 4.5 gives the RPE error for the translation and rotation on a frame-to-frame basis over each of the datasets.

Table 4.5. Using the RPE error metric, the average translation and rotation error is computed in a frame-to-frame basis. This does not compare how accurate the global map representation is, but rather indicates the performance of the front-end visual odometry. Using the parameters indicated in Table 4.1, the error is computed for each of the datasets using the full potential frame rate.

Sequence	SIFT		SURF		ORB	
	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]
fr1/xyz	6.03	0.478	9.14	0.655	14.94	0.894
fr1/rpy	9.19	0.904	12.66	1.284	17.73	1.85
fr2/xyz	2.51	0.118	3.7	0.149	5.12	0.12
fr2/rpy	2.83	0.142	4.66	0.216	7.27	0.297
fr1/desk	9.26	0.742	12.73	1.087	18.81	1.553
fr1/room	11.67	0.664	16.07	0.888	25.66	1.355
fr1/pioneer/360	19.99	0.937	29.29	1.341	41.69	1.939
fr1/pioneer/slam	30.58	0.95	43.4	1.38	61.6	2.067

As expected, the best performing datasets are both the fr2 benchmark datasets. This is because the average rotation and translation is very small. It can also be seen when comparing the *xyz* and rpy datasets that a high rotation has more of an effect than a high translation. This is because a rotation causes more scene variance than translation does. The fr1/room dataset also had a higher error rate than the fr1/desk dataset because a few sections of the dataset are fairly featureless, and inflated the overall error. The rotation and translation experienced in the fr1/desk dataset is also often isolated to a single axis, where the fr1/room dataset is a large mixture of movement and rotation on all the axes.

On average SURF produced results that were 30% higher than that of SIFT. This is to be expected even though the number of feature pairs that SURF produces, is larger. SURF only produces more features because the parameters were relaxed, allowing for more SURF features per image than SIFT does. This, in turn, made the matching process prone to a higher rate of false-positives. While ORB has fewer feature pairs on average, the features pairs are worse than those for SIFT and SURF. This can be seen by the relative performance of ORB compared to SIFT and SURF.

The Pioneer dataset performed significantly worse in general than any of the other datasets, mainly because the scene is featureless and very monotone, resulting in feature descriptors that are much less discriminative. The average and standard deviation for each of the methods are given in Table 4.6.

Table 4.6. Average and standard deviation for each image processing method over all the datasets.

	SIFT	SURF	ORB
	mean $\pm$ std	mean $\pm$ std	mean $\pm$ std
RPE [mm]	11.51 ± 9.51	16.45 ± 13.49	24.10 ± 18.95
RPE [deg]	0.617 ± 0.340	0.875 ± 0.492	1.259 ± 0.748

Clearly SIFT performs much better compared to both SURF and ORB. Even though SIFT is normally much more expensive computationally, the biggest disadvantage of SIFT has been negated by doing most of the calculations on a GPU. Considering that SURF processes almost three times faster than SIFT, with both running on the GPU, SURF only lags behind SIFT by approximately 30%. ORB almost doubles the error rate of SIFT, but the time it takes to extract features on the CPU is comparable to the GPU implementations of SIFT and SURF. Thus, ORB will be ideal for situations where only a CPU is available for processing, such as a small development board. SIFT and SURF are thus better suited for implementations with an abundance of processing power, and ORB is more suited for limited resource situations, such as onboard processing.

4.2.4 Loop-closure

Even though the local and global loop-closure processes combine to form the back-end, each still has to be evaluated on its own. However, as the global loop-closure depends on the local loop-closure

to create keyframes, the global loop-closure process can only be evaluated in conjunction with the local loop-closure. For the following sections SIFT will be used for the in-depth evaluations at each point, while SURF and ORB will be compared in a more general sense. This is because SIFT will return the most optimal results compared to the others, and does not lag behind in terms of processing time.

4.2.4.1 Keyframe selection

For the given implementation two methods for keyframe selection were created. The older method of selecting frames based on an arbitrary passage of time has been proven to be very inefficient in many previous studies. The newer method and the one used to generate all the keyframes for this implementation uses the number of RANSAC inliers to determine if a new keyframe should be created. Table 4.7 indicates the number of keyframes created for each of the given datasets, using both methods.

Table 4.7. A comparison between the number of keyframes generated for each dataset. Even though the RANSAC method is the *de facto* approach for most of the recent implementations, the older method is still implemented and compared. This data is produced using the full frame rate for each of the datasets while SIFT is used for feature detection.

Sequence name	**Duration** [s]	**Number of frames**	**Keyframes selected(old)**	**Keyframes selected(new)**
fr1/xyz	30.09	798	88	29
fr1/rpy	27.67	723	79	44
fr2/xyz	122.74	3669	336	35
fr2/rpy	109.97	3290	327	61
fr1/desk	23.4	613	69	96
fr1/room	48.90	1362	145	275
fr2/pioneer_360	72.75	1225	216	397
fr2/pioneer_slam	155.72	2921	341	743

In general the RANSAC method selects fewer keyframes in the datasets where the scene being captured stayed consistent. Thus, for the benchmark datasets the number of keyframes were reduced significantly. This is because the keyframe selection does not add another keyframe if a local or global loop-closure has been detected. Consequently, for the majority of observations, a positive match is found in either the local or global loop-closure. For datasets where the scene changes consistently, the number of keyframes detected depend on the relative spatial velocity that the sensor underwent. In some cases, particularly the fr1/room dataset, the number of keyframes are significantly higher, as the sensor is moving at very high levels of rotation and translation.

The number of keyframes generated using the older methods will produce unacceptable results, particularly for the fr1/room dataset, where a larger number of keyframes are required, since the sensor is moving very quickly between scenes that are relatively new. Alternatively, there is the fr2/xyz dataset that has seen a huge drop in keyframes selected. This is because the scene variation in the entire dataset is very low, resulting in only a few keyframes being created.

The Pioneer datasets both produced a high number of keyframes because of the type of environment. Because of the limited features the process will create a new keyframe much easier than for a cluttered scene.

4.2.4.2 Back-end egomotion estimation

The final section of the SLAM implementation that needs to be evaluated is the overall performance with the full loop-closure process. The system was tested over all the datasets using only the front-end in Section 4.2.3, and the same metric and evaluation process was followed for both the local loop-closure and the full loop-closure.

Table 4.8 shows the RPE error for all three instances on a frame-to-frame basis. VO donates the accuracy using only visual odometry, VO+LLC donate the use of visual odometry with only local loop-closure and VO+GLC is the full SLAM implementation, using visual odometry and both the local and global loop-closure processes. For this evaluation only SIFT was used and the full frame rate for each dataset was used.

Table 4.8. Direct comparison between only using the front-end (VO), using the front-end and local loop-closure (VO+LLC) and the full SLAM implementation (VO+GLC). The RPE error metric is used on a frame-to-frame basis.

Sequence	VO		VO+LLC		VO+GLC	
	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]
fr1/xyz	6.03	0.478	5.37	0.422	5.1	0.398
fr1/rpy	9.19	0.904	9.67	0.94	7.74	0.752
fr2/xyz	2.51	0.118	2.26	0.101	2.16	0.095
fr2/rpy	2.83	0.142	2.54	0.123	2.42	0.117
fr1/desk	9.26	0.742	8.2	0.658	7.7	0.623
fr1/room	11.67	0.664	12.17	0.697	9.76	0.558
fr1/pioneer/360	19.99	0.937	17.67	0.826	21.63	1.794
fr1/pioneer/slam	30.58	0.95	26.99	0.843	33.08	1.827

Interestingly, this indicates that the local loop-closure and graph optimisation process does not always benefit the frame-to-frame accuracy experienced, as the process is not perfect and can over-generalise the error across the frames included within the loop.

For most of the datasets the addition of local loop-closure improved the frame-to-frame accuracy, except for the fr1/rpy and fr1/room datasets. It can be surmised that the effect of a large rotation and translation between each frame result in a very large transformation between each of the three keyframes used; thus, increasing the chances that a bad feature matching process or a bad transformation estimation can occur. The effect of two or three bad matches is minimised by the limitation put on the local loop-closure, but the effect can still be seen on the results.

For both the Pioneer datasets the addition of local and global loop-closure differed greatly. The local-loop closure increased the effective accuracy by a fair margin. This most likely occurred because there is a larger than normal amount of drift between each observation due to the type of environment. The global loop-closure, however, drastically increased the error rate, resulting in a much higher rotation and translation error, and was the result of false loop-closures. Thus, because the scenes are very monotone, a few false loop-closures resulted in the pose-graph optimization correcting for a false drift

value. The system was retested, using a much stricter global loop-closure process, and the error was reduced to near the VO+LLC level, but a large amount of potential loop-closures were ignored due to this.

The global loop-closure process showed most improvement for datasets where there were multiple large loop-closure opportunities, given a feature-rich environment. For datasets where the scene stays relatively the same (xyz and rpy datasets) the effect of global loop-closures were minimal, where the effect was fairly extensive for the room and desk datasets.

By increasing the sensitivity of the global loop-closure, the Pioneer datasets were improved but all the others were negatively affected. Conversely, by relaxing the parameters that dictated a loop-closure the error decreased for some datasets but increased for others. The benchmark datasets did not see a large improvement, as these datasets have a large amount of features and the scene stayed the same. The fr1/room dataset saw a marginal increase in performance, but a major drop once the parameters were relaxed too much. The pioneer dataset saw a rapid increase in the error rate as the parameters were relaxed too much, as a large amount of false local loop-closures were being made.

Table 4.9 shows the average error over all the datasets for each of the image processing techniques.

Table 4.9. The average RPE error recorded for each of the image processing techniques.

Sequence	VO		VO+LLC		VO+GLC	
	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]
SIFT	11.51	0.617	10.61	0.576	11.20	0.770
SURF	16.45	0.875	14.28	0.763	13.48	0.719
ORB	24.10	1.260	18.34	0.987	16.33	0.871

Both SIFT and SURF saw notable improvement when using GLC, but ORB showed the most improvement with a very large effect for both the local loop-closure and global loop-closure. This can be attributed to the larger error that is imposed for each frame; thus, the drift correction that can be made, is much larger than for SIFT and SURF. The VO+GLC values are inflated here because of the Pioneer datasets' false loop-closures.

4.2.5 Absolute trajectory error

The RPE error in a frame-to-frame evaluation does not accurately represent the overall performance of a SLAM implementation. Unlike the RPE error, the ATE error first aligns the two trajectories for each dataset before finding the absolute error. This metric is much more informative on how well the SLAM implementation estimated the global trajectory of the sensor.

For all the ATE errors computed in this dissertation the online tool provided with the datasets were used to compute the ATE error for each sequence. Table 4.10 compares each of the SLAM implementations with regards to their ATE error.

Table 4.10. The ATE error is computed for each dataset, using the image processing technique indicated. Both the translation and rotation error is recorded, using the full framerate available in each dataset. This is the error for the full SLAM implementation.

Sequence name	**SIFT** **ATE** [m]	**SIFT** **ATE** [deg]	**SURF** **ATE** [m]	**SURF** **ATE** [deg]	**ORB** **ATE** [m]	**ORB** **ATE** [deg]
fr1/xyz	0.025	0.93	0.027	0.97	0.047	1.97
fr1/rpy	0.047	2.54	0.051	2.57	0.083	5.06
fr2/xyz	0.021	0.79	0.022	0.76	0.029	1.34
fr2/rpy	0.023	1.72	0.024	1.83	0.036	4.93
fr1/desk	0.051	2.46	0.056	2.43	0.781	3.76
fr1/room	0.232	9.08	0.233	8.64	0.407	11.371
fr2/pioneer_360	0.392	13.08	0.408	15.14	0.525	19.71
fr2/pioneer_slam	0.719	23.14	0.861	21.07	FAILURE	FAILURE

As expected, SIFT produced the best results among the three implementations. SURF performance is comparable with that of SIFT, but ORB lagged behind significantly, especially with datasets that did not have a lot of feature-rich keyframes. This can be seen in the fr1/room dataset where sections of the dataset was from a scene with little defining features; thus, the overall performance degraded significantly. Figure 4.3 shows the top down trajectory error for the fr1/room dataset.

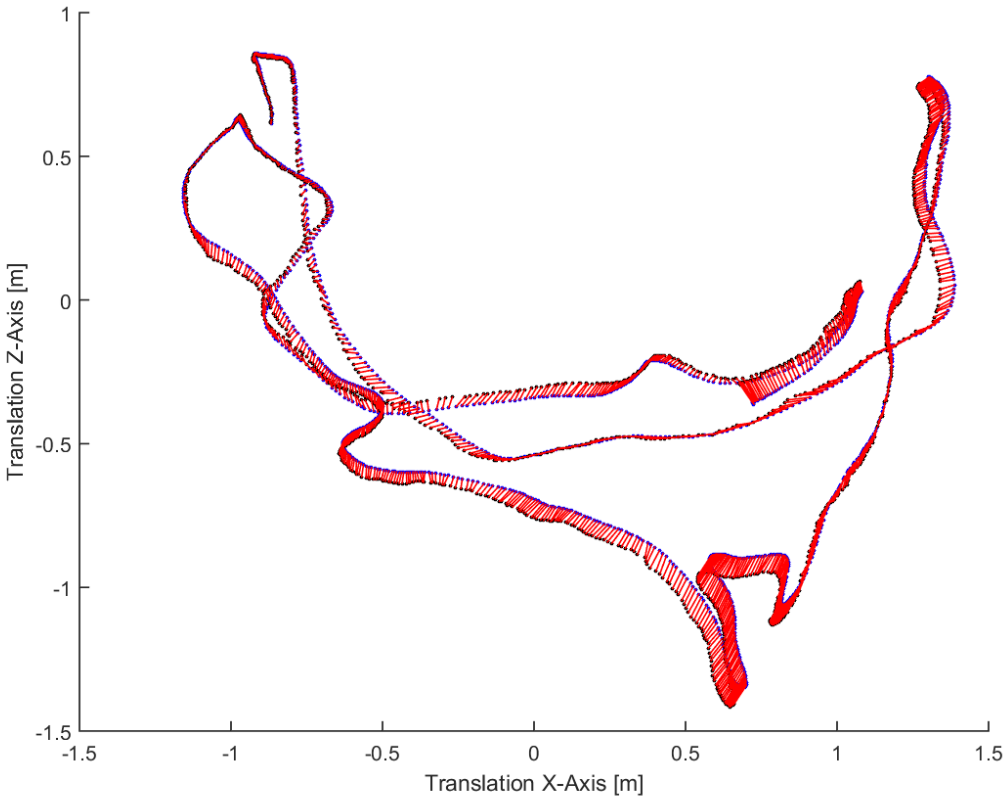


Figure 4.3. Top down view of fr1/room dataset trajectory. The black trajectory is the estimated position, where the blue is the ground truth. Each corresponding observation between the two sets of points are indicated by a red line.

Even though the Pioneer dataset has a low average transformation per frame, the dataset contains sections where the frame rate drops very low. In combination with the type of environment, that does not deliver a large number of unique features compared to the office environment, it was much more challenging to calculate the full trajectory path for the Pioneer datatsets. Similar to the RPE error, it only takes a few poor global loop-closures to throw off the trajectory. Even though there were some very poor global loop-closures, because of the least squares approach that ATE uses the error for both SIFT and SURF is still below 1m. The ORB feature detection method, however, failed to complete the full fr2/pioneer_slam dataset because major false loop-closures were detected in the environment. This made the graph optimisation process to try and correct for a drift that far exceeded what it should be, resulting in a global trajectory that completely deviated from the truth. Even tough the translational ATE error might not be overly bad, the biggest error is seen in the rotation error for each pose. Because the two trajectories are matched, based on the Euclidean position of the sensor and not the orientation,

the rotation error is very high.

Both the fr2 debugging datasets returned the best results obtained, giving a translation RMSE of 2.1 cm and 2.3 cm respectively. Figure 4.4 shows an image of the point cloud generated using the fr2-xyz dataset.

Figure 4.4. Dense point cloud of the fr2-xyz dataset.

Table 4.11 compares the SIFT ATE error computed for each dataset with that of four very influential or state-of-the-art implementations of RGB-D SLAM. Note that not all works presented resulted for datasets that had been used; thus some implementations will not have data to be compared to. Additionally, it has become common practice to only evaluate the ATE error with regards to the translation, and not the rotation error. This is because a rotation error will technically present itself within the translation error for successive frames.

The implementation for this dissertation matches the one performed by Felix *et al.* in 2012 the closest, as this implementation follows the guidelines on creating an RGB-D SLAM implementation as

Table 4.11. Comparison of the ATE translation error across this implementation and four other peer-reviewed implementations. The data from the 2012 [72] paper will coincide the most with this one, since the implementations are nearly the same. The other implementations use state-of-the-art techniques to improve the front and back-end of RGB-D SLAM. ORB-SLAM 2 ,in particular, has a combination of many state-of-the-art techniques that deliver the best results recorded.

Sequence name	**This Implementation**	**Felix *et al.*** 2012 [72]	**Felix *et al.*** 2014 [15]	**Kerl *et al.*** 2013 [50]	**ORB-SLAM 2** 2017 [73]
fr1/xyz	0.025	0.021	0.014	0.011	-
fr1/rpy	0.047	0.042	0.028	0.020	-
fr2/xyz	0.021	-	-	-	0.004
fr2/rpy	0.023	-	-	-	0.019
fr1/desk	0.051	0.049	0.027	0.021	0.016
fr1/room	0.232	0.219	0.089	0.053	0.047
fr2/pioneer_360	0.392	-	0.301	-	-
fr2/pioneer_slam	0.719	-	0.269	-	-

proposed in that paper. The biggest difference is how the loop-closure detection is done, as their approach did not attempt to operate in real-time. The other three implementations incorporate large amounts of state-of-the-art techniques that overshadow the implementation done for this dissertation. Some of the techniques implemented are the environment measurement model (EMM) proposed by Felix *et al.* in 2014, that uses ICP and distribution models to improve the back-end egomotion transformation. The effect of this can best be seen on both the Pioneer datasets. Other approaches include the use of the photometric and depth error across all the pixels in each observation, building a predictive model for the front end. ORB-SLAM 2 is the latest publication that claims massive performance increases, using updated ORB feature detection and matching algorithms that use a wide variety of map re-localisation and optimisation techniques.

Even though these state-of-the art implementations sometimes outperforms the current implementation for this dissertation by far, the results pertaining to the research goals will still be valid, as the improvements that can be made to the RGB-D SLAM system will translate directly to the results found in the next section.

4.2.6 Timing results

One of the goals of this dissertation was to create a SLAM implementation that is able to run RGB-D SLAM in real-time. This would suggest that the system is able to perform calculations on every observation before the next observation becomes available. Table 4.12 shows the timing for each separate element in the SIFT, SURF and ORB implementation. These times were recorded over four datasets, both the fr2 benchmark datasets and both the fr2/pioneer datasets.

Table 4.12. Processing time for each individual process in the SLAM-implementation. The results were generated, using SIFT and four datasets. These results were taken in a full operations setting; thus, all the parallel processes are running while these results were being obtained.

Process name	**SIFT** **Mean $\pm$ std dev.**	**SURF** **Mean $\pm$ std dev.**	**ORB** **Mean $\pm$ std dev.**
Feature extraction	18 $\pm$ 5 ms	8 $\pm$ 2 ms	15 $\pm$ 3 ms
Feature matching	12 $\pm$ 9 ms	23 $\pm$ 15 ms	9 $\pm$ 1 ms
Feature pair validation	<1 ms	<1 ms	<1 ms
Egomotion estimation	2 $\pm$ 1 ms	2 $\pm$ 1 ms	3 $\pm$ 1 ms
Local loop-closure	27 $\pm$ 17 ms	41 $\pm$ 25 ms	12 $\pm$ 2 ms
Keyframe selection	2 $\pm$ 2 ms	2 $\pm$ 2 ms	2 $\pm$ 2 ms
Global loop-closure	63 $\pm$ 16 ms	82 $\pm$ 21 ms	46 $\pm$ 8 ms
Graph optimisation	106 $\pm$ 96 ms	94 $\pm$ 68 ms	63 $\pm$ 28 ms

The processing time for ORB with regards to feature matching is significantly decreased when compared to SIFT and SURF, partly because the descriptors are not that expensive to compare and because the number of features detected will not exceed 1024. For all three processes the feature matching process takes up a significant amount of time, since a brute-force search is used to match features. This can particularly be seen for SURF, where the number of features detected on average are very high.

The feature pair validation and egomotion estimations both take a very short amount of time, as they only consist of comparing precomputed values. On average the RANSAC process returns a positive

transformation on the first feature pair transformation; thus, it rarely has to recompute with a different set of feature pairs.

The local loop-closure process compares the current observation to the last three keypoint detected. This process includes the feature pair validation and egomotion estimation steps as well. This step does not include graph optimisation. Keyframe selection is done directly after the local loop-closure, and uses the RANSAC data obtained. This process takes a relatively long time to complete with regards to its purpose, as this step includes reallocating data block sizes within the GPU.

The global loop-closure and graph optimisation process is the most time-consuming process by far, and is the main reason the back-end has to run in a parallel thread to the front-end. The current observation is compared to seven sequentially selected keyframes, and it is determined if a loop-closure has been detected. This step is highly dependent on the level of tasks currently assigned to the GPU, and is significantly faster on a datastream below 20 fps. If a global loop-closure was detected, the **g^{2}o** optimisation process is triggered. The **g^{2}o** optimisation has to run on the CPU as the implementation does not allow for CUDA, and can take an excessive amount of time. The processing time is highly dependent on the size of the pose-graph that has to be optimised and the size of the drift that has to be corrected. This process does not trigger very often, as the parameters for a loop-closure is very strict to prevent as many false-positive loop closures as possible. For the benchmark datasets fr/xyz and fr/rpy there is typically only a handful of full loop-closures, and for the larger datasets only a few dozen, dependent on the environment and the frame rate. Without the local loop-closure process, this number would be much higher, increasing the average processing time of the back-end. Table 4.13 shows the average time associated with the front-end, back-end without a global loop-closure and the back-end with a global loop-closure.

Table 4.13. Average processing time for the front-end, back end without global loop-closure and the full SLAM implementation.

Method	SIFT	SURF	ORB
	Mean $\pm$ std dev.	Mean $\pm$ std dev.	Mean $\pm$ std dev.
Front-end	35 $\pm$ 15 ms	33 $\pm$ 19 ms	27 $\pm$ 5
Back-end without GLC	92 $\pm$ 34 ms	125 $\pm$ 48 ms	60 $\pm$ 12
Back-end with GLC	199 $\pm$ 129 ms	207 $\pm$ 116 ms	122 $\pm$ 41

The front-end thread is able to process most data from a 30 Hz sensor; thus, the Kinect running at 30 fps will be perfect for this implementation. The front-end sometimes takes more than the allocated 34 ms to complete a single observation, typically happening when there are a large number of features within the observation made. By disabling the back-end of the SLAM implementation the processing time of the front-end decreases by about 10%, indicating that though the back-end utilising the GPU in parallel with the front-end effects performance, it does not hamper the system overly much. This is due to the fact that a priority queue is used to allocate tasks to the GPU, with the front-end always taking preference.

If no global loop-closure was detected, the average processing time for the back-end is under 100 ms. This would indicate that the system drops roughly two out of three frames presented to the system. It should be noted that these times recorded, were for the largest datasets available, so the pose-graph associated with each dataset is much longer than it would have been for a typical dataset. If a successful global loop-closure was detected, the processing time near doubled, due to graph optimisation.

4.3 ESTIMATING PERFORMANCE AGAINST SPATIAL VELOCITY

To be able to give an estimate of the performance that can be expected at a particular spatial velocity, a few parameters must first be discussed. First is the error metric. For the purpose of estimating the accuracy versus the spatial velocity the RPE error will be used. The ATE error is based on how well the global trajectory matched the ground-truth trajectory, and thus it is highly dependant on the dataset itself as a whole and the level of loop-closures that occur within the dataset. For the ATE error the least squares matching performed on the trajectory will average out the error across the various spatial velocities experienced. Thus, to prevent this, the RPE error is used, as this will accurately represent the error experienced for that egomotion alone. Therefore, the ATE error can be said to average the error across all poses, where the RPE error is much more representative of the frame-to-frame operations.

In an attempt to be impartial to the nature of the dataset itself as whole, the RPE between sequential frames will be used. Obviously the system is still optimising the poses using the back-end, but each individual error metric between two observations will not be reliant on the average accuracy over the entire dataset.

The second parameter that needs to be discussed, are the observations themselves. As could be seen from the previous section, the type of environment will play a crucial role in the overall performance. Thus, for the results given, it should be considered that the environment can affect the results by a fair margin. The majority of datasets used to generate the results in this section were that of a cluttered office environment; thus, the results would reflect that environment. For any implementation on a dataset that contains sparse features it should be noted that the real accuracy will be lower.

This section will only be evaluated using SIFT, as it has proven to be the most versatile and accurate image processing technique. Because of the large amounts of data that will be worked with, it was seen as unnecessary to evaluate the lesser performing methods.

4.3.1 Translation effects

For this section the effect of the translation on the overall performance will be evaluated. To accomplish this only spatial motions where the sensor underwent very little rotation will be considered. From all the datasets available, each observation pair was evaluated with regards to the rotation the sensor underwent. If it was found that the sensor underwent a rotation smaller than 0.3°, the egomotion and RPE error associated with that observation pair were recorded. Thus, any motions that exceeded 9 deg/sec rotation were eliminated.

4.3.1.1 Single axis translation

To account for the effect that each axis has on the overall performance, the dataset of samples obtained were further decomposed. Each egomotion translation was evaluated based on the axis of movement. If it was found that the predominant axis accounted for more than 80% of the relative egomotion, the sample was used for that particular axis performance calculations. Table 4.14 shows the results obtained using this methods of sampling.

Even though there was little difference seen between the x and z-axis translations, a noticeable decrease in the error can be seen for the z-axis. The z-axis corresponds to a scale change in the image; thus, it can be inferred that the SIFT feature extraction algorithm matches feature pairs with a scale change much better than features with a sideways motion. This effect becomes more pronounced as the size of

Table 4.14. Single axis translation comparison over the three axes of translation. This data was computed by evaluating the various image pairs that had an egomotion associated with the parameters. Only egomotions that had a rotation below the threshold were accepted. The number of image pairs declined significantly as the translation increases, as the possibility that a small rotation occurred with a large translation decreased sharply.

Translation	X-axis		Y-axis		Z-axis	
	Frames	RPE [mm]	Frames	RPE [mm]	Frames	RPE [mm]
1- 2 mm	517	2.14 ± 1.27	463	2.05 ± 1.27	617	1.76 ± 1.13
2- 4 mm	942	2.85 ± 1.47	831	2.75 ± 1.51	1,173	2.31 ± 1.25
4- 6 mm	673	3.44 ± 1.69	721	3.37 ± 1.51	833	3.07 ± 1.44
6- 8 mm	427	4.03 ± 1.57	573	3.87 ± 1.62	682	3.35 ± 1.43
8- 10 mm	431	5.09 ± 1.84	483	5.20 ± 1.91	541	4.49 ± 1.61
10- 15 mm	176	6.52 ± 2.50	211	6.73 ± 2.32	250	5.76 ± 1.97
15- 19 mm	92	10.93 ± 3.57	102	10.23 ± 3.63	119	8.82 ± 3.03

the translation increases. The ability to match features with a different scale is highly dependent on the number of layers within the SIFT pyramid constructed during feature detection. The more layers, the better the SIFT algorithm will be with regards to scale invariance.

The error rate when comparing the actual translation with the detected one is fairly high for all the datasets used. This is because of two factors. The first is the image quality. While the images captured are sufficient, they are far from ideal. In most of the images a single pixel can represent a real world area that covers a few centimetres, adding a lot of inherent noise into the data. The second factor is the error metric being used. The RPE error, for any single frame-to-frame measurement, will still contain the accumulated drift or potential erroneous calculations made previously. This is presented by the fairly large standard deviation experienced.

It should also be noted that while the error itself increases as the size of the translation increases, the effective error percentage with regards to the translation decreases. This is most likely due to the small size of the images captured by the Kinect, as a 640×480 image will have a limit to how well it can match features. If the RGB image size is increased, the error percentage seen for small transitions

should decrease significantly.

4.3.1.2 Multi-axis translation

Typically the translation experienced between two observations will have an effective motion that is not predominantly on a single axis. Thus, the sensor can move in any direction that is a combination of the three axes. To compute the RPE error for this situation, the only filtering that was done on the observation pairs was to check that the rotation was below the 0.3° threshold. This yielded a much larger dataset of observations. This table also shows the relative rotation and translation experienced per 1 mm translation at each level.

Table 4.15. The translation and rotation RPE error values over a range of translations. These translation can be a combination between any of the three translation axes. Only egomotions that had a rotation below the rotation threshold was accepted.

Translation mm	Frames	RPE [mm] mean $\pm$ std	RPE [mm] per 1 *mm*	RPE [deg] mean $\pm$ std	RPE [deg] per 1 *mm*
1- 2 mm	1,847	2.04 ± 1.24	1.36	0.106 ± 0.053	0.071
2- 4 mm	3,319	2.63 ± 1.41	0.88	0.135 ± 0.059	0.045
4- 6 mm	2,530	3.35 ± 1.55	0.67	0.165 ± 0.058	0.033
6- 8 mm	2,046	3.86 ± 1.53	0.55	0.188 ± 0.057	0.027
8- 10 mm	1,727	4.89 ± 1.80	0.54	0.226 ± 0.071	0.025
10- 15 mm	753	6.44 ± 2.20	0.52	0.296 ± 0.101	0.024
15- 19 mm	342	9.89 ± 3.36	0.58	0.386 ± 0.131	0.023

No noticeable changes were seen when combining the three axes of translation. This is to be expected, as each axis should contribute equally to the final error in translation. For each of the different levels of translation the effective rotation RPE was also calculated. The rotation error associated with each level of translation seems to scale up exponentially with a very slight angle. This should increase to a much larger rotation error as the translation becomes unreasonably large.

The table shows the translation error per millimetre at each of the indicated levels. This shows the intrinsic minimum error rate associated with this particular type of data. The resolution size will affect

this number the most. Thus, for very small translations, the accompanying error is much higher with regards to the translation experienced, when compared to larger translation. This is because of the error rate associated with the Kinect sensor. For the best total translation versus error percentage a translation of 10 mm is shown, and is also similar for the rotation error. It can be seen that the effective error per millimetre starts to increase again as the translation reaches $15 - 19$ mm, resulting from a larger scene variation.

4.3.1.3 Translation velocity only

If the platform that has the sensor mounted to is only prone to translation between observations, and does not contain rotations larger than 9 deg/sec, an effective comparison between translation velocity and frame rate can be made, using the results obtained above. Even though this is not a typical situation that a SLAM implementation will find itself in, it would be advantageous to evaluate this possibility.

Given the velocity and frame rate, the effective average translation per observation can be calculated, and the resulting RPE can be found. For this dissertation an average velocity of between 0.05 m/s and 0.4 m/s will be considered. This covers the relative range of velocities that might be found for typical land and air based platforms. Values that fall outside the scope of evaluation were estimated, using the relative curvature of the RPE error.

4.3.2 Rotation effects

For this section the effect that the rotation has on the overall performance will be evaluated. To accomplish this, only spatial motions where the sensor underwent very little translation will be considered. From all the datasets available, each observation pair was evaluated with regards to the translation the sensor underwent. If it was found that the sensor underwent a translation that was smaller than 3 mm, the egomotion and RPE error associated with that observation pair was recorded. Thus, any motions that exceeded 0.09 m/s translation were eliminated.

Table 4.16. For a system that has translation only, this table can be used to estimate the RPE error for both the rotation and translation. A range of translation velocities versus three frame rates are given.

Translation	30 *fps*		20 *fps*		10 *fps*	
velocity	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]
0.05	2.04	0.106	2.5	0.126	3.21	0.159
0.10	2.77	0.145	3.21	0.159	5.23	0.236
0.15	3.21	0.159	4.09	0.197	7.43	0.335
0.20	3.64	0.179	5.23	0.236	12.77	0.436
0.25	4.55	0.217	6.53	0.297	21.94	0.568
0.30	5.23	0.236	7.43	0.335	37.69	0.739
0.35	5.93	0.277	10.27	0.397	64.77	0.963
0.40	6.7	0.312	12.77	0.436	111.28	1.253

4.3.2.1 Single axis rotation

Similarly to the translation on axes, the single axis rotation was evaluated on a dataset of samples that were obtained by evaluating the rotation experienced between successive observations. Each egomotion rotation was evaluated based on the axes of movement. If it was found that the predominant axis accounted for more than 85% of the relative egomotion, the sample was used for that particular axis performance calculations.

For a rotation on the lower end of the spectrum, the RPE error does not depend too much on the axis of rotation, but a much more pronounced effect can be seen for larger rotations. The effect of roll on the RPE is roughly 20 - 30% less than that of the pitch and yaw. This is because of the fact that the level of scene coherence does not scale on all three axes the same. The level of scene coherence with regards to roll is much higher, as the actual scene that is being observed does not change, but rather the rotation of the window observing the scene changed. Even though this still has an effect on the performance, it is not nearly as much of an effect as pitch and yaw. Thus, the pitch and yaw are responsible for actual scene changes, while the roll affects the windows through which the scene is observed.

On average an increase in the roll velocity of the sensor increases the error rate linearly by 25%, while

Table 4.17. The rotational RPE error is shown across a range of different rotations experienced between two observations. If the rotation experienced was predominantly on one of the rotation axes the sample was used to compute this data.

Translation	Yaw		Pitch		Roll	
	Frames	RPE [deg]	Frames	RPE [deg]	Frames	RPE [deg]
0.1- 0.2 deg	831	0.115 ± 0.057	641	0.114 ± 0.056	715	0.095 ± 0.044
0.2- 0.4 deg	861	0.160 ± 0.075	868	0.150 ± 0.078	675	0.160 ± 0.081
0.4- 0.6 deg	718	0.195 ± 0.096	783	0.190 ± 0.093	756	0.142 ± 0.069
0.6- 0.8 deg	581	0.226 ± 0.107	614	0.220 ± 0.106	437	0.162 ± 0.081
0.8- 1.0 deg	461	0.260 ± 0.125	513	0.251 ± 0.122	381	0.186 ± 0.086
1.0- 1.5 deg	472	0.391 ± 0.192	345	0.408 ± 0.187	408	0.275 ± 0.134
1.5- 1.9 deg	402	0.698 ± 0.327	307	0.669 ± 0.354	346	0.495 ± 0.242

the rate is closer to 30% for the pitch and yaw. For angles above 2°, or 60 deg/sec, the error rate starts to increase exponentially, as the rotation experienced changes the scene significantly for each observation.

Even if it is not shown in the data, it can be surmised that the error rate for the roll axis will reach a ceiling and then stay constant. This is because the roll does not change the scene being perceived, except for the outer portions of the image. The scene represented by the circular area in the center will stay the same, except that it will be rotated. Thus, where the pitch and yaw will eventually move the scene being observed completely away, the roll will only increase in error up to a point, and then stays constant.

4.3.2.2 Multi-axis rotation

Theoretically while it is possible to contain the rotation experienced by the sensor to a single axis, typically by mounting it to a ground-based robot, some, if not most, applications will contain rotation on all three axes at the same time. To compute the RPE error for this situation, observation pairs were evaluated based only on the translation experienced. Thus, an egomotion was added based only on if

the translation experienced was below the 3 mm threshold. This, once again, yielded a much larger dataset of observations. The results are shown in Table 4.18.

Table 4.18. The translation and rotation RPE error values over a range of rotations. The rotations experienced for the samples used to calculate this data can be any combination of rotation axes. Samples were only used if their translation was below the threshold.

Rotation deg	Frames	RPE [mm] mean ± std	RPE [mm] per 1°	RPE [deg] mean ± std	RPE [deg] per 1°
0.1- 0.2 deg	2,649	2.09 ± 1.24	13.94	0.104 ± 0.051	0.696
0.2- 0.4 deg	2,631	2.55 ± 1.51	8.5	0.138 ± 0.068	0.461
0.4- 0.6 deg	2,437	3.22 ± 1.90	6.45	0.178 ± 0.086	0.356
0.6- 0.8 deg	1,937	3.67 ± 2.17	5.24	0.204 ± 0.100	0.292
0.8- 1.0 deg	1,831	4.39 ± 2.59	4.88	0.229 ± 0.112	0.255
1.0- 1.5 deg	1,437	6.72 ± 3.94	5.38	0.354 ± 0.172	0.283
1.5- 1.9 deg	760	10.09 ± 5.96	5.94	0.629 ± 0.306	0.37

Similar to the effects experienced under translation only, the proportional error rate experienced for small rotations is very high compared to the larger orientation changes. The effective RPE for rotations under 0.2° is 13.79 mm per degree for translation, and 0.701° per degree. This rate is reduced drastically up to around 1.5° rotation per observation, and then steadily increases again. From the data it can be seen that the rotation experienced will drastically impact both the rotational and translational accuracy between two successive frames.

4.3.2.3 Rotation velocity only

Even though it is much more likely that a SLAM implementation might be used in a system that only has rotation, and no translation, it is still not the everyday purpose of SLAM. If the system is only prone to rotation and does not have any translations greater than 3 mm between successive frames, the effective RPE can be calculated based on the rotational velocity and frame rate. For this dissertation an average rotational velocity of between 6 deg/s and 48 deg/s will be evaluated. Table 4.19 shows these results.

Table 4.19. For a system that has translation only, this table can be used to estimate the RPE error for both the rotation and translation. A range of translation velocities versus three key frame rates are given.

Rotation	30 *fps*		20 *fps*		10 *fps*	
velocity [deg/s]	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]	RPE[mm]	RPE[deg]
5.00	2.09	0.104	2.4	0.13	3.11	0.171
10.00	2.71	0.147	3.11	0.171	4.58	0.237
15.00	3.11	0.171	3.85	0.215	8.03	0.471
20.00	3.49	0.194	4.58	0.237	12.4	0.854
25.00	4.21	0.222	6.59	0.344	19.14	1.548
30.00	4.58	0.237	8.03	0.471	29.55	2.806
35.00	6.14	0.301	10.2	0.671	45.62	5.085
40.00	7.54	0.394	12.4	0.854	70.43	9.214

4.3.3 Spatial velocity

Considering the typically purpose of SLAM, the platform that will be used will be a land or air based vehicle that can move in any 6 DoF at any time. Thus, the rotation and translation movement will not be separated, but will occur at the same time. Hence, to give an accurate estimation of the RPE error based on the spatial velocity, the combination of rotation and translation must be evaluated. To evaluate the RPE error of the system over both the rotation and translation, all the frames in the datasets and subsampled datasets are used in an approach that is similar to that which was used in the previous sections. For each individual egomotion the rotation and translation is evaluated, and each sample's egomotion and RPE error is saved into a 2D database that corresponds to the given rotation and translation. This 2D database consists of a 9×9 grid, where the first element corresponds to a translation between 1 mm and 2mm, and a rotation between 0.1° and 0.2°. The final element consists of a translation between 16 mm and 18 mm, and a rotation between 1.6° and 1.8°. Using this 2D table, an accurate RPE estimate can be found for each combination of rotation and translation in the 2D table.

Table 4.20 on page 145 shows the effective RPE that was observed for each level of translation and rotation. For each combination the translational RPE and rotational RPE are given.

From the data it can be seen that the rotation experienced by the sensor has a much larger impact on the overall accuracy than the translation does, when comparing the effect of a 1 mm translation against 1° of rotation. For a comparison of 1 cm translation against 1° of rotation, the size of the error induced is much closer together.

During translation, the translational error introduced was roughly equivalent to 45% of the translation experienced, and the rotation error was 0.2° on average for every 1 cm transversed. In comparison, the rotation error introduced for rotation was only 20% of the rotation undergone on average, but the translation error was 4 mm for every 1° rotated. Thus, the effect of rotation had a substantially larger effect on the translation error than the translation had on the rotation error. From this it can be deduced that a system whose spatial velocity is based predominantly on translation rather than rotation, will have a significantly better performance than one that rotates more often than moving on the translational axis.

Another factor that presented itself in the results is the effect of an increased rotation versus an increased translation. Thus, as the size of the transition or rotation between each frame increased, the effect that each had on their respective RPE differed. For the translation, the RPE versus the translation stays consistent. At 18 mm the RPE error percentage was still roughly the same as at 10 mm. Conversely the RPE error per degree experienced at 1.8° mas notably larger than at 1°. For a rotation of 1.8° the rotation error was just over 30% of rotation experienced, and the translation error was at 5.2 mm for every 1° rotated.

Combining the effects of translation and rotation, the RPE error metric rose exponentially as the effect of both compounded onto the effective error. For a translation of 18 mm and 1.8° rotation per observation, an average translation error of 36.10 mm was found, and a rotation error over 1.8°; thus, it has over a 200% error margin for the translation and a 100% error margin for the rotation. If a SLAM implementation was to operate on a datastream that had an average error rate that large, it would be nearly impossible to create a consistent global trajectory, and even the normal front-end could struggle to maintain accuracy.

4.4 QUADCOPTER PLATFORM

The quadcopter platform is incapable of capturing its ground-truth readings, and thus it cannot be evaluated based on an error metric. From the relevant results returned by the SLAM implementation the average translation and rotation per frame can be estimated, as shown in Table 4.21. The performance expectations for higher frame rates are also shown.

Table 4.21. Estimated performance of the quadcopter with regards to the RPE error between successive frames. For the higher frame rates the RPE error is estimated based on what the effective spatial velocity would be at that frame rate.

Frame rate	F2F translation	F2F rotation	RPE [mm]	RPE [deg]
fps	*mm*	*deg*	*mean* $\pm$ *std*	*mean* $\pm$ *std*
6	24.61	2.18	55.65 ± 17.071	2.75 ± 0.917
8	18.46	1.64	40.34 ± 13.377	1.93 ± 0.658
10	14.76	1.31	21.63 ± 7.132	1.04 ± 0.370
15	9.84	0.87	9.05 ± 3.245	0.43 ± 0.140
20	7.38	0.65	6.07 ± 2.348	0.31 ± 0.094
30	4.92	0.44	4.26 ± 2.031	0.22 ± 0.084

Other than the estimated performance that the SLAM implementation returns, only a general discussion can be made on the performance of the quadcopter.

4.4.1 Performance

The quadcopter is able to achieve its goal of providing a robust testing environment for a SLAM implementation. With regards to actually performing a 3D model scan of a given environment, the platform has two major areas that could be improved upon.

The first is the frame rate that the onboard development board is able to capture and deliver to the remote computer. This frame rate fluctuates between 6 and 8 Hz, but is not near the ideal 30 Hz

that the Kinect can deliver. This, however, does not prevent the platform from functioning, but only limits the spatial velocity it should operate on. The platform is capable of stabilising itself within the environment and is also able to move to designated locations. The quadcopter does oscillate around the given waypoint, but not enough to pose a major danger. This oscillation is, however, large enough to prevent the quadcopter from transversing through doorways, preventing it from scanning fully enclosed environments. The oscillation is a combination of the error incurred in the SLAM implementation and the relative weight of the quadcopter, thereby hindering smooth operation. In the given table the effective RPE is given for the potential full frame rate of the Kinect. Thus, if the Pandaboard could have delivered a full 30 fps, the system would have had a much lower average error rate.

The second issue with the quadcopter is the stabilisation within an enclosed area. Ideally the quadcopter should be completely stable, and it is possible for most small quadcopter to achieve this. Unfortunately because of the weight and size of the quadcopter, a large amount of thrust is required to stay in flight. This, in turn, creates large amounts of drafts in the enclosed area, that in turn make the quadcopter fairly unstable. The larger the area that the quadcopter operates in, the better the stabilisation is. For an average office or room size, it could not be guaranteed that the quadcopter will not jerk into a wall or object due to drafts pushing it around. For open areas the quadcopter operates very well, except that it is unable to operate in sunlight, limiting the potential open areas that can be used. This factor is the main reason the estimated RPE error is so high for the quadcopter itself.

Because of both of the challenges mentioned, the quadcopter system is not as stable as envisioned when it was created. The quadcopter has a fairly large frame-to-frame error rate when compared to datasets that run a full 30 fps, and thus the performance of the quadcopter was lacking.

4.4.2 Full system latency

A small delay is found in the communication between the remote terminal and the quadcopter, but it is small enough to still allow the quadcopter to stabilise within the environment. Table 4.22 shows the latency induced at each stage of the communication loop. It was found that the system had a much lower latency between the Pandaboard and the remote PC if only 6 fps were used. This is most likely due to the saturation of the wireless network at the full 8 fps, where a fully saturated connection has a much higher latency than one operating at roughly 85%.

Table 4.22. Full loop latency for the quadcopter system using 6 fps and 8 fps. The 6 fps system has a much lower latency due to the 8 fps system saturating the wireless network.

Process	Time 6 fps	Time 8 fps
Arduino connection	28 ms	28 ms
Pandaboard	68 ms	116 ms
Quad control application	4 ms	4 ms
SLAM implementation (SIFT)	35 ms	35 ms
Total	139 ms	187 ms

Because the Arduino constantly updates the quadcopter based on the position controller, the latency would depend on when the SLAM implementation returns the latest egomotion estimates. The worst case will be when it returns the results just as a command was being sent; thus, having to wait the full duration before the updated command can be sent. The worst case latency for the 6 fps system is 137 ms and 187 ms for 8 fps.

4.5 CHAPTER SUMMARY

This chapter presented the results obtained for this dissertation. A breakdown of the frame-to-frame error rate was given for each of the stages in the front-end and back-end of the RGB-D SLAM implementation. The ATE error was computed for eight datasets, and a comparison was made between this implementation and four other prominent implementations. The effect of rotation and translation was investigated with regards to the frame-to-frame error rate. It was found that the effective rotation speed has a larger impact than the effective translation speed. The performance of the quadcopter platform was estimated and discussed.

The final chapter will give a concluding discussion on the results obtained as well as indicate potential areas that can be improved on in the future.

Table 4.20. Relevant RPE error values for a range of rotations and translations that the sensor experienced across all the datasets. The RPE values were calculated with the back-end optimisation running, but is still a frame-by-frame reference.

Translation	1 - 2 mm	2 - 4 mm	4 - 6 mm	6 - 8 mm	8 - 10 mm	10 - 12 mm	12 - 14 mm	14 - 16 mm	16 - 18 mm
Rotation	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$	RPE $\frac{[mm]}{[deg]}$
0.1° - 0.2°	1.91 ± 1.16	2.50 ± 1.37	3.15 ± 1.45	3.72 ± 1.47	4.46 ± 1.64	5.45 ± 1.91	5.99 ± 2.04	7.41 ± 2.57	9.39 ± 3.26
	0.096 ± 0.047	0.127 ± 0.053	0.154 ± 0.053	0.178 ± 0.051	0.213 ± 0.064	0.248 ± 0.080	0.287 ± 0.094	0.325 ± 0.107	0.371 ± 0.122
0.2° - 0.4°	2.36 ± 1.42	2.99 ± 1.62	3.71 ± 1.68	4.35 ± 1.70	5.20 ± 1.88	6.33 ± 2.19	7.00 ± 2.35	8.66 ± 2.96	10.96 ± 3.75
	0.129 ± 0.064	0.161 ± 0.070	0.192 ± 0.068	0.220 ± 0.066	0.261 ± 0.082	0.302 ± 0.101	0.348 ± 0.118	0.395 ± 0.134	0.450 ± 0.154
0.4° - 0.6°	3.01 ± 1.80	3.71 ± 2.00	4.54 ± 2.05	5.31 ± 2.06	6.33 ± 2.27	7.67 ± 2.63	8.52 ± 2.84	10.51 ± 3.56	13.27 ± 4.50
	0.168 ± 0.084	0.204 ± 0.088	0.241 ± 0.085	0.275 ± 0.082	0.324 ± 0.101	0.375 ± 0.124	0.432 ± 0.145	0.491 ± 0.166	0.561 ± 0.191
0.6° - 0.8°	3.43 ± 2.07	4.21 ± 2.29	5.15 ± 2.35	6.07 ± 2.39	7.25 ± 2.64	8.82 ± 3.07	9.87 ± 3.34	12.20 ± 4.20	15.42 ± 5.31
	0.192 ± 0.096	0.233 ± 0.100	0.276 ± 0.097	0.317 ± 0.094	0.375 ± 0.116	0.435 ± 0.144	0.504 ± 0.168	0.576 ± 0.193	0.661 ± 0.223
0.8° - 1.0°	4.12 ± 2.47	5.00 ± 2.71	6.10 ± 2.77	7.19 ± 2.81	8.58 ± 3.10	10.40 ± 3.60	11.69 ± 3.93	14.39 ± 4.91	18.11 ± 6.19
	0.219 ± 0.110	0.264 ± 0.115	0.315 ± 0.112	0.364 ± 0.109	0.431 ± 0.136	0.503 ± 0.169	0.584 ± 0.199	0.670 ± 0.229	0.771 ± 0.265
1.0° - 1.2°	5.41 ± 3.24	6.46 ± 3.49	7.79 ± 3.53	9.14 ± 3.56	10.81 ± 3.89	12.98 ± 4.47	14.59 ± 4.88	17.74 ± 6.03	22.04 ± 7.51
	0.264 ± 0.133	0.317 ± 0.137	0.377 ± 0.134	0.437 ± 0.131	0.517 ± 0.163	0.602 ± 0.202	0.699 ± 0.237	0.801 ± 0.273	0.921 ± 0.316
1.2° - 1.4°	6.69 ± 4.02	7.93 ± 4.30	9.55 ± 4.34	11.19 ± 4.38	13.20 ± 4.77	15.74 ± 5.45	17.71 ± 5.96	21.37 ± 7.30	26.29 ± 8.99
	0.351 ± 0.176	0.415 ± 0.179	0.492 ± 0.174	0.569 ± 0.170	0.668 ± 0.210	0.773 ± 0.258	0.891 ± 0.301	1.016 ± 0.345	1.160 ± 0.396
1.4° - 1.6°	8.12 ± 4.86	9.61 ± 5.20	11.57 ± 5.24	13.57 ± 5.29	15.97 ± 5.75	18.96 ± 6.54	21.36 ± 7.15	25.57 ± 8.70	31.18 ± 10.62
	0.478 ± 0.237	0.560 ± 0.240	0.662 ± 0.231	0.764 ± 0.225	0.890 ± 0.275	1.023 ± 0.336	1.171 ± 0.390	1.327 ± 0.444	1.505 ± 0.507
1.6° - 1.8°	9.41 ± 5.72	11.16 ± 6.13	13.48 ± 6.22	15.87 ± 6.33	18.69 ± 6.90	22.15 ± 7.83	24.99 ± 8.59	29.79 ± 10.39	36.10 ± 12.61
	0.605 ± 0.302	0.708 ± 0.305	0.838 ± 0.296	0.970 ± 0.288	1.127 ± 0.352	1.292 ± 0.429	1.474 ± 0.495	1.665 ± 0.562	1.880 ± 0.639

CHAPTER 5 CONCLUSION

5.1 CONCLUSION

In this work a large scope was envisioned to not only showcase the utility of SLAM, but also evaluate the relative pose error associated with each time instance and how that error corresponds with the spatial velocity. The concept of SLAM has been around for quite a few years, and a large amount of research has already been performed to continuously improve on previous methods. RGB-D SLAM is one of the more recent methods for solving the SLAM problem, and has become widespread since the advent of the RGB-D sensor.

This dissertation aimed to recreate a landmark RGB-D SLAM implementation and alter it for real-time use on a quadcopter platform. A fully working quadcopter system was created that is able to provide the aforementioned RGB-D SLAM implementation with a datastream that would allow it to build a systematic 3D map of the environment, and localise the quadcopter within it. Additionally, the SLAM implementation is able to accept publicly available datasets that allow it to compute the relevant performance of this implementation against other RGB-D implementations.

The quadcopter platform that was built for this project was not the ideal solution envisioned, but it meets the required criteria. A maximum frame rate of 8 RGB-D images can be captured each second, and this data can be used as the datastream for the SLAM implementation. The quadcopter is capable of self-stabilising flight, given the correct conditions. In an enclosed area with significant back-draft from the quadcopter, the system struggles with stabilisation, but is still able to maintain flight with continuous corrections to the position.

The SLAM implementation is capable of operating at near 30 fps; thus, it is able to operate in real-time

with the quadcopter system. A full loop latency of 187 ms (139 ms - 6 fps) was achieved between the quadcopter and the remote processing computer. A user interface allows the user to specify 2D positions in the environment that the quadcopter will attempt to assume, using a series of PID controllers to move the quadcopter to the location. On average the quadcopter has a translation velocity of 0.147 m/s and a rotation velocity of 49.2 deg/s. A frame-to-frame translational RPE error of 40.34 mm and rotational RPE error of 1.93° were estimated for the quadcopter system.

A full evaluation was performed on the SLAM implementation, using similar error metrics as other works in this field. Three feature detector and descriptor pairs were implemented and a direct evaluation was done among them. SIFT outperformed both SURF and ORB, obtaining an average accuracy that is 30% better than SURF and nearly doubling ORB's performance. These results are inline with those presented by other works, although ORB underperformed when comparing with some of the most recent studies. All three feature detectors were able to operate at near 30 fps, but a CUDA implementation was used for SIFT and SURF, where ORB was implemented using a standard CPU approach. The system's average ATE error was computed over a range of datasets and compared to four prominent RGB-D SLAM implementations. This dissertation's implementation was on par with a similar implementation of 2012, while the other three had much better performance. This was to be expected, as state-of-the-art techniques were used that improved on the front-end and back-end estimation. This, however, does not invalidate any results obtained, as the results obtained for the spatial velocity should scale linearly with any performance increase.

By utilising a method of downsampling, the amount of data available at varying levels of spatial velocity was increased. Using the public datasets with their subsampled versions, a classification system was created that measures the real-world egomotion that the sensor undergoes between each successive frame. Each egomotion is grouped, based on its relative rotation and translation. These groups of image pairs are then used to estimate the effect that translation and rotation have on the receptive RPE error. It was found that the rotation has a larger effect on the overall RPE error, as the level of scene coherence changes much more as the sensor rotates, than it does while translating. During translation, the translational error introduced was roughly equivalent to 45% of the translation experienced, and the rotation error was 0.2° on average for every 1 cm transverses. In comparison, the rotation error introduced for rotation was only 20% of the rotation undergone on average, but the translation error was 4 mm for every 1° rotated. Thus, the rotation has a more profound effect on the translation error than the translation has on the rotation error. This would suggest that a system with a

high average translation but a low rotation, will perform better than a system with a high rotation and low translation. It was also observed that the z-axis and roll-axis of the translation and rotation had a much lower effect on the error rate. This is because they represent the scale and scene rotation, and SIFT is notorious for being able to handle large levels of scale and rotation within an image.

5.2 FUTURE IMPROVEMENTS

Multiple areas can be improved upon for this dissertation. The largest improvement can be made on the quadcopter system, where the relative weight of the quadcopter, and the associated thrust needed for flight, make it hazardous in enclosed areas. Ideally the quadcopter's weight should be scaled down to roughly 1.1 kg. This can be accomplished by removing the secondary battery and using a smaller development board to connect to the Kinect. The maximum frame rate that the quadcopter is capable of should also be increased. If the quadcopter is capable of a full 30 fps, then it would cut down on the effective RPE error by as much as 80%, if not more. Unfortunately this is not possible with the Pandaboard, as it is at maximum performance with regards to the wireless capabilities and USB bus speeds.

Additionally, better techniques can be implemented for the front and back-end of the RGB-D SLAM implementation. The front-end can be improved by adding keyframe ICP to the RANSAC process, thus optimising the transformation before calculating the number of inliers. Both the front-end and back-end can be improved by adopting the environmental measuremnt model proposed by Felix *et al.* [15]. Because a third party graph optimisation process was used for this dissertation, it does leave that area open for improvement with regards to processing time. Because the $\mathbf{g^2o}$ operates on a single CPU core, CUDA can be used to streamline the process. Because the graph optimisation process is so expensive to compute on the CPU, the back-end does not operate on the optimal number of frames. By improving the graph optimisation, it is possible to have the back-end operate at a much higher speed.

REFERENCES

[1] H. H. Coppejans and H. C. Myburgh, "A primer on autonomous aerial vehicle design," *Sensors*, vol. 15, no. 12, pp. 30 033–30 061, 2015.

[2] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE, 2005, pp. 2247–2252.

[3] B.-Y. Lee, D.-W. Yoo, and M.-J. Tahk, "Performance comparison of three different types of attitude control systems of the quad-rotor uav to perform flip maneuver," *International Journal Aeronautical and Space Sciences*, vol. 14, no. 1, pp. 58–66, 2013.

[4] A. Ryan, M. Zennaro, A. Howell, R. Sengupta, and J. K. Hedrick, "An overview of emerging results in cooperative uav control," in *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, vol. 1. IEEE, 2004, pp. 602–607.

[5] J. P. How, B. BEHIHKE, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE control systems*, vol. 28, no. 2, pp. 51–64, 2008.

[6] N. Michael, J. Fink, and V. Kumar, "Cooperative manipulation and transportation with aerial robots," *Autonomous Robots*, vol. 30, no. 1, pp. 73–86, 2011.

[7] H. Durrant-Whyte, D. Rye, and E. Nebot, "Localization of autonomous guided vehicles," in *Robotics Research*. Springer, 1996, pp. 613–625.

[8] J.-Y. Wen and K. Kreutz-Delgado, "The attitude control problem," *IEEE Transactions on Automatic control*, vol. 36, no. 10, pp. 1148–1162, 1991.

[9] A. R. Jimenez, F. Seco, C. Prieto, and J. Guevara, "A comparison of pedestrian dead-reckoning algorithms using a low-cost mems imu," in *Intelligent Signal Processing, 2009. WISP 2009. IEEE International Symposium on*. IEEE, 2009, pp. 37–42.

[10] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (slam): Part ii," *IEEE Robotics & Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[11] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots*, vol. 5, no. 3-4, pp. 253–271, 1998.

[12] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor mav," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4557–4564.

[13] S. Weiss, D. Scaramuzza, and R. Siegwart, "Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments," *Journal of Field Robotics*, vol. 28, no. 6, pp. 854–874, 2011.

[14] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[15] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-d mapping with an rgb-d camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[16] A. Kushleyev, D. Mellinger, C. Powers, and V. Kumar, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, vol. 35, no. 4, pp. 287–300, 2013.

[17] G. Flores, S. Zhou, R. Lozano, and P. Castillo, "A vision and gps-based real-time trajectory planning for a mav in unknown and low-sunlight environments," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1-2, pp. 59–67, 2014.

[18] C. N. Taylor, M. J. Veth, J. F. Raquet, and M. M. Miller, "Comparison of two image and inertial sensor fusion techniques for navigation in unmapped environments," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, no. 2, pp. 946–958, 2011.

[19] O. El Hamzaoui, J. C. Espino, and B. Steux, "Autonomous navigation and mapping with coreslam," in *Recent Advances in Robotics and Automation*. Springer, 2013, pp. 91–101.

[20] R. Valencia and J. Andrade-Cetto, "Active pose slam," in *Mapping, Planning and Exploration with Pose SLAM*. Springer, 2018, pp. 89–108.

[21] R. C. Smith and P. Cheeseman, "On the representation and estimation of spatial uncertainty," *The international journal of Robotics Research*, vol. 5, no. 4, pp. 56–68, 1986.

[22] H. F. Durrant-Whyte, "Uncertain geometry in robotics," *IEEE Journal on Robotics and Automation*, vol. 4, no. 1, pp. 23–31, 1988.

[23] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot, "Consistency of the ekf-slam algorithm," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 3562–3568.

[24] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," in *Aaai/iaai*, 2002, pp. 593–598.

[25] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[26] S. Thrun, "Learning occupancy grid maps with forward sensor models," *Autonomous robots*, vol. 15, no. 2, pp. 111–127, 2003.

[27] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 2262–2269.

[28] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[29] J. B. Kuipers *et al.*, *Quaternions and rotation sequences*. Princeton university press Princeton, 1999, vol. 66.

[30] A. J. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *null*. IEEE, 2003, p. 1403.

[31] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular slam," *Robotics: Science and Systems VI*, vol. 2, 2010.

[32] N. K. Logothetis and D. L. Sheinberg, "Visual object recognition," *Annual review of neuroscience*, vol. 19, no. 1, pp. 577–621, 1996.

[33] S. Ahn, M. Choi, J. Choi, and W. K. Chung, "Data association using visual object recognition for ekf-slam in home environment," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 2588–2594.

[34] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, 2012.

[35] P. Piniés, T. Lupton, S. Sukkarieh, and J. D. Tardós, "Inertial aiding of inverse depth slam using a monocular camera," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2797–2802.

[36] P. J. Besl, N. D. McKay *et al.*, "A method for registration of 3-d shapes," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[37] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.

[38] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, "Robust real-time visual odometry for dense rgb-d mapping," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 5724–5731.

[39] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake, "A robust rgb-d slam algorithm," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1714–1719.

[40] C. Ascher, C. Kessler, M. Wankerl, and G. Trommer, "Dual imu indoor navigation with particle filter based map-matching on a smartphone," in *Indoor positioning and indoor navigation (IPIN), 2010 international conference on*. IEEE, 2010, pp. 1–5.

[41] D. Droeschel, S. May, D. Holz, P.-G. Plöger, and S. Behnke, "Robust ego-motion estimation with tof cameras." in *ECMR*, 2009, pp. 187–192.

[42] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp." in *Robotics: science and systems*, vol. 2, no. 4, 2009, p. 435.

[43] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust rgb-d object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 681–687.

[44] R. Arandjelović and A. Zisserman, "Three things everyone should know to improve object retrieval," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 2911–2918.

[45] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 6, pp. 756–770, 2004.

[46] O. Chum, J. Matas, and J. Kittler, "Locally optimized ransac," *Pattern recognition*, pp. 236–243, 2003.

[47] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[48] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell *et al.*, "Detailed real-time urban 3d reconstruction from video," *International Journal of Computer Vision*, vol. 78, no. 2, pp. 143–167, 2008.

[49] K. Konolige and M. Agrawal, "Frameslam: From bundle adjustment to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1066–1077, 2008.

[50] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 2100–2106.

[51] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustmentâĂŤa modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[52] M. I. Lourakis and A. A. Argyros, "Sba: A software package for generic sparse bundle adjustment," *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, p. 2, 2009.

[53] K. Konolige and W. Garage, "Sparse sparse bundle adjustment." in *BMVC*, vol. 10, 2010, pp. 102–1.

[54] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3607–3613.

[55] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[56] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[57] J. Bauer, N. Sünderhauf, and P. Protzel, "Comparing several implementations of two recently published feature detectors," *IFAC Proceedings Volumes*, vol. 40, no. 15, pp. 143–148, 2007.

[58] T. Lindeberg, "Image matching using generalized scale-space interest points," *Journal of Mathematical Imaging and Vision*, vol. 52, no. 1, pp. 3–36, 2015.

[59] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision–ECCV 2006*, pp. 430–443, 2006.

[60] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *Digital Signal Processing (DSP), 2013 18th International Conference on*. IEEE, 2013, pp. 1–7.

[61] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE, 2011, pp. 2564–2571.

[62] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[63] S. Filipe and L. A. Alexandre, "A comparative evaluation of 3d keypoint detectors in a rgb-d object dataset," in *Computer Vision Theory and Applications (VISAPP), 2014 International Conference on*, vol. 1. IEEE, 2014, pp. 476–483.

[64] F. Tombari, "Keypoints and features," in *CGLibs Conference in Pisa*, 2013, pp. 303–312.

[65] F. Tombari, S. Salti, and L. Di Stefano, "Performance evaluation of 3d keypoint detectors," *International Journal of Computer Vision*, vol. 102, no. 1-3, pp. 198–220, 2013.

[66] R. Hänsch, T. Weber, and O. Hellwich, "Comparison of 3d interest point detectors and descriptors for point cloud fusion," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 3, p. 57, 2014.

[67] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3212–3217.

[68] S. Salti, F. Tombari, and L. Di Stefano, "Shot: Unique signatures of histograms for surface and texture description," *Computer Vision and Image Understanding*, vol. 125, pp. 251–264, 2014.

[69] G. Arbeiter, S. Fuchs, R. Bormann, J. Fischer, and A. Verl, "Evaluation of 3d feature descriptors for classification of surface geometries in point clouds," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1644–1650.

[70] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 13, no. 4, pp. 376–380, 1991.

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

[72] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1691–1696.

[73] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, 2017.