Fusion-SLAM by Combining RGB-D SLAM and Rat SLAM

A thesis presented in partial fulfilment of the requirements for the degree of

Master of Engineering

In

Mechatronics

At

Massey University,

Albany, New Zealand.

Robert Tsunemichi Tubman

2016

Abstract

Robotic Simultaneous Localization and Mapping (SLAM) is the problem of solving how to create a map of the environment while localizing the robot in the map being created. This presents a causality dilemma where the map needs to be created in order to localize the robot, but the robot also needs to be localized in order to create the map. In past research there have been many solutions to this problem ranging from Extended Kalman Filter (EKF) to Graph SLAM systems. There has also been extensive research in bioinspired methods, like ratSLAM implemented in aerial and land-based robots. The different research setups use sensors such as Time of Flight (ToF) e.g. laser scanners and passive devices e.g. cameras. Over the past few years a new type of combined apparatus has been developed by Microsoft called the Kinect. It combines active and passive sensing elements and aligns the data in a way which allows for efficient implementation in robotic systems. This has led to the Kinect being implemented in new research and many studies, mostly around RGB-D SLAM. However these methods generally require a continuous stream of images and become inaccurate when exposed to ambiguous environments.

This thesis presents the design and implementation of a fusion algorithm to solve the robotic SLAM problem. The study starts by analysing existing methods to determine what research has been done. It then proceeds to introduce the components used in this study and the Fusion Algorithm. The algorithm incorporates the colour and depth data extraction and manipulation methods used in the RGB-D SLAM system while also implementing a mapping step similar to the grid cell and firing field functions found in the ratSLAM. This method improves upon the RGB-D SLAM's weakness of requiring a continuous stream and ambiguous images. An experiment is then conducted on the developed system to determine the extent to which it has solved the SLAM problem. Moreover, the success rate for finding a node in a cell and matching its pose is also investigated.

In conclusion, this research presents a novel algorithm for successfully solving the robotic SLAM problem. The proposed algorithm also helps improve the system's efficiency in navigation, odometry error correction, and scan matching vulnerabilities in feature sparse views.

Acknowledgments

I would like to express my deepest gratitude and sincerest appreciation to Dr Khalid Arif for his supervision, enthusiastic guidance and continuous encouragement throughout the course of this study.

I would also like to express my appreciation to my co-supervisor A/Professor Johan Potgieter, and the University staff for continuous support and encouragement throughout the course of this study.

I am also thankful to my family and friends for continuous support, encouragement and emotional support throughout all of my achievements in life.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BRIEF – Binary Robust Independent Elementary Features

CMOS – Complimentary Metal-Oxide Semi-conductor

DoF – Degrees of Freedom

EKF - Extended Kalman Filter

FAST – Features from Accelerated Segment Test

GTSAM – Georgia Tech Smoothing and Mapping

ICP – Iterative Closest Point

IR – Infra Red

MAV - Micro Aerial Vehicle

ORB – Oriented FAST and Rotated BRIEF

RANSAC – Random Sample Consensus

RFID – Radio Frequency Identification Device

RGB-D – Red Green Blue Depth, Colour and Depth data

ROS - Robot Operating System

SIFT – Scale Invariant Feature Transform

SLAM - Simultaneous Localisation and Mapping

SPmodel - Symmetries and Perturbations model

SURF – Speeded Up Robust Features

ToF – Time of Flight

TSDF - Truncated Signed Distance Function

Chapter 1 - Introduction

Simultaneous localisation and mapping (SLAM) is the process of creating a map of the environment while simultaneously localizing the robot in the said map. A robot, when introduced to an unknown environment must create an internal representation of its surroundings to be able to know where it and objects are. There have been a variety of research that uses different sensors such as lasers and cameras to obtain 2D and 3D information about the environment, and the robot must interpret and process this to begin comprehending the layout of its surroundings. For a robot to be able to conduct tasks in real-time without user input, it must be able to internalize the data and update the map and its position simultaneously which presents a causality dilemma where the robot needs to create the map of the environment in order to localize itself, but the robot also needs to localize itself using its surroundings in order to create the map. The way that the environment is represented within the machine varies across studies, where the two main mapping methods are grid-based and topological. Once this internal representation has been determined the robot uses this information to calculate its current position and moves around in the environment, updating and adding to its database which is process of SLAM. By solving this issue, the robot becomes able to conduct tasks and interact with objects within the environment in a meaningful manner, which can be expanded to aid humans in a variety of applications.

1.1 - Problem Statement

Robotic SLAM has been researched using a variety of locomotive and sensory components. The solution requires the robot to manoeuvre in its environment in an efficient and effective manner, and existing systems have created solutions with varying results. Although many of these studies have produced successful results, there has still been no outstanding solution which solves this problem. Therefore a new system is proposed which combines different aspects of existing systems, namely the RGB-D and ratSLAM algorithms.

1.2 - Proposed Solution and Scope

The proposed system is a Fusion SLAM approach consisting of RGB-D and ratSLAM components. The RGB-D system analysed processes colour image and depth data in an effective manner which extracts highly descriptive features, and ratSLAM processes and combines colour images as experience nodes using a grid cell and firing field method which separates each cell using pose information. Therefore the solution will involve integrating the strong points of the ratSLAM and the RGB-D algorithms to solve the robotic SLAM problem. The solution involves developing the system in stages, by first combining the effective elements of the introduced algorithms. The second stage will be to add a navigation algorithm and the final stage will be to test it to determine whether it could conduct SLAM in a real world environment

1.3 - Thesis Layout

This thesis is laid out as follows: Chapter 2 presents the literature review which discusses the methods used by different systems that have been created in the past. Chapter 3 covers the robotic system setup which includes the hardware and software components used in this study. Chapter 4 introduces the solution algorithm which is the combination of parts from RGB-D and ratSLAM. Chapter 5 presents the experimental setup and results that were obtained. Chapter 6 will conclude this study discussing the results and future research directions.

Chapter 2 - Literature Review

This chapter analyses the studies conducted by previous researchers covering the topics that include EKF-SLAM systems, ratSLAM systems, and RGB-D SLAM systems and concludes with the summary of findings. Each topic will introduce a few different systems related to the topic highlighting the main contribution of each research.

2.1 - EKF-SLAM Systems

A variety of research has been conducted using the EKF SLAM method, ranging from estimating the odometry error using augmented EKF to creating a Symmetries and Perturbations map (SPmap) with 3D data [1-9].

The study conducted by [3] implements a fast loop closure technique using a clustering method involving the use of a k-means algorithm and an iterative end point fitting algorithm for indoor mapping, while implementing an EKF algorithm to solve the loop closure problem within SLAM. This method involves scan matching between different laser scans at different points in time similar to other matching algorithms, but the addition of extra landmark data, by clustering laser scans and using the k-means algorithm, improves the loop closure detection unlike those in [1, 4]. The k-means algorithm divides sensor data into k divisions where each point in the data belongs to a cluster. This is done by randomly choosing the k centres and then clustering the data points into data objects using the Euclidean distance to calculate the distance to each mean cluster k. These data objects are then regrouped by iterating through the clusters to optimize the objective function. Iterative end point filtering is then applied to the regrouped data object to obtain line segments from the scans to define the wall's corners and centres. The centre points and corner points are defined as the new landmarks. These landmark corners and centres are scan matched to other landmarks from consecutive images to close loops in a quick manner. The robot was tested in an indoor environment which included medium to large rooms, corridors, and cluttered complex rooms. Their algorithm is shown to be able to quickly correct loop closure errors which can allow the robot to carry on mapping the environment.

In [4] a different approach is taken to solve the SLAM problem using the EKF algorithm. Multiple robots, each with a 2D laser scanner and a webcam, were used to map the environment by extracting line and corner features. They used the laser scanner to extract data on the horizontal plane and used the camera to extract the vertical data.

To estimate the pose of each robot they used a unified EKF algorithm with the camera. They first extract a vertical feature from an image and use it as reference 1, the robot then moves a small distance to one of the sides and then they extract the feature from the image again. They compare this image with the stored reference and if it meets their threshold they determine that the extracted vertical feature and the previously extracted vertical feature are the same point and so the two features are correlated in the map. The use of the laser scanner is to extract the line and corner information and this feature is matched in a similar way to the vertical image features using the nearest neighbour method. Each robot builds its own map, which is shared to the other robots. To share this map, the robots use Wi-Fi to send feature data packets to nearby robots. When this data is sent the robot does not always know where the other robots are in the environment so to calculate the poses of other robots the robot uses the feature data packets to match corresponding features. When a robot's internal map starts to overlap with the maps from the other robots the map alignment transformation is calculated using the Iterative Closest Point (ICP) algorithm and is then realigned for the global map. The realigned map is then added to the other robot's map data creating the global map which is then used to estimate the poses of other robots which was initially uncertain. These robots are tested in an indoor environment with tables, walls, and boxes and are shown to be able to create an accurate map of the environment although there are some parallax errors.

Another approach using the EKF system, studied in [1], uses a rotating SICK laser scanner to create a 3D map of the environment. The sensor is a 2D scanner built onto a stepper motor driven belt transmission system to obtain the 3rd dimension. To represent uncertain geometric data, the use of a SPmodel is applied to a reference point and its vector. To extract features from the 3D data, a probabilistic process is used. First the transform from the robot's base to the scanner's centre is calculated, and then the 3D data is segmented by decomposing the data into plane cells using the Random Sample Consensus (RANSAC) algorithm. The infinite plane is then extracted from the segmented data using the linear least-squares regression, which is minimized to place the plane on the centre of gravity of the segment as shown in Figure 2.1. The SPmap is used with the EKF SLAM model to map the environment where the SPmap contains the vectors from the world, local, and robot frame to feature points in the environment. The robotic system is tested using a simulation and in a real environment. The simulation

proved that their system works and is capable of solving the SLAM problem. The method is also tested in a real location because it is expected that certain conditions will exhibit uncertain behaviours. The experimental result of the proposed robot successfully creates a map of the environment and corrects for the pose error accumulated to prevent it from distorting the map.

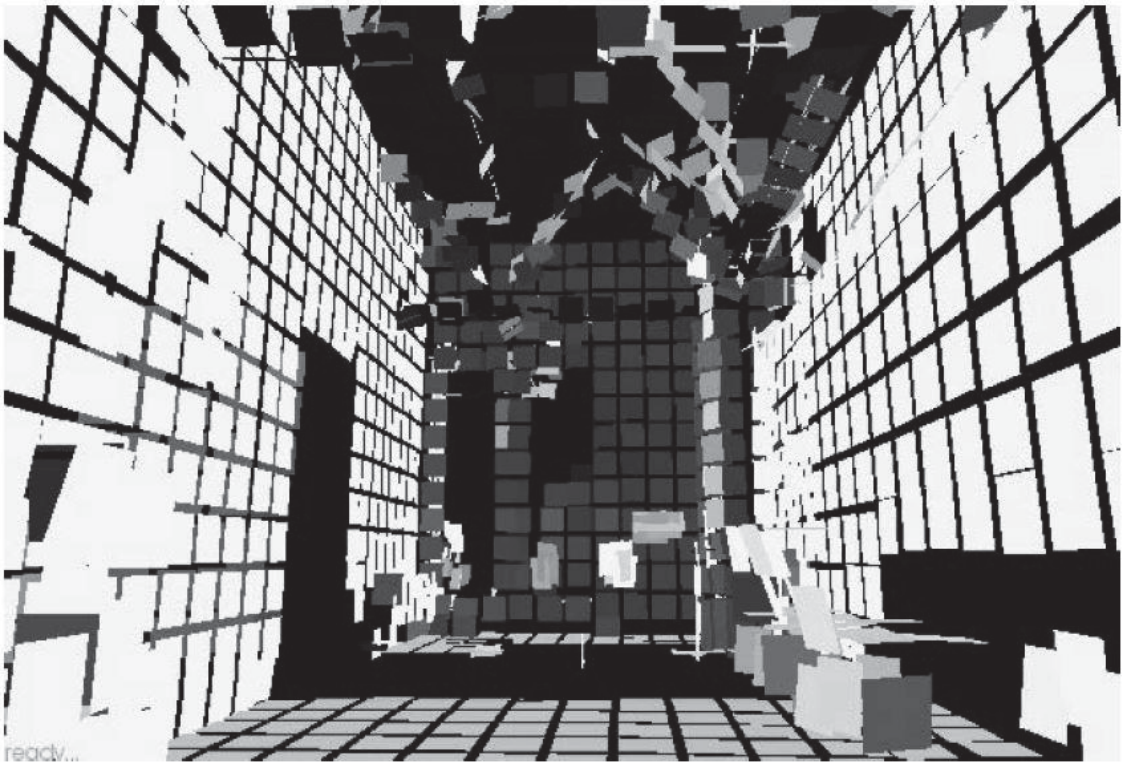

Figure 2.1: Plane data on the centre of gravity of the point segments [1].

2.2 - RatSLAM Systems

Many different studies have been conducted using the ratSLAM algorithm, ranging from systems being implemented on Micro Air Vehicles (MAV) to goal memory systems [10-19]. RatSLAM is an algorithm based on a rodent's neural processes and involves three main stages. It includes a continuous attractor network called a pose cell, a graphical map called an experience map and local view cells. The pose cells filter the robot's odometry data as self-motion cues and image information as image nodes. The image nodes known as local view cells process the images for distinctive features which are combined with the pose cells to form the experience map. This system, being a flying robot, uses visual odometry based on two patch locations in an image where the top patch is used to track the robot's yaw while the lower patch is used to track the

robot's translational velocities. The patch-based visual odometry system is conducted by calculating the average intensity difference between the previous and stored image patches. The local-view cells converted the colour images to grey-scale and applied a Gaussian blur to form visual templates which were used in the ratSLAM algorithm. To calculate differences in the visual templates the normalized sum of pixel intensity differences was used over a moving sub-frame. The system compares all current view templates with previously stored templates to determine whether to add the current view due to being a new location or to "recall" a stored template due to matches in the image.

The system constructed in [12] uses ratSLAM on a MAV and implements an expectation algorithm to conduct SLAM. The proposed method splits the camera view into two patches one above the other and the top patch calculates the vehicles yaw while the other tracks the translational speed. To compare the differences between views, the average intensity differences between pixel patches are calculated. Due to the system being an aerial vehicle, the images were passed through a de-shaking filter before reducing the resolution, applying a Gaussian blur, and gray-scaling the image. Unlike other ratSLAM systems that use a constant global threshold for recall, the proposed method implements an expectation algorithm to recall all images that immediately follow the current scene. The issue that comes with this is a false positive feedback cycle but this is negated by the ratSLAM's pose filter. To test the proposed method an expectation recall graph was generated by running the system with and without the algorithm and the results obtained show that the expectation recall tests had lower average error and the matching results were more accurate. Therefore the MAV ratSLAM system successfully conducts SLAM.

The system used by [11] uses a combination of ratSLAM and four wireless sensors which are an active and passive Radio Frequency Identification (RFID), a geomagnetic flux sensor, and a Wi-Fi sensor to activate the algorithms firing fields to determine the mean mutual information rate of different combinations. The system obtains mean mutual information using the wireless sensors by taking the mean number of Monte Carlo samples. The system also defines a number of pose headings and possible measurements as an ensemble. The measurement for the active RFID is the hardware address and the received signal strength in dBm and the geomagnetic flux sensor measures 3 axes. The passive RFID measures the tag ids where each tag is spread at 1m intervals throughout the environment and the Wi-Fi scanner records the hardware

addresses of each access point that are visible and has no overlapping channels. The Wi-Fi also measures the signal strengths of each access point like the active RFID sensor. The mobile robot also implemented a laser scanner and a camera and used the Robot Operating System (ROS) framework. The robot was driven at a speed of 0.25m/s to increase the measurement density and the recorded data was fed into the ratSLAM algorithm in real-time. The results from this work show that as the mean mutual information increases the mean error of the experience map decreases. It is further shown that the flux sensor, which was sampled the most frequent, has significantly lower mean error, which also created the most accurate maps.

The system introduced by [10] used ratSLAM that implemented a goal memory system which was used to successfully navigate to a particular point in the environment unlike other systems which doesn't create a navigation loop to utilise the map created. The experience map is used to create and maintain a collection of "experiences" in the form of pose cells and local view cells. As the robot moves in the environment, repeating odometry data is averaged to simplify the information. The created map is different to the systems studied by [11-13] because it is corrected by the number of experiences connected between one node and another which is determined by the learning rate. The learning rate for this system was chosen through experimentation to converge to stability. None of the systems studied uses a robot to travel from experience to experience autonomously, with the researcher either controlling the sensor manually by hand or through the use of a remote controlled robot. Furthermore the system studied by [10] reduces the confidence in a particular experience if there is any travel to an unexpected experience. Once the confidence drops below a given threshold the node is removed to keep it up to date. The connections between the experiences in the map are used to create a temporal map for goal recall (e.g. Figure 2.2). The temporal map creation starts at the peak experience and all other experiences are given very large values. Each experience is also assigned a time-stamp value depending on the links between the experiences and this process is iterated for each experience in the map. To find the shortest route, a gradient climbing procedure is used where the route is stored as a sequence of nodes. Each node represents an experience and is time stamped to represent the distance in time to other experiences. The robot will choose the node which is spatially closest to the current node but as the robot traverses the environment new image experiences can be learnt. These new experiences cannot be immediately

used in the temporal map because the time-stamp cannot be propagated from an experience that is not linked to the other experiences yet. In this situation the robot will rely on an already determined path. The robotic system proposed was tested in an indoor environment and was first set to create a map, and then navigate towards the goals that were established. It was tested over 12 trials and was able to successfully navigate to the goal positions using the stored experiences on all iterations. Therefore the proposed system successfully conducted SLAM and recalled the stored data to navigate.

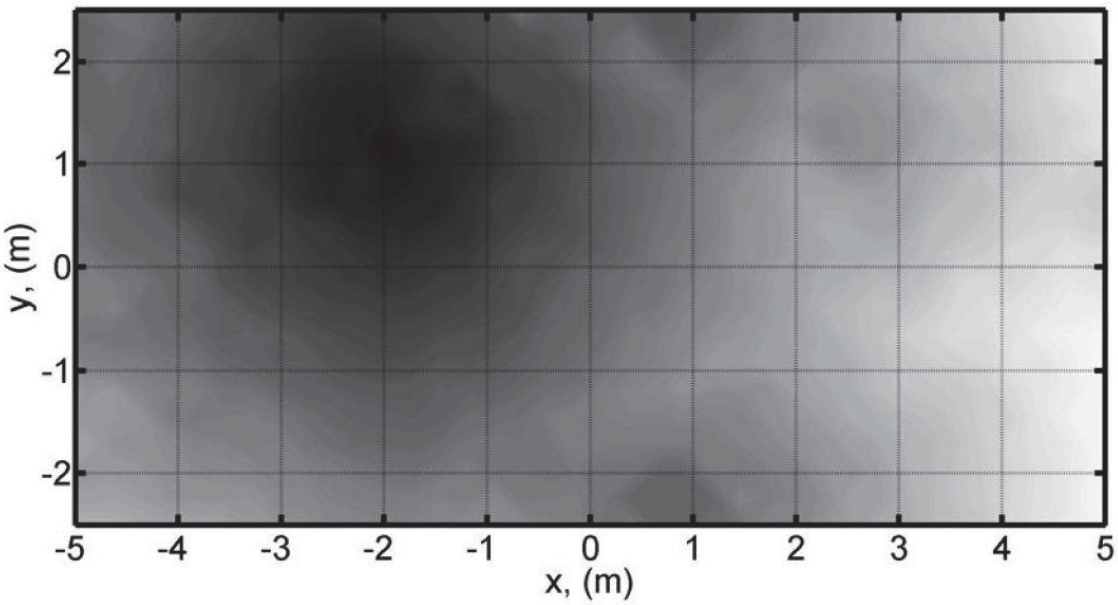

Figure 2.2: The temporal map of the ratSLAM navigation [10].

The system proposed by [13] uses a single-camera and the ratSLAM algorithm to conduct SLAM similar to the work introduced by [12]. This system contains the experience map and the experience transitions much like the other ratSLAM algorithms but this system conducts visual odometry in a unique way where only a single colour camera is used. Although similar this method calculates the angle using a cropped and grey-scaled image, where the number of pixels shifting to either left or right corresponds to an angular rotation of a number of degrees. The system required the assumption that there is no translation parallel to the camera lens plane. To calculate the speed of the system, optical flow and perceptual space, rather than actual space, was used. To obtain the rate of change the average array intensity difference values were

calculated between the current and previous views. This calculation also negated most of the effects of the rotation to obtain the translational velocity. In addition to calculating the angular and translational velocities the actual images were stored when a unique feature could be extracted. For each image, a vector of array differences is calculated and is compared to a threshold to determine whether the new view will be added or not. This system was tested in an outdoor environment and closely recreated an internal representation of the surroundings using the proposed algorithm to calculate the speed and angular velocities using a single camera.

2.3 - RGB-D SLAM Systems

For the RGB-D SLAM system there have been a number of different studies that have been completed, ranging from creating 2D and 3D maps to using feature vectors to determine landmarks [20-29].

2.3.1 - 2D Mapping

The evolutionary filter proposed by [30] constitutes a maximum likelihood to the robot's pose state at any given time to localise the robot in its environment. The best estimate of the current pose is selected by searching stochastically along the state space using evolutionary computation concepts unlike the method used in [31]. The algorithm has two stages, of which the first stage is to fit a state space model on the robot's pose estimates using a fitness function in the form of a sum of squared errors and a loss function which incorporates predicted and actual measurements. The second stage includes the use of a probabilistic model to eliminate any ambiguities to obtain the final pose estimate to localize the robot. The proposed algorithm was tested in an indoor environment such as corridors, offices, and laboratories. One issue that is raised that may arise in the system is the rate at which it converges to a solution. The system is compared with other, similar systems that use an evolving population, and they found that their system can converge accurately with a smaller population size per square metre of experiment space.

The work introduced in [31] implements a single leader robot with two laser scanners and two other robots with a laser scanner each to solve the SLAM problem using multiple machines. The leader is used to create a 2D and 3D map of the environment that uses a full posterior of the poses instead of using a maximum likelihood representation to incorporate the scans into a 2D map. To reduce the error that the

odometry is prone to, this system applies a backwards correction step once a similar feature is found which initiates loop closure. This can be contrasted to the method proposed by [30] which constantly calculates an estimate of its current pose so the error accumulation is reduced. The error correction for this solution is divided into three steps where the first step involves calculating the size of the loop. The second step applies the correction proportionately among the poses and the last step applies the correction iteratively to approximate the maximum likelihood. The application of multi-robot SLAM is also tested by setting the two other robots to first localize itself within the map created by the leader. Further mapping of the environment is conducted based on the first map created. The 3D mapping system is implemented by the first robot which scans vertically and calculates walls by filtering the outliers using an expectation algorithm first. This is followed by the filtered points being simplified to create a polygonal mesh of the environment. The introduced system only added views into the map every two metre intervals although all scan data was used when localising the robots. During testing, the robot was tested in an indoor environment and was able to create an accurate map, although it is mentioned that the robot would not be able to create a map without the odometry information.

The method used in [24] produces a solution to the SLAM problem by creating a map of the environment using a two-stage algorithm, where the first stage involves recreating the surroundings in sections and the second stage involves the usage of the map assembled. The mapping system introduced, is different to other methods like [21, 30, 31] because they divide the total map into smaller segments which can be processed and matched separately, rather than trying to compare and match all existing points in the map. The first stage is unique where the use of local and global segments is implemented to reduce the complexity of map creation. The robot uses a Microsoft Kinect sensor and the Speeded up Robust Features (SURF) algorithm to obtain 3D feature information from the environment which is used to create a 3mx3m local feature map. As the robot leaves this area the robot creates a new 3mx3m section as the new local map. The previous segment is stored into global space and as new local maps are created, the system transforms these to match the coordinate system of the global map. This system uses the EKF algorithm to conduct SLAM and to detect loop closure where they compare the new local map’s global pose with previously created local segments, and if there is a match between any maps the algorithm switches to the usage stage

instead of creating a new map. The proposed algorithm was tested in an indoor rectangular environment and successfully internalised the map in segments. Once loop closure was detected, the global map was created and the robot successfully proceeded to the usage stage.

The system introduced in [21] uses 3 Microsoft Kinect cameras to monitor their environment and conduct SLAM. Two of the sensors face outwards and are positioned horizontally to obtain 2D wall scan data, while the other faces the roof to obtain 3D data for scan matching and visual odometry calculations. The two horizontal Kinect sensors have had their covering removed and one of these is flipped upside down so the blind spot directly in front of the robot from the RGB colour stream is reduced. They use these two devices to obtain 2D laser-like data which is used to create a 2D map of the environment. The map is created using an open source map building software called Gmapping which is used on top of the ROS framework, which is also open source. To conduct visual odometry, the upward facing Kinect obtains RGB colour and depth data to determine the robot's pose which is a significantly different method compared to [30, 31] which estimates their poses using a maximum likelihood or full posterior. The different textures and lines on the roof is detected and extracted for features first using the SURF and then the Features from Accelerated Segment Test (FAST) algorithms. The depth data is used to determine the plane of the roof. Any feature extracted that does not belong to the roof's plane is removed to increase the accuracy. A Hough transform is then applied to extract lines and to determine the direction using a voting scheme. The final odometry can be calculated by using the ICP method on two consecutive frames. The proposed robot system was tested in an office environment and it was shown that visual odometry was superior to conventional wheel odometry. The results show that their robot accurately creates a map of the environment.

2.3.2 - 3D Mapping

The work introduced by D. Marzorati, M. Matteucci, and D. G. Sorrenti [32], places the most importance in data association using the sensors rather than odometry pose estimation. It is proposed that a trinocular vision system with a robust data association method be used to overcome linear and Gaussian approximations. The particle based system proposed uses the particles to represent possible 3D extrema points from an image segment which is different to the method of using point cloud data in [22]. These

particles are sampled and compared to the particle points in the map using three tests. The first is to check whether the Mahalanobis distance falls within the specified threshold α. This threshold is the minimum required particles with respect to each extrema segment. The second test computes and compares the angle with a given threshold and the last is a comparison to check whether the extrema point falls within or out of the map's frame while still being within the view frame. The samples that succeed all the tests are stored and rearranged on the order of matches in decreasing order and only the first pair sample in each segment is considered as a solution. The proposed system is compared to existing Gaussian approximation algorithms and is found to correctly associate features within the environment better than the existing systems.

The work conducted by [22] conducts SLAM by reconstructing the 3D environment using the Microsoft Kinect's depth data. They tested two algorithms using the Kinect sensor to verify the applicability to mobile robots. The two algorithms were 6Degrees of Freedom (6DoF) 3D RGB-D SLAM and the Kinect sensors Fusion SLAM. The 6DoF method used the SURF and RANSAC algorithms to obtain features from the environment while scanning two consecutive images. The resulting graph was optimized using the Hierarchical Optimization on pose Graphs – Manifolds (HOG-Man) algorithm. They optimized and tracked the sensor's transformations using the ICP algorithm and used Truncated Signed Distance Function (TSDF) to match two consecutive images. To test the system it was mounted on a tripod and rotated on the spot in an indoor room. Their system, although noisy, produced an accurate recreation of the environment.

The system created in [23] conducts robotic SLAM using a Kinect sensor to create a highly detailed 3D map of the environment similar to the method proposed in [22]. The process is divided into three main stages, the front-end, back-end and the final map representation stage. The front-end is used to extract the geometric relationships between the robot and the environment at different points in time. They calculate the robot's motion by first extracting the feature vector from the RGB colour image and then this feature vector is projected to a 3D feature vector by aligning the pixel data with the depth data. These feature vectors are matched using Scale Invariant Feature Transform (SIFT), SURF and Oriented FAST and Rotated Binary Robust Independent Elementary Features (ORB) to test the effectiveness of each feature matching algorithm.

The feature data that produces a match is filtered to reduce the number of false positive matches using the RANSAC algorithm. The remaining feature vectors are stored in the backend for optimization using the g2o algorithm based on a graph of vectors. The maximum likelihood solution of the robot's motion trajectory is obtained after the optimization which is then used to project the data into a common coordinate frame using a voxel based map representation called Octomap. By representing the map as a voxel grid, the redundant information can be compressed to show the general layout of the environment for navigational purposes which is a strength compared to a point cloud representation used in [22] This robot was tested in an indoor environment using the SIFT, SURF, and ORB algorithms which produced varying results. The SIFT algorithm produced the least mean error followed by SURF with ORB having the most amongst the three. With respect to the time taken to process, ORB was the fastest followed by SIFT then SURF. The system with the algorithm had reduced mean error compared to the same system without the proposed solution, therefore solving the SLAM problem.

The system employed by [20] conducts edge detection SLAM by obtaining and processing the data from RGB-D sensors which is similar to the work conducted in [22, 23]. First the RGB colour image is used to obtain edge information on occluded, occluding and boundary edges (as shown in Figure 2.3). This edge information is then back-projected into a 3D edge by using the depth data. They then calculate the high-curvature edges by first calculating the edge gradient normal from the depth data by using a variant of the Canny Edge detector and then applying non-maximum suppression and hysteresis thresholding afterwards. After the edge detection is completed, edge-based SLAM is conducted using the ICP algorithm and Georgia Tech Smoothing and Mapping (GTSAM) for optimization. Pose estimation is conducted by using the transformation data produced by the ICP algorithm and loop closure detection which passes the pose result from scan matching between the current, and stored, poses through a threshold. This system was tested in an indoor environment and the edge detection SLAM was comparable to existing systems while outperforming them in two datasets, the plant, and the room. The advantage of using an edge detection system is that they can predict the shape of objects, while the systems such as those in [22, 23] cannot do this.

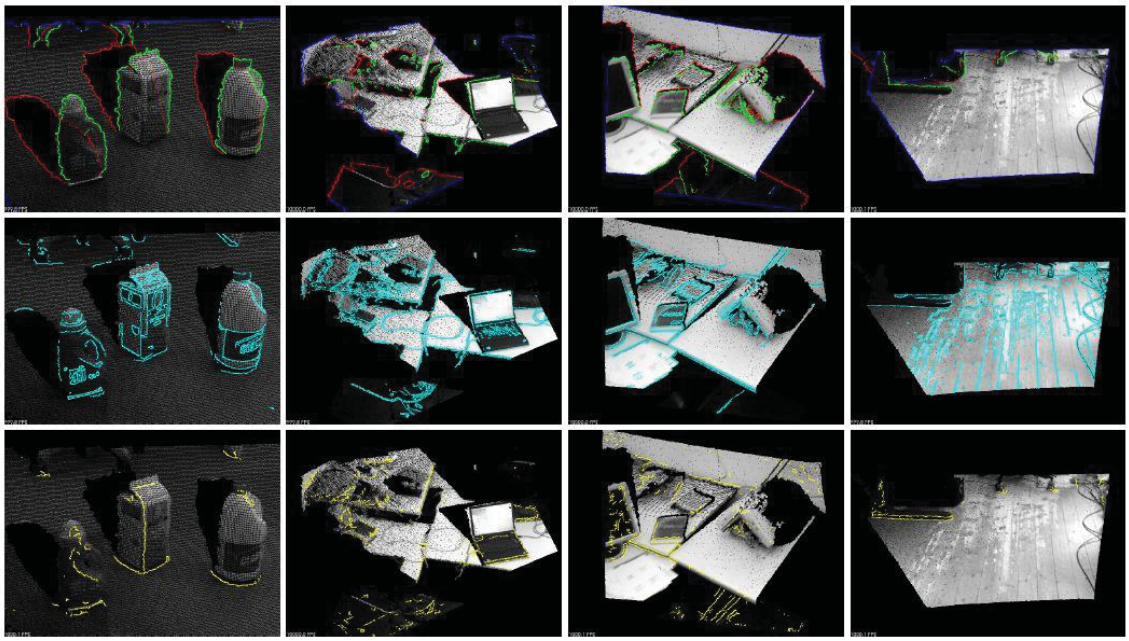

Figure 2.3: Visualization of different edge detections [20].

The work presented in [25] tracks points and planes in the environment to conduct key-frame SLAM. The system conducts visual odometry to estimate the camera's current pose, which is further used to extract the points and planes from the environment. The use of Lucas-Kanade's optical flow method is used to project landmarks obtained in the 2D image into the 3D plane by implementing the camera calibration parameters. To extract the plane data from the environment the use of the camera's predicted pose is incorporated. To determine whether planes in other images are overlapping the current plane the RANSAC algorithm is used. If the current image does not match with the other images within a threshold then the new plane or point data is added to the map. The proposed method is tested in two indoor scenes where one has repeating textures and the other has feature rich views. The results show that the proposed system using points and planes instead of just points is more robust to repeating features within the environment. Therefore their system successfully conducts SLAM.

2.4 - Analysis and Summary of Findings

Each topic covered in this chapter introduces a number of unique systems and solutions to the SLAM problem. Many older studies used laser scanners in different configurations to obtain data for 2D maps such as the research conducted in [1, 3, 10, 11, 21, 24, 30] while newer methods used a combination of 2D and 3D data to create maps with 3D data or 3D maps using all the information gathered such as the works conducted by [1, 12, 20, 22, 23, 25, 32]. The use of 3D data has improved the overall effectiveness of mapping by adding finer details about the environment for scan matching. Therefore by using a sensor that can combine the 3D data and also extract features from the surroundings like the sensors used in [20-29], an effective robotic system can be created.

Separate from the way the data is obtained, the work introduced in [3] groups the scanned data to form new landmarks within a processed image, which has similarities to the studies conducted in [1, 23-25]. The similarities are that they group data to form compressed segments. The work by [3] uses this to conduct fast loop closure while [1, 24, 25] create semi-large regions which represent the surroundings, and the study done by [23] creates smaller sections of the environment reducing the amount of data while maintaining some semblance to its actual shape. Other methods which resemble different forms of data reduction is done by [4, 20, 32], where the study in [4, 32] extracted less data from the environment by concentrating on only vertical and horizontal lines using a laser and webcam. The work conducted by [20] starts by only extracting edges from the image instead of getting each point in the view. A contrasting method to the concept of reducing the data size or extracting less data, is the research conducted by [10, 21, 22, 31] which all uses point cloud data. Another method which reduces the total amount of data collected is the ratSLAM systems in [10-19], which uses grid cells and firing fields to determine when to store location data. This method has been shown to be an effective and light weight alternative to reducing the data size while also giving the information positions within the map.

All of the analysed methods store their data in maps, so data can be recalled. The two main methods of map storage are to use a grid or a topological representation. The grid system is a quick way to represent an environment but to alter portions of the map, the whole grid needs to be traversed compared to a topological map method which only

requires the knowledge of a transform from one node to another to alter or fix orientations of views. Most systems that have been created recently use the topological mapping method which includes the ratSLAM and RGB-D SLAM systems in [10-29], while the systems that use the grid mapping are [3, 24, 30, 31].

Although the mapping methods of the RGB-D and ratSLAM systems are similar, the differences in the way that the odometry is calculated can affect the effectiveness of the map, which varies across the different studies. The way that the odometry is calculated in the different solutions is visual odometry such as those studied by [12, 13, 20, 23] and actual odometry by [1, 10, 11, 32]. The actual odometry is to obtain the data straight from wheel or joint information, which is prone to accumulated error and sudden drifts, but is much more stable to ambiguous or sudden changes in the image view or laser. While, although visual odometry is more accurate in terms of positioning, it is extremely vulnerable to ambiguous images and loss of data such as when there is a sudden drift of the actual robot.

By analysing the different systems that have been studied, the RGB-D method will be used for the rich, image features combined with depth data, and the mapping and navigation components will be taken from ratSLAM because of its more effective method of scanning and separating the images by positioning.

Chapter 3 - Robotic System Setup

3.1 - Hardware Components

The hardware components that were used in this study were assembled as shown in Figure 3.1. All of the robots programming was conducted on the Intel NUC where it connected to all of the other hardware components to control the individual robot behaviours and movements.

The components used for this project are listed in Table 3.1 shown below.

Table 3.1: Hardware and software components used in the project.

Mobile robot	P3-DX (Pioneer 3-DX)
RGB-D sensor	Kinect for Windows v1
Computer	Intel NUC5i7RYH, Ubuntu 14.04
Robot control library	ARIA 2.8.0
Kinect library	freenect_launcher, ROS package
Image processing library	OpenCV 2.4.8, OpenCV_Bridge, ROS package
Software IDE	ROS framework

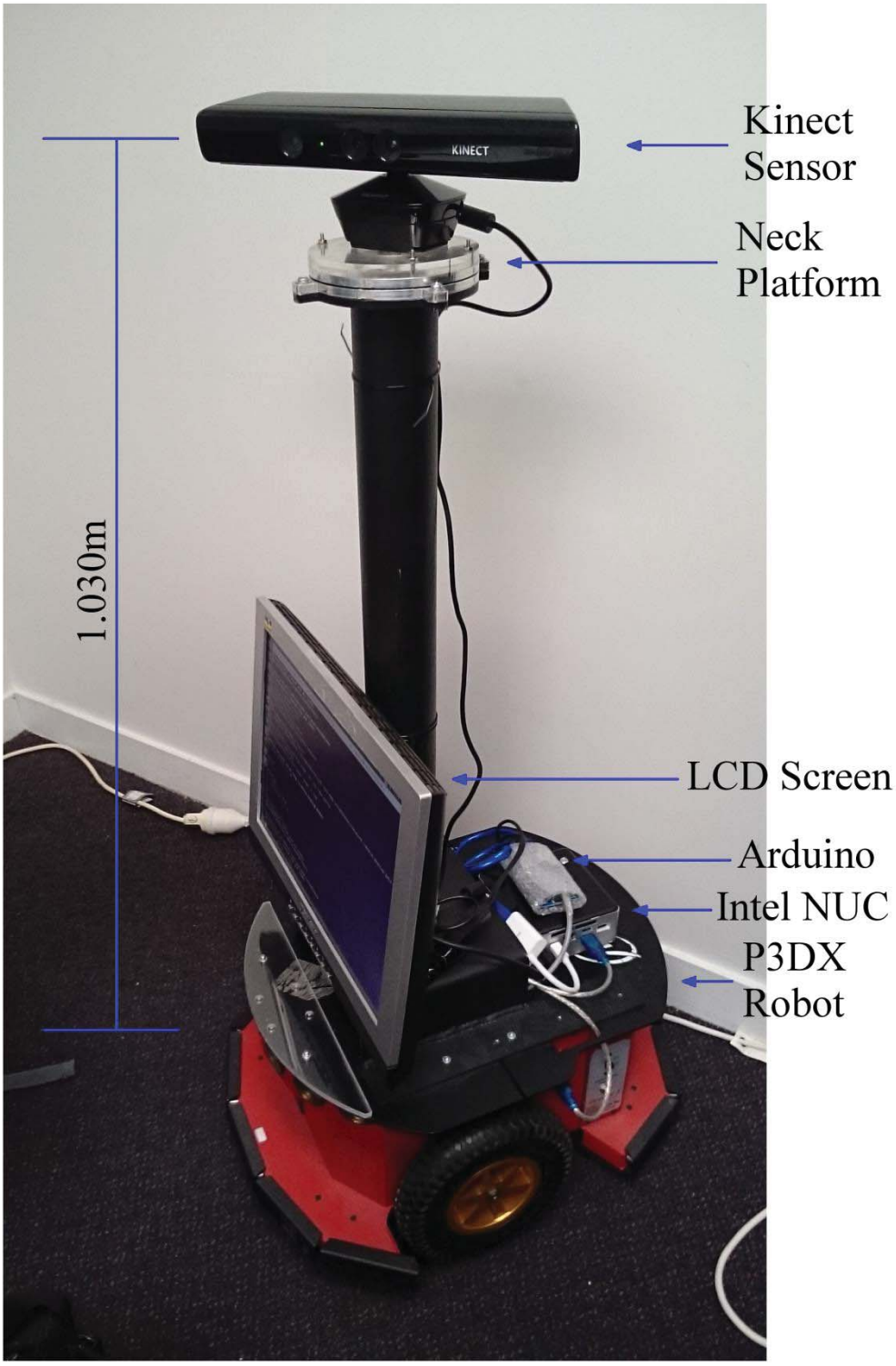

Figure 3.1: Robot assembled with hardware components.

3.1.1 - P3DX Mobile Robot Base

The P3DX mobile robot base is a differential drive robot base that comes with an on-board embedded microcontroller. The wheels have a diameter of 19.5cm as shown in Figure 3.2 and the robot comes with a 500-tick encoder. The robot also has 8 forward facing sonar sensors (as shown in Figure 3.3) and 10 bumpers, with 5 covering the front and the remaining 5 covering the back for any direct contact with obstacles. The bumpers are arranged symmetrically on both the front and back ends (as shown in Figure 3.4). This mobile robot base can reach speeds of up to 1.6 metres per second and can carry a total weight of up to 17kg.

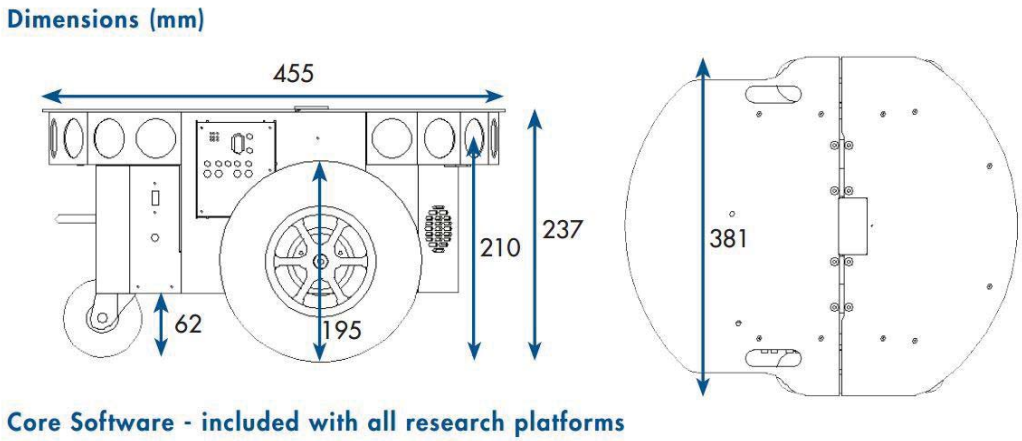

Figure 3.2: P3DX robot base dimensions [39].

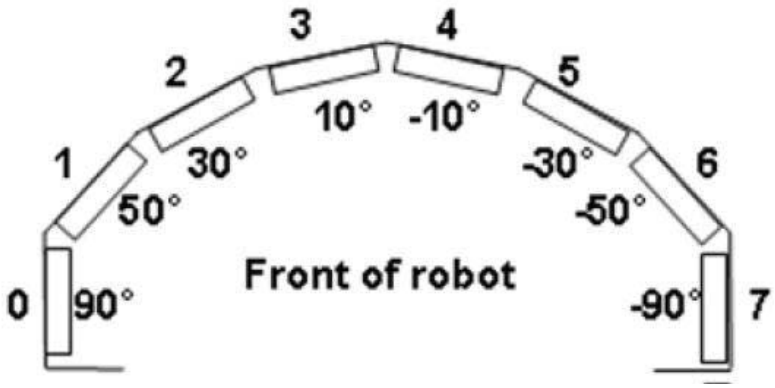

Figure 3.3: Top-down view of P3DX's front sonar arrangement [39].

Figure 3.4: Bumper arrangement on P3DX robot base [39].

3.1.2 - Microsoft Kinect Sensor

The Kinect sensor is similar to other ToF sensors such as laser or sonar sensors but is different in that it creates a structured light array (as shown in Figure 3.5). The Kinect sensor emits a structured Infra-Red light array in a speckle shape by passing through a lens with the particular pattern, which is then, measured using a Complimentary Metal-Oxide Semi-conductor (CMOS) based image sensor array [33]. The speckle pattern that is projected onto the object is viewed from multiple angles which are cross-correlated to calculate the depth data. All of the data is converted to a depth array by the Kinect's on-board hardware to achieve a frame rate of 30 frames per second with a total view angle of 43° vertically and 57° horizontally [40].

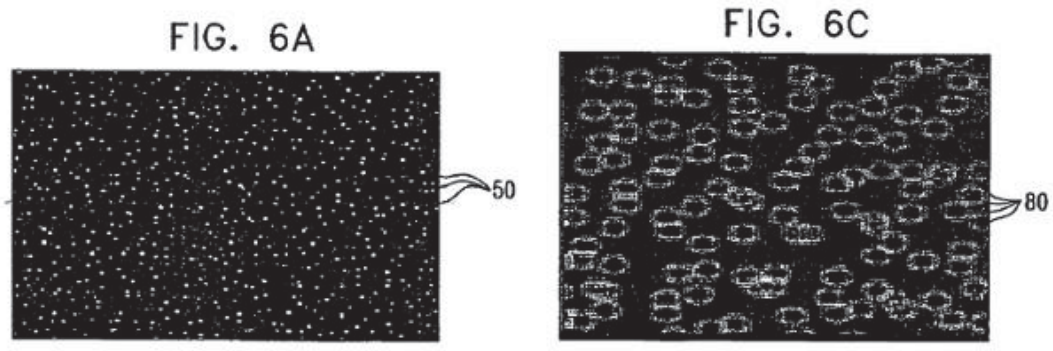

Figure 3.5: Speckle image produced by the Kinect [33].

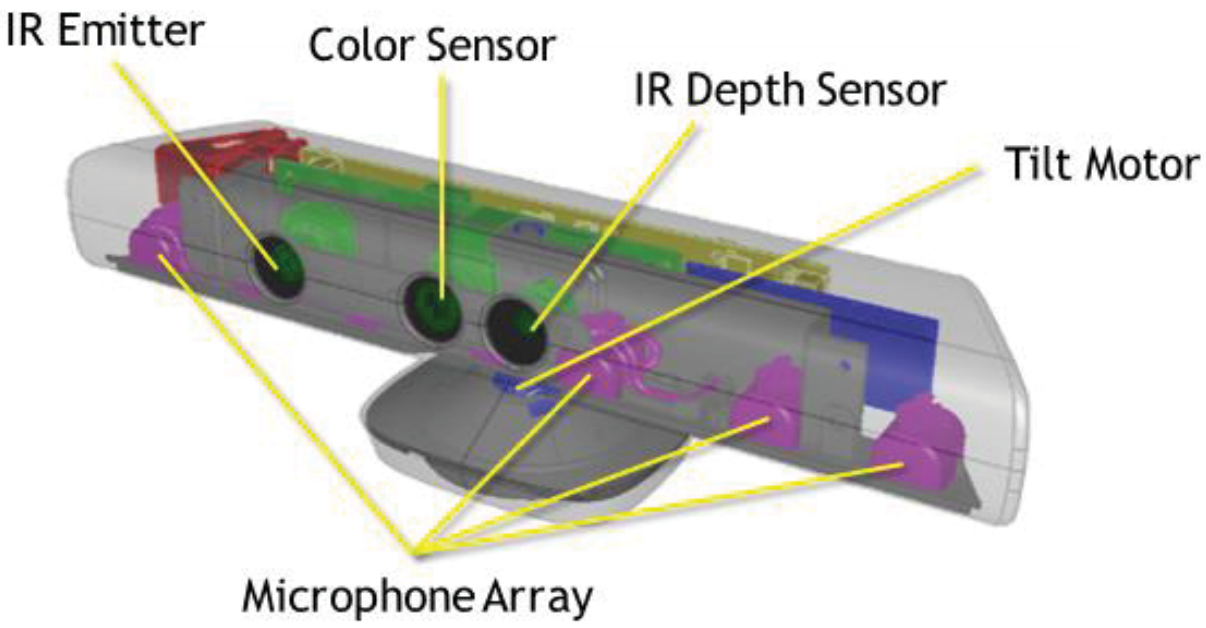

Figure 3.6: Kinect Sensor components [40].

3.1.3 - Custom Neck Platform

The custom-built neck platform extends the total viewing height of the Kinect sensor to 1.030m, which allows the system to better view feature-rich landmarks such as posters, billboards, and objects on top of tables and desks. The neck structure has a height of 0.700m and the built-in servo motor has a total turning range of 180°. When the robot neck is facing the default angle or straight ahead, the angle is at 90°.

3.1.4 - Arduino Neck control

An Arduino microcontroller was used to control the servo motor in the custom neck platform which was used in obstacle avoidance and image node searching while running the main program. The Arduino was programmed to receive a message via the ROS topics that are published by the main robot node in the ROS framework, explained in the Software Components section of the chapter. The Arduino controller knows what the neck's current angle is and calculates the difference between the current and required angle to rotate the necessary amount and direction.

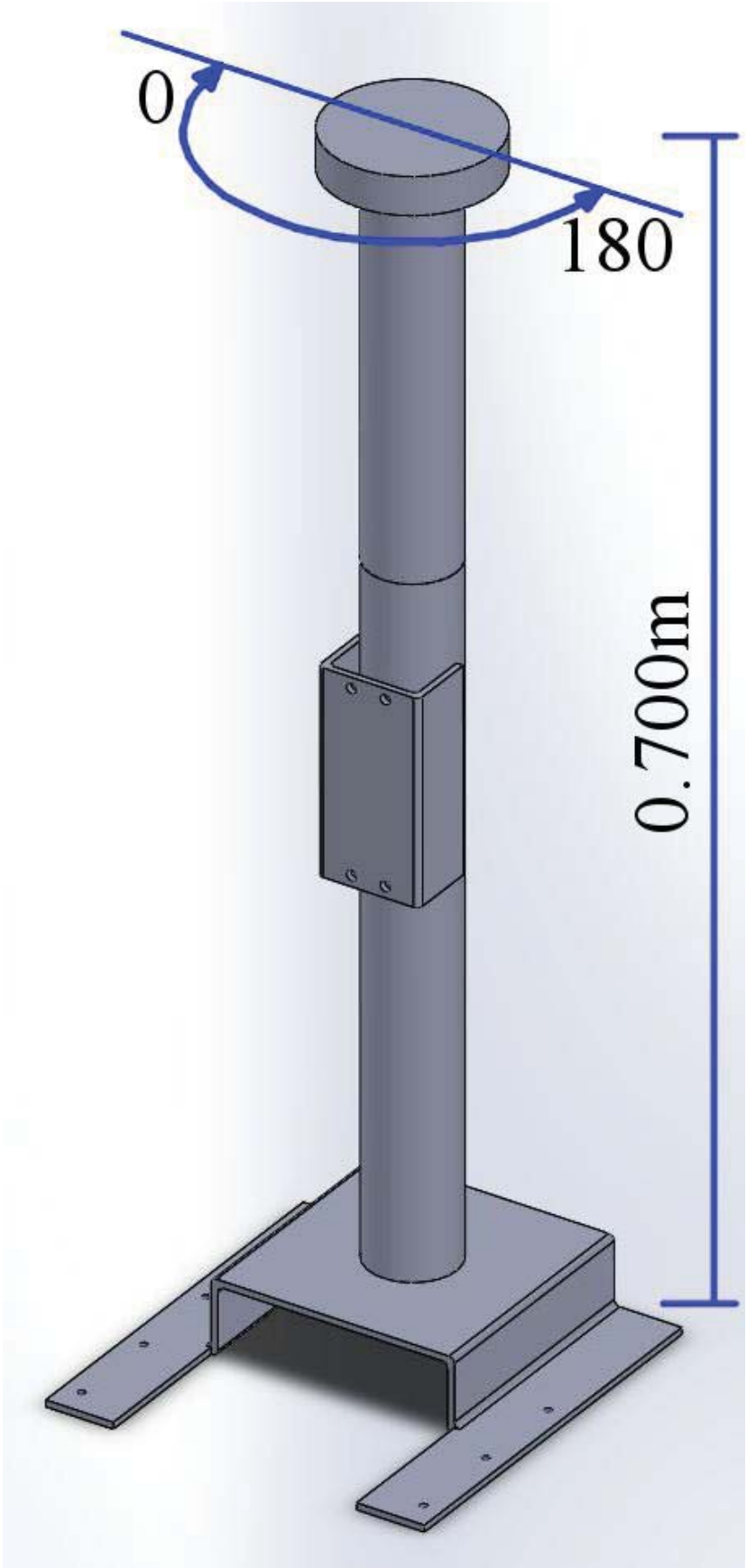

Figure 3.7: The neck platform height and the angle range.

3.2 – Software Components

The main software systems and frameworks used are ROS, ARIA, Standard C++, Qt, and Boost. The ROS framework was used for programming most of the robot's algorithms and core functions. The ARIA library was used to communicate to the P3-DX robot platform because the robot platform requires the use of this library. The standard C++, Qt, and Boost libraries were also used in programming the functions for the robot algorithm.

3.2.1 – Robot Operating System

ROS was used as the foundation framework to build the robot's algorithm and functions because it has the built-in ability to run multiple programs in different packages that can communicate between each other. The communication system used in ROS is not a direct computer to computer connection; it is a broadcast of a ROS topic, which can be viewed and read by any and all other packages on the same network. By separating out the programs as packages, the complexity of each package can be limited while still being able to increase the whole system complexity. This leads to easier maintenance of the system as any code failures in one package cannot directly affect the other packages. This allows for easily narrowing down the problem code. Also by using the ROS framework, packages created by other people can be easily incorporated into any project or system being constructed. The RosAria, freenect, and RosSerial packages were used in this project to form the cross system link between the hardware. The RosAria package was used to connect the ROS packages created and the P3DX's ARIA library. The freenect package was used to extract the Kinect's image streams and the RosSerial package was used to establish a connection between the ROS package and the Arduino microcontroller.

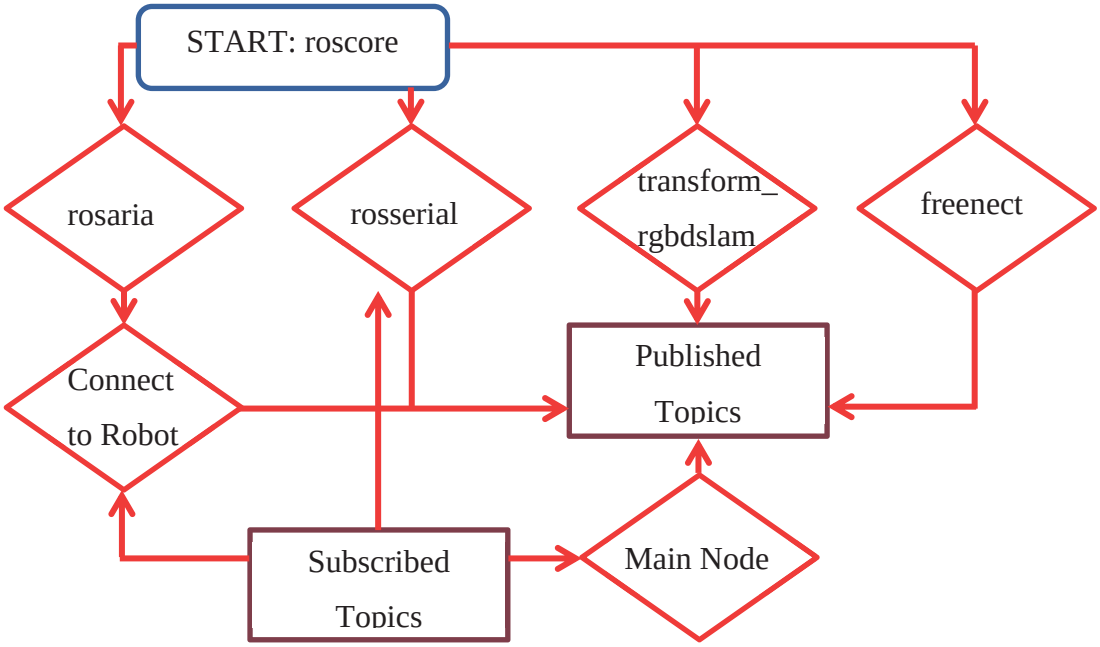

Figure 3.8: The ROS system architecture.

3.2.2 - ARIA

The on-board microcontroller on the P3DX mobile robot base is pre-programmed to receive and respond to the ARIA software library. This library contains the basic programming code required to initialize and control the robot's built-in sensors and motors. The ARIA software library is also built as part of the ROS framework and was incorporated using the RosAria package.

3.2.3 - Standard C++, Qt and Boost

The majority of programs written for this robot were written in C and C++. The Qt and Boost libraries were used for synchronization of the ROS topics being read in each code segment, and to prevent data corruption from multiple read and write threads and call-back loops.

3.2.4 - Feature Detection and Matching Implemented

3.2.4.1 - SIFT Algorithm

Scale Invariant Feature Transform (SIFT), is a feature detector and descriptor algorithm to detect and describe points of interest in an image [34]. These points of interest are invariant to scale, rotation, illumination, affine distortion, perspective and similarity

transforms, and noise. To extract the features from an image there are four stages. The first stage identifies potential key point candidates that are invariant to scale and orientation by searching over a multitude of different scales and locations using a difference-of-Gaussian. The second stage involves applying a detailed model of scale and location for each key point candidate depending on each key point's measure of stability. The third stage measures the key point's gradients and applies an orientation for future operations. The last stage involves measuring the key point's local gradients at a selected scale. The key points are then transformed to account for local shape distortion and varying illumination.

To conduct scan matching this method uses the nearest-neighbour algorithm to scan for features in the stored image database and compares it to features in a new image following the four steps for each new image. In a cluttered image the number of matches found can include false matches. The correct features are filtered out from the incorrect features by identifying a subset of key points that agree on a location, orientation and scale. The filter is applied by using an efficient hash table of a generalised Hough transform.

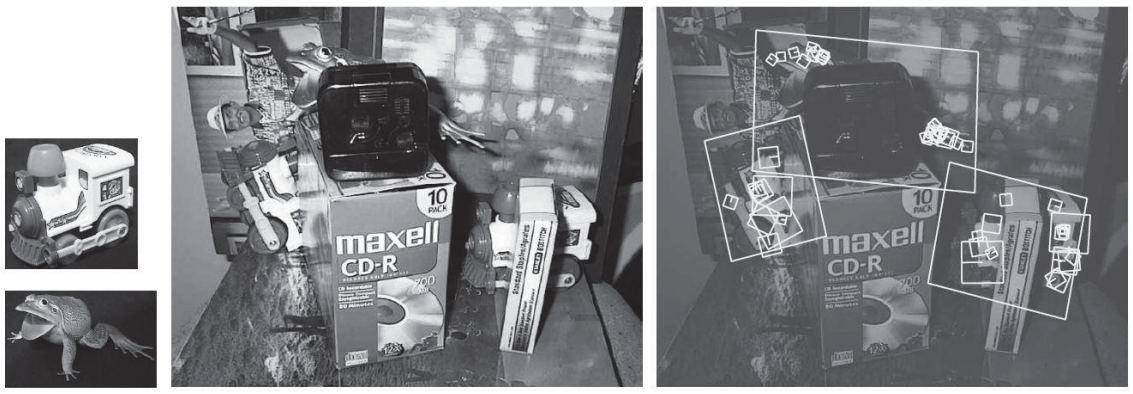

Figure 3.9: SIFT matching algorithm [34].

3.2.4.2 - SURF Algorithm

Speeded Up Robust Features (SURF) is a feature detector and descriptor that is used to match two images taken at different points in time or lighting conditions [35]. This method has three main steps in matching the images. First the key points in the image are detected such as corners, blobs and T-Junctions. The image key points are selected using the Fast-Hessian Detector, and when selecting these features, the repeatability of the key points is important. Secondly, the feature vectors are created using the key

point's neighbourhood, where the robustness of these features in noise and distortion is required for accurate matching. The final stage involves matching the feature vectors from the processed image to other images stored at earlier stages.

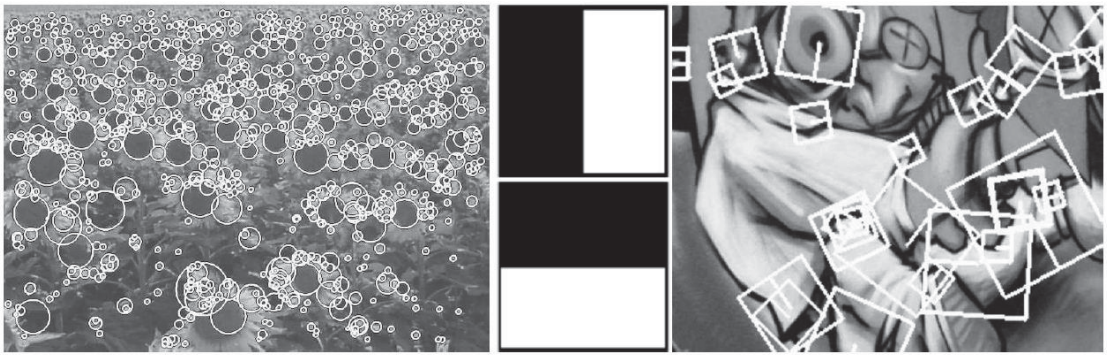

Figure 3.10: SURF matching algorithm [35].

3.2.4.3 - BRIEF Algorithm

Binary Robust Independent Elementary Features (BRIEF) is a feature detector and descriptor algorithm that takes points of interest in an image and converts it into a binary string for faster matching and efficient storage [36]. This method involves comparing pair-wise intensity values in an image patch which is computed after smoothing is conducted on the image. Due to the method taking only information from single pixels the smoothing process becomes important for performance stability and repeatability.

3.2.4.4 - FAST Algorithm

Features from Accelerated Segment Test (FAST) is an algorithm that is used to detect corners in an image. This is done by considering a circle of 16 pixels around the central candidate pixel and checking each surrounding pixel for changes in the intensity [37]. If there is N or more contiguous pixels with greater or less pixel intensity than the candidate, then that candidate is declared a corner. This algorithm has been improved by applying a machine learning method to address the first two weaknesses of the high-speed test which is that the high-speed test cannot be generalized for N less than 12 and the second test which is that the efficiency of the detector depends on the ordering of the selected pixels.

3.2.4.5 - ORB Algorithm

Oriented FAST and Rotated BRIEF (ORB) is an algorithm that combines the FAST and BRIEF algorithms [38]. The algorithm starts by using FAST to obtain the corner data from an image but because this does not produce multi-scale features, a scale pyramid of the image is used where the FAST algorithm is applied to each level of the pyramid. The corner features extracted are then oriented by applying an intensity centroid on each feature. This method assumes that a corner's intensity is offset from the centre, which is represented by a vector that is used as the orientation of the feature. Due to the BRIEF algorithm being vulnerable to rotation along the image plane it is steered by applying the orientation at discretized angles to calculate a lookup table from which to obtain the correct points to compute the images descriptors. After applying the steered BRIEF the correlation is increased and the variance is lost, so to recover from this, a training set of 300k points is used, and all possible image patches are enumerated.

Chapter 4 - Fusion SLAM System

The solution to the RGB-D SLAM and robotic SLAM problem is presented in this chapter starting from the elements taken from the analysed systems.

4.1 – RGB-D SLAM Elements

The RGB-D SLAM system used by [23] has a requirement that the images must be consecutive. This means that the total transform of the robot cannot exceed a specific value when moving or rotating. This inherently slows down the robot's ability to move in both translational and rotational directions. Also because the RGB-D SLAM method is dependent on consecutive images for the matching algorithm, which calculates the movement transform and the map construction, a sudden loss of images due to an unusually quick turn by the robot can lead to broken environment data being introduced into the map which can hinder further mapping processes and cause the robot's perceived location to jump from one point in the map to another. It can also be noted that the RGB-D SLAM constructed by [23] is vulnerable to repeating features or landmarks in the surroundings. When a feature or a landmark is repeated and the matching algorithm processes this image, the resulting transform and alignment of the image in the map space is incorrect due to being matched to a previously stored, similar image. This unwanted matching of ambiguous features and landmarks causes the system to lose track of its actual location in the environment. The RGB-D SLAM system layout is shown in Figure 4.1.

Although the RGB-D SLAM system has some inherent features that are unwanted, the strong aspect of the algorithm is that the 3D features from the environment lead to strong matches due to the orientation in 3D space unlike standard 2D feature matching algorithms.

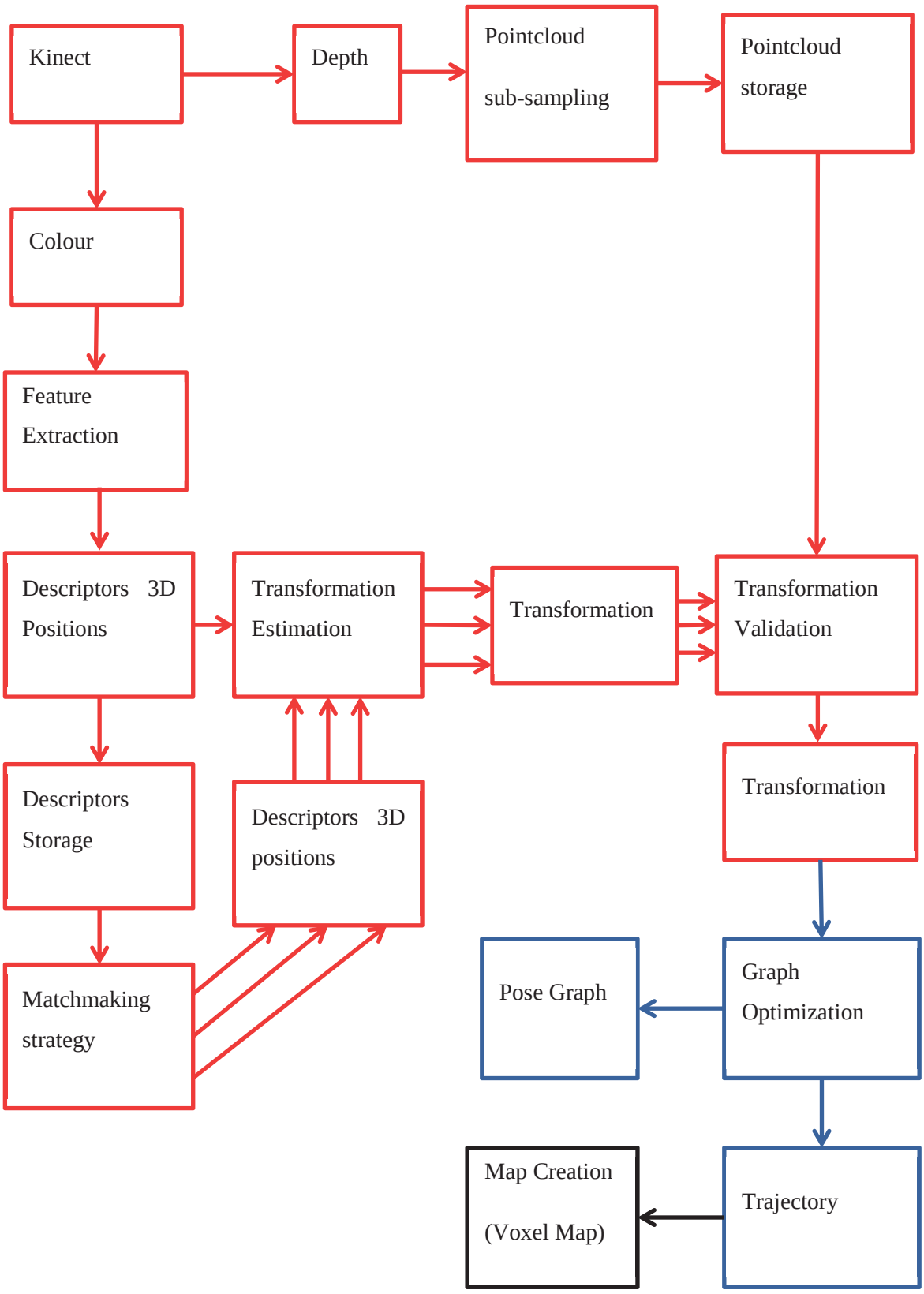

Figure 4.1: RGB-D SLAM system layout.

4.2 - RatSLAM Elements

The study conducted to determine the path planning and navigation of rodents has resulted in the discovery that the mammals brain relating to locomotion and memory react to the environment at specific intervals in the environment by receiving feedback from its sensors, such as from the rodents heading pose and images from its eyes. This led to the implementation of firing fields, grid cells and place cells to conduct SLAM for robots [10]. From the ratSLAM system the grid cell and firing field concept was incorporated into this project. The ratSLAM system layout is shown in Figure 4.2.

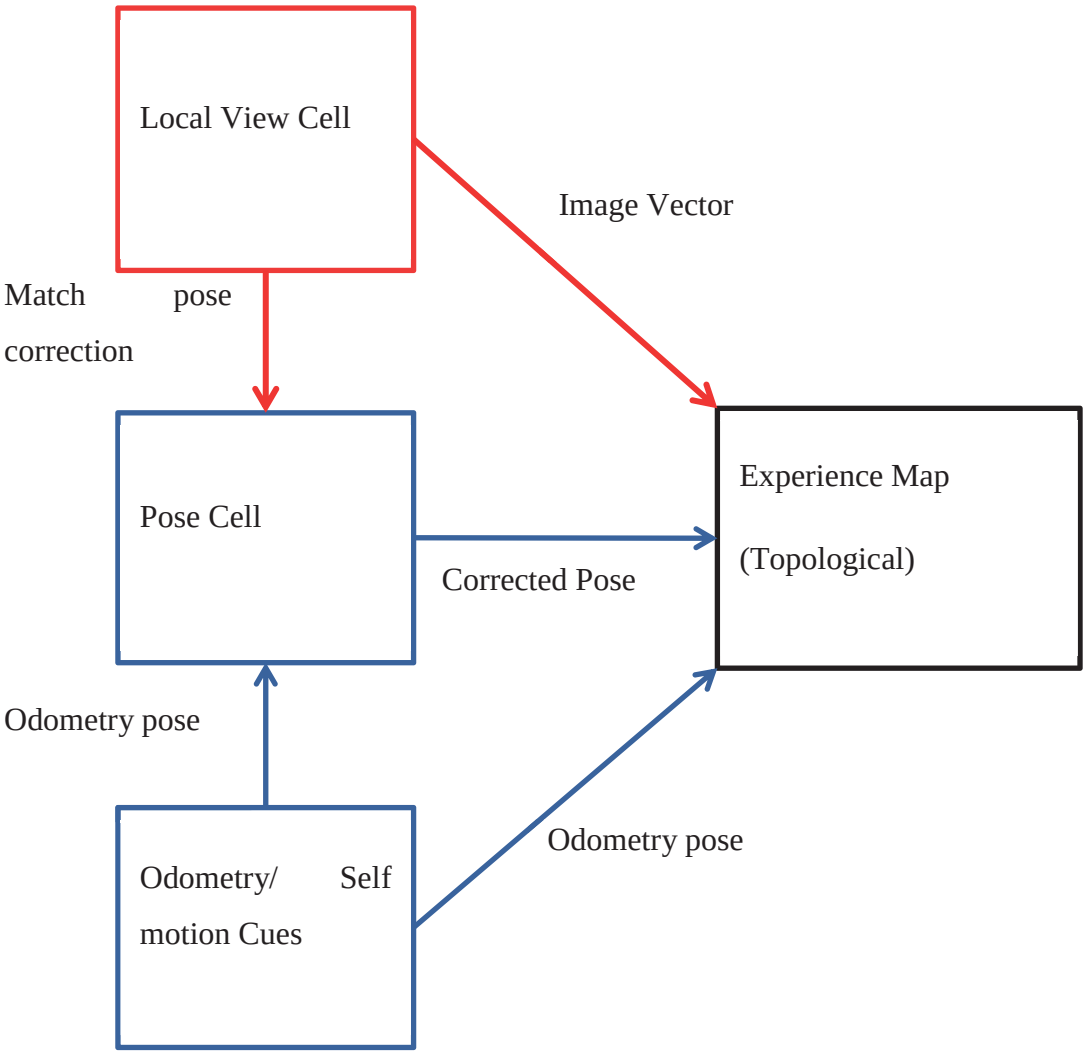

Figure 4.2: ratSLAM relation between local view cells and pose cells.

4.3 - System Architecture

The robot conducts SLAM in a similar process to the RGB-D systems front-end where it acquires and processes the data (Figure 4.3). In addition to acquiring the sensor information, this step also obtains the odometry data from the robot's wheel encoders, which are incorporated into the image nodes and odometry poses to initialize the grid cell and firing field nodes. The robot created for this study uses a combination of the grid cells, firing fields, odometry data and image nodes to conduct SLAM where the grid cells and firing field's concept is taken from the ratSLAM system to form the Fusion SLAM algorithm.

The data streams that the robot incorporates into the main loop are read from the topics published using the ROS framework. When the RosAria node is started to allow the main node to connect to the robot base, the sonar, bumper and odometry topics are published from the RosAria node. In addition to the RosAria node the transform_rgbdslam node is initiated so the system obtains the correct transforms between the frames. The last two nodes started are 1) the rosserial node, which is used to communicate with an Arduino that controls the robot's neck when searching for obstacles and image nodes and 2) the main node, which is run in conjunction with the freenect node to do the main processing. The freenect node publishes the Kinect's depth and colour streams which the main node connects to.

After all of the nodes have been started the main node subscribes to all of the published topics to begin the processing as shown in Figure 4.4. The main node implements a call-back system when subscribing to the topics so the data streams can be read and processed asynchronously until it is critical for synchronization as shown in Figure 4.5.

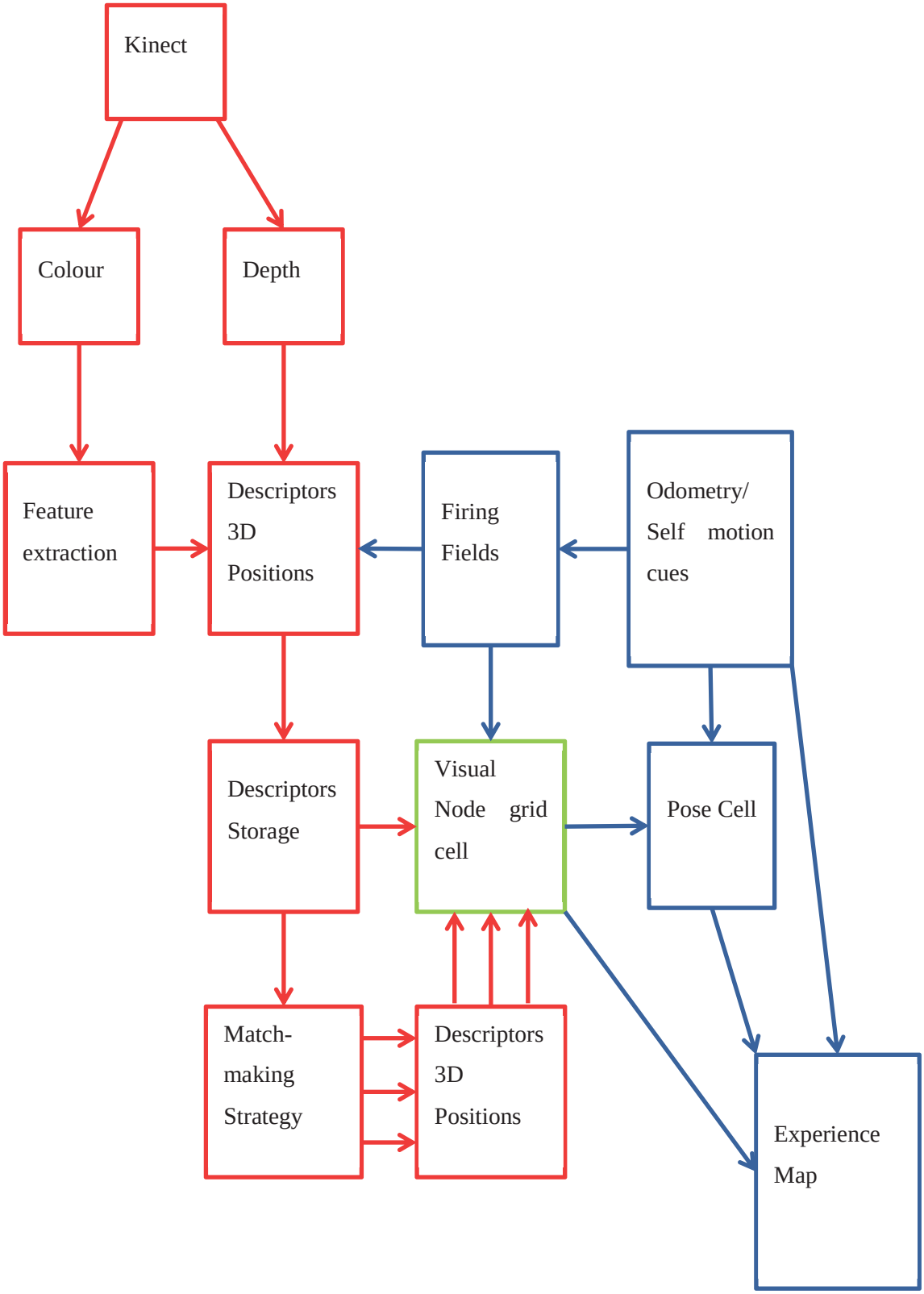

Figure 4.3: RGB-D SLAM and ratSLAM Fusion system layout.

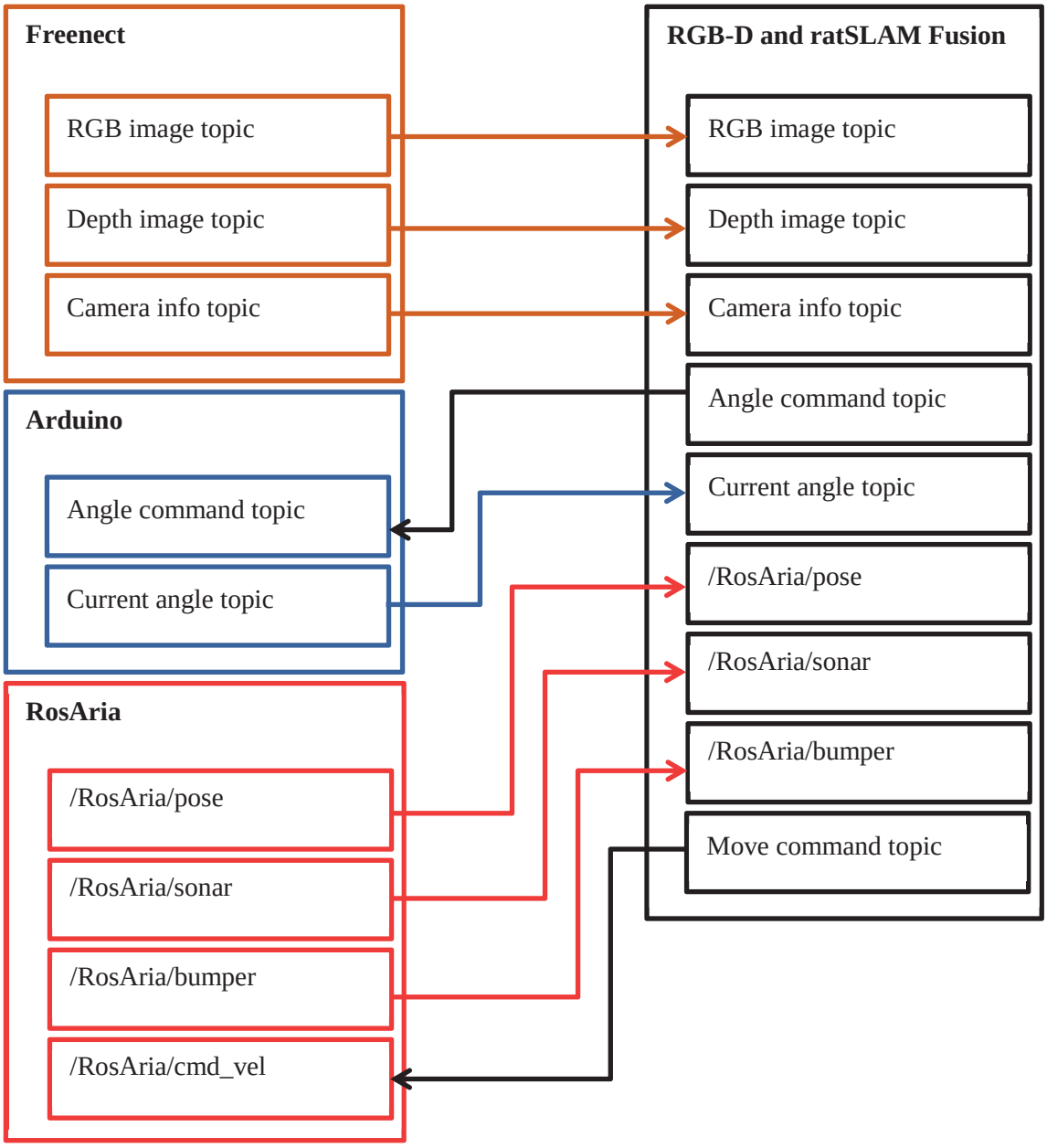

Figure 4.4: Topics layout in the robot system.

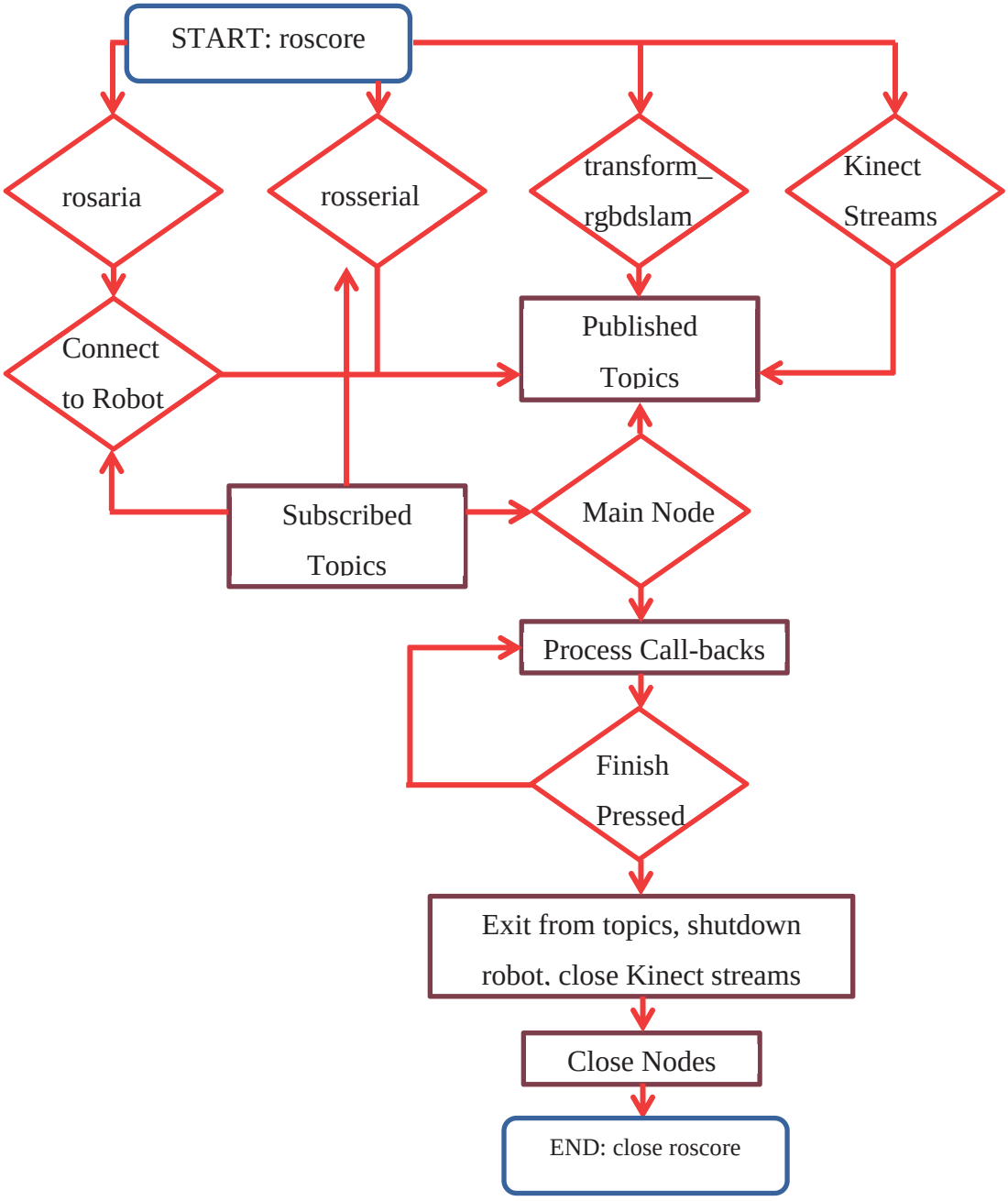

Figure 4.5: Main Processing Loop for Fusion ratSLAM.

4.3.1 - Image Call-back Loop

The image call-back loop extracts the RGB and depth data and stores it as an opencv matrix. Once the stream data is in a format where image processing can be applied the image is checked for a sufficient number of features and is checked to determine whether the features that can be extracted fall within the Kinect sensor's effective range. If both the number of features and the depth is good, then a further check to see whether the robot is in a firing field is conducted. The check for a firing field is implemented to react in a manner similar to the competitive attractor network of the ratSLAM algorithm. The three checks filter the image to reduce the total complexity when scan matching is conducted in the later processing stages and also allows the system to distinguish ambiguous images at different locations by using this firing field method.

Once all of the checks are passed the image node is created with the current pose of the robot and neck. The image node converts the image to grayscale and extracts the 2D features from the converted image. The 2D features are projected to 3D using the depth image and the point cloud is created and stored in the node. Three different feature extraction methods were used in this project which were SIFT, SURF and ORB.

This image node is then passed to the image storage function. This function adds the current grid cell id to the image node and compares it with other nodes using the RANSAC algorithm and the nearest neighbour distance to conduct scan matching between the 3D features. After the scan matching has finished comparing the current node with the stored nodes, the transform to the nearest image matched is stored to determine how much error the robot has accumulated.

Table 4.1: Pseudocode of the image callback process

Algorithm 1 "Image callback process"

Requires: 3 image matrices I, D and D_m. 1 header H. 1 parameter matrix C.

```
1  if process_frame == true then
2    process_frame ← false
3  else if paused then
4    return
5  end if
6  good_distance ← f_n("check distance", D)
7  good_features ← f_n("check features", I, D_m, detector)
8  if good_distance == true and good_features == true then
9    if start_map == false then
10     Start_map ← true
11   end if
12   if in_firing_field == true then
13     if odometry frame is processed then
14       odom_frame ← odom_topic
15       base_frame ← base_topic
16       error ← f_n("check transform", odom_frame, base_frame)
17       if error == no_error then
18         odom_pose ← transform
19         neck ← neck_angle
20         robot_stationary ← is_stationary
21         image_node ← f_n("image node", I, D, D_m, C, H, detector, extractor, odom_pose, neck,
   robot_stationary)
22         image_node_added ← f_n("add image node to grid cell", image_node)
23         if image_node_added == false then
24           delete image_node
25         end if
26       else
27         print("warning, no transform")
28       end if
29     end if
30   else
31     print("not in firing field")
32   end if
33 end if
```

4.3.2 - Odometry Call-back Loop

The odometry call-back loop subscribes to the sonar, bumper, and the odometry topics and synchronises them to be processed at the same time. The first few odometry call-back loops will be ignored because the image call-back needs to confirm that the current view is adequate to store as the first node. Once the image call-back has confirmed that the current pose is good then the odometry pose creates the first grid cell point with the firing field centred on it.

Table 4.2: Pseudocode of the odometry callback process

Algorithm 2 "Odometry callback process"

Requires: 1 pointcloud S. 1 pose matrix O. 1 array B.

```
1   if process_frame and pause then
2     return
3   end if
4   transf ← f_n("pose to transform", O)
5   sttransf ← f_n("stamped transform", O)
6   if start_map == true then
7     gridcell_pose ← f_n("gridcell pose", O)
8     gridcell_added ← f_n("in grid cell", gridcell_pose)
9     if gridcell_added == false then
10      delete gridcell_pose
11    end if
12    x_diff ← 0
13    y_diff ← 0
14    if transf.x != prev_pose_node.x then
15      x_diff ← f_n(abs, transf.x – prev_pose_node.x)
16    end if
17    if transf.y != prev_pose_node.y then
18      y_diff ← f_n(abs, transf.y – prev_pose_node.y)
19    end if
20    dist_to_prev_pose ← √(x_diff² + y_diff²)
21    if prev_pose == dist_to_prev_pose then
22      is_stationary ← true
23    else
24      is_stationary ← false
25    end if
26    prev_pose ← dist_to_prev_pose
```

```
27    if dist_to_prev_pose > threshold then
28      pose_node ← f_n("pose node", sttransf)
29      pose_node_added ← f_n("add odom to grid cell", pose_node)
30      if pose_node_addded == false then
31        delete pose_node
32      end if
33      prev_pose_node ← transf
34    end if
35  else
36    print("wait for good features and distance")
37  end if
39  neck_a ← neck_angle
40  robot_s ← is_stationary
41  close_obj_dist ← closest_point_distance
42  obs_ros ← closest_row
43  obs_col ← closest_col
44  comm ← comm_var
45  image_comm ← image_comm_var
46  string_comm ← string_comm_var
47  gridcell_mgr ← f_n("robot navigation", S, O, B, neck_a, robot_s, close_obj_dist, obs_row, obs_col,
    comm, image_comm, string_comm)
48  transform ← sttransf
49  tflistener ← f_n("set transform", transform)
```

To calculate the neighbouring grid cell positions, the grid cells were divided into 6 neighbours, which is similar to the ratSLAM competitive attractor network. The *x* and *y* pose of each neighbour was calculated using the equations in Table 4.3 below and the position of each new cell is as shown in Figure 4.6.

Table 4.3: Formulas used to calculate the grid cell neighbours where O is the current cell centre pose and d is the grid cell radius. The current cell centre x pose is O_x and y pose is O_y

Neighbour number	X pose	Y pose
0	O_x	$O_y + d$
1	$O_x - \sqrt{d^2 - \left(\frac{d}{2}\right)^2}$	$O_y + \left(\frac{d}{2}\right)$
2	$O_x - \sqrt{d^2 - \left(\frac{d}{2}\right)^2}$	$O_y - \left(\frac{d}{2}\right)$
3	O_x	$O_y - d$
4	$O_x + \sqrt{d^2 - \left(\frac{d}{2}\right)^2}$	$O_y - \left(\frac{d}{2}\right)$
5	$O_x + \sqrt{d^2 - \left(\frac{d}{2}\right)^2}$	$O_y + \left(\frac{d}{2}\right)$

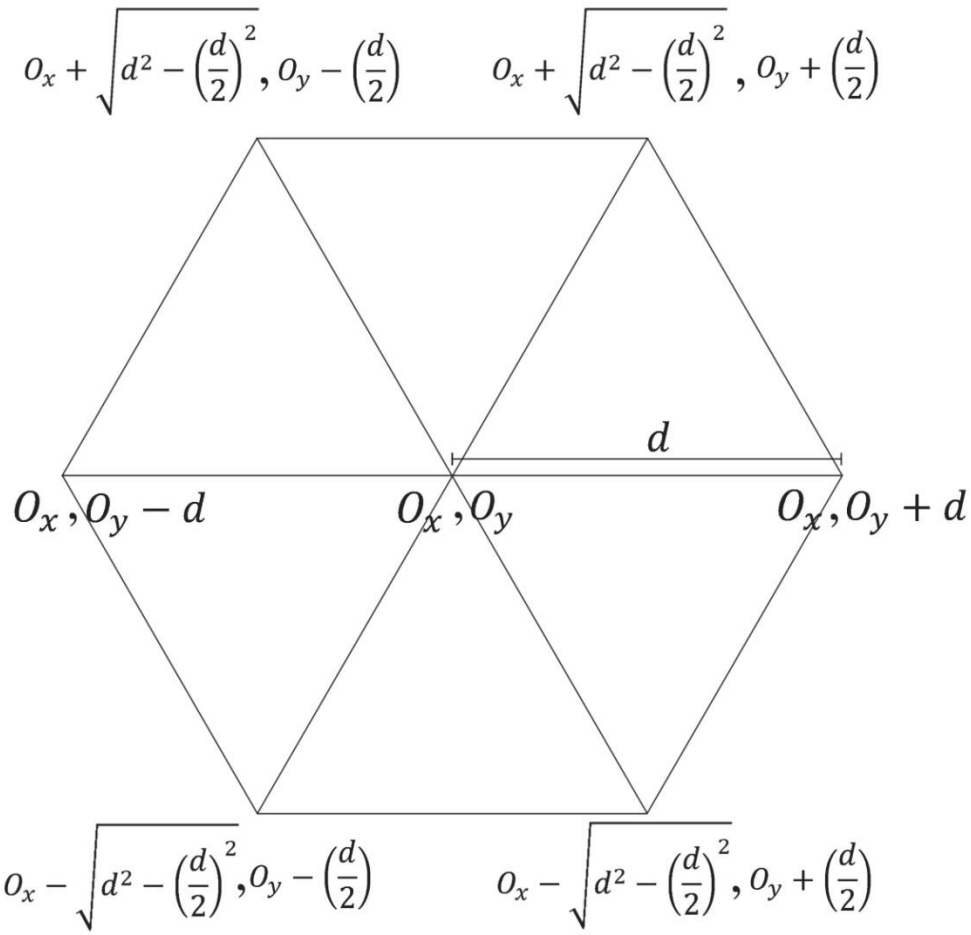

Figure 4.6: The grid cell neighbour calculation from the centre of the current cell.

The grid cell radius was obtained through experimentation where it was found that a radius of 0.3m produces finer navigational control when processing to reach a goal.

As the robot moves in the environment the distances to the neighbouring cells are repeatedly calculated and when the robot exceeds the boundary of the current segment the robot enters a new section which is the closest cell at the time. This new block is added to the grid map and the new neighbours are calculated repeating the process to store new positions. As new segments are added to the map the relationship between neighbours are stored as being adjacent to each other. This adjacency is used in the navigation process as shown in Figure 4.7.

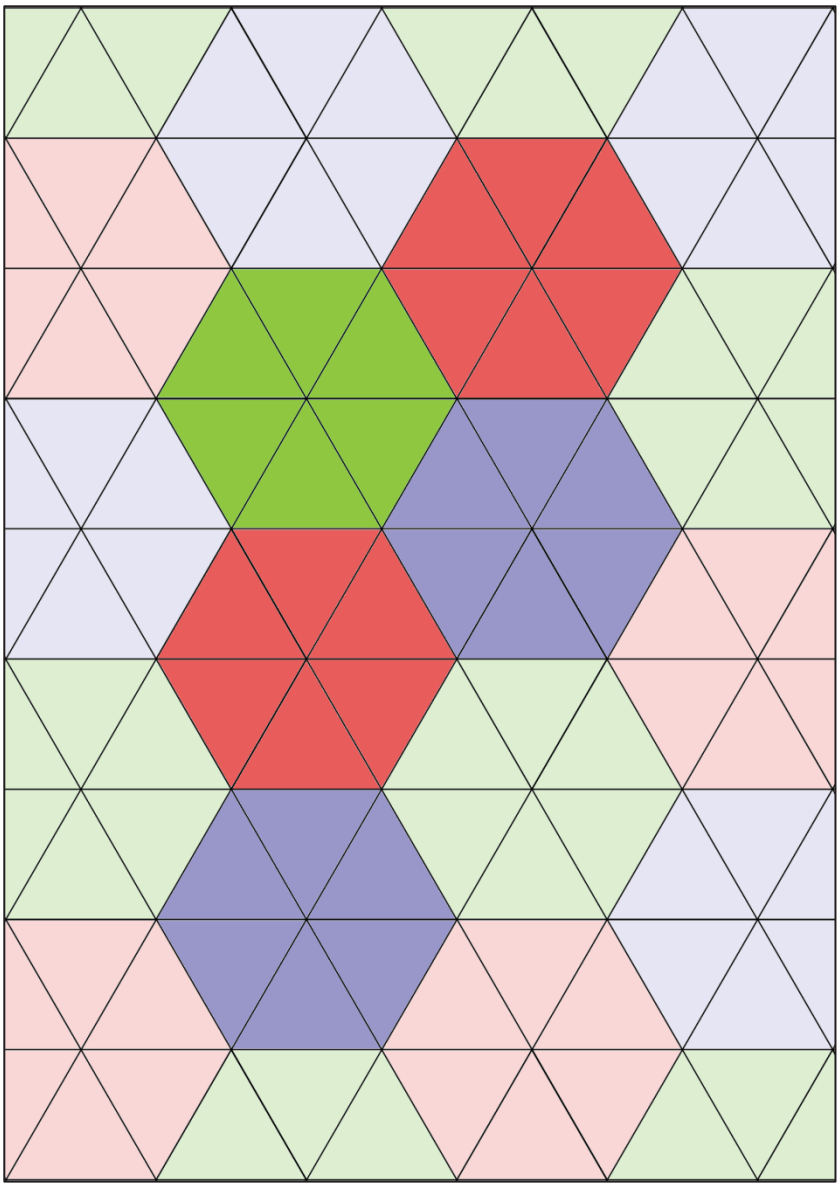

Figure 4.7: The bottom red cell is adjacent to both the blue cells and the green cell but is not adjacent to the top red cell. The bottom red cell also contains no other adjacency information regarding the other cells nearby even if the blue cell directly above to the left is mapped for example. Colours are for visual distinction of the cells.

While the robot moves throughout the environment the robot will near the current cells centre which contains the firing field. The distance to the centre is compared to the firing field threshold to determine when the robot has entered the triggering region which calls the image call-back to store an image node.

In addition to calculating the grid cells and firing fields the odometry call-back stores the poses at regular intervals to create a topological representation of the path travelled. This topological sub map only adds further nodes if the current pose do not overlap existing paths by 0.15m. The value for this threshold was chosen through experimentation.

4.3.3 - Neck Angle Call-back Loop

The neck call-back extracts the neck angle data from the stream and sends it to the automation processing loop when conducting obstacle avoidance.

4.3.4 - Robot Process Loop

The robot navigation function was implemented in parallel to the Fusion SLAM system presented. This section is composed of a three state, state machine which is run in the same processing loop as the odometry call-back to reduce unnecessary complexity. The sonar, bumper, and odometry messages are processed in this block as well as the neck angle from the neck function, the distance value and position from the image call-back, and the three experiment topics. The bumper state is checked every iteration to determine whether an obstacle or object has been crashed into. If any of the bumpers show a collision the position of the crashed bumper position is used to turn away from the obstacle by a speed of 0.1rad/s times the distance to the outside bumpers.

The sonar and distance values are checked for any obstacles within range. The sonar checks out to a distance of 1.0m and the Kinect checks out to a distance of 0.6m. The function determines which position has the closest obstacle and turns away from the obstacle direction. The speed to turn away by is 0.3rad/s for obstacles out to 1.0m and 0.5rad/s for any obstacle within 0.4m.

The robot process loop changes states from learning the map, navigating, and finding the image node in a cell by processing the commands received from the experiment call-back loop. In the learning state the robot must be taught the goal position by manually

driving it through the environment while the robot maps using the Fusion SLAM presented. As the environment is being mapped, the user can issue the robot a command to go to a particular grid cell. Once this command has been issued the robot will calculate the shortest path to the goal using the grid cells in the map as way points.

The command to find an image node in the cell can also be given. The robot will determine whether the specified image node exists in the current cell or not and will display to the on-board screen that the cell cannot be found if it is not stored in the current cell. If the image node specified is in the current cell of the robot, then the robot will move to the pose position of the image node and will turn its neck to face the heading of the image node.

4.3.5 - Robot Navigation

To navigate from the current cell to the specified grid cell, the robot uses the stored grid map. The shortest path calculation is conducted by applying decrementing points to all adjacent cells starting from the goal cell. All of the stored cells in the grid map are initialized to contain (-1) which represents cells not heading towards the goal. The goal cell is assigned a point value equal to the total number of cells stored. The grid map is then iterated through multiple times assigning a decremented point to each cell with adjacency to the goal. The exit condition is when the current cell is assigned a point value because once the current cell contains a point value the path to the goal is known by the robot. This system created is based on the work conducted by [10] which is also a ratSLAM system.

The robot uses the cells as way points heading toward the goal by trying to increase the points of the current cell by checking all the adjacent cells for a higher point value as shown in Figure 4.8.

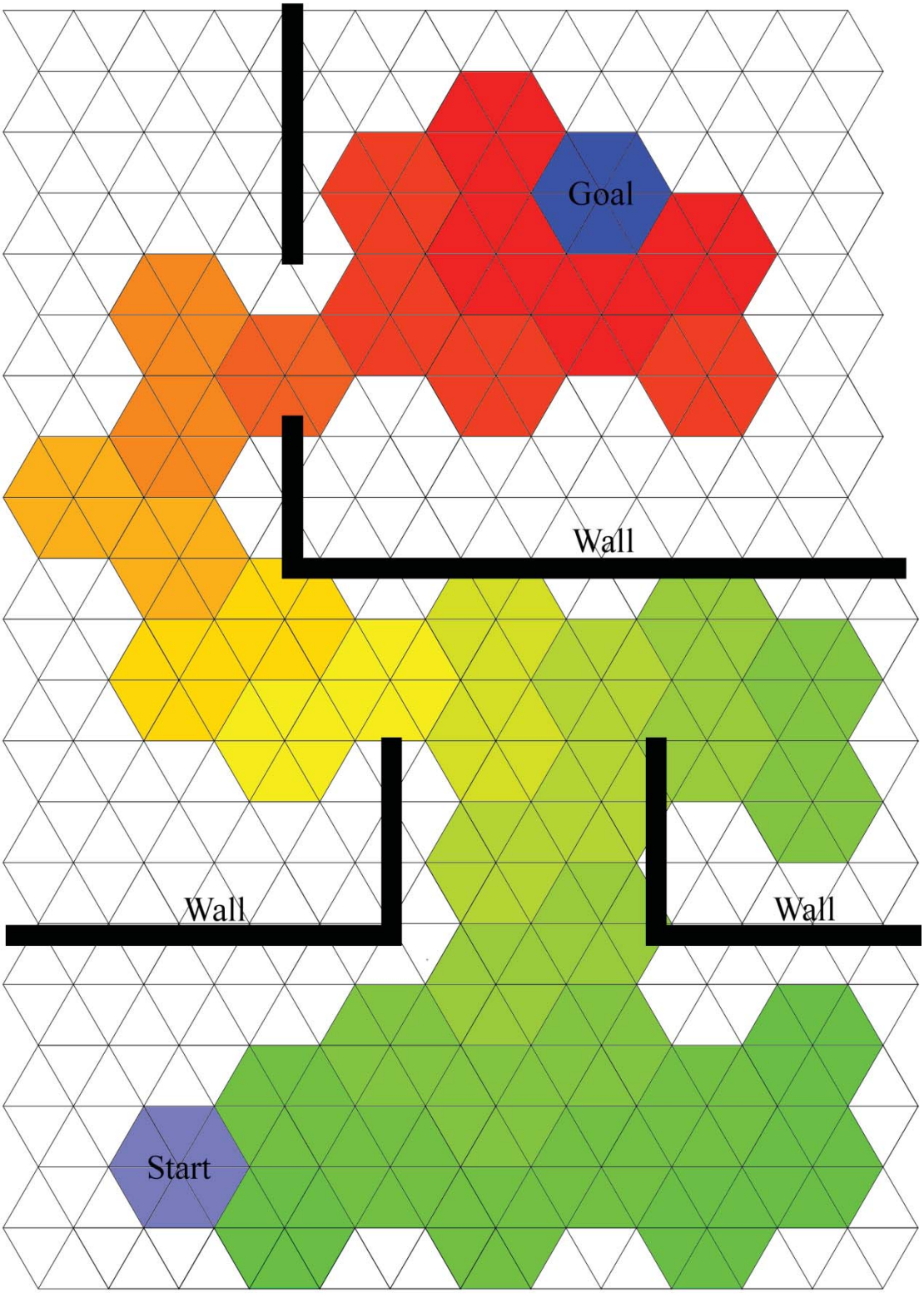

Figure 4.8: The robot navigation planner. The starting point is the light blue cell and the goal point is the dark blue cell. The robot will keep moving to the lighter cells until the goal cell is reached.

Chapter 5 - Experiment and Results

5.1 - Experimental Setup

5.1.1 - Testing the Robotic Fusion SLAM Algorithm

The completed Fusion SLAM system obtains RGB-D data and processes it internally to create a topological map which isolates image nodes to hex-based grid cells. It also creates pose nodes at given intervals and uses the connection information between the nodes to setup a point system with the goal having the highest value. The robot navigates through its environment to increase its current cell value to reach the goal.

To determine the effectiveness of the proposed Fusion SLAM algorithm and the effectiveness of the matching algorithm, the experiment was divided into two stages. In the first stage of the experiment the robot was driven through the environment from a fixed start position to a fixed goal position. The environment was setup so there was a small number of obstacles and a few bends as shown in Figure 5.1.

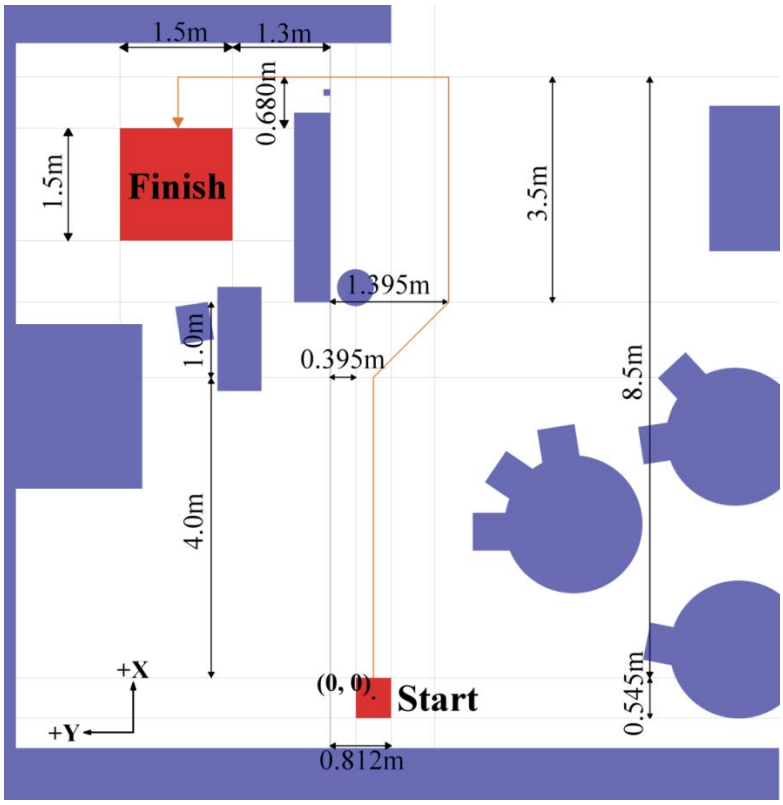

Figure 5.1: Layout of the experiment environment. The red boxes are the target positions and the blue objects are obstacles in the environment. The orange line is the path of the robot when creating the initial map.

The first stage of the test involves teaching the robot the path to the goal by driving it through the environment. The second stage involves driving the robot back to the start position and allowing it to navigate to the goal position by itself. Table 5.1 shows the full steps for testing the robot in the two stages. To maintain consistency during the experiment the robot was driven to follow the orange path passing through each position on each iteration of the test for both heading towards and heading away from the goal position to the start position before the navigation stage.

Table 5.1: Steps taken to test the robot in two stages.

Step	**Task**
1	- The robot is turned on at the start position - The robot creates the starting node - The robot starts in State 1
2	- The robot is driven to position 1 - The robot is turned to face position 2
3	- The robot is driven toward position 2 - The robot is turned to face position 3
4	- The robot is driven toward position 3 - The robot is turned to face position 4
5	- The robot is driven toward position 4 - The robot is turned to face the goal area
6	- The robot is driven into the goal area - The robot is stopped in a firing field to acquire an image node - Record the robot's actual pose
7	- The current grid cell is recorded with the current image node - The robot is driven back to the start position in reverse order
8	- The robot is issued the command to navigate to the cell id in the goal area - Once the robot reaches the goal cell, issue the command to find the image node in the goal area
9	- Once the robot reaches the target, record the robot's actual pose - Record the robot's internal state data
10	- Repeat the steps 1 to 9 for SIFT, SURF and ORB algorithms

5.1.2 – Testing the Image Search Algorithm

On every iteration of the Fusion SLAM and navigation test the robot was also tested to find an image node stored in the goal cell. After the robot reaches the goal cell, it was given the command to find a particular node in that cell. Each image node was stored in the cell which was created during SLAM step while being driven to the goal position. The target image node was chosen every time the robot was driven to the goal position on each iteration.

5.1.3 – Testing the Image Node Transform Algorithm

In addition to testing the robots SLAM algorithm through the use of a simple navigation algorithm the robot's transform estimation algorithm was tested for the amount of error that was present when the robot remained stationary.

To test the image node transform estimation, the robot was started as though it was a normal test run, but instead of moving the robot in an environment the robot was issued the command to just turn the neck to face different views in the environment. The first view was facing a desk that was moderately cluttered, the second view was facing a cabinet that was cluttered and the last view was facing an uncluttered blank white wall. These images are shown in the Figures 5.3, 5.4 and 5.5. Analysis of the results is appended in Figure 5.20, 5.21, 5.22.

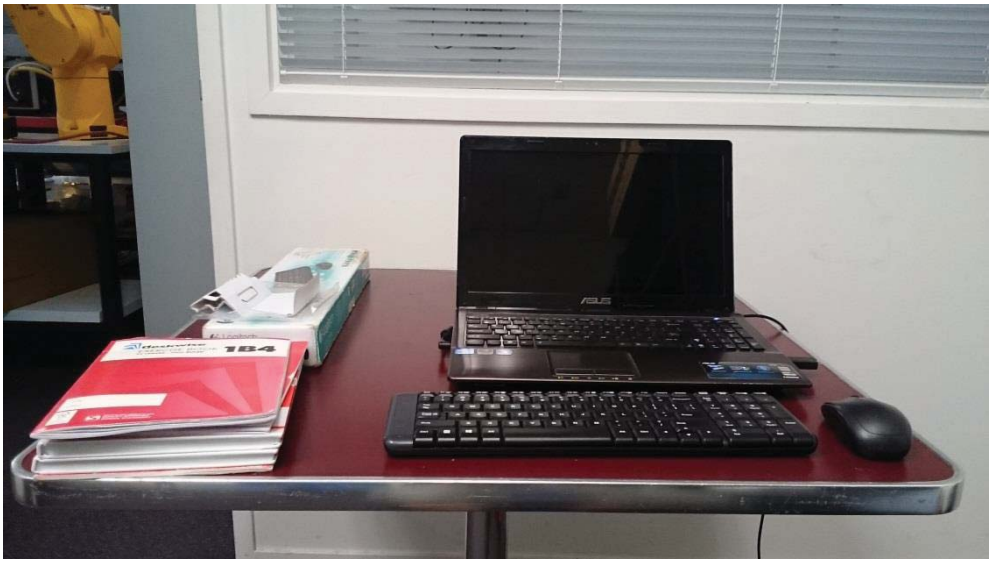

Figure 5.2: The moderately cluttered desk view.

Figure 5.3: Cluttered cabinet view.

Figure 5.4: Uncluttered blank wall.

5.2 - Experimental Results

The experimental results are shown in Figures 5.5 to Figure 5.24 in this chapter.

5.2.1 – Results of the Robotic Fusion SLAM Algorithm

For all iterations of the test, the robot successfully navigated to the goal position showing that the robot also successfully learnt a map of the environment to the goal position. The robot map was reset for each iteration of the test so the robot could recreate the map from the start position to the goal position. The figures below show the X, Y scatter plots of the robot's internal odometry state and the actual position in the environment when it reached the goal. The tables below show the final position of the robot and the accumulated error. During the experiment it was found that the system often zigzagged while travelling towards the finish.

The coordinates of the end position and the error accumulated is shown in the tables below. For all of the data see Appendix A.

Table 5.2: The average distance covered, error accumulated and the standard deviation for internal vs. measured odometry results.

	SIFT			SURF			ORB		
	dist.	err.	stddev	dist.	err.	stddev	dist.	err.	stddev
x	7.298	0.474	0.212	7.313	0.448	0.321	7.658	0.554	0.318
y	2.387	0.625	0.221	2.328	0.625	0.257	2.136	0.984	0.664

Table 5.3: The average distance covered, error accumulated and the standard deviation for odometry results before and after navigation.

	SIFT			SURF			ORB		
	dist.	err.	stddev	dist.	err.	stddev	dist.	err.	stddev
x	7.289	0.492	0.193	7.245	0.585	0.230	7.574	0.722	0.277
y	2.423	0.697	0.513	2.295	0.559	0.218	2.118	0.947	0.566

Table 5.4: The total average distance covered, error accumulated and the standard deviation for odometry results.

	Total Average.		
	distance	error	stddev
x	7.396	0.546	0.259
y	2.281	0.740	0.407

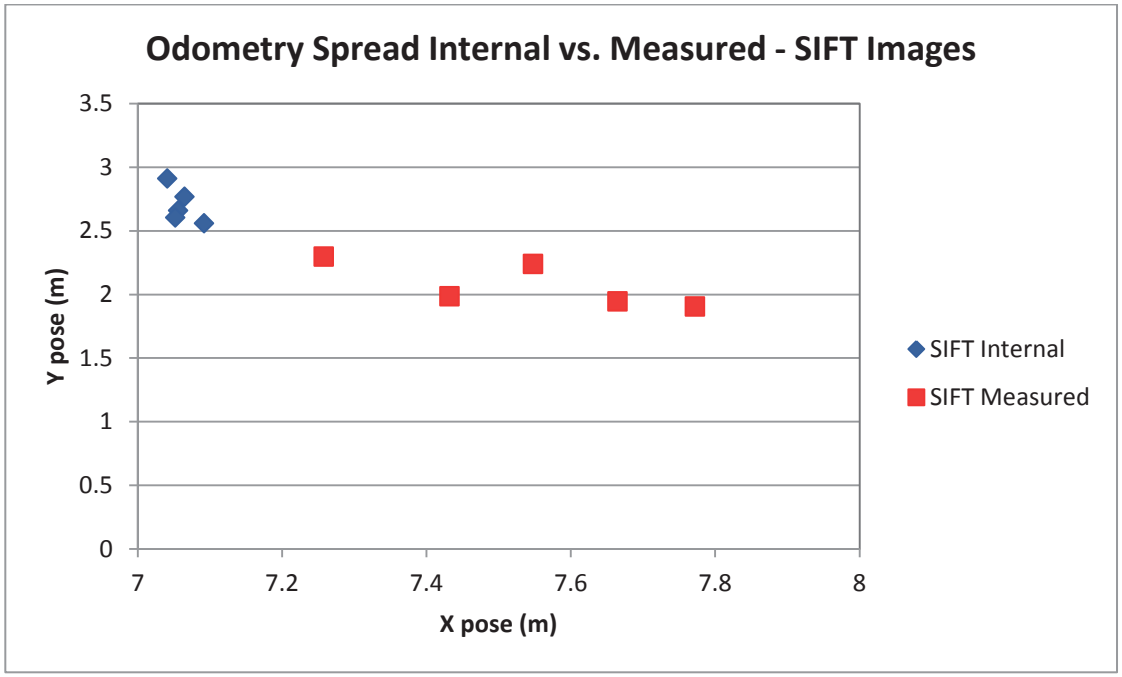

Figure 5.5: The robot's internal odometry vs. actual odometry in the environment using the SIFT algorithm for image nodes.

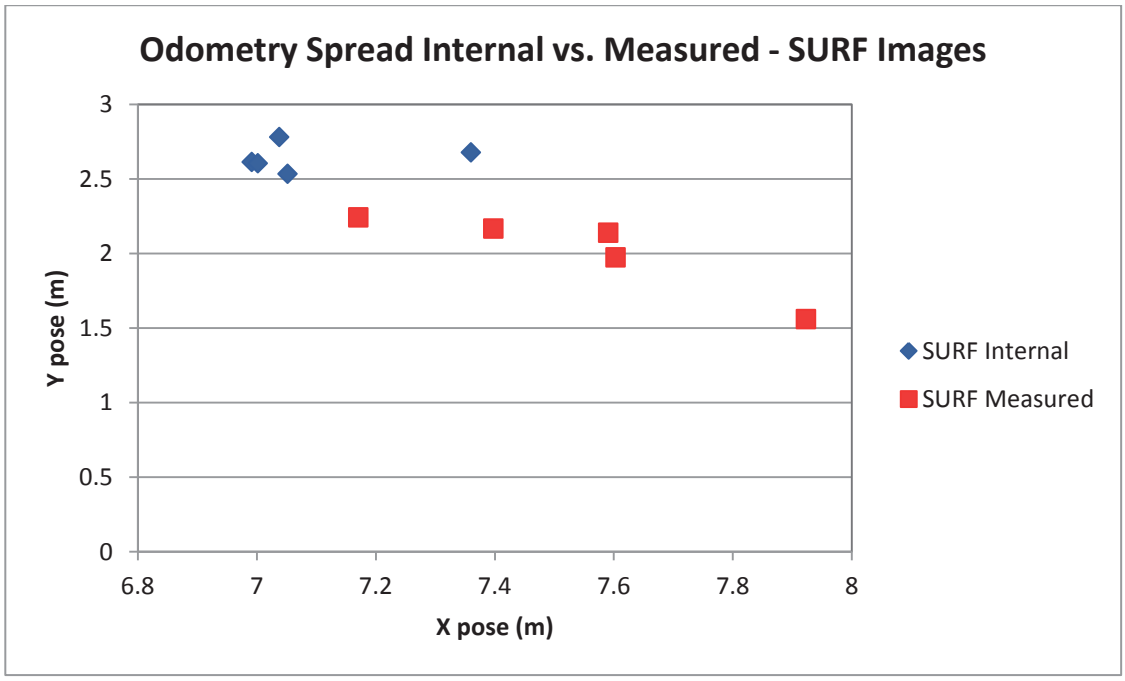

Figure 5.6: The robot's internal odometry vs. actual odometry in the environment using the SURF algorithm for image nodes.

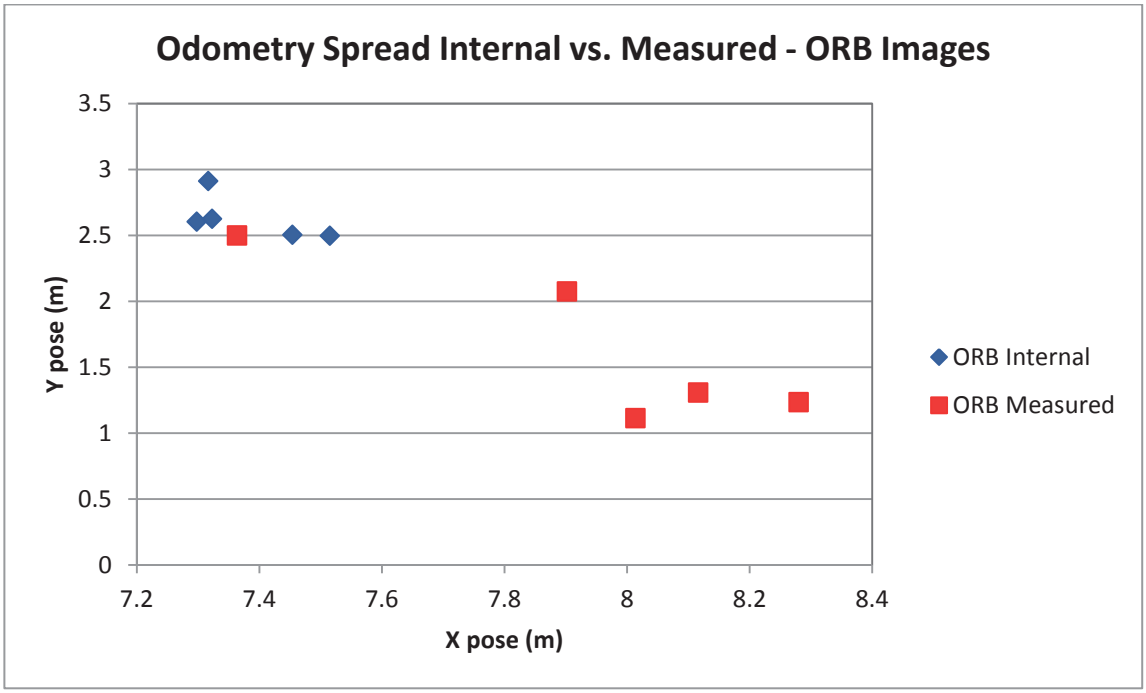

Figure 5.7: The robot's internal odometry vs. actual odometry in the environment using the ORB algorithm for image nodes.

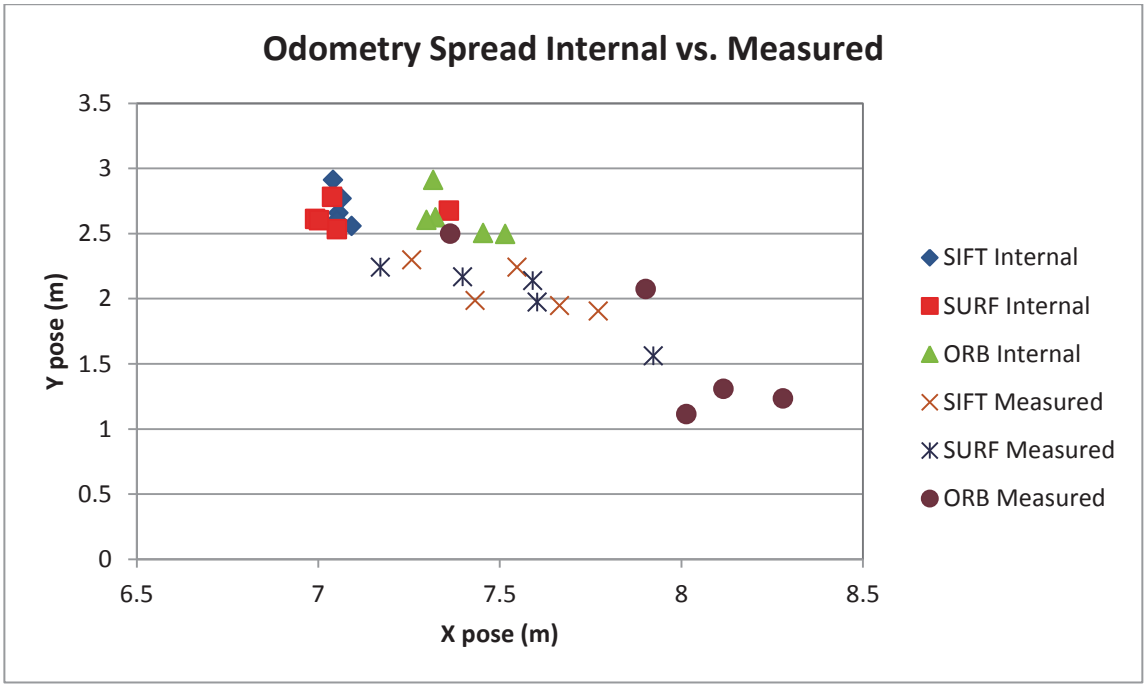

Figure 5.8: The robot's internal odometry vs. actual odometry in the environment with all of the algorithms data combined.

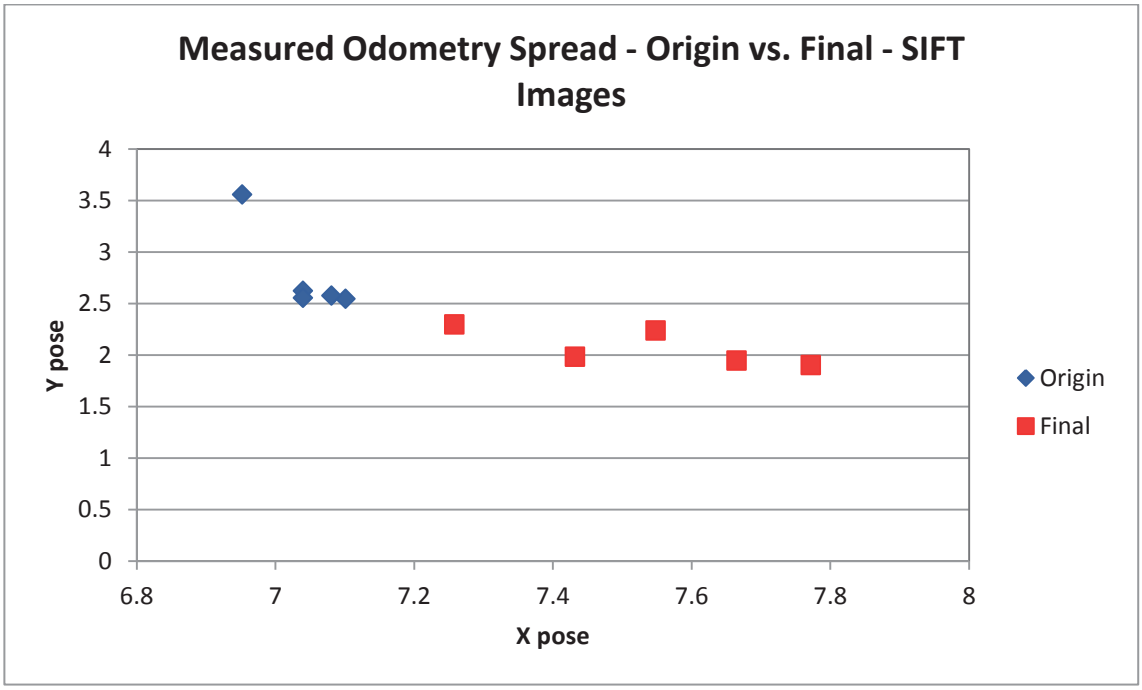

Figure 5.9: The robot's actual odometry at the goal before navigating vs. actual odometry after navigating the environment using the SIFT algorithm for image nodes.

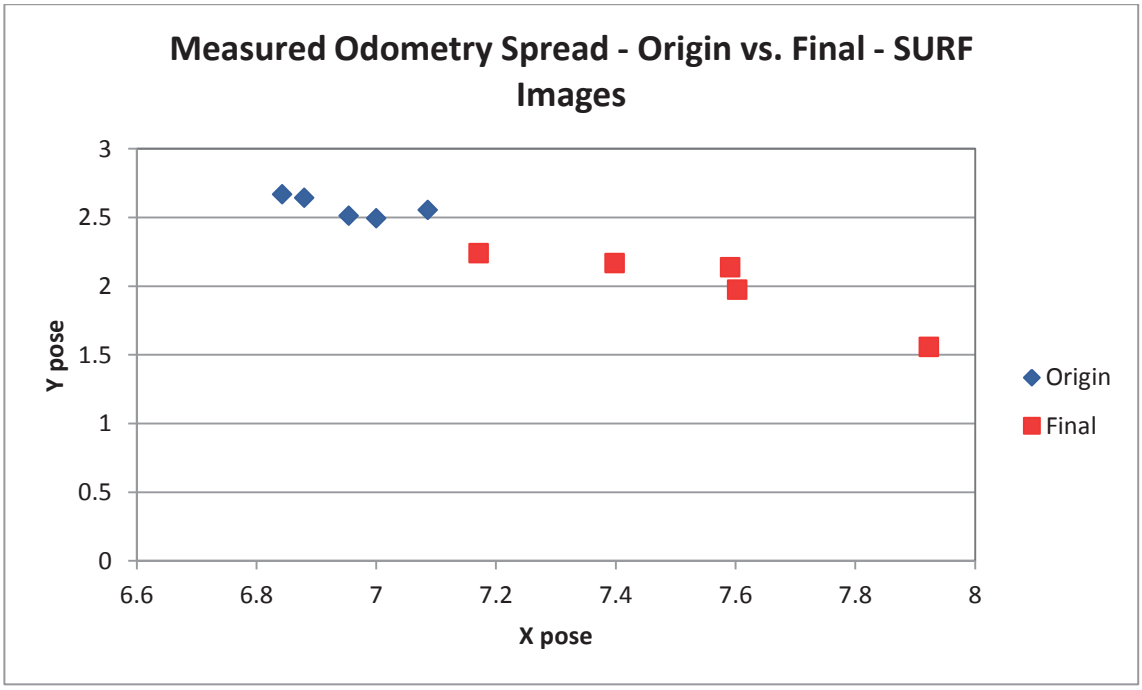

Figure 5.10: The robot's actual odometry at the goal before navigating vs. actual odometry after navigating the environment using the SURF algorithm for image nodes.

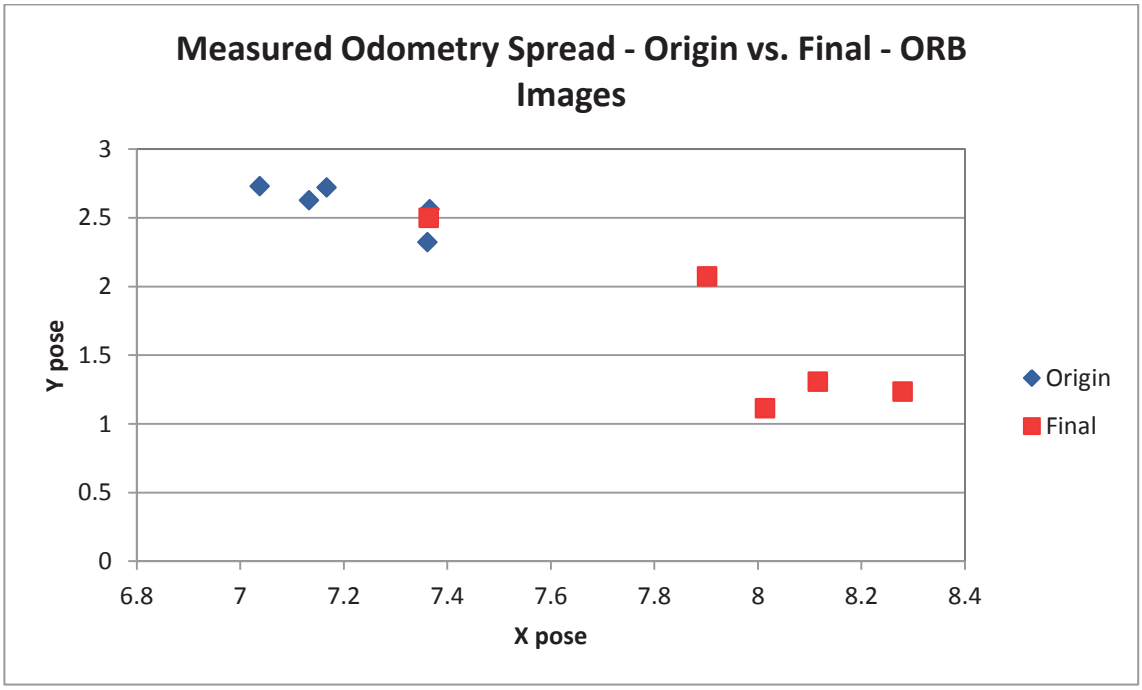

Figure 5.11: The robot's actual odometry at the goal before navigating vs. actual odometry after navigating the environment using the ORB algorithm for image nodes.

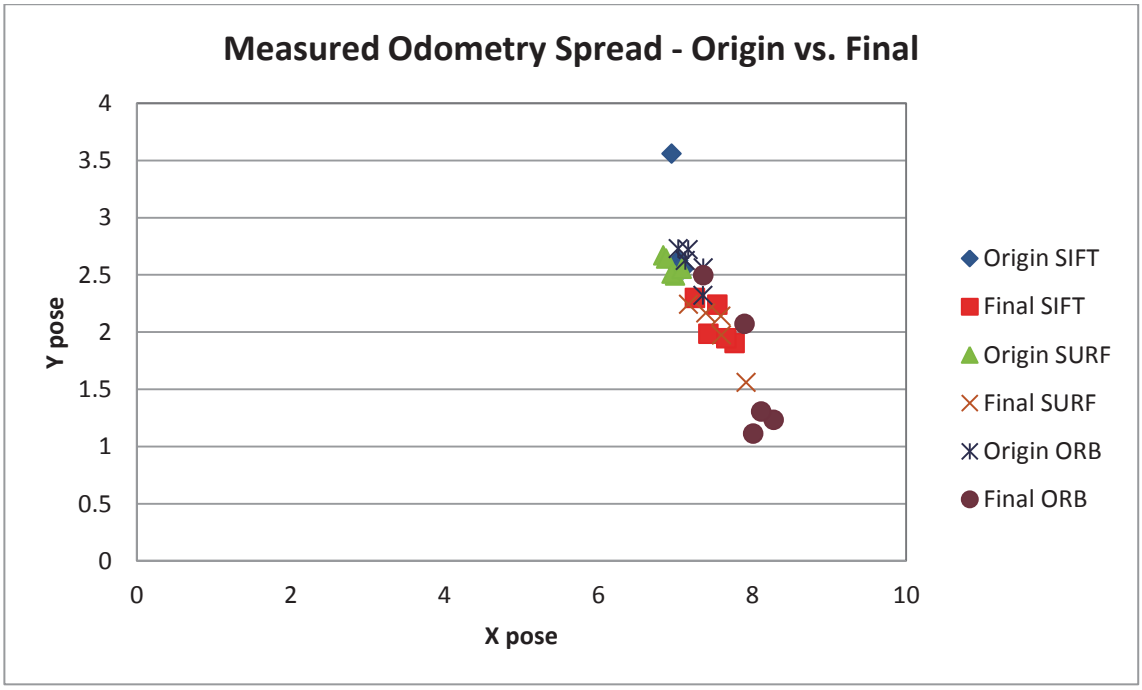

Figure 5.12: The robot's actual odometry at the goal before navigating vs. actual odometry after navigating the environment with the entire algorithm data combined.

By comparing the total grid cells created to the goal cell ids it can be seen that the SLAM system creates roughly double the number of grid cells compared to the goal cell id. This shows that because the map created is in the pattern of a hex map the robot can only traverse through one half of the possible cells on a path on the way towards a point and the remaining possible cells are added on the return path. The average of the number of total cells created is 89.867 cells with a standard deviation of 5.901 cells and the average cell *id* at the goal cell is 55.733 with a standard deviation of 4.559.

The comparison between the total image nodes created to the number of cells in the goal cell it can be seen that the mapping side of the system is only adding new image node cells at any given firing field. The average image node created in the goal cell is 5.267 nodes with an average total cells created of 128.2. The image nodes stored in the goal cell takes an average of 4% of the total image nodes created.

The plot showing the total number of cells created is a good measure on the number of cells that are created over a given distance. From the Figure 5.17 and Figure 5.18 it can be seen that over an average distance of 13.79m the average number of nodes created is 218 +/- 20 nodes. The data used in the figures below are presented in Appendix B.

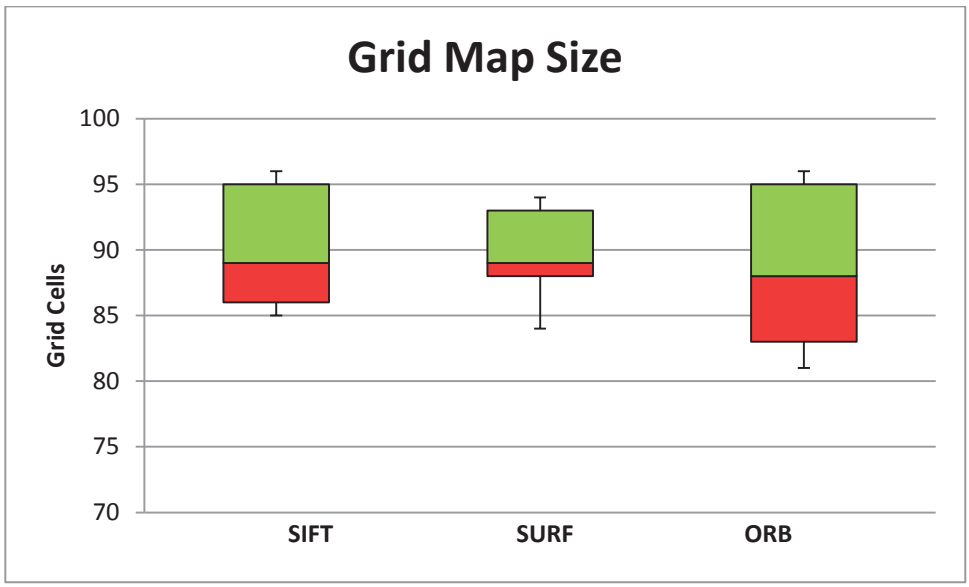

Figure 5.13: The total number of grid cells created.

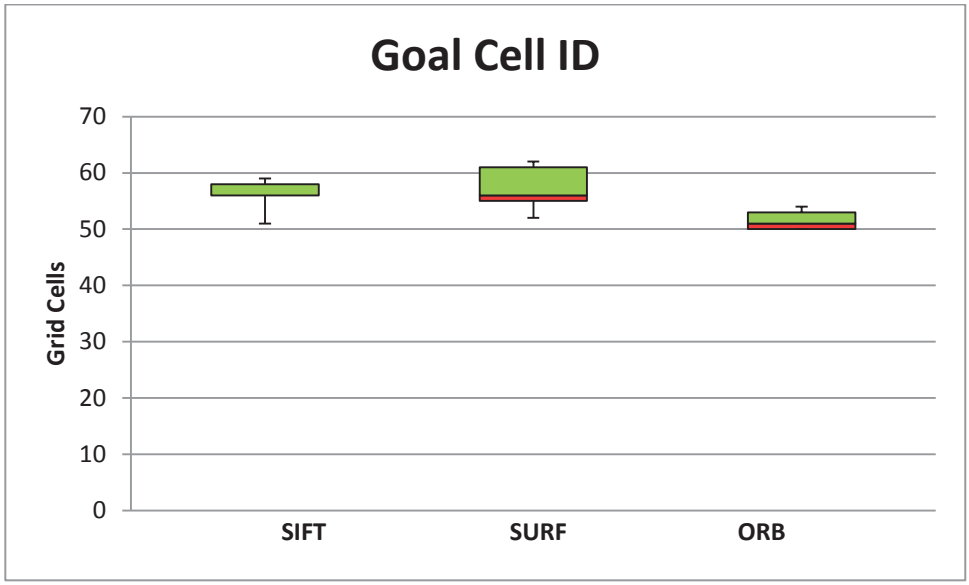

Figure 5.14: The average position of the goal cell.

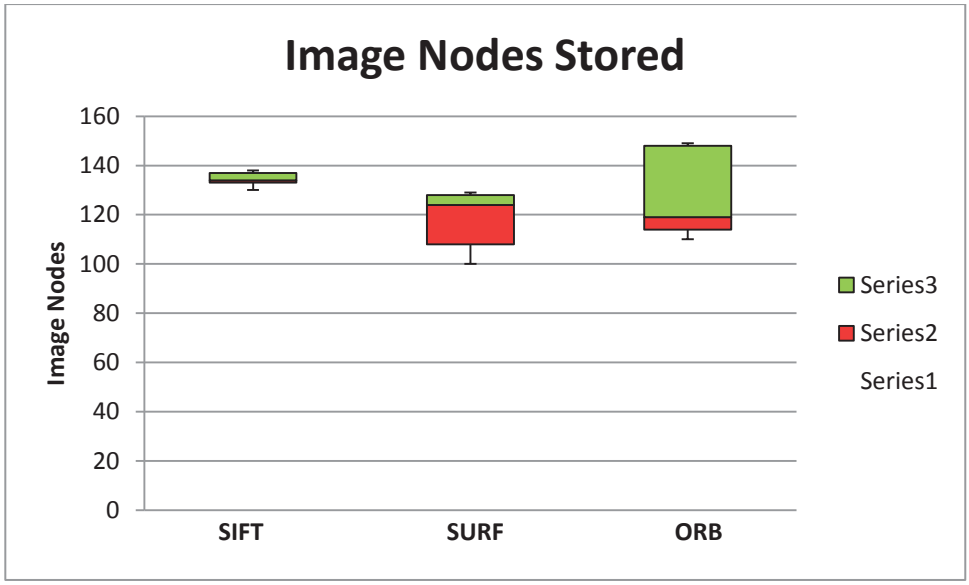

Figure 5.15: The total number of image cells created.

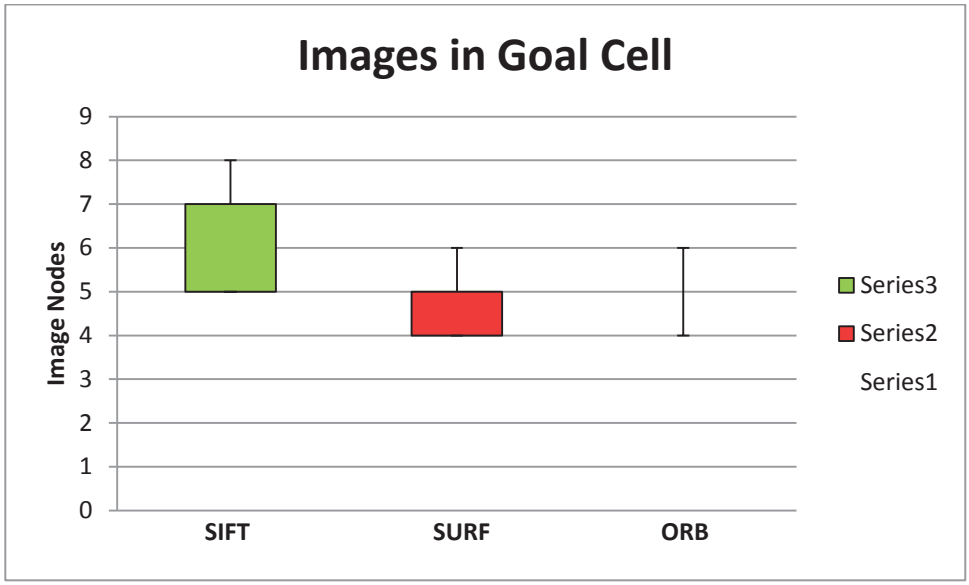

Figure 5.16: The portion of image nodes that are part of the goal cell.

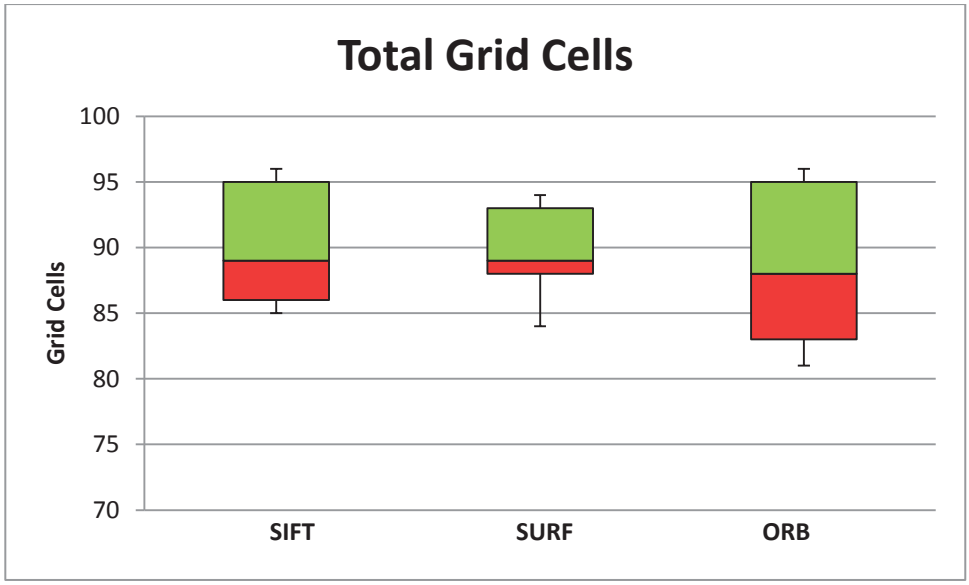

Figure 5.17: The total number of grid cells created in the map.

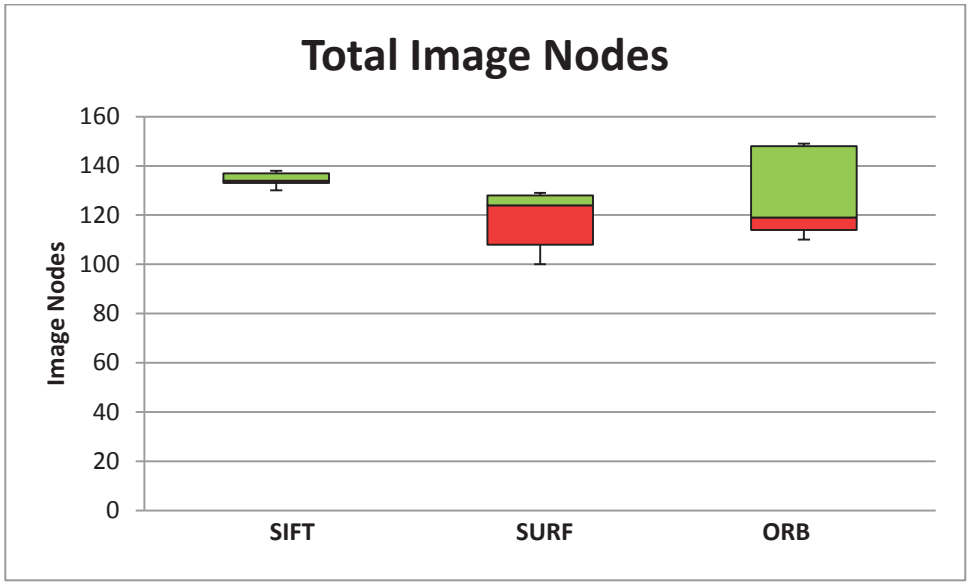

Figure 5.18: The total number of image nodes created in the map.

5.2.2 – Results of the Image Search Algorithm

From the experiment conducted to find the node in the current cell it can be seen by observing the plot below that the robot added a new image node and tried to match the image specified to the newly created node instead. The success rate of actively searching for a stored node using the odometry and trying to match the pose had a rate of 40% over 5 separate instances while using the SIFT algorithm. The success rate for SURF was even worse than SIFT being 27% and ORB had the highest success rate of 48%.

By further observation of the results it can be seen that for each of the successful poses acquired for the image node specified, the transforms that is obtained from the match is closely aligned to the odometry error which shows that for a successful active image node search the robot can reasonably estimate the total error accumulated. The successful match while using the SIFT algorithm produced an average estimated transform of 0.936m which when compared to the odometry when the data was taken shows that the odometry had an error drift of 0.671m. The successful match from the ORB algorithm has a much more accurate estimated transform of 0.586m where compared to the odometry when the data was taken shows an error drift of 0.488m. The outliers that can be seen in the figures are the points when the robot stopped before the

goal image was fully within view. The robot completed its search when the image was partially visible, which caused the robot to believe that the final match was found resulting in the transform estimate being unable to calculate the value properly. The data used in the figures below are presented in Appendix C.

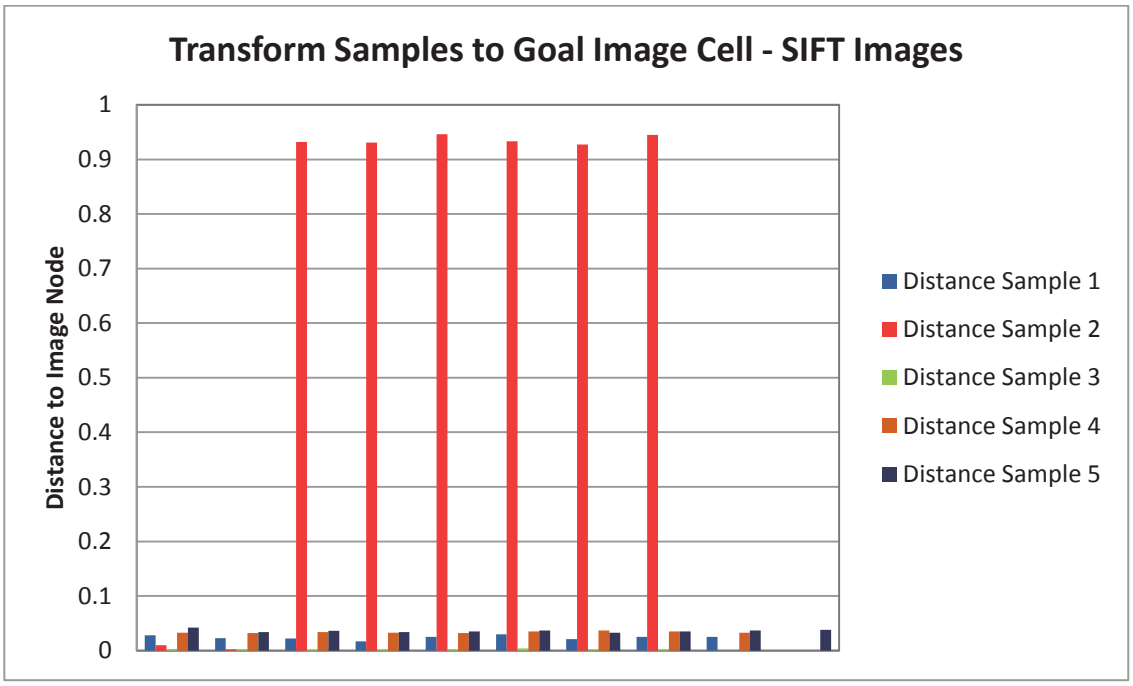

Figure 5.19: Transform estimates obtained by actively searching for an image node specified while using the SIFT algorithm.

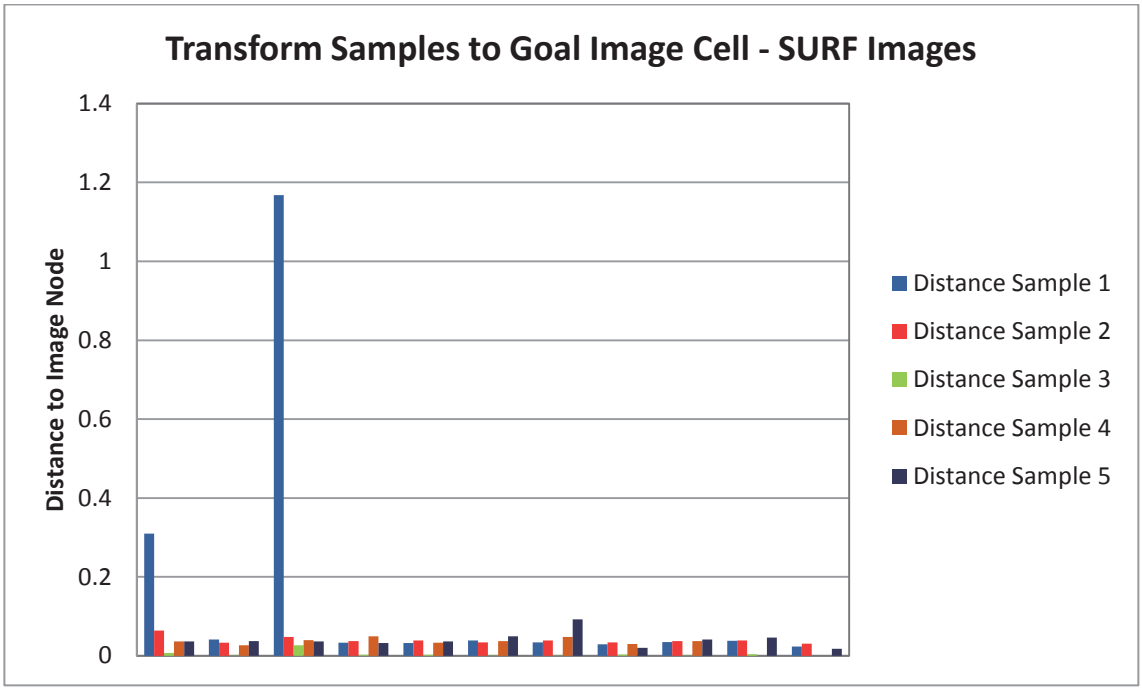

Figure 5.20: Transform estimates obtained by actively searching for an image node specified while using the SURF algorithm.

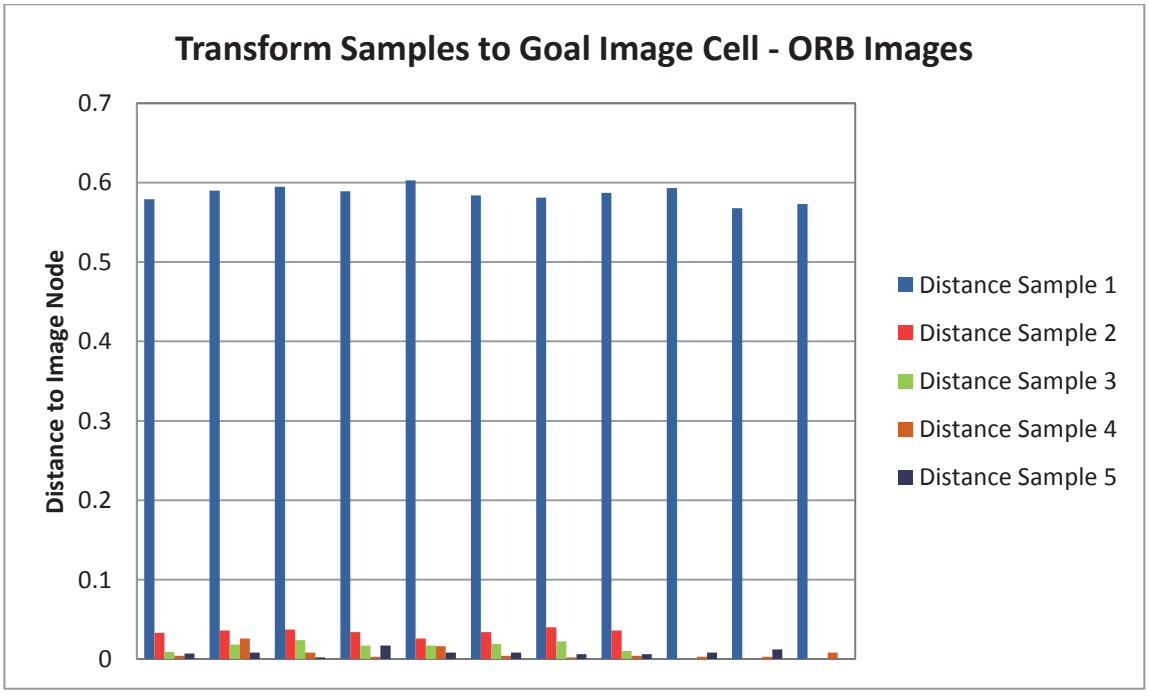

Figure 5.21: Transform estimates obtained by actively searching for an image node specified while using the ORB algorithm.

5.2.3 – Results of the Image Node Transform Algorithm

The results that were obtained when testing the transform estimation from comparing stored image nodes showed fairly large estimation errors from landmarks which contained seemingly well-defined features.

The extremely large error values that can be seen on Figure 5.24 were the result for the blank wall test. The reason for why the error was so large can be explained by the method which is used to match image views. When the algorithm determines that there is a match between images, it compares the number of features that can be obtained from the view. In the case of the blank white wall, extremely few features are extracted and in most cases these features are randomly spaced therefore trying to find a match with little features inherently causes large errors in the transform estimates.

The data used in the table below is presented in Appendix D.

Table 5.5: The average transform estimates obtained by the different algorithms observing a stationary, desk, cabinet, and blank wall.

	SIFT		SURF		ORB	
	average	stddev	average	stddev	average	stddev
desk	0.029	0.077	0.091	0.130	0.159	0.373
cabinet	0.043	0.068	0.036	0.057	0.037	0.047
blank wall	0.380	0.571	0.782	1.014	0.797	0.571

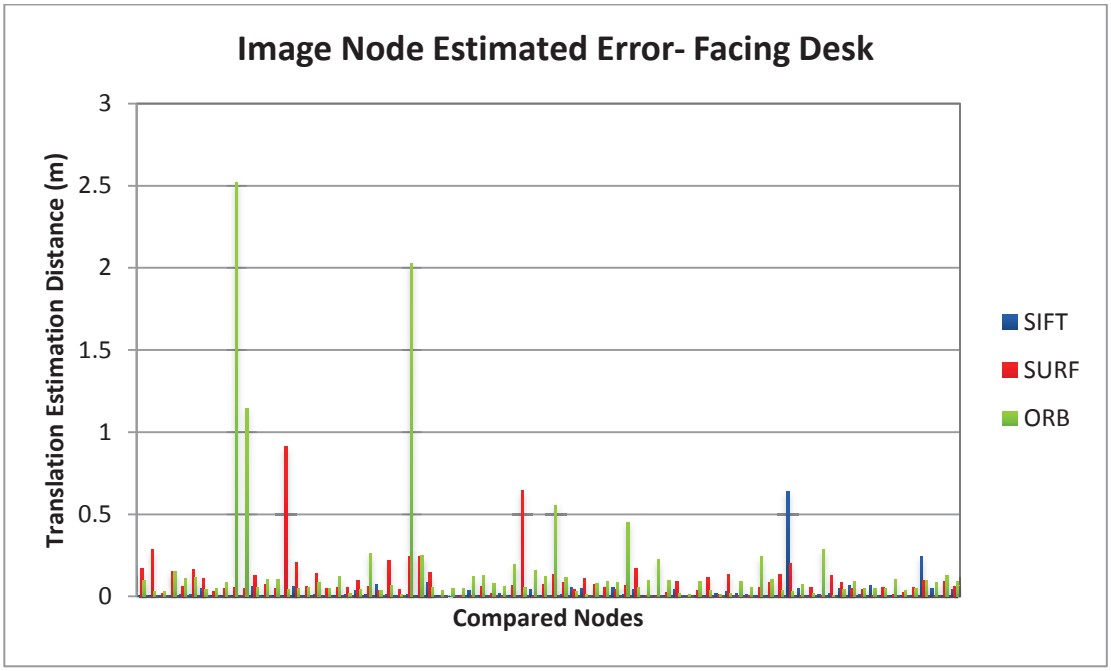

Figure 5.22: Distance transforms calculated by the stationary robot viewing the moderately cluttered desk.

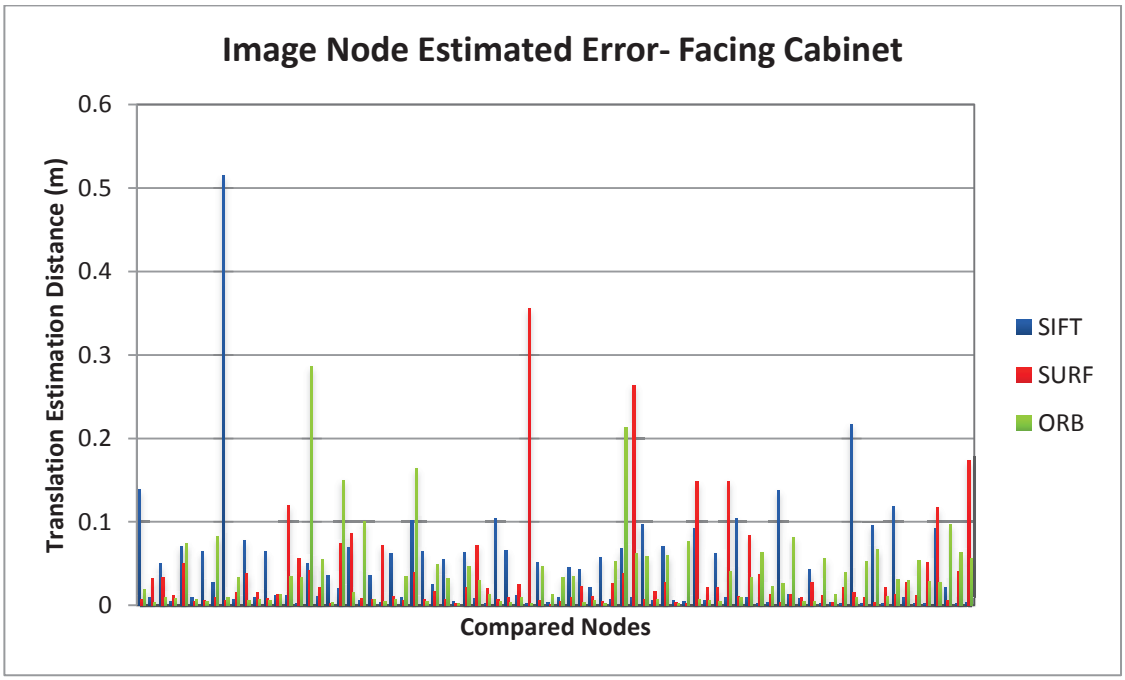

Figure 5.23: Distance transforms calculated by the stationary robot viewing the cluttered cabinet

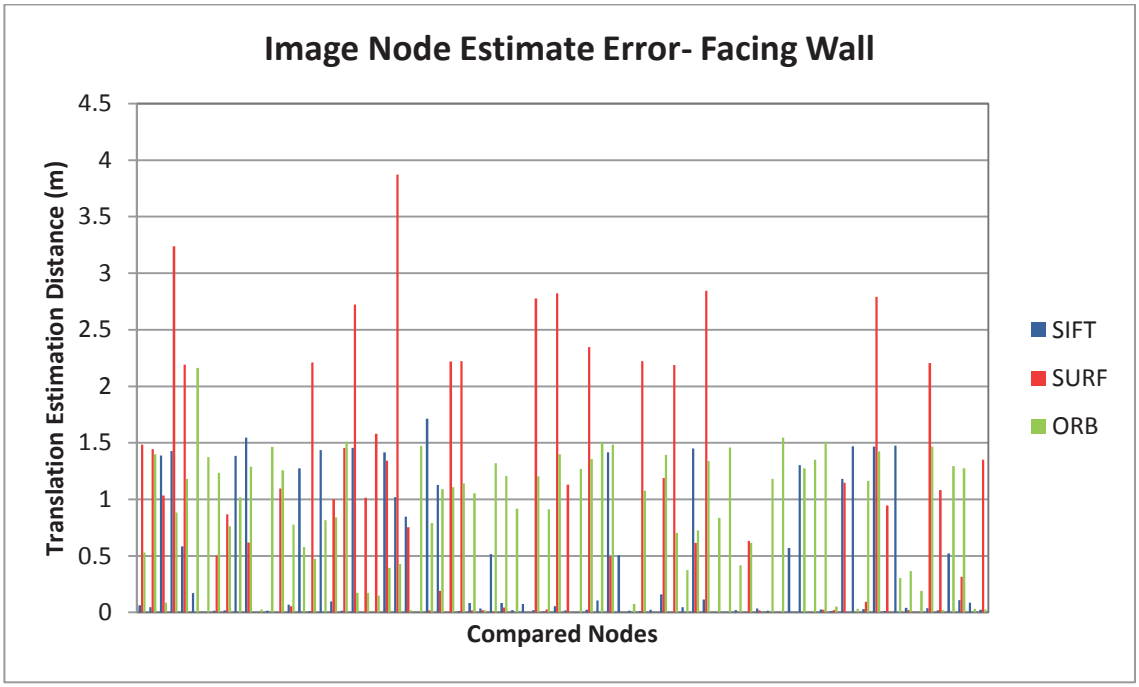

Figure 5.24: Distance transforms calculated by the stationary robot viewing the blank wall

Chapter 6 - Conclusion and Further Research

6.1 – Conclusion

From the results gathered in the previous chapter it can be stated that the Fusion SLAM algorithm created in this study was successful and solved the robotic SLAM and RGB-D SLAM problem. The robot was able to successfully map the environment using the RGB colour and depth data combined with the robots odometry to create the RGB-D SLAM and ratSLAM fusion map. Although the robot successfully conducted SLAM using the proposed method, the results from the Fusion SLAM algorithm using the different methods for conducting image matching shows that there is on average, a 7% accumulation of error in the x direction and 33% accumulation of error in the y direction of the global map. The larger error in the y direction can be caused by the error in the rotation measurement, error in the wheel dimension where one side is slightly larger, and from multiple turns which serves to amplify any errors in this particular frame of reference.

In addition to the difference in the error between the internal and measured odometry the comparison between the odometry before and after the navigation is also important because it serves to show the error that the robot gains while moving autonomously. The total odometry error that the robot gained in the x direction is on average, 8% of the distance travelled and 32% in the y direction. Comparing the difference in the internal vs. measured odometry error and the odometry error before and after navigation shows that the robot is constantly accumulating odometry error of about half the standard deviation in both the x and y directions. The effect that this will have on the SLAM system increases as the robot travels and is proportional to the total distance travelled. The effect will be that the perceived grid cell and firing field positions will appear to drift with the robot causing image matching within one particular node to fail while travelling over a large distance. This will be because the image node that should be present in the map at a particular point in the actual environment will not be where the robot remembers it to be because of the odometry drift. To reduce this accumulated odometry error the transformation information from the image matching nodes is required but this will also have significant effects on the robot’s ability to conduct SLAM accurately.

From the results obtained by the image node transform algorithm using SIFT, SURF, and ORB, it was found that the error in this transform calculation had the potential to reduce the robots odometry error when there were sufficient features. While there was sufficient features in the environment during the desk and cabinet stationary test, the error in the transform had an average of 0.036m drift with a standard deviation of 0.072m for SIFT, an average of 0.063m drift with a standard deviation of 0.094m for SURF and an average of 0.098m drift with a standard deviation of 0.210m for ORB which is very promising, but as soon as there are not enough features the transform estimation error grows significantly to have an average of 0.380m drift with a standard deviation of 0.571m for SIFT, an average of 0.782m drift with a standard deviation of 1.014m for SURF and an average of 0.797m drift with a standard deviation of 0.571m for ORB. When the robot viewed the environment such as a blank wall, the large transform estimation error skewed the robot's perceived position in the map which caused the robot to head straight toward the said wall on some occasions.

It was initially thought that the robot's error could be sufficiently corrected while this transform estimation correction was implemented but it was found that the three algorithms tested identified and matched image nodes with featureless views. This shows that although the view should contain no features to match, the algorithm still managed to find features, which are then forcibly matched with existing nodes which causes the large error being shown for the blank feature test.

Although the transform estimation calculation is accurate and effective in a feature-rich environment the transform correction applied to the robots accumulated error when viewing a featureless environment causes the SLAM algorithm to skew in unpredictable directions, which is problematic.

To further confirm the above transform estimation error, the test to search for an image node in the goal cell shows that the transform estimation can be prone to large errors. Although the actual success rate for this test was not very high, for a successful image node match the second iteration for the navigation test shows an average transform estimation of 0.936m where the actual odometry error for this iteration was 0.671m. The difference in the calculated transformation estimate and the actual odometry error was 0.265m which will cause further error drifts in the robot's odometry error.

While the robot was conducting the two-stage SLAM and navigation test it was noticed that the robot displayed an unusual zigzagging behaviour, which was not anticipated. The navigation algorithm was created to try and increase the total points in the occupied cell by comparing the adjacent neighbours and heading toward the one with a higher point. A possible explanation for this behaviour is that the robot does not fully leave the previous position in the map with a lower point when it heads towards another location with a higher value, which causes the system to begin evaluating the next pose before leaving the previous position. As it begins to head towards the new heading, the movement caused the robot to enter the previous cell again creating the effect of turning backward and forward as it edges closer to its actual heading. Another possibility is that the machine enters the region between two cells with the same value so it keeps recalculating to update its target every time it crosses the neighbour's boundary. The robot perceives that it is entering a new cell each time because the border causes a ping-pong effect.

Although the robot's Fusion SLAM algorithm has a number of aspects with room for improvement the robot successfully solved the problem in the RGB-D SLAM system while also solving robotic SLAM. The robot has separated the images into image nodes in grid cells which do not require a continuous stream of images and the ambiguous images that cause incorrect loop closure and map distortion are removed by the grid cell and firing field map system from the ratSLAM-like algorithm having a special value by the cell sizes. The robot also creates an internal map and localizes itself within the map created therefore solving the robotic SLAM problem.

6.2 - Further Research

Improvements that could be made to the robotic Fusion SLAM algorithm are:

1. The scan matching algorithm is very effective when facing feature rich views but the algorithm is vulnerable to feature-sparse views. Improvements to filter or rectify feature-sparse views should improve the robustness for scan matching and for calculating the transformation estimate between different views.
2. The robot is vulnerable to odometry error drift so a filter or method to rectify this drift will improve the robot's ability to conduct more accurate SLAM.
3. The robot's navigation algorithm is cumbersome and had moments where it spent a lot of time zigzagging. This can be improved by conducting an improved heading calculation to minimize inefficiencies such as the zigzagging.
4. The success rate of the robot's image node search algorithm was not very high. Therefore the image node search function can be improved to better find and re-observe images from the environment and internal storage.

Further research possibilities for the robot are:

1. The 3D data stored in the robot's image nodes can be re-used to conduct object recognition on objects in the environment.
2. A gripper or manipulation actuator can be integrated into the robot to allow the robot to use the RGB colour and 3D depth data to interact with objects in the environment.
3. At the moment the robot does not display the data in a visually-appealing manner, therefore a more interactive GUI could be implemented to allow for smoother data acquisition and interaction with the robot, its algorithms, and its functions.
4. A person recognition algorithm can be implemented to distinguish people from obstacles and static landmarks to improve the map building algorithm.

References

[1] J. Weingarten and R. Siegwart, "EKF-based 3D SLAM for structured environment reconstruction," in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005, pp. 3834-3839.

[2] J. Z. Sasiadek, A. Monjazeb, and D. Necsulescu, "Navigation of an autonomous mobile robot using EKF-SLAM and FastSLAM," in *Control and Automation, 2008 16th Mediterranean Conference on*, 2008, pp. 517-522.

[3] A. A. Ravankar, Y. Kobayashi, and T. Emaru, "Clustering Based Loop Closure Technique for 2D Robot Mapping Based on EKF-SLAM," in *2013 7th Asia Modelling Symposium*, 2013, pp. 72-77.

[4] S. R. u. N. Jafri and R. Chellali, "A distributed multi robot SLAM system for environment learning," in *Robotic Intelligence In Informationally Structured Space (RiiSS), 2013 IEEE Workshop on*, 2013, pp. 82-88.

[5] K. Lee, S. h. Ryu, J. Kang, H. Choi, C. Jun, and N. L. Don, "Analysis of the reference coordinate system used in the EKF-based SLAM," in *Ubiquitous Robots and Ambient Intelligence (URAI), 2014 11th International Conference on*, 2014, pp. 33-38.

[6] J. G. Kang, W. S. Choi, S. Y. An, and S. Y. Oh, "Augmented EKF based SLAM method for improving the accuracy of the feature map," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 2010, pp. 3725-3731.

[7] M. Liu and H. Leung, "EM-EKF based visual SLAM for simple robot localization," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2014, pp. 3121-3125.

[8] W. S. Choi and S. Y. Oh, "Robust EKF-SLAM method against disturbance using the Shifted Mean based Covariance Inflation Technique," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 4054-4059.

[9] K. P. B. Chandra, G. Da-Wei, and I. Postlethwaite, "SLAM using EKF, EH and mixed EH2/H filter," in *2010 IEEE International Symposium on Intelligent Control*, 2010, pp. 818-823.

[10] M. Milford, G. Wyeth, and D. Prasser, "RatSLAM on the Edge: Revealing a Coherent Representation from an Overloaded Rat Brain," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006, pp. 4060-4065.

[11] R. Berkvens, M. Weyn, and H. Peremans, "Asynchronous, electromagnetic sensor fusion in RatSLAM," in *SENSORS, 2015 IEEE*, 2015, pp. 1-4.

[12] M. J. Milford, F. Schill, P. Corke, R. Mahony, and G. Wyeth, "Aerial SLAM with a single camera using visual expectation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 2506-2512.

[13] M. J. Milford and G. F. Wyeth, "Single camera vision-only SLAM on a suburban road network," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 2008, pp. 3684-3689.

[14] L. D. Alfonso, A. Grano, P. Muraca, and P. Pugliese, "A cells covering based method for Simultaneous Localization and Mapping in an unknown indoor environment," in *Control Conference (ECC), 2015 European*, 2015, pp. 1986-1991.

[15] G. Tejera, M. Llofriu, A. Barrera, and A. Weitzenfeld, "A spatial cognition model integrating grid cells and place cells," in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pp. 1-8.

[16] M. J. Milford and A. Jacobson, "Brain-inspired sensor fusion for navigating robots," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013, pp. 2906-2913.

[17] S. Saeidi and F. Towhidkhah, "Different spatial scales in mapping from grid cells to place cells: A neural network model," in *2009 4th International IEEE/EMBS Conference on Neural Engineering*, 2009, pp. 706-709.

[18] Z. Huhn, Z. Somogyvari, T. Kiss, and P. Erdi, "Extraction of distance information from the activity of entorhinal grid cells: a model study," in *2009 International Joint Conference on Neural Networks*, 2009, pp. 1298-1303.

[19] G. Tejera, A. Barrera, M. Llofriu, and A. Weitzenfeld, "Solving uncertainty during robot navigation by integrating grid cell and place cell firing based on rat spatial cognition studies," in *Advanced Robotics (ICAR), 2013 16th International Conference on*, 2013, pp. 1-6.

[20] C. Choi, A. J. B. Trevor, and H. I. Christensen, "RGB-D edge detection and edge-based registration," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1568-1575.

[21] M. Li, R. Lin, H. Wang, and H. Xu, "An efficient SLAM system only using RGBD sensors," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013, pp. 1653-1658.

[22] P. Be, x, F. Ducho, x, T. M, x00F, *et al.*, "3D map reconstruction with sensor kinect: Searching for solution applicable to small mobile robots," in *Robotics in Alpe-Adria-Danube Region (RAAD), 2014 23rd International Conference on*, 2014, pp. 1-6.

[23] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3-D Mapping With an RGB-D Camera," *IEEE Transactions on Robotics*, vol. 30, pp. 177-187, 2014.

[24] C. H. Liu and K. T. Song, "A new approach to map joining for depth-augmented visual SLAM," in *Control Conference (ASCC), 2013 9th Asian*, 2013, pp. 1-6.

[25] E. Ataer-Cansizoglu, Y. Taguchi, S. Ramalingam, and T. Garaas, "Tracking an RGB-D Camera Using Points and Planes," in *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, 2013, pp. 51-58.

[26] T. Whelan, M. Kaess, J. J. Leonard, and J. McDonald, "Deformation-based loop closure for large scale dense RGB-D SLAM," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 548-555.

[27] T. k. Lee, S. Lim, S. Lee, S. An, and S. y. Oh, "Indoor mapping using planes extracted from noisy RGB-D sensors," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 1727-1733.

[28] Q. Hieu Pham and Q. Ngoc Ly, "Some improvements in the RGB-D SLAM system," in *Computing & Communication Technologies - Research, Innovation, and Vision for the Future (RIVF), 2015 IEEE RIVF International Conference on*, 2015, pp. 112-116.

[29] Y. Wang and S. Huang, "Towards dense moving object segmentation based robust dense RGB-D SLAM in dynamic scenarios," in *Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on*, 2014, pp. 1841-1846.

[30] L. Moreno, M. L. Munoz, S. Garrido, and F. Martin, "Evolutionary Filter for Mobile Robot Global Localization," in *Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on*, 2007, pp. 1-6.

[31] S. Thrun, W. Burgard, and D. Fox, "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, 2000, pp. 321-328 vol.1.

[32] D. Marzorati, M. Matteucci, and D. G. Sorrenti, "Particle-based Sensor Modeling for 3D-Vision SLAM," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 4801-4806.

[33] A. Shpunt and Z. Zalevsky, "Depth-varying light fields for three dimensional sensing," ed: Google Patents, 2008.

[34] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91-110, 2004.

[35] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, vol. 110, pp. 346-359, 6// 2008.

[36] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a Local Binary Descriptor Very Fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, pp. 1281-1298, 2012.

[37] E. Rosten, R. Porter, and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, pp. 105-119, 2010.

[38] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, 2011, pp. 2564-2571.

[39] Whitbrook, A. (2010). *Programming Mobile Robots with Aria and Player*. New York, United States: Springer-Verlag London Limited 2010.

[40] Microsoft. (2016). *Kinect for Windows Sensor Components and Specifications.* Retrieved from https://msdn.microsoft.com/en-us/library/jj131033.aspx

Appendix A

Raw data of internal odometry and measured odometry of the robot after the goal position has been reached using different scan matching algorithms.

Table A.1: Measured odometry at the goal position before and after navigation using the SIFT algorithm for the image nodes.

SIFT	before x	after x	diff x	before y	after y	diff y
1	6.952	7.432	0.480	3.558	1.985	1.573
2	7.101	7.772	0.671	2.546	1.904	0.642
3	7.081	7.258	0.177	2.576	2.298	0.278
4	7.040	7.548	0.508	2.625	2.240	0.385
5	7.040	7.665	0.625	2.555	1.946	0.609
Average	7.043	7.535	0.492	2.772	2.075	0.697
StdDev	0.057	0.200	0.193	0.440	0.181	0.513

Table A.2: Internal and measured odometry at the goal position after navigation using the SIFT algorithm for the image nodes.

SIFT	internal x	measured x	diff x	internal y	measured y	diff y
1	7.056	7.432	0.376	2.658	1.985	0.673
2	7.065	7.772	0.707	2.768	1.904	0.864
3	7.092	7.258	0.166	2.558	2.298	0.260
4	7.041	7.548	0.507	2.911	2.240	0.671
5	7.052	7.665	0.613	2.603	1.946	0.657
Average	7.061	7.535	0.474	2.700	2.075	0.625
StdDev	0.019	0.200	0.212	0.142	0.181	0.221

Table A.3: Measured odometry at the goal position before and after navigation using the SURF algorithm for the image nodes.

SURF	before x	after x	diff x	before y	After y	diff y
1	7.000	7.923	0.923	2.494	1.559	0.935
2	6.843	7.398	0.555	2.670	2.167	0.503
3	6.880	7.171	0.291	2.643	2.241	0.402
4	6.954	7.603	0.649	2.512	1.974	0.538
5	7.086	7.591	0.505	2.554	2.138	0.416
Average	6.953	7.537	0.585	2.575	2.016	0.559
StdDev	0.097	0.278	0.230	0.078	0.273	0.218

Table A.4: Internal and measured odometry at the goal position after navigation using the SURF algorithm for the image nodes.

SURF	internal x	Measured x	diff x	internal y	measured y	diff y
1	6.992	7.923	0.931	2.612	1.559	1.053
2	7.052	7.398	0.346	2.533	2.167	0.366
3	7.038	7.171	0.133	2.781	2.241	0.540
4	7.002	7.603	0.601	2.603	1.974	0.629
5	7.360	7.591	0.231	2.676	2.138	0.538
Average	7.089	7.537	0.448	2.641	2.016	0.625
StdDev	0.154	0.278	0.321	0.093	0.273	0.257

Table A.5: Measured odometry at the goal position before and after navigation using the ORB algorithm for the image nodes.

ORB	before x	after x	diff x	before y	after y	diff y
1	7.366	7.902	0.536	2.561	2.073	0.488
2	7.362	8.280	0.918	2.322	1.234	1.088
3	7.038	7.364	0.326	2.729	2.498	0.231
4	7.167	8.116	0.949	2.720	1.306	1.414
5	7.133	8.014	0.881	2.626	1.112	1.514
Average	7.213	7.935	0.722	2.592	1.645	0.947
StdDev	0.146	0.348	0.277	0.166	0.608	0.566

Table A.6: Internal and measured odometry at the goal position after navigation using the ORB algorithm for the image nodes.

ORB	internal x	measured x	diff x	internal y	measured y	diff y
1	7.454	7.902	0.448	2.504	2.073	0.431
2	7.515	8.280	0.765	2.497	1.234	1.263
3	7.323	7.364	0.041	2.624	2.498	0.126
4	7.317	8.116	0.799	2.912	1.306	1.606
5	7.298	8.014	0.716	2.604	1.112	1.492
Average	7.381	7.935	0.554	2.628	1.645	0.984
StdDev	0.097	0.348	0.318	0.169	0.608	0.664

Appendix B

Raw Map data using different scan matching algorithms image node algorithms for scan matching.

Table B.1: Map data with the SIFT algorithm for image nodes.

SIFT	image node size	images in cell	grid cell size	current grid cell id
1	137	5	95	60
2	133	7	98	58
3	138	7	85	51
4	134	5	86	56
5	130	5	89	56
Average	134.4	5.8	90.6	56.2
StdDev	3.209361307	1.095445115	5.683308895	3.346640106

Table B.2: Map data with the SURF algorithm for image nodes.

SURF	image node size	images in cell	grid cell size	current grid cell id
1	128	5	95	56
2	124	4	93	61
3	108	5	89	55
4	143	6	88	64
5	100	4	84	52
Average	120.6	4.8	89.8	57.6
StdDev	16.96466917	0.836660027	4.324349662	4.827007354

Table B.3: Map Data with the ORB algorithm for image nodes.

ORB	image node size	images in cell	grid cell size	current grid cell id
1	114	5	83	51
2	148	4	88	50
3	110	7	99	53
4	157	5	81	63
5	119	5	95	50
Average	129.6	5.2	89.2	53.4
StdDev	21.38457388	1.095445115	7.694153625	5.504543578

Appendix C

Table C.1: The transform estimate of the difference in the robot's pose when viewing the goal image while using SIFT.

SIFT		1 dist	2 dist	3 dist	4 dist	5 dist
transform estimate	1	0.028	0.010	0.003	0.033	0.042
	2	0.023	0.002	0.002	0.032	0.034
	3	0.022	0.932	0.001	0.034	0.036
	4	0.017	0.931	0.002	0.033	0.034
	5	0.025	0.946	0.002	0.032	0.035
	6	0.030	0.933	0.004	0.035	0.037
	7	0.021	0.927	0.002	0.037	0.033
	8	0.025	0.945	0.002	0.035	0.035
	9	0.025	0.000	0.000	0.033	0.037
	10	0.000	0.000	0.000	0.033	0.038
	11	0.000	0.000	0.000	0.000	0.000

Table C.2: The transform estimate of the difference in the robot's pose when viewing the goal image while using SURF.

SURF		1 dist	2 dist	3 dist	4 dist	5 dist
transform estimate	1	0.310	0.064	0.007	0.036	0.036
	2	0.041	0.033	0.001	0.027	0.037
	3	1.168	0.048	0.027	0.040	0.036
	4	0.033	0.037	0.002	0.049	0.032
	5	0.032	0.039	0.002	0.033	0.036
	6	0.039	0.034	0.001	0.037	0.049
	7	0.034	0.039	0.001	0.048	0.092
	8	0.029	0.034	0.003	0.030	0.020
	9	0.035	0.037	0.001	0.037	0.041
	10	0.038	0.039	0.004	0.000	0.046
	11	0.023	0.031	0.000	0.000	0.018

Table C.3: The transform estimate of the difference in the robot's pose when viewing the goal image while using ORB.

ORB		1 dist	2 dist	3 dist	4 dist	5 dist
transform estimate	1	0.579	0.033	0.009	0.004	0.007
	2	0.590	0.036	0.018	0.026	0.008
	3	0.595	0.037	0.024	0.008	0.002
	4	0.589	0.034	0.017	0.003	0.017
	5	0.603	0.026	0.017	0.016	0.008
	6	0.584	0.034	0.019	0.004	0.008
	7	0.581	0.040	0.022	0.002	0.006
	8	0.587	0.036	0.010	0.004	0.006
	9	0.593	0.000	0.000	0.003	0.008
	10	0.568	0.000	0.000	0.003	0.012
	11	0.573	0.000	0.000	0.008	0.000

Appendix D

Table D.1: The transforms obtain while scan matching when observing the desk.

Facing Desk - Number	SIFT	SURF	ORB
1	0.003	0.169	0.096
2	0.003	0.285	0.033
3	0.007	0.015	0.028
4	0.006	0.152	0.151
5	0.011	0.062	0.107
6	0.012	0.165	0.115
7	0.046	0.108	0.041
8	0.004	0.028	0.049
9	0.005	0.046	0.082
10	0.004	0.056	2.518
11	0.002	0.049	1.143
12	0.059	0.129	0.052
13	0.003	0.07	0.105
14	0.004	0.049	0.101
15	0.003	0.914	0.045
16	0.059	0.208	0.046
17	0.004	0.058	0.053
18	0.014	0.142	0.083
19	0.008	0.051	0.05
20	0.005	0.055	0.121
21	0.012	0.057	0.015
22	0.036	0.094	0.044
23	0.009	0.061	0.264
24	0.073	0.036	0.034
25	0.014	0.217	0.068
26	0.007	0.043	0.012
27	0.013	0.242	2.029
28	0.006	0.245	0.247
29	0.086	0.143	0.053
30	0.005	0.001	0.036
31	0.002	0	0.048
32	0.006	0.003	0.051
33	0.037	0.004	0.121
34	0.004	0.06	0.127
35	0.002	0.015	0.08
36	0.019	0.002	0.06
37	0.003	0.069	0.195
38	0.005	0.645	0.052
39	0.043	0.002	0.159
40	0.004	0.073	0.12
41	0.005	0.131	0.556
42	0.011	0.083	0.117
43	0.052	0.042	0.033

44	0.05	0.11	0.012
45	0.002	0.073	0.077
46	0.007	0.053	0.09
47	0.055	0.044	0.085
48	0.011	0.067	0.45
49	0.042	0.169	0.053
50	0.006	0.006	0.099
51	0.006	0.003	0.225
52	0.005	0.021	0.098
53	0.044	0.089	0.015
54	0.002	0.002	0.01
55	0.005	0.037	0.09
56	0.006	0.115	0.035
57	0.015	0.01	0.013
58	0.031	0.131	0.019
59	0.015	0.003	0.091
60	0.011	0.007	0.053
61	0.007	0.052	0.243
62	0.003	0.084	0.102
63	0.003	0.132	0.039
64	0.639	0.2	0.032
65	0.047	0.003	0.073
66	0.003	0.055	0.015
67	0.012	0.005	0.287
68	0.016	0.127	0.006
69	0.05	0.083	0.045
70	0.065	0.048	0.088
71	0.012	0.041	0.047
72	0.068	0.005	0.05
73	0.005	0.054	0.051
74	0.004	0.009	0.105
75	0.003	0.025	0.038
76	0.005	0.054	0.048
77	0.245	0.094	0.094
78	0.046	0.003	0.085
79	0.005	0.093	0.128
80	0.042	0.058	0.09

Table D.2: The transforms obtain while scan matching when observing the cabinet.

Facing Cabinet - Number	SIFT	SURF	ORB
1	0.139	0.007	0.019
2	0.009	0.032	0.003
3	0.05	0.033	0.01
4	0.005	0.012	0.008
5	0.071	0.05	0.074
6	0.009	0.005	0.007
7	0.064	0.006	0.005
8	0.027	0.01	0.082
9	0.515	0.006	0.009
10	0.007	0.016	0.033
11	0.078	0.038	0.006
12	0.009	0.015	0.007
13	0.064	0.008	0.006
14	0.012	0.013	0.013
15	0.0113	0.12	0.035
16	0.002	0.056	0.033
17	0.05	0.042	0.286
18	0.011	0.021	0.055
19	0.036	0.002	0.004
20	0.02	0.074	0.15
21	0.069	0.086	0.016
22	0.006	0.008	0.1
23	0.036	0.007	0.007
24	0.004	0.072	0.005
25	0.062	0.011	0.007
26	0.01	0.006	0.035
27	0.102	0.039	0.164
28	0.065	0.007	0.005
29	0.025	0.017	0.049
30	0.055	0.007	0.032
31	0.005	0.002	0.002
32	0.063	0.021	0.047
33	0.008	0.072	0.03
34	0.002	0.02	0.013
35	0.104	0.007	0.005
36	0.066	0.01	0.004
37	0.012	0.025	0.009
38	0.002	0.356	0.002
39	0.051	0.006	0.047
40	0.003	0.001	0.013
41	0.01	0.005	0.033
42	0.045	0.009	0.035
43	0.043	0.023	0.004
44	0.021	0.011	0.006
45	0.057	0.005	0.002
46	0.007	0.026	0.052

47	0.068	0.038	0.213
48	0.01	0.264	0.062
49	0.097	0.007	0.059
50	0.006	0.017	0.007
51	0.071	0.027	0.06
52	0.006	0.003	0.002
53	0.005	0.002	0.077
54	0.092	0.148	0.007
55	0.006	0.021	0.006
56	0.062	0.022	0.005
57	0.009	0.149	0.041
58	0.104	0.011	0.009
59	0.01	0.084	0.033
60	0.002	0.037	0.063
61	0.004	0.013	0.023
62	0.138	0.003	0.026
63	0.013	0.013	0.081
64	0.008	0.01	0.005
65	0.043	0.027	0.005
66	0.002	0.012	0.056
67	0.004	0.003	0.013
68	0.002	0.022	0.04
69	0.217	0.016	0.01
70	0.002	0.01	0.053
71	0.096	0.004	0.067
72	0.002	0.022	0.011
73	0.118	0.013	0.031
74	0.009	0.027	0.03
75	0.002	0.012	0.054
76	0.002	0.051	0.029
77	0.092	0.117	0.028
78	0.022	0.006	0.097
79	0.002	0.041	0.063
80	0.004	0.173	0.056

Table D.3: The transforms obtain while scan matching when observing the white wall.

Facing Wall - Number	SIFT	SURF	ORB
1	0.063	1.483	0.53
2	0.046	1.444	1.4
3	1.387	1.033	0.085
4	1.428	3.238	0.884
5	0.584	2.191	1.18
6	0.172	0	2.164
7	0	0	1.374
8	0.014	0.504	1.235
9	0.018	0.867	0.762
10	1.386	0	1.017
11	1.546	0.618	1.288
12	0	0	0.027
13	0.015	0	1.465
14	0	1.097	1.258
15	0.068	0.056	0.776
16	1.275	0	0.578
17	0.013	2.21	0.473
18	1.435	0	0.817
19	0.097	1	0.841
20	0.016	1.455	1.51
21	1.454	2.723	0.173
22	0	1.014	0.174
23	0	1.58	0.149
24	1.417	1.343	0.393
25	1.021	3.873	0.428
26	0.848	0.753	0.022
27	0	0	1.473
28	1.712	0.017	0.79
29	1.126	0.191	1.091
30	0	2.219	1.107
31	0.011	2.221	1.141
32	0.084	0.017	1.053
33	0.036	0.021	0.019
34	0.516	0.008	1.319
35	0.082	0.044	1.207
36	0.022	0	0.917
37	0.076	0	0
38	0.021	2.778	1.203
39	0.008	0.027	0.912
40	0.055	2.823	1.399
41	0.017	1.131	0.008
42	0	0	1.269
43	0.023	2.348	1.357
44	0.107	0	1.503
45	1.416	0.497	1.483
46	0.506	0	0

47	0.014	0	0.075
48	0.011	2.221	1.076
49	0.024	0	0
50	0.159	1.188	1.392
51	0	2.188	0.703
52	0.046	0	0.374
53	1.45	0.616	0.725
54	0.114	2.845	1.34
55	0	0	0.835
56	0	0	1.457
57	0.022	0	0.417
58	0.003	0.632	0.614
59	0.036	0.019	0
60	0.014	0	1.181
61	0	0	1.546
62	0.571	0	0
63	1.303	0	1.275
64	0	0	1.351
65	0.026	0.025	1.51
66	0.012	0.021	0.053
67	1.181	1.146	0
68	1.47	0	0.032
69	0.03	0.094	1.163
70	1.467	2.79	1.424
71	0.013	0.946	0
72	1.475	0	0.303
73	0.041	0.02	0.365
74	0	0	0.191
75	0.039	2.204	1.467
76	0.018	1.083	0.022
77	0.521	0	1.295
78	0.108	0.314	1.274
79	0.086	0	0.032
80	0.023	1.35	0.027