Using Regions of Interest to Track Landmarks for RGBD Simultaneous Localisation and Mapping

Presented by:
Jatin I. Harribhai

Prepared for:
A/Prof F. Nicolls and R. Verrinder
Department of Electrical Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Masters of Science degree in Electrical Engineering.

October 15, 2019

Declaration

1. I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:........................... Signed by candidate

J. I. Harribhai

Date:.........14/10/2019.........

Acknowledgments

I would like to thank my guru H.D.H Pramukh Swami Maharaj for giving me the guidance and support throughout my whole thesis.

I would also like to thank my parents for always having faith in me and not letting me give up on this tough journey. I am truly blessed for having amazing parents who have sacrificed everything to give me this opportunity. Thank you for everything Mom and Dad.

Also not forgetting my friends who have encouraged me and kept my spirits up throughout this long journey. I would like to acknowledge Himal Patel, Josh Siame, Tumelo Gabaraane, Prashila Patel and Keerathi Patel. Thanks guys.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at are those of the author and are not necessarily to be attributed to the NRF.

Abstract

The simultaneous localisation and mapping (SLAM) algorithm have been widely used for autonomous navigation of robots. A type of visual SLAM that is popular among the researchers is RGBD SLAM. However processing immense image data to identify and track landmarks in RGBD SLAM can be computationally expensive for smaller robots. This dissertation presents an alternate method to reduce the computational time. The proposed algorithm extracts features from a region of interest (ROI) to track landmarks for RGBD SLAM. This strategy is compared to the traditional method of extracting features from an entire image. The ROI algorithm is implemented via a pre-processing algorithm, which is then integrated into the RGBD SLAM framework.

The pre-processing pipeline implements image processing algorithms in three stages to process the data. Stage one uses a ROI algorithm to detect ROIs in an image. For visual SLAM such as RGBD SLAM, objects that are highly detailed are used as landmarks. Hence the ROI algorithm is designed to detect ROIs containing highly detailed objects. Stage two extracts features from the image and stage three uses feature matching algorithms to re-identify a ROI. Once a ROI has been successfully re-identified, it is stored and categorised as a landmark for RGBD SLAM. Scale invariant feature transform (SIFT), speeded up robust features (SURF) and orientated FAST and rotated BRIEF (ORB) are three feature extraction algorithms that are used in stage two.

The outcomes from this study revealed that the pipeline was able to successfully create a database of landmarks which can be re-identified in subsequent frames. In addition, the results showed that when the pipeline is configured such that SURF features are used with a bigger ROI, RGBD SLAM produced more accurate results in determining the position of the robot compared to the traditional method of extracting features from an entire image. However, this strategy requires more computational time. The findings further revealed that this strategy still out performs the traditional method when the number of features extracted is reduced. This indicated that this strategy performs more robustly compared to the traditional method in environments that can contain few features. The method presented in this study did not improve the computational time of RGBD SLAM but did improve the accuracy in localizing the robot.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Autonomous robotic navigation is an important field of research with several navigation algorithms having been developed [1]. In recent years the most popular framework is Simultaneous Localisation and Mapping (SLAM) due to the two unique functions it offers: (a) Its ability to construct a map and (b) that it can simultaneously localise the robot in that map. SLAM accomplishes these functions by making use of sensors on board a robot which gather data from the environment. The captured data are used to create landmarks of the environment. Landmarks are sets of recognizable objects that can be identified later on to improve the accuracy in map building and in localization of the robot [1, 2].

There are various types of sensors that are used to capture data for SLAM such as LIDAR, sonar and image based sensors. To build a detailed map of the environment, researchers have integrated vision based sensors with SLAM. This combination is known as visual SLAM. Visual SLAM can offer 3D point cloud data of the environment to build intricate maps for the robot to navigate safely. In visual SLAM highly detailed objects are often used as landmarks as they are unique and recognizable later on. To identify landmarks in visual SLAM feature extractors are used [1].

1.2 Problem Statement

Due to the immense quantity of image data, visual SLAM requires more computational time and data storage. This can be a computational burden on the robot and is a problem for smaller robots with limited processing power and resources. One type of visual SLAM called RGBD SLAM requires even more computational resources as the image data also contains depth data [3, 4]. However, a problem with RGBD SLAM is that it requires substantial computational time and data storage to process the image and depth data. The way RGBD SLAM processes image data needs to be optimized and improved such that the computational burden is reduced for smaller robots while ensuring the robot can still navigate accurately in the environment.

The traditional method used by RGBD SLAM to process image data is to extract a distribution of features from the entire image frame that can be used to identify and track landmarks in the environment. An alternative method that has been used in previous studies made use of regions of interest (ROIs) to select a sub-image whereby features were only extracted from the ROI. A ROI is designed to select a sub-image data that contains highly detailed objects that could be used as landmarks for SLAM thereby optimizing the search of uniquely identifiable landmarks in image data. By optimizing the search for landmarks the performance of SLAM would improve. This method was used by Frintop [5] and Aulinas et al. [6]. However, in those studies this method was only applied to the conventional visual SLAM that uses a normal RGB camera. Thus far this method has not been applied specifically to the RGBD SLAM framework to improve the processing of image data.

1.3 Research Outline

This dissertation presents an alternative solution of using ROIs to process image data to identify and track landmarks for RGBD SLAM to reduce the computational time. But for this method to achieve a reduced computational time a trade-off is made between the computational time and the performance of RGBD SLAM which is investigated in this study. The method of using ROIs is integrated into RGBD SLAM using a pre-processing pipeline and then this method is compared to the traditional approach.

The pre-processing pipeline is made of three stages where each stage uses an image processing algorithm. The first stage uses a ROI algorithm to detect ROIs in an image.

The ROI algorithm is designed to detect regions that contains highly detailed objects, as RGBD SLAM uses these objects as landmarks. The second stage makes use of feature extraction to extract features from the ROI and a subsequent frame. The last stage uses feature matching to re-identify a ROI in the subsequent frame. When a ROI is successfully re-identified, it is considered as a landmark and stored in a database.

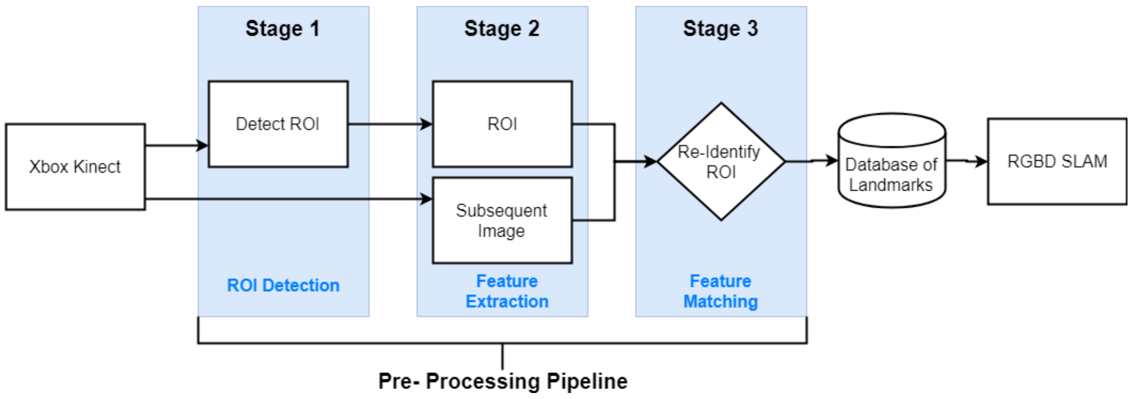


Figure 1.1: A diagram of the pre-processing pipeline used to process the image data to identify and track landmarks for RGBD SLAM.

In stage two of the pipeline an option of three popular feature extractors (scale invariant feature transform (SIFT), speeded up robust features (SURF) and orientated FAST and rotated BRIEF (ORB)) can be chosen. Furthermore, RGBD SLAM also utilizes these feature extractors for an easy integration of the pipeline into the RGBD SLAM framework. This dissertation also investigates which of the three feature extractors would best suit the pipeline to ensure that the computational time is minimised for RGBD SLAM.

In testing the pipeline with RGBD SLAM, three investigations were conducted to ensure that the pipeline functions correctly and is a sensible method for RGBD SLAM. The first investigation determines if the pipeline can identify and track landmarks correctly for RGBD SLAM. The second compares the pipeline with the traditional method of extracting features from an entire image frame, in terms of computational time and accuracy of determining the position of the robot. The last investigation compares the performances of both the methods on an image frame with a reduced feature set.

1.4 Scope

All tests were conducted on RGBD SLAM datasets consisting of a typical indoor office environment. The dataset was captured using the Microsoft Xbox Kinect as the depth camera and was specifically produced for RGBD SLAM. The dataset was constructed in the study conducted by Endres et al. and is used as the benchmark in research for testing performance of RGBD SLAM [7]. As there was no robot available to do an on-line implementation of the methods in this study, all implementations were conducted off-line. These were simulated on a desktop computer with Intel Core I7 processor and 8 GB of RAM. Only the RGB data was analysed.

Before the pipeline was embedded into the RGBD SLAM framework it was first tested in isolation to determine whether it could create a database of landmarks and re-identify landmarks later on. This test ensures that the pipeline is able to process the image data correctly to identify and track landmarks for RGBD SLAM.

Once the pipeline was integrated into the RGBD SLAM framework then two ROI versions of RGBD SLAM were created and compared to the original version of RGBD SLAM which uses the traditional method to process the image data. The two versions are called smaller ROI RGBD SLAM and bigger ROI RGBD SLAM. The smaller ROI version detects a smaller size ROI, and the bigger ROI version detects a bigger size ROI. The ROI versions of RGBD SLAM are tested and compared to the traditional version of RGBD SLAM to determine whether the strategy of using ROIs would improve the performance of RGBD SLAM in terms of accuracy in the robot's position and computational time. The number of features extracted were limited to 600 in these tests. The versions were further tested again reducing the number of features to 50. This test was used to determine if the strategy of ROIs can still track landmarks for RGBD SLAM with reduced features and again the performance of the ROI versions of RGBD SLAM is compared with the original version. These tests determine if the method of using the pipeline, which consists of using ROIs to process the image data, is a sensible method compared to the traditional method used in the original version of RGBD SLAM and if the pipeline reduces the computational time of RGBD SLAM.

1.5 Plan of Development

This dissertation consists of seven chapters. The second chapter presents a literature review of the SLAM framework with a detailed discussion of RGBD SLAM and the Xbox Kinect as a depth camera for RGBD SLAM. This is followed by a review of the image processing algorithms that can be used for the pre-processing pipeline. The chapter ends with a summary highlighting the algorithms used in the pre-processing pipeline to process the image data from the Xbox Kinect to identify and track landmarks for SLAM.

The third chapter provides a detailed background discussion of image processing theory used in the pre-processing pipeline. The fourth chapter discusses the implementation of each stage of the pipeline. A flowchart is presented of the pipeline algorithm that was developed for this study. Each stage of the pipeline is evaluated before the pipeline is used in tests. Additionally the chapter presents an overview of the RGBD SLAM framework and how the pre-processing pipeline is integrated into it.

The fifth chapter discusses the tests that were conducted, and results from each test are presented. The fifth chapter presents a discussion of the results presented in the previous chapter and draws conclusions. The last chapter provides recommendations and refinements that can be made to improve the performance and accuracy of the pre-processing pipeline for RGBD SLAM to reduce the computational time.

Chapter 2

Literature Review

Currently the method used in RGBD SLAM to process the image and depth data can be a computational burden to smaller robots with limited computational power. This dissertation presents an alternative method of processing image data using ROIs via a pre-processing pipeline to identify and track landmarks for RGBD SLAM to reduce the computational time. This method ensures that the features that are used to identify and track landmarks are only extracted from the ROI compared to the traditional method which extracts features from the entire image. Both methods are compared in this study. This chapter provides a background to this study and is made up of three sections.

Section 2.1 presents a detail discussion of the fundamental SLAM framework used in RGBD SLAM. A detailed RGBD SLAM framework is also shown to determine how the pipeline should be integrated into the framework. This section will also highlight the computational challenges of RGBD SLAM.

Section 2.2 provides a background discussion into depth cameras also known as RGBD cameras as these are the type of vision based sensors that are used for RGBD SLAM. The dataset used in this study was captured by a Xbox Kinect which is a RGBD camera and knowing how this sensor captures the image and depth data is crucial, such that the limiting factors of the data such as the resolution and field of view of the camera are known. This would ensure that the pipeline is developed appropriately to RGBD SLAM.

Section 2.3 provides an overview of the image processing algorithms used to develop the pre-processing pipeline. The first stage of the pipeline requires the detection of a ROI. From recent studies that have used ROIs in visual SLAM applications several options of detecting ROIs are presented. The second stage of the pipeline uses feature extractors and the three most popular feature extractors used in RGBD SLAM are as follows:

- SIFT
- SURF
- ORB.

A brief discussion of the feature extractors is presented. The last stage requires feature matching and a background discussion of the feature matching process is presented which uses fast library for approximate nearest neighbours (FLANN) and random sample consensus (RANSAC) to accurately match the features. This last stage is crucial in re-identifying a landmark for RGBD SLAM. .

2.1 Robotic Navigation

Navigation is a key field of the study of robotics. Currently robotic navigation makes use of the three most popular navigation algorithms, GPS, dead reckoning and SLAM. This dissertation uses SLAM for robotic navigation as it has been proven that it is the most robust form of navigation [1].

2.1.1 Simultaneous Localisation and Mapping (SLAM)

A robust approach used to improve localisation and mapping is SLAM [1]. This method can build a map while localising the robot within it.

SLAM offers a probabilistic approach to localising the robot and constructing a map of the environment. This is done by moving the robot one step at a time, where at each step data are collected via sensors. Data are captured either in the form of images, depth or velocity and direction through time. If the data captured are not depth, then it must be processed into range data before it can be fed into SLAM. At each step SLAM processes the data to create landmarks. Landmarks are uniquely identifiable objects in the scene that can be robustly recognized again. To relate the landmarks to each other, visual or point cloud data can be used to match the landmarks. As the landmarks are matched, a map of the environment is constructed. Hence when the landmarks are matched more than once, the accuracy in localizing the robot increases. Each movement (controlled action) from location to location is recorded as well. Thus based on a robot's movement and data (observations) at each step SLAM can build a map with landmarks located in their correct positions and is able to localise the robot within this map [1].

Due to SLAM's recent popularity, it has been improved to produce many variants, but there are three types that have become most popular. The first algorithm uses a Kalman filter and is known as the extended Kalman filter SLAM (EKF SLAM), while the second uses a Rao-Blackwellised particle filter and has been given the name FastSLAM [1, 2]. The third popular method is graph-based SLAM, which uses a nonlinear sparse optimization that produces an occupancy grid map as shown in Figure 2.1 [1]. Depending on the type of sensors used to extract data from the environment and the application of the robot, an appropriate SLAM algorithm must be chosen to obtain good results [1].

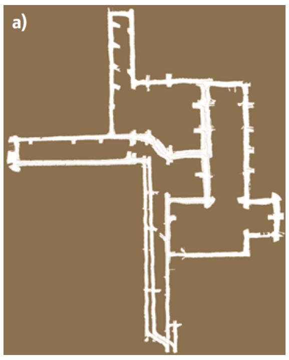


Figure 2.1: Occupancy grid built using graph-based SLAM algorithm used for robotic navigation taken from [1].

SLAM Framework

The SLAM framework described and presented in this section, largely is based on works by Durrant-Whyte and Bailey [2] and the Springer Handbook of Robotics [1].

The SLAM framework can be described with four vector states:

- $X = \{x_0, x_1, x_2, ...x_t\}$, locations of the robot at all times t
- $U = \{u_0, u_1, u_2, ...u_t\}$, control movements of the robot at all times t
- $M = \{m_0, m_1, m_2, ...m_t\}$, locations of landmarks at all i
- $Z = \{z_{i0}, z_{i1}, z_{i2}, ...z_{it}\}$, observations (data) of the i^{th} landmark at all times t.

Figure 2.2 demonstrates the relationship of the different states with each other. In Figure 2.2 only one landmark is shown relating it to the different states. However, if there are more landmarks then the same relation would be applied to them as well.

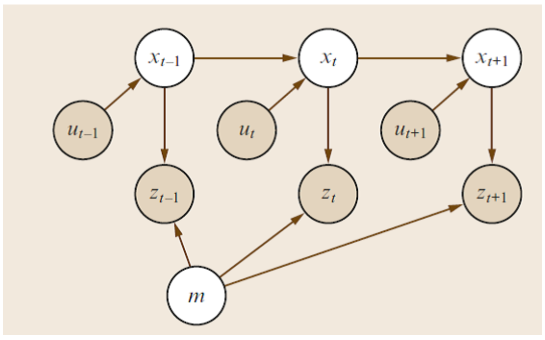


Figure 2.2: Basic SLAM diagram illustrating the relationship of the four states with each other taken from [1].

Using the x_0, U and Z vector states for all times t, the location of the robot x_t and location of the landmark m can be computed by the probability distribution in Equation (2.1):

$$P\left(x_t, m|Z_{0:t}, U_{0:t}, x_0\right). \tag{2.1}$$

The result from Equation (2.1) is the joint posterior density of the landmark and robot location x_{t+1}, where Bayes' theorem is applied. This equation is known as the measurement update model. In the beginning the location is not known, hence it is determined from Equation (2.2):

$$P\left(x_{t-1}, m|Z_{0:t-1}, U_{0:t-1}, x_0\right). \tag{2.2}$$

To apply Bayes theorem, the observation model needs to be known to describe the probability of observation z to location and landmark m from Equation (2.3):

$$P\left(z_t|x_t, m\right). \tag{2.3}$$

It also requires the motion model to describe the probability of the next location of the robot x_t from previous location x_{t-1} and the control movement u_t which moves the robot to that location. The motion model is determined by Equation (2.4):

$$P\left(x_t|x_{t-1}, u_t\right). \tag{2.4}$$

The time update model is shown in Equation (2.5):

$$P\left(x_t, m|Z_{0:t-1}, U_{0:t}, x_0\right) = \int P\left(x_t|u_t\right) P\left(x_{t-1}, m|Z_{0:t-1}, U_{0:t-1}\right) dx_{t-1}. \tag{2.5}$$

Using the above equation, the measurement update model (Equation (2.1)) can be determined using Equation (2.6):

$$P\left(x_t, m|Z_{0:t}, U_{0:t}, x_0\right) = \frac{P\left(z_t|x_t, m\right) P\left(x_t, m|Z_{0:t-1}, U_{0:t}, x_0\right)}{P\left(z_t|Z_{0:t-1}, U_{0:t}\right)}. \tag{2.6}$$

A map can now be constructed where the location of landmark m can be determined on its own relative to the position of the robot, observations made and the controlled movements at all times t using Equation (2.7). It is assumed from the initial location of the robot that the location at x_t is known or can be computed at all times t

$$P\left(m|Z_{0:t}, U_{0:t}, X_{0:t}\right). \tag{2.7}$$

Finally the location of the robot x_t can be determined relative to the location of the landmark m using the observations and controlled movements at all times t, which is determined by Equation (2.8). The locations of the landmarks are assumed to be known with certainty,

$$P\left(x_t|Z_{0:t}, U_{0:t}, m\right). \tag{2.8}$$

These equations form the fundamental backbone of SLAM. In Section 2.4 a summary is presented to demonstrate how the SLAM framework is used in this work.

In visual and RGBD SLAM, the observations (Z) of the landmarks are feature descriptors that are extracted from the captured images. In addition RGBD SLAM also embeds depth to the features.

2.1.2 Application of SLAM

Since SLAM is a well-known topic in the research community, several applications have been developed for land based robots [1]. Each application demonstrates the versatility and robustness of the method, increasing the robotic community's confidence in its use for autonomous navigation. More recently the use of SLAM-based algorithms has been expanded to underwater and aerial environments [1, 8].

Initially, SLAM was used with sonar or laser range finders to extract data for performing SLAM on a robot. In fact the majority of the robots using SLAM used LIDAR to extract data; it was only recently that the SLAM algorithm had been integrated with cameras, where images are used to extract data [1]. The use of cameras with SLAM is known as visual SLAM. Monocular SLAM is the use of one camera and binocular (stereo) SLAM is the use of two cameras [9, 10]. One of the main reasons for the development of visual SLAM is the advancement of the image processing algorithms that can improve the matching of landmarks to produce a more accurate map. The use of images has been beneficial to SLAM as it is able to produce 3D maps with high detail and accuracy [1].

Lemaire and Lacroix [9] were able to use EKF SLAM with panoramic images. The images were obtained using a panoramic lens on a standard camera. A comparison was made between SLAM and another method called visual motion estimation (VME), where it was found that SLAM produced more accurate location estimates. Davison and Reid [11] used real-time EKF SLAM on a humanoid robot to develop 3D SLAM. As the robot walked, it was able to map the landmarks into the 3D map.

RGBD cameras such as the Kinect device have had a major impact in the field of robotic navigation due to their ability to capture visual data with depth data at a high speed. Henry et al. [12] demonstrated this by using SLAM to create a 3D model of an indoor environment. Soon thereafter the Kinect was used in the SLAM algorithms, leading to a new form of SLAM called RGBD SLAM [3, 4]. Focus then fell on methods to extract data efficiently and accurately from the images captured by the Kinect for RGBD SLAM. Thus image processing techniques such as feature extraction algorithms to extract features were used to improve the performance of RGBD SLAM [7, 13–15]. To improve the performance of RGBD SLAM such that it can function in real time, Shen et al. used a bionic vision depth perception model with RGBD SLAM [16]. The study showed that this model did improve the real time performance of RGBD SLAM. A limitation to RGBD SLAM is the field of view due to the Kinect which led to inaccuracies and misalignment of the map being built of the environment. To produce a more accurate map, Yousif et al. integrated monocular SLAM using a GoPro with RGBD SLAM [17]. With the GoPro the field of view is wider capturing more data to allow for SLAM to produce a more accurate map. However, due to the increase in data being captured, the computational time to process data increases as well.

2.1.3 Overview of RGBD SLAM Architecture

RGBD SLAM is made of three modules: a front end, a back end and map representation. A schematic of the modules is shown in Figure 2.3. The front end processes the image data to determine the robot's pose relative to the landmarks observed. The back end optimizes the poses of the robot at different observations into a pose graph. The last module uses the pose graph to build a 3D map of the environment.

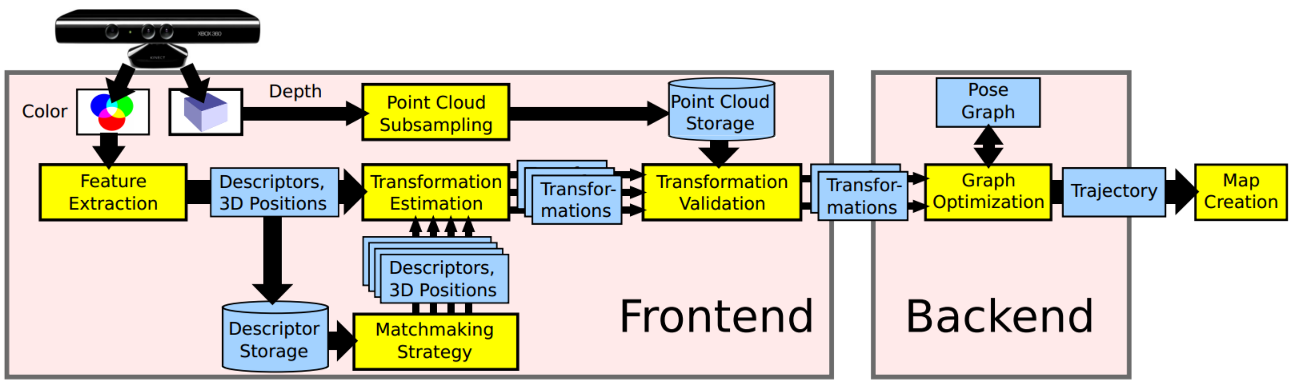


Figure 2.3: Schematic of RGBD SLAM architecture taken from [7].

Front end

The landmarks are observed from the images using feature detectors and matching. Each image captured is processed as a node and each node is compared to a previous node. If the node is matched to a prior node, the matched feature descriptors computed are projected to 3D using the depth measurements and a transformation is found between the nodes. This transformation is used to determine the robot's pose. If a node is not matched, it is added to the database of prior nodes.

This module allows for various types of feature detectors that can be used such as SIFT, SURF and ORB. For matching the features two methods are offered, namely the brute force method and FLANN.

Back end

The robot's poses calculated from the front end are used to build a pose graph. The pose graph is built and optimized using the **g^{2}o** framework [18]. The final result is an estimated trajectory of the robot. Detection of loop closures is also used to improve the accuracy of the trajectory.

Map Representation

Once the robot's trajectory is found the depth measurements are projected into a 3D global coordinate system. This is done by using a 3D occupancy grid to represent the environment. The framework that is used to compute the 3D occupancy grid is the OctoMap framework [7].

2.1.4 Computational Performance of RGBD SLAM

RGBD SLAM is the most recent advancement in solving the problem of using a low cost sensor like a RGBD camera to capture depth and image data to provide more accurate calculations of the robot's position.

One of the challenges is the fundamental SLAM problem of loop closure. This problem occurs when a landmark is revisited again and needs to detected as a landmark that has been previously seen. Once the landmark has been detected as a previously visited landmark then loop closure occurs, but detecting previously visited landmarks accurately is a challenge. In RGBD SLAM feature matching is used in this process to ensure that the landmark is correctly detected and recognized as a previous landmark. Xiangkui et al. improved this process by using Freak (Fast Retina Key-point) feature detector and principal component analysis (PCA) to match the feature detectors [15]. The study compared their proposed method with the conventional SIFT and ORB feature detectors and matching, and the results did show an improvement in computational time, but did not perform as well as ORB. Another method to improve the matching process is using the iterative closest point (ICP) algorithm and bundle adjustment (BA) [17]. This method has improved the process of matching features which resulted in accurate loop closures.

There has been significant research in improving loop closure for RGBD SLAM to reduce computational time and increase the accuracy in localizing the robot and map building. However, an aspect of RGBD SLAM that could improve its performance is the way it processes image data which is part of the front end of the framework and is another challenge for RGBD SLAM. Traditionally a feature extractor is used to extract a distribution of features over an entire image frame to identify and track landmarks. In the past a method used in visual SLAM to improve this process applied a ROI to the images where features are only extracted within the ROI. Additionally the ROI would be designed to detect uniquely identifiable objects that could be used as landmarks thereby optimizing the search for landmarks and improving loop closure. This method was introduced by Frintrop [5] and further extended to underwater applications by Aulinas et al. [6]. Both studies showed promising results in improving the performance of visual SLAM.

2.2 RGBD Cameras

RGBD cameras also known as depth cameras have been gaining interest in the research community [19]. Even though RGBD cameras were introduced into the market as a gaming accessory, they found many uses in research and continue to find many more [20]. Microsoft was the first company to release such a camera into the market as a gaming accessory to adopt a "controller-free" environment for their Xbox gaming console, and called it the Xbox Kinect. Microsoft was able to do this with the help of Primesense, an Israeli Company who were the originators of the design of the RGBD camera [21]. Soon after Microsoft had released a version of the Kinect for the PC platform, other companies followed such as ASUS with their Xtion Pro (seen in Figure 2.4) [19].

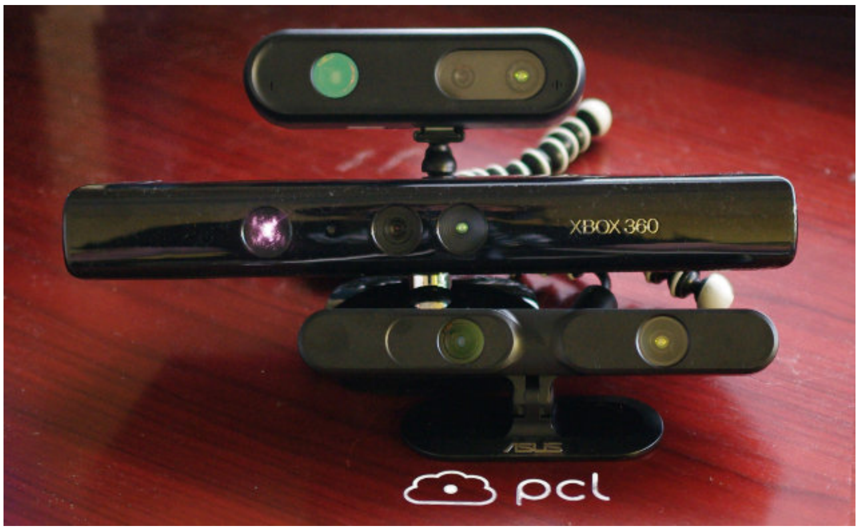


Figure 2.4: Most popular depth cameras used in research (from top to bottom: the Primesense reference design, Microsoft Kinect and Asus Xtion Pro) taken from [22].

The key aspect that sets RGBD cameras apart from other cameras is that they can capture colour and depth information generating a 3D point cloud of the environment. Each pixel from a RBGD camera has channels for red, green, blue and depth, hence the name RGBD. In robotic navigation with a RGBD camera, a robot will be able to capture its surroundings in terms of an image and can add depth to its surroundings without the use of any other sensors or post-processing. Due to this functionality of capturing depth for each pixel, RGBD cameras are able to capture more detail in low light environments compared to conventional cameras [23].

2.2.1 Xbox Kinect

This section provides details into the inner workings of the Kinect which consists of hardware and software. For this study the software used to code the Kinect takes into account the sensor calibration required and hence the Kinect does not need to be further calibrated.

Hardware

The Kinect is made of three fundamental hardware components: a VGA camera, an infrared sensor and an infrared projector [21]. The infared sensor and projector constitute a depth sensor. Figure 2.5 shows where these components are located, along with additional secondary components namely microphone array and motorized tilt. Internally the Kinect also contains a three-axis accelerometer.

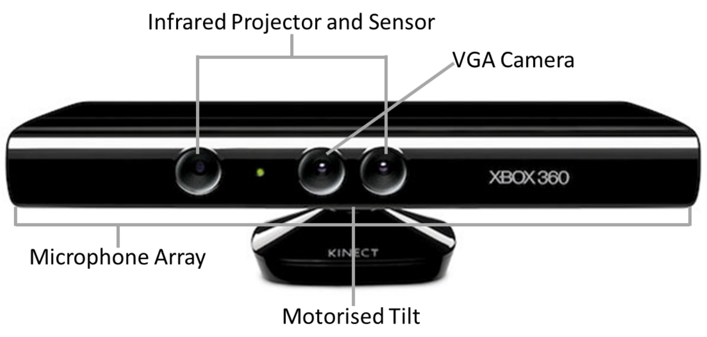


Figure 2.5: Front view of the Kinect showing the placement of the hardware components taken from [21].

Figure 2.6 shows the internal schematic of the Kinect and how the components interact with each other. The cameras used are a CMOS colour sensor and a CMOS infrared (depth) sensor. It also contains its own image processing microchip PS1080 provided by Primesense [24] to calculate the depth from the infrared image captured by the Kinect.

Figure 2.7 shows the field of view (in degrees) of the Kinect for both the RGB and depth camera, which physically restricts the Kinect in capturing a scene. The smaller the field of view, the less image data of the scene it can capture. Table 2.1 provides a summary of the technical specifications of the Kinect.

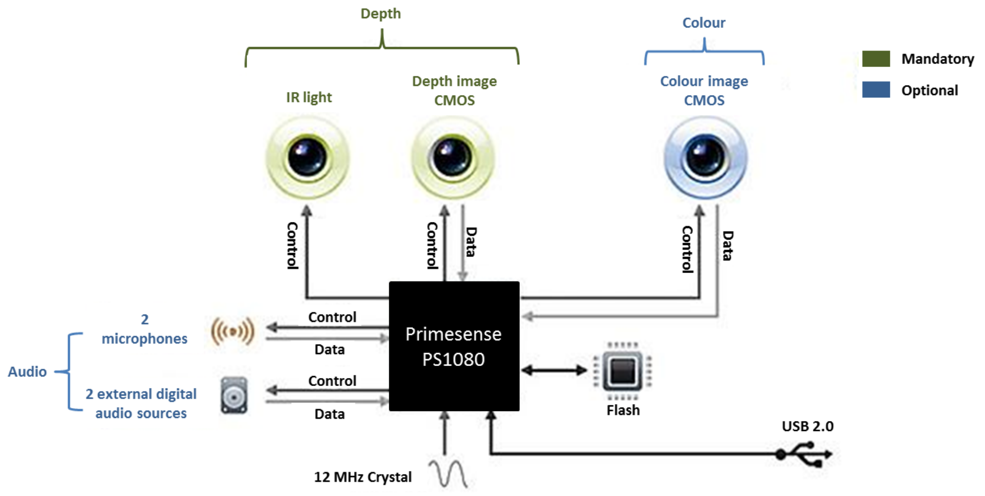


Figure 2.6: Internal schematic of the Kinect showing the connections between the three hardware components and the PS1080 microchip taken from[24].

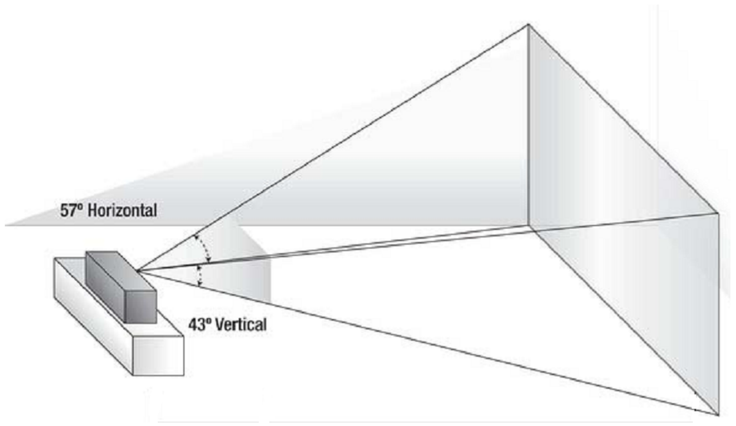


Figure 2.7: A side view of the Kinect where the field of view is shown in degrees taken from [20].

Table 2.1: Technical specifications of the Kinect [20, 24].

Field of View	
Horizontal Field of View	57 degrees
Vertical Field of View	43 degrees
Physical Tilt Range	± 27 degrees
Depth Sensor Range	0.8 m – 4 m
Data Streams	
Depth	640 × 480 16-bit depth @ 30 frames/sec
Colour	640 × 480 32-bit color @ 30 frames/sec
Audio	16-bit audio @ 16 kHz

The downside of using the Kinect is it cannot capture data in an outdoor environment due to infrared light content in sunlight meaning that it is unable to detect depth in an outdoor scene.

Software Development Kits for the Kinect

To use the Kinect there are various software development kits (SDKs) available. The most easily available is the Microsoft SDK which can be used with C, C++ and C# coding languages [25]. Unfortunately this SDK is not compatible with other open source libraries such as the OpenCV libraries, which cannot be directly applied to the Microsoft SDK. Another SDK which is popular in the research community is the OpenNI SDK as it can be easily installed and integrated with the other open source libraries [26]. It can also be coded in C, C++, C# and more recently in Matlab [27].

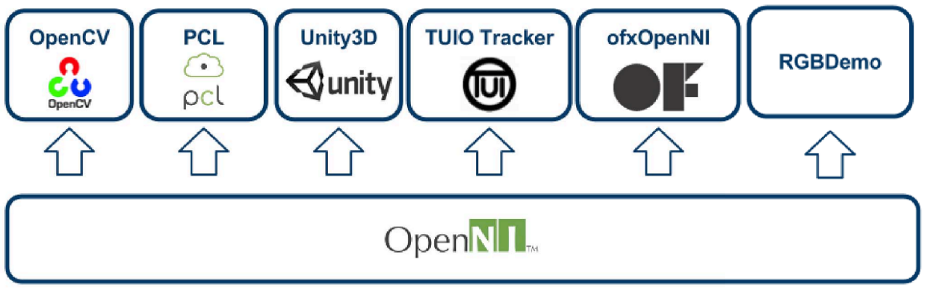


Figure 2.8: Diagram of some open source libraries that use the OpenNI SDK taken from [21].

In Figure 2.8 it can be seen that OpenCV is one of the libraries that OpenNI supports, hence it was used for this dissertation. All the code produced in this dissertation makes use of the vision libraries of OpenCV3.5 and was coded in C++. The OpenNI and OpenCV libraries include the intrinsic and extrinsic parameters of cameras, and the calibration calculations needed such that the colour and infrared images already correspond correctly. Additionally the OpenCV libraries provided image processing algorithms required build the pre-processing pipeline.

2.3 Image Processing in RGBD SLAM

In RGBD SLAM, finding and matching landmarks is done using image processing algorithms. Landmarks are often static objects, edges of objects or corners in the environment, which can be identified using a ROI algorithm. Feature extractors and descriptors are the most popular methods used to describe a landmark, where it takes into account shapes and colours. The output from the feature extraction process is an array of feature descriptors which then can be matched using a matching algorithm. These image processing algorithms are combined into a pre-processing pipeline for this study. This section provides a brief background on these algorithms. Further details are provided in Chapter 3.

2.3.1 Regions of Interest

Aulinas et al. [6] used ROIs in an underwater application of visual SLAM. ROIs were used to identify stationary objects such as rocks and used as landmarks for the SLAM algorithm. Aulinas et al. presented two procedures to find the landmarks, the first using edge detection with erosion and dilation and the second using the hue channel [6]. However, a ROI can also be produced by creating saliency maps, as shown by Frintrop [5].

For the purposes of the study, the method proposed by Aulinas et al. using edge detection with erosion and dilation filters to detect ROIs was selected for its ability to identify objects with high level of detail that could be classified as landmarks such that feature detectors would be able to detect sufficient features to describe the landmark [6].

2.3.2 Feature Extraction

Feature extraction as an area of research has produced many algorithms to obtain features from images [28]. Salient features, Harris corners and Lowe's SIFT are algorithms that can be used to extract features [29]. SIFT has an unique quality that it is both scale and rotation invariant. There has been an improvement to this algorithm called SURF; it is much faster at detecting features [1]. Recent work has produced a new rival algorithm that claims to be faster than SURF called oriented FAST and rotated BRIEF (ORB). ORB makes use of the feature accelerated segment test (FAST) and binary robust independent elementary features (BRIEF) algorithms as its backbone [30]. Endres et al. [13] demonstrated that these three feature extractors can be used with RGBD-SLAM and it was found that ORB is faster than SURF and SIFT, but produces a larger number of errors.

The only disadvantage feature extractors have, is that they extract features over the whole image captured. This can get computationally expensive especially when trying to keep a database of features over a set of images, which is the case when used with SLAM. Hence Frintrop [5] introduced ROIs whereby instead of looking at the whole image, only an area of the image is used for visual SLAM. Furthermore, Frintrop also demonstrated that the ROIs detected can be used as a landmark for the SLAM algorithm [5]. Aulinas et al. reiterated this idea of using ROIs for SLAM but in conjunction with feature extractors where features are detected from within a ROI instead of the whole image. In the work done by Aulinas et al. [6] this concept was only applied to underwater navigation [6]. SIFT, SURF and ORB were used in this dissertation, and a comparison is made to determine which one would ensure the pipeline reduces the computational time of RGBD SLAM. Chapter 3 presents further details into all the algorithms.

2.4 Summary

This chapter presented a background into the SLAM algorithm and how it can can accurately calculate the localization of the robot and build a map. The fundamental SLAM framework was discussed in detail to show it can create landmarks from captured data to map and localise the robot. RGBD SLAM similarly to visual SLAM uses feature extractors to extract features in the images to identify and track landmarks. In the RGBD framework two challenges was discussed which were the SLAM problem of loop closure and way the images are processed to extract features. Both challenges impose a computational burden on a robot and increase inaccuracies in localizing the robot. There have been many solutions researched to solve the first challenge, but the research in solving the second challenge is limited.

A background into the Xbox Kinect as a RGBD camera which is used with RGBD SLAM where the limitations of the camera are highlighted which imposes limitations on the data and the performance of RGBD SLAM. The major limitations that need to be considered is the low resolution of the camera and limited field of view.

The last section of this chapter presented the image processing algorithms that have been used with other versions of visual SLAM was discussed. The ROI algorithm has been used in visual SLAM but it has not yet been implemented into RGBD SLAM. Currently RGBD SLAM uses feature extraction and feature matching to identify and track landmarks.

Chapter 3

Image Processing Theory

From the previous chapter the background of RGBD SLAM was discussed in detail and the challenges of the algorithm. One of the challenges RGBD SLAM has is the immense image data processed to identify landmarks which can be a computational burden to smaller robots with limited computational power. The traditional method that is used applies a feature extractor to the entire image to extract a distribution features.

This chapter provides the image processing theory needed to apply the proposed method of using ROIs. The ROI are applied via a pre-processing pipeline. The pre-processing pipeline is made of three stages using various image processing algorithms as shown in Figure 3.1.

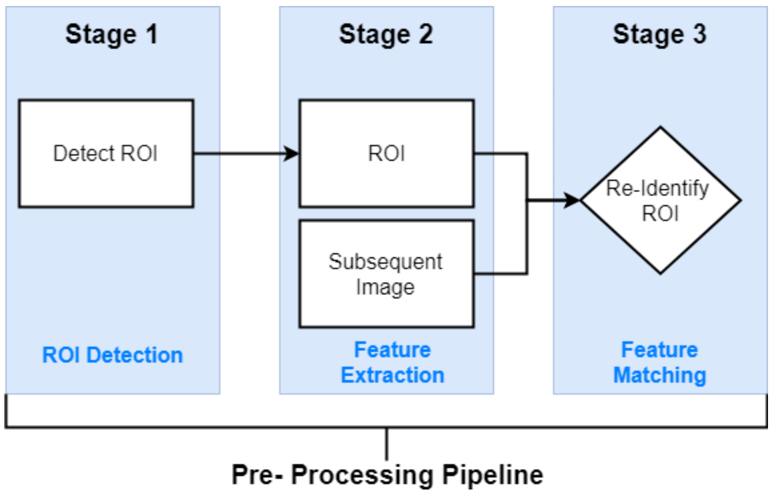


Figure 3.1: A diagram of the pre-processing pipeline used to process the data from the Xbox Kinect for SLAM.

The first stage of the pipeline discusses the use of a ROI algorithm to detect ROIs in the images and is presented in the Section 3.1. An in-depth discussion is made of how the ROI algorithm was implemented.

As discussed in Chapter 2, there is an option of three feature extractors that can be used: SIFT, SURF and ORB. This is the second stage of the pipeline and is discussed in more detail in Section 3.2 of this chapter.

Once features are extracted from a ROI, feature matching is used to re-identify the ROI. Once the ROI has been re-identified successfully, it is saved into the database as a landmark. The steps involved in the matching process are presented. This is the last stage of the pipeline and is discussed in Section 3.3 of this chapter. Section 3.4 presents a summary of the theory used developing the pre-processing pipeline.

3.1 Regions of Interest

From the work presented by Aulinas et al. [6]. the use of ROIs was chosen as a sensible strategy to save on computational time and performance to identify and track landmarks. Using ROIs will allow the feature extractors to only extract features from a ROI rather than extracting features from the whole image, which will save on computational time and performance.

According to SLAM a definition of a landmark is a uniquely identifiable object in the world that can be robustly recognised later on. Highly detailed objects in a scene are often detected and used in visual SLAM as landmarks. Hence, the ROI is designed to detect such objects. It is assumed that in the environment there are highly detailed objects that can be used as landmarks. If there are few or no detailed objects the ROI algorithm will struggle to find landmarks.

The method of determining a ROI is shown in Figure 3.2, where an input image is captured from the VGA camera of the Kinect. Edge detection is used to find objects with a high degree of detail for example, a box that has writing or shapes on it. A Canny edge detector is used for this step. The output from the edge detection is a binary image. The second step is the erosion and dilation operations that are used to further enhance the edges detected into foreground (black) and background (white) patches. The final step uses connected component analysis, where the largest foreground patch is found and highlighted on the colour image by drawing a red box surrounding the region to indicate a ROI. This process of determining a ROI was also used by Aulinas et al. [6].

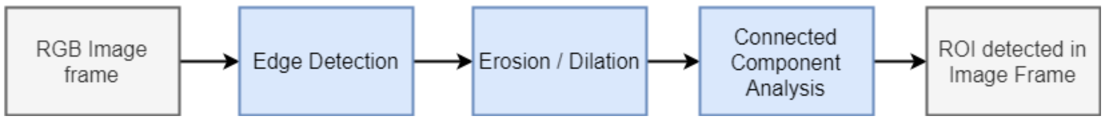


Figure 3.2: A flow diagram of the process where a ROI is obtained from an image.

3.2 Feature Extractors

The second stage of the pre-processing pipeline is to build unique feature descriptors of the ROIs saved, using feature extractors. The unique descriptors are used to track the ROIs in the environment. Once a ROI is identified and matched, it can be used as a landmark in RGBD SLAM. The pipeline has the option of using three feature extractors which are SIFT, SURF and ORB to determine which one would best suit the pipeline in reducing the computational time. This section provides further details of each feature extractor. These three algorithms are investigated as they are the three most popular feature extractors used in RGBD SLAM.

3.2.1 Scale Invariant Feature Transform

One of the most widely used feature extraction algorithm in image processing is Lowe's SIFT. Due to its ability to produce descriptors that are invariant to scale and orientation it has become one of the most popular algorithms. Hence, this algorithm was considered for the pre-processing pipeline. A summary of SIFT is presented in this section. A detailed explanation is provided by Siciliano and Khatib, Lowe and Clemons [1, 29, 31].

SIFT uses the terminology of key-points instead of features. The key-points are determined by using Difference of Gaussian (DoG) and are located by finding the local extrema. DoG key-points are scale invariant as shown in Figure 3.3. To make them invariant to rotation an orientation assignment is conducted where the neighbourhood around a key-point is used to calculate the gradient magnitude and direction of that region.

The key-point descriptors are built by creating a bin orientation histogram around it as shown in Figure 3.4. The descriptors also take into account changes in illumination to produce robust results. Once the descriptors are found then the nearest neighbour classifier can be used to match key-points [28].

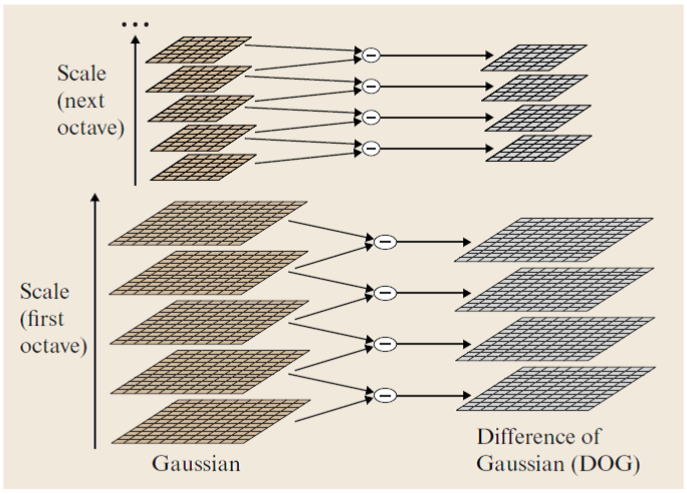


Figure 3.3: A visual representation of the calculation of DOG from the different scales used to extract SIFT key-points taken from [1].

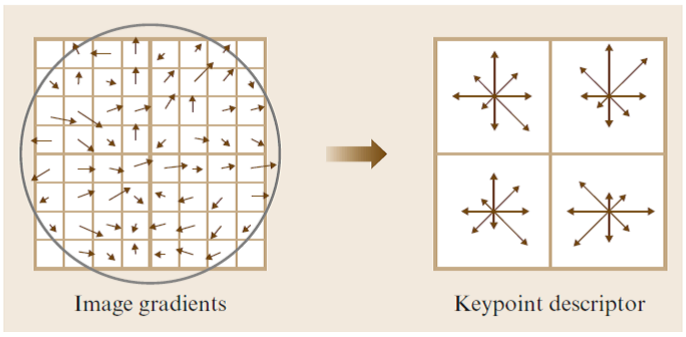


Figure 3.4: Building a SIFT key-point descriptor by creating a bin orientation histogram taken from [1].

3.2.2 Speeded Up Robust Features

SURF is based on the same principles as SIFT. However, it tries to improve on the time taken to extract features while ensuring the features are still robust. This section presents a summary of SURF. Bay et al. present a detailed explanation of SURF [32].

SURF finds the key-points by using the determinant of a Hessian matrix of second order derivative Gaussian functions. The descriptors are built by applying a Haar-wavelet around the neighboured of a key-point and summing the responses of those points. The determinant of the Hessian matrix makes the points scale invariant and the response from the Haar-wavelet ensures the points are invariant to rotation. In the matching process, the key-points are matched according to their contrast.

3.2.3 Orientated FAST and Rotated BRIEF

In response to SIFT and SURF, ORB was developed to extract features faster and improve on the matching of features for better accuracy. ORB combines a FAST key-point detector to locate the key-points and BRIEF to build the descriptors for the key-points. The theory of ORB feature extraction is summarized in this section. A detailed explanation is presented by Rublee et al. [30].

The first stage of ORB is the detection of FAST key-points, which is done by taking the intensity threshold between the centre pixel and those pixels enclosed within a circular patch centred on the pixel. Typically a pixel radius of nine is used, as it is demonstrated to give better performance in terms of accuracy and time [33].

However, a disadvantage of FAST is that it has no way to detect corners in an image, hence a Harris corner measure is used to further refine the key-points detected. FAST does not produce multi-scale features, so in order to consider multi-scale features a scale pyramid is produced of the image. Once the scale pyramid is produced the FAST features are detected at each level of the pyramid and filtered using the Harris corner measure. This process ensures the points are invariant to scale.

The key-points need to be rotation invariant and the orientation of the key-points is unknown. Therefore to take into consideration the orientation, the intensity centroid is calculated. The intensity centroid makes an assumption that the corner intensity is offset from the centre, and this is used to determine the orientation [34].

Once the key-points are found the descriptors are built using BRIEF. However, BRIEF performs poorly when orientation is considered, hence steer BRIEF is used to determine orientation accurately. In matching the descriptors, Locality Sensitive Hashing (LSH) is used to search for the nearest neighbour [30].

3.3 Feature Matching

The last stage in the pre-processing pipeline is to track the ROIs in the environment, which can be used as landmarks for RGBD SLAM. From the features extracted in the previous stage, feature matching is used to accomplish this task. The feature matching process for each feature extraction algorithm uses FLANN and RANSAC with homography matrices.

This last stage is crucial to re-identify the ROI in the subsequent image frame so that it can be saved as a landmark. Additionally this stage also ensures that loop closure is achieved correctly in RGBD SLAM when a landmark is revisited.

3.3.1 Fast Approximate Nearest Neighbours

Based on the feature descriptors built by the feature detectors, feature matching is used to match the descriptors. OpenCV offers two methods for feature matching which is the brute-force method and the fast approximate nearest neighbours classifier also known as FLANN [35]. The brute force method finds the closest match using distance calculations. However, these calculations are known to take more computational time. Hence, for faster performance, FLANN is used in this dissertation as the feature matching algorithm. FLANN is able to perform well as it can handle large datasets and optimize for speed the search for nearest neighbours. For SIFT and SURF the default FLANN function was used, but for ORB the FLANN function had to be specified to use LSH to match the features. For further details into FLANN refer to the research done by Muja and Lowe [36].

3.3.2 RANSAC

During the feature matching process, false matching does occur and needs to be eliminated to produce robust and accurate matches. Once FLANN is applied, RANSAC is used to eliminate the false matching. In the context of RANSAC good matches are known as inliers and false matches are known as outliers. To find the inliers the feature matches from FLANN are converted into a mathematical model, where RANSAC is used to estimate the inliers of the model. RANSAC is primarily implemented to ensure that the location of a ROI is accurately found in the image.

3.3.3 Homography

The last step in feature matching is to locate the ROI if it is in the image no matter what the geometry. Hence, a perspective transformation is needed to find the landmark in the image. A homography matrix needs to be found to perform the perspective transformations. In OpenCV, the homography matrix is implemented with RANSAC to ensure that from the good feature matches the most accurate matrix for the perspective transformation is found.

3.4 Summary

This chapter presented the theory required to develop the proposed method of using ROIs to process the image data to identify and track landmarks for RGBD SLAM, rather than the traditional method. The ROI is implemented using a pre-processing pipeline. The pipeline is made of three stages. The first stage detects a ROI in the image; the second stage applies a feature extractor within the ROI and the last stage uses feature matching to accurately re-identify the ROI and classify it as a landmark for RGBD SLAM.

The first stage uses the method proposed by Aulinas et al. [6] where the ROI is detected using an edge detector with erosion dilation filters. The second stage presents an option of three feature extractor (SIFT, SURF and ORB) that be used to detect features from the ROI. The last stage uses feature matching with RANSAC to ensure that the ROI is correctly re-identified in the environment.

Based on the theory presented in this chapter, the next chapter presents the methodology of the pipeline in form of a flowchart and how it is integrated into the RGBD SLAM framework.

Chapter 4

Pre-processing Pipeline

The previous chapter presented the image processing theory to construct the pre-processing pipeline. The pipeline implements an alternative method of using ROI to process the image data to identify and track landmarks for RGBD SLAM. This method is used to improve the computational time of RGBD SLAM compared to the traditional method.

Section 4.1 discusses the methodology of the pipeline in the form of a flowchart. Section 4.2 presents an evaluation of the pipeline where each stage is evaluated to ensure the image data are processed correctly to detect ROIs and features. This process is vital to ensure that each stage functions correctly before the full pipeline algorithm is tested and integrated into the RGBD SLAM framework. Section 4.3 presents the integration of the pipeline into the framework and Section 4.4 presents a summary of the chapter.

4.1 Flowchart of the Pre-Processing Pipeline

This section presents in detail the image processing theory and algorithms used for each stage of the pre-processing pipeline, which can be summarized into a flowchart. The input is images and depth measurements from the Kinect and the output is a database of landmarks with depth measurements for RGBD SLAM, as shown in Figure 4.1. The flowchart of the pre-processing pipeline was coded into one algorithm.

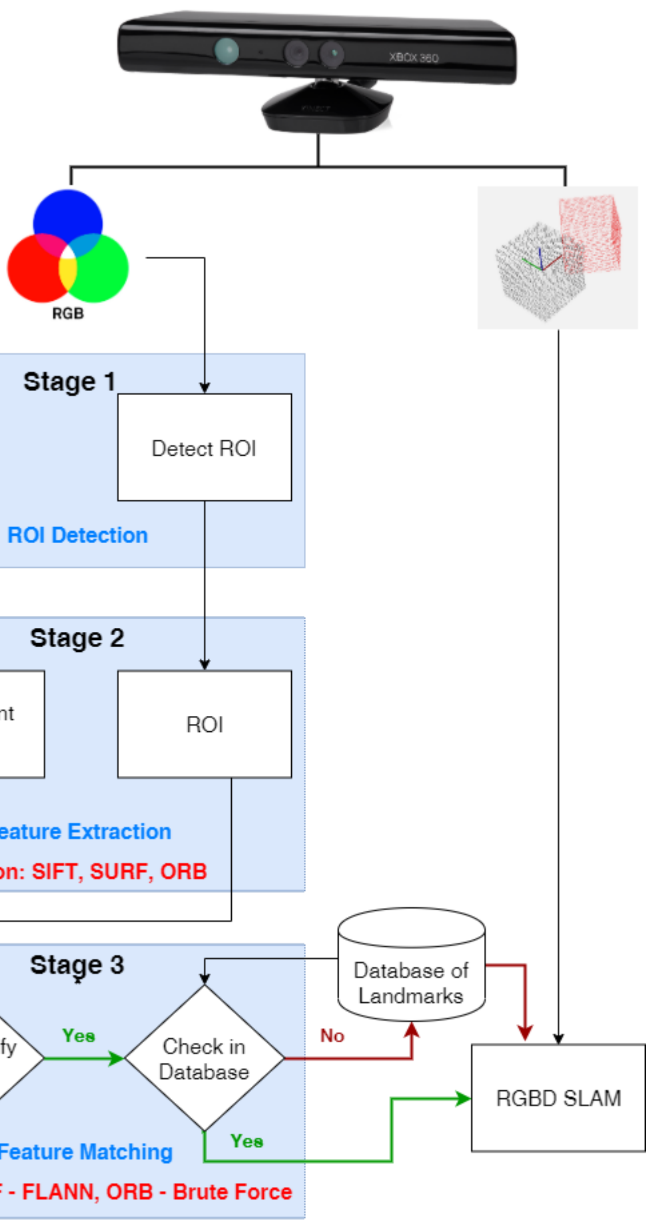


Figure 4.1: A flowchart summarizing the pre-processing pipeline into one algorithm.

To read and access the data from the Kinect, OpenNI drivers were used [21]. OpenCV image processing libraries were used and the pre-processing pipeline was coded in C++.

The pipeline initialises with no landmarks in the database. As a robot explores an environment, the algorithm creates a database of landmarks. The ROI detected in the first image captured by the Kinect is automatically stored and considered a landmark.

As the images of the environment are sent to the pipeline, the first stage in the pipeline will detect a ROI. The ROI is designed to contain uniquely identifiable objects that could be used as landmarks. Once a ROI is detected then the second stage uses feature extractors to build a feature descriptor of the ROI. The feature descriptors are used in the last stage to find the ROI in the most recent image capture. If the ROI is in the image than it is confirmed to be landmark. If the ROI is not in the image then a new ROI is detected in the most recent image. When the ROI is confirmed to be in the image a second check is conducted to see if the landmark has been seen before, hence it will be checked against a database of landmarks. If the landmark has not been seen before then it considered to be a new landmark and is saved to the database.

When a duplicate landmark occurs in the database, the algorithm failed to re-identify the landmark. When this occurs, this is considered a false negative.

Once the landmark has been checked, the depth measurements from the point cloud data of the landmark are sent to RGBD SLAM to update the current location of the landmark. In this manner RGBD SLAM can keep track of all the landmarks.

4.2 Evaluating the Pipeline

After the pipeline was constructed into an algorithm as seen in the flowchart in Figure 4.1, the pipeline was evaluated at each stage. Each stage of the pipeline is evaluated to ensure the image data are processed correctly. The pipeline was evaluated using images of an object captured in a room with a featureless and low detail background. The object was chosen to be a box containing a fair amount of writing to ensure a high degree of detail that can be detected. This scene is chosen as the object is an ideal landmark that can be used for RGBD SLAM, hence the ROI stage can be fine tuned to detect this object as a landmark. Additionally the feature extractors would be able to extract features of the object for the feature matching stage. Once each stage of the pipeline is evaluated then the full pipeline algorithm can be used for further testing. An image of the scene is shown in the top left of the diagram in Figure 4.2.

4.2.1 Stage 1: ROI Detection

In Chapter 3 the Aulinas et al. method is mentioned as the method used to detect a ROI. The ROI is detected using an edge detector with erosion and dilation filters. To segment the ROI from the image connected component analysis is used. These processes are shown in Figure 3.2.

Edge Detection

In this study Canny edge detection was used in the first step in determining a ROI. The Canny edge detector offers three adjustable parameters, namely the size of the Gaussian filter and a low and high threshold. The Gaussian filter is a smoothing filter to either sharpen or blur the edges. The default 3×3 kernel size was used for the Gaussian filter. The low and high threshold parameters are used to limit the sparse definitions and were set to 250 and 300 respectively. These parameters were chosen such that the edges from an object are detected optimally in a range of 0.2 m to 1.8 m from the Kinect.

In Figure 4.2 (top right) it is shown that the edge detector was able to detect the degree of detail on the object and output image as a binary image. The object in the image is a box with a fair amount of writing to ensure that the edge detector is able to pick up the detail. For the ROI algorithm to function there must be some detail in the environment for the edge detector to detect a ROI which limits the use of the ROI algorithm.

Erosion and Dilation

Once the edge detection is applied to the image, erosion and dilation operations are applied to further enhance the degree of detail and filter out any background noise picked up by the edge detector. For both the erosion and dilation operations a 3×3 kernel was used. The number of iterations for applying these operations can be and were adjusted. Number of iterations for erosion and dilation were set to six and seven respectively as these were the default values in the OpenCV library. The erosion operator is first applied to remove the background noise then the dilation operation is applied to the binary image to convert the edges detected into larger foreground patches. During testing after the pipeline was tested in isolation it was required to detect for a larger ROI. To enable the ROI algorithm to detect a larger ROI the number of iterations the of dilation were operation reduced to two.

As shown in Figure 4.2 (bottom right) the background edges are filtered out and the details on the box are converted to larger foreground patches.

Connected Component Analysis

The last step in the ROI process is to identify the largest foreground patch, since from the previous step these patches indicate a high degree of detail in the image. Connected component analysis is used to find the largest patch. For displaying purposes once it is found a red box is drawn around the area containing it.

In Figure 4.2 (bottom left) the box was successfully determined to be the area in the image containing the highest degree of detail using the connected component analysis method. Hence the red box is drawn around the box and is segmented as a landmark that will used in the next stage of the pre-processing pipeline. The segmented region is shown in Figure 4.2 (bottom centre) and is marked as a landmark for SLAM.

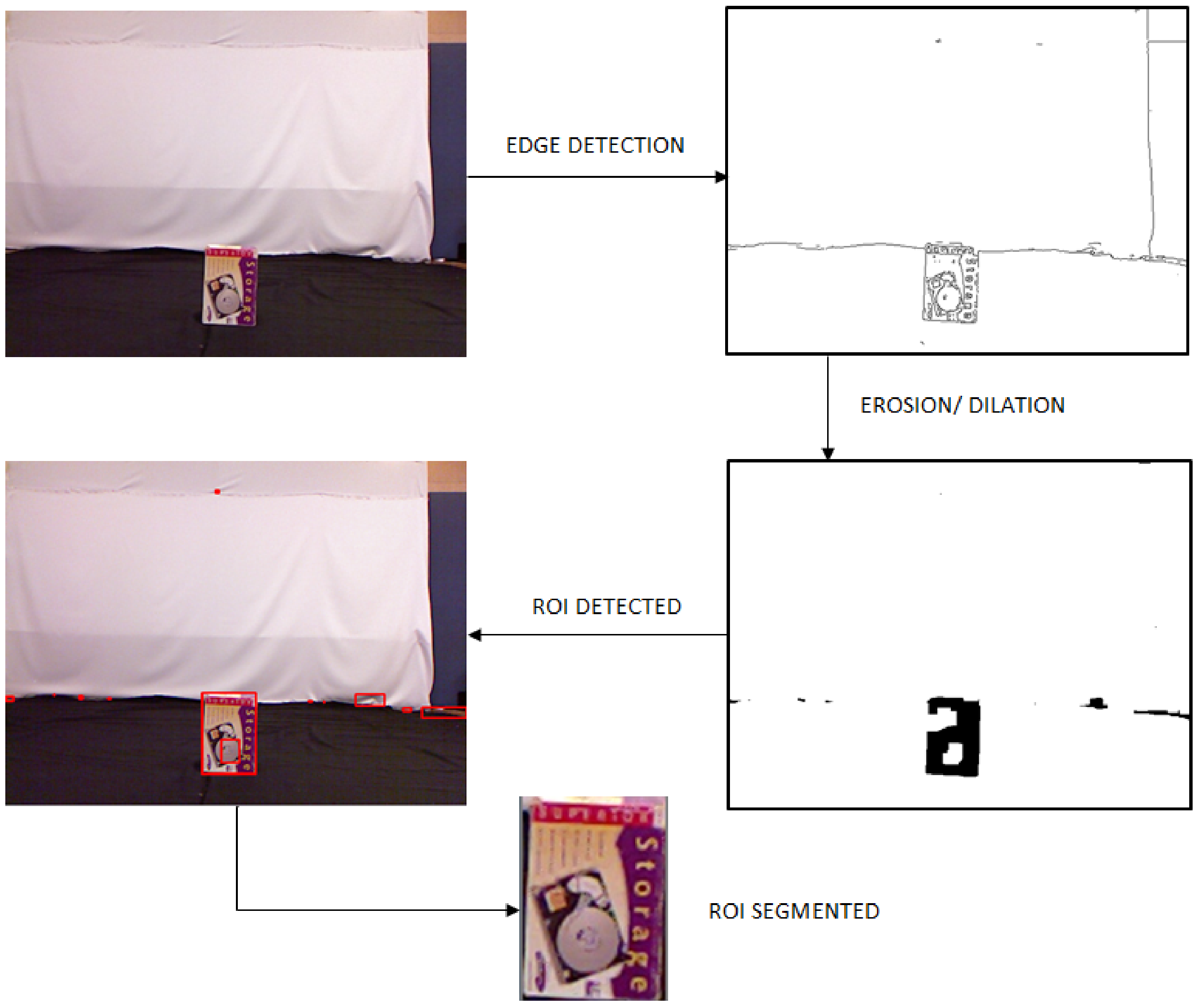


Figure 4.2: Step-by-step process of the ROI algorithm identifying a highly detailed object that could be used as landmark for RGBD SLAM.

4.2.2 Stage 2: Feature Extraction

After stage 1 where the ROI is detected and segmented, stage 2 applies the feature extractor. The feature extractor is applied to both the ROI and the most recent image as shown in Figure 4.3. Once the features have been detected, the features with its feature descriptors are sent to the next stage for feature matching.

Additionally this stage offers three types of feature extractors that can be used. The three options are SURF, SIFT and ORB. Figure 4.3 shows the three feature detectors applied to the ROI and the most recent image. In the algorithm the default setting parameters where used for SURF, SIFT and ORB as specified in the OpenCV libraries. These default values were chosen as RGBD SLAM also uses the same default values in the traditional method to extract features. This is to ensure comparable results when testing the pipeline with RGBD SLAM.

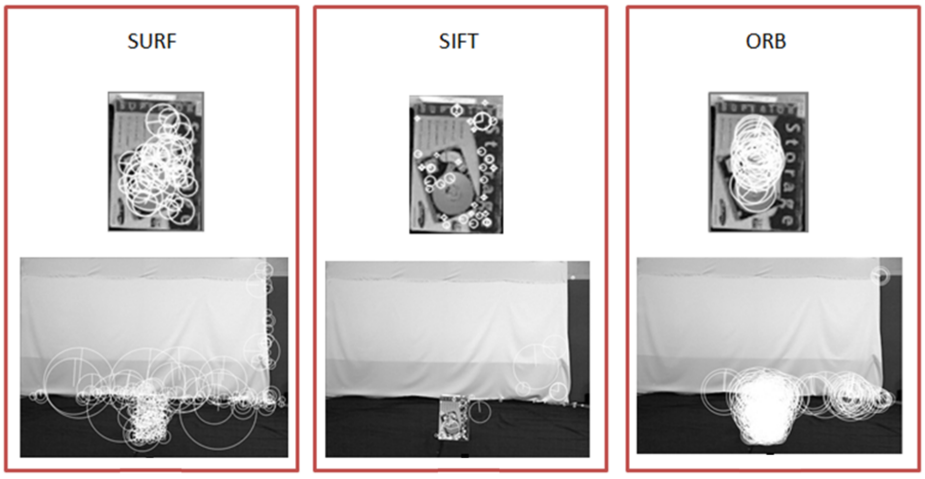


Figure 4.3: The three feature extraction algorithms used in the pre-processing pipeline to extract features.

4.2.3 Stage 3: Feature Matching

Once the feature is extracted and its descriptors built, the descriptors are used in stage 3 for feature matching. The feature matching process is used to check if the ROI is in the most recent image. As shown in Figure 4.4 stage 3 is successful in locating the ROI in the image.

Depending on the type of feature extractor used the feature matching method differs.

For SURF and SIFT the FLANN method is used, and for ORB the brute force method is used. Feature matching for SURF, SIFT and ORB is shown in Figure 4.4.

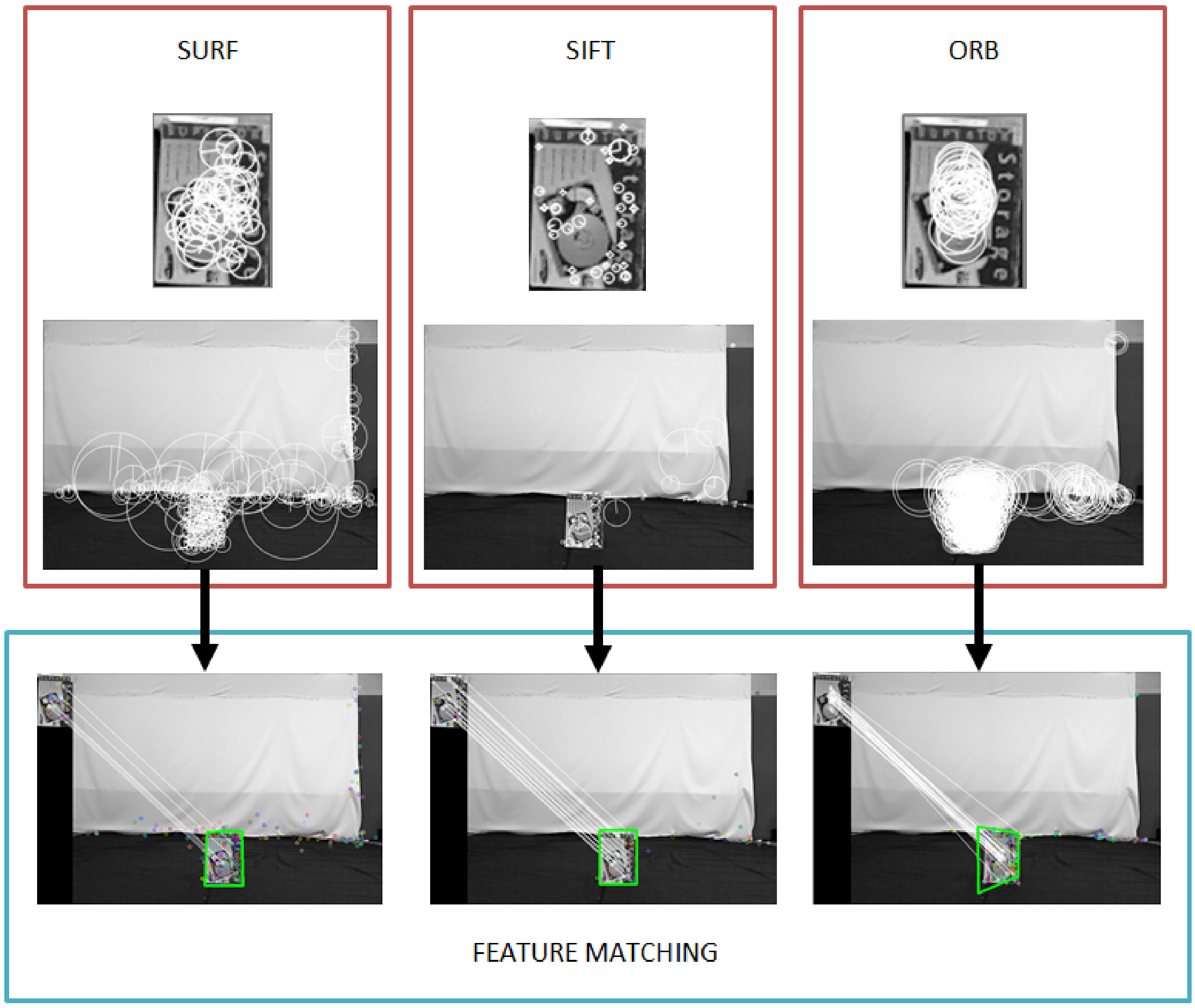


Figure 4.4: The feature extraction and matching process implemented in the final algorithm.

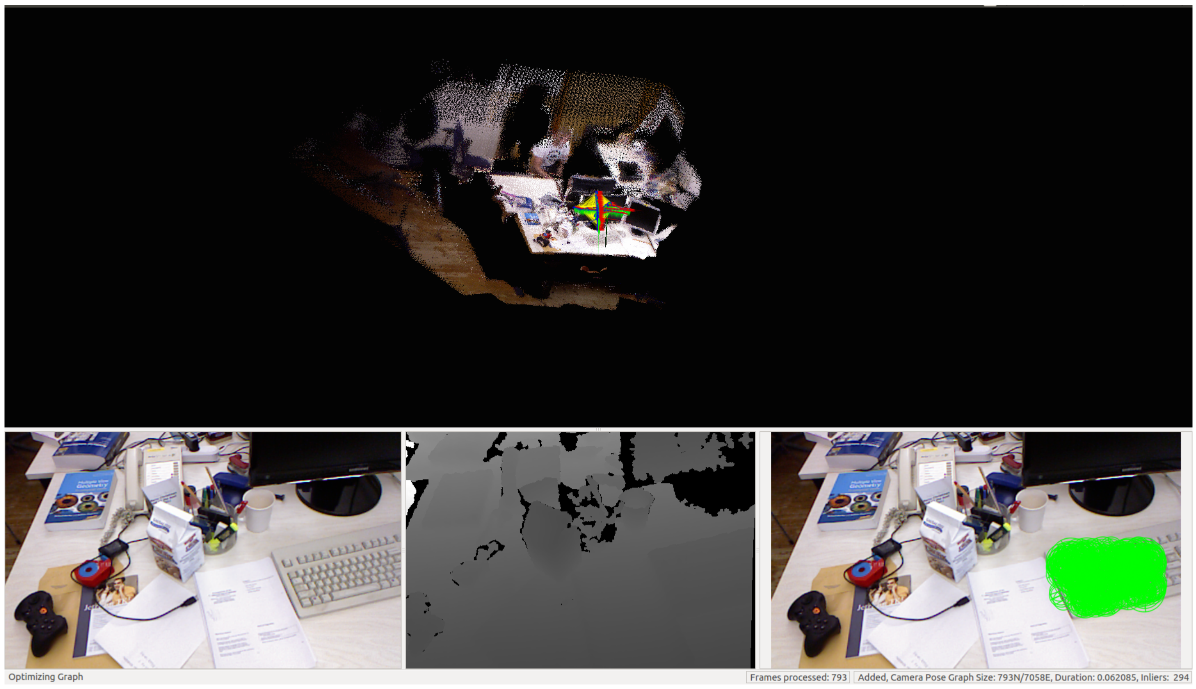
Feature matching is used again on its own in the pipeline when the newly found landmark is checked against the database of landmarks to determine if it was seen before.

4.3 Integration of the Pipeline in RGBD SLAM

After evaluating the pipeline, it is tested in isolation before it is integrated into the RGBD SLAM framework. To integrate the pipeline the module that processes the image data within the RGBD frameworks must first be identified. The front end of the module is shown to do this as discussed in Chapter 2.

The feature extraction and matching is done in the front end of the framework. To complement this process the pipeline is added to this module. The image data from the Kinect will enter the front end of the framework and first get processed through the pipeline to detect the ROI. Once the ROI is detected the data will continue through the normal process of RGBD SLAM.

Endres et al. developed RGBD SLAM using the Robot Operating Software (ROS) which is freely available to the research community. This version was used and modified in this study. A graphical user interface (GUI) is shown in Figure 4.5 that contains the 3D map created, the extracted features, the input colour image and the depth image.



Optimizing Graph

Frames processed: 793

Added, Camera Pose Graph Size: 793N/7058E, Duration: 0.062085, Inliers: 294

Figure 4.5: The GUI of RGBD SLAM.

4.4 Summary

From the last chapter the theory of image processing algorithms is discussed and is used in this chapter to construct the pre-processing pipeline. A flowchart of the pipeline is shown and each stage of it is evaluated. This evaluation process is needed to ensure the image data is processed to correctly to detect a ROI with feature detection and matching.

The evaluation process showed that the pipeline was able to successfully detect a ROI in the image. The ROI detected was an ideal object that is uniquely identifiable and can be used as a landmark for RGBD SLAM. The feature extractors applied did allow the feature matching stage to locate the ROI correctly in the most recent image.

Lastly the pipeline needed to be integrated into the RGBD SLAM framework. The front end of the framework was designed to be modulus to integrate the pipeline into the framework. The next chapter performs test of the pipeline in isolation and thereafter tests of the RGBD SLAM with pipeline integrated into it.

Chapter 5

Testing and Results

The last chapter presented the methodology of the pre-processing pipeline, and this chapter will present the tests performed on the pipeline in isolation and with the pipeline integrated fully into the RGBD SLAM framework.

The pipeline is developed as an alternate method of processing the image data where ROIs are used to identify and track landmarks for RGBD SLAM. The pipeline serves as a solution to improve the way RGBD SLAM processes the image data and optimizes the identification of landmarks to reduce the computational time. This method is compared to the traditional method currently used where a distribution of features is detected from an entire image frame to identify and track landmarks. This chapter will perform tests to compare the two methods to determine if the pipeline does improve the computational time. There are three sets of tests that are done. The first set tests called test 1 analysed the pipeline in isolation. The second set of tests, called test 2, integrated the pipeline into RGBD SLAM which is analysed and compared to the traditional RGBD SLAM. The last set of tests, called test 3, conducts the same tests performed in test 2, but with a reduced number of features.

Section 5.1 presents the analysis of the results from test 1. The results from test 2 are presented in Section 5.2 and Section 5.3 presents the results from test 3.

5.1 Test 1: Pipeline Tested in Isolation

Before the pre-processing pipeline is integrated into the RGBD SLAM framework, the functionalities of the pipeline are first tested. It is tested to determine whether the pipeline can create a database of landmarks using ROIs, the precision of re-identifying a ROI in the scene and the accuracy of being able to recall a landmark that has been previously seen and stored in the database. The tests applied the pipeline to typical indoor SLAM office environments. Seven indoor office environment datasets downloaded from: http://vision.in.tum.de/data/datasets/rgbd-dataset [14] capturing typical scenarios for SLAM were used.

Each dataset was fed into the pipeline and processed. While the pipeline is processing each dataset, a log file is created. In the log file the number of matched feature and falsely matched features are stored. Additionally, the computational time required to process a dataset is also recorded and stored in the log file. The computational time logged the measured duration for the entire implementation to execute. Once the pipeline has processed the dataset, a database of landmarks is created. After processing each dataset the log file and the database are analysed. The log file is analysed to determine the average number of true positives and false positives of matches. The database is analysed to determine the number landmarks stored and the number of duplicate landmarks.

Each dataset was processed three times to allow the pipeline to apply the three feature detectors to the dataset. After all the datasets are processed, a comparison is made of all the datasets to determine which dataset posed a challenge to the pipeline in terms of correctly identifying and tracking landmarks.

The seven datasets are labelled and listed as follows with the number of frames per dataset:

- 1 = fr1/xyz (797 frames).
- 2 = fr3/structure_texture_far (914 frames).
- 3 = fr3/structure_texture_near (1065 frames).
- 4 = fr3/nostructure_texture_far (454 frames).
- 5 = fr3/nostructure_texture_near_withloop (1648 frames).
- 6 = fr1/360 (755 frames).

- 7 = fr1/floor (1242 frames).

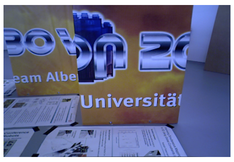
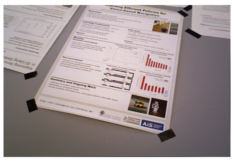
The first five datasets were captured in a small office environment, where the Kinect was moved slowly through it. These five datasets were used to investigate the robustness and performance of the feature extraction algorithms in terms of the number of features extracted and features matched. The datasets vary from an office desk to highly detailed boxes and a highly detailed floor.

The last two datasets are typical environments where SLAM can be used for navigation. The Kinect was moved a little faster compared to the previous five datasets. One dataset is a 360-degree view of an office and the other is a sequence of images navigating through an open office at floor level. This was to test the overall robustness of the pre-processing pipeline.

5.1.1 Description of the Datasets

A description for each of the datasets is presented in Table 5.1. Each dataset captured different typical scenarios of an office environment. A description of the input files for the datasets is presented in Appendix A.

Table 5.1: Decription of each dataset and a sample image.

Dataset No.	Description of Dataset	Sample Image
1	In this dataset the Kinect moved around a typical office desk very slowly. The movement was in the x, y and z direction of the Kinect.	
2	This dataset contained highly detailed boxes and papers stuck to the floor. The Kinect was placed further away from the boxes.	
3	Same as dataset 2 but the Kinect was placed closer to the boxes.	
4	The dataset is of the same environment as the previous one except it only has the highly detailed papers in the scene stuck to the floor. This dataset has no boxes in the scene to eliminate structured objects, hence only non-structured, flat, detailed papers was left in the scene.	
5	Same as dataset 4 but the Kinect's movement formed a loop.	
6	This dataset captured a 360-degree sequence of images of a typical office. The movement of Kinect was faster than the previous dataset causing blurry images.	
7	This dataset captured the movement of the kinect through a typical open office environment. The Kinect was placed at floor level. The floor was wooden and contained knotholes that could be used as landmarks.	

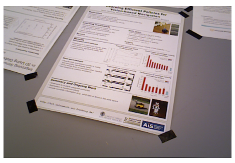
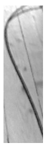
5.1.2 Results from Dataset Testing

All the log files recorded from the datasets were analysed in Matlab and are presented in this section for each dataset. The depth measurements captured of the landmarks were not analysed as ground truth was not available in the datasets. A description of the output log files from the pre-processing pipeline is presented in Appendix A.

In these tests three sets of measurements are produced. The first measurement is the number of landmarks in the database; the second measurement is the number of false positives of a ROIs (number of falsely matched ROIs to existing landmarks in the database) and the last measure is the number of false negatives of landmarks. As the pre-processing pipeline processes each dataset, it builds a database of possible landmarks that could be used for SLAM. The first and third measurements are determined by analysing the database and second measurement is determined by analysing the log files. A sample image frame from each dataset is shown in Table 5.2 with a ROI detected from it.

Once a ROI has been detected, feature detectors are used to detect features within the ROI and feature matching is used to identify the ROI in the next image frame. During the matching process unique feature descriptors are built of the ROI and of the new image. When the ROI has been detected in the new image, any feature descriptor that has been matched outside the ROI in the new image is considered a falsely matched feature.

Table 5.2: Sample Frame from each Dataset with its corresponding ROI.

Dataset No.	Image Frame	ROI
1		
2		
3		
4		
5		
6		
7		

When a ROI is re-identified, it is considered to be a possible landmark and the whole ROI is stored into the database with its feature vectors. As more landmarks are added to the database, it is checked to see if it has been seen before. However, after the tests had been completed the database was analysed and it was found that duplicate landmarks occurred, indicating that this was an error in re-identifying the landmark. The number of duplicates that have occurred is the number of times it failed to re-identify a landmark which are false negatives.

5.1.3 Analysis of Dataset Results

The results from the log files from all the datasets were analysed in Matlab to see the robustness, effectiveness and performance of the functionalities of the pre-processing pipeline. The raw results that was captured and analysed from this test are presented in Appendix B. Over all the datasets three sets of measurements were analysed which were the number of landmarks stored in the database, the precision of re-identifying a ROI in terms of false matches and the accuracy of recalling a landmark in the database. The analysis of these three sets of measurements would determine whether the pipeline would be able to identify and track landmarks in the environment for SLAM accurately.

In addition the identifying and tracking of landmarks per feature extraction algorithm is analysed over all the datasets. Thereafter the performances of each feature extraction algorithm is analysed in terms of the average number features matched, average false features matched and execution time over all the datasets. This analysis would give an indication into which feature extraction algorithm performed the best in the pre-processing pipeline.

Figures 5.1, 5.2 and 5.4 plot the bar graphs of the results recorded of the landmarks for all datasets.

Landmarks

In Figure 5.1 the number of landmarks saved in the database per feature extraction algorithm for all the datasets is shown. It was found that in dataset 2 all the feature extraction algorithms recorded the lowest number of landmarks in the database compared to the other datasets. This is as expected due to the environment in dataset 2 containing few objects which were non-structured, flat and highly detailed. However in the last two datasets all the feature extraction algorithms resulted in the greatest number of landmarks saved to the database. The main cause of such high numbers was due to Kinect moving quite fast through the environment capturing blurry images. The blurry images affected the ability of the feature extraction algorithms to precisely re-identify a ROI, resulting in a higher number of false positives. Hence the pipeline incurred duplicate landmarks in the database which produced higher number of false negatives of landmarks. Datasets 6 and 7 also required more computational memory to store the landmarks compared to the rest. ORB recorded the highest number of landmarks for each dataset.

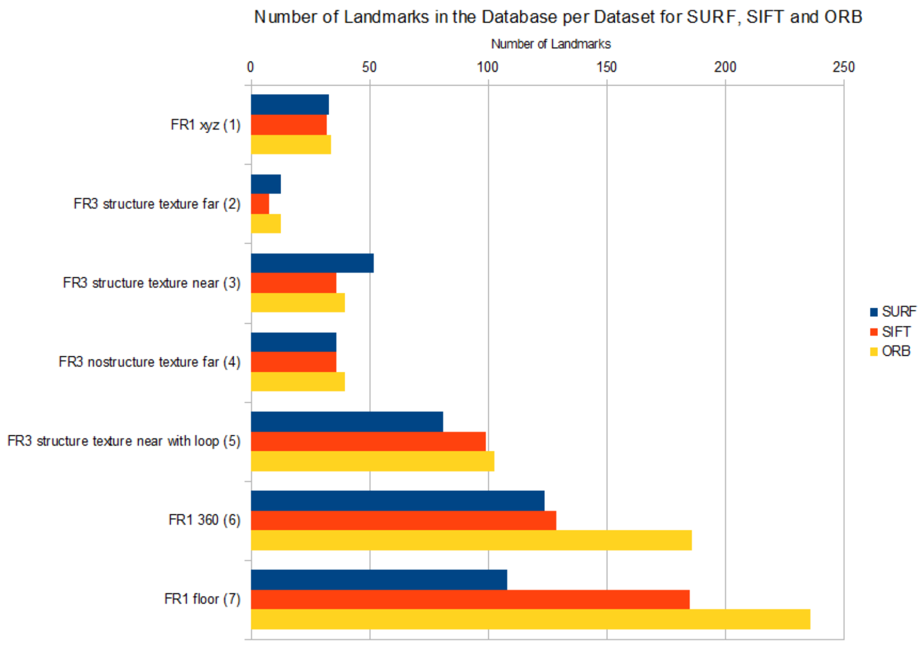


Figure 5.1: Bar Graph of landmarks for SURF, SIFT and ORB for all the datasets.

False Positive Landmarks

Figure 5.2 shows the number of false matches of ROIs to an existing landmark in the database, for each feature extraction algorithm for all the datasets.

SIFT and SURF performed well in dataset 2, where they both recorded no false positives, which in this case showed these feature extractors detected the ROIs correctly all the time. This could be due to the Kinect moving more slowly in the environment compared to the other datasets which made it easier to re-identify a ROI in the database.

In most cases it was either SIFT or SURF that produced the lowest false positives, while ORB produced the most. Dataset 4 was the only case where SIFT recorded the highest compared to SURF and ORB, but still low when compared through all the datasets. For Dataset 6 containing blurry images, SIFT appears to perform well in recording the lowest false matching of landmarks compared to SURF and ORB.

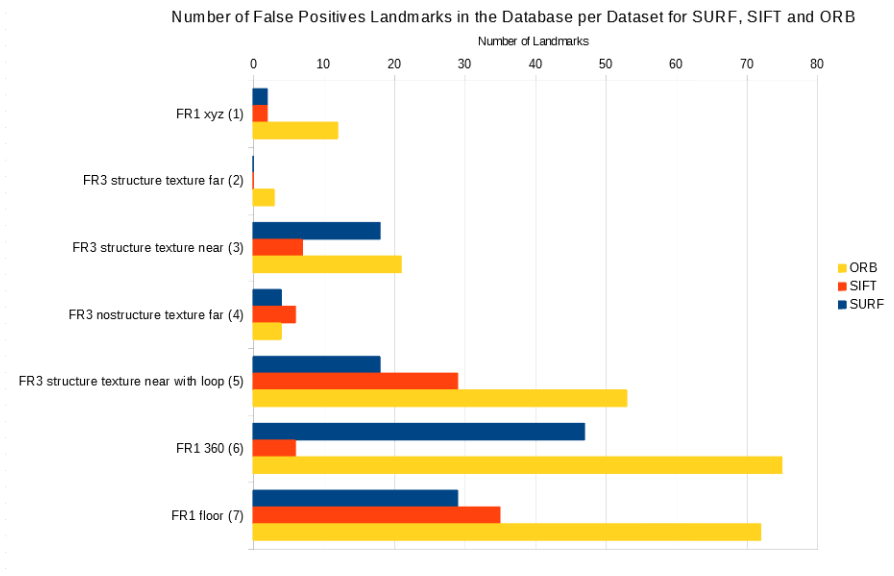
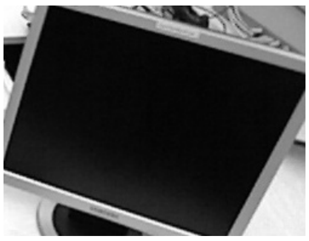


Figure 5.2: Bar graph of number of false positive ROIs for SURF, SIFT and ORB for all the datasets.

False Negative Landmarks

In Figure 5.4 the number of false negatives of landmarks in the database for each feature extraction algorithm for all the datasets is shown. False negatives are when a landmark is already in the database but it is saved as a new landmark, hence the database would incur duplicate landmarks. The number of duplicate landmarks is the number of times the pipeline failed to re-identify an existing landmark. An example of a failed re-identification is shown in Figure 5.3 where a duplicate landmark is found in the database.


(a) Landmark No. 16 in the database.

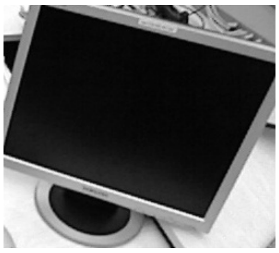

(b) Landmark No. 17 in the database.

Figure 5.3: An exmple of a failed recall when a duplicate landmark is found in the database.

In dataset 2 SIFT outperformed SURF and ORB in producing no false negatives in the database. ORB in all the datasets produced the highest number of failed false negative in the database. SURF produced the lowest number of false negatives for datasets 4 and 7, compared to SIFT and ORB. However in dataset 7 with blurry images SURF is noticeably better compared to SIFT and ORB with high numbers of false negatives of landmarks in the database.

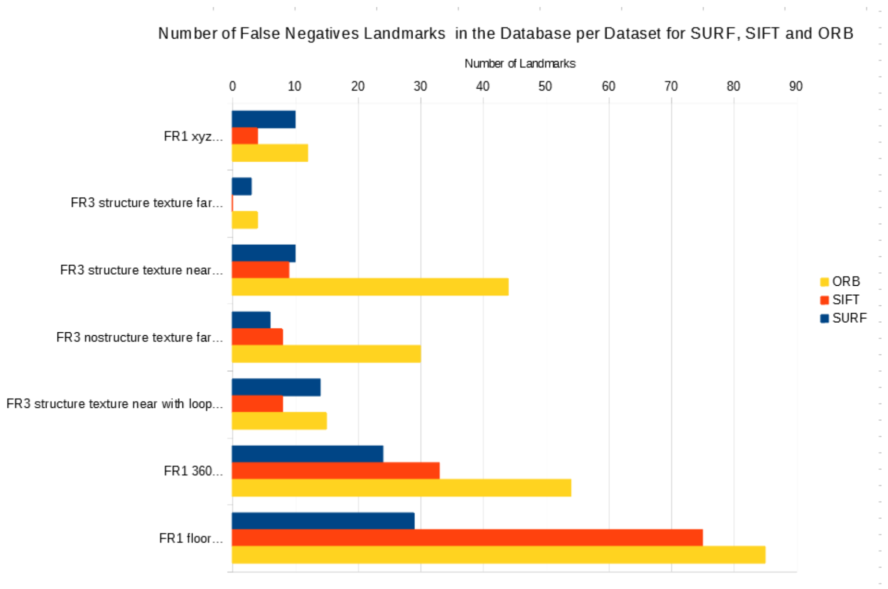


Figure 5.4: Bar graph of number of false negatives of landmarks in the database for SURF, SIFT and ORB for all the datasets.

Feature Matching

False positives and false negatives of landmarks are fundamentally produced through feature matching. Graph analyzing the feature matching for each feature extractor is presented in Appendix B. SIFT produced the lowest number of matched features while ORB produced the highest number. In datasets 6 and 7 all three feature extraction algorithms recorded lower numbers of matched features compared to the other datasets. This is to be expected as datasets 6 and 7 have blurry images where it is more difficult to extract features. Hence, this had a cascade effect where landmarks were not tracked accurately which resulted in higher number of landmarks and failed recalls of landmarks in the database. An example of this detecting a ROI in a blurry image is shown in Figure 5.5 where fewer features are extracted and matched. In general ORB extracted more features compared to SURF and SIFT considerably.

SIFT produced the lowest average matched features for all the datasets, it also produced the lowest average of false features matched. As expected ORB produced the highest average of false features matched.

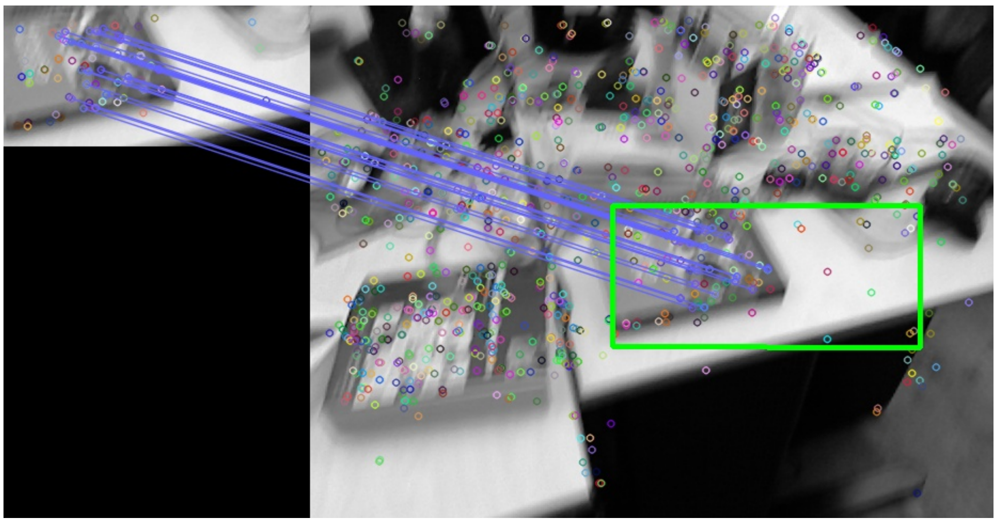


Figure 5.5: An example of a ROI detected in a blurry image which detects fewer features and matches.

Duration

The execution time of each feature extraction algorithm was recorded for all the databases and is shown in Figure 5.6. The duration measurements presented in this section are not averages as the datasets tests were only executed once.

From these tests it was found that if there is a high number of features extracted the execution would be longer and if there is low number of features extracted then the execution time is shorter. However, dataset 6 stands out from the rest as SURF recorded fewer features matched compared to ORB, but has the highest execution time. In this case due some of the images being blurry, SURF took longer to extract features compared to SIFT and ORB. SIFT took longer than ORB to extract features in dataset 6. With the exception of dataset 6, SIFT was fastest and ORB was the slowest in terms of execution time.

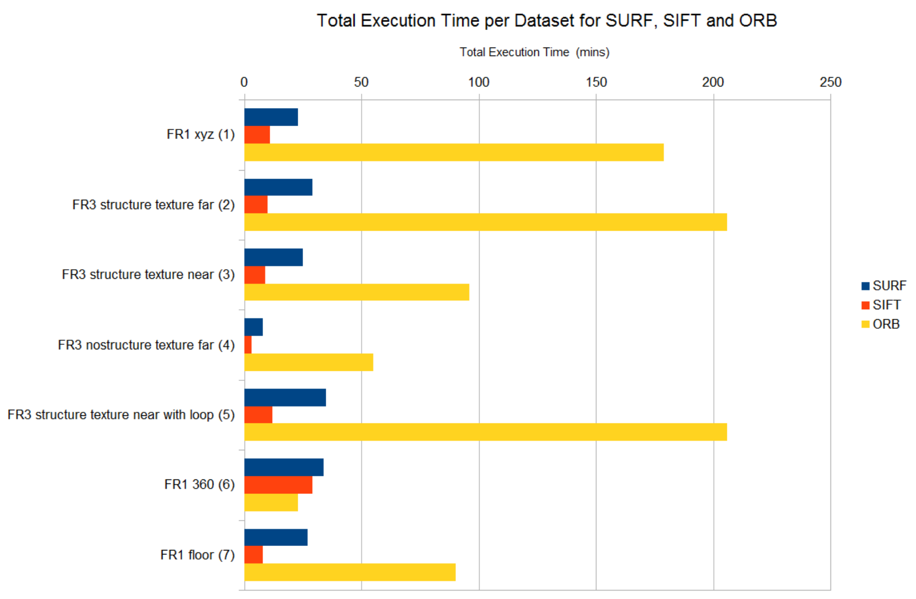


Figure 5.6: Graph of duration for SURF, SIFT and ORB for all the datasets.

5.1.4 Summary of Results from Dataset Tests

In these tests the pre-processing pipeline was tested in isolation before it was integrated into the RGBD SLAM framework. The primary focus of these tests was to test the functionality of the pre-processing pipeline to be able to accurately identify and track landmarks in an environment. The first criterion was to see if the pipeline is able to build a database of landmarks, the second was to determine the precision of re-identifying a ROI in the database and the last criterion is to check if the pipeline can recall a landmark stored in the database accurately. Therefore three sets of measurements which were used to fulfil these criteria. The three sets of measurements are the number of landmarks in the database, false positives of landmarks and number of false negatives of landmarks in the database.

Overall the pipeline was able to successfully build a database of landmarks with all the feature extractors. SURF and SIFT were found be better in re-identifying the ROI compared to ORB. It was also observed that when there were blurry images (dataset 6 and 7) the extractors have difficulties in re-identifying ROI. This makes the pipeline produce more false positives and false negatives of landmarks in the database.

Each of the feature algorithms was analysed individually in terms of number of features matched, number of false features matched and execution time for all the datasets. ORB resulted with the most number of feature matches and false matches, hence it required a longer time to execute. SIFT appeared to outperform SURF and ORB in producing the least numbers of matched features and false matching of features with a short execution time.

However, these results are not comparable as each feature extractor extracted different numbers of features. ORB tended to extract the most number of features, hence it produce the most number of feature matches, false feature matches and false negatives of landmarks. SIFT resulted with few features which produced lower number of feature matches, false feature matches and false negatives of landmarks.

The pipeline was shown to be effective in identifying and tracking landmarks for SLAM with all the feature extractors. The next sets of tests integrated the pipeline with RGBD SLAM framework. However to improve the performance and to produce more comparable results the number features for all the features extractors was set to the same number of 600 in the next set of tests. This number was chosen as the traditional method used in RGBD SLAM had a default value of 600. This test also revealed that to reduce the number of false positives and false negatives of landmarks the ROI algorithm should be altered to detect a larger ROI. In the next test this change was made to the pipeline.

5.2 Test 2: RGBD SLAM Testing

The previous tests investigated the pre-processing pipeline in isolation in terms of robustness and performance to ensure landmarks can be identified and tracked for SLAM. Traditionally RGBD SLAM uses the method of extracting features from the entire image forming a distribution of features. The pipeline only extracts features from the ROI. In these tests the pipeline was integrated into the RGBD SLAM framework. Once it was integrated, it was compared to the traditional version of RGBD SLAM to determine whether the pipeline would improve the accuracy and computational time of SLAM. Details of the integration of the pipeline in the RGBD SLAM framework are given in Chapter 3. This comparison is made to see if the strategy of only extracting features from the ROI can outperform the traditional method (distribution of features) used in RGBD SLAM in terms of accuracy in robot's position and computational time.

From the pre-dataset tests five datasets were chosen to be used in these tests which were dataset 1, 2, 4, 6 and 7. An additional dataset was also used to test the limits of the pipeline in terms of robustness and performance. This dataset is named "fr3/nostructure _notexture" and labelled as dataset 8 for this study. In this dataset a featureless environment was captured which contained a plain white wall with no objects of detail. A sample image from this dataset is shown in Figure 5.7.

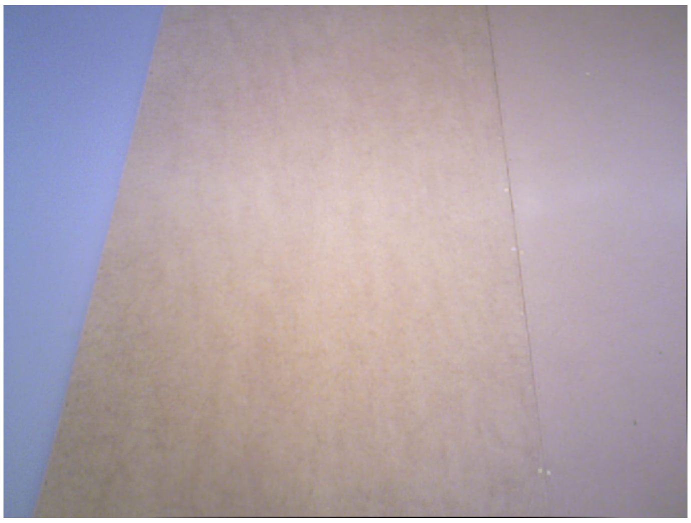


Figure 5.7: Sample image from dataset 8.

From the previous test it was found that to improve the accuracy of identifying and tracking a landmark the ROI algorithm should be adjusted to detect larger ROIs. To allow the pipeline to detect a larger ROI the number of iterations for the dilation operation was reduced to two. In these tests both sizes of ROIs are integrated into the RGBD SLAM and tested. In the tests the previous ROI is known a "RGBD_Small_ROI" and the new bigger ROI is known as "RGBD_Big_ROI". The results from both versions of RGBD SLAM are compared to the original RGBD SLAM to see if the performance of SLAM has improved with pipeline. All versions of RGBD SLAM used all three feature extractors. To ensure all the results are comparable all the feature extractors have been set to extract the same number of features for matching and tracking of a landmark. The number was set to 600 features in these test as this was the default value used in RGBD SLAM.

The results produced error values in localising the robot's position. If the landmarks were accurately identified and tracked then these error values are reduced. If the landmarks were not accurately identified and tracked then these error values increase. Therefore, these error values were used to determine whether landmarks are identified and tracked accurately. The time taken for each test was also recorded to evaluate the performance of all three versions of SLAM in terms of computational time.

As discussed in Chapter 4, RGBD SLAM was developed in ROS and it is also executed in ROS. The datasets were entered into RGBD SLAM and were processed one at a time. By using the datasets RGBD SLAM is executed off-line meaning that data entered into

RGBD SLAM simulated a robot navigating through the environment. Each dataset was processed by all three versions of RGBD SLAM and for each version of RGBD SLAM all three feature extractors were used. After each dataset was processed an output file is created containing the error in the trajectory, the duration and the error in the pose of the robot (robot's position) in both translation and rotation. This file was analysed for all the datasets and compared.

5.2.1 Results from RGBD SLAM

The results from all tests were compared in terms of speed and accurate estimation of the robot's trajectory. All tests were conducted using the RGBD SLAM software that was developed by Endres et al [13]. The software was executed on an Ubuntu 14.3 platform using ROS. The software outputs were a 3D map with the robot's trajectory and a log that documents the processes of RGBD SLAM. In Figure 5.8 the 3D map that is created by RGBD SLAM is shown. The computer that was used in these tests had a Intel Core i7 and 8 Gb of RAM.

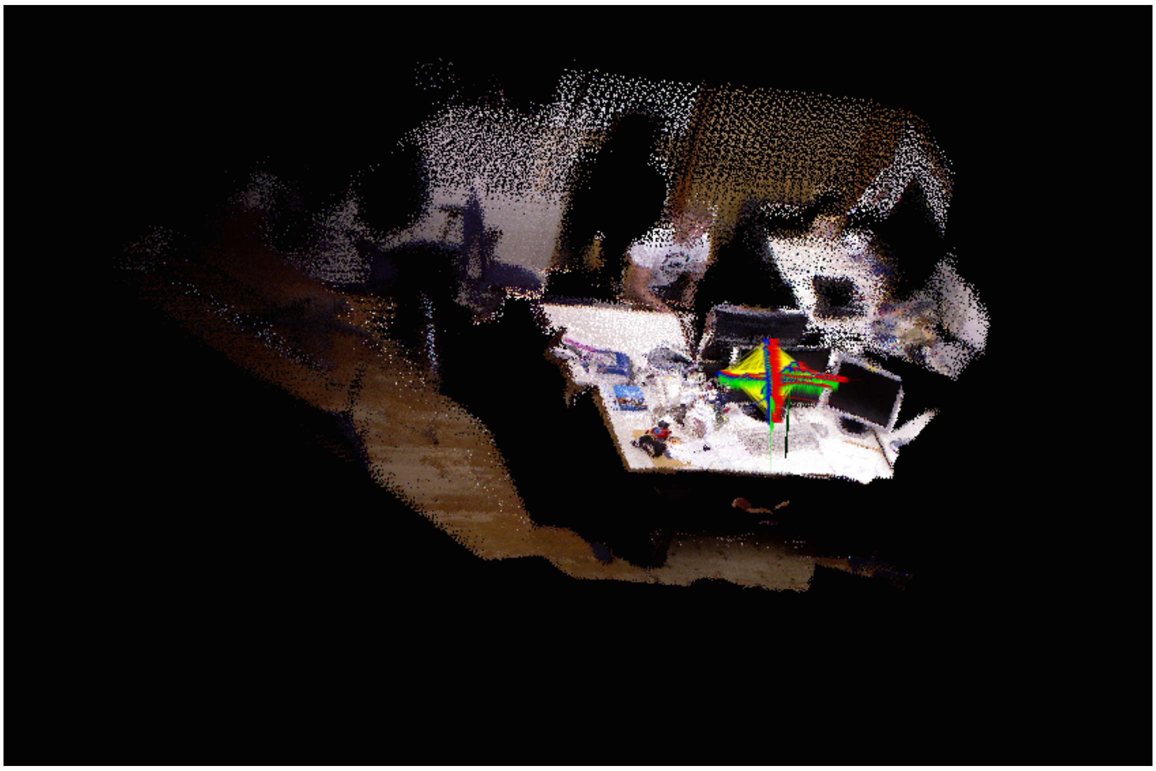


Figure 5.8: A 3D map with the robot's trajectory that is produced from RGBD SLAM.

A post-processing evaluation tool developed by Endres et al. was used on the outputs from RGBD SLAM [13]. The tool produced a spreadsheet containing the error values of the robots trajectory with duration. To calculate the error values the tool read in the ground truth of the robot and compared it to the estimated trajectory of the robot. The ground truth was supplied with each dataset that was downloaded and was captured with a motion-capture sensor. There were three error values of the robot's position that are calculated namely the absolute trajectory error, the translational and the rotational root mean square error (RMSE).

The absolute trajectory error is the difference in the estimated robot's trajectory and the ground truth of the robot. The translational and rotational error is the relative pose error of the robot. These error values are analysed to determine the accuracy of identifying and tracking landmarks in the environment. This can be done as the SLAM algorithm uses the landmarks to estimate the robot's position. If the landmarks are not tracked accurately, the error values of the robot's position will be larger but if the landmarks are tracked accurately then the error values would be minimised. It can also be deduced that if the landmarks are tracked accurately the number of false positives and false negatives of landmarks is reduced. This would also ensure that loop closure in SLAM is conducted correctly.

After each test the results are analysed and compared for all three versions of RGBD SLAM. This comparison is used to evaluate the performance and to see whether the integration of the ROIs improves the computational time of SLAM and the accuracy of SLAM in determining the robot's position. Graphs analysing the absolute error and relative pose error was produced for each feature extractor. Only graphs for SURF are shown in this section and the graphs for SIFT and ORB are presented in Appendix B.

Absolute Trajectory Error

This section discusses the absolute trajectory error of all three versions of RGBD SLAM for each feature extractor. During the tests it was found that as the environment and movements of the robot in the datasets became more complex the absolute trajectory error increased. For dataset 1 and 2 the results for all three versions of RGBD SLAM using all three feature extractors were found to produce similar error values.

SURF

In Figure 5.9 a graph of the absolute error of the robot for all the datasets using the SURF feature extractor is shown. For all the iterations across all the datasets, the absolute trajectory error never exceeded 0.4 m. The SURF feature extractor produced similar results for all the three versions of RGBD SLAM for dataset 1 and 2. For the other datasets the use of a bigger ROI outperformed the other two versions of RGBD SLAM. This is to due SURF being more precise in re-identifying the ROI with a bigger ROI. In the last dataset (8) the smaller ROI completely failed to detect landmarks therefore the pose of the robot could not be found. This was to be expected as the dataset contained a featureless environment. The bigger ROI was able to capture enough features to track landmarks for RGBD SLAM to evaluate the pose of the robot for the last dataset. From these results when the bigger ROI is used in RGBD SLAM with SURF, it was found that this version of RGBD SLAM outperforms or performs as well as the traditional version in estimating the trajectory of the robot.

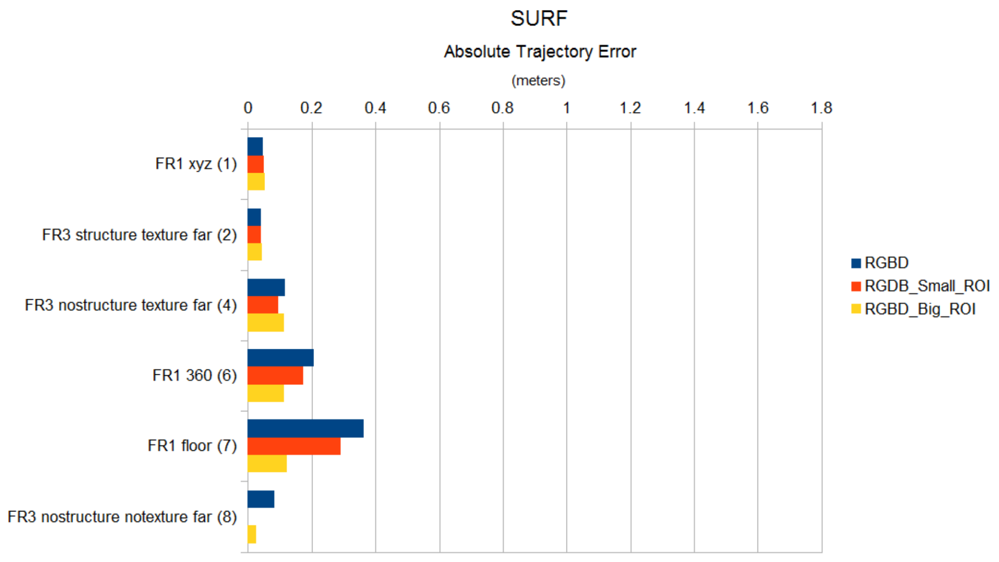


Figure 5.9: Absolute trajectory error of the robot for all datasets using the SURF feature extractor.

SIFT

SIFT produced similar results to SURF because SIFT extracts features in a similar manner to SURF. For datasets 4, 6 and 7 the smaller ROI struggles to track landmarks using the SIFT extractor, but performs better with the bigger ROI. In dataset 4 SIFT produced the largest error with the smaller ROI compared to the rest. This is due to the SIFT extractor struggling to re-identify landmarks accurately. Hence determining the pose robot is more difficult which results in a larger trajectory error. For the last dataset SIFT failed with the smaller ROI and produced a larger error using the bigger ROI. Similar to the results of SURF, the version of SLAM using the bigger ROI outperformed or performed just as well as the original version of SLAM in estimating the trajectory of the robot.

ORB

Although ORB produces similar results to SIFT and SURF for datasets 1 and 2, but for the rest of the datasets it performs much differently. In datasets 4 and 7, the normal RGBD SLAM outperforms the ROIs. This was expected as in the pre-dataset tests ORB was found to produce more imprecise results with ROIs compared to SURF and SIFT; and therefore it struggled to find landmarks, even though in these tests all three feature extractors extracted the same number of features. However for dataset 6 the bigger ROI using ORB outperformed the normal RGBD SLAM by a small margin. The last dataset ORB completely failed to track landmarks with ROIs. In these results traditional SLAM was clearly observed to perform better than using ROI with ORB in estimating the robot's trajectory.

Summary

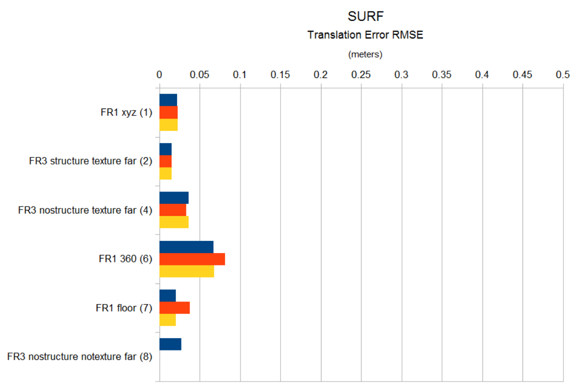
In summary the RGBD SLAM using the bigger ROI outperformed or performed just as well as the traditional RGBD SLAM in minimising the absolute trajectory error. SURF was found to produce the least amount of error when used with ROIs and the error further reduced when the bigger ROI. ORB produced the greatest error with ROIs and least amount of error with the traditional RGBD SLAM.

Relative Pose Error

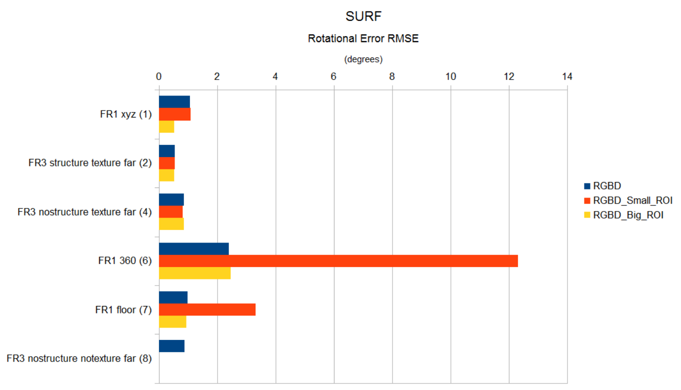
The relative pose error is determined by calculating the translation and rotational RMSE of the robot's pose between each step the robot moves in the environment. This is used to estimate the drift of the robot when comparing the estimated trajectory to the ground truth trajectory. The translation error considers movement in the x, y and z plane, and the rotational error considers movement in roll, pitch and yaw. Similar to absolute trajectory error the relative pose error was observed to increase when the environment and the robot's movement became more complex. The use of smaller ROI yielded larger error values. This is due to again that the smaller ROI struggling to accurately re-identify the landmarks and hence it is not able to correctly determine the pose of the robot. This section presents the relative pose error for all three versions of RGBD SLAM for each feature extractor.

SURF

Graphs of the translation and rotational RMSE for all the datasets using the SURF feature extractor are shown in Figure 5.10. In dataset 1 and 2, the translation error for all versions of RGBD SLAM is similar which is an expected outcome as it was seen that the error in trajectory for SURF was also found to be similar for dataset 1 and 2 in the last section. However, the rotational error for the version of SLAM using the bigger ROI was smaller than the other two versions. A reason for this is that the bigger ROI is able capture more reliable features compared to the versions of RGBD SLAM. This allows the bigger ROI to be more precise in re-identifying a ROI to be able to track landmarks accurately.



(a) Translations error.



(b) Rotational error.

Figure 5.10: Graphs of Translational and Rotational RMSE for all the datasets using the SURF feature extractor.

In dataset 4 it was found that SURF produced a lower translation and rotational error with the smaller ROI. For datasets 6 and 7 the smaller ROI produced the largest errors compared to the other iterations of RGBD SLAM. For the last dataset that used SURF with bigger ROI, the translation and rotational error was not recorded even though the trajectory error was recorded as shown in the previous section. This indicates that the bigger ROI failed in the last dataset as it was not able to detect a ROI that contained a set of reliable features to determine the pose of the robot.

Besides from dataset 4 and 8, when SURF was used the RGBD SLAM version using the bigger ROI outperformed or performed just as well in determining the pose of the robot with less drift as compared to the version of SLAM using the smaller ROI.

SIFT

For dataset 1 and 2 the original version of RGBD SLAM and the versions using ROIs produced similar results as SIFT for both translational and rotational error. Therefore all three versions of SLAM experience the same amount of drift in determining the robot's pose.

In datasets 4, 6, and 7 the smaller ROI using SIFT produced the largest errors in translation and rotation implying the smaller ROI induced more drift compared to the other two versions of SLAM. This also indicated the smaller ROI captured fewer reliable features to accurately identify and track landmarks using ROIs. For dataset 8 although the bigger ROI was able to find the pose of the robot with SIFT, but the original version of RGBD SLAM produced smaller errors.

In all the datasets except for dataset 8, the bigger ROI outperformed or performed just as well as the original version of RGBD SLAM in determining the pose of the robot while incurring the least amount of drift.

ORB

For all datasets the smaller ROI produces more errors in translation and rotations compared to the two versions of RGBD SLAM using the ORB feature extractor. Dataset 4 was the only case where the original version of RGBD SLAM performed better than the ROI versions when ORB was used for determining the pose of the robot. It was expected for the original version of SLAM using ORB to perform better in all the datasets as seen in the last section when the robot's trajectory was estimated.

All the datasets except for dataset 4 showed that the bigger ROI and the original version of SLAM experience similar amounts of drift which is less than the drift incurred with the smaller ROI. Thus far it was seen that ORB performs better when used in the original RGBD SLAM in determining the robots position compared to when it is used with the ROI versions.

Summary

In most cases the use of a bigger ROI performed similarly to the original RGBD SLAM and in some cases the traditional RGBD SLAM marginally outperformed in terms of the translational and rotational errors. Again this variation in results is dependant on which feature extractor is used with the versions of RGBD SLAM. SURF produced lower errors in the robot's pose using the bigger ROI while SIFT predominantly produced lower errors when used in the original version. ORB yet again was found to perform better when used in the original RGBD SLAM, which further proves ORB struggles to re-identifying the landmarks when the pipeline is included into the RGBD SLAM framework.

Duration

For all the versions of RGBD SLAM with the three feature extractors the duration of the time taken to execute is recorded. The last section compared all three versions of RGBD SLAM in terms of determining the robot's position accurately. This section evaluated the versions in terms of computational time to determine whether the ROI versions are faster than the traditional version. This comparison was done for all three feature extractors. Most datasets took less than 1000 s to execute where the normal RGBD SLAM took the shortest time.

SURF

In Figure 5.11 a graph of the duration of RGBD SLAM for all the datasets using the SURF feature extractor is shown. All the datasets took less than 1000 s to execute, however dataset 2 with a smaller ROI took the longest where it surpassed 1000 s. Overall the datasets with the original version of RGBD SLAM took the shortest amount time. The results show that the ROI versions take longer as more time is required to first re-identify a ROI accurately and then track the landmarks accurately to determine the robot's position using SURF.

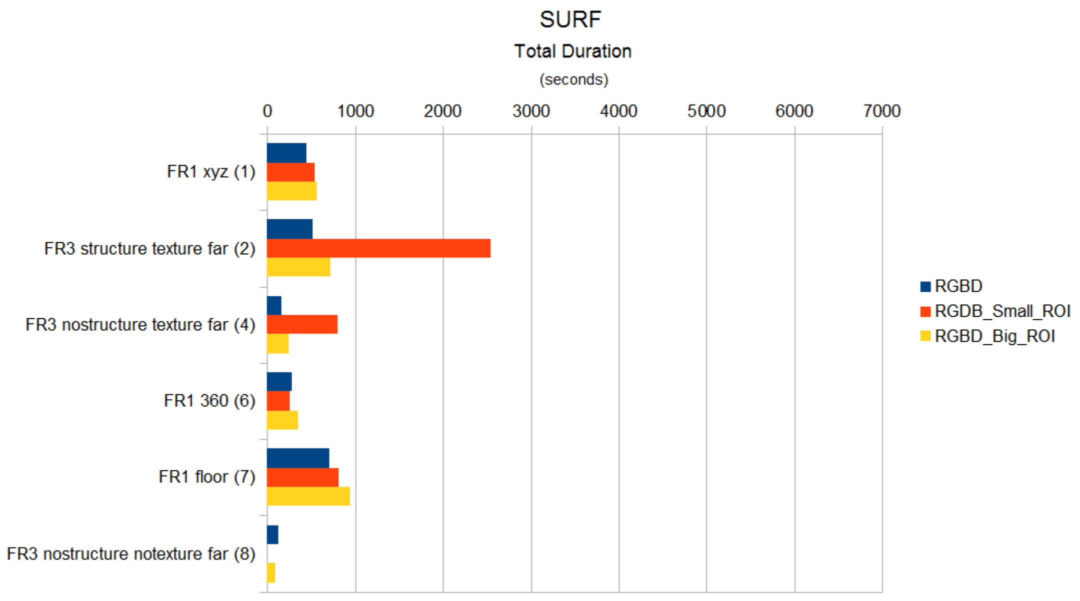


Figure 5.11: Duration of RGBD SLAM for all datasets using the SURF feature extractor.

Datasets 2 and 4 takes the longest with the smaller ROI. This indicated that the smaller ROI captures features that are less reliable than the bigger ROI, therefore it took longer to re-identify landmarks to determine the position of the robot. In dataset 8 the bigger ROI took less time than the original version of RGBD SLAM, but it was due to SURF not being able to capture enough features to accurately identify and track landmarks. Hence, the robot's position was not found, which reduces the computational time it needed.

Although SURF produced the most accurate results with the bigger ROI in determining the robot's position, it took longer than the original version of RGBD SLAM in all the datasets (except for 8).

SIFT

A graph of the time taken of RGBD SLAM for all the datasets using the SIFT feature extractor is shown in Figure 5.12. All the datasets using the SIFT feature extractor took less than 1000 s to execute. From the results of SIFT, the original RGBD SLAM took the shortest time while the bigger ROI took the longest time to execute, while providing the most accurate results in determining the pose of the robot.

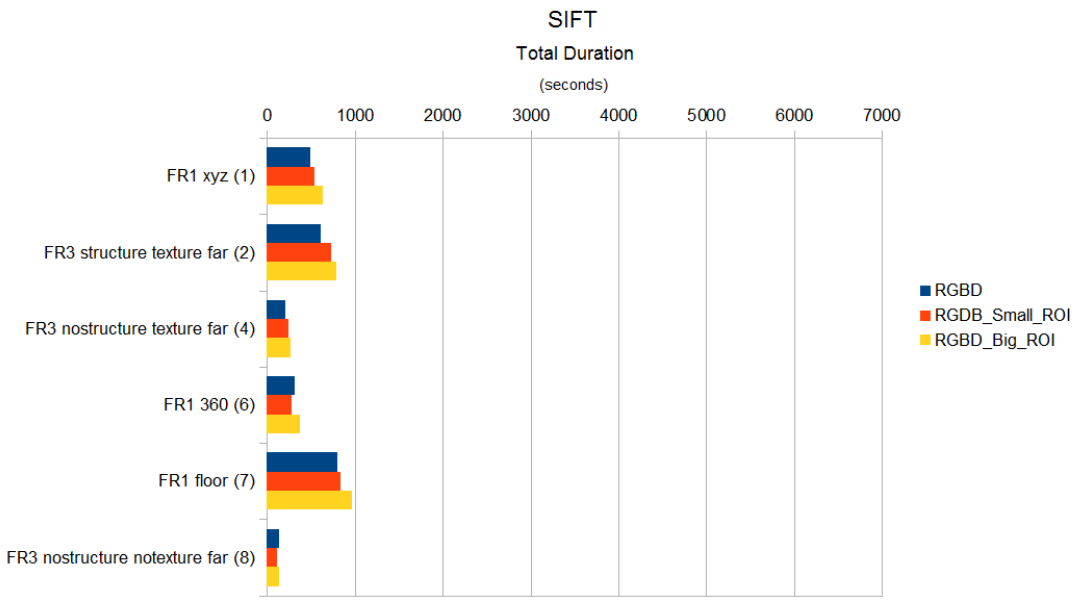


Figure 5.12: Duration of RGBD SLAM for all datasets using the SIFT feature extractor.

The only two cases where the smaller ROI took the shortest amount of time was for datasets 6 and 8. However in dataset 6 there was a greater error in the position of the robot and in dataset 8 the smaller ROI completely failed to determine the position of the robot. For dataset 6 there were less reliable features captured with the smaller ROI which shortened the computational time but incurred more error in determining the position of the robot. In dataset 8 the smaller ROI and bigger ROI could not capture any reliable features, which caused the system to completely fail.

Similar to the results observed in SURF, SIFT produced the most accurate results in finding the robot's position with the bigger ROI, but it required more computational time.

ORB

Figure 5.13 shows a graph of the duration of RGDB SLAM using the ORB extractor. From the results from all the datasets, it was found the normal RGBD SLAM took as long as the smaller ROI. The bigger ROI took the longest as more reliable features were captured and it required more computational time to re-identify and track landmarks accurately to determine the position of the robot. Dataset 2 was the only dataset where the normal RGBD SLAM took the longest. Compared to SURF and SIFT, ORB recorded the highest computational time surpassing 6000 s which was for dataset 7.

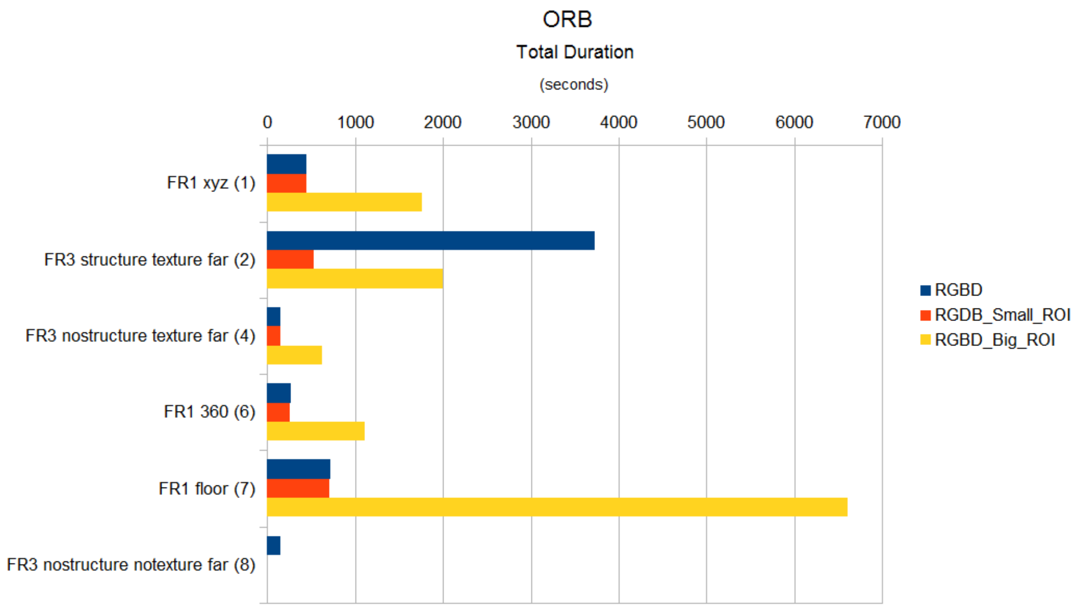


Figure 5.13: Duration of RGBD SLAM for all datasets using the ORB feature extractor.

In dataset 2 smaller ROI was found to take the least amount of time and the normal RGBD SLAM took the longest amount of time. However in this case the smaller ROI incurred more error and this is the only instance where the traditional RGBD SLAM took longer than the ROI versions of SLAM. This indicated that in this case the bigger ROI captured more reliable features compared to the normal RGBD SLAM. Dataset 7 was recorded to take the longest computational time across all the tests. In this dataset as seen in the previous tests with SIFT, the smaller ROI captured less reliable features compared to the bigger ROI which required less computational time and incurred more error in determining the robot's position. For dataset 8 ORB completely failed to detect any features and therefore there was no computational time recorded.

As observed in the previous test with SURF and SIFT the bigger ROI required more computational time, but incurred less error compared to the smaller ROI. Dataset 2 was the only anomaly where the traditional version of RGBD SLAM took longer than the ROI versions.

Summary

In summary the ROI versions of RGBD SLAM for all the feature extractors require additional time to traditional RGBD SLAM. This was due to the additional time required for the detection of the ROI. This was an unexpected result as even though there is an additional step of detecting a ROI, identifying and tracking of landmarks would be optimized and should have reduced the computational time.

5.2.2 Summary of RGBD SLAM Tests

In the previous tests the pre-processing pipeline was tested in isolation to test the robustness of the functionalities of the pipeline to be able to identify and track landmarks accurately. In this set of tests, the pipeline was embedded into the RGBD SLAM framework and compared to the traditional version of RGBD SLAM. In the comparisons both versions were analysed to determine if the modified version is able still to produce more accurate results of the position of the robot while requiring less computational time.

From the previous test it was found that the ROI area needed to be expanded to allow the feature extractors to capture more reliable features to accurate identify and track the landmarks. The previous size of the ROI was still used in the these tests but was called 'RGBD_Small_ROI' and the expanded ROI was called 'RGBD_Big_ROI'. All three versions also used SURF, SIFT and ORB feature extractors to identify and track the landmarks in the environment. Five datasets from the pre-dataset test (dataset 1, 2, 4, 6, and 7) and a featureless dataset (dataset 8) was used as inputs to these three versions of SLAM.

To determine which version of RGBD SLAM performed the best in terms or accuracy of robot's position and computational time three sets of measurements are used. The first two sets of measurements that were recorded was the error robot's trajectory (absolute trajectory error) and pose (relative pose error). Ground truth is used to calculate the errors. These errors were analysed to reflect the accuracy for each version of RGBD

SLAM in identifying and tracking the landmarks. The last set of measurements was the duration that version of RGDB SLAM required to process each dataset. This was used to determine which version took the least amount of computational time.

It was observed, in most cases the smaller ROI version performed the worst in terms accuracy of the robot's position and time. This performance was seen in all three feature extractors used in these tests. It was found the reason the smaller ROI produced poor results compared to the other two versions of RGBD SLAM, is due to the size of the ROI. This was expected as in the previous test it was found that the smaller ROI led to fewer reliable features being extracted. With fewer reliable features the landmarks are not identified and tracked accurately which lead to larger errors in determining the robot's position. In addition with fewer features to process, the computational time is shorter. This was seen in these tests as the smaller ROI had larger errors in the trajectory and pose of the robot while requiring less time.

When the SURF feature extractor was used in all the versions of RGBD SLAM it was found the that bigger ROI produced the most accurate results in determining the trajectory and pose of the robot. However, it required more computational time. This was expected as the bigger ROI led to the feature extractor capturing more reliable features to be able to identify and track landmarks accurately. Hence in determining position the of the robot the errors were minimized. Due to more uniquely identifiable features being detected more time is required to process the features.

When SIFT and ORB feature extractors were used the original version was seen to produce the most accurate results for the robot's position. There were some cases where the bigger ROI performed just as well as the traditional version of RGBD SLAM in terms of finding the robot's position but the bigger ROI required a longer time.

For the last dataset that was used in these tests, it was seen that for both versions of the modified RGBD SLAM (Small_ROI and Big_ROI) the tests failed. The last dataset contained images of a featureless environment hence the modified versions of RGBD SLAM failed as the pre-processing pipeline failed to create ROIs. ROIs are designed to be created when there are highly detailed areas in the image that could contain uniquely reliable features for the feature extractor to capture. If there are no ROIs then there are no features captured and due to this, landmarks could not be identified and tracked which led to the robot's position undefined. When the robot's position is undefined then the RGBD SLAM failed as seen when the ROI versions are tested with the last dataset. However, the original version of RGBD SLAM without the ROI was found that it could detect reliable features that was used to identify and track landmarks to determine the position of the robot. The original version of RGBD SLAM used the traditional method where features are captured from the whole image which led to a more distributed set of reliable features being captured to identify and track landmarks. Thus, the original version of RGBD SLAM successfully executed and was able to determine the position of the robot.

The best set of results recorded from theses tests was when SURF was used in the bigger ROI version of RGBD SLAM where the robot's position was determined with minimized error. This outcome indicates that with the SURF feature extractor with the strategy of extracting features from a ROI seems to produce better results than the traditional method (distribution of features) of extracting from the entire image for SLAM.

Due to this result all three versions of RGBD SLAM in the next set of tests are tested again with the same datasets with using only SURF as the feature extractor and reducing the number of reliable features than can be extracted from 600 to 50. The features are reduced to 50 to simulate an environment with sparse feature where only 50 features or less can be extracted. These tests are used to observe if the bigger ROI version of SLAM can still outperform the original version of SLAM in terms of the accuracy of the robot's position and computational time. In addition the results will also indicate if the strategy of extracting features from ROI will still outperform the traditional method with a reduced number of features for SLAM.

5.3 Test 3: Reduced Feature Tests

The first two sections of this chapter presents the results from testing the pipeline in isolation and thereafter integrating the pipeline into the RGBD SLAM framework respectively.

In this section the previous test is conducted again using the same datasets for all versions of RGBD SLAM using only the SURF extractor and reducing the number of features captured to 50. The results recorded in these tests are analysed to determine whether bigger ROI version of RGBD SLAM can still outperform the smaller ROI and original version. This will also verify if the ROI strategy via the pipeline is a better method to extract features to identify and track landmarks for RGBD SLAM compared to the traditional method.

In the analysis the results recorded from the previous test for all three version of RGBD

SLAM using SURF that has a limit of 600 features will be compared to results from reduced feature tests. The tests with 600 SURF features are named 'SURF_600' and the tests with reduced number of 50 features are named 'SURF_50'. Again the performance of each version will be analysed based on the accuracy of the robot's position and the computational time required to execute.

5.3.1 Comparison of Reduced Feature Tests

As shown in the previous section three sets of measure are recorded from all three versions of RGBD SLAM to analyse the performance of each version. The three sets of measure recorded were the error in the absolute trajectory of the robot, error in the relative pose of the robot (translation and rotation) and the duration of time required for each version to execute. The absolute trajectory and relative pose error of the robot are analysed to determine which version of RGBD SLAM produced the most accurate position of the robot.

Absolute Trajectory Error

The error in the absolute trajectory is the difference of the estimated trajectory of the robot and the ground truth of the robot. In Figures 5.14 and 5.15 the graph of the absolute error recorded for all versions of RGBD SLAM for all the datasets from the SURF_600 and SURF_50 tests are shown respectively. It was observed in Figure 5.15 that for all version of RGBD SLAM the absolute error using reduced features (SURF_50) are larger than the errors recorded from the SURF_600 tests as seen in Figure 5.14. This result is expected as there are fewer reliable features captured with reduced features which cause the landmarks not being identified and tracked accurately. With this effect, the robot's position is not accurately determined hence a larger error in the absolute trajectory of the robot.

In the SURF_600 tests it was clearly shown that the bigger ROI version performed just as well or outperformed the traditional version of RGBD SLAM across all the datasets. However in the SURF_50 for dataset 1, 2 and 4 the original version of RGBD SLAM produced the lowest error in absolute trajectory, and for dataset 6 and 7 the bigger version produced the lowest error marginally compared to the original version. Dataset 1, 2 and 4 contain environments less complex with slower movements of the robot compared with the environments in dataset 6 and 7. It seems that with datasets that contain simpler and slower movements of the robot that the original version of SLAM is able to capture more reliable features to accurately identify and track the landmarks which produce the most accurate trajectory of the robot. But in datasets that contained more complex environments where the robot is moving faster the bigger ROI version of RGBD SLAM captures more reliable features. Due to the robot moving faster blurry images are captured. The bigger ROI version was able to capture more reliable features in these blurry images compared to the traditional version, as the ROI is used help in finding detailed areas in a blurry image that could contain more reliable features.

For the last dataset the SURF_50, the original version of SLAM was the only one to be able process the dataset to be able yield a result. Both ROI versions of RGBD SLAM failed to capture any reliable features to be able to determine the trajectory of the robot. Due this dataset containing a featureless environment capturing reliable features is a difficult task and therefore the both the modified versions of SLAM was not able to detect a ROI where features could be captured. However, the original version of RGBD SLAM was able to capture features as it extracts features from the entire image.

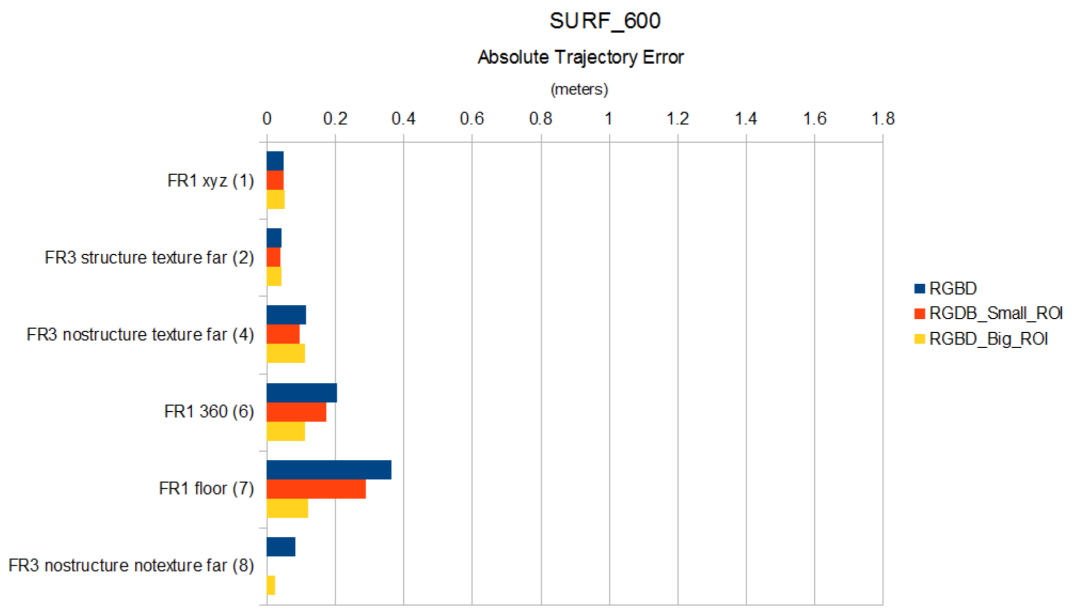


Figure 5.14: Absolute trajectory error of the robot for all datasets using the 600 SURF features.

Relative Pose Error

The error in the relative pose of the robot is the difference in the translation and rotational pose compared to ground truth at each step of the robot. In Figures 5.16 and 5.17 the graph of the translational and rotational error of the robot for all the datasets in the

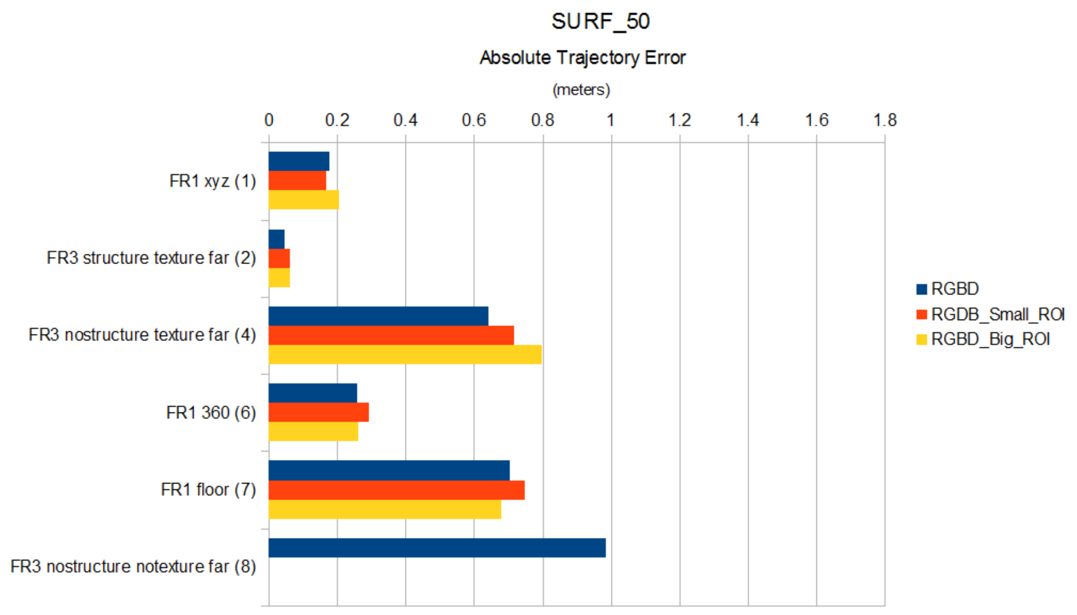


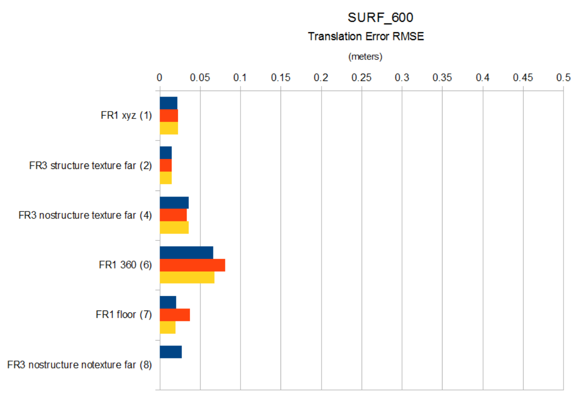
Figure 5.15: Absolute trajectory error of the robot for all datasets using the 50 SURF features.

SURF_600 and SURF_50 tests are shown respectively. As expected the translational and rotational error of the robot shown in Figure 5.17 are larger than the errors recorded from the SURF_600 tests as seen in Figure 5.14. Due to SURF_50 tests capturing fewer features the translation and rotational pose of the robot incur more errors as the landmarks cannot be identified and tracked accurately for all versions of RGBD SLAM.

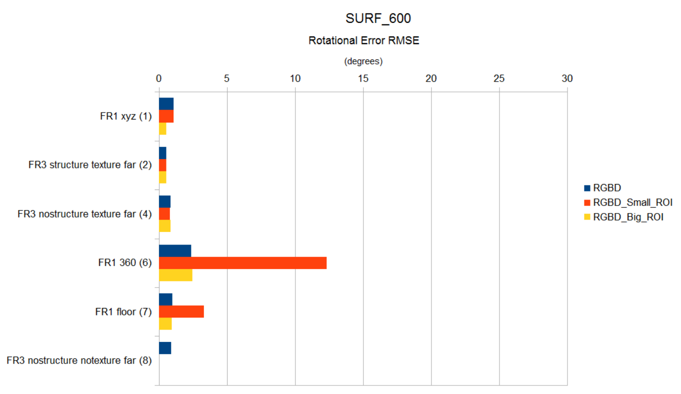
For both the tests the bigger ROI version of RGBD SLAM performed the best in estimating the robot's pose in producing the lowest error recorded for translation and rotation of the robot across all the datasets. These results show that with the number of features reduced the bigger ROI version is still able to capture reliable features to estimate the pose robot accurately compared to the other two versions of RGBD SLAM. As mentioned earlier in this section the traditional version of RGBD SLAM was the only version that was able to process the last dataset and yield a result.

Duration

The last measurement that was analysed was the duration for each version of RGBD SLAM to execute for both the SURF_600 and SURF_50 tests. In the Figures 5.18 and 5.19 the graphs of the time taken for each version of RGBD SLAM for all datasets for the SURF_600 and SURF_50 test is shown respectively. The duration recorded for the SURF_50 test was lower as seen in Figure 5.19 compared to the SURF_600 test shown in

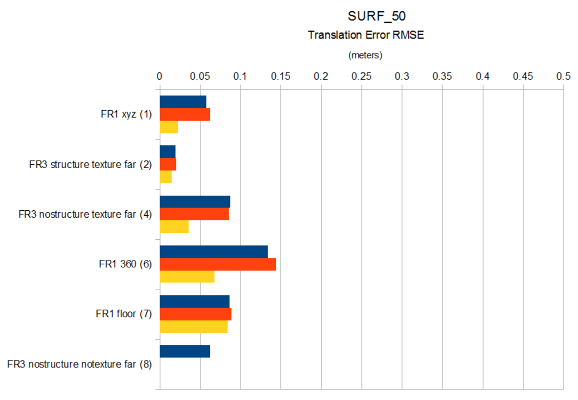


(a) Translations error.

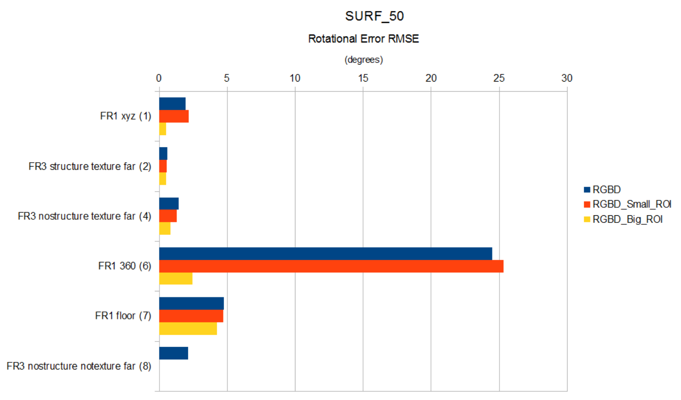


(b) Rotational error.

Figure 5.16: Graphs of Translational and Rotational RMSE for all the datasets using 600 SURF features.



(a) Translations error.



(b) Rotational error.

Figure 5.17: Graphs of Translational and Rotational RMSE for all the datasets using 50 SURF features.

Figure 5.18. This outcome was expected as the SURF_50 tests captured fewer features which required less computational to process compared to the SURF_600 tests where more features are captured and more time is needed to process them.

The bigger ROI version of RGBD SLAM recorded similar duration times to execute compared to the traditional version across all the datasets for the SURF_50 tests. This indicates that the computational time required to capture features from a ROI to track and identify landmarks for SLAM to accurately determine the position of the robot is similar to the traditional method used in the traditional version, where features are captured from the entire image for SLAM to determine the position of the robot. The original version of RGBD SLAM was the only version that was able to process the last dataset and record the time taken to execute whereas the modified version failed to process this dataset and therefore the duration was not recorded.

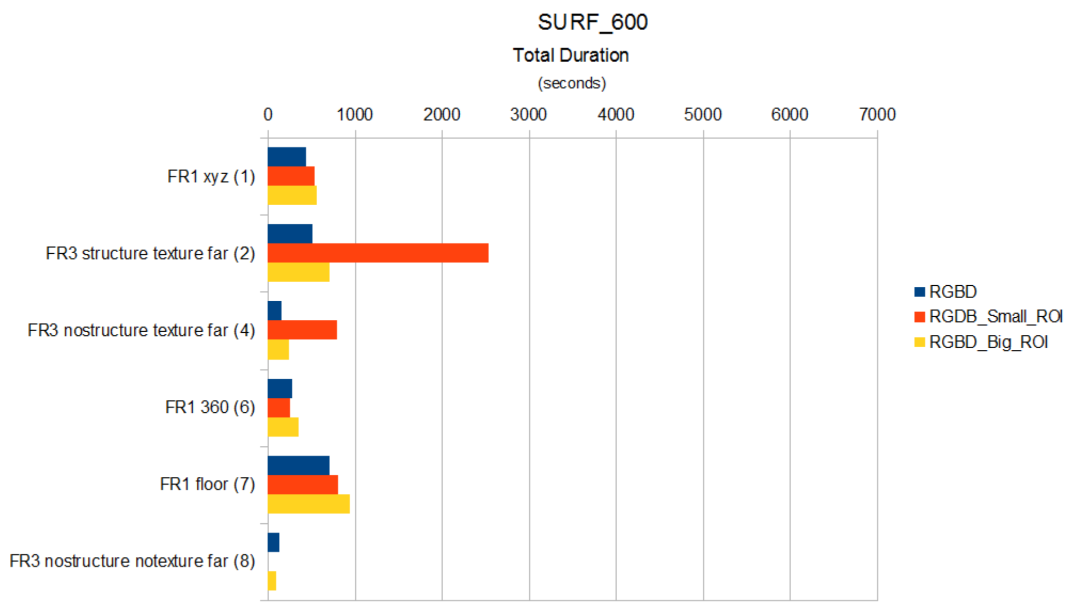


Figure 5.18: Duration of RGBD SLAM for all datasets using 600 SURF features.

5.3.2 Summary of the Reduced Feature Tests

The last section presented the results from the integration of the pipeline into the RGBD SLAM framework where the number of features captured was limited to 600. The results showed that the SURF feature extractor with the bigger ROI version of RGBD SLAM produced the most accurate results in determining the position of the robot. This section conducts these tests again, but only using the SURF extractor with all the three versions of RGBD SLAM and further reducing the number of features that can be captured to 50. The results from this section (SURF_50) are compared to the results from the last section when SURF was used (SURF_600). This comparison is used to determine if the strategy of only extracting features from a ROI to track landmarks for RGBD SLAM will improve the accuracy of the position robot and the computational time required compared to the traditional method used in the traditional version of RGBD SLAM where features are extracted from the entire image.

In the analysis three sets of measurements are used to determine which version of RGBD SLAM performs the best in accurately finding the robot's position while requiring the least amount of computational time. The three sets of measurements were the error in the absolute trajectory of the robot, error in the pose of the robot (translational and rotational) and the duration for each version of RGBD SLAM to execute. If these two sets of errors are minimized, this indicates that reliable features have been captured to accurately identify and track the landmarks for RGBD SLAM to produce the most accurate position of the robot.

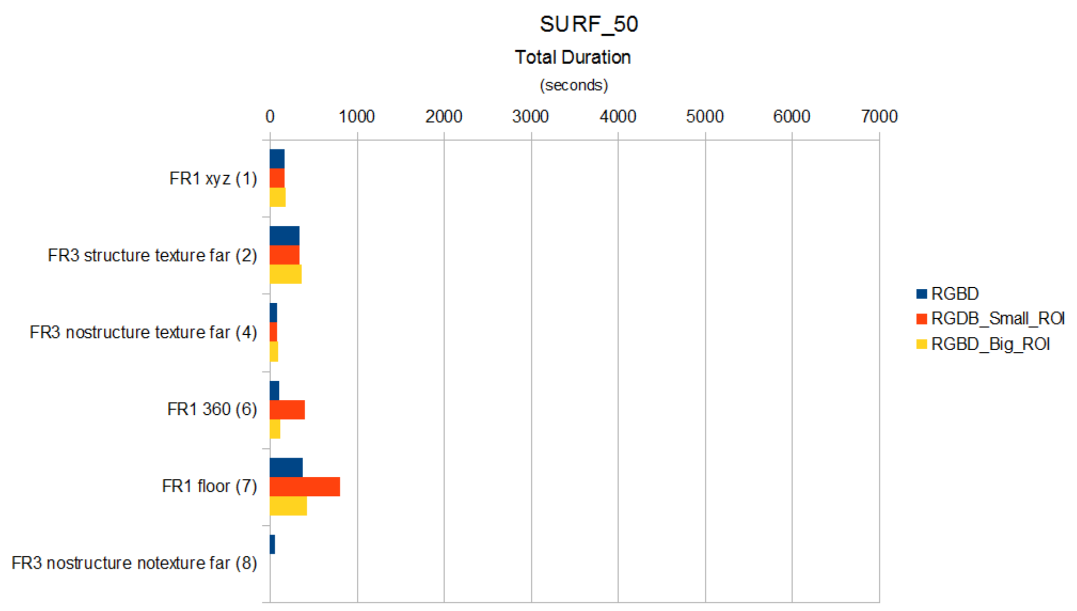


Figure 5.19: Duration of RGBD SLAM for all datasets using 50 SURF features.

The errors recorded for all versions of RGBD SLAM from the SURF_50 tests where larger than the errors from the SURF_600 tests. Additionally the SURF_50 test required less time to execute compared to the SURF_600 tests. This outcome was expected as when fewer features are captured it becomes more difficult to accurately determine the robots position and less time is required to process the features. The last dataset in the SURF _50 tests could only be processed by the traditional version of RGBD SLAM. The modified versions failed to extract any features in this dataset and hence no measurements were recorded.

It was found in simpler environments where the robot moves slower that the traditional version of RGB SLAM produces the lowest error in the absolute trajectory of the robot. However when the environments became more complex and the robot has faster movements the bigger ROI version produced the lowest error. In these cases blurry images are captured and it was found that the bigger ROI was able to detect more reliable features from the blurry images to accurately determine the trajectory of the robot compared the original version.

The bigger ROI outperformed the traditional version of SLAM in producing the most accurate pose of the robot for all the datasets. It was also observed that the bigger ROI performs just as well in requiring marginally the same amount of computational time to execute.

In summary it was found from these tests that the bigger ROI version with SURF outperforms or performs just as well as the original version of RGBD SLAM, with a reduced number of features that can be captured, in terms of accurately determining the position of the robot and computational time. This shows that the strategy of extracting features from a ROI with a reduced number of features can improve the accuracy in determining the position of the robot compared to the traditional method (distributed features) used in the traditional version of RGBD SLAM.

5.4 Summary

This thesis investigated the strategy of extracting features from a ROI for SLAM to accurately find the position of the robot. This strategy is compared to traditional method of extracting features from an entire image (distribution of features). This method of using ROIs is investigated as an alternate method as solution to improve the way RGBD SLAM processes the image data to identify and track landmarks.

Both methods are compared to determine which one can produce the most accurate position of the robot in the least amount of computational time. The ROI is implemented into the RGDB SLAM framework via a pre-processing pipeline. The development and integration of the pipeline into the framework in presented in Chapter 3. This chapter presented the tests that was conducted to gather results to compare both methods of capturing features.

In test 1 the functionality of the pre-processing pipeline was tested to be able to identify and track landmarks for RGBD SLAM to determine the position of the robot. In Test 2 the pipeline was embedded into the RGBD SLAM framework and tested if the modified versions would improve the accuracy of determining the position of the robot and the computational time needed. Test 3 conducted the same test as test 2, but with the versions of RGBD SLAM only using SURF features with a limit of 50 features that can be captured.

The results from test 1, where the pre-processing pipeline was tested in isolation found, that the pipeline can create a database of landmarks where landmarks can be identified and tracked for SLAM to calculate the position of the robot. It was also observed the size of the ROI used in the pipeline should be expanded to improve the precision of identifying and tracking a landmark. It was also observed that number of features extracted varied during the tests and to produce more comparable results a limit for the number features that can be captured should be set for all the feature extractors.

In test 2 the ROI was expanded and a limit of 600 features was used in the pipeline which was integrated into the RGBD framework. The previous size of the ROI was still used and tested. Therefore, there were three versions of RGBD SLAM tests which were the traditional version, a smaller ROI (RGBD_Small_ROI) and a bigger ROI (RGBD_Big_ROI) version. The results did show that the bigger ROI version with SURF features produced the most accurate position of the robot but requires more computational time to execute.

The results from test 3 retested all versions of RGBD SLAM with only using SURF features and reducing the number of features that can be captured to 50. It was found from these results that the bigger ROI version of RGBD SLAM with SURF features outperformed the traditional version in terms of accurately determining the position of the robot and marginally performing just as well in taking the similar time to execute. It was also observed in more complex environments with faster movements of the robot the bigger ROI seemed more robust in being able to be the most accurate in determining the position of the robot.

In summary the bigger ROI version of RGBD SLAM with SURF features using the strategy of extracting features just from the ROI seemed to improve the performance of RGBD SLAM compared to the original version of SLAM using the traditional method of capturing features from the entire image frame.

Chapter 6

Discussion and Conclusion

The last chapter performed tests on the pre-processing pipeline and embedded it into the RGBD SLAM framework. The pipeline was tested as a solution to improve the way RGBD SLAM processed the image data to identify and track landmarks. This method was compared to the traditional method used in RGBD SLAM. The pipeline extracts features from ROIs to identify and track landmarks for RGBD SLAM and the traditional method extracts a distribution of features from the entire image. Both methods were tested in the last chapter and the results were presented comparing the methods. This chapter will discuss and draw conclusions from the results.

Section 5.1 presents a discussion of the results captured from testing the pipeline in isolation. The results were analysed to determine whether the pipeline can create a database of landmarks and be able to recall a landmark later on which is stored in that database. In this section the precision of identifying a landmark with the three features extractors is also discussed and the overall outcomes are presented.

In Section 5.2 the results from the RGBD SLAM test and the reduced feature tests are discussed. The results from these tests were analysed to determine whether the strategy of extracting features from ROIs to identify and track landmarks for RGBD SLAM would improve the accuracy in determining the position of the robot and the computational time. The tests used two modified versions of RGBD SLAM that embedded the ROI algorithms via the pre-processing pipeline. The first version used smaller ROIs which was called 'RGBD_Small_ROI' and the second version used bigger ROIs call 'RGBD_bigger_ROI'. The results from these versions were compared to the results captured from the traditional version of RGBD SLAM. The outcomes from this comparison are also presented in terms of the accuracy in determining the position of the robot and the computational time.

The last section presents a summary of the overall outcomes of embedding the pre-processing pipeline in the RGBD SLAM framework compared to the traditional RGBD SLAM and is the pipeline a better method to identify and landmarks compared to the traditional method.

6.1 Pre-Processing Pipeline

To ensure the pre-processing pipeline can identify and track landmarks accurately for RGBD SLAM using ROIs two functionalities needed be verified. The first functionality is the ability for the pipeline to create a database of landmarks that have been identified by the ROI algorithm, and the second functionality is be able to precisely recall a landmark from the database. The results captured from Test 1 in Chapter 4 were analysed to verify the two functionalities.

The results recorded the number landmarks, the number of false positive and the false negatives of landmarks in the database for each feature extractor. The number of false positives is a reflection of the accuracy of each feature extractor in re-identifying a landmark. The number of false negatives in an indication of the number of duplicate landmarks that were saved, as the pipeline failed to re-identify them. The last result that was analysed was the duration of processing each dataset in the pipeline.

6.1.1 Database of Landmarks

The pipeline was able to successfully create a database of landmarks for SLAM for each feature extractor. One of the findings revealed that as the indoor office environments became more complex and when the robot's movement became faster, more landmarks were recorded. When there were more landmarks in the database it was observed that the number of false negatives increased as well. This is expected as in a complex environment or if the robot is moving faster the likelihood, of re-identifying a landmark accurately decreases hence more failed recalls and more duplicate landmarks being saved to the database.

The pipeline performed the worst on average with ORB as it was found to record the most amount of landmarks in the database. This was due to ORB extracting a higher number of unreliable features which led to higher false matches and false negatives.

The performance of the pipeline in identifying landmarks was also compared to the results from the study by Aulinas et al. The pipeline used the same method from that study to process the image data to detect a ROI to identify and track landmarks for SLAM. This thesis further proved the method shown by Aulinas et al. can detect ROIs but their study did not show the method performed poorly in complex environments and when movement of the robot is quick [6].

6.1.2 Matching of Landmarks

In re-identifying the landmarks the pipeline as mentioned above recorded the number of false matched ROIs and number of false negatives of landmarks within the database. These results were further analysed to determine the average number of true positives in percentage for each feature extractor in re-identifying a landmark accurately and the average percentage true negatives of landmarks in the database for each feature extractor. Both of these percentages are presented in Table 6.1. The average percentage of true positives is determined by using the number of correctly matched ROIs that each feature extractor recorded when re-identifying a landmark.

Table 6.1: Average percentage of true positives and true negatives of landmarks in the database.

	SURF	SIFT	ORB
True Positives	80.29 %	86.39 %	69.82 %
True Negatives	77.69 %	80.87 %	62.31 %

It was found when SIFT was used the pipeline produced on average the highest true positives and true negatives in re-identifying landmark as seen in Table 6.1. This outcome was due to SIFT extracting more reliable features that are uniquely re-identifiable compared to SURF and ORB which led to a lower number of false matched features and false negatives of landmarks. However, the second outcome that was observed was that in these tests the number of features that can be extracted varied with each feature extractor.

6.1.3 Computational Performance

The duration of the each feature extractor was recorded for all the datasets. These results was further analysed to determine the average frames per second that each feature extractor produced to process a dataset. This outcome is shown in 6.2.

Table 6.2: Average frames per second for each feature extractor.

	SURF	SIFT	ORB
Average Frames per second	0.67	1.79	0.2

From the above table SIFT is observed to produce the highest frames per second, indicating that the pipeline processes a dataset in the shortest time with SIFT. This is expected since SIFT was found to extract the most amount of reliable features, the time needed to re-identify a landmark is shorter compared to SURF and ORB.

The overall outcome of SIFT performing faster than SURF is unexpected. According to Bay et al. SURF was shown to outperform SIFT in accuracy and speed [32]. However as mentioned above the number of features extracted per feature extractor varied. Due to this SIFT extracted more features which led to a higher accuracy while SURF extracted less features and the number of features extracted varied from frame to frame which produced a lower accuracy and required more time to re-identify a landmark as the matching of features took longer.

Therefore, it was concluded that to produce more comparable results the number of features that could be extracted by all three feature extractors should be limited to a number (specifically to 600 for test 2 and 50 for test 3). In addition to improve the precision of identifying and recalling a landmark, the ROI algorithm should be altered to allow for a larger ROI to be detected in the images. These two findings were considered and implemented in the RGBD SLAM tests.

6.2 RGBD SLAM

The pipeline was fully integrated into the RGBD SLAM framework and tested where the results were presented in Chapter 4. The integration of the pipeline led to two versions of RGBD SLAM which were 'RGBD_Small_ROI' and 'RGBD_Big_ROI'. The first version used detected a smaller ROI which was used in the pre-dataset tests and the second version detected a bigger ROI. These two versions were compared to the traditional version of RGBD SLAM to determine if the modified versions would trade-off the accuracy in determining the location of the robot and a reduced computational time. If there is an improvement either one, this would indicate that the strategy of extracting features from a ROI to identify and track landmarks will be better than the traditional method of extracting features from an entire image. The results in the previous chapter were analysed to verify this outcome.

Test 2 analysed results of all three version of RGBD SLAM where the features were limited to 600. Test 3 analysed the results from the traditional version and the bigger ROI version of RGBD SLAM where only the SURF feature extractor was where the features were limited to 50. The results recorded the accuracy of the robot's position in terms of absolute trajectory and the pose of the robot. The accuracy in position of the robot would reflect on the precision of identifying and tracking a landmark. The last result recorded the duration of each test.

6.2.1 Accuracy in Determining the Position of the Robot

All three versions were able to successfully determine the robot's position successfully in all the datasets, except for the last dataset where the ROI versions of RGBD SLAM failed. Since the last dataset contained a featureless environment, the features extractors struggled to extract reliable features from the ROIs to be able to identify and track landmarks. Whereas the original version of RGBD SLAM was able to re-identify and track landmarks accurately using the traditional method of extracting features from the entire image. Hence, the original version was able to determine the position of the robot accurately. The ROI versions failed to process a featureless dataset as the ROI algorithm is designed to create ROI containing highly detailed objects that could be used as landmarks for SLAM. The featureless dataset does not contain any highly detailed object and therefore the ROI algorithm could not detect a ROI to extract features. With no or little features, the landmarks cannot be re-identified and tracked accurately for RGBD SLAM to determine the position of the robot accurately. This outcome showed that the ROI versions of RGBD SLAM were not as robust as the original version. This also indicates that the traditional method of extracting features from an entire image to re-identify and track landmarks is more robust to the strategy of only extracting features from a ROI.

The results of the absolute trajectory and pose of the robot were further analysed to determine which feature extractor produced the most accurate position of the robot for all three versions of RGBD SLAM. Table 6.3 shows the average error in absolute trajectory and Table 6.4 and Table 6.5 shows the average error in the pose of the robot in both translation and rotation respectively. These results are from the tests where the features extracted were limited to a number of 600.

Table 6.3: Average absolute trajectory error for each iteration of RGBD SLAM with using each feature extractor.

	SURF	SIFT	ORB
	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)
RGBD	0.143 m ± 0.123 m	0.097 m ± 0.054 m	0.133 m ± 0.099 m
RGBD Small ROI	0.131 m ± 0.104 m	**0.365 m ± 0.601 m**	**0.367 m ± 0.433 m**
RGBD Big ROI	**0.078 m ± 0.042 m**	0.259 m ± 0.417 m	0.191 m ± 0.299 m

Table 6.4: Average translational RMSE for each iteration of RGBD SLAM with using each feature extractor.

	SURF	SIFT	ORB
	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)
RGBD	**0.031 m ± 0.019 m**	0.041 m ± 0.027 m	0.035 m ± 0.018 m
RGBD Small ROI	0.038 m ± 0.025 m	**0.121 m ± 0.185 m**	0.066 m ± 0.037 m
RGBD Big ROI	**0.032 m ± 0.021 m**	0.050 m ± 0.046 m	0.043 m ± 0.028 m

Table 6.5: Average rotational RMSE for each iteration of RGBD SLAM with using each feature extractor.

	SURF	SIFT	ORB
	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)
RGBD	1.121 deg ± 0.646 deg	1.371 deg ± 0.809 deg	1.225 deg ± 0.599 deg
RGBD Small ROI	3.617 deg ± 4.983 deg	**5.312 deg ± 5.485 deg**	2.740 deg ± 0.930 deg
RGBD Big ROI	**1.065 deg ± 0.809 deg**	1.522 deg ± 0.801 deg	1.451 deg ± 0.635 deg

The RGBD SLAM with a bigger ROI using the SURF extractor was found to produce the lowest average absolute trajectory RMSE of the robot. This value is indicated by green in Table 6.3. For the translational pose of the robot both the original version and the bigger ROI of RGBD SLAM using SURF were found to produce on average the lowest RMSE which is indicated by green in Table 6.4. There was a difference of 1 mm between the two versions which is insignificant. The rotational pose of the robot with the lowest RMSE was the bigger ROI version of SLAM using SURF indicated by green in Table 6.5. With these three results the overall outcome indicates that the RGBD SLAM with the bigger ROI using SURF did to produce the most accurate position of the robot. SIFT and ORB produced the lowest average RMSE of the absolute trajectory, translational and rotational pose of the robot, when they were used in the original version of RGBD SLAM. The ROI versions did not improve the accuracy in determining the position with SIFT and ORB.

This outcome is due to SURF extracting the most reliable features from the bigger ROI to re-identify and track landmarks accurately for SLAM to determine the most accurate position of the robot. This outcome is expected as this was seen in the results from the studies conducted by Endres et al. [13] SURF produced similar results in the accuracy of the robot's position with the original version of RGBD SLAM. In that study Endres et al. found that SURF performed better than ORB and performed just as well as SIFTGPU in determining the translation and rotational pose of the robot.

This outcome is further verified in test 3 where SURF is used again with the bigger ROI version and the traditional version of RGBD SLAM where the number of features extracted is reduced. This result is also compared to previous results from using 600 features in terms of the robot's position. The results from the reduced features are presented in Table 6.6.

Table 6.6: Average absolute trajectory, translational and rotational RMSE for RGBD and RGBD Big ROI using a reduced number of 50 SURF features

	Avg Abs Traj	Avg Trans	Avg Rot
	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)	(Avg.± Std. Dev.)
RGBD	0.470 m ± 0.363 m	0.074 m ± 0.038 m	**5.914 deg ± 9.221 deg**
RGBD Big ROI	0.402 m ± 0.320 m	**0.045 m ± 0.030 m**	1.728 deg ± 1.631 deg

As expected the RMSE increases with reduced features as there are less reliable features to accurately re-identify and track the landmarks which causes the robot's position to be less accurate. However with reduced features the bigger ROI version of RGBD SLAM still produced lower RMSE compared to the original version. This outcome shows that with a reduced number of features the strategy of extracting features from a ROI still performs better than the traditional method of extracting features from an entire image, in accurately re-identifying and tracking landmarks for RGBD SLAM.

6.2.2 Duration

The average duration for each of the versions RGBD SLAM with each feature extractor to process the datasets were analysed and is shown in Table 6.7. The average times analysed were from the tests where the feature extractors were limited to 600 features. The original version of RGBD SLAM using SURF and SIFT was found to be quicker than the ROI versions as the integration of the pipeline into the RGBD framework add an additional step which did increase the execution time. This result was not expected as even though there is additional step, the time required to re-identify landmark was expected to reduce. However, ORB seemed to perform the quickest with the smaller ROI version of RGBD SLAM compared to the original version. This was due to ORB extracting few reliable features which allowed for a fast execution but incurred the most error in determining the position of the robot.

SURF was found to be the quickest with the traditional version of RGBD SLAM but with incurred more error compared to the ROI version. This is indicated by green in Table 6.7. This outcome indicated that SURF was more precise in re-identifying and tracking a landmark with the ROI version of RGBD SLAM which required more execution time as it extracted more reliable features. This outcome also reflects that with SURF the strategy of extracting features from a ROI may require more execution time but produces more accurate results in determining the position of the robot which is a trade-off in performance for RGBD SLAM.

Table 6.7: Average duration for each version of RGBD SLAM with using each feature extractor.

	SURF	SIFT	ORB
RGBD	**6.23 mins**	7.11 mins	15.20 mins
RGBD Small ROI	16.48 mins	7.62 mins	7.00 mins
RGDB Big ROI	8.09 mins	8.78 mins	**40.36 mins**

Due to SURF producing the most accurate results in determining the robot's position with bigger ROI version of RGBD SLAM compared with the original version, this is what was further tested where the number of features extracted were limited to 50. The duration from these tests was analysed to produce the average duration for each of RGBD SLAM and is presented in Table 6.8.

Table 6.8: Average duration for each version of RGBD SLAM with using a reduced number of 50 SURF features.

	SURF _50
RGBD	3.18 mins
RGDB Big ROI	4.00 mins

The average duration with the reduced features decreases which was expected as there are fewer features to process from frame to frame and do not vary. This reduces the time in the feature matching process. This outcome verifies that with SURF, additional execution time is required for the addition of the pipeline but it produces more accurate results of the position of the robot.

6.3 Summary

This chapter presents a discussion of the results recorded to determine whether the strategy of extracting features from a ROI to re-identify and track landmarks for RGBD SLAM, would improve the performance of RGBD SLAM compared to the traditional method of extracting features from the entire image. From the discussions the following conclusions were drawn:

- The pre-processing pipeline containing the ROI algorithm was able create a database of landmarks for RGBD SLAM. In addition all three feature extractor were able to recall the landmarks later on for RGBD SLAM.
- SIFT was found to have the highest precision in re-identifying and recalling a landmark in the pipeline. To further improve the precision, the ROI algorithm was altered such that a larger ROI can be detected from the images. This outcome was implemented when the pipeline was integrated into RGBD SLAM framework.
- Once the pipeline was embedded into the RGBD SLAM framework it was found the SURF with bigger ROI version of RGBD SLAM produced the most accurate

positions of the robot. When the features were further reduced SURF with the bigger ROI version of RGBD SLAM still produced more accurate results in the position of the robot compared the traditional version.

- The performance of RGBD SLAM did not improve when the ROI version was used with SIFT and ORB.
- The ROI versions of RGBD SLAM required more processing time compared with the traditional version, therefore the computational time was not improved.
- In a featureless environment the ROI versions of RGBD SLAM failed compared to the original version. Hence, the original version of RGBD SLAM is more robust in being to determine a robot's positions in a both feature rich or feature less environment.

The final conclusion found that using ROIs to track landmarks for RGBD SLAM can be used as an alternate method to process the image data. However compared to the traditional method it did not improve the computational time and increased the computational burden. But in terms of further improving the accuracy of the robot's position this method was found to be promising as especially in environments where the number of features extracted are limited. This method can only produce this performance with the SURF feature extractor. Additionally, this method trades-off computational time for an improved accuracy for RGBD SLAM. From this conclusion the next chapter presents
recommendations that can implemented to improve the performance of the pipeline with SLAM framework.

Chapter 7

Recommendations

The last chapter presented the outcomes of using ROIs to process image data as an alternate method to identify and track landmarks for RGBD SLAM was presented. For the purpose of this study, this method was applied via a pre-processing pipeline and integrated into the RGBD SLAM framework. In conclusion, this method of processing the image data did not improve the computational time for RGBD SLAM compared to the traditional method. However the accuracy of localizing the robot improved. This chapter presents possible recommendations and refinements that can be used to improve this method such that computational time can be reduced and improve the accuracy of localizing of the robot for RGBD SLAM. These recommendations can be applied to hardware presented in Section 7.2 and software presented in Section 7.3.

7.1 Hardware Recommendations

Microsoft has released a new version of the Kinect to use with their new gaming console. The new Kinect has a wider field of view and a higher resolution RGB camera. With a higher-resolution camera, the Kinect can capture more detail in an image which will help the feature extraction algorithms extract more reliable features to track landmarks more accurately. The depth sensor has also been improved to provide more accurate depth measurement which will improve the location of landmarks, and RGBD SLAM would be able to build a more accurate map of the environment.

Intel has recently released their own version of a depth camera called the Intel RealSense Depth Camera. This camera also has higher resolution, but can function in an outdoor environment. If this sensor is used with RGBD SLAM, this will allow RGBD SLAM to be used in both indoor and outdoor environments. RGBD SLAM has only been used in an indoor environment thus far.

Both of these hardware improvements will improve the quality of data that will be fed into the pre-processing pipeline and RGBD SLAM. With higher-quality data more reliable features can be extracted to precisely identify and track landmarks for SLAM.

7.2 Software Recommendations

The pre-processing pipeline is made of three stages as presented in this dissertation. Each stage of the pipeline consists of an image processing algorithm as seen in Figure 1.1. The pipeline is used to integrate the use of ROIs with SLAM.

The first stage of the pipeline is the ROI algorithm; the second is feature extraction and the last is feature matching. Each of the image processing algorithms used in each stage can be refined for an optimal performance. This section presents recommendations for each stage of the pipeline.

7.2.1 ROI Algorithm

It is vital to detect landmarks robustly in the environment for SLAM and this process needs to ensure that landmarks can be correctly re-identified which crucial for loop closure. Therefore the ROI algorithm needs to be able to detect an optimal ROI such that the feature extractors can identify and track landmarks accurately for SLAM. The ROI algorithm used in this thesis is designed to detect ROIs that contain highly detailed objects which make ideal landmarks for visual SLAM. However to ensure that ROI detected contains the most amount of detail a verification step should be added. A method of verification was presented in a study by Aulinas et al. [6]. The method manipulated the hue channel of the image to detect for a ROI that contains highly detailed objects. If the ROI detected by the verification stage matches or overlaps with the ROI detected initially this would confirm that the ROI detected is the optimal ROI that contains the most amount of detail. The only downside of adding an additional step is extra computational time.

By ensuring that optimal ROI is detected, the feature extractors would improve their precision and successful recalls of the landmarks. This would improve the overall accuracy of the robot's position.

7.2.2 Feature Extraction

In this study when the SURF feature extractor was used with the bigger ROI version of RGBD SLAM, this produced the most accurate results in determining the position of the robot. In the study by Endres et al. SIFTGPU was found to outperform SURF in terms of the accuracy [13]. This indicates the SIFTGPU extracted more reliable features and faster than SURF. Hence further studies should be conducted to determine whether the SIFTGPU with ROI version of RGBD SLAM can outperform SURF and improve the computational time.

7.2.3 Improved Feature Matching

In this thesis precision of re-identifying landmarks is based on the number of false matches made. For SURF a study conducted by Aulinas et al. used a Joint Compatibility Branch and Bound (JCBB) algorithm to match the features which, reduced the number of false features [6]. This would improve the robustness in identifying the landmarks which increases the accuracy in determining the robot's position.

7.3 Further Testing

It was found from this study when SURF features are extracted from a bigger ROI to track landmarks then RGBD SLAM produces more accurate results of the robot's position. However, this study was restricted to testing only indoor environments and only used the RGBD SLAM framework. To further extend this strategy, other types of environments and other versions of SLAM should be considered and tested.

7.3.1 Other Environments

This thesis was limited to using only indoor environments. The strategy presented in this thesis should be extended to outdoor environments to determine if SURF features extracted from the ROIs can still perform better than the traditional method to track landmarks for RGBD SLAM. The Intel RealSense depth camera can be used to capture outdoor environments.

This study also showed that if there was an environment with few features the strategy of SURF features with a bigger ROI allowed RGBD SLAM to produce more accurate results in determining the robot's position. Other environments such as underwater and underground caves or mines can be used to determine the robustness of this strategy as these environments contain few features.

7.3.2 Other Versions of SLAM

The pre-processing pipeline was only integrated into the RGBD SLAM framework in this thesis. RGBD SLAM is one version of visual SLAM. There are many other versions of visual SLAM such as monocular and stereo SLAM that can use the pipeline. The method by which the pipeline detects for an ROI was from the study by Aulinas et al. which implemented it to monocular SLAM in an underwater environment, but this method can be extended to stereo SLAM [6].

This study showed that the method of using ROIs to identify and track landmarks can work in RGBD SLAM. Therefore, the pipeline can be extended to other versions of RGBD SLAM that use ICP and BA to optimize loop closure.

7.4 Summary

In summary this chapter presented the possible refinements the can be made to the pre-processing pipeline. The refinements discussed are improvements that can be made to the pipeline to improve the performance it has on the computational time and accuracy in localizing the robot when integrated with SLAM. Hardware and software refinements were discussed.

References

[1] B. Siciliano and O. Khatib, Eds., *Springer Handbook of Robotics.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ISBN: 978-3-540-23957-4. DOI: `10.1007/978-3-540-30301-5`.

[2] H. Durrant-Whyte and T. Bailey, "Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms," *Robotics & Automation Magazine, IEEE*, vol. 13, pp. 99–110, 2006.

[3] T. Emter and A. Stein, "Simultaneous Localization and Mapping with the Kinect sensor," *Proceedings of ROBOTIK 2012*, pp. 239–244, 2012.

[4] J. Hartmann and D. Forouher, "Real-time visual SLAM using FastSLAM and the Microsoft Kinect Camera," *Proceedings of ROBOTIK 2012*, pp. 458–463, 2012.

[5] S. Frintrop, "Attentional landmark selection for visual SLAM," *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2582–2587, Oct. 2006. DOI: `10.1109/IROS.2006.281711`.

[6] J. Aulinas, M. Carreras, and X. Llado, "Feature extraction for underwater visual SLAM," *OCEANS 2011 IEEE, Spain*, pp. 1–7, Jun. 2011. DOI: `10.1109/Oceans-Spain.2011.6003474`.

[7] F. Endres, J. J. Hess, J. J. J. Sturm, D. Cremers, and W. Burgard, "3D Mapping with an RGB-D Camera," *Www2.Informatik.Uni-Freiburg.De*, vol. 30, no. 1, pp. 1–11, 2012, ISSN: 1552-3098. DOI: `10.1109/TRO.2013.2279412`.

[8] M. Hammond and S. M. Rock, "A SLAM-based approach for underwater mapping using AUVs with poor inertial information," *2014 IEEE/OES Autonomous Underwater Vehicles, AUV 2014*, 2015. DOI: `10.1109/AUV.2014.7054419`.

[9] T. Lemaire and S. Lacroix, "SLAM with panoramic vision," *Journal of Field Robotics*, vol. 24, pp. 91–111, 2007. DOI: `10.1002/rob`.

[10] F. Bertolli, P. Jensfelt, and H. Christensen, “SLAM using Visual Scan-Matching with Distinguishable 3D Points,” *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4042–4047, Oct. 2006. DOI: 10.1109/IROS.2006.281865.

[11] A. Davison and I. Reid, “MonoSLAM: Real-time single camera SLAM,” *Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–67, Jun. 2007, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1049.

[12] 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, Feb. 2012, ISSN: 0278-3649. DOI: 10.1177/0278364911434148.

[13] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, “An evaluation of the RGB-D SLAM system,” *2012 IEEE International Conference on Robotics and Automation*, vol. 1, no. c, pp. 1691–1696, May 2012. DOI: 10.1109/ICRA.2012.6225199.

[14] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems,” *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 573–580, Oct. 2012. DOI: 10.1109/IROS.2012.6385773.

[15] C. Xiangkui, J. Min, and Z. Liangyu, “A Feature Matching Method For Simultaneous Localization And Mapping,” *2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1091–1094, 2017.

[16] X. Shen, H. Min, and Y. Lin, “Fast RGBD-ICP with bionic vision depth perception model,” *2015 IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems, IEEE-CYBER 2015*, no. 61175094, pp. 1241–1246, 2015. DOI: 10.1109/CYBER.2015.7288121.

[17] K. Yousif, Y. Taguchi, and S. Ramalingam, “MonoRGBD-SLAM: Simultaneous Localization and Mapping Using Both Monocular and RGBD Cameras,” *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4495–4502, 2017.

[18] C. Kerl, J. Sturm, and D. Cremers, “Dense Visual SLAM for RGB-D Cameras,” *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.

[19] M. Andersen, T. Jensen, and P. Lisouski, *Kinect depth sensor evaluation for computer vision applications*. Aarhus University, 2012, ISBN: 2245-2087.

[20] J. Webb and J. Ashley, *Beginning Kinect Programming with the Microsoft Kinect SDK*. 2012, ISBN: 9781430241041.

[21] L. Cruz, D. Lucio, and L. Velho, "Kinect and RGBD Images: Challenges and Applications," *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*, pp. 36–49, Aug. 2012. DOI: 10.1109/SIBGRAPI-T.2012.13.

[22] *Documentation - Point Cloud Library (PCL).* [Online]. Available: http://pointclouds.org/documentation/tutorials/openni_grabber.php\#openni-grabber.

[23] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, "3D registration in dark environments using RGB-D cameras," *2013 International Conference on Digital Image Computing: Techniques and Applications, DICTA 2013*, 2013. DOI: 10.1109/DICTA.2013.6691470.

[24] M. Draelos, "The Kinect up close: Adaptations for short-range imaging," *2012 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 1280–1288, 2012.

[25] *Micrsoft SDK.* [Online]. Available: http://www.microsoft.com/en-us/kinectfor.windowsdev/start.aspx.

[26] *OpenNI SDK (open-source)-Download — OpenNI.* [Online]. Available: http://www.openni.org/openni-sdk/.

[27] *Kinect Matlab - File Exchange - MATLAB Central.* [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/30242-kinect-matlab.

[28] R Laganière, *OpenCV 2 computer vision application programming cookbook.* Packt Publishing Ltd, 2011, ISBN: 9781849513241.

[29] D. Lowe, "Object recognition from local scale-invariant features," *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1150–1157 vol.2, 1999. DOI: 10.1109/ICCV.1999.790410.

[30] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," *2011 International Conference on Computer Vision*, pp. 2564–2571, Nov. 2011. DOI: 10.1109/ICCV.2011.6126544.

[31] J. Clemons, *SIFT-Scale Invariant Feature Transform.* [Online]. Available: http://www.csce.uark.edu/~mqhuang/courses/5013/f2011/proj/sift.presentation.pdf.

[32] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," *Computer Vision–ECCV 2006*, pp. 404–417, 2006.

[33] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision–ECCV 2006*, pp. 1–14, 2006.

[34] P. L. Rosin, "Measuring Corner Properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, Feb. 1999, ISSN: 10773142. DOI: 10.1006/cviu.1998.0719.

[35] *Fast Approximate Nearest Neighbor Search — OpenCV 2.4.8.0 documentation.* [Online]. Available: `http://docs.opencv.org/modules/flann/doc/flann\_fast\_approximate\_nearest\_neighbor\_search.html\#muja2009`.

[36] M. Muja and D. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.," *VISAPP (1)*, 2009.

Appendix A

Description of Input and Output Files for Dataset Tests

The datasets used in the testing of the pipeline have specific input files to read in the colour and depth images. Once the pipeline process the data, output files are created. This appendix provides a description of the input and output files for the dataset tests.

A.1 Description of Input Files

Each dataset provides the following:

- Folder named ”rgb/” containing all colour images.
- Second folder named ”depth/” containing all the depth images.
- A text file containing the file names of the colour images called ”rgb.txt”.
- The names of the depth images are also stored in a text file called ”depth.txt”.
- Accelerometer readings from the Kinect are recorded to a text called ”imu.txt”.
- Actual 3D positioning of the Kinect is recorded to a text called ”groundtruth.txt”.

For this dissertation from the datasets the two folders containing the images and the text files containing the names of the image, are the only items used. The colour and depth images have already been calibrated such that the pixels from one correspond correctly with the other. The depth measurements from the depth images are scaled by a factor of 5000, hence Equation (A.1) was used to calculate the actual depth measurements.

$$z = d_p \times \frac{1}{5000}, \tag{A.1}$$

where z is depth in meters and d_p is the measured depth from the image in meters.

A.2 Description of Output Files

Results of the datasets were obtained from text files that were output from the algorithm. As the algorithm processed each image a text file was produced with the name of the text file corresponding to the image number. When there was a new ROI detected, the algorithm saved it as a .jpg with name at which image number it was detected at. At the end of the test the number text files outputted would be the as the number of images in the dataset. If the algorithm cannot find a ROI, it moves on to the next image. Each text file contains the following information:

- If there was a new ROI and if it was matched to any of the previous ROI, the number of that ROI was printed; otherwise "No New ROI" is printed. If there is a "NaN" then there was no ROI detected.
- Number of matches.
- Number of false matches.
- Duration to process the images in seconds.
- Depth measurements of the features in meters.

The depth measurements were not used as these could not be validated.

Thereafter all the text files are processed in Matlab and a final output file was produced containing the number of ROIs, the average number of matches, the average number of false matches, the average duration to process each image in seconds, total duration to execute the dataset in minutes and the average depth of the ROI at that step.

Appendix B

Raw Results Captured

This appendix presents the raw results captured from the three tests that was conducted for this thesis.

B.1 Raw Results from Test 1

This section presents the raw measurements and detailed graphs that caputured from test1.

B.1.1 Raw Measurement for each Dataset

For each dataset that was processed by the pipeline, all three measurements are tabulated for each feature detector that was used. The results from each dataset are presented below with analysis and comparison of all seven datasets.

Results from Dataset 1

ORB produced the higest number of false matches and number of failed recalls of landmarks in the database compared to SIFT and SURF. SIFT and SURF produced the same number of false matches but SURF results in a higher number of failed recalls in database. Results are shown in Table B.1.

Table B.1: Landmark results of Dataset 1.

	SURF	SIFT	ORB
No. Landmarks in Database	33	32	34
No. False Matches	2	2	12
No. of Failed Recalls	10	4	12

Results from Dataset 2

As dataset 2, contained mostly highly detailed boxes and pages stuck to the floor in a small office environment, there are fewer objects hence fewer landmarks were identified. When SIFT was used there was zero false matching and failed recalls that occurred meaning compared to the SURF and ORB, SIFT always correctly identified the ROI and tracked the landmarks correctly with no duplicates being stored in the database as shown in Table B.2.

Table B.2: Landmarks results of Dataset 2.

	SURF	SIFT	ORB
No. Landmarks in Database	11	8	13
No. False Matches	0	0	3
No. of Failed Recalls	3	0	4

Results from Dataset 3

Dataset 3 used the same environment as dataset 2 but the Kinect was placed and moved closer to the objects. Although there were the same number of objects as in dataset 2, more landmarks were detected due to the Kinect being positioned closer. It was found in some instances that the ROI algorithm in the pipeline identified parts of an object as separate landmarks. This occurred when an object with a high degree of bold writing and shapes was in the scene where the bold writing and shapes were detected as separate landmarks. This is due to the Kinect being closer to objects in this environment. The results were similar to dataset 2 as SIFT outperformed SURF and ORB as shown in Table B.3.

Table B.3: Landmark results of Dataset 3.

	SURF	SIFT	ORB
No. Landmarks in Database	52	36	102
No. False Matches	18	7	21
No. of Failed Recalls	10	9	44

Results from Dataset 4

Dataset 4 just contained the highly-detailed pages stuck on the floor, hence the ROIs detected fewer landmarks. In this dataset SURF outperformed SIFT and ORB as shown on Table B.4 where SURF results in the lowest error in tracking of landmarks with lowest number of recalls. More landmarks were produced with ORB but it also produced a higher number of failed recalls in the database.

Table B.4: Landmark results of Dataset 4.

	SURF	SIFT	ORB
No. Landmarks in Database	36	36	40
No. False Matches	4	6	4
No. of Failed Recalls	6	8	30

Results from Dataset 5

This dataset is similar to dataset 3, with only-highly detailed pages stuck to the floor. SURF resulted in fewer number of landmarks stored in the database and false matches. SIFT produced the lowest number of failed recalls of landmarks in the database. ORB performs the worst as it produces the most landmarks, highest false matches and failed recalls. Results from this dataset are shown in Table B.5.

Table B.5: Landmark results of Dataset 5.

	SURF	SIFT	ORB
No. Landmarks in Database	81	99	103
No. False Matches	18	29	53
No. of Failed Recalls	14	8	15

Results from Dataset 6

Due to the Kinect moving faster compared to the previous datasets, some of the images captured were blurred. Therefore identifying landmarks became difficult and a higher number of landmarks were stored in the database. The blurred images also produced more errors in tracking of landmarks. SURF recorded the lowest number of landmarks that were stored in the database and lowest number of failed recalls of landmarks in the database. SIFT recorded the lowest number of false matches while ORB performed the worst out the of three resulting in the most landmarks stored in the database and most of errors in tracking the landmarks. The results of this database are shown in Table B.6.

Table B.6: Landmark results of Dataset 6.

	SURF	SIFT	ORB
No. Landmarks in Database	124	129	186
No. False Matches	47	6	75
No. of Failed Recalls	24	33	54

Results from Dataset 7

Datasets 7 is was found to have blurry images similar to the previous dataset, therefore the pre-processing pipeline reacted in the same manner as the previous dataset yielding similar results. However SURF performed the best overall compared to SIFT and ORB with minimum number of landmarks in the database and lowest number of errors in tracking the landmarks. ORB performed the worst as expected from the previous dataset. The results recorded from this dataset is shown in Table B.7.

Table B.7: Landmark results of Dataset 7.

	SURF	SIFT	ORB
No. Landmarks in Database	108	185	236
No. False Matches	29	35	72
No. of Failed Recalls	29	75	85

B.1.2 Graphs Produced from the Raw Results

Figures B.1 and B.2 plot bar graphs showing average matching of features and false matching of features for each feature extraction algorithm for all the datasets. The graphs can also give an indication of which feature extraction algorithm performed the best in the pre-processing pipeline.

In the graph plotted in Figure B.1 the average number of matched features are shown.

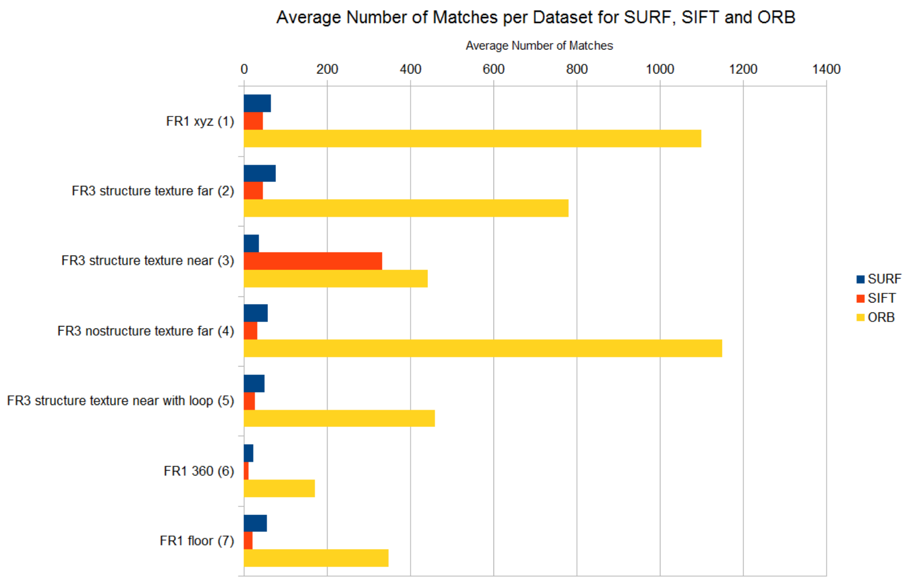


Figure B.1: Graph of average matches of features for SURF, SIFT and ORB for all the datasets.

In the graph plotted in Figure B.2 the average number of false matched features are shown.

B.2 Raw Results from Test 2

This section presents the raw measurements and detailed graphs captured from test 2.

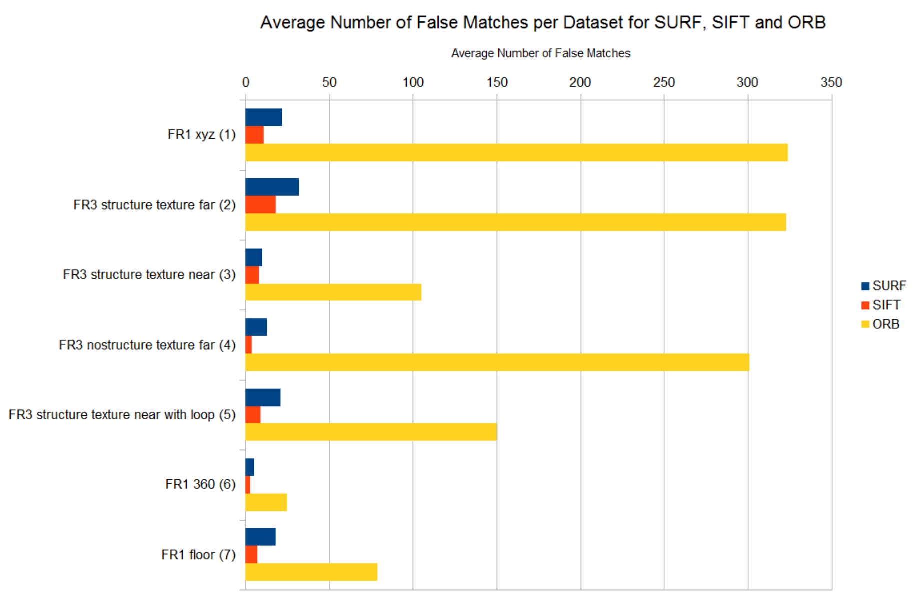


Figure B.2: Graph of average false matches of features for SURF, SIFT and ORB for all the datasets.

B.2.1 Graphs Produced from the Raw Results

Test 2 integrated the pre-processing pipeline into the RGBD SLAM framework. The tests produced absolute trajectory error and realtive pose error.

Absolute Trajectory Error

In Figure B.3 a graph of the absolute error of the robot for all the datasets using the SURF feature extractor is shown.

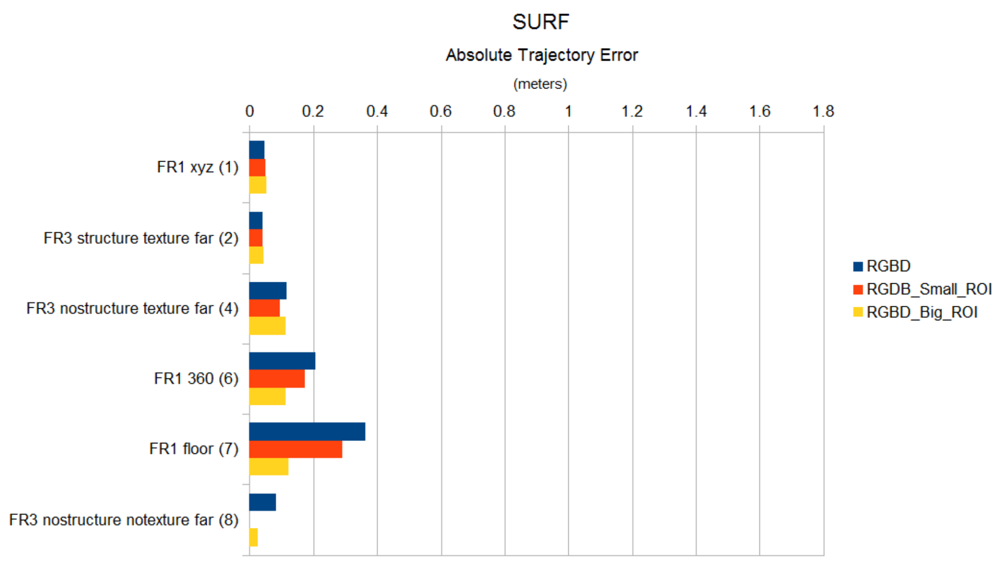


Figure B.3: Absolute trajectory error of the robot for all datasets using the SURF feature extractor.

Figure B.4 shows a graph of the absolute error of the robot for all the datasets using the SIFT extractor.

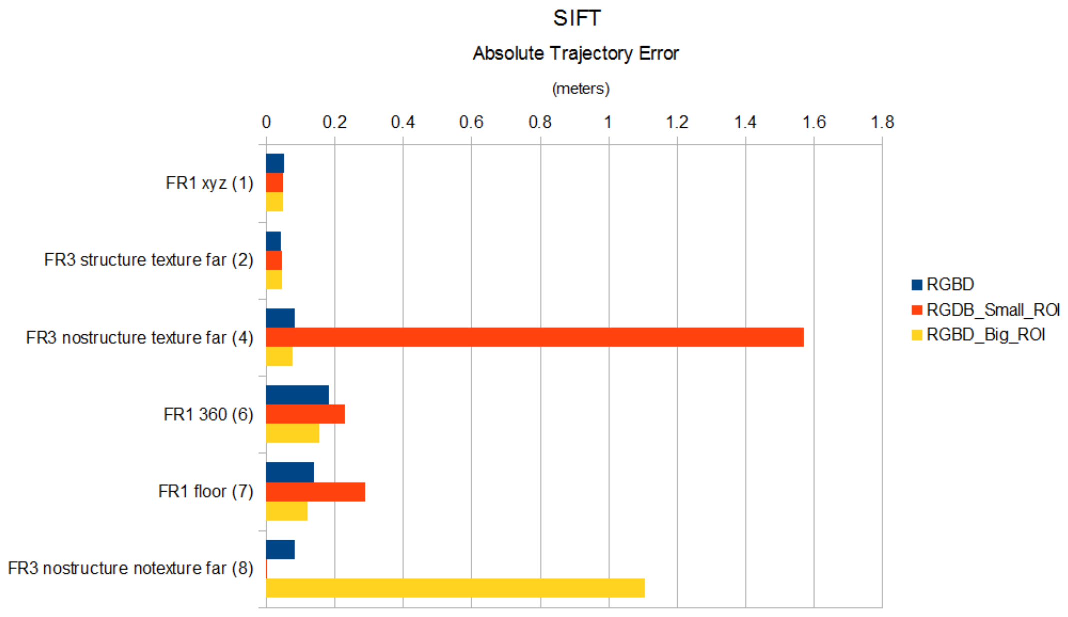


Figure B.4: Absolute trajectory error of the robot for all datasets using the SIFT feature extractor.

A graph of the absolute error of the robot for all the datasets using the ORB extractor is shown in Figure B.5.

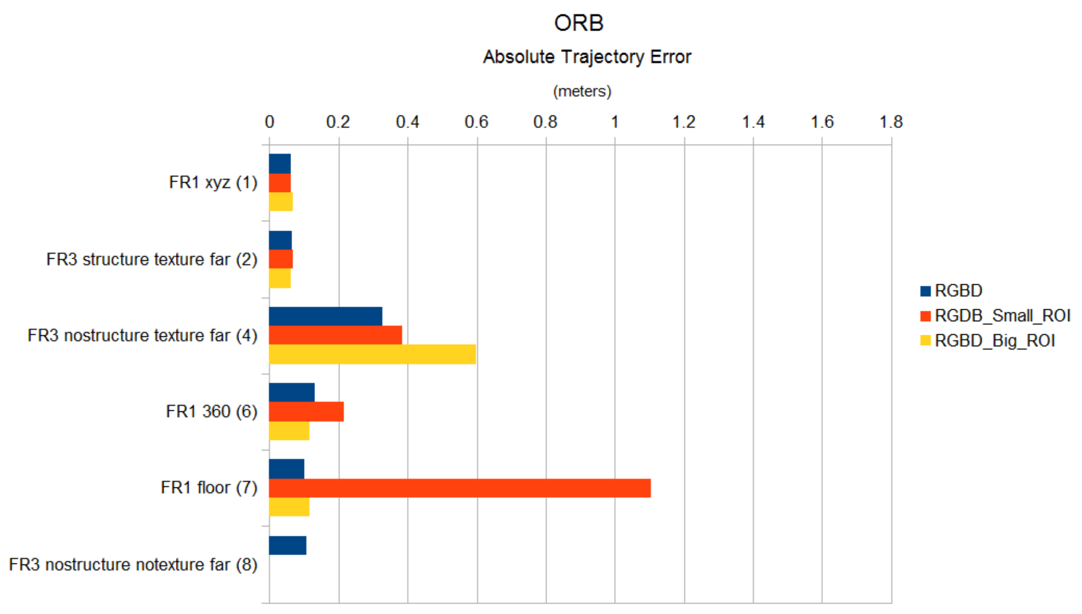
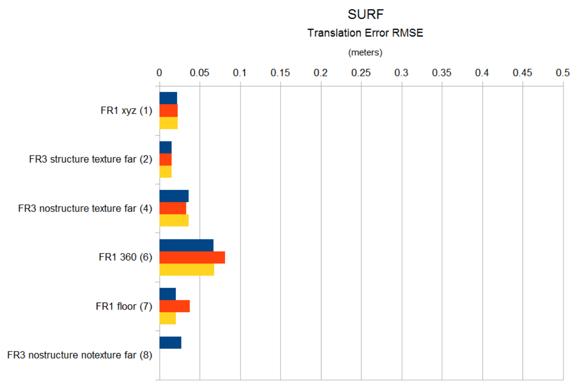


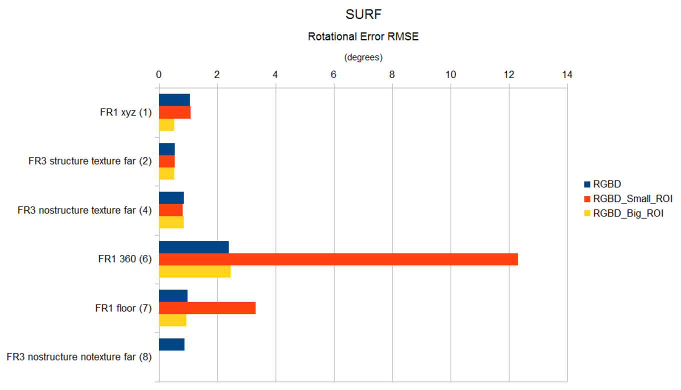
Figure B.5: Absolute trajectory error of the robot for all datasets using the ORB feature extractor.

Relative Pose Error

Graphs of the translation and rotational RMSE for all the datasets using the SURF feature extractor is shown in Figure B.6.



(a) Translations error.

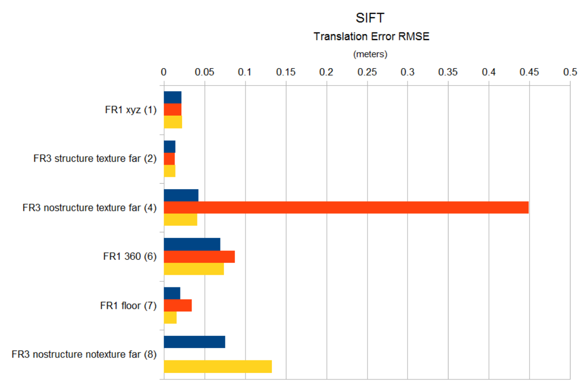


(b) Rotational error.

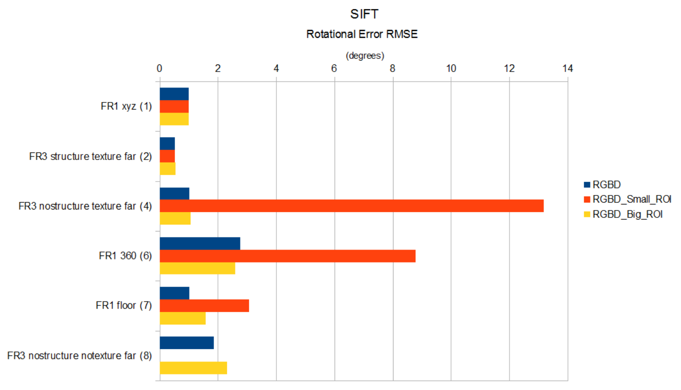
Figure B.6: Graphs of Translational and Rotational RMSE for all the datasets using the SURF feature extractor.

In Figure B.7 the graphs of translational and rotational RMSE for all the datasets using the SIFT feature extractor is shown.

Figure B.5 shows the graphs of the translational and rotational RMSE for all the datasets using ORB feature extractor.

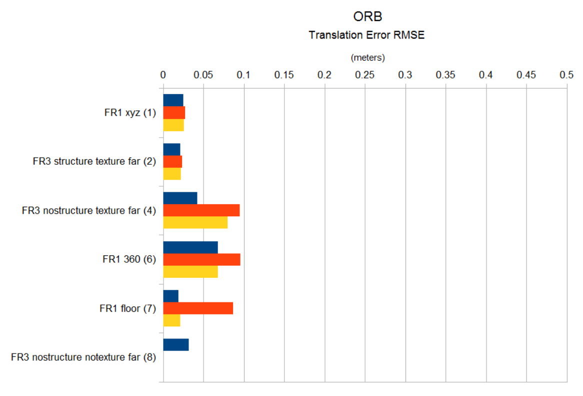


(a) Translations error.

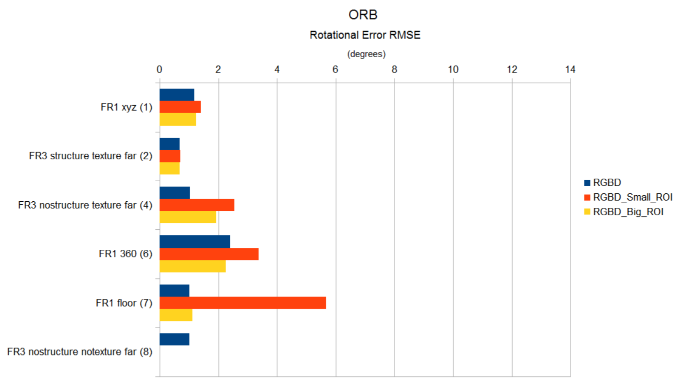


(b) Rotational error.

Figure B.7: Graphs of Translational and Rotational RMSE for all the datasets using the SIFT feature extractor.



(a) Translations error.



(b) Rotational error.

Figure B.8: Graphs of Translational and Rotational RMSE for all the datasets using the ORB feature extractor.