Multiple Mobile Robot SLAM for collaborative mapping and exploration

Boitumelo Dikoko

Department of Electrical Engineering
University of Cape Town
Rondebosch, Cape Town
South Africa

Supervisors:
R.A Verrinder
Prof. E. Boje

June 2021

MSc. thesis submitted in fulfilment of the requirements for the degree of Master of Science in the Department of Electrical Engineering at the University of Cape Town

Keywords: Multiple Robot SLAM; Map Merging; Multi-Session Mapping

Declaration

I, Boitumelo Dikoko, hereby:

- grant the University of Cape Town free licence to reproduce the above thesis in whole or in part, for the purpose of research;
- declare that:
  - this thesis is my own unaided work, both in concept and execution, and apart from the normal guidance from my supervisor, I have received no assistance except as stated.
  - neither the substance nor any part of the above thesis has been submitted in the past, or is being, or is to be submitted for a degree at this University or at any other university, except as stated below.
  - I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed by candidate

Boitumelo Dikoko

Department of Electrical Engineering
University of Cape Town

Tuesday 15th June, 2021

Abstract

Multiple Mobile Robot SLAM for collaborative mapping and exploration

Boitumelo Dikoko

Tuesday 15th June, 2021

Over the past five decades, Autonomous Mobile Robots (AMRs) have been an active research field. Maps of high accuracy are required for AMRs to operate successfully. In addition to this, AMRs needs to localise themselves reliably relative to the map. Simultaneous Localisation and Mapping (SLAM) address the problem of both map building and robot localisation. When exploring large areas, Multi-Robot SLAM (MRSLAM) has the potential to be far more efficient and robust, while sharing the computational burden across robots. However, MRSLAM encounters issues such as difficulty in map fusion of multi-resolution maps, and unknown relative positions of the robots. This thesis describes a distributed multi-resolution map merging algorithm for MRSLAM. Hector-SLAM, which is one of many single robot SLAM implementations, has demonstrated exceptional results and was selected as the basis for the MRSLAM implementation in this project. We consider the environment to be three-dimensional with the maps being constrained to a two-dimensional plane. Each robot is equipped with a laser range sensor for perception and has no information regarding the relative positioning of the other robots. The experiments were conducted both in simulation and a real-world environment. Up-to three robots were placed in the same environment with Hector-SLAM running, the local maps and localisation were then sent to a central node, which attempted to find map overlaps and merge the resulting maps. When evaluating the success of the map merging algorithm, the quality of the map from each robot was interrogated. Experiments conducted on up to three AMRs show the effectiveness of the proposed algorithms in an indoor environment.

Acknowledgements

I would like to thank the Harry Crossley Foundation as a well as the the National Research Foundation (NRF), for financial assistance towards this research.

I would like to express my sincerest gratitude and appreciation to my supervisors, Robyn Verrinder and Prof. Edward Boje. Thank you for your endless support, guidance and motivation during my candidature, and for your feedback on this manuscript.

Endless thanks to my family for all their love, support and words of encouragement from afar. Finally, my deepest and sincerest gratitude to my wife, Viwe. Your patience and unconditional support has made this possible.

Contents

List of Figures

List of Tables

Acronyms

AMR	Autonomous Mobile Robot
CAD	computer-aided-design
DoG	Difference of Gaussian
EKF	Extended Kalman Filter
FAST	Features from Accelerated and Segments Test
FLANN	Fast Library for Approximate Nearest Neighbors
FOV	Field of View
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
IMU	inertial measurement units
IR	infrared
KLT	Kanade Lucas Tomas
LIDAR	Light Detection and Ranging
LMEDS	Least Median of Square
LoG	Laplacian of Gaussian
MIT	Massachusetts Institute of Technology
MRSLAM	Multi-Robot SLAM
MSE	mean-square error
OpenCV	Open Source Computer Vision Library
ORB	Oriented FAST and Rotated BRIEF
PGVD	probabilistic generalised Voronoi diagram
RANSAC	random sample consensus
RBPF	Rao–Blackwellised particle filter
ROS	Robot Operating System
SIFT	Scale-invariant feature transform
SLAM	Simultaneous Localisation and Mapping
SOM	self-organised map
SURF	Speed up Robust Features
TsICP	trimmed and scalingiterative closest point
USAR	Urban Search and Rescue

Chapter 1

Introduction

1.1 A brief background to the study

A key feature of robot autonomy is the ability of a platform to both sense its environment and situate itself within it [2]. Therefore, while exploring an unknown environment, an autonomous mobile robot must build a local map and position itself within the map using available sensor data (such as LIDAR, cameras and inertial measurement units (IMU)). This process is referred to as Simultaneous Localisation and Mapping (SLAM) [3, 4].

Multi-Robot SLAM (MRSLAM) is a far more robust and efficient strategy when exploring large areas, as computational burdens are shared across many robots. Multiple robots reduce the time needed to map the same area by exploring different parts of the environment in parallel. The robot team is robust as a decentralised formation as the failure of one robot does not necessarily halt the mission [5], and for a mapping mission to be completely decentralised, each robot needs to operate independently, and no robot should be of higher importance than any other robot in the mapping task. Furthermore, local maps generated by the robots can be shared amongst the platforms and merged to be used for additional mission tasks such as navigation and path planning.

Even though using MRSLAM can be advantageous, the following considerations arise when merging a map:

- Unknown relative robot poses and the uncertainty associated with their local and relative positions in a global map.
- Varying map representation, resolution and orientation across robots.
- Incorporating both location and mapping uncertainty into the final map fusion [6].

Merging existing maps from multiple robots is the process of using raw map data from multiple robots to produce a final global map and as such can result in various map representations across the robot platforms. There are two approaches: one approach takes raw data from all the robots, produces a global map centrally and then shares the resulting map with all the robots (this may need to be transformed into a representation

that the local robot can use), and the second approach robots share raw data among each other, and each robot produces their own global map [6].

Previous MRSLAM approaches have often assumed robots know their location in a global map using measurements from Global Positioning Systems (GPS) [7, 8]. With this assumption, robots have a straightforward objective to explore as much of the map as possible, and it further allows for more detailed map merging as the ground truth is always known. However, GPS is not always available, especially in indoor environments. Work done by [9], [10] addresses the lack of GPS data, by relying on rendezvous positions in the environment to verify the relative robot pose and to initialise a map merging event. The drawback of this is if robots fail to reach the rendezvous position, a merging event may never occur.

Other studies approach map merging as an optimisation problem [11–13], by performing a transformation that optimises map overlap. These approaches promise an optimal solution when the number of iterations approaches infinity, although optimality is not always guaranteed in real-time applications. Saeedi et al. tackle the map merging problem as a particular case of image registration [14, 15]. Even though this method has proven to be computationally superior to the others, they assume that the initial maps are of the same scale and generated at the same resolution. This approach is not robust when multi-robots generate maps of different scales and resolutions or when higher resolution is needed in a particular map region, and this, therefore, needs to be considered. Multiple resolution maps are ideal in scenarios such as a mine, as mines are usually large environments, and some areas may include more information than others. For areas where the environment has many details, a fine resolution map can be built; and for areas, with fewer details, a coarse resolution map can be generated. Thus, it is necessary to consider merging multiple grid maps at different scales and resolutions.

1.2 Problem statement

This thesis was motivated by the need to map known and unknown structured environments using multiple mobile robot platforms. Each mobile robot needs to produce a local map, retrieve a previous map, and update the global map. Maps provided by multiple robots are then merged on-board each robot to produce a global map locally. In general, some areas of the environment may include more details than others. For compactness purposes, the map could be built at a fine resolution in the area with more details and a coarse resolution in the area with fewer details. Thus, maps generated at various resolutions need also to be handled by the merging process.

1.3 Objective of research

The research aim in this study is to investigate and produce an implementation of Multiple Mobile Robot SLAM for Collaborative Mapping and Exploration. The project was focused on two main components of this process: (a) multi-session mapping and (b) map merging.

In this thesis, multi-session mapping uses an existing local map to update and expand on a previous map of the environment. This means a robot will either use a map previously created by another robot or itself, to update and expand the current local map. The robot needs to either localise within the previously mapped region or match features in its local map to that region.

The map merging algorithm was based on a multi-session implementation and used a 2D occupancy grid map to represent the mapped structured environments. Application of the map merging algorithm was conducted in a distributed manner across robot platforms to deal with potential communication loss between the mobile robots in the environment. Also, since the relative pose geometric transformation between the robots is unknown, improved pose geometric transformation estimation techniques are considered.

Additionally, this study addressed problems of: (a) unknown re-entry into a previously mapped area and (b) map merging of maps created by independent robots with unknown initial poses, and unknown relative pose transformations between the robots.

1.4 Thesis scope and contributions

In this thesis, an occupancy grid-based multi-session mapping and the merging algorithm is developed. In this approach robots individually build maps using HectorSLAM to generate initial local occupancy grid maps of the environment. When robots can communicate, local maps are then shared with other robots. An image registration technique, Scale-invariant feature transform (SIFT), is used to identify common features between the maps before merging takes place [16]. It can handle both mappings; mapping a new unknown environment and merging a current local map to a previously mapped region. The multi-session map merging algorithm includes rules that address multi-session mapping and merging of maps generated by other mobile robot platforms with unknown initial position and relative pose transformations. Additionally, it can identify overlap or the lack of overlap between multiple maps and is run in a decentralised manner on each robot in the system.

The developed algorithm is implemented on ROS (Robot Operating System) using python and validated in simulation (using Gazebo Simulator) and real-world environments. In the real-world environments, raw data from the Ray and Maria Stata Center [1] and raw data produced for this thesis using a Kobuki platform were used. The Kobuki platform and the platform used in the Ray and Maria Stata Center dataset were both equipped with a two-dimensional Hokuyo UTM-30LX LIDAR scanner in an indoor structured environment.

This approach us designed to work with data generated from a two-dimensional laser scanner and is not evaluated for a three-dimensional laser scanner. To simulate the algorithm's real-time running, the datasets were stored as rosbag files (a ROS data store format) then played back as though multiple robots were recording data as different locations in the map. Additionally, all maps were represented using occupancy grid maps and were shared through ROS communication, then converted into image files before map

[1] https://projects.csail.mit.edu/stata/

merging events on each robot. Transmission of map data between robots and storage constraints of maps on-board each robot were not evaluated in this study.

1.5 Plan of development

This thesis is organised into six chapters, bibliography and appendices, each referring to the project phases. Firstly this introductory chapter introduces the context and motivation for this study. Chapter 2 provides background to the SLAM problem as a basis for some of the study choices; furthermore, the chapter reviews the advantages and disadvantages of existing multiple-robot SLAM algorithms. Chapter 3 address map merging, which includes parameter tuning for an image registration implementation. Chapter 4 addresses the implementation of multiple-robot SLAM and several important steps to achieve this using ROS. In this chapter, more details of the basic ROS modules are given for the multiple-robots used in this work. In Chapter 5, the experimental validation results are presented for both the simulated and real-world environments. Advantages and disadvantages of the approach are also highlighted and discussed. To conclude, Chapter 6 sums up the work, providing conclusions and suggestions for future work for this research.

Chapter 2

SLAM overview

In this chapter, a review of SLAM fundamentals is presented. Background concepts are discussed to establish a framework for the map merging problem presented in later chapters, and basic concepts of autonomous mobile robot systems are introduced as a basis for the SLAM framework.

2.1 Autonomous mobile robot system overview

Imagine a mobile robot has been given a mission to explore an unknown environment. To achieve this mission, the robot first gathers data from the unfamiliar environment (**perception**), then maps and localises itself within the environment (**SLAM**), followed by developing a plan on how to move through the world based on the goal of the mission (**navigation** and **path planning**). Finally, it initiates the robot's controlled motion along the decided path (**motion control**). Each module in the system is dependent on the quality of outputs from the previous modules. For example, to perform path planning successfully, a high-quality map is needed from the SLAM block, which requires that the data collected from the environment by the perception block is of a suitable format, scale and resolution to generate a map and localise the robot within it. In the sections that follow, the perception and SLAM blocks are explored in detail and, while an in-depth explanation of path planning and motion control is omitted, the output of the SLAM sub-system will be discussed concerning required input to the path planning module.

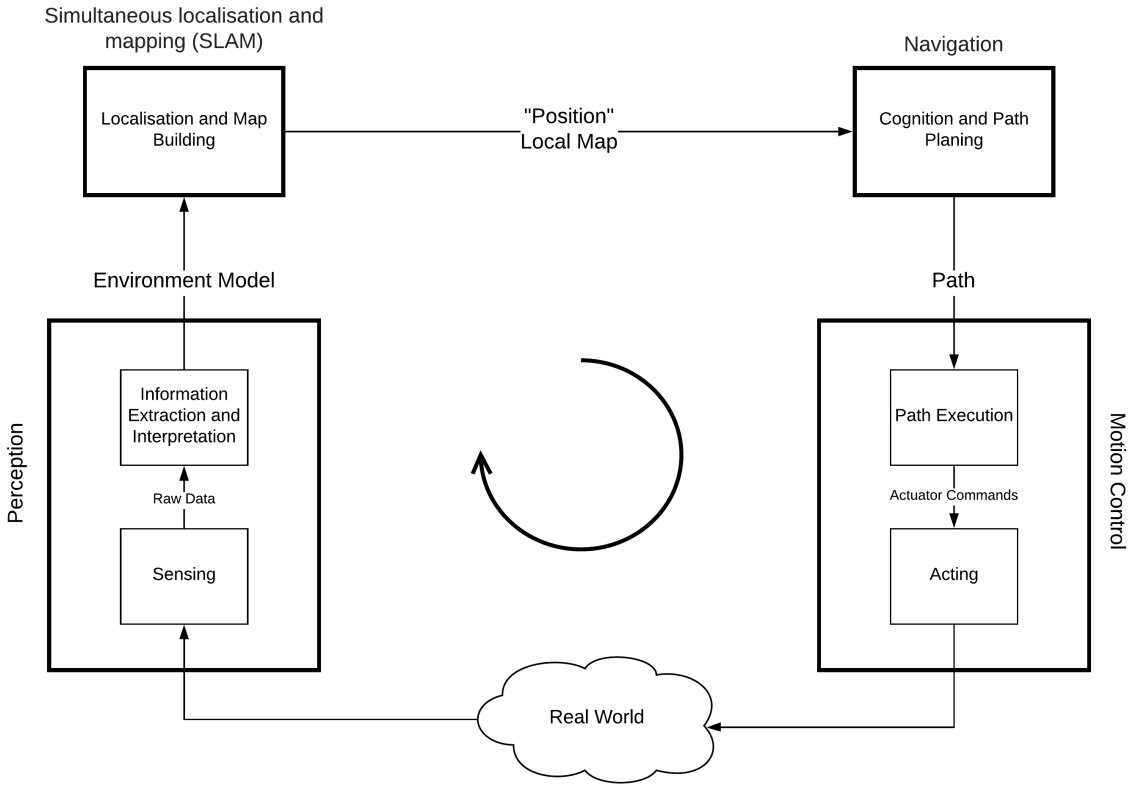


Figure 2.1: A high-level overview of a typical autonomous mobile robot system. Illustrating the interactions between the real-world environment, its perceptions system, the robots internal model of the world, its planning unit and finally its motion control centre. Adapted from [2].

Figure 2.1 is an illustration of an AMR system where different interdependent modules are displayed, each module both receives and shares information with other modules in the system [2].

2.2 Perception system

For an AMR system to interact with the real world, it needs to acquire information about the environment as it moves through it. This process is referred to as perception and involves taking measurements using multiple sensors followed by extracting useful information from these measurements. There are various sensors one can choose from when designing an autonomous robot system and depend on the type of robot platform and the environmental context in which it will operate. For example, a terrestrial robot might use GNSS positioning systems, inertial measurement units (IMU), Light Detection and Ranging (LIDAR), RADAR, and cameras to take both the robot's measurements internal state and external environmental variables. In contrast, underwater robots might use a combination of SONAR and inertial sensors. A robot's perception system can sense both internal properties of the robot and external properties of the environment [17]. These sensors can be primarily classified as exteroceptive/proprioceptive and active/passive.

Exteroceptive sensors acquire information about the external environment. For example, distance and angle measurements from the robot to objects in the environment, environmental light intensity or acoustic amplitude. Therefore, exteroceptive sensors are used by the robot to extract information about the state of the environment. **Proprioceptive** sensors measure internal parameters of the robot; for example, wheel rotations, motor speed, angular velocity, linear acceleration and battery current/voltage.

Furthermore, sensors can also be distinguished by whether they are active or passive, i.e. whether they emit energy into the environment or not. **Passive** sensors measure ambient energy from the environment; for example, temperature probes, microphones and cameras. **Active** sensors, such as LIDAR, stimulate the environment by emitting energy into it and then measure the reaction from the environment.

Most mobile robots use some form of **range sensor** to measure their relative position to objects in the environment. These sensors include ultrasonic, infrared (IR), stereo cameras, RADAR and LIDAR devices. A range sensor gives the distance to the nearest object in a given direction. Range sensors, typically, form the basis of methods for detecting obstacles in the environment and as inputs to map generation and localisation. Range sensors also play a role in correcting odometry errors where they are present.

Odometry refers to the use of the motion of actuators such as wheels and treads, to estimate the overall motion of a robot. This estimation is based on developing a mathematical model of how actuator motion induces motion on the robot itself. This motion mapping is integrated over time to develop a model of the robot's pose as a function. This process of estimating the pose is known as **dead reckoning** [17]. Even though odometry sensors provide information about robot motion, the information accumulates errors over time, which are not corrected by the sensors themselves. In many cases, LIDAR and GPS are used used to complement odometric information [18].

The perception system provides input information to the localisation and map building (SLAM) modules, which need to process the data further to extract a map model of the environment and localise the robot within it. The following section discusses the SLAM module in more detail.

2.3 SLAM process

SLAM builds a new map of an unknown environment or updating an existing map, while simultaneously localising the robot within the map [3]. There are many techniques used to handle errors generated by the system. These techniques include feature detection between steps [8], data association [19] or loop closure detection [20]. SLAM is mainly solved using a probabilistic approach, as the robot's perception system has an uncertainty associated with measurements and inherent sensor noise.

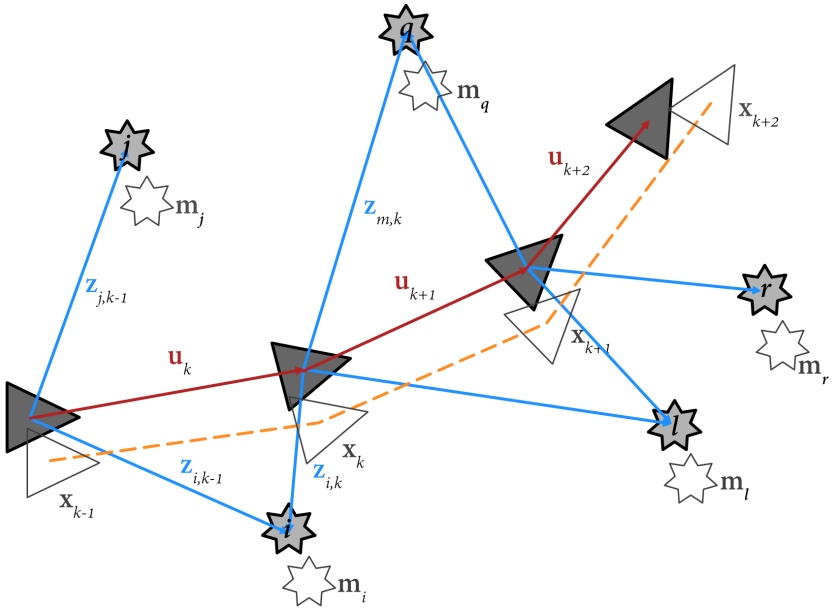


Figure 2.2: Graphical representation of the SLAM process. As a robot (▷) moves through (—) an environment, which contains a series of discrete landmarks (✶), it needs to simultaneously estimate the locations of the landmarks (☆) and itself (▷) relative to the mapped landmarks. The locations of both the robot ($\mathbf{x}_k$) at time k and the i^{th} landmark ($\mathbf{m}_i$) are estimated from observations (→, $\mathbf{z}_k$) taken by the robot's perception system and models of the robot's motion overtime after a control input ($\mathbf{u}_k$) has been applied to the robot at time $k-1$. Additionally, it uses this information to build up a history of its trajectory through the environment (- - -, $\mathbf{X}_{0:k}$). Modified from [21].

The SLAM problem can be formulated as follows[1] : Continuing with the scenario described in Section2.1, a mobile robot enters an environment, which contains a number of discrete landmarks, whose true locations are assumed to be time-invariant. As it moves through the environment it needs to simultaneously produce a map (m of all the estimated landmark positions in the environment ($\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n\}$), where $\mathbf{m}_i$ is a vector representing the estimated location of the i^{th} landmark in a set of n observed landmarks, and estimate the robot's trajectory ($\mathbf{X}_{0:k}$), where ($\mathbf{X}_{0:k} = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k\}$) is a set of all estimated robot poses within the environment over time, starting from an initial state $\mathbf{x}_0$. $\mathbf{x}_k$ is a state vector describing the global location and orientation of the vehicle, at time k, where $k \in \{0, 1, \ldots, k\}$.

For the robot to move from one location ($\mathbf{x}_{k-1}$) to the next ($\mathbf{x}_k$), a control input $\mathbf{u}_k$, is applied to the robot at time $k-1$. $\mathbf{U}_{0:k} = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_k\}$ is a set of all the control inputs applied to the robot to move it from its initial state $\mathbf{x}_0$ to its current state $\mathbf{x}_k$. At each sample in time, k, the robot takes measurements $\mathbf{z}_{i,k}$ of the relative position and pose between its true location and the i^{th} landmark. $\mathbf{z}_k$ represents the observations to all detectable landmarks at time k and $\mathbf{Z}_{0:k} = \{\mathbf{z}_0, \mathbf{z}_1, \ldots, \mathbf{z}_k\}$ is the history of the relative position measurements to all observed landmarks for from the initial to the current location. See Figure 2.2 for a graphical representation of the process.

[1] **Note:** The notation used in formulating the SLAM problem is largely based on [3].

The uncertainty in robotics can be represented using the calculus of probability theory, which instead of relying on a single *"guess"*, the probabilistic algorithms represent information by a probability distributions over a whole space of guesses. The probabilistic form of SLAM uses Bayes Theorem to estimate the **joint posterior** probability distribution ($bel(\mathbf{x}_k, \mathbf{m})$), for both the current robot state ($\mathbf{x}_k$) and map ($\mathbf{m}$) at time k given a history of all observations $\mathbf{Z}_{0:k}$ and control $\mathbf{U}_{0:k}$ inputs to the platform from its initial state $\mathbf{x}_0$ to its current state $\mathbf{x}_k$,

$$bel(\mathbf{x}_k, \mathbf{m}) = P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0). \tag{2.1}$$

The SLAM algorithm is implemented in two recursive steps, time-update $\rightarrow$ measurement-update to integrate new sensor data while marginalising out the previous robot pose $\mathbf{x}_{k-1}$. The prediction step incorporated the control input ($\mathbf{u}_k$), and the measurement-update incorporates the measurement ($\mathbf{z}_k$) into the posterior, hence their respective naming. The approach models the SLAM problem by assuming that the dynamics are Markov, where the future is assumed to be independent of the past, given the present state. The Bayes filter below uses an odometry-based model, while Prof. Thrun provides a more general Bayes filter in [22]. The steps are in the following form:

Time-Update: The current robot state is predicted using the prior belief $bel(\mathbf{x}_{k-1}) = P(\mathbf{x}_{k-1}, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0)$ and the robot's motion model (with associated assumptions such as constant velocities). Using the law of total probabilities, the prior belief $bel(\mathbf{x}_{k-1})$ is marginalised out to form the prediction $\bar{bel}(\mathbf{x}_k)$:

$$\bar{bel}(\mathbf{x}_k, \mathbf{m}) = \underbrace{P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}_{\text{prediction}} = \int \underbrace{P(\mathbf{x}_k | \mathbf{x}_{k-1}, \mathbf{u}_k)}_{\text{motion model}} \times \underbrace{P(\mathbf{x}_{k-1}, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0)}_{\text{prior belief}} d_{\mathbf{x}_{k-1}} \tag{2.2}$$

Here, the motion model describes the probability of robot's pose, $\mathbf{x}_k$, given the preceding pose, $\mathbf{x}_{k-1}$, and the applied control input, $\mathbf{u}_k$. The motion model is assumed to be a Markov process independent of both observation, $\mathbf{z}_k$, and map, $\mathbf{m}$.

Measurement-Update: The posterior belief ($bel(\mathbf{x}_k)$) is then formed using Bayes rule to form:

$$\begin{aligned} bel(\mathbf{x}_k, \mathbf{m}) = \underbrace{P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)}_{\text{posterior}} &= \frac{\overbrace{P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m})}^{\text{observation model}} \overbrace{P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}^{\text{prediction}}}{P(\mathbf{z}_k | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \\ &= \eta \overbrace{P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m})}^{\text{observation model}} \overbrace{P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}^{\text{prediction}} \end{aligned} \tag{2.3}$$

Here, the observation model $P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m})$ describes the probability of an observation, $\mathbf{z}_k$, when a robot's pose,$\mathbf{x}_k$, and map/landmark locations, $\mathbf{m}$, are known. Assuming that a robot's pose, $\mathbf{x}_k$, and map/landmark locations, $\mathbf{m}$, are defined, observations are conditionally independent given the map and current robot pose. The term $P(\mathbf{z}_k | \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})$

does not depend on $\mathbf{x}$ and in practice this term is not calculated; rather the posterior distribution is normalised to one [22]. The normalisation term η, is therefore, given by $\eta = 1/P(\mathbf{z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})$.

Therefore in a Bayes filter the joint posterior for $P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)$ can be calculated recursively using equations 2.2 and 2.3, given all observations $\mathbf{Z}_{0:k}$ and all control inputs $\mathbf{U}_{0:k}$ up to and including the current time k.

The time-update and measurement-update can then be represented as an algorithm. Algorithm 2.1, describes the time-update step $\bar{bel}(\mathbf{x}_k)$ which predicts the state of the mobile robot at time k based on previous sate posterior without incorporating the measurement at time k, and the measurement-update step $bel(\mathbf{x}_k)$ which incorporates the measurement to improve the posterior belief distribution.

Algorithm 2.1: The algorithm calculates the belief distribution bel from measurements $\mathbf{z}_k$ and control inputs $\mathbf{u}_k$. The belief $bel(\mathbf{x}_k)$ is calculated from $bel(\mathbf{x}_{k-1}), \mathbf{u}_k, \mathbf{z}_k$, where the $\mathbf{z}_k$ and $\mathbf{u}_k$ are the most recent measurements and input control [22], where $bel(x_k) = P(x_k, m|Z_{0:k}, U_{0:k}, x_0)$. This is well illustrated in 2.2, where the SLAM problem is shown.

Data: $bel(x_{k-1}), u_k, z_k$
1 **for** *all* x_k **do**
2 $\quad \bar{bel}(x_k) = \int p(x_k|u_k, z_k) bel(x_{k-1}) dx_{k-1}$;
3 $\quad bel(x_k) = \eta p(z_k|x_k) \bar{bel}(x_k)$;
4 **end**
5 **return** $bel(x_k)$

With SLAM's foundations established, the following section will expand on some of the most commonly used approaches.

2.4 SLAM approaches

Expanding on the above-mentioned probabilistic approach to SLAM, several methods can solve the SLAM problem, which involves finding an appropriate representation for both the observation and motion models, that allow efficient computation of the prior (2.2) and posterior (2.3) distributions. The three most commonly used SLAM solutions are EKF-SLAM, Fast-SLAM and Graph-SLAM. However, new methods such as Hector-SLAM are also useful. The following section explored the solutions mentioned above, even though there are many alternative approaches [23].

2.4.1 EKF-SLAM

The Extended Kalman Filter (EKF) is the most common representation as it is one of SLAM's earliest implementation. The basic Kalman Filter implementation only deals with linear systems with Gaussian noise, and the EKF implementation extends to non-linear systems [24]. The EKF replaces the Kalman Filter motion model with a nonlinear

model, as a linear system is rarely observed in the real-world.

The EKF approach is a state-space model, with additive Gaussian noise. The EKF model was first proposed using a single state vector to estimate the mobile robot's locations and features in an environment, and these estimations included an associated error covariance matrix that represents the uncertainty of results [25]. During exploration, the state vector and covariance matrix are updated using the environment measurements and an EKF is introduced [3]. The covariance matrix grows quadratically as new features are observed, and thus new states are added.

In the EKF, the nonlinear model is linearised using a Taylor expansion around the current state estimate [26]. Where, the current state is represent by a mean state vector $\mu_k = [\mathbf{x}_k, \mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n]^T$ as well as a covariance matrix $\mathbf{\Sigma}_k$ which signifies the co-variance between the state variables also a measure of confidence at time k. As with the general Bayes filter, the EKF approach iterates between the time-update $\rightarrow$ measurement-update cycle. A brief overview is given mathematically in the rest of this section. In chapter 10 of [22], Thrun et al. present an in-depth EKF based SLAM tutorial.

Time-Update: In this stage the filter computes the co-variance matrix $\bar{\mathbf{\Sigma}}_k$ and predicted the state vector $\bar{\mu}_k$ and time k:

$$\begin{aligned} \bar{\mu_{\mathbf{k}}} &= g(\mathbf{u}_k, \mu_{k-1}) \\ \bar{\mathbf{\Sigma}_{\mathbf{k}}} &= \mathbf{G}_k \mathbf{\Sigma}_{k-1} \mathbf{G}_k^T + \mathbf{Q}_k, \end{aligned} \tag{2.4}$$

Here, g is the motion model of the robot updated using given odometry. The covariance matrix $\bar{\mathbf{\Sigma}}_{\mathbf{k}}$ is updated by the motion model's Jacobian $\mathbf{G}_k = \frac{\delta g}{\delta \mu_{k-1}}$ and the process noise $\mathbf{Q}_k$. The process noise results from the motion model, which is assumed to be additive and is typically the same as the odometry noise.

Measurement-Update: In this stage, updated measurement of state vector μ_k and it is associated $\bar{\Sigma}_k$ are computed, by first computing the Kalman gain $\mathbf{K}_k$. The Kalman gain $\mathbf{K}_k$ is the relative weight given to the measurements and current state estimate given by Equation 2.5.

$$\mathbf{K_k} = \bar{\mathbf{\Sigma}}_k \mathbf{H}_k^T (\mathbf{H}_k \bar{\mathbf{\Sigma}}_{k-1} \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{2.5}$$

Here, the sensor model's Jacobians $\mathbf{H}_k = \frac{\delta h}{\delta \bar{\mu_k}}$ apply a linearisation and the sensor model noise is incorporated by the co-variance $\mathbf{R}_k$. This gain can be tuned to achieve a particular performance; with a higher gain placing more weight on current measurements and lower gain, placing more weight on the prediction [22].

The Kalman gain is then used to compute the posterior estimate state vector, μ_k,and its associated covariance matrix $\mathbf{\Sigma}_k$.

$$\begin{aligned} \mu_k &= \bar{\mu_k} + \mathbf{K}_k (\mathbf{z}_k - h(\bar{\mu_k})) \\ \mathbf{\Sigma}_k &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\mathbf{\Sigma}}_k \end{aligned} \tag{2.6}$$

Here, h is the sensor model and $\mathbf{z}_k$ is the sensor measurements. The posterior μ_k contains updated estimates for the state vector, which includes both the robot's pose $\mathbf{x}_k$ and $\mathbf{m} = \{\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n\}$), where $\mathbf{m}_i$ landmarks

Algorithm 2.2 shows the summarised Extended Kalman Filter algorithm used to compute the Gaussian posterior in 2.4, where $\mu_k = [\mathbf{x}_k, \mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n]^T$ is a mean vector and $\mathbf{\Sigma}_k$ is a co-variance matrix at time k.

Algorithm 2.2: Extended Kalman Filter Algorithm [22], where $\mu_k = [\mathbf{x}_k, \mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_n]^T$ is a mean vector, $\mathbf{\Sigma}_k$ is a co-variance matrix at time k

Data: $\mu_{k-1}, \Sigma_{k-1}, u_k, z_k$

$\bar{\mu_k} = g(u_k, \mu_{k-1})$;

$\bar{\Sigma_k} = G_k \Sigma_{k-1} G_k^T + Q_k$;

$K_k = \bar{\Sigma_k} H_k^T (H_k \bar{\Sigma_k} H_k^T + R_k)^{-1}$;

$\mu_k = \bar{\mu} + K_k(z_k - h(\bar{\mu_k}))$;

$\Sigma_k = (I - K_k H_k)\bar{\Sigma_k}$;

return μ, Σ

EKF-SLAM employs linearised models of nonlinear motion and observation models, where errors are introduced by linearisation. At each time step, k, as the robot moves in the environment, its current pose uncertainty increases, the uncertainty is reduced by recurringly observing landmarks. This process in EKF-SLAM aims to marginalise out the previous estimate μ_{k-1}, to only incorporate the most recent linearisation errors into the current state μ_k. This process helps limit pose errors that accumulate due to linearisation and odometry errors. Thus, an EKF will always underestimate uncertainties and eventually become inconsistent [27, 28]. An alternative formulation aimed to improve the inconsistencies is proposed in Castellanos et al.; Where, a robot-centred coordinate system is employed to delay the inherent inconsistency [27, 29], which is adopted in [30, 31] too.

The EKF-SLAM solution is fragile to incorrect data association of landmark observations, given that the pose representation is uni-modal with no opportunity to track alternative hypotheses. This incorrect data association causes a problem referred to as loop closure. The loop closure problem occurs when a robot revisits a previously visited location after a lengthy expedition [32]; therefore, the problem is compounded in environments where landmarks look different when observed from a different viewpoint [33].

At each time step, k, the measurement-update step needs to update all landmarks and the joint covariance matrix, every time an observation, $\mathbf{z}_k$ is made. Inherently, this scales EKF-SLAM with $\mathcal{O}(M^2)$, where M is the number of landmarks in the map $\mathbf{m}$. This scaling limitation of EKF-SLAM curbs the online execution to several hundred landmarks [34].

Even though implementations using Field Programmable Gate Arrays (FPGAs) have been demonstrated with 1800 features, [35]; solutions such as this scale with silicon size, which is ultimately still limited. Other approaches to scale the EKF-SLAM solution to larger map size, propose fragmenting maps into smaller "sub-maps". The sub-mapping technique aims to maintain online performance by restricting the map updates to the landmarks around the robot, such as in [36–40]. However, these methods inherently scale

with $\mathcal{O}(M^2)$ on global state and covariance updates.

The method, FastSLAM described in the section to follow attempts to improve the computational, data association and convergence limitations of EKF-SLAM. It was the first model to consider nonlinear models and multi-modal distributions to solve the SLAM problem, with the further advantage of better robustness to loop-closure [41].

2.4.2 FastSLAM

Montemerlo et al. first introduced FastSLAM in [41], while previous efforts were placed on EKF-SLAM. FastSLAM is a recursive sampling or particle filter, representing the robot pose as a non-Gaussian distribution, unlike the EKF, which represents distributions with a multivariate Gaussian. FastSLAM utilises Rao-Blackwellized Particle Filter, which exploits the conditional independence between landmarks in the fundamental SLAM problem 2.1. Therefore, Rao-Blackwellisation factorises the SLAM problem by separating them into the robot's complete trajectory, $\mathbf{X}_{0:k}$, and map, $\mathbf{m}$, components:

$$\underbrace{P(\mathbf{X}_{0:k}, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)}_{posterior} = \underbrace{P(\mathbf{m}|\mathbf{X}_{0:k}, \mathbf{Z}_{0:k})}_{map} \underbrace{P(\mathbf{X}_{0:k}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0)}_{trajectory} \tag{2.7}$$

Furthermore, in FastSLAM, the mapping problem is factorised into a separate problem, one for each feature. The features are estimated by two-dimensional EKFs, one for each feature:

$$P(\mathbf{m}|\mathbf{X}^{[q]}_{0:k}, \mathbf{Z}_{0:k}) = \prod_{n}^{\mathbf{N}} P(\mathbf{m}_n|\mathbf{X}^{[q]}_{0:k}, \mathbf{Z}_{0:k}). \tag{2.8}$$

This approach is different from the EKF SLAM discussed in the previous Section2.4.1, where a single Gaussian is used to estimate all features in the map.

FastSLAM maintain Q particles, at some point in time, k, where q $(1 \leq q \leq Q)$ particles of:

$$\mathbf{X}^{[q]}_{0:k}, \mu^{[q]}_{k,1}, \ldots, \mu^{[q]}_{k,N}, \mathbf{\Sigma}^{[q]}_{k,1}, \ldots, \mathbf{\Sigma}^{[q]}_{k,N} \tag{2.9}$$

Here, $[q]$ is the index of the particles. Each particle contains a sample path, $\mathbf{X}^{[q]}_{0:k}$; and a set of N two-dimensional Gaussians with means, $\mu^{[q]}_{k,n}$ and variance, $\mathbf{\Sigma}^{[q]}_{k,n}$ one for each landmark, where n $(1 \leq n \leq N)$ is the index of the landmark. Therefore, Q particles contain Q paths and QN Gaussians that describe landmarks in the environment.

FastSLAM is initialised by setting each particle's robot location to its known starting coordinates and zero the map. Particles are updated in two steps:

- **Time-update**, which is when a new odometry reading $\mathbf{u}_k$ at time k is received, where a new location variable is stochastically generated for each particle. The

distribution for generating these particle locations is based on the motion model $\mathbf{x}_k^{[q]} \approx P(\mathbf{x}_k|\mathbf{x}_{k-1}^{[q]}, \mathbf{u}_k)$, where $\mathbf{x}_{k-1}^{[q]}$ is the previous location of particle M.

- **Measurement-Update**, which is when a new measurement $\mathbf{z}_k$ at time k is received. Where, firstly FastSLAM computes the probability of the measurement $\mathbf{z}_k$, defined by

$$\mathbf{w}_k^{[q]} : = \mathcal{N}(\mathbf{z}_k|\mathbf{x}_k^{[q]}, \mu_{k,n}^{[q]}, \mathbf{\Sigma}_{k,n}^{[q]}). \tag{2.10}$$

  Here, $\mathbf{w}_k^{[q]}$ is called the importance weight, which measures how important the particle is given the new measurement $\mathbf{z}_k$; and $\mathcal{N}$ denotes the normal distribution, for the specific value, $\mathbf{z}_k$.

  Secondly, particles are drawn from the existing set based on a criterion, and this is referred to as the re-sampling step. The idea behind re-sampling is merely selecting the particle for which the measurement is more likely to survive the re-sampling process; for example, some implementations re-sample every time interval, others at specific time intervals and some when the weight variance exceeds a threshold [42].

  Finally, the new particle set mean and covariance are updated, based on the measurement, $\mathbf{z}_k$ at time k. This update follows the EKF update rule discussed in Section2.4.1.

There are two versions of FastSLAM, version 1.0 [41] and 2.0 [43]. FastSLAM 2.0 improves FastSLAM 1.0 by forming a different proposal distribution (see Equation 2.12) which takes into account the measurement $\mathbf{z}_k$, when sampling the pose $\mathbf{x}_k$. The improvement is more evident in cases where $\mathbf{u}_k$ has a relative accuracy lower than $\mathbf{z}_k$.

The proposal distribution for FastSLAM 1.0 is the motion model

$$\mathbf{x}_k^{[q]} \approx P(\mathbf{x}_k|\mathbf{x}_{k-1}^{[q]}, \mathbf{u}_k) \tag{2.11}$$

where the proposal distribution for FastSLAM 2.0 includes the current observation

$$\mathbf{x}_k^{[q]} \approx P(\mathbf{x}_k|\mathbf{X}_{0:k-1}^{[q]}, \mathbf{Z}_{0:k}, \mathbf{u}_k) \tag{2.12}$$

The FastSLAM algorithm has three remarkable features. First, each particle representing the entire path, but the update equation only uses the current robot pose, allowing SLAM to be solved in real-time. Secondly, FastSLAM makes it possible to estimate various data association hypotheses on a per-particle basis. This data association representation provides FastSLAM with a powerful tool to solve the data association problem, which leads to a robust loop closure technique. Thirdly, FastSLAM can be implemented with advanced tree methods such as *ancestry trees*, representing the map estimates [44]; this enables particles to share parts of the maps that are identical. Enabling FastSLAM to update in $\mathcal{O}(Q \log M)$, where Q is the number of particles and M the size of the map [2]. FastSLAM has demonstrated on Victoria Park extensive data set to 25 times faster than EKF SLAM [45].

The most significant limitation of FastSLAM is the fact that even with a large number of particles, there may be no particles in the vicinity of the correct state [22]. This limitation is referred to as particle deprivation. Particle deprivation occurs due to random sampling variance; an unlucky series of random numbers can wipe out all particles near the correct

state. Therefore, when implementing FastSLAM, care has to be taken to reduce the effects of particle deprivation.

In the short-term FastSLAM produces accurate estimation, but in the long-term, it is unable to explore the state-space to be a reasonable Bayesian estimator [46]. However, the results of FastSLAM in real environments show that the algorithm can generate accurate maps [19, 47–49]. There are also improvements suggested by [50–52].

In the next section, a Graph-based SLAM approach is discussed.

2.4.3 GraphSLAM

Graph-based SLAM was first formulated by [53, 54], but the first working solution was introduced by [55]. The method draws its intuition from the graphical representation of SLAM. Unlike EKF-SLAM and FastSLAM, Graph-SLAM models the posterior as a graph where the pose and landmarks are nodes, and motions and measurements are edges between nodes. Edges are viewed as *soft constraints* (which will be discussed in detail later in the section) between nodes. Motion and measurements are subject to noise; therefore, we expect the constraints to be contradictory, represented by a *sparse graph*. In, a *sparse graph* each node is only connected to a small number of other nodes, furthermore the number of constraints scale linearity with time, k, in the number of nodes in the graph. Figure 2.3, describes the process of adding an arc between nodes and edges to the constraint matrix [22].

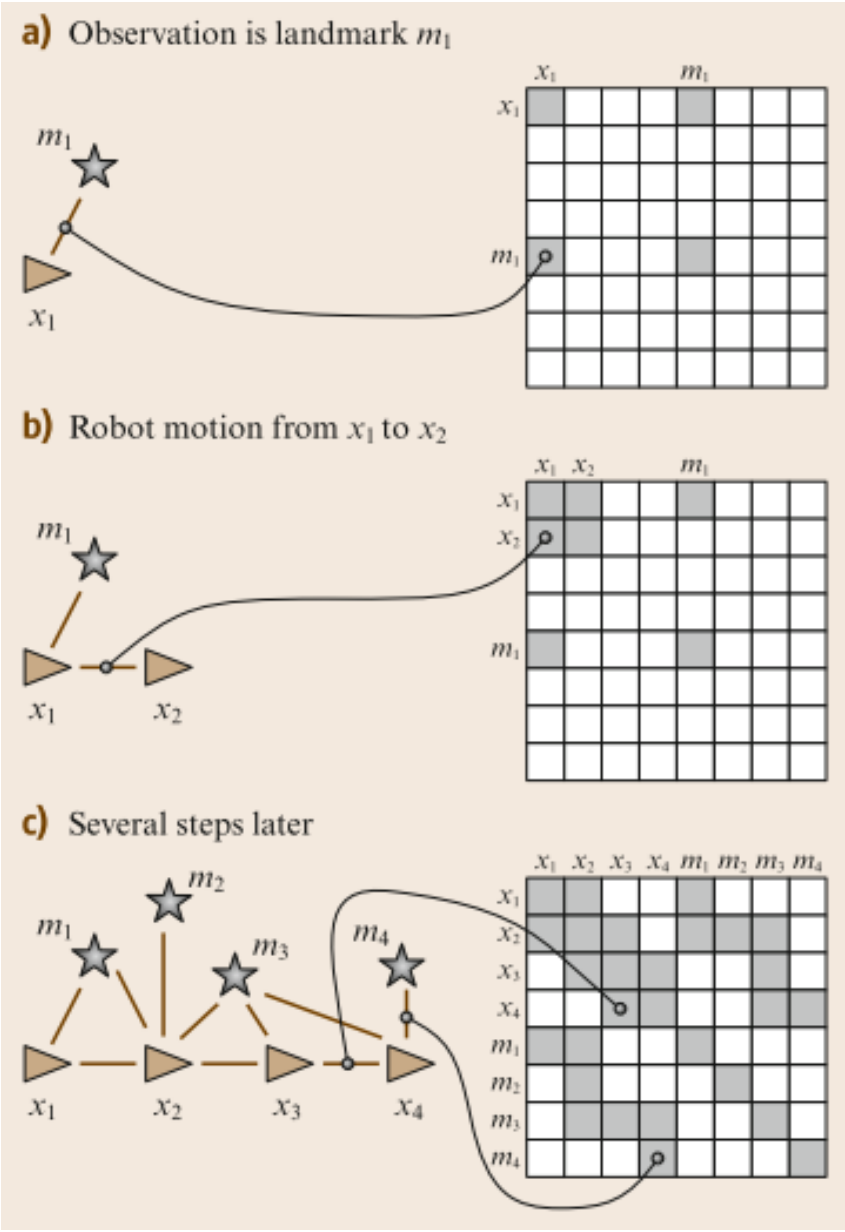


Figure 2.3: Graphical representation of the SLAM process, where the left shows the graph and the right the constraint matrix. In (a), when the robot identifies a landmark $\mathbf{m}_1$ at time $k = 1$, an arc is added to the graph between $\mathbf{x}_1$ and $\mathbf{m}_1$, followed by adding the edges to the constraint matrix. In (b), when the robot moves from $\mathbf{x}_1$ to $\mathbf{x}_2$ an arc is added between the two robot locations. Subsequently, the application at (a) and (b) continue as the robot moved through the environment as shown in (c). Taken from [2]

.

The graphical construction of the graph is illustrated by Figure 2.3. At time $k = 1$ the robot senses landmark $\mathbf{m}_1$, an arc is then added to the graph between $\mathbf{x}_1$ and $\mathbf{m}_1$, followed by adding the edges to the constraint matrix. Then, at time $k = 2$ the robot moves from $\mathbf{x}_1$ to $\mathbf{x}_2$ an arc is then added between the two robot locations. Subsequently, the application of steps at $k = 1$ and $k = 1$ continue as the robot moved through the environment, as shown in (c).

Golfarelli et al., thinks of graph as a spring-mass model, where the SLAM solution can be equated to computing the state of minimal energy of the model [56]. Accordingly, the SLAM solution can be represented by the log-posterior of the SLAM problem:

$$\log P(\mathbf{X}_{0:k}, \mathbf{m}|\mathbf{z}_{1:k}, \mathbf{u}_{1:k}) = \mathbf{const} + \sum_k \log P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) + \sum_k \log P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}). \quad (2.13)$$

Here, **const** is the anchoring constraint, which anchors the absolute coordinates of the map to the initial position of the robot. Constraint of the form, $\sum_k logP(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)$, represents one motion of the robot and corresponds to an arc in the graph; and constraint of the form, $\sum_k logP(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m})$, represents one sensor measurement which also corresponds to an arc in the graph. The Graph SLAM problem is then to find the model of the equation:

$$[\mathbf{X}_k^*, \mathbf{m}^*] = \underset{\mathbf{X}_k, \mathbf{m}}{\operatorname{argmax}} \log P(\mathbf{X}_{0:k}, \mathbf{m}|\mathbf{z}_{1:k}, \mathbf{u}_{1:k}). \quad (2.14)$$

The sum of all the constrains results in a nonlinear least-squares problem, which is entirely derived in [22] to give the quadratic form of Equation 2.13:

$$\begin{aligned} \log P(\mathbf{X}_{0:k}, \mathbf{m}|\mathbf{z}_{1:k}, \mathbf{u}_{1:k}) = \mathbf{const} \\ + \sum_k [\mathbf{x}_k - g(\mathbf{x}_{k-1}, \mathbf{u}_k)]^T \mathbf{R}_k^{-1} [\mathbf{x}_k - g(\mathbf{x}_{k-1}, \mathbf{u}_k)] \\ + \sum_k [\mathbf{z}_k - h(\mathbf{x}_k, \mathbf{m})]^T \mathbf{Q}_k^{-1} [\mathbf{z}_k - h(\mathbf{x}_k, \mathbf{m})]. \end{aligned} \quad (2.15)$$

Here, h is the measurement function and $\mathbf{Q}_k$ is the measurement noise covariance at a time, k. Similarly, g is the kinetic motion model of the robot and $\mathbf{R}_k$ is the covariance of the motion noise at a time, k. This equation is referred to as a *sparse function* graph, because the number of constraints in the graph scale linearly with the time elapsed, k. Several different optimisation techniques can be applied to the *sparse function*, but the common choices include gradient descent, and conjugate gradient [22]. The optimisation techniques focus on linearising the functions g and h, where the *sparse function* 2.15 then becomes quadratic as mentioned above.

Suppose that landmarks $\mathbf{m}_i$ and $\mathbf{m}_j$ are in the map, and corresponded to the same landmark. Either we remove $\mathbf{m}_i$ from the graph and attach all adjacent arcs to $\mathbf{m}_j$, or we can add a *soft correspondence constraint* of the form:

$$[\mathbf{m}_j - \mathbf{m}_i]^T \mathbf{\Gamma} [\mathbf{m}_j - \mathbf{m}_i]. \quad (2.16)$$

Where, $\mathbf{\Gamma}$ is a two by two diagonal matrix whose coefficients determine the penalty for not assigning exact locations to two landmarks [57], enabling GraphSLAM to handle the data association problem. GraphSLAM further performs data association by calculating the probability that two features have identical world coordinates. Since GraphSLAM methods are mostly offline as they optimise for the entire robot path, in a context where

the robot path is long, the optimisation can be cumbersome; the optimisation can then be done for any pair of features at any time.

Therefore, to practically implement GraphSLAM to reduce computation and avoid false data association, authors tend to extract local maps and match multiple features at a time [55]. Methods such as [58–61] are considered to be online and they imitate information filter methods like [61–65]. However, these methods are considered to be hybrid approaches.

2.4.4 Hector SLAM

Hector SLAM was developed for *Urban Search and Rescue* (USAR) and enables accurate localisation while keeping computational costs low [16]. Distance measurements come from accurate high-resolution and high-frequency range measurement. Hector SLAM relies on fast scanning rates that modern Light Detection and Ranging (LIDAR) equipment provide. Even though the USAR environment is typically unstructured, where robots can experience roll and pitch, the algorithm supports only full 3D motion estimation within 2D occupancy grid map representation. The algorithm uses LIDAR to implement a scan matching for (2D) SLAM and IMU for 3D attitude estimation system. Loop closure is not required to build a good map as no odometry data is needed.

Algorithm 2.3: Hector SLAM Algorithm

Data: z_k:laser scan at time k,i_k:IMU data at time k
Output: m_k^l:L-Level map at time k

for $l \leftarrow 1$ *to* L **do**
  $x_0^l \leftarrow$ initPose();
for $l \leftarrow 1$ *to* L **do**
  $pc_k \leftarrow$ 3dTransformLaser(z_k, i_k) ;
  $z_k^* \leftarrow filterPointCloud_{z-coord}(pc_k)$;
  for $t \leftarrow 1 tot_{end}$ **do**
    if $l = 1$ **then**
      $\hat{x}_k^l \leftarrow initPose(x_{k-1}^L)$;
    else
      $\hat{x}_k^l \leftarrow initPose(x_k^{l-1})$;
    $g_{k-1}^l \leftarrow calcOccupancyGradient(m_{k-1}^l)$;
    repeat
      $\hat{x}_k^l \leftarrow gaussNewton(g_{k-1}^l, \hat{x}_k^l, z_k^*)$;
    until $OptimisationCriteriaMet(x_k^l \leftarrow \hat{x}_k^l)$;

The high level algorithm represented in algorithm 14, starts by writing the first LIDAR scan to the occupancy grid map. The following scans are matched with the map in-order to estimate a rigid transformation (rotation and translation). The full 3D state is represented by $\mathbf{x} = (\mathbf{\Omega}^T \mathbf{p}^T \mathbf{v}^T)$, where $\mathbf{\Omega} = (\phi, \theta, \psi)^T$ are Euler angles, and $\mathbf{p} = (p_x, p_y, p_z)^T$ and $\mathbf{v} = (v_x, v_y, v_z)^T$ are position and velocity vectors. The inertial measures which represent the input vector $\mathbf{u} = (\boldsymbol{\omega}^T \mathbf{a}^T)^T$, where $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)^T$ and $\mathbf{a} = (a_x, a_y, a_z)^T$ are body-fixed angular rates and accelerations. The LIDAR scan is then written to the

map depending on criterion (minimum displacement in rotation and translation). This procedure is called scan matching, presented initially in [66].

Scan matching is the process of aligning LIDAR scans with each other in a map. Modern LIDAR scanners offer distance measurement with low noise at high scan rates, and hence in many cases, the LIDAR scanner is more precise and accurate than the odometry data available [16]. In Hector SLAM, the alignment of LIDAR scan endpoints with current map, are based on a Gauss-Newton approach to optimises the rigid transformation $\boldsymbol{\xi} = (p_x, p_y, \phi)^T$.

$$\boldsymbol{\xi}^* = \underset{\boldsymbol{\xi}}{\operatorname{argmin}} \sum_{i=1}^{n} [1 - M(\mathbf{S}_i(\boldsymbol{\xi}))]^2, \tag{2.17}$$

where the function $M(\mathbf{S}_i(\boldsymbol{\xi}))$ returns the map value at coordinates given by $\mathbf{S}_i(\ \boldsymbol{\xi})$. $\mathbf{S}_i(\boldsymbol{\xi})$ is the world coordinates of the scan endpoint, $\mathbf{s}_i = (s_{i,x}, s_{i,y})$, of a measurement at i. In this approach there is no need for data association between LIDAR scan endpoints or an exhaustive current pose search, because as scans get aligned with the current map, the match is implicitly performed with all preceding scans. The coordinates are a function of $\boldsymbol{\xi}$ which is the pose of the robot in the world coordinates:

$$\mathbf{S}_i(\boldsymbol{\xi}) = \begin{pmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{pmatrix} (s_{i,x}, s_{i,y})^T + (p_x, p_y)^T. \tag{2.18}$$

Given some starting position for $\boldsymbol{\xi}$, we want to estimate $\Delta\boldsymbol{\xi}$ which optimises the error:

$$\sum_{i=1}^{n} [1 - M(\mathbf{S}_i(\boldsymbol{\xi} + \Delta\boldsymbol{\xi}))]^2 \to 0, \tag{2.19}$$

Where, solving for $\Delta\boldsymbol{\xi}$ yields the Gauss-Newton equation for minimising 2.17. The resulting equation is given by 2.20, the derivation can be found in [16]:

$$\Delta\boldsymbol{\xi} = \mathbf{H}^{-1} \sum_{i=1}^{n} [\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\delta(\mathbf{S}_i(\boldsymbol{\xi})}{\delta\boldsymbol{\xi}}]^T [1 - M(\mathbf{S}_i(\boldsymbol{\xi}))]. \tag{2.20}$$

Where, $\mathbf{H}$ is a Hessian matrix (second-order partial derivatives of a scalar-valued function) given by

$$\mathbf{H} = [\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\delta(\mathbf{S}_i(\boldsymbol{\xi})}{\delta\boldsymbol{\xi}}]^T [\nabla M(\mathbf{S}_i(\boldsymbol{\xi})) \frac{\delta(\mathbf{S}_i(\boldsymbol{\xi})}{\delta\boldsymbol{\xi}}]. \tag{2.21}$$

The covariance of the Hessian matrix is then approximated to arrive at a covariance estimation of

$$\mathcal{R} = \sigma^2 . \mathbf{H}^{-1}, \tag{2.22}$$

where σ is a scaling factor dependent on the property of a LIDAR device. The full derivation can be found in [67].

The nature of an occupancy grid map where cells are either occupied, unoccupied or unknown limits the precision than can be achieved. The authors of Hector SLAM employ bi-linear filtering for both estimating occupancy probabilities and derivatives. Given a continuous map coordinate P_m, the occupancy value $M(P_m)$ as well as the gradient $\nabla M(P_m) = (\frac{\delta M}{\delta x}(P_m), \frac{\delta M}{\delta y}(P_m))$ can be estimated by using the four closest coordinates $(P_{00}, P_{01}, P_{10}, P_{11})$, see Figure 2.4.

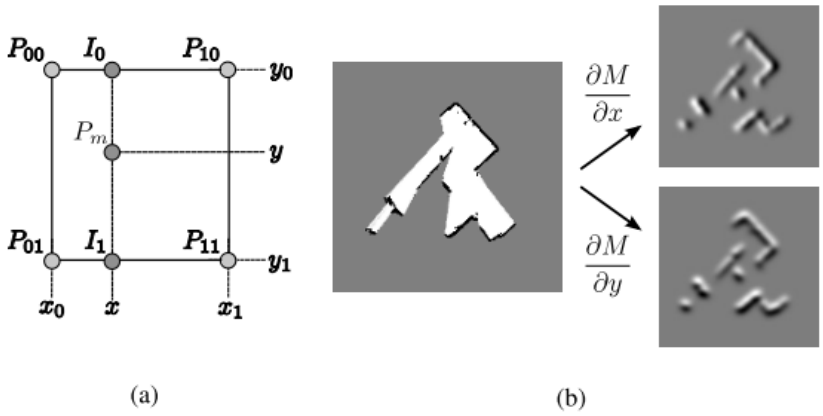


Figure 2.4: The figure illustrates aspects of filtering used to obtain sub-grid cell accuracy. (a) Bi-linear filtering of the occupancy grid map. Point P_m is the point whose value shall be interpolated. (b) Occupancy grid map and spatial derivatives. Taken from [16].

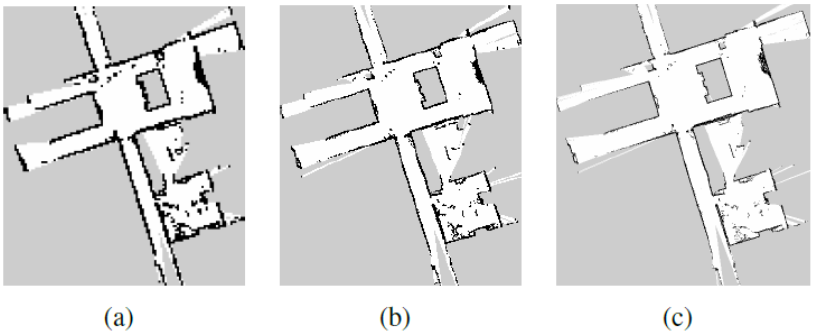


Figure 2.5: Multi-resolution representation of the occupancy grid map: (a): 20 cm occupancy grid cell length (b) 10 cm occupancy grid cell length (c) 5 cm occupancy grid cell length. Where grey cells are unknown, black are occupied cells and white are unoccupied cells.Taken from [16].

The scan matching is further optimised to reduce the effect of getting stuck in local minima as the procedure is based on gradient ascent. The optimisation is achieved by using *image pyramids*, where the map is stored at different resolutions. Scan matching is then done at all levels of the image pyramid starting from the coarsest to finest representation. Instead of implementing a computationally costly method such as Gaussian filtering or down-sampling, at the different layers LIDAR scans are written at the scale of each map. The method of writing LIDAR scans to maps, ensuring computational effectiveness and consistency, see Figure 2.5 [68].

To help the scan matching algorithm, odometry and other position data sources can estimate the scan match. This initial estimation can be done with a Kalman filter or other methods of sensor fusion.

Hector SLAM discussed in this section solves the SLAM problem sufficiently accurately for both ground, and water robots [16]. Hector SLAM has proven to require half the computation time as FastSLAM requires [69]. The approach relies on accurate scan to reduce the need for loop closure. However, when loops are significant, or when portions of the loop are separated, e.g. joining scans from multiple floors. Loop closure will be needed to obtain a good map. Even though Hector SLAM can work without odometry, odometry can determine the starting point for the optimisation process to align new scans with the map, to reduce computation time.

2.4.5 SLAM approach comparison

Even though all four paradigms discussed in previous sections perform in some instances, they have limitations associated with them, influencing the navigation block's performance, shown in Figure 2.1. The fundamental needed for navigation in robots is an algorithm that covers high-level specifications of humans' tasks into low-level descriptions of how to move. For example, imagine being giving two computer-aided-design(CAD) models: a house and a piano. An algorithm is then expected to move the piano from one room to another without colliding with any obstacles. This problem is referred to as the *Piano Mover's Problem* and defines the classic navigation problem [70].

Navigation's main objective is to move an object through the environment without collisions; the object and environment's current state needs to be accurately represented. Since the SLAM algorithm feeds the robot's current position and a local map to the navigation block, as shown in Table 2.1. The quality of these need to be considered when selecting a SLAM algorithm.

Shown in Table 2.1 are the limitations of the SLAM paradigms discussed in the previous sections. With a focus on data association, computational complexity and the ability to perform SLAM online. Hector-SLAM is selected in this work due to its robustness to loop closure, ability to perform SLAM online, and lack of dependency on odometry data, which is usually less accurate than modern LIDAR scanners. Furthermore, works done by [69] shows that Hector-SLAM has higher accuracy than FastSLAM, even though its computation time is twice fast as FastSLAM. The representation of local maps passed to the navigation block will be discussed in the next section.

Table 2.1: A comparison of the reviewed SLAM methods, highlighting their advantages and disadvantages with regards to loop closure events, computational complexity, online capabilities and map size handling.

Method	Advantages	Disadvantages
EKF-SLAM	• Works well with unique observations. • Can perform SLAM online.	• Computational complexity of $\mathcal{O}(M^2)$, where M is the map size. This complexity makes EKF computationally expensive as the map grows to update the covariance matrix as the map grows. • Susceptible to data association errors due to unimodal pose representation. Hence, loop closure is more frequent.
FastSLAM	• Overcomes the loop closure problem, with multimodal data association. • Copes well with nonlinear and non-Gaussian systems. • Can perform SLAM online.	• Suffers from particle deprivation. Which limits long-term exploration. • Becomes computationally expensive as the map size M grows and the number of particles Q increases, giving a computational complexity of $\mathcal{O}(Q \log M)$.
GraphSLAM	• Ideal for building large-scale maps. • Overcomes the loop closure problem.	• Computationally expensive if not optimised. • Can not perform SLAM online effectively.
Hector SLAM	• Does not require loop closure. • Hector SLAM depends on modern LIDAR scanners rather than odometry data which is less accurate. • Can perform SLAM online. • Hector SLAM is two times fast computationally compared to FastSLAM [69].	• Initial guess of the pose may take longer to converge if odometry data are not used to initialise the scan matcher algorithm.

2.5 Map representation

The SLAM block discussed in the previous section produces both a local map and current robot pose in the map. The representation of the map depends on how the map is going to be used. For example, in the *Piano Mover's Problem* discussed in the previous section, a piano requires moving from room to room. In this case, our assumptions will be a three-dimensional environment with room labels. In other examples, robots may need all objects in the environment labelled. Therefore, an effective map representation is required to match the scenario. In the case of this thesis, the map representation needs to be compatible with merging maps generated from multiple robots to form a global map of the environment.

There are various types of map representations, varying from geometrical to logical maps which have been developed to date; the following factors characterise the complexity of mapping problem [22]:

- **Size**: If the environment size is larger than the relative perceptual range of the robot, mapping becomes more difficult. This makes it more challenging to compute the full posteriors over maps.
- **Noise perception and actuation**: Noise on the sensors and actuators of the robot decreases the accuracy, as the errors accumulate.
- **Perceptual ambiguity**: If different locations look-alike more frequently, it is more difficult to establish a correspondence between the locations at different time points.
- **Loop Closure**: With odometry error, it is easier to correct if the robot travels up and down a corridor. The difficulty comes when a robot re-enters a location from a different path. This same problem was a point of discussion in the previous sections.

2.5.1 Occupancy grid maps

Occupancy grid maps were first introduced in the 1980s and are one of the most common map representations [17]. The mapping approach is represented by discrete cells. Where the posterior probability is calculated to determine if a cell is occupied or not, in the case of range sensors, cells will be empty if rays are unobstructed and rays that have an endpoint will be observed as occupied. Occupancy grid maps also account for unknown space, where the area of the map has not been explored as yet. Figure 2.6 is an example of an occupancy grid map created using a laser range finder. Black represents occupied cells, white represents empty cells, and grey represents unexplored territory.

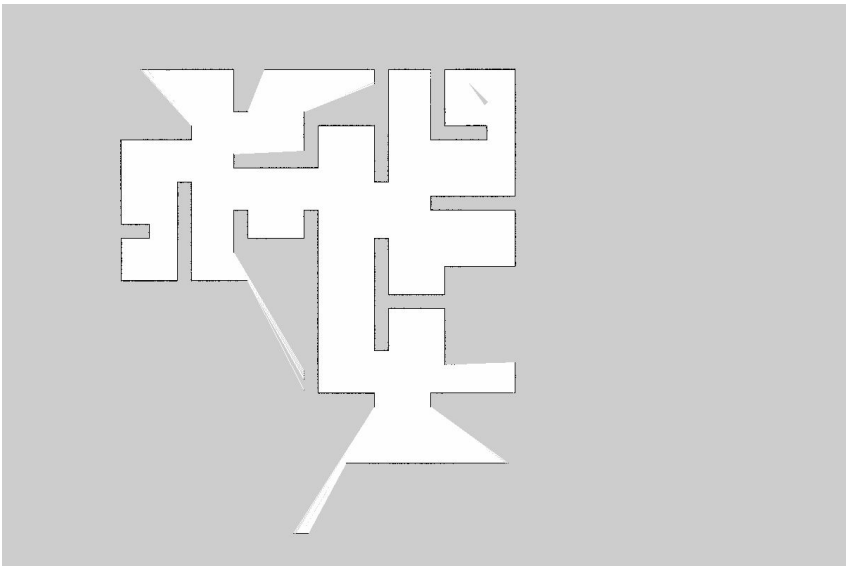


Figure 2.6: An example of an occupancy grid map, which was produced in simulation using Gazebo and ROS (explanation of how the data were acquired is explained in further chapters). Black represents occupied cells, white represents empty cells, and grey represents unexplored territory.

During an exploration, new sensor observations are used to update the grid cells according to the *Binary Bayes Filter*. The belief of a particular cell $\mathbf{m}_i$ is given by:

$$bel_k(\mathbf{m}_i) = P(\mathbf{m}_i|\mathbf{z}_{1:k}, \mathbf{x}_{1:k}), \tag{2.23}$$

where ,$\mathbf{m}$, represents the entire map, $\mathbf{m}_i$, is an individual cell in the map, $\mathbf{z}_{1:k}$, is a set of measurements and $\mathbf{x}_{1:k}$ is the pose of the robot at time ,k. Therefore the belief for the entire map ,$\mathbf{m}$, is given by:

$$bel_k(\mathbf{m}) = \prod_i P(\mathbf{m}_i|\mathbf{z}_{1:k}, \mathbf{x}_{1:k}). \tag{2.24}$$

A common approach is to represent the belief using *log-odds* [71]:

$$l_{k,i} = \log \frac{P(\mathbf{m}_i|\mathbf{z}_{1:k}, \mathbf{x}_{1:k})}{1 - P(\mathbf{m}_i|\mathbf{z}_{1:k}, \mathbf{x}_{1:k})}, \tag{2.25}$$

the *log-odds* representation is used to avoid numerical instabilities for probabilities near zero or one. The probabilities can be recovered using:

$$P(\mathbf{m}_i|\mathbf{z}_{1:k}, \mathbf{x}_{1:k}) = 1 - \frac{1}{1 + \exp l_{k,i}}. \tag{2.26}$$

Finally combining the *log-odds* beliefs can be represented as:

$$l(\mathbf{m}_i|\mathbf{z}_{1:k}, \mathbf{x}_{1:k}) = l(\mathbf{m}_i|\mathbf{z}_{1:k-1}, \mathbf{x}_{1:k-1}) + l(\mathbf{m}_i|\mathbf{z}_k, \mathbf{x}_k). \tag{2.27}$$

The advantage of these types of maps is that they can handle the dynamic environment, do not rely on predefined features, offer constant-time access to grid cells, and represent unobserved areas [6, 17]. However, when dealing with dynamic environments, the

drawback is that unlearning that a cell is free requires as much observation as the robot received to declare the area occupied [17].

These maps also have disadvantages which mainly pertain to high memory requirements (when working in higher dimensions). Therefore, even though 3D maps can be represented using this representation, the implementation will have high memory costs [6].

2.5.2 Feature (also known as landmark) maps

When environments have distinguishable features such as a pillar or tree, for example, landmark-based map representations are typically used. Signatures of the features are usually used to occupy the position of the feature. This representation is more efficient of localisation as the landmark features are initially defined, but feature extraction is the drawback to this implementation [17].

2.5.3 Topological Maps

Topological representation was firstly presented in work done by [72] and is one of the more popular mapping techniques. This technique represents the environment in a graph-like structure, focusing on the geometric structure of the environment. In topological mapping, unique, distinguishable places are defined as nodes, and the connections between the nodes are edges where the robot can travel [17].

Topological maps can also be derived from metric maps such as occupancy grids. An example of a Voronoi graph topological map presented in [73] produced a topological map from an occupancy grid map. A metric-free topological maps approach is introduced in [74], enabling minimalistic robot localisation.

Note that topological maps are well suited for high-level planning but need intensive map processing, and they are limited in way-point following [6].

2.5.4 Map representation comparison

In selecting the appropriate representation for this work, the representations need to be favourable for both map merging and navigation (see Table 2.2 for characteristics of the different approaches). Thus, occupancy grid map representation has been selected for their flexibility when representing the environment, and do not require as much computational ability as Feature and Topological maps [22]. In the case of map merging, they allow for image registration, and optimisation based approaches [6]. Occupancy grid maps can also be abstracted to topological maps for navigation [6].

Table 2.2: Map representation characteristics

Type of Map	Characteristics	Pros	Cons
Occupancy Grid Maps	The map is defined as a grid, where each cell is either occupied, free or unknown.	It is easy to build, probabilistic, suitable for 2D maps, easy to maintain, and suited for low-level planning.	Expensive on fine resolutions and 3D representations, exponential cost of memory with growing size.
Feature Maps	Maps are represented by landmarks such as trees and shapes.	Very useful for localisation.	Requires computationally expensive data association.
Topological Maps	Abstract spacial information is used to represent the maps.	It is suited for high-level planning.	Is limited to way-point.

2.6 SLAM extended to Multiple Robots

This section will briefly describe multiple-robot systems and multiple-robot SLAM. Generally, multiple robot SLAM algorithms are built on top of single robot SLAM systems; therefore, methods and map representation are based on the approaches mentioned previously. Jung suggests that the following advantages drive research in the field of multiple-robot SLAM [75]:

- **Space and time distribution** - Multiple robots can be in many places while performing different tasks simultaneously. This allows robots to, for example, explore a vast area in a shorter period.
- **Simpler is better** - In the multiple agent case, each robot can be more straightforward rather than have a single agent that is complex. This simplicity not only decreases the cost of each agent but increases reliability and robustness (see Figure 2.7).
- **Divide and conquer** - Certain problems are best solved when divided into smaller problems and distributed among the robots.

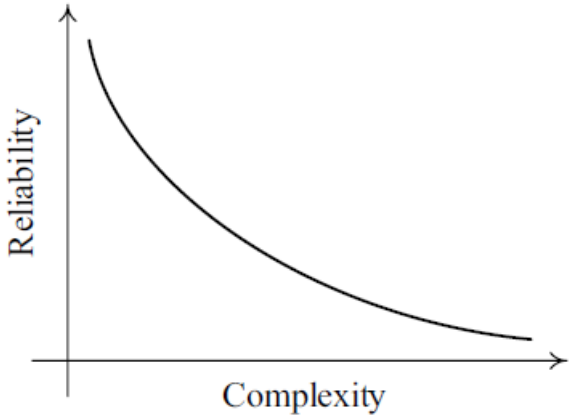


Figure 2.7: Relationship between reliability and complexity. The figure suggests that the simpler the system, the more reliable it is. Taken from [75].

However, having a multiple robots system has its associated problems. According to [6], these are the potential problems:

- **Relative poses** - Maps produced by individual autonomous robots are referred to as local maps, which are produced in the robot's frame of reference. To produce a global map, each robot needs to incorporate maps from all the robots. This can be difficult as it requires transformation matrices to relate the maps to each other. The relative pose also has its associated uncertainty which can be represented as a covariance matrix. This uncertainty can be caused by modelling, sensor noise, and linearisation effects [22].

- **Updating maps and poses** - After the relative transformation matrix has been found, the local maps need to be fused; thus, the poses also need to be updated. The process needs to consider the robot trajectory and new information added to the local maps. The processes also need to consider the type of map; for example, in feature-based maps, duplicate landmarks may need to be removed [2].

- **Line-of-sight observations** - Line-of-sight observation is when robots can see or detect each other. Even though robots might already possess the other robots' pose estimation, line-of-sight can improve the estimates. In some cases, this can be more reliable than the indirect approaches, but in some cases, robots might not have a sight of each other, especially during a mission such as search and rescue [76]. Furthermore, line-of-sight has been used in loop closure scenarios [10].

- **Loop closure** - This happens when a robot revisits a previously visited location but not very recently. As the issue is already complicated enough for single robot missions [32], extending this to multiple robots, the issue requires information from the robots in the system to be solved [77].

- **Complexity** - Especially when considering that usually robotic applications are real-time, space and time advantages mentioned above are essential to solving the multi-robot SLAM problem with minimum time and memory requirements. This complexity directly affects the scalability of the multi-robot SLAM operation.

- **Information sharing** - The quality of the information sharing among the robots mainly depends on the medium and environment which they are communicating [78]. For example, underwater robots face different issues to that of aerial robots. Different environments introduce different limitations to bandwidth, data rate and throughput.
- **Homogeneous vs. heterogeneous** - Having a diverse robot team can be a considerable advantage, especially in different terrain environments, this allows for a wholesome model of the environment. For example, a team may have robots on the ground, in the air and underwater [17]. The synchronisation between the different robots and sensors, is an essential part of multiple robot systems, firstly on each robot and all the robots. However, this advantage requires processing and integrating different types of robots [79], such as integrating different map representations.
- **Performance measure** - Measuring the accuracy of multiple robot systems, can be challenging considering the issues mentioned above and the lack of a model of the environment. This process is essential, primarily if the robot relies on SLAM to coordinate and perform tasks. Performance measures can be used to ensure the reliability of multiple-robot SLAM systems.

Furthermore, in Multiple Robot SLAM can either be centralised or decentralised [5]. In a **centralised system**, there is a predetermined robot in the team or an external agent which handles the computation of tasks. The assumption is that the central agent has a global map of the "World", which theoretically means that the central agent can optimally allocate tasks to the robot team. The robots would need to send partial maps continuously to the central agent, and the central agent will be responsible for aligning and merging the maps then the maps would be sent back to the robot team to produce the global map. This requires constant communication between the robot's in the team and the central agent. The central approach can either be implemented wholly (apart from the central agent other, agents have no control or autonomy) or partially (there is still a central agent, but other agents can also take some autonomous local decisions).

In a **decentralised system**, more than one robot is tasked with performing computations for tasks. However, it requires each robot to have adequate computational ability to act independently. In the map merging case, the robots may exchange maps with each other then align and merge them internally. The global map built by each robot may vary depending on which robots they have exchanged information.

Centralised systems allow for coherent and comprehensive solutions. However, depending on the size of the team in a centralised system it may become difficult or impractical to store all the information of the environment on one agent [5], a communication bottleneck may arise due to the massive communication between the central agent and other agents, high complexity leads to low reliability as per the following Figure 2.7. On the other hand, a decentralised system is tolerant to failure, reliable and robust, scalable, and tasks can be decomposed and run in parallel compared to the centralised approach [5].

In this section, the advantages and disadvantages of Multiple Robot SLAM are discussed. In the next section, related work on Multiple Robot SLAM is discussed concerning these advantages and disadvantages.

2.7 Related work on Multiple Robot SLAM

In the previous section, three key advantages (space and time distribution; simplicity; and divide and conquer) and eight fundamental problems (relative poses of robots; uncertainty of the relative poses; updating maps and poses; line-of-sight observations; closing loops, complexity communications; team diversity; synchronisation and performance measures) of multiple-robot SLAM were discussed, which are adapted from [6]. Furthermore, the centralised vs. decentralised multiple-robot SLAM systems are discussed. The following section will explore multiple-robot SLAM solutions.

Several researchers proposed the Multi-Robot SLAM problem, with work dating back to the early 2000s such as [80]. Nettleton et al. applied the information formulation to the multiple-robot SLAM problem [80]; the work shows that the Extended Information Filter (EIF) is more suitable than EKF multiple-robot SLAM since the information has the additivity attribute. However, the authors noted that data association and communication between robots would need to be further researched.

Later [81] improved on the work done by [80], by proposing a multiple-robot SLAM with Sparse Extended Information Filers (SEIF). An algorithm that enables robot teams to build joint maps where the relative starting position is unknown and landmarks are ambiguous. The alignment of local maps to generate a global map is achieved by a tree-based algorithm which searches for local landmarks arrangements combined with a hill-climbing method to maximise the overall likelihood.

Fenwick et al. propose an EKF based multiple-robot SLAM approaches and explore merging sensor and navigation information from multiple robots [82]. The method is based on stochastic estimation and uses a feature-based map representation, where a theoretical convergence theorem is proved for the first time. This approach quantifies gains of using a team of robots, therefore determining the number of robots required in a team to perform tasks.

Another EKF based multiple-robot SLAM technique is proposed by [83], where a team of robots (both stationary and dynamic) directly observe each other to reduce the dead reckoning errors. Furthermore, the algorithm is designed to cope with hard-to-sense reflectance properties.

Later, [84] proposed an EKF based multiple-robot SLAM technique, which aimed to solve the with the aid of robots rendezvousing to consolidate the relative position. In the method, robots do not require knowledge of their initial relative poses to merge maps to form a joint map. Furthermore, the method optimises the map merging and not the map overlap.

In [85], a multiple-robot SLAM method using Square Root Information Smoothing is presented. This method allows a team of robots to built a global without initial information of the robots relative poses, during a line-of-sight observation. In contrast to EKF approaches, the smoothing is better equipped to deal with the nonlinear process and measurement models. The method joins the maps from different robots and recovers each robot's complete trajectory in the map merging process.

In 2001 Thrun [86] proposed a multiple-robot SLAM solution based on a particle filter, where the fast maximum likelihood map grows with a Monte Carlo localiser that uses a particle representation. This yields an online algorithm that can cope with significant odometric errors, typically found when dealing with loop closures. However, the method has the following limitations: firstly, it cannot handle nested cycles in the map, secondly does not deal well with lower quality range sensors such as sonar sensors, and thirdly robots need to know their initial relative poses to each other.

Later in 2006, the work was extended by [77], where a multiple-robot SLAM using Rao–Blackwellised particle filter was proposed. The robots use line-of-sight to estimate relative position, using cameras and unique markers mounted on them. At the first meeting point, a Rao–Blackwellised particle filter (RBPF) is applied to the data in reverse time sequence to fuse all the data from the starting point, to determine relative position. Furthermore, the maps and poses are updated in real-time, and the decentralised nature of the system increases reliability, robustness and the scalability of the approach. However, it cannot be applied if the robots cannot see each other, and are unable to determine their relative positions.

Carlone [78] similarly explores multiple-robot SLAM using the RBPF extending on [77]'s work relative initial position of the robots are unknown. In this work, the main focus is information sharing among the robots, with limited communication between them. The information shared when the robots are in-range includes all the odometry and corresponding laser range-finder data. Line-of-sight observation is used then to compute the relative pose with an incorporated uncertainty.

In [87], a heterogeneous multiple-robot SLAM algorithm using sub-map matching is developed on a team of robots comprised of one ground robot and a helicopter. A global graph maintains the relative relationships between the local sub-maps. Monocular cameras are used as the primary exteroceptive sensors in this work. 3D points are estimated using an inverse-depth parametrisation, 3D line segments are estimated by adding endpoints to extend anchored Plücker coordinates (a coordinate system that assigns six homogeneous coordinates to each line in projective 3D-space [88]).

In [40], robustness to connectivity loss for multiple-robot SLAM method is proposed. In the framework, no central authority is required while the robot team contributes to the global map. However, this method requires systematic coordination among the robots in order to add to the global map. Hence, if a robot leaves the system unannounced, it can potentially make the map inconsistent.

Work done by [32] proposes a Hector mapping architecture for multi-robot. In their work (which they implemented in ROS), each robot shares its observations and poses a central note. The central node computes the global map, using the poses (including the initial poses) and observations from the robot team. Furthermore, parallelising optimisation steps and sequentially registering laser scans to the jointly built map are implemented to enable real-time running. Table 2.3 states the algorithms' main features in this section.

Table 2.3: Solutions to Multiple Robot SLAM problems

Problem	Solutions
Relative poses	[81], [84], [83], [78], [87], [40]
Updating maps and poses	[81], [82], [77], [87], [85], [40]
Line-of-Sight observations	[83], [84], [77], [85]
Loop closure	[83], [86]
Complexity	[77], [32], [85]
Information sharing	[78], [40]
Homogeneous vs. Heterogeneous	[87]
Performance measures	[82]

2.8 Summary

In this chapter, an overview of SLAM is explored, including SLAM formulation, single robot SLAM solutions, map representation, and multiple-robot SLAM extension. The multiple-robot SLAM literature, problems such as relative poses, updating maps and pose and homogeneous vs. heterogeneous are discussed in Section2.6, along with some solutions that tackle the problems. The work done in this chapter, also prompts the use of Hector SLAM as the SLAM algorithm each robot will be equipped; occupancy grid maps as the map representation; and a distributed architecture where robots communicate when they discover each other in the network. In the next chapter, merging maps from single independent robots are discussed, along with details on how maps are acquired, how robots share maps, and how they are aligned and merged.

Chapter 3

Map merging

This chapter discusses map merging for multiple robots; the discussion includes reviewing the literature and parameter tuning experiments. In multi-robot SLAM, two or more robots explore an environment independently and generate local partial maps. Merging of all the partial maps to form a global map, is functional to applications such as finding frontiers for all the robots in the environment [89]. Map merging can be simplified into two steps which involves finding alignments (such as rotation, resolution and translation) between maps, and then merging maps using information from the first step. The maps and other information about the robots are shared to find the alignment between the different maps in the steps mentioned. The focus, as mention in the previous chapter, is on merging 2D occupancy grid maps.

3.1 Problem statement

Consider a large and complex environment, where a team of robots is tasked to maps different parts of an environment at different grid map resolutions. The resolution is defined by a metric x per cell; in this work, we will be using meters/cell (m/cell), which is the size in meters of an individual grid map cell. The partially overlapping local grid maps generated from the task must be integrated into a global map that can be useful for robots to perform operations such as path planning and frontier finding.

3.1.1 Definitions

Definition 1: Let $\mathbf{m}$ be a map of N by M, where N, M are positive real numbers:

$$\mathbf{m} : [0, N] \times [0, M] \rightarrow \mathbb{R} \tag{3.1}$$

We can also denote a set of N by M maps as $\mathbf{I}_{N \times M}$

Definition 2: According to [90], the goal of pairwise map merging is to find a relative

transformation denoted by:

$$\mathbf{T}_{t_x,t_y,\theta} = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

, where $s \in \mathbb{R}$ is the scale factor, $\mathbf{R} \in \mathbb{R}^{2\times 2}$ represents a two-dimensional counterclockwise rotation matrix about origin point (x, y) of θ angle θ and $\mathbf{t} \in \mathbb{R}^2$ indicates a translation vector (t_x, t_y). More specifically, $\mathbf{R}$ and $\mathbf{t}$ can be denoted as follows:

$$\mathbf{R} = \begin{bmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{bmatrix}, \mathbf{t} = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{3.3}$$

Definition 3: Let $\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_k$ be k maps in $I_{N\times M}$

$$\omega(\mathbf{m}_1, \mathbf{m}_2) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} Eq(\mathbf{m}_1[i,j], \mathbf{m}_2[i,j]). \tag{3.4}$$

Here $Eq(a, b)$ is one when a=b and zero otherwise, and ω is an overlapping function that measures how much the two maps agree. In an ideal world, robots would cover the same area, where there exists a transformation which yields $\omega(\mathbf{m}_1, \mathbf{m}_2) = N \times M$. In the real world this is not the case, so the objective is to find the transformation which maximises overlap. Therefore, x, y, θ-map transformation, $\mathbf{T}_{t_x,t_y,\theta}$ is determined by maximising $\omega(\mathbf{m}_1, \mathbf{T}_{t_x,t_y,\theta}(m_2))$.

Suppose there are a couple of overlapping maps: the subject map $\mathbf{P}$ and the seed map $\mathbf{Q}$, which have different resolutions built by robots exploring two parts of the same environment. To merge the seed map $\mathbf{Q}$ into map $\mathbf{P}$, a transformation $\mathbf{T}_{t_x,t_y,\theta}(x, y)$ is applied to map $\mathbf{Q}$, to obtain new map $\mathbf{Q}'$ indicated as $\mathbf{Q}' = \mathbf{T}_{t_x,t_y,\theta}(x, y)\mathbf{Q}$. Once transformed map $\mathbf{P}$ and $\mathbf{Q}'$ are then merged to form a global map. The following section explores solutions to solving the map merging problem defined in this section.

3.2 Related work

Various solutions are presented in the literature for maps from a team of mobile robots. The following four cases, where most are discussed in [91] are used to study map merging in this section:

a. **Known initial configuration**: This case is the simplest as the initial pose of the robots is known, which simplifies the map alignment step. This assumption is a limited case as many times the initial poses are not known as described in [78]. In the case of Figure 3.1, for the initial poses to be known, the relative pose would be described by the distance between G_i and G_j.

b. **Rendezvous**: This case is where two robots meet at a point such as in Figure 3.2, where the relative pose is calculated using methods such as line-of-sight measurement.

The distance between the robots determines the accuracy at the meeting point and the method used to compute the relative distance. However, this adds a level of complexity to the system as either the robots need to coordinate a meeting point or use a method such as an image processing to recognise other robots [84].

c. **Relative position estimation**: In this case, a robot localises other robots in its local map. The critical element to this case is that there is no need for the robots to rendezvous or have knowledge of initial relative positions [92].

d. **Overlaps**: In this case, overlaps in the map are used to calculate the relative transformation between different maps such as in Figure 3.3. The main challenge is finding the maps' overlaps using features such as doors, junctions or corners. The optimal algorithm is one that can identify both false and true matches, in the least amount of time with the highest accuracy [93].

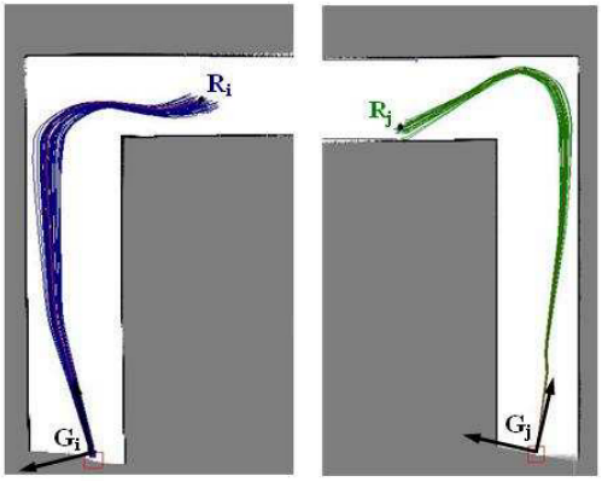


Figure 3.1: This figure shows an example of two robots with unknown initial configurations. Each robot estimates both trajectory and map hypotheses in its own reference frame. In the case where initial configuration is known, the relative distance between G_i and G_j is known. Taken from [78]

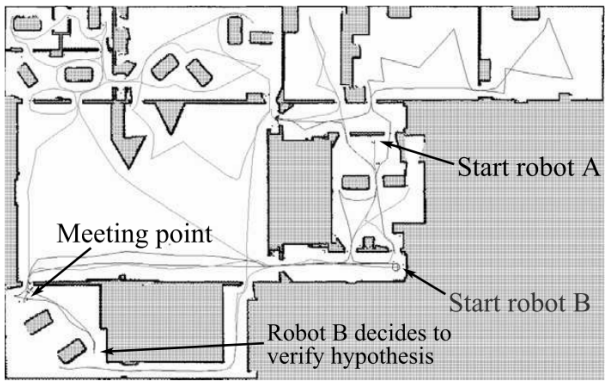


Figure 3.2: Example of a coordinated exploration of an unknown environment, from unknown locations. When then the robots meet at the meeting pint, they estimate and verify relative locations using a rendezvous approach. At the meeting point they share a map and their coordinate. Taken from [22].

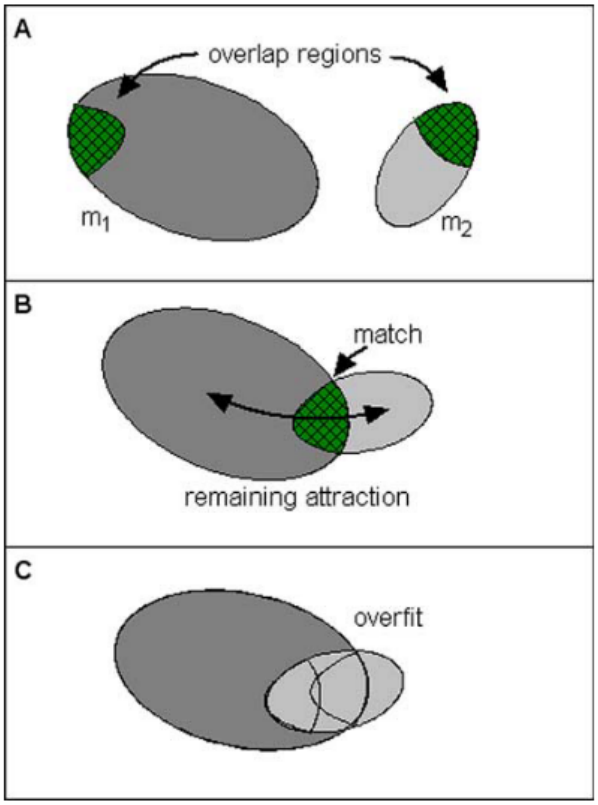


Figure 3.3: (A) describes the overlap region of the m_1 and m_2, where similarity is maximised. In (B) the identical regions are matched, where the unmatched regions remain unchanged. (C) is an example of over-fitting, where an area without a match is merged into another map. Taken from [93].

Early strategies of solving map merging with multiple robots such as [89] assume a known initial position of the robots. Although this method required minimal computational power, the method will fail if robots are unaware of each other's initial position. The more flexible approaches started presenting themselves in [94], where map merging was solved as an optimisation problem. The authors in [94], proposed a stochastic search approach to solve the problem. This method is an exhaustive approach based on adaptive random walks aiming to find suitable transformation to overlap the maps. Work was further done by [95] to improve the algorithm, and the improvement attempts to detect failures and guide the search to find the optimal solution. However, the exhaustive nature of the algorithm makes it computationally expensive to implement.

Xin Ma et al. later proposed a generic adaptive algorithm implemented without standard reference frames and relative pose information of robots [12]. The authors claim that their algorithm outperforms one presented by [94] because the generic adaptive algorithm can search the optimal match overlapping area more effectively. Avoiding premature convergence, low convergence speed and low stability. However, they did not present a merging of complex real-world maps, and the experimental results are of simple simulation maps.

Saeedi et al. propose an artificial intelligence-based approach to map merging [96]. The algorithm clusters the maps using neural-networks based on self-organised map (SOM); the cluster norms are used to find the relative transformation between the maps. Although

this reduces dimensionality and improves processing time, the downfall is that some information is lost in the process. Dinnissen et al. proposed another artificial intelligence-based novel approach using reinforcement learning; the model is trained to determine a policy capable of deciding the best merge [97]. This approach allows the robot to incur less error during merging compared to simply merging after alignment. However, the model is validated using a simulated dataset instead of a real-world dataset.

Blanco et al. proposed a novel map-matching algorithm from image registration [98]. In this method, the maps are represented both as points and occupancy grid maps. The grid maps are firstly matched then the corresponding point maps are matched to help remove false matches. The transformation is determined by removing high-frequency noise and using feature detectors such as Harris or Kanade Lucas Tomasi (KLT). Matches are then filtered using a customised RANSAC algorithm based on the sum of Gaussian distributions. Based on the experimental results, this method deals adequately with uncertainty and ambiguity when matching maps.

Alternatively, Hough transforms are a way of representing geometrical data within maps. Carpin proposes a Hough spectrum-based approach in [13], which was first introduced by [99]. The maps are transformed into the Hough space, and the resulting Hough images are then correlated to find the potential transformations (rotation/translation). The authors introduce an acceptance index which tells how well-aligned maps are for each transformation. This index has values between 0 and 1, where 1 is a correct transformation, and 0 is an incorrect transformation. However, this solution fails when there is not enough overlap between maps. Saeedi et al. further improved the approach by adapting the new cross-correlation operator allowing the algorithm to deal with less overlap in the maps [14].

Topology is a field of mathematics that studies the properties preserved through stretching, deforming, and twisting of an object [72]. Topologies can be used in mapping to abstract objects while ignoring details. Topological maps can use a graph to represent navigation possibilities through an environment, where vertices are “places” and edges represent paths between the“places”. Even though maps can lose information from this abstraction, it can improve communication and mutual perception. Topological approaches have applications in map merging such as [15, 100].

Huang et al. proposed a topological approach which uses both structural and geometrical attributes of the Voronoi graph, to determining best correspondence between maps with overlapping areas [100]. The topological map is built on the occupied space instead of using free space because the assumption is that robots will be able to recognise the area of the map that correspond to vertices. The limitations of this method are that the maps are not updated. Saeedi et al. proposed a probabilistic topological representation, the essential to add probabilistic information to generalised Voronoi diagram [15]. They refer to the method as probabilistic generalised Voronoi diagram (PGVD). The method is used to match parts of the map, which are more specific, therefore improving the reliability of finding relative transformation between maps and fusing them.

All the strategies mentioned above are only merging grid maps that have the same resolution. However, there is a need for representing maps with multiple resolutions for building large scale maps. Some areas of a large scale environment may include more details than others. Therefore for reasons of compactness, the map can be built at a fine resolution in

areas of more details, and at a coarse resolution in areas with fewer details [101]. In [102], map merging is approached as a unique image registration case. The solution considers non-common areas and an objective function based on the trimmed mean-square error (MSE) is designed. The objective function is then solved using a trimmed and scaling iterative closest point (TsICP) algorithm. The TsICP algorithm needs a good initial transformation to converge locally. Therefore the initial transformation is obtained by extracting scale-invariant feature transform (SIFT) features, and then the random sample consensus (RANSAC) algorithm is used to find geometrically consistent features matched. The results presented show that this algorithm is more accurate and robust compared to [98] and [13]'s approach.

Further sections will discuss the use of image registration as a solution to multi-resolution map merging.

3.3 Image registration approach

As mentioned in the review above, map merging can be view as an image registration problem. For a pair of occupancy grid maps, features are extracted, matched, and then aligned and merged to give a global grid map. This section will discuss the following topics of image registration: feature detection, matching, and alignment.

3.3.1 Feature detection

The process of abstracting an image is called feature detection. The abstraction is computed by deciding at every point of the image if there is a feature based on a set of rules. Features have no universal definition and depend on the application, and a feature is usually defined as a region of "interest" within an image. Features can be specific structures such as points, edges, corners or objects. Ideally, the detection of features should be robust to transforms (rotation, scale, illumination, noise and affine transformation) in an image, with distinctive features to be matched with high probability.

The main components of feature detection are:

- **Region of Interest:** This is a point or region which is expressive in appearances such as corner, edge and objects. This interest region is usually stable to: rotation, scale, illumination, noise and affine transformation.
- **Descriptor:** This is the local appearance around each region of interest, which is ideally described in a way that is invariant to: rotation, scale, illumination, noise and affine transformation. Typically each region of interest will have a unique descriptor.

Once features have been identified and described, they can be applied to image-stitching processes, where features from multiple partial images are matched to align and merge

the images to form a single picture. This section will further discuss three scale-invariant feature detection methods (SIFT, SURF and ORB).

Scale Invariant Feature Transform (SIFT) is a feature detector developed by Lowe in 2004 [103]. SIFT solves image rotation, affine transformations, intensity, and viewpoint change in matching features. The SIFT algorithm employs four basic steps:

1. **Estimate of scale-space extrema:** Some objects are only detectable at specific scales. For example, if zoomed into the eye's pupil, one might mistake it for a flower or other object, however, if zoomed out to the point of seeing their eyebrows one will realise that it is an eye pupil. SIFT recognises this attribute in nature and attempts to replicate the effect by using scale-space filtering. The scale-space of an image is represented by $L(x, y, \sigma)$, where (x, y) is a potentially key-point location and σ is the scaling parameter. In the scale-space filtering, Laplacian of Gaussian (LoG) is found for the image at various σ values so that we can find local maxima across various scales and the space of the image. This LoG is computationally costly; therefore, the Difference of Gaussian (DoG) is used instead. The DoG is obtained as the difference of Gaussian blurring of the image at different scales. The scale-space is divided into octaves, and the number of octaves is dependent on the original image size, see Figure 3.4.

2. **Finding key points:** Once key-point location candidates are localised from the previous step. They are refined by eliminating the low contrast points. A Taylor series expansion of the scale-space is used to get a more accurate extrema location, and if the intensity at this extrema is less than a set threshold, it is rejected.

3. **Orientation assignment:** Since we already know at which scale a key-point was detected, we have achieved scale in-variance. An orientation is assigned to key points to achieve invariance to image rotation, based on a local image gradient. A neighbourhood is taken around the key-point depending on the scale, and a gradient magnitude and direction is computed in this region.

4. **Key-point descriptor:** So far a location, scale and orientation has been assigned to each key-point identified. Next, a key point descriptor is generated to compute the local image descriptor for each key-point based on image gradient magnitude and orientation. Ensuring that the key-point is highly distinctive to changes such as viewpoint and illumination. A neighbourhood around 16×16 is taken around the key-point and then divided into sub-regions of 4×4. An eight bin orientation histogram is created around each sub-region, making 128 bin values available. These 128 bin values are represented as a feature vector to form the key-point descriptor for each key-point. To further improve the descriptor, the key-points rotation is subtracted from each orientation and achieved illumination independence, and large numbers are thresholded.

SIFT has been proven to be very useful in mainly object recognition applications; however, it requires large computational complexity [103, 104]. Several variants and extensions of SIFT improve the computational complexity [105–107].

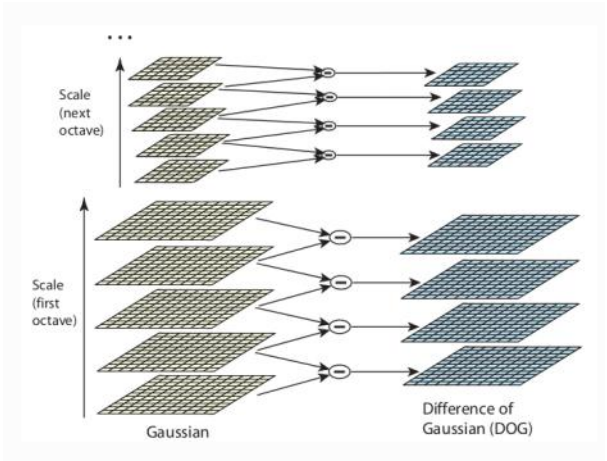


Figure 3.4: Shown on the left are scale-space images produced by convolving the initial image with Gaussians repeatedly for each octave. The difference-of-Gaussian (DoG) images on the right are produced by subtracting adjacent Gaussian images. The Gaussian image is down-sampled by two after each octave, and then the process is repeated. Taken from [103].

Speed up Robust Features (SURF) is the feature detector technique, which is an approximation of SIFT. The algorithm performs fast computation operations using box filters, thus enabling real-time applications such as object recognition [108]. SURF approximates the LoG with box filters, the advantage being that convolution with box filter can be easily calculated with the assistance of image integrals. Squares are used for approximation instead of Gaussian averaging because convolution with a square is faster than an image's integral. Furthermore, the estimation can be performed in parallel, the following Figure 3.5 shows such an approximation.

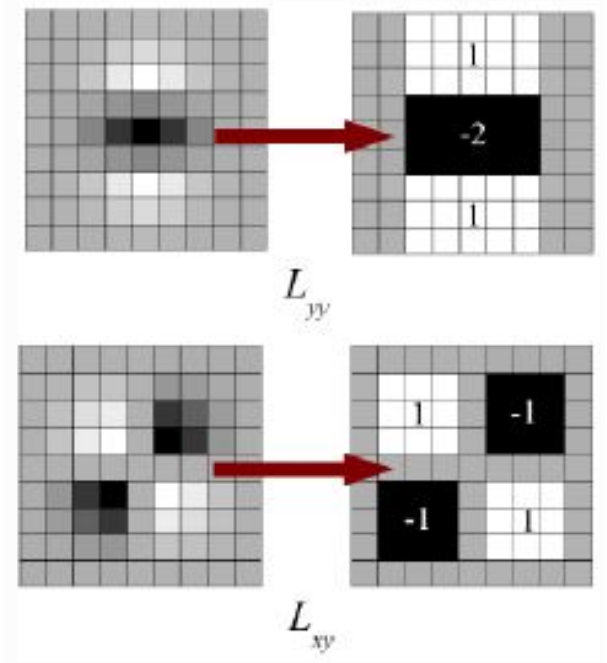


Figure 3.5: Approximation of Laplacian of Gaussian (LoG) with Box Filter, where the grey regions are equal to zero. Taken from [108].

SURF relies on the determinant of the Hessian matrix for both scale and location of key-points. To find the key-points' orientation, SURF uses wavelets responses in both horizontal and vertical directions for a neighbourhood of size 6s and then applying Gaussian weights. They are then plotted as shown in Figure 3.6, where the red arrow is the dominant orientation, computed by summing of all the responses within a sliding orientation window.

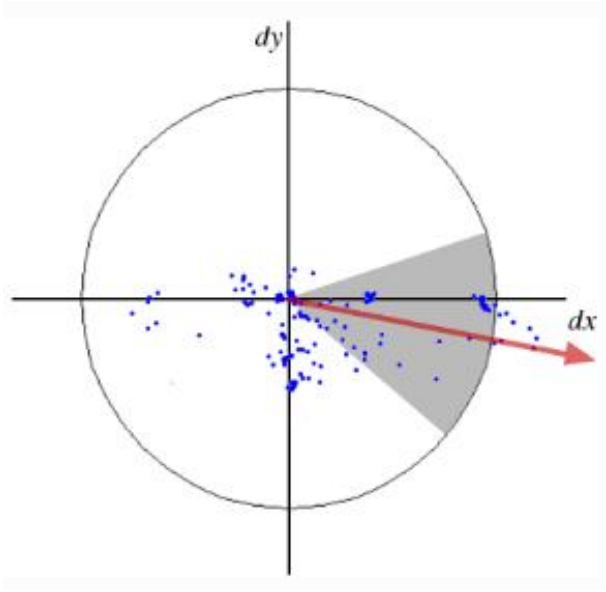


Figure 3.6: SURF uses wavelets responses in both horizontal and vertical directions then applying Gaussian weights, where the red arrow is the dominate orientation, computed by summing of all the responses within a sliding orientation window. Taken from [108].

Feature descriptor is achieved by again using wavelet responses in horizontal and vertical direction around a neighbourhood of size 20×20 around the key-point, which is then divided into sub-regions of size 4×4 and then for each sub-region the wavelet responses are taken and represented to get SURF feature descriptor. SURF allows faster matching by using the sign of the Laplacian which is already computed by the detection, the sign distinguishes bright blobs on a dark background, which in case of matching the features are compared if they have matching contrast.

Oriented FAST and Rotated BRIEF (ORB) was developed at OpenCV labs as an alternative to SIFT and SURF. ORB fuses the well-known Features from Accelerated and Segments Test (FAST) key-point detector and BRIEF descriptor with modification to enhance performance [109].

FAST key-points are identified by comparing the brightness of 16 pixels surrounding a specific pixel, p. The pixels are then sorted into three categories lighter than p, darker than p or similar to p. If more than eight of surrounding pixels are darker or brighter than p, then p is selected as a key-point. Therefore, FAST key-points determine edges in the image. FAST key-point is, however, not an orientation and scale-invariant.

Similarly to SIFT, ORB identifies features at multiple scales, by using pyramids and representing the image at multiple scales. Change in intensity around the key-point is

used to identify the orientation of the key-point. The intensity weighted centroid of the patch with located key-point (edge) at centroid is computed, to detect the change in intensity. The vector from the edge to centroid gives the orientation.

To compute the BRIEF descriptor, the algorithm takes all the key-points already found and converts them in binary feature vectors. In BRIEF, each key-point descriptor is defined by a 128 dimension vector, where each element is a floating point number (i.e. 4 bytes each) that produces a 512 byte sized vector for the feature. To prevent sensitivity to high-frequency noise, BRIEF begins by smoothing the image using a Gaussian kernel. Followed by selecting a pair of pixels in a defined neighbourhood of some height and width around the key-point. The first pixel is randomly selected from a Gaussian distribution centred around the key-point. Then the second on is randomly selected from a Gaussian distribution centred around the first pixel. If the first pixel is brighter than the second, a one is assigned else a zero. For a 128 dimension vector, this process is repeated 128 times, for each key-point. Given that BRIEF is not orientation invariant, ORB employs Rotation-aware BRIEF, by "steering" BRIEF descriptor in the key-point direction.

SIFT, SURF and ORB describe a feature with a key-point and a descriptor. In the next section, matching these features from images with partial overlap is discussed, followed by merging the images.

3.3.2 Image matching

The features detected in the map images need to be matched with high accuracy to get the best transformation (scaling, translation and rotation). The features are matched from one map to another using either a Brute-Force matcher or Fast Library for Approximate Nearest Neighbors (FLANN) Matcher provided by OpenCV. Brute-Force matcher uses the distance between features of the map to another and finds the closest one. Different distance measurements can be used with this matcher, depending on the feature detector. It is recommended to use the Euclidean distance for SIFT and SURF, and Hamming distance binary string based descriptors such as ORB, BRIEF and BRISK [1]. FLANN stands for Fast Library for Approximate Nearest Neighbours, and it is optimised for large datasets and high dimensional features. However, FLANN Matcher is not necessary for this application.

3.3.3 Matching features

Brute-Force Matcher is used to matching features between two different images. Given that features (key-points and descriptors) have been identified, Brute-Force Matcher takes descriptors from each image and matches them using a distance calculation to determine the closest feature.

Different distance calculations can be using, euclidean distance is recommended for SIFT and SURF, and hamming distance is recommended for ORB [110].

[1] https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

Euclidean distance is defined as the absolute value distance between two coordinates and is described as:

$$d(p, q) = \sqrt{(p - q)^2} \tag{3.5}$$

Where, p, q are two coordinates on a line.

On the other hand, Hamming distance is a metric for comparing binary data strings. As discussed in the previous section, ORB produces binary descriptors, which are compared with an XOR operation. After XORing the two values, the total number of 1s is counted, which is the Hamming distance. The smaller the number of 1s, the smaller the Hamming distance.

3.3.4 Map alignment

Even though maps may have a large number of matches between them, there will be outliers present. These outliers can cause some possible errors while aligning, which may affect the result.

To remove outliers, algorithms such as Random sample consensus (RANSAC) or LEAST_MEDIAN (LMEDS) are used so that good matches which provide correct estimation can be used. These good matches are called inliers, and remaining outliers are removed. OpenCV's **estimateAffinePartial2D()** has been used to remove outliers and estimate the transformation matrix $T_{t_x,t_y,\theta}$, defined by Equation 3.2 earlier.

This method (**estimateAffinePartial2D()**) has the following parameters:

- **method**: Robust method used to compute transformation. The following methods are possible, RANSAC and LMEDS.
- **ransacReprojThreshold**: Maximum re-projection error in the RANSAC algorithm to consider a point as an inlier.
- **maxIters**: The maximum number of robust method iterations.
- **confidence**: Confidence level, between 0 and 1, for the estimated transformation.

RANSAC was used, RANSAC is a robust estimation iterative procedure that uses a minimal set of randomly sampled correspondences to estimate image transformation parameters, and finds a solution that has the best consensus with the data [111]. The parameter value of confidence has been selected to be 0.99 to achieve high accuracy and $ransacReprojThreshold$, and $maxIters$ will be discussed later.

The transformation T from Equation 3.2, can now be used to align the one image to match the other. An affine transformation fixed to rotation, translation and scaling transformations is used. Affine transformations can also do the following transformation: identity, reflection and shear but these do not apply to this thesis.

3.3.5 Method selection

Paper [112], investigates the sensitivity of SIFT, SURF, and ORB against intensity, rotation, scaling, shearing, fish-eye distortion, and noise. The investigation is general, and in the case of the work done in this thesis, further investigation is required for occupancy grid maps. This section investigates SIFT, SURF, and ORB's sensitivity against each rotation, scaling, translation, and noise, not includes in shearing and fish-eye distortion, which are not of interest in 2D occupancy grid maps, due to how they are generated. Open Source Computer Vision Library (OpenCV) [2], which is an open-source BSD-licensed library that includes several hundred computer vision algorithms, is used for the image registration in this thesis.

In Figure 3.7 are the maps used to test the robustness of the feature detection methods. These maps were produced in a real-world environment at the University of Cape Town due to work done in this thesis[3][4]. The maps were produced using Hector-SLAM which each robot will be equipped with, and these maps are partial maps of the same area. The data used is later discussed in Chapter 5.

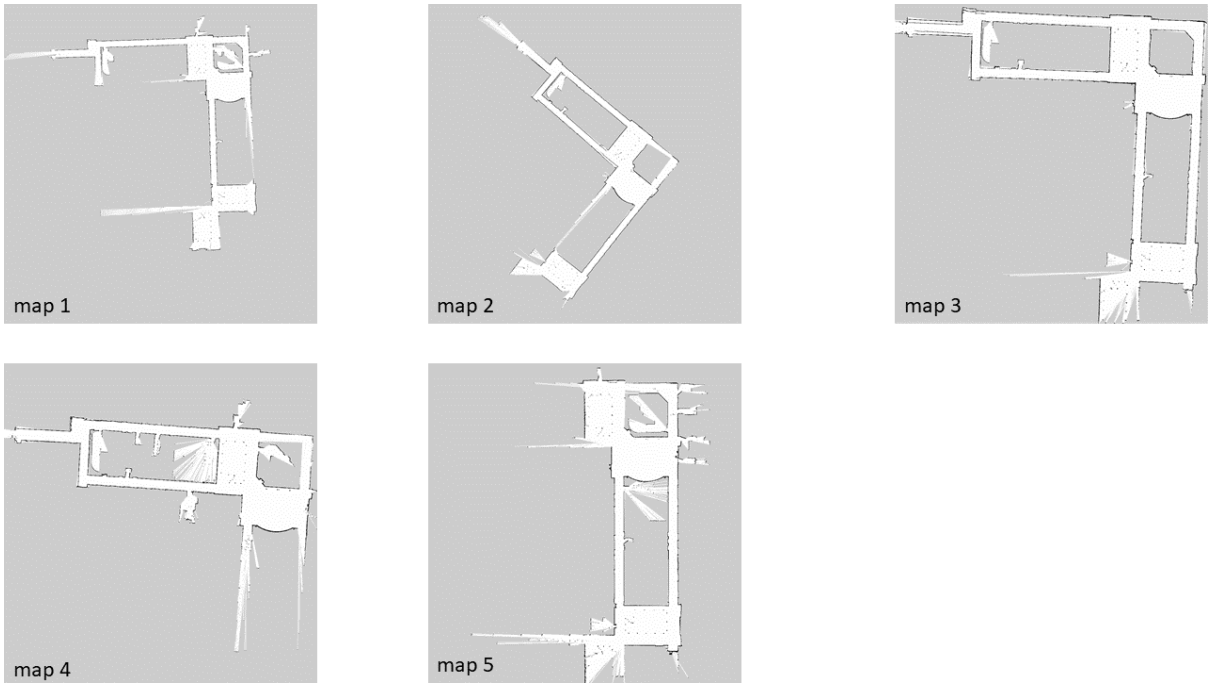


Figure 3.7: The following maps are of a large indoor environment used to test the robustness of the feature detection methods. The maps are produced using hector slam with a resolution of 0.5 m/cell. The generation of the maps are discussed in further details in Chapter 5.

The match ratio is used to indicate the feature detection methods' sensitivity to geometric transformation (such as translation, scaling, sheer and rotation) and noise for occupancy grid maps. For this thesis, the match ratio is used to investigate rotation, translation,

[2]http://opencv.org
[3]Data-sets: https://drive.google.com/drive/folders/1c_2-T8FcrSmE0VDyHw20-jRLHKRuAvpE
[4]GitHub repository: https://github.com/dikokob/DikokoMScEng

scaling and noise sensitivity. The match ratio is calculated by

$$\frac{matches}{\min(num\_features\_orignal, num\_features\_mod)} \tag{3.6}$$

Where $num\_features$ is the number of features, and $matches$ is the number of matches found between the original and modified map features, using the earlier discussed Brute-Force matcher. The higher the match ratio, the more matches were found between the two maps. Note that in the investigations *Kpnts1* are key-points from the original map and *Kpnts2* are key-points from the modified map.

3.3.5.1 Number of features

SIFT, SURF and ORB each produce a certain amount of features based on the algorithms. Figure 3.8 shows the features with descriptors shown by the feature's size and direction by the line within the circle. SIFT and ORB have a similar number of features whereas SURF has almost ten times more features, shown in 3.1. The number of features detected impacts the time cost of the process, more specifically the matching process. Also to be noted in Figure 3.8, most of the ORB features are on the corners due to the nature of the algorithm and SIFT, and SURF features are highly variable size and location.

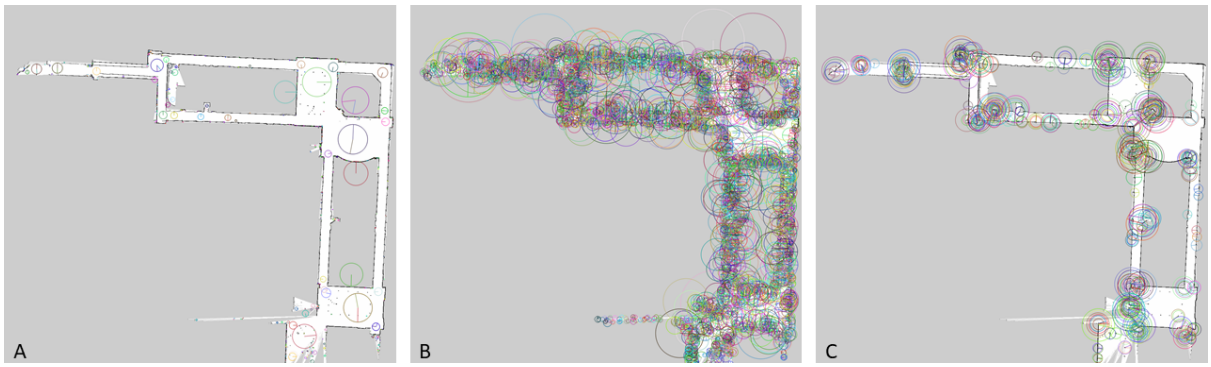


Figure 3.8: Features detected on map_3 from Figure 3.7 where A) SIFT, B) SURF and C) ORB. See [5]for raw data.

Table 3.1: Average number of detected features (SIFT, SURF and ORB) of maps 1-5 in Figure 3.7.

Maps	SIFT	SURF	ORB	Total
map 1	492	4291	501	5284
map 2	528	3786	500	4814
map 3	530	4503	500	5533
map 4	505	4224	500	5229
map 5	539	3282	500	4321

[5]https://github.com/dikokob/DikokoMScEng

3.3.5.2 Rotation

In this investigation, map_3 is rotated in steps of 30 degrees. The rotated map is matched with the original map. The results are shown in Figure 3.9 and Table 3.2 are of map_3 rotated, where ORB is the fastest, followed by SIFT and SURF being the slowest. SIFT also has the highest match ratio, followed by ORB, with the lowest match ratio.

Table 3.3 shows the match ratio versus the rotation angle, where rotation 0, 90, and 180 degrees ORB and SURF have the highest match ratio. However, they do not perform as well for other angles. The best overall performing feature detector is SIFT with an average of 0.90 match ratio, with SURF being the worst-performing. Also, angles 0, 90, and 180 degrees performed the best overall, with angles of 30, 60, 120 and 150 degrees performing poorly comparatively. The results indicate a potential limitation where relative angles are not 0, 90 and 180 degrees.

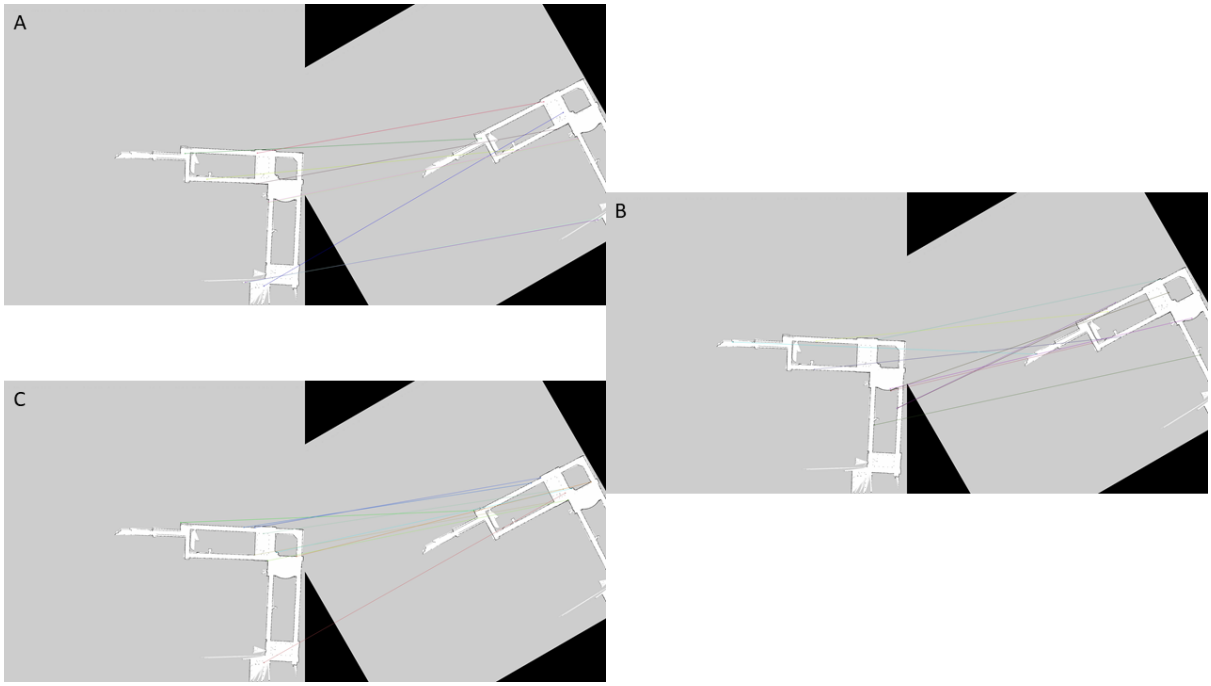


Figure 3.9: The matching of 30° rotation images using (a) SIFT (b) SURF(c) ORB.

Table 3.2: Results of comparing the images with 30° rotation.

	Time(sec)	Kpnts1	Kpnts2	Good Matches	Match Ratio
SIFT	1.4	530	489	305	0.62
SURF	1.2	4503	5128	1370	0.30
ORB	0.39	500	500	280	0.56

Table 3.3: Matching ratio versus the rotation angle.

Angle →	**0°**	**30°**	**60°**	**90°**	**120°**	**150°**	**180°**	**Average**
SURF	1	0.30	0.30	1	0.30	0.30	1	0.60
SIFT	0.95	0.62	0.63	0.92	0.63	0.65	0.90	0.90
ORB	1	0.56	0.58	1	0.56	0.58	1	0.75

3.3.5.3 Translation

In this investigation, map_3 is translated in steps of 200, with 0 being the control. The translated map is matched with the original map. The results are shown by Figure 3.10, Table 3.4 and Table 3.5.

Table 3.4 shows that ORB is the fastest performing method followed by SIFT, and then SURF. Table 3.5, shows that SURF as best performing method at match ratio of 0.99 followed closely by SIFT at 0.97. ORB has the lowest performance to varying translations, with a match ratio of 0.59.

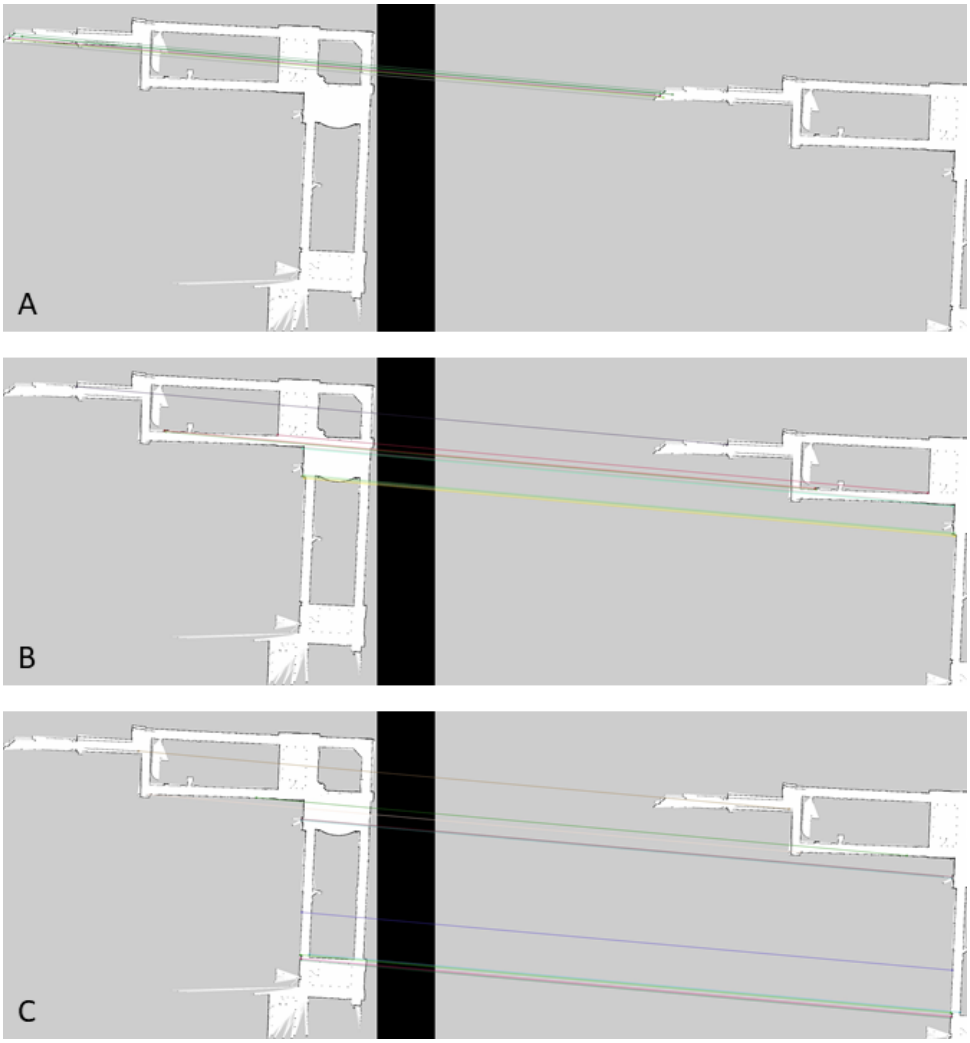


Figure 3.10: The matching of translation(x:200 y:200) images using (a) SIFT (b) SURF (c) ORB.

Table 3.4: Results of comparing the images of translation in 3.10.

	Time(sec)	**Kpnts1**	**Kpnts2**	**Good Matches**	**Match Ratio**
SIFT	3.23	530	285	278	0.98
SURF	3.79	4503	2598	2595	1
ORB	1.70	500	500	280	0.56

Table 3.5: Matching ratio versus the translation.

Translation →	**x:0 y:0**	**x:100 y:100**	**x:300 y:300**	**x:500 y:500**	**x:700 y:700**	**x:900 y:900**	**Average**
SIFT	0.95	0.95	0.99	0.98	0.99	0.97	0.97
SURF	1	0.99	0.99	0.99	0.99	0.99	0.99
ORB	1	0.66	0.52	0.49	0.47	0.41	0.59

3.3.5.4 Scaling

In this investigation, map_3 is scaled in steps of 0.2. The scaled map is matched with the original map. The results are shown by Figure 3.11, Table 3.6 and Table 3.7.

Table 3.6 shows that for a scaling operation of 0.4, ORB is the fastest followed by SIFT then SURF. Furthermore, SIFT has the highest match ratio of 0.88 at a scale of 0.4, followed by SURF, then ORB.

Now Table 3.7, show the match ratio at multiple scales, which presents SIFT at the best overall performing method, with a match ratio of 0.82. Followed by SURF at 0.67, then ORB at 0.49.

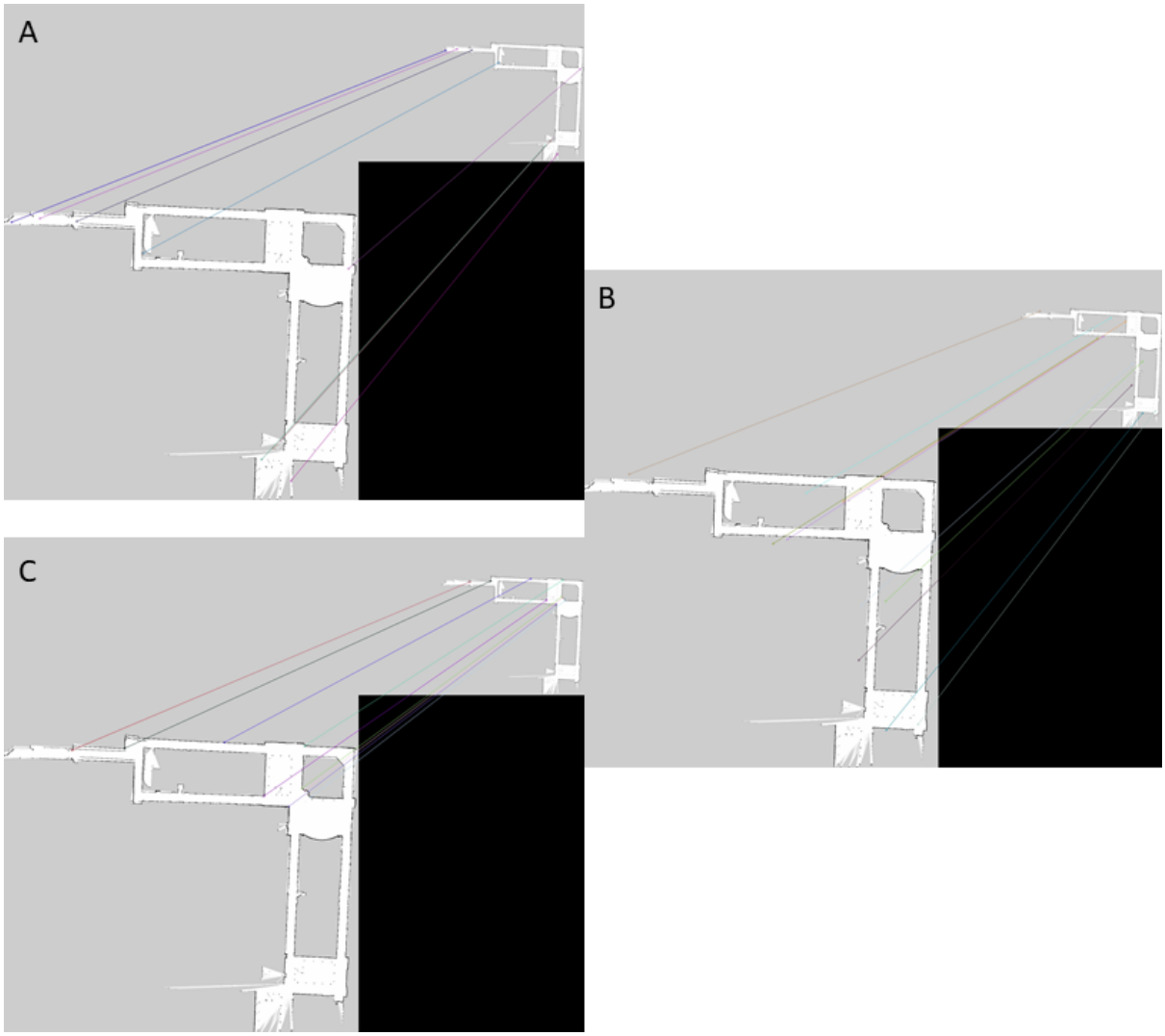


Figure 3.11: The matching of varying scaling images using (a) SIFT (b) SURF (c) ORB.

Table 3.6: Results of comparing the images with 0.4 scaling.

	Time(sec)	**Kpnts1**	**Kpnts2**	**Good Matches**	**Match Ratio**
SIFT	1.05	530	84	74	0.88
SURF	1.35	4503	1013	671	0.66
ORB	0.29	500	496	208	0.42

Table 3.7: Matching ratio versus the scaling.

Scaling →	**0.2**	**0.4**	**0.6**	**0.8**	**Average**
SIFT	0.81	0.88	0.85	0.76	0.82
SURF	0.68	0.66	0.67	0.66	0.67
ORB	0.39	0.42	0.52	0.65	0.49

3.3.5.5 Noisy maps

In this investigation, map_3 has salt and pepper noise applied at steps of 0.01. The modified image is then matched with the original image. The results are shown by Figure 3.12, Table 3.8 and table3.9.

Table 3.8 shows match ratio and computation time at 3% noise, where ORB has the fastest computation time at 4.82 seconds followed by SIFT at 7.92 seconds, and SURF at 35.02 seconds close to ten times that of ORB.

Now Table 3.8 shows the match ratio of the different methods at steps of 0.01 noise. SIFT is best performing with an average of 0.83 match ratio followed by ORB at a 0.67 match ratio, and then SURF at a 0.61 match ratio.

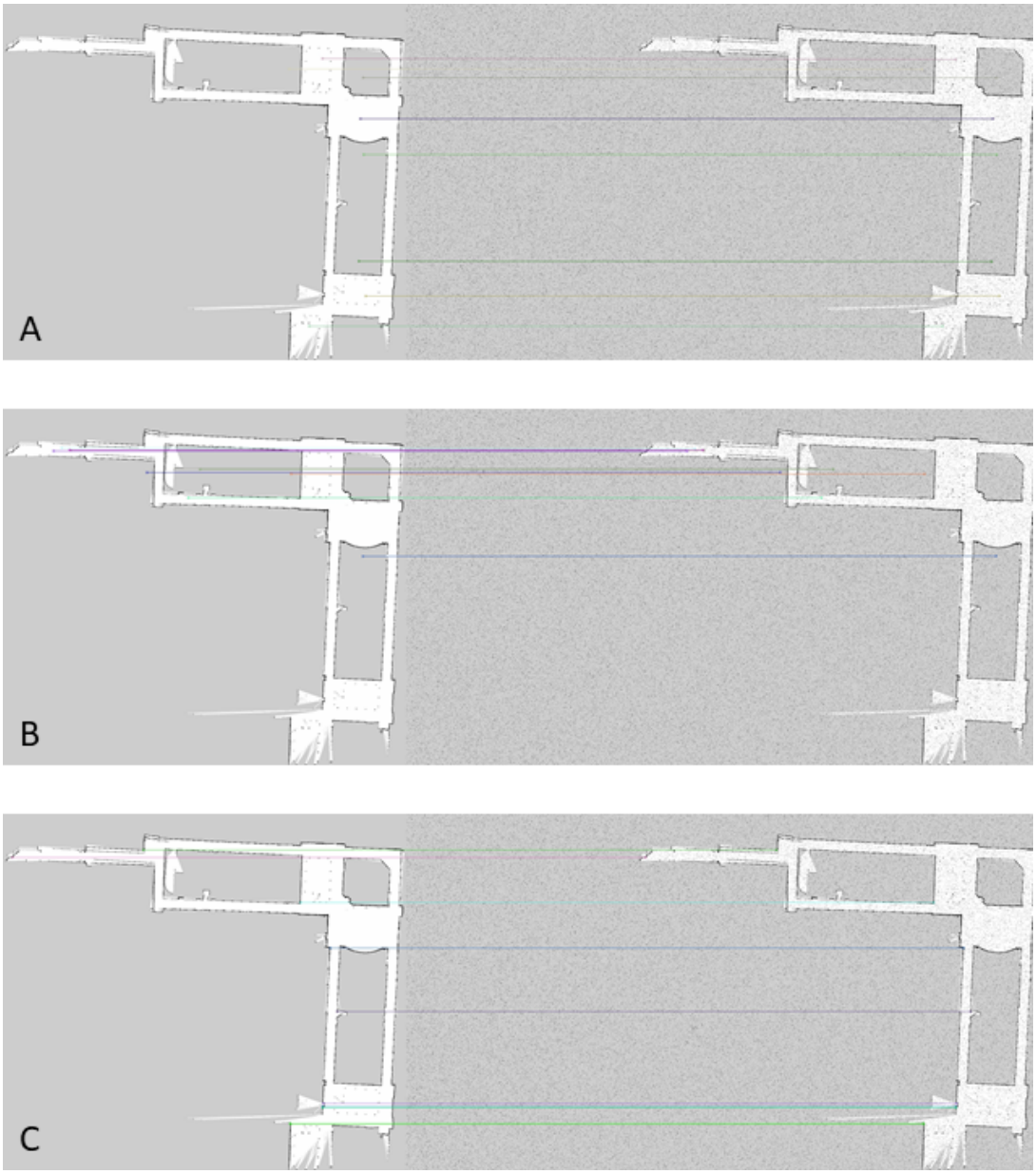


Figure 3.12: The matching of 0.03 salt and pepper noise on varying images using (a) SIFT (b) SURF (c) ORB.

Table 3.8: Results of comparing the images with 0.03 salt and pepper noise.

	Time(sec)	Kpnts1	Kpnts2	Good Matches	Match Ratio
SIFT	7.92	530	13436	441	0.83
SURF	35.02	4503	115422	2105	0.47
ORB	4.82	500	500	311	0.62

Table 3.9: Matching ratio versus the noise.

Noise →	0.01	0.02	0.03	0.04	Average
SIFT	0.82	0.82	0.83	0.83	0.83
SURF	0.76	0.65	0.55	0.47	0.61
ORB	0.74	0.67	0.63	0.62	0.67

3.3.5.6 Method selection summary

In this section, three different feature detection methods are compared similarly to work done in [112], with different translation and deformation such as scaling, rotation, noise and translation. For purposes of the work in this thesis, these transformations were applied to an occupancy grid map produced for this work. The results present display matching evaluation parameters such as the number of key-points, match ratio and the execution time for each method.

The results show that ORB is the fastest, while SIFT performs the best in most scenarios. For exceptional cases when rotation angle is in steps of 90 degrees ORB and SURF outperform SIFT, these results match those of [112]. In translation operations, SIFT and SURF perform similarly. Unlike results in [112] ORB and SIFT do not show similar performance in noisy maps. In noisy maps, SURF's execution time is close to ten times the performance of ORB. In conclusion, SIFT is the most consistent in producing match ratios larger 0.80 for all the transformations and deformations. Therefore, SIFT will be used in this thesis because it has superior accuracy, and the time cost is not too high compared to SURF and ORB.

3.4 Summary

In this chapter, the map merging problem was firstly defined, followed by investigating the literature around map merging. From the literature, image registration modules are discussed. Finally, image registration feature detectors SIFT, SURF and ORB are investigated to find the optimal methods for application in this thesis. The investigation is conducted due to the lack of literature comparing the methods in an occupancy grid map merging application. In the next chapter implementation of multi-robot, multi-session and multi-resolution map merging is discussed and described.

Chapter 4

Implementation

This chapter provides a detailed description of each module used in the multi-robot multi-session system to generate a global map on each mobile robot in the team.

4.1 Overview

In this work, a multi-robot, multi-session map merging algorithm capable of merging maps of different resolutions, is presented. Each robot in the system can generate a local and global map, hence it is a distributed system. Hector SLAM is used to generate the local maps, which are inputs to the map merging algorithm that produces a global map. A map merging algorithm is presented. The implementation of the algorithm is then presented for real-time capability. The map merging algorithm has input maps from other robots in the network and previously mapped session, enabling the robot to deal with multiple session mapping.

4.2 Map merging algorithm

This section describes the map merging algorithm used in this thesis, which is based on image registration. The image registration is transforming different images into the same coordinate system, using the following steps feature detection, matching, alignment and merging. In this work, occupancy grid maps are represented as images where a pixel defines whether an area is occupied, unoccupied or unknown. These images can be taken at multiple resolutions, viewpoints, and using different sensors; therefore, the spatial relationships or the maps include rotation, translation and scaling. As described in the previous chapter Scale-invariant feature transform (SIFT) will be used as a feature detector as it deals well with the spatial relationship described above.

OpenCV was started at Intel in 1999 by Gary Bradsky, however, the first release in 2000 when it was used on Stanley (the vehicle that won the DARPA Grand Challenge that year). OpenCV is supported in C++, Python and Java; hence it can be used across various systems. The Python API of OpenCV (OpenCV-Python) is used to implement

the solution; the API includes a wide range of tools such as feature detection, object detection, machine learning and video analysis. The image registration module used in this work is based on Scale-invariant feature transform (SIFT), results presented in Chapter 3 show that it is superior for this scenario. The scale and rotation invariant nature of SIFT, it can be used in multi-resolution scenarios.

Feature detection, matching features, and alignment are included and merging the maps to form a global map.

4.2.1 Transformation

Assume there are two maps map_1 and map_2, where map_1 is the local map of $robot_1$ and consequently map_2 is the local map of $robot_2$. This section describes the process of transforming map_2 into map_1's frame of reference. Algorithm 4.1 describes this process.

Feature detection is the initial step described in lines 2-3 in algorithm 4.1, where a pair of key-points and descriptors are returned (see Chapter 3 for more information). The features are then matched using a Brute-Force Matcher (line 4). The Brute-Force Matcher takes descriptors from each image and matches them using a distance calculation to determine the closest features. In this case the Euclidean distance described by Equation 3.5 [103]. These matches are referred to as $good\_matches$. The $good\_matches$ are then used to get the locations of the features that were $good\_matches$ in lines 5-10. Next, the affine transformation matrix described by Equation 3.2 is computed using RANSAC. In Section3.3.4, parameters $ransacReprojThreshold$ and $maxIters$ were introduced and here we define their values.

$maxIters$ is the maximum number of iterations and is described by the following equation :

$$1 - confidence = (1 - match\_ratio^n)^{maxIters} \tag{4.1}$$

where, $\{confidence | confidence \in \mathbb{R}, 0 < confidence < 1\}$, $match\_ratio$ is the ratio of possible matches to good matches described by Equation 3.6, n is the minimum number of features needed for a match [111]. By taking the log of both side get the following equation:

$$maxIters = \frac{\log(1 - confidence)}{\log(1 - match\_ratio^n)}. \tag{4.2}$$

Then we choose a $confidence$ of 0.99 which ensures minimal error, and minimum number of features n of 2, to get:

$$maxIters = \frac{\log(0.01)}{\log(1 - match\_ratio^2)} \tag{4.3}$$

The maximum iterations can then be described by the following Figure 4.1. Where the

Algorithm 4.1: Transform map into main maps frame of reference

```
/* This function transforms map_2 into map_1's frame. The function
   requires two 2D occupancy grid maps of size n by m and RANSAC
   parameters(confidence, and reproj_threshold) */
Function Transform(map_1, map_2, confidence, reproj_threshold) :
 transformed_map_2, scale
    /* extract feature key-points and corresponding descriptors */
    kp_1, des_1 = detect_keypoint_descriptors(map_1)
    kp_2, des_2 = detect_keypoint_descriptors(map_2)
    /* extract feature matches between two maps */
    good_matches = match_features(des_1, des_2)
    /* get feature locations of good matches */
    feature_locations_1 = [0, good_matches.length][0, 2] : 2D array of float
    feature_locations_2 = [0, good_matches.length][0, 2] : 2D array of float
    for i, match in enumerate(good_matches) do
        feature_locations_1[i, :] = kp_1[match.des_1_index].pt
        feature_locations_2[i, :] = kp_2[match.des_2_index].pt
    end
    /* compute match ratio */
    match_ratio = good_match.length/min(kp_1.length, kp_2.length)
    /* compute maximum iterations needed to estimate affine
       transformation matrix using RANSAC */
    max_iter = log(1 − confidence)/ log(1 − match_ratio^2)
    /* estimate affine transformation matrix */
    affine_transformation_matrix =
     estimate_affine_transformation(feature_locations_1, feature_locations_2,
     max_iter, confidence, reproj_threshold)
    /* transform map using affine transformation matrix */
    transformed_map_2 = affine_transform(map_2, affine_transformation_matrix)
    /* get scale and angle of rotation from the transformation matrix
       */
    ss = affine_transformation_matrix[0, 1]
    sc = affine_transformation_matrix[0, 0]
    scale = √(ss * ss + sc * sc)
    angle_of_rotation = arctan ss/sc * 180/π
    return transformed_map_2, scale
end
```

smaller the match ratio, the more iterations are required to estimate the best model fit.

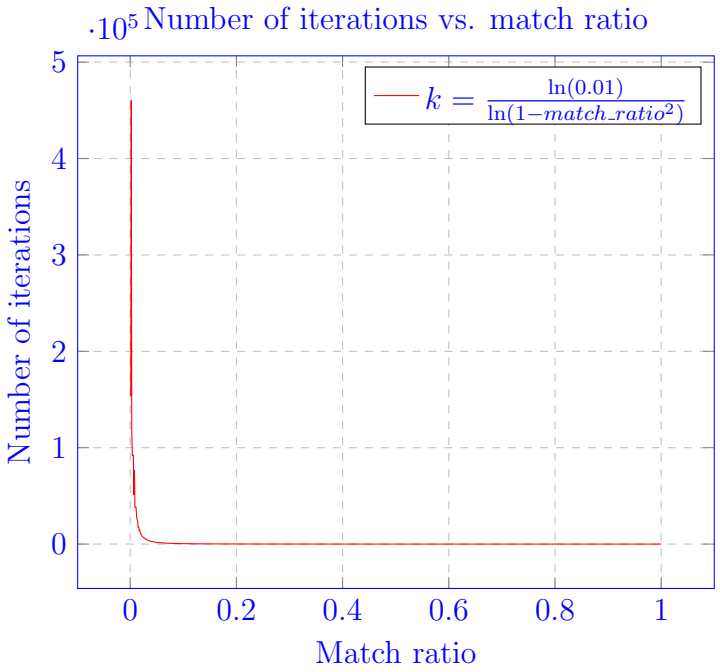


Figure 4.1: This figure illustrates the relationship between the number of iterations and the match ratio. The lower the match ratio, the higher the number of iterations. Therefore, lower match ratio matches are computationally costly compared to higher match ratio matches.

RANSAC uses *ransacReprojThreshold* in order to determine if a data sample agrees with a model or not. The samples under this threshold will then form that consensus for that model and the data set's inliers if the correct model is found. Hence, it should be chosen according to the model needs. To determine the threshold for this thesis's work, maps in Figure 3.7 are paired into groups of two giving a combination of 10. These where then run on line 1 - 14 of algorithm 4.1. These maps were then merged and then recorded if the merge was successful if they failed. Table 4.1 shows that *ransacReprojThreshold* values of 8,9 and 10 show a 100% success rate this work we will use a *ransacReprojThreshold* value of 9.

The affine transformation matrix is then used to transform map_2 into map_1's frame of reference, using bi-linear interpolation. The transformation matrix is of the form:

$$M = \begin{bmatrix} cos(\theta) \cdot s & -sin(\theta) \cdot s & t_x \\ sin(\theta) \cdot s & cos(\theta) \cdot s & t_y \end{bmatrix} \tag{4.4}$$

where, θ is the rotation angle, s the scaling factor and t_x,t_y are translations in x, y axes respectively.

Finally, from the affine transformation matrix, the scaling factor and angle of rotation are calculated to be used in further processes. The scaling factor (s) and rotation angle

Table 4.1: Results from running merging on threshold between 1 and 10 to determine the optimal *ransacReprojThreshold*. Where P represents a successful transformation and F a failed transformation.

Threshold	**Combination Num**										**P Rate (%)**
	1	2	3	4	5	6	7	8	9	10	
1	F	F	P	P	F	P	F	P	P	F	50
2	F	F	P	P	F	P	F	P	P	P	60
3	P	P	P	P	P	P	F	P	P	P	90
4	P	P	P	P	P	P	P	P	F	P	90
5	P	P	P	P	P	P	P	P	F	P	90
6	P	P	P	P	P	P	P	P	F	P	90
7	P	P	P	P	P	P	P	P	F	P	90
8	P	P	P	P	P	P	P	P	P	P	100
9	P	P	P	P	P	P	P	P	P	P	100
10	P	P	P	P	P	P	P	P	P	P	100

(θ) can be computed using:

$$
\begin{aligned}
s &= \sqrt[2]{M_{1,2} * M_{1,2} + M_{1,1} * M_{1,1}} \\
\theta &= \arctan\left(\frac{M_{1,2}}{M_{1,1}}\right) \times \frac{180}{\pi}
\end{aligned}
\tag{4.5}
$$

.

4.2.2 Multiple map transformation

The **Transformation** function (algorithm 4.2)is then used to compute and transform more than one map into the local map's frame of reference. The **Main** function (algorithm 4.2) receives multiple maps and their corresponding resolutions, and then they are merged

to form a global map.

Algorithm 4.2: Main Function

```
/* This is the primary function which receivers all the maps and
   corresponding scales, then transforms them relative to map[0] and
   returns the global_map */
Function Main(maps_list, map_resolutions_list) : global_map
    /* initialise variables */
    confidence = 0.99
    reproj_threshold = 9
    successfully_transformed_maps = []
    /* Loop through maps and transform them relative to maps_list[0] */
    for i in range(1, len(maps_list)) do
        /* transform map based on the reference maps_list[0] */
        transformed_map, scale =
          Transform(maps_list[0], maps_list[i], confidence, reproj_threshold)
        /* check success of transformation based on resolution */
        if (map_resolutions_list[0] × (1 − confidence)) >
          abs(map_resolutions_list[0] − (map_resolutions_list[i]/scale)) then
            successfully_transformed_maps.append(transformed_map)
        end
    end
    global_map = maps_list[0]
    /* merge only successfully transformed maps */
    for map in successfully_transformed_maps do
        global_map = merge(global_map, map)
    end
    return global_map
end
```

Given that not all transformations will be successful, the success is determined by comparing the expected scaling factor and the computed scaling factor. The expected scaling factor is determined by the local map's resolution divided by the resolution of maps from the other robot. This expected scaling factor needs to be within $1 - confidence$, which is the expected RANSAC estimation error.

Table 4.2 describes the merging rule. Since the partial maps are of the same resolution, the rule assumes that the interpolation or extrapolation is sufficient.

Table 4.2: Rules of Map Merging between two partial maps.

		Partial Map 2		
		Unknown	**Free**	**Occupied**
Partial Map 1	**Unknown**	Unknown	Free	Occupied
	Free	Free	Free	Occupied
	Occupied	Occupied	Occupied	Occupied

The successful maps are then merged using the following rule in Table 4.2, which allows

for a suitable global map to be obtained. If there are one or more successful alignments the global map is then returned with it is corresponding resolution.

4.3 ROS implementation

The algorithm 4.2 described above is then implemented on the Robot Operating System (ROS) to perform the map merging algorithm in real-time. Figure 4.2 describes the system overview of a robot. The robot will receive maps from other robots, maps from previous sessions, and a local map that it produces onboard, and merge them to produce a global map. Each robot must scan the network for possible partial local maps from other robots, to retrieve the maps.

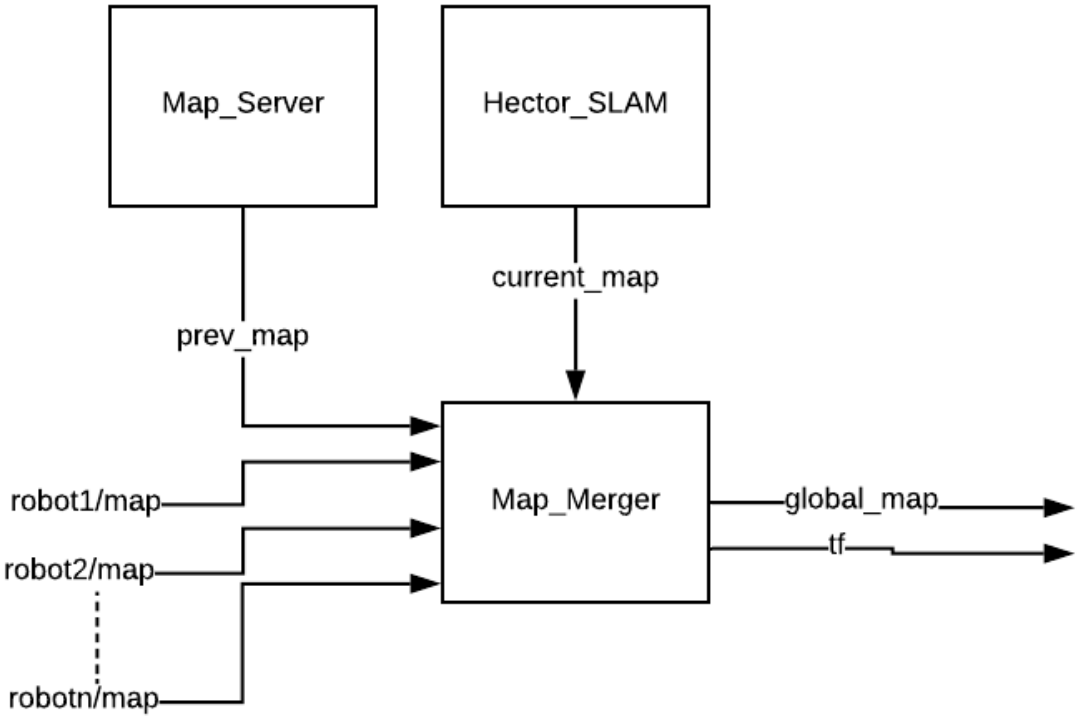


Figure 4.2: System overview of the solution. The map merging algorithm's inputs included the current partial map (*current_map*), previous partial map (*prev_map*), and partial maps of other robots in the environment ($robot1/map$,). The resultant map includes the global map ($global_map$) and transformations (tf).

Therefore, using ROS needs to solve the following problems:

- **Communication** between robots
- **SLAM** implementation to produce a local map

- **Serve** maps from previous sessions
- **Map merging** algorithm needs to be incorporated and the resulting global map needs to be published

4.3.1 What is ROS?

Robot Operating System (ROS) is an open-source, meta-operating system used widely in robotics, and currently maintained by Open Robotics [1]. ROS is designed to work with both physical and simulated robots and provides hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. The creators had a philosophy to *"allow anyone to create functionalities that can be shared and used in other robots without much effort so that we do not reinvent the wheel"* [1]. For example, a driver for ROS to interact with an Arduino was created and shared with the ROS community. The community makes the framework easy to adopt for many users from Researchers to Hobbyists.

ROS defines a recommended file structure and software build to promote reuse and ease of sharing in the community. ROS uses the concept of packages (similar to the UNIX operating systems) as the ROS ecosystem's fundamental building block. ROS currently supports the following programming languages Python, C++, and Lisp, and has experimental libraries in Java and Lua.

4.3.2 ROS communication

For the robots to communicate, ROS implements several different communication styles, including synchronous communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. The communication is facilitated by the peer-to-peer network, which can be distributed across machines. The network is based on a graph architecture with a decentralised topology where processing occurs in *nodes* that may publish or subscribe messages, such as multiplex sensor, control, state, planning and actuators. Topics are asynchronous, and *nodes* do not explicitly communicate with each other, services provide synchronous communication, as shown in Figure 4.3.

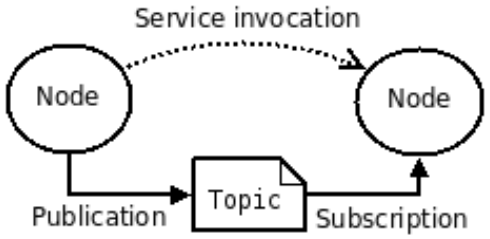


Figure 4.3: This illustrates the basic node concept.Note that *nodes* do not explicitly communicate with each other, services provide synchronous communication. Taken from [1].

[1] https://www.openrobotics.org/

The ROS *master* acts as a name-service for the graph; it stores topics and services information for ROS *nodes*, as shown in Figure 4.3. *Nodes* report information to the *master*, and receive messages (include arbitrarily nested structures and arrays) from other *nodes* through the *master*. Nodes need to be registered to the *master* to receive and pass messages. Messages (shown in Figure 4.4) are communicated via topics between *nodes* using publish/subscribe architecture. A node subscribes to a topic, requests a connection through the *master* and connects to a publisher node, hence receiving messages. There may be multiple concurrent *nodes* publishing and subscribing to a single topic, and a node may publish and subscribe to multiple topics. The *master* will also make callbacks to *nodes* when registration information changes, which allows *nodes* to create connections as new *nodes* are registered dynamically. Bag files are used to save and playback ROS messages such as sensor data to develop and test algorithms.

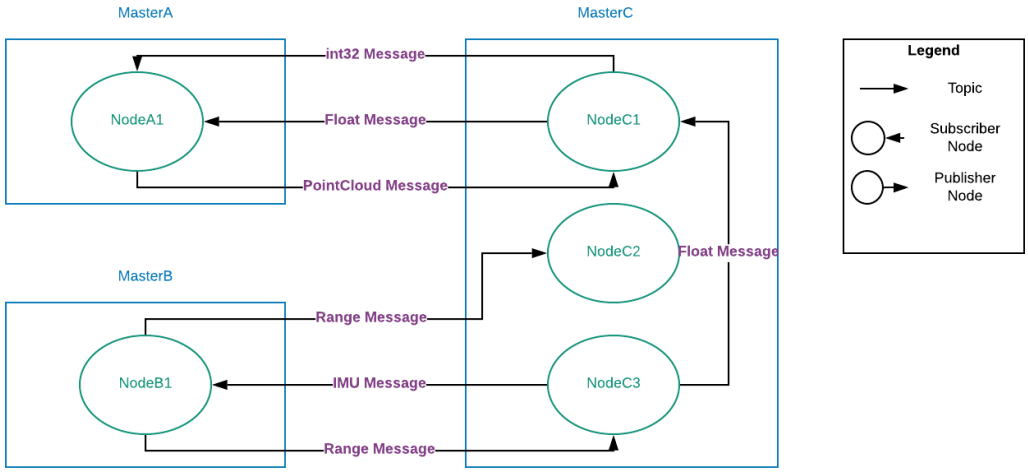


Figure 4.4: An example of multiple machine/robot ROS implementation architecture. This shows the interaction of nodes between multiple machines/robots. Messages are passed through topics.

For example, given that there is a need to control and access data from a Hokuyo laser range-finder, *hokuyo node* driver can be used to talk to the laser and publishes messages of the following type sensor *msgs/LaserScan* to a topic named *scan*. A node (i.e. *laser process*) can be written to subscribe to messages on the scan topic, to process *msgs/LaserScan* messages. Note that all *hokuyo node* does is to publish messages to the *scan* topic, without knowledge of any nodes subscribed to the topic. On the other hand, the *laser process* node subscribes to *scan* the topic without knowledge of whether any node is publishing. Hence the nodes are decoupled, the nodes can be started, killed and restarted.

When dealing with multiple robots, there may be a need to remap topics for each robot, primarily when robots used the same nodes internally. ROS supports command-line remapping of names, which means a compiled program can be reconfigured at run-time to operate in a different graph topology. For example, the outputs from a laser range-finder for robot1 should not be published to the same topic as the outputs from the laser range-finder of robot2; therefore each robot should publish the scan message to different

topics. Defining a prefix will allow each robot to use a unique name-space for all data and transformations; for example, the two robots' laser scan topic will be robot0/scan, robot1/scan.

4.3.3 SLAM in ROS

Examples of ROS packages dedicated to SLAM algorithms include *slam_gmapping*[2], *mrpt_slam*[3] and *hector_slam*[4]. The stable and commonly used *hector_slam* is implemented in this work, due to the advantages discussed in Section2.4.4. The *hector_mapping* node is in the package, and it is a SLAM approach that can be implemented without odometry and platforms that exhibit roll/pitch motion(of the sensor, the platform or both) such as quad-rotor UAVs. It leverages the high update rate of modern LIDAR systems like the Hokuyo UTM-30LX and provides 2D pose estimates at the sensors' scan rate. While the algorithm does not provide explicit loop closing ability, it is sufficiently accurate for many real-world scenarios as described in [69]. Furthermore, the system has successfully been used on unmanned ground robots, unmanned surface vehicles, handheld mapping devices and logged data from quad-rotor UAVs and was developed by [16].

4.3.4 Map and pose exchange

Robots must be aware of other teammates who join and leave the network, to perform the mapping task cooperatively. This communication is achieved using a non-distributed *roscore* which is a collection of nodes and programs of a ROS system. Instead of running *roscore* on each robot a dedicated robot or networked computer will be used to facilitate the communication, to do this $*roscore* is run on the machine chosen to be the master, then the IP of the communication machine is used to run the following $*exportROSMASTERURI* = *http* : //*masterrobotIP* : 1234/on each robot in the network, to add them to the network. Even though the master *roscore* has been selected, nodes will still be computed on the individual robots,*roscore* only facilities communication between the teammates.

Assuming that each robot is aware of its ID, i.e. robot1, the robot searches through topics to find which robots exist in the network. Since robots can leave the network(i.e., map an area with low-quality network link), each robot maintains a list of all the mobile robots. Assigning a prefix allows robots to share information such as occupancy grids, poses, laser scans or messages. The following information is shared between the robots *OccupancyGrid* and *Pose*.

[2]http://wiki.ros.org/gmapping
[3]http://wiki.ros.org/mrpt_slam?distro=kinetic
[4]http://wiki.ros.org/hector_slam

4.3.5 Multi-session

Multi-session mapping considers the problem of combining the results of mobile robot SLAM missions performed repeatedly over time in the same environment. The goal is to robustly combine multiple maps from different missions in the same environment in a standard coordinate system. Multi-session mapping involves having to deal with the fact that robots over long period missions eventually will be shut down and may revisit areas where they previously mapped. One way of doing multi-session mapping is having a robot localise it is self in a previously-built map. The solution allows the robot to use the same referential, and only one map is created per session; however, this requires the robot the start in the already mapped portion of the environment. In this work, the previously mapped portion of the environment is dealt with as a map from another robot. Hence the robot is not required to start in the portion of the environment previously mapped. Using ROS *map_server*, a map is stored and can be retrieved later for use in another process. After a session, the map is stored, and if a map were previously stored on startup, it would be fed into the map merging algorithm.

4.3.6 ROS node

Figure 4.5 illustrated the ROS *node* used in this work. The *node* is implemented using Python, the *rospy* library is used to interact with the ROS system. The *node* is designed with parameters to alter the image registration process and parameters to alter the rate at which the *node* runs. In step one the parameters are initialised to default values of which the image registration parameters are determined in Chapter 3, it is also necessary to know the current robot map topic. The current robot topic is used to ensure that all the partial maps are transformed into that frame. Step two involves getting the list of all the topics which are in the network using *rospy.get_published_topics()*; the published topics messages of type *nav_msgs/OccupancyGrid* are then selected for the map merging process. In step 6 and 7, if there are maps in the network, the maps are then stored using the *map_server node*. In the final step 8, the image registration based map merging algorithm is used. The map merging algorithm is described in Section4.2[5].

[5]GitHub repository: https://github.com/dikokob/DikokoMScEng

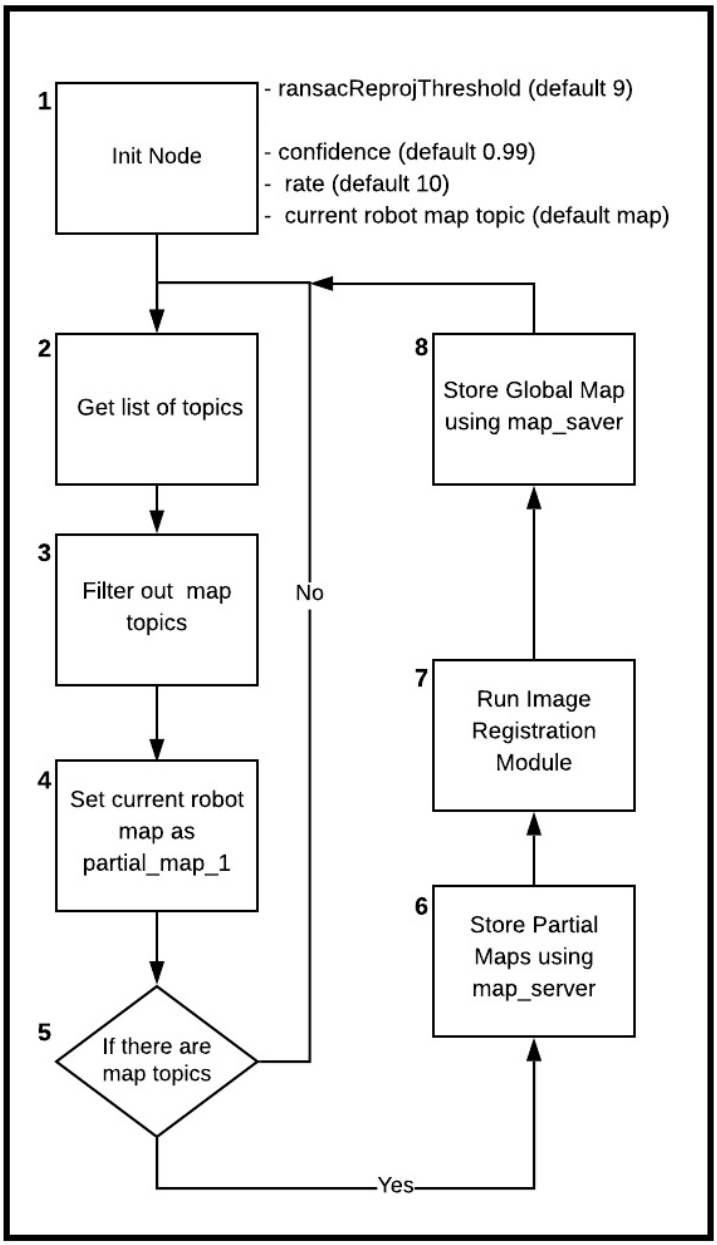


Figure 4.5: ROS node implementation.

4.4 Summary

In this chapter, a multi-robot multi-session map merging approach is presented. In the next chapter, validation experiments run in simulation and real-world scenarios are discussed. The simulation tests were conducted using two and three partial maps at different resolutions, and so are the real-world experiments.

Chapter 5

Experimental validation

This chapter describes and discusses the experimental methods and results used to validate the solution proposed in Chapter 4.

5.1 Introduction

The results were obtained using a distributed approach where each robot is tasked with both map generation and the full map merging process(alignment and merging) to obtain a global map relative to its local map.

Experimental tests were performed in both simulation and real-world scenarios to evaluate the proposed solution's performance. Note that the experimental tests are not performed for ideal cases; therefore, mapping errors will be present in the tests. For the real-world scenarios data generated for the proposes of this thesis is used, alongside publicly available data collected in the Maria Stata Center at MIT. This data from MIT has also been used in [113–115]. The data-sets generated and used can be found on Google drive. Given that the proposed solution in the previous chapter is aimed at multi-session, multi-robot and multi-resolution map merging, this requires several robots generating maps during multiple sessions at varying map resolution in indoor environments.

In all the experiments, ROS is used to both record and process the data. The data is recorded and saved as *rosbag* files, and these files are then used to generate the partial maps, at different resolutions.

5.2 Hector mapping parameters

Hector mapping which is discussed in a previous chapter2, is used to generate the partial maps. The parameters will be kept as default except for the frame parameters(which are *base_frame*, *map_frame* and *odom_frame*), *map_resolution* and *map_size*. The frame parameters are changed to match the robot frame. The *map_resolution* is varied to evaluate the solutions ability to deal with multiple resolutions. Finally, the*map_size* is

varied to match the environment size. Unlike *gmapping*, *hector_mapping* maps do not change size dynamically therefore a static value must be set.

5.3 Performance measures

The following performance measures are presented alongside the resultant global map of the map merging operation.

Resolution and calculated resolution

Assume that we have map_1 and map_2, where map_2 is transformed into map_1's frame of reference. Then the calculated new resolution of map_2 should be within

$$\begin{aligned} 1 - confidence &= 1 - 0.99 \\ &= 0.01 \end{aligned}$$

of the resolution of map_1. This performance measure is there to confirm that the error checking of the algorithm performs adequately.

Angle of rotation

The angle of rotation is computed from the affine matrix, shown in Equation 4.5. The angle is presented to show the angle which maps were rotated through to match the reference map.

Good matches and Match ratio

The good matches represent the number of features matched between to maps. These matches are presented with their corresponding match ratio. The match ratio introduced here is the same match ratio introduced in Section3.3.5. The match ratio is determined by:

$$\frac{matches}{\min(num\_features\_map_1, num\_features\_map_2)} \tag{5.1}$$

, where $num\_features$ is the number of features, and $matches$ is the number of matches found between map_1 and map_2 features, using the earlier discussed Brute-Force matcher. The higher the match ratio, the more matches were found between the two maps.

Percentage overlap

Assume we are merging two images, for the images to be merged an overlapping area is required to find matching features. The percentage of overlap is described by:

$$P_{overlap} = \frac{A_{overlap}}{A_1 + A_2 - A_{overlap}} \times 100 \tag{5.2}$$

Here, A_1 is the occupied and unoccupied area of map_1, A_2 is the occupied and unoccupied area of map_2 and $A_{overlap}$ is the occupied and unoccupied of the overlapping area. The

following sections focus on simulation results, followed by real-world results produced in this work and from the MIT Stata Center dataset [116]. In each experiment, the global map is produced relative to partial map_1 (the local map, generated using Hector Mapping). Therefore, the global map takes on the resolution of partial map_1.

5.4 Simulation experiments

This section presents experiments done with a simulated using a ROS and Gazebo. The experiments were performed to evaluate the performance of map merging algorithm on 2 and 3 robots. The simulation environment set up is described in Section5.4.1. The map merging algorithm results are presented in Section5.4.2.

5.4.1 Environment setup

Gazebo is an open-source simulation environment, which offers an accurate and efficient simulation of indoor and outdoor robots, a robust physics (such as illumination, gravity and inertia) engine, high-quality graphics and a graphical interfaces. Gazebo also works alongside ROS, allowing for simulated development and testing. To achieve ROS integration with stand-alone Gazebo, a set of ROS packages named *gazebo_ros_pkgs* provides wrappers around the stand-alone Gazebo. The wrappers provide the necessary interfaces to simulate robots in Gazebo using ROS messages, services and dynamic reconfigure [1]. The use of Gazebo enables the testing of scenarios without the need for a fully functional robot team, which are usually expensive. It also eliminates the limitation of battery life, which can be ignored in a simulation environment.

The main components of Gazebo are:

- **World file** - This file contains all the elements in the simulation world, such as robots, lights, sensors, and static objects.
- **Models** - The purpose of these files is to facilitate model reuse and simplify world files.
- **gzserver** - This server reads the world file and models to generate the simulation world.
- **gzclient** -The client connects to a running *gzserver* and visualises the simulation environment. This client also allows the user to modify the simulation environment, shown in Figure 5.1.

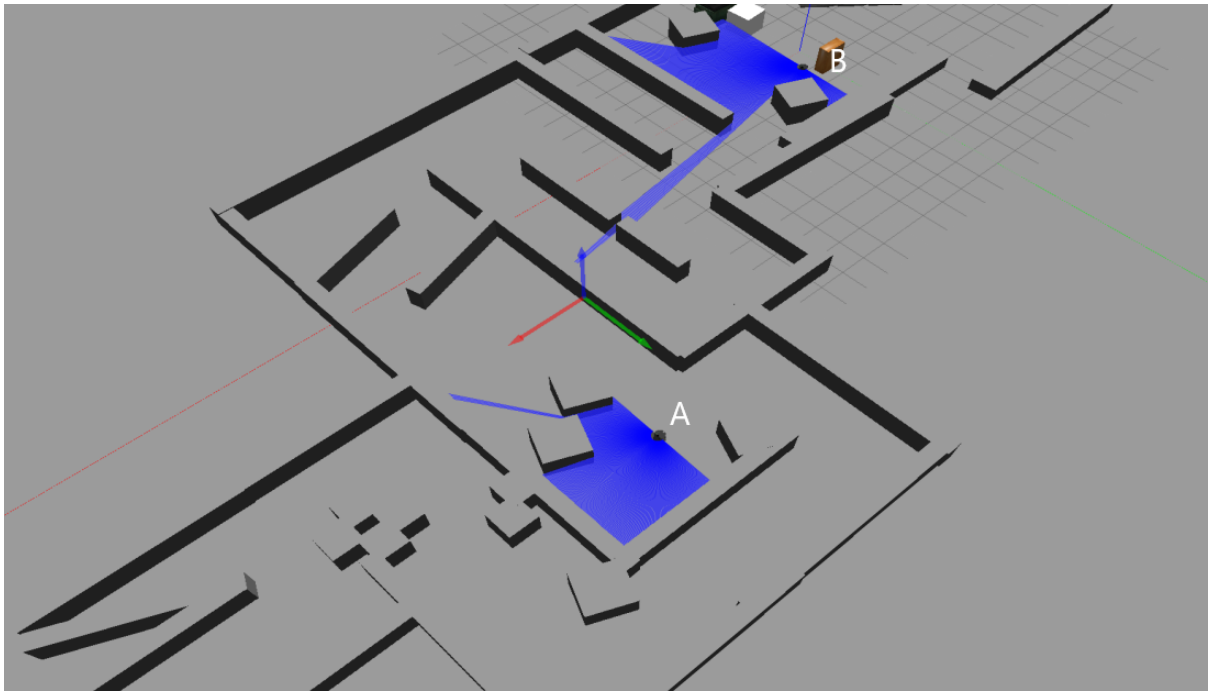


Figure 5.1: Gazebo *gzclient* generated from the *gzserver*. The *gzserver* generated the simulation world using a world file and model files. This figure describes two robots (annotated by A and B) and lidar sensor (which are the blue rays are laser rays) in an environment.

Figure 5.4, is the layout of the simulation environment used in this experiment. The environment shows two robots simulated in Gazebo building partial maps of the environment, a transform tree diagram with the coordinate frames associated with each robot is presented in Figure 5.2, each frame having a prefix to inform ROS which robot it belongs to. The graph also shows the average rate of message exchange transforms and the existing frames of the robots. The simulated robots are Kobuki platforms (see Figure 5.3) equipped with Hokuyo laser range finders, these robots are selected to match the real-world robot platform used. The simulated Kokoki Platform is controlled using the *turtlebot_teleop*[1] node, through the keyboard. The simulated environment is run on a Windows 10 laptop with the following specifications; Intel Core i5-8250U processor with 16 GB of RAM. The simulated environment is run on a virtual machine using VirtualBox, and the machine can be found on [2]. The simulated environment is used to initially test the solution proposed in a controlled environment with known parameters.

[1]http://wiki.ros.org/turtlebot_teleop

[2]Data-set: https://drive.google.com/drive/folders/$1c_2 - T8FcrSmE0VDyHw20 - jRLHKRuAvpE$

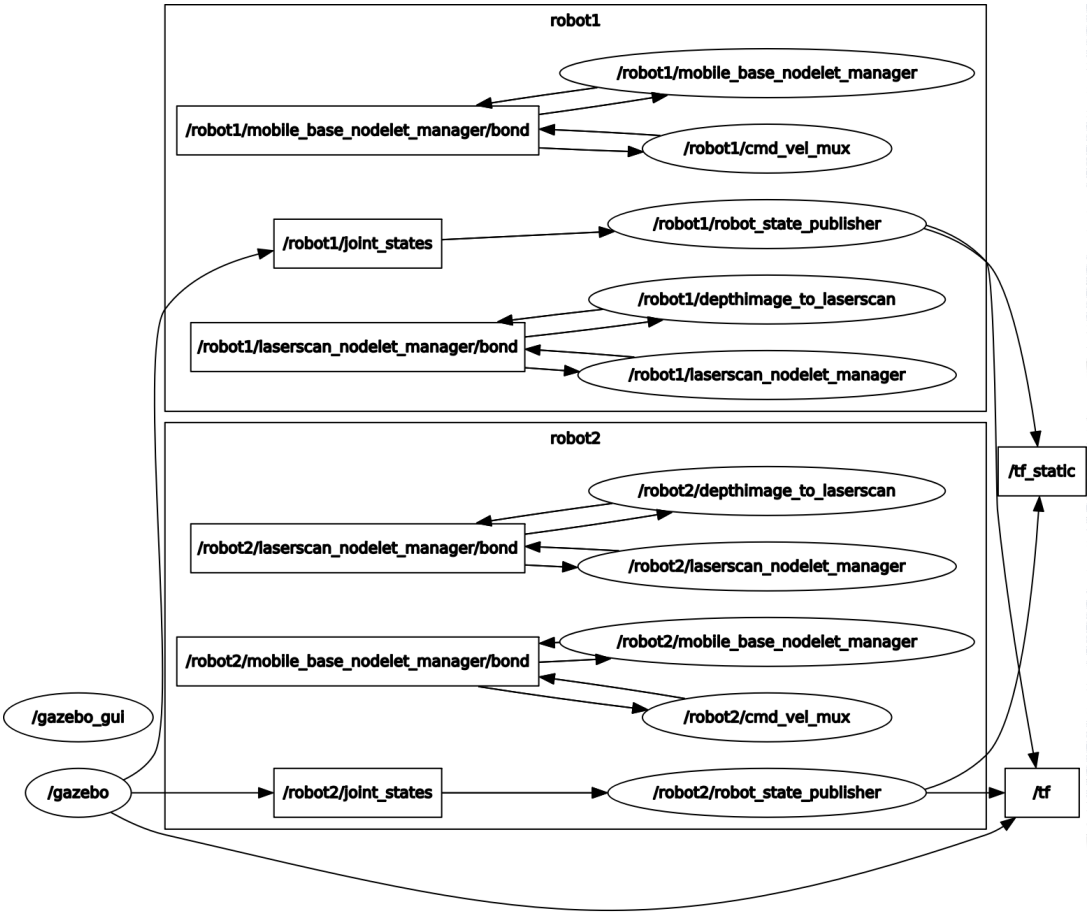


Figure 5.2: ROS graph for simulation environment shown in Figure 5.1.

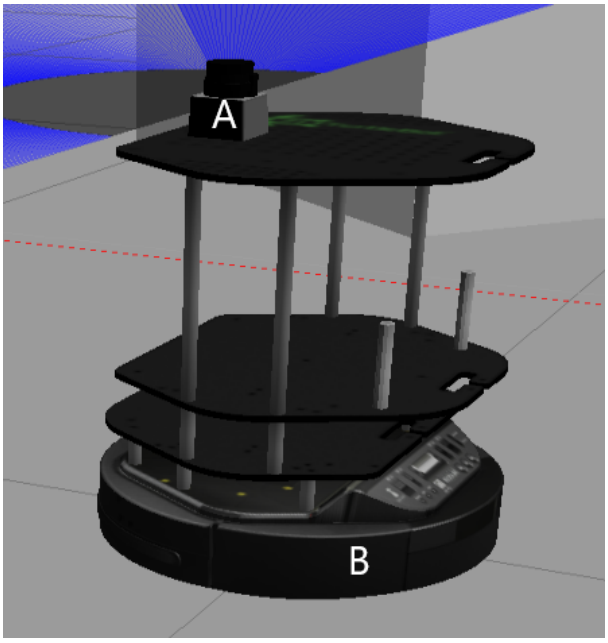


Figure 5.3: Kobuki Platform (annotated by (B)) in Gazebo environment. The Platform has a Hokuyo UTM-30LX (annotated by (A) also simulated. Note that the laser rays (blue lines) were limited to 180 degrees. This is all simulated using Gazebo.

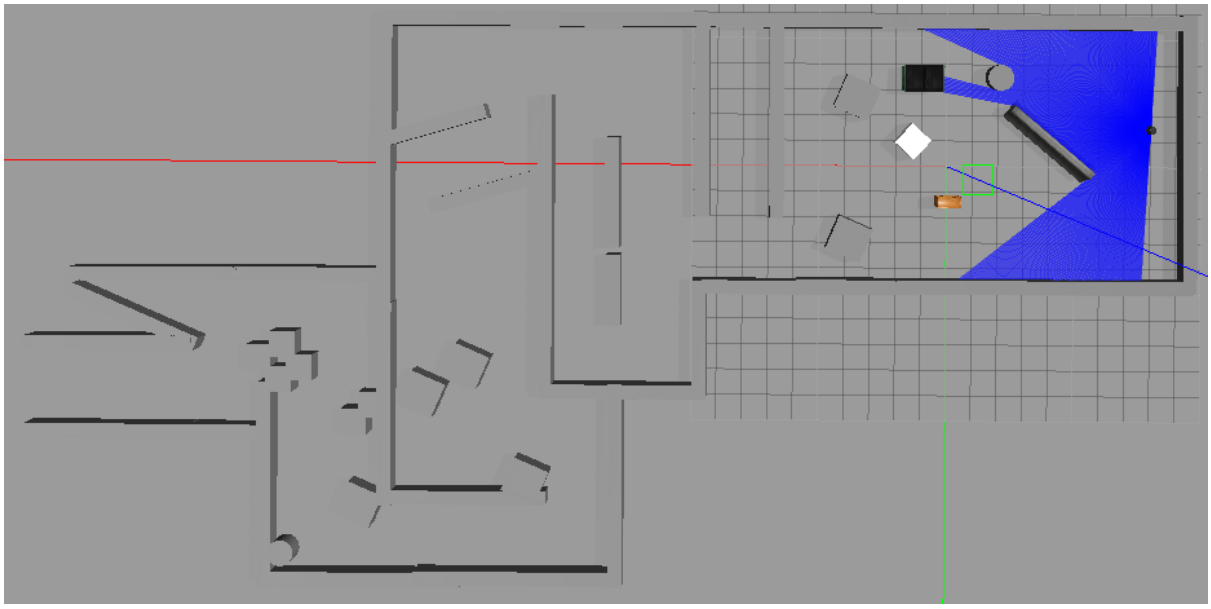


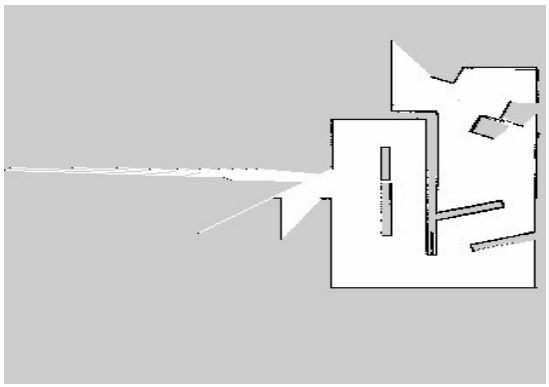
Figure 5.4: Gazebo simulation environment, with obstacle around the map. There is a Kobuki Platform, with a Hokuyo UTM-30LX attached to it, these are also simulated.

5.4.2 Results and discussion

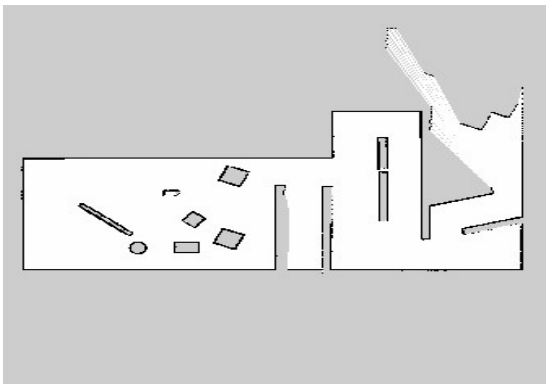
In this section, simulation results using Gazebo and ROS are presented.

5.4.2.1 Experiment one

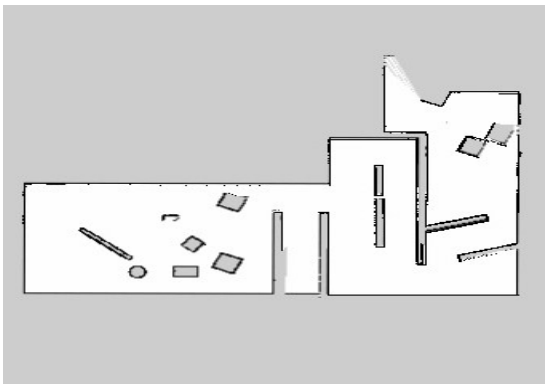
Figures 5.5a and 5.5b represent each robot's partial local map before the merging operation. The local partial maps have a resolution of 0.1 m/cell. Figure 5.5c presents the global map with a resolution of 0.1 m/cell, after merging the partial maps. The global map shows that the partial maps were adequately aligned and merged. The new resolution of partial_map_2 is within 1% of partial_map_1's resolution (shown in Table 5.1), therefore validating the success of the alignment operation.



(a) Partial map_1 with resolution of 0.1 m/cell



(b) Partial map_2 with resolution of 0.1 m/cell



(c) Global map with resolution of 0.1 m/cell

Figure 5.5: Partial maps and a global map are presented. The partial maps are generated from simulated data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1.

Figure 5.6 presents the matched features in the partial maps' overlapping area. There are 30 matched features with a match ratio of 0.46, where the percentage of overlapping area is 35.36% (see Table 5.1). The match ratio value is influenced by the fact that partial_map_2 has most of its features within the overlapping area.

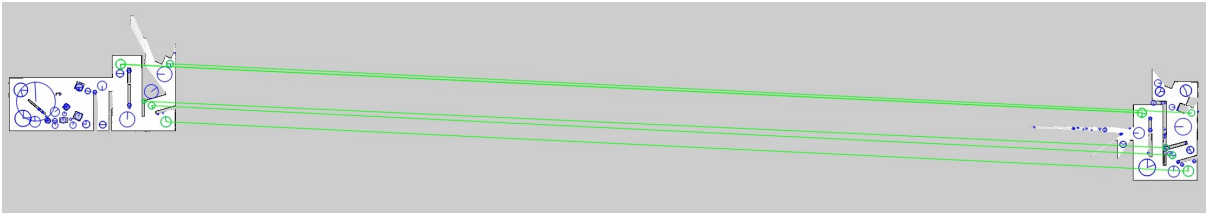


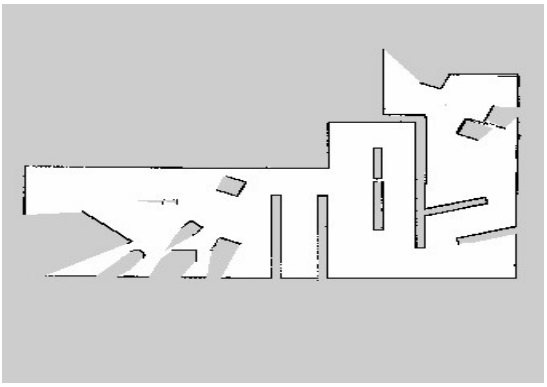
Figure 5.6: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

Table 5.1: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1, the results are relative to map_1, which has a resolution of 0.1 m/cell.

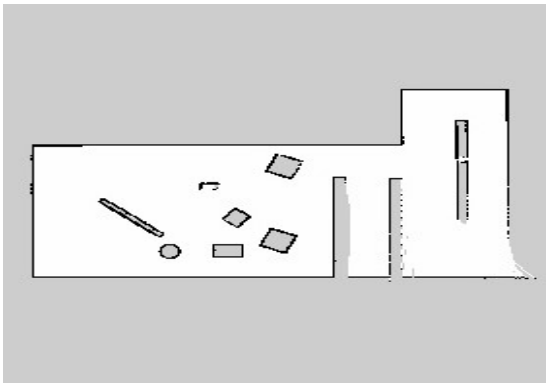
Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.09956828	-0.16716242	30	0.46	35.36

5.4.2.2 Experiment two

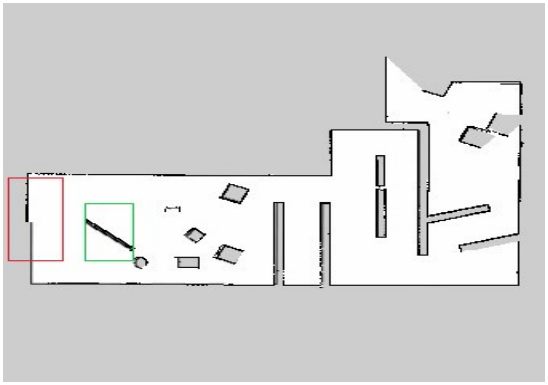
The next results still consider two robots with the same resolution partial_maps, but the maps have a bigger overlapping area of 48.03% (see Table 5.2). Figures 5.7a and 5.7b represent each robot's partial local map before the merging operation. The local partial maps have a resolution of 0.1 m/cell. Figure5.5c presents the global map of 0.1 m/cell, after the merging of the partial maps. The global map shows that the partial maps were adequately aligned and merged. The new resolution of partial_map_2 is within 1% of partial_map_1's resolution (shown in Table 5.2), therefore validating the success of the alignment operation.



(a) Partial map_1 resolution of 0.1 m/cell



(b) Partial map_2 resolution of 0.1 m/cell



(c) Global map of resolution of 0.1 m/cell

Figure 5.7: Partial maps and a global map are presented. The partial maps are generated from simulated data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1. Highlighted in green and red, are slight misalignments present in the resultant global map

Unlike results Figure 5.5, Figure 5.7 a slight misalignment which is highlighted in red and green. Figure 5.8 shows that the features in the green and red area were not matched, therefore causing the misalignment. Despite the fact that the match ratio and percentage overlap of 0.59 and 48%, are higher here (see Table 5.2) than in the previous ((see Table 5.1) results. This result suggests that it is important that the matches are evenly distributed throughout the overlapping region with a high match ratio.

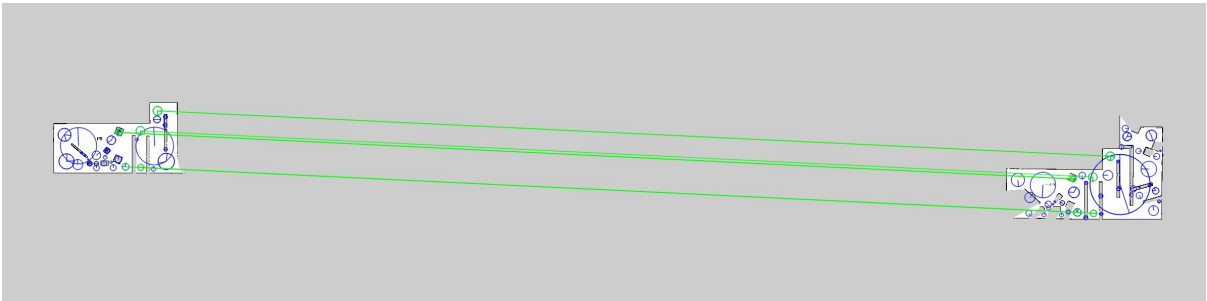


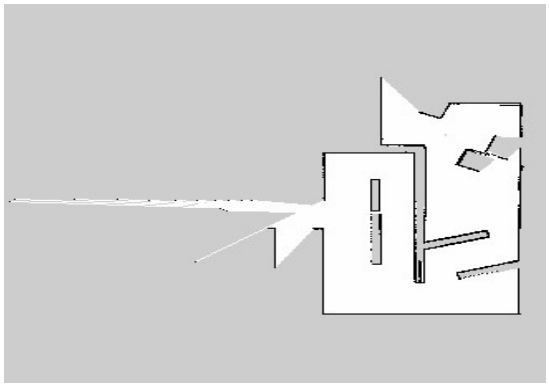
Figure 5.8: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

Table 5.2: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.1 m/cell.

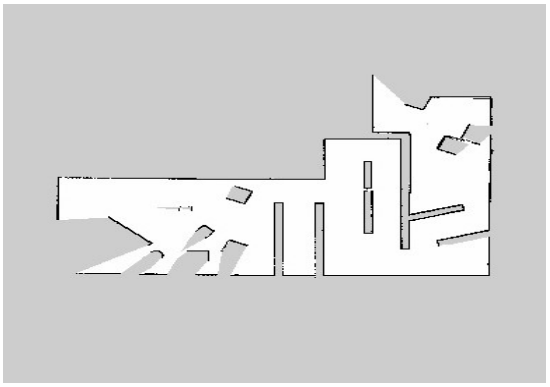
Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.10111790	-0.08469242	36	0.59	48.03

5.4.2.3 Experiment three

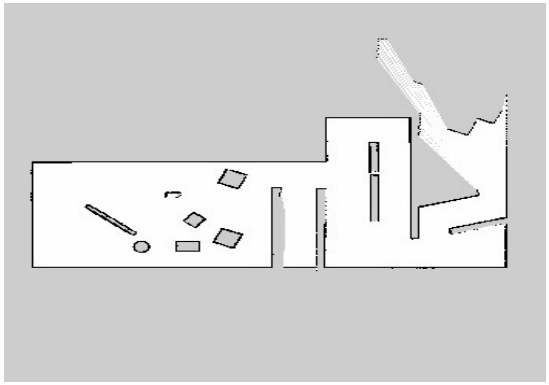
Figures 5.9a, 5.9b and 5.9c represent partial maps from three robots before the merging operation. The local partial maps have a resolution of 0.1 m/cell. The maps are to be transformed into partial_map_1's frame of reference. Figure 5.9d presents the global map of 0.1 m/cell, after merging the partial maps. The global map shows that the partial map_2 and map_3 were adequately aligned into map_1 and merged. The new resolution of partial_map_2 and partial_map_3 are both within 1% of partial_map_1's resolution (shown in Table 5.3) validating the success of the alignment operation.



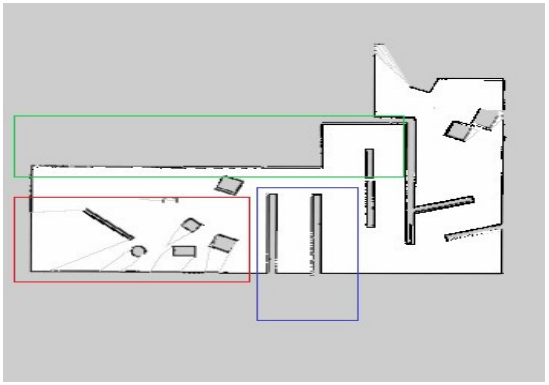
(a) Partial map_1 with resolution of 0.1 m/cell



(b) Partial map_2 with resolution of 0.1 m/cell



(c) Partial map_2 with resolution of 0.1 m/cell



(d) Global map with resolution of 0.1 m/cell

Figure 5.9: Three partial maps and a global map are presented. The partial maps are generated from simulated data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1. The red region presents lines of unknown(grey) areas; and the green and blue regions show successful incorporation of information from map_2 and map_3 in map_1

Given that the matches are mostly on the rightmost position (see figures 5.10 and 5.11) due to partial map_1, results similar to Figure 5.7 are expected. The red regions in Figure 5.9d have unknown(grey) regions, likely because map_2 and map_3 are unaware of each other. However, the blue and green region in Figure 5.9d shows information from map_2 and map_3 being successfully included in the global_map.

Table 5.3 shows match ratio results and percentage overlap, suggesting that a higher percentage produces a higher match ratio. This relationship will be evaluated in the next experiments too.

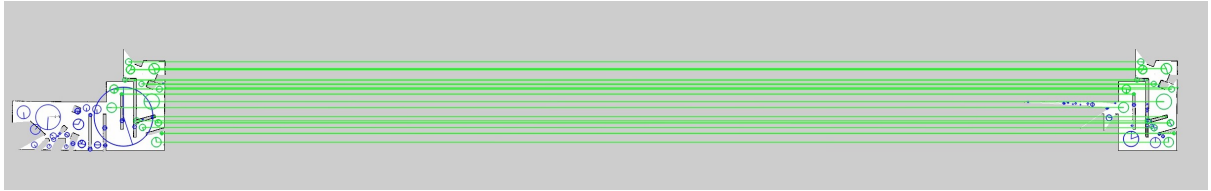


Figure 5.10: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

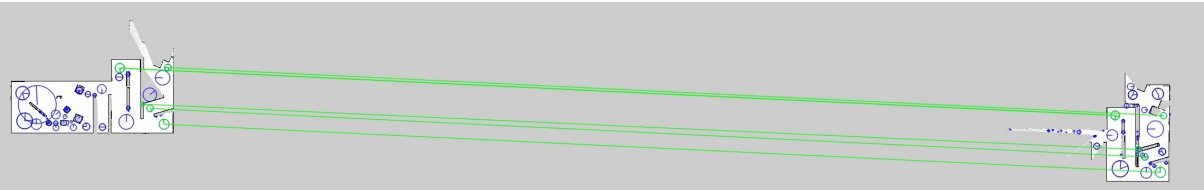


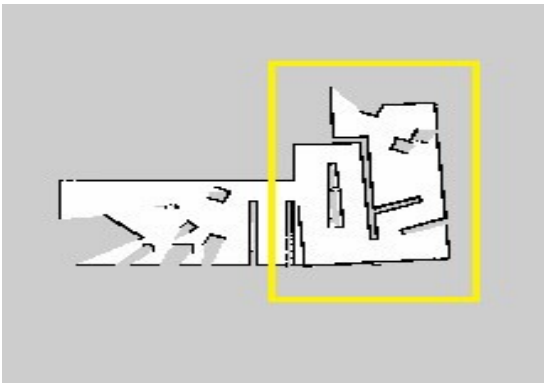
Figure 5.11: The matched features between partial map_1 and map_3 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

Table 5.3: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.1 m/cell.

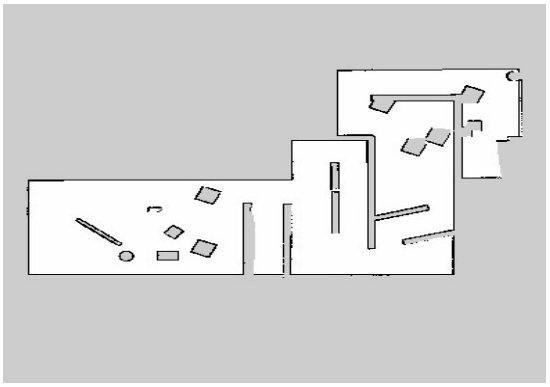
Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage Overlap
Partial map_2	0.1	0.09985444	-0.13717321	40	0.56	50.83502376
Partial map_3	0.1	0.09956828	-0.16716242	30	0.46	35.36

5.4.2.4 Experiment four

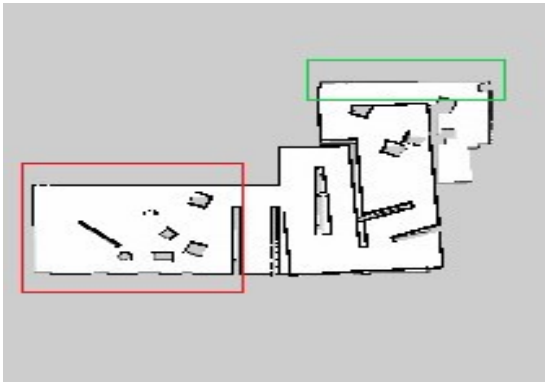
Figures 5.12a and 5.12b represents the partial maps of two robots at different resolutions (0.2 m/cell and 0.1 m/cell) before the alignment and merging operation. Partial_map_1 in Figure 5.12a, presents a mapping error highlighted in yellow, compared with original arena in Figure 5.4. The mapping error has caused the map in the yellow region to rotate slightly. Despite the error a global map was successfully produced (see Figure 5.12c). These merging defects can be seen mostly in the red and green regions of the global_map(Figure 5.12c); however the defects the information of partial_map 2 have successfully been included in the map. The new resolution of partial_map_2 is with in 1% of partial_map_1, which further validates the success of the merge (see Table 5.4).



(a) Partial map_1 with resolution of 0.2 m/cell



(b) Partial map_2 with resolution of 0.1 m/cell



(c) Global map with resolution of 0.2 m/cell

Figure 5.12: Two partial maps and a global map are presented. The partial maps are generated from simulated data, and they are of varying resolution. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1. In (a), the yellow region highlights a mapping error in map_1. In (c), the resultant global map shows the mapping error in map_1 being propagated, highlighted by the red and green regions.

Figure 5.13 represents the matches between partial_maps 1 and 2, with a match ratio of 0.66 and percentage overlap of 52.93% (see Table 5.4). Again there is evidence of a directly proportional relationship between percentage overlap and match ratio compared to the previous experiment.

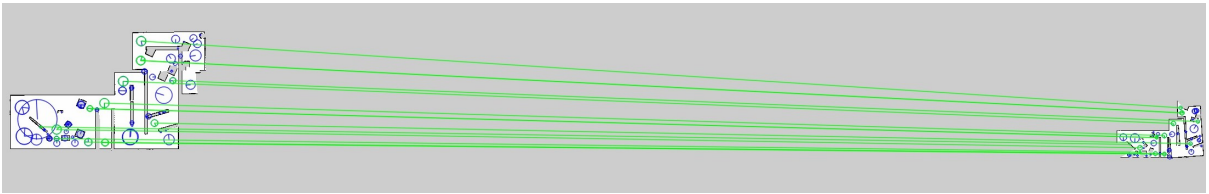


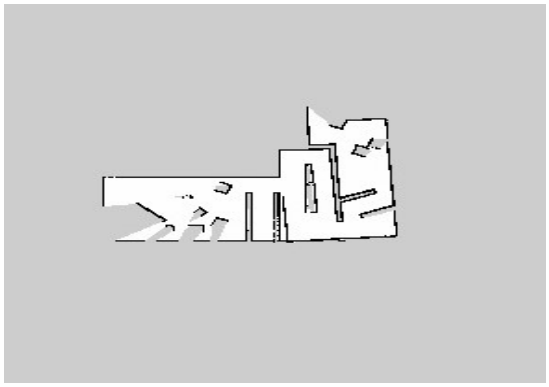
Figure 5.13: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

Table 5.4: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.2 m/cell.

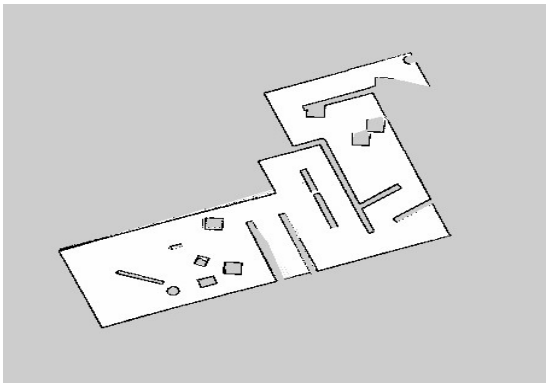
Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.20177466	0.03968788	35	0.66	52.93

5.4.2.5 Experiment five

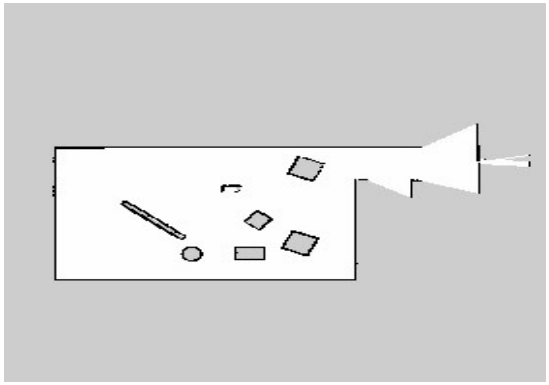
The final simulation experiment presents three partial maps aligned and merged to form a global map (see Figure 5.14). Partial map_2 and map_3 have a resolution of 0.1 m/cell, which is different from map_1's resolution of 0.2 m/cell. The global map in Figure 5.14d, is a result of the alignment and merging of maps 2 and 3 into map_1's reference frame. Similarly to Figure 5.12a, Figure 5.14a has a mapping error, which has been incorporated into the global map. However, the errors both map_2 and map_3's new resolutions are within 1% of map_1's resolutions.



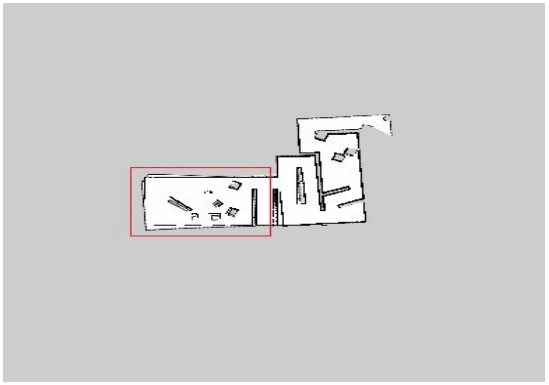
(a) Partial map_1 with resolution of 0.2 m/cell



(b) Partial map_2 with resolution of 0.1 m/cell



(c) Partial map_3 with resolution of 0.1 m/cell



(d) Global map with resolution of 0.2 m/cell

Figure 5.14: Three partial maps and a global map are presented. The partial maps are generated from simulated data, and they are of varying resolutions. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1. The mapping error similar to figure 5.12a, is propagated in the resultant global map shown in the red region.

Figure 5.15 show feature matches between map_1 and map_2, with match ratio and percentage overlap of 0.68 and 54.43 respectively (see Table 5.5). Figure 5.15 shows feature matches between map_1 and map_3, with match ratio and percentage overlap of 0.52 and 27.62 respectively (see Table 5.5). This result again confirms the proportional relationship between the match ratio and percentage overlap.

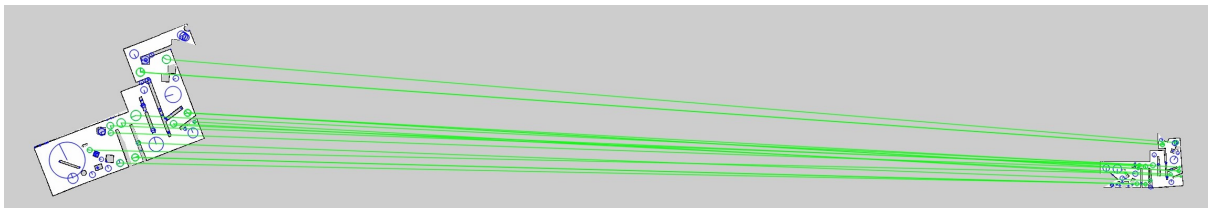


Figure 5.15: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

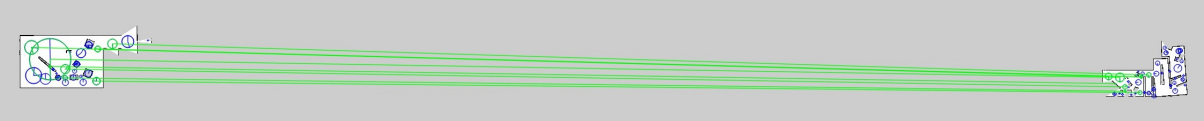


Figure 5.16: The matched features between partial map_1 and map_3 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps.

Table 5.5: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.2 m/cell.

Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.1980208	-17.8182736	36	0.68	54.43
Partial map_3	0.1	0.2019860	-2.91611928	24	0.52	27.62

5.4.2.6 Summary

The results in this section show that up-to three partial maps with different translations, orientations and resolutions can be aligned and merged successfully to form a global map. The experiments are limited to small rotation with the largest being -17.8 degrees, and only higher or same resolution partial maps are merged into a lower resolution global map. The results also show that mapping errors are propagated into the global map. The map merging operation can be limited by the matched feature spread. If matches are concentrated in one area, the merging operation will struggle to match other overlapping areas. We have also observed that the match ratio is directly proportional to the percentage overlap.

Further experiments can now be carried out to test the solution's real-world performance, using this section's results. The next section discusses the experiments carried out in the real world.

5.5 Real-world experiments

The promising results from the simulation experiments naturally suggested transferring the experiments into the real-world. This section presents a real-world experiment setup and results.

5.5.1 Environment setup

The experiments were conducted in the Leslie Social building at the University of Cape Town[3]. This building is a structured indoor environment with minimal natural lighting. Figure 5.17 is a floor plan of the fifth floor in the building, and the yellow highlight shows the path travelled by the robot. Using ROS's capability, the robot data is recorded using *rosbag*[4], and later partial maps are generated at a varying resolution for merging.

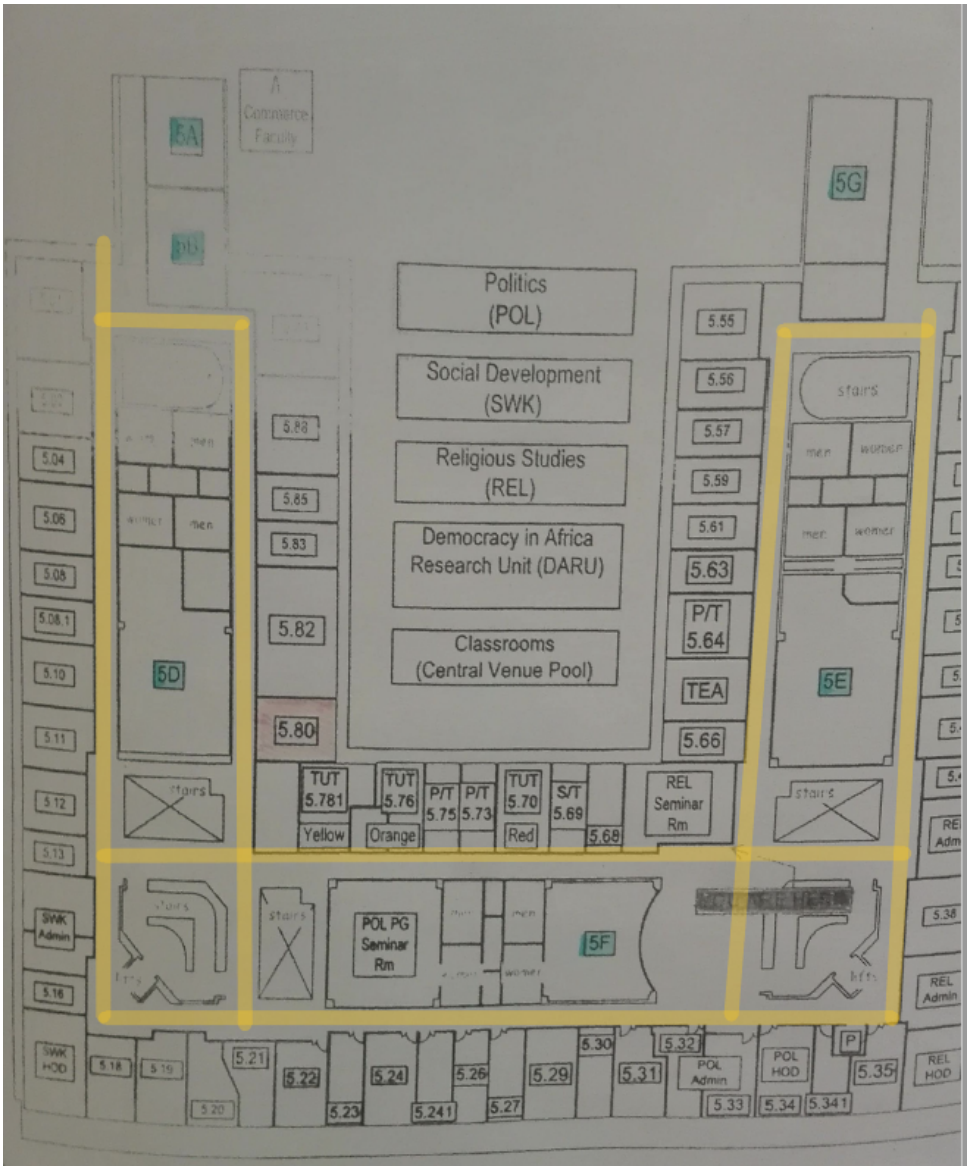


Figure 5.17: Leslie Social 5th Floor at the University of Cape Town, the yellow shows where the robot moved during the experiment. The partial maps are produced in this area.

[3]http://www.uct.ac.za/main/contacts/building-list
[4]http://wiki.ros.org/rosbag

5.5.1.1 Kuboki Platform

The chassis shown in Figure 5.18 is the iClebo Kobuki used in simulation and real-world testing. This platform is a low-cost mobile robot designed for education and research on the state of art robotics. The Kobuki provides power supplies for an external computer and any additional sensors.

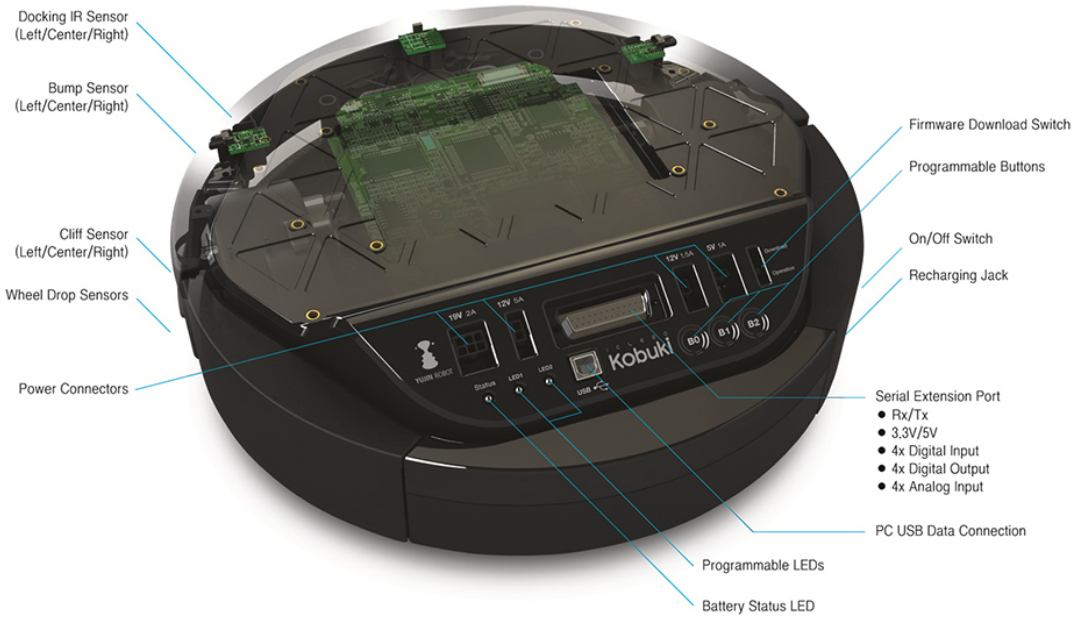


Figure 5.18: Kobuki platform without attachments. Taken from [5].

The platform in Figure 5.19, is set up with a Hokuyo UTM-30LX and Raspberry Pi 3B with ROS installed. The Kobuki platform can travel at a maximum of 70 cm/s translational velocity, and a maximum rotational velocity of 180 °/s, however above 110 °/s performance will degrade. The platform can take a maximum payload of 5 kg. The platform has safety measures to ensure safe usage, and these include a cliff detector (to avoid driving off cliffs of a depth greater than 5cm) and a climbing threshold (to avoid climbing elevations above 12 mm). The expected operating time is 3/7hours (small/large battery), and the expected charging time is 1.5/2.6 hours (small/large battery).

The Kobuki platform can be controlled using the Robot Operating System (ROS), a repository is provided on [6]. This repository also offers simulation models that can be used to simulate the platform in Gazebo or *stage_ros*.

[6]https://github.com/yujinrobot/kobuki

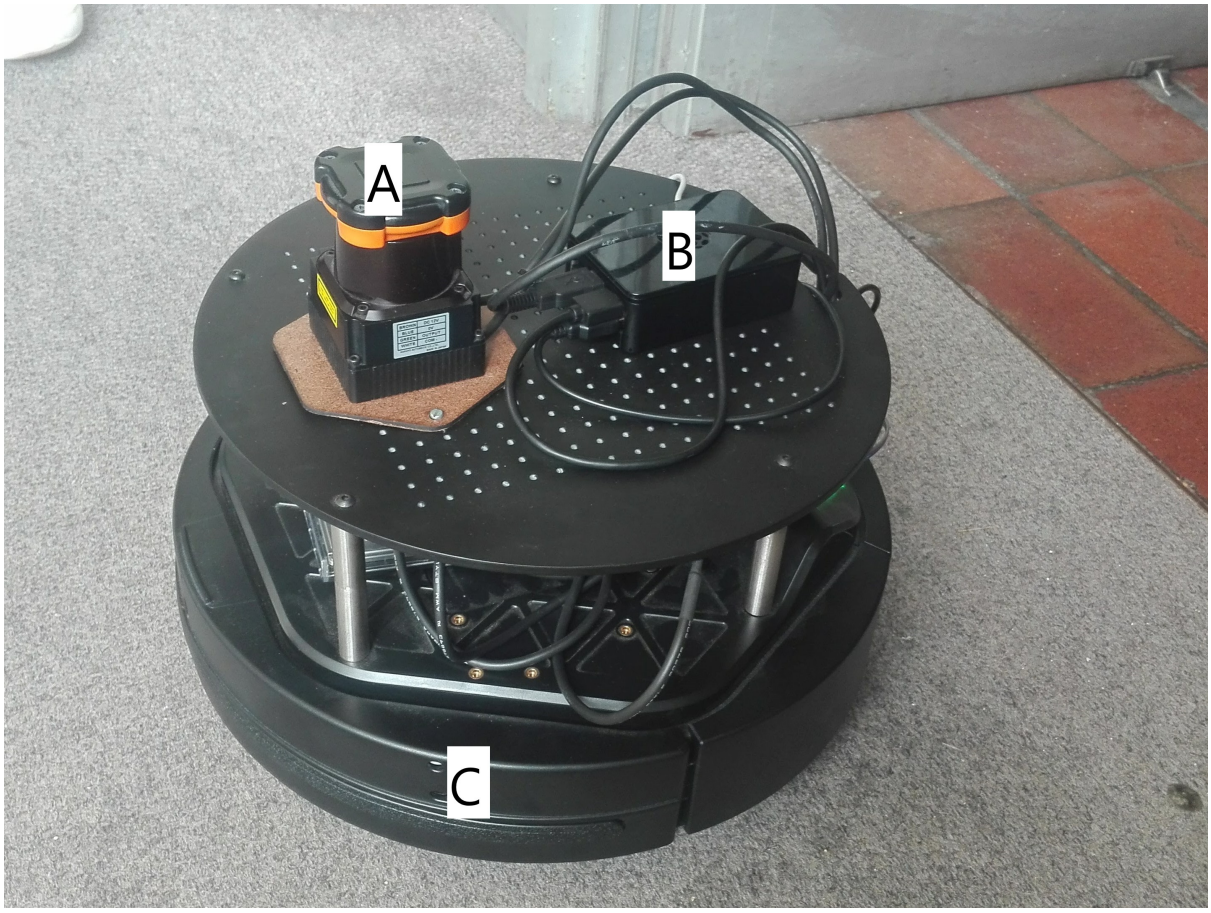


Figure 5.19: This figure shows the Kobuki Platform (C) used in the experiments. The platform is equipped with a Hokuyo UTM-30LX (A) and a Raspberry Pi 3B (B) with ROS installed.

5.5.1.2 Hokuyo UTM-30LX

Attached to the Kobuki platform is a Hokuyo UTM-30LX (see Figure 5.20), this is a dedicated laser range finder (LIDAR) used in many robotic applications. The sensor requires a 12 volts direct current power supply that can operate between 0.7 amperes and 1 amperes. This sensor uses a semiconductor laser diode with a wavelength (λ) of 905 nanometers. The Field of View (FOV) of the sensor is 270°; hence the sensor will typically be mounted on top of the robot to maximise unobstructed FOV. The sensor has an accuracy of 30 millimetres at a range of 0.1 metres to 10 metres, and an accuracy of 50 millimetres at a range of 10 metres to 30 metres. This accuracy can be less reliable if the sensor receives intense light, such as sunlight directly in the outdoor environment. The sensor has an angular resolution of 0.25°; this is the minimum angle between different objects in a scan. The sensor is equipped with a USB2.0 connector for communication with other devices.

Figure 5.20: Hokuyo UTM-30LX. Taken from [7]

5.5.1.3 System integration

The Kobuki Platform is ROS compatible, which is installed on a Raspberry Pi 3B. ROS package turtlebot_create[8] is also installed to connect to the Kuboki platform, and commands are sent through a USB data connection to the platform. To control the motion of the platform and a Logitech F710 wireless gamepad (Figure 5.21 alongside the ROS package *turtlebot_teleop*[9] are used.

Figure 5.21: The Logitech Wireless Gamepad F710 uses a powerful and reliable cable-free 2.4 GHz connection while dual-motor vibration and is used to control the Kobuki Platform.

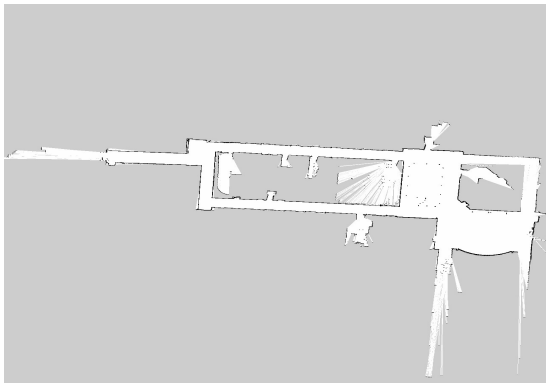
5.5.2 Results and discussion

In this section, results from real-world data are presented.

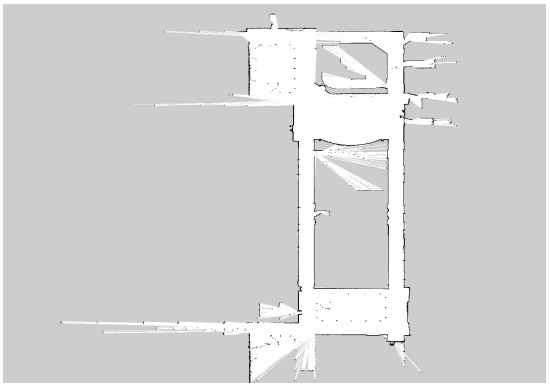
5.5.2.1 Experiment one

Figures 5.22a and 5.22b represents local partial maps from two robots. The local partial maps both have a resolution of 0.05 m/cell. Figure 5.22c presents the global map with a resolution of 0.05 m/cell after the alignment and merging of the two partial maps.

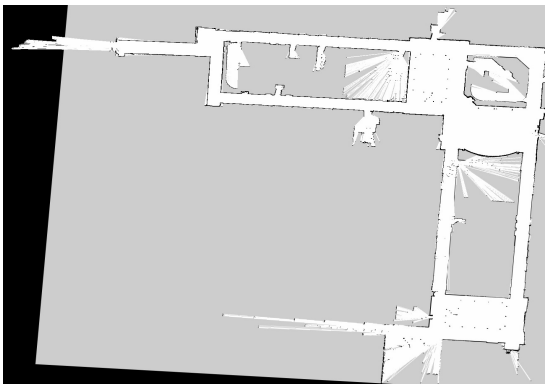
[7] Hokuyo Sensor: https://www.hokuyo-usa.com/products/scanning-laser-rangefinders/utm-30lx
[8] https://github.com/turtlebot/turtlebot_create
[9] http://wiki.ros.org/turtlebot_teleop



(a) Partial map_1 with resolution of 0.05 m/cell



(b) Partial map_2 with resolution of 0.05 m/cell



(c) Global map with a resolution of 0.05 m/cell

Figure 5.22: Two partial maps and a global map are presented. The partial maps are generated from real-world data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1.

The new resolution of partial_map_2 is within 1% of partial_map_1's resolution (shown in Table 5.6), therefore validating the success of the alignment operation. Figure 5.23 presents the matched features between the two partial maps, within the 29.78% overlap region. There are 254 matched features with a match ratio of 0.50. There are significantly more features matched here than in the simulation environment. This result is due to the differing sizes of the maps. The real-world environment is larger than the simulation environment.

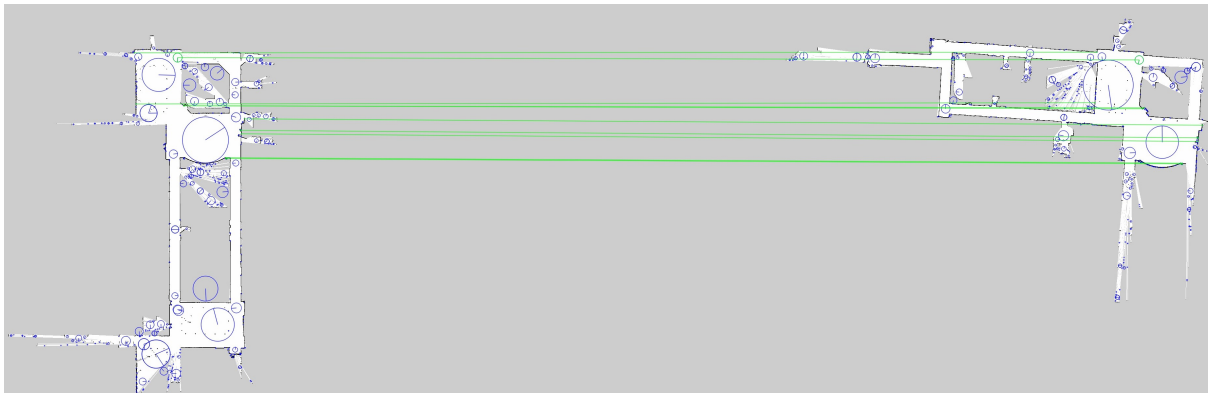


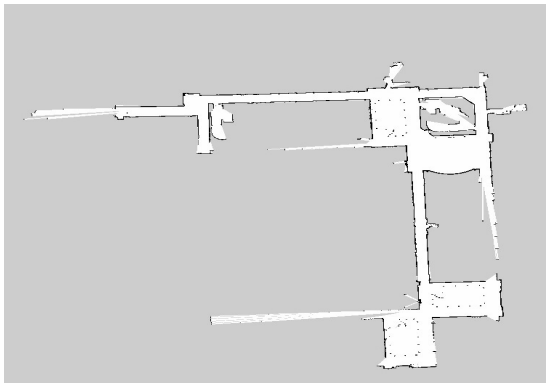
Figure 5.23: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

Table 5.6: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.05 m/cell.

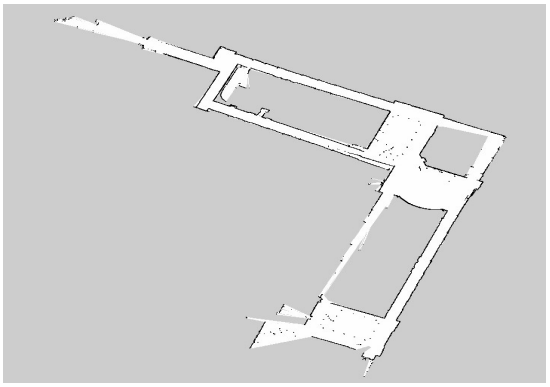
Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.05	0.049870607	-4.048134489	254	0.50	29.78

5.5.2.2 Experiment two

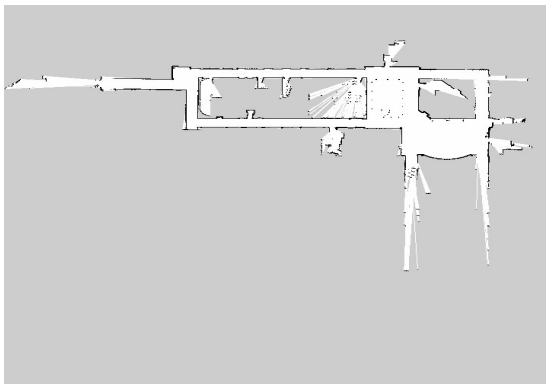
Figures 5.24a, 5.24b and 5.24c represent three partial local maps from robots, before the alignment and merging process. The partial maps have the same resolution of 0.1 m/cell. Figure 5.24d presents the global map with a resolution of 0.1 m/cell after the alignment and merging of the three partial maps. Furthermore, note that: Figure 5.24d in the red region, partial map_2 has a SLAM error which is evident in the global_map; And Figure 5.24d in green region new information from partial map_3 is now present in the global_map.



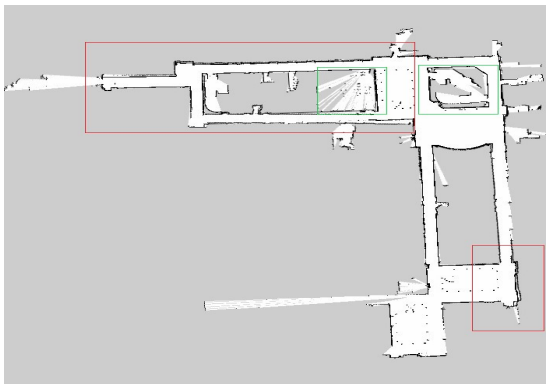
(a) Partial map_1 with a resolution of 0.1 m/cell



(b) Partial map_2 with a resolution of 0.1 m/cell



(c) Partial map_2 with a resolution of 0.1 m/cell



(d) Global map with a resolution of 0.1 m/cell

Figure 5.24: Three partial maps and a global map are presented. The partial maps are generated from real-world data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1. The green region in (d) hows information from map_2 being incorporated successfully into map_1. On the other hand, the red region in (d) shows slight misalignments due to SLAM errors.

The new resolution of partial_map_2 and partial_map_3 are within 1% of partial_map_-1's resolution (shown in Table 5.7), therefore validating the success of the alignment operation. Figure 5.25 shows the feature matches between partial map_1 and map_2, with a match ratio of 0.61 in an overlap region of 47.47% (see Table 5.7). And Figure 5.26 shows the feature matches between partial map_1 and map_3, with a match ratio of 0.52 in an overlap region of 37.83% (see Table 5.7). Again here we observe that the match ratio and percentage overlap are directly proportional.

Table 5.7: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.1 m/cell.

Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.100179351	25.60591752	107	0.61	47.47
Partial map_3	0.1	0.099236774	2.310055427	106	0.52	37.83

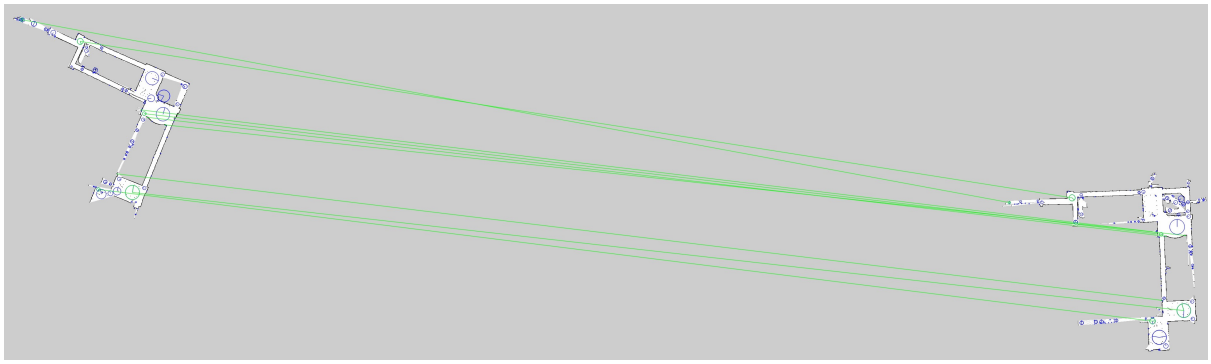


Figure 5.25: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

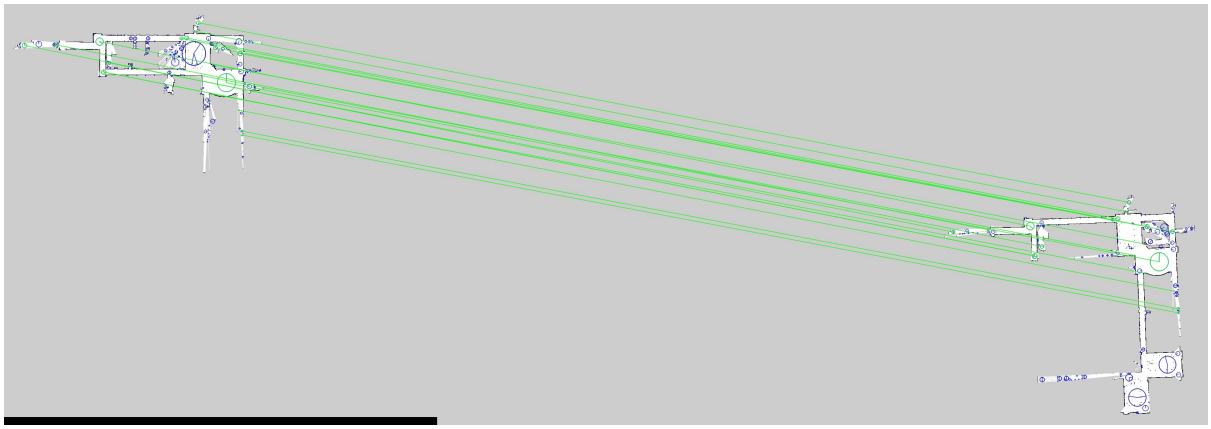
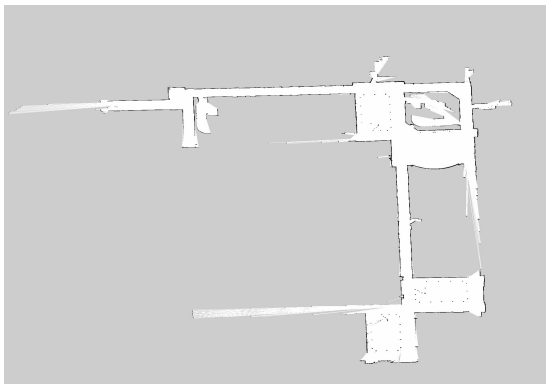


Figure 5.26: The matched features between partial map_1 and map_3 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

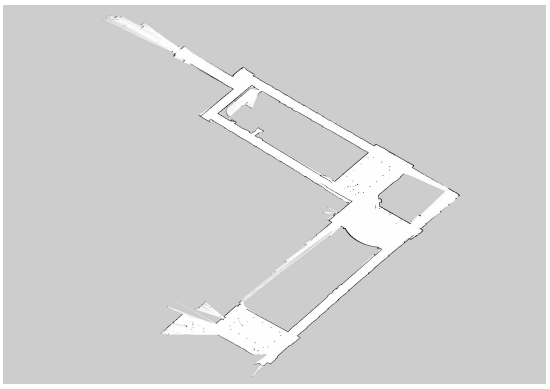
5.5.2.3 Experiment three

Figures 5.27a, 5.27b, 5.27c and 5.27d represent four partial local maps from robots, before the alignment and merging process. The partial maps 1, 2 and 3 have the same resolution

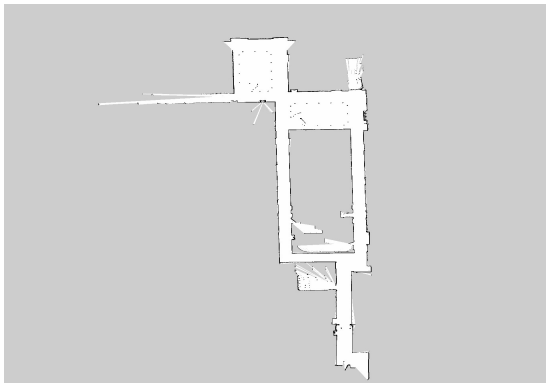
of 0.05 m/cell, and partial map_4 has a resolution of 0.1 m/cell. Figure 5.27e presents the global map with a resolution of 0.05 m/cell after the alignment and merging of the four partial maps. The global map represents the environment's full map shown in 5.17. In Figure 5.27e, the final map has two main areas of interest; green and red regions. The red region shows the propagated SLAM error from partial map_2 (see Figure 5.27b). The green region shows a misalignment of the partial maps, potentially caused by the SLAM error in partial map_2.



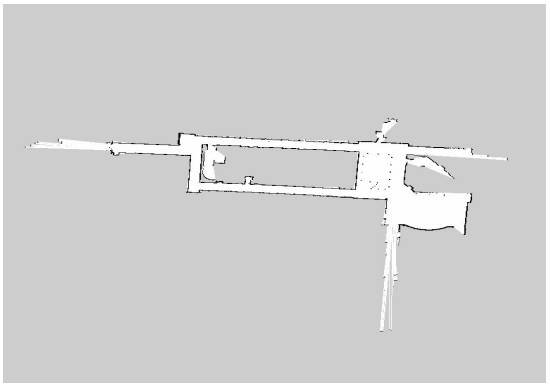
(a) Partial map_1 with resolution of 0.05 m/cell



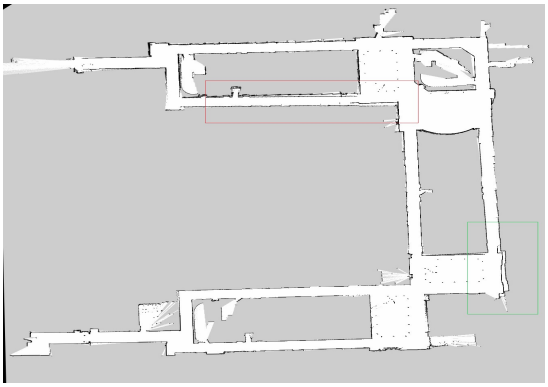
(b) Partial map_2 with resolution of 0.05 m/cell



(c) Partial map_3 with resolution of 0.05 m/cell



(d) Partial map_3 with resolution of 0.1 m/cell



(e) Global map with resolution of 0.05 m/cell

Figure 5.27: Four partial maps and a global map are presented. The partial maps are generated from real-world data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1.

The new resolution of partial_map_2, partial_map_3 and partial_map_4 are with in 1% of

partial_map_1's resolution (shown in Table 5.8), therefore validating the success of the alignment operation. Figure 5.28 shows the feature matches between partial map_1 and map_2, with a match ratio of 0.49 in an overlap region of 51.60% (see Table 5.8). Figure 5.29 shows the feature matches between partial map_1 and map_3, with a match ratio of 0.55 in an overlap region of 23.71% (see Table 5.8). Figure 5.29 shows the feature matches between partial map_1 and map_4, with a match ratio of 0.91 in an overlap region of 29.37% (see Table 5.8). Here we see that the direct proportionality of match ratio and percentage overlap do not hold.

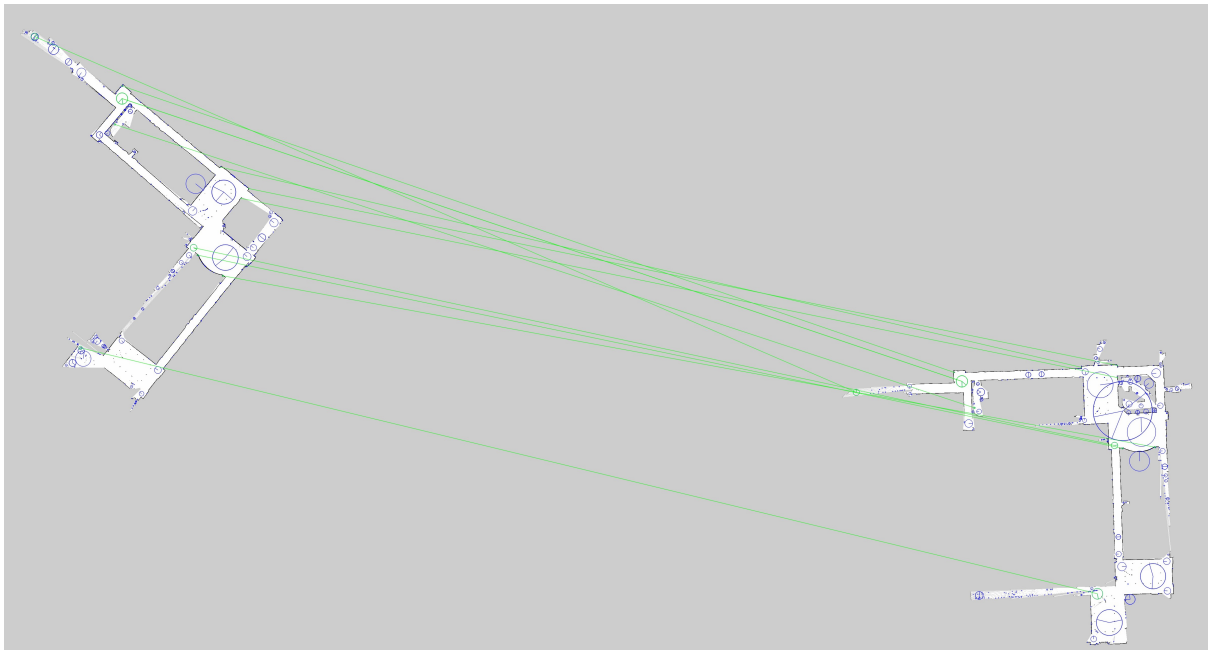


Figure 5.28: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

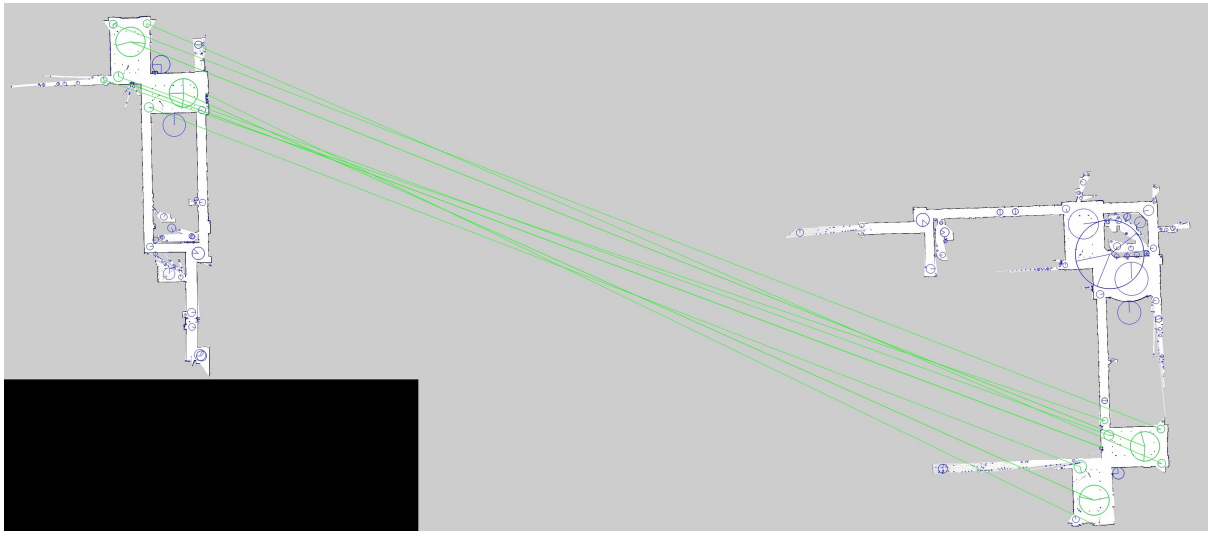


Figure 5.29: The matched features between partial map_1 and map_3 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

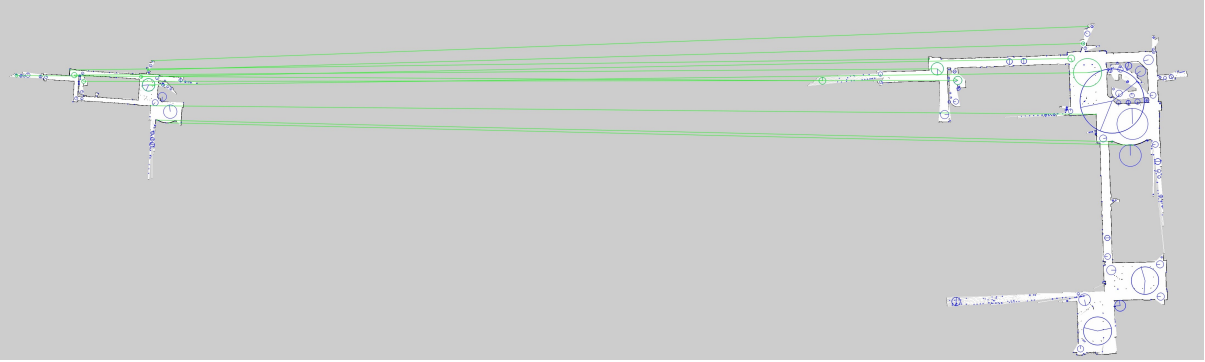


Figure 5.30: The matched features between partial map_1 and map_4 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

Table 5.8: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.05 m/cell.

Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.05	0.05014256	41.6208106	242	0.49	51.60
Partial map_3	0.05	0.05007269	-89.0525206	201	0.55	23.71
Partial map_4	0.1	0.05011043	5.67170818	115	0.91	39.37

5.5.2.4 Summary

Overall the real-world experiments show that the solution can handle real-world scenarios. Hence partial maps of varying resolution, translation and orientation; can be aligned and merged. Furthermore, this section shows that the match ratio and percentage overlap are not directly proportional, as observed before. The next section will further test the solutions ability to handle partial maps generated from publicly available data.

5.6 MIT Stata Center experiments

The solution is further tested using a publicly available data-set, in this section.

5.6.1 Environment setup

The data-set is presented in Section5.6.1, and this data is accessed using the capability of ROS *rosbag*. The Hokuyo UTM-30LX Laser scan data is used as an input to *Hector_Mapping* to generate the partial maps at varying resolutions. The Massachusetts Institute of Technology (MIT) Stata Center Data Set is a vast scale data-set collected between 2011 and 2013. The academic building is a size of 67,000 m^2 and contains 10 storeys [116]. The collected data-set comprises of over 2.3TB, 38 hours and 42 km of sensor data, and can be accessed on the university website[10]. This set is of particular interest to robotics and computer vision researchers; due to the following rich sensor data:

Sensor	Frequency
Hokuyo UTM-30LX Laser	40 Hz
Wheel Odometry [Raw]	44 Hz
Wheel Odometry [Integrated]	25 Hz
Microstrain 3DM-GX2 IMU	10 Hz
Willow Garage Stereo	30 Hz
Microsoft Kinect	30 Hz

The data were recorded using *rosbag* on ROS; therefore, it can be played back and used to produce partial maps in this thesis. To test the solution, the *hector_mapping* ROS node is run using Hokuyo UTM-30LX Laser sensor data to produced partial maps at varying resolution.

5.6.2 Results and discussion

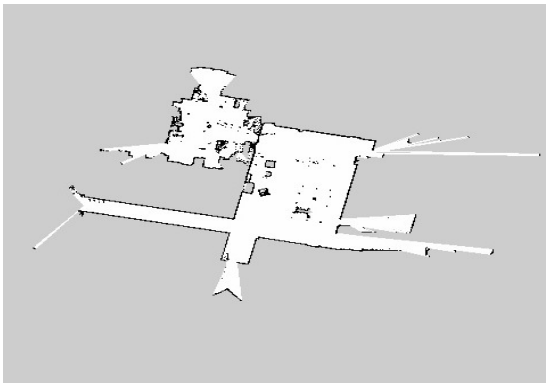
In this section, results from the MIT Stata Center data are presented.

5.6.2.1 Experiment one

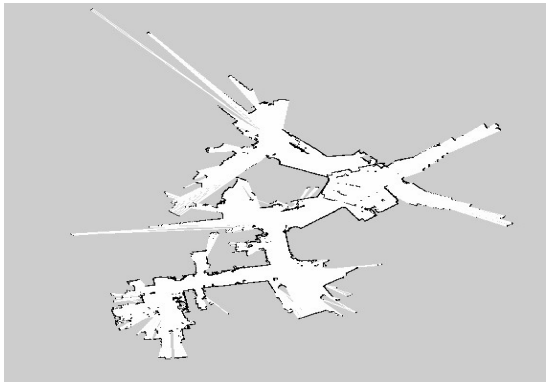
The initial MIT experiment is Figure 5.31 considers partial maps of the same resolution of 0.1 m/cell. Figures 5.31a, 5.31b and 5.31c presented the partial maps before the alignment and merging operation. Figure 5.31d presents the global map after alignment and merging of the three partial maps. The new resolution of partial_map_2 and partial_map_3 are within 1% of partial_map_1's resolution (shown in Table 5.9), therefore validating the success of the alignment operation.

[10]https://projects.csail.mit.edu/stata/

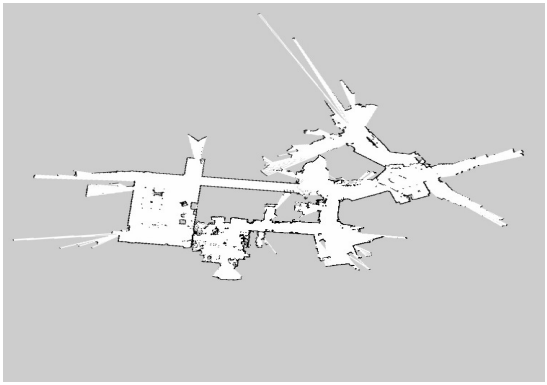
(a) Partial map_1 resolution of 0.1 m/cell



(b) Partial map_2 with resolution of 0.1 m/cell



(c) Partial map_3 with resolution of 0.1 m/cell



(d) Final map with resolution of 0.1 m/cell

Figure 5.31: Three partial maps and a global map are presented. The partial maps are generated from publicly available data. Since the maps are aligned relative to map1, the resultant global map will have the same resolution as partial map_1

Figure 5.32 shows the feature matches between partial map_1 and map_2, with a match ratio of 0.55 in an overlap region of 11.08% (see Table 5.9). The overlap percentage of 11.08% is the lowest value observed so far. This percentage overlap suggests that the map merging algorithms performance is dependent on the match ratio rather than the percentage of overlap. Figure 5.33 shows the feature matches between partial map_1 and map_3, with a match ratio of 0.70 in an overlap region of 29.03% (see Table 5.9).

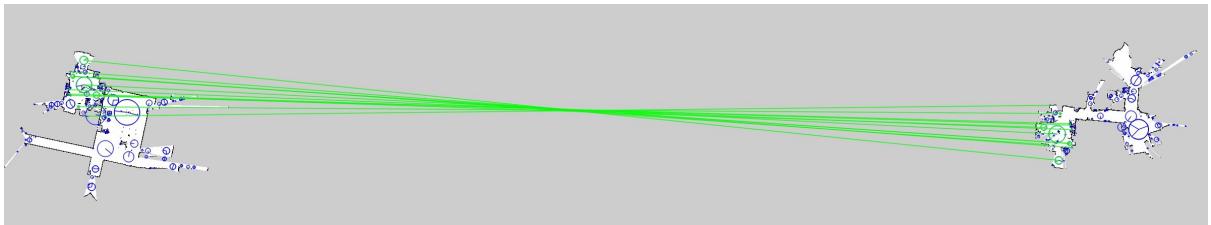


Figure 5.32: The matched features between partial map1 and map2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

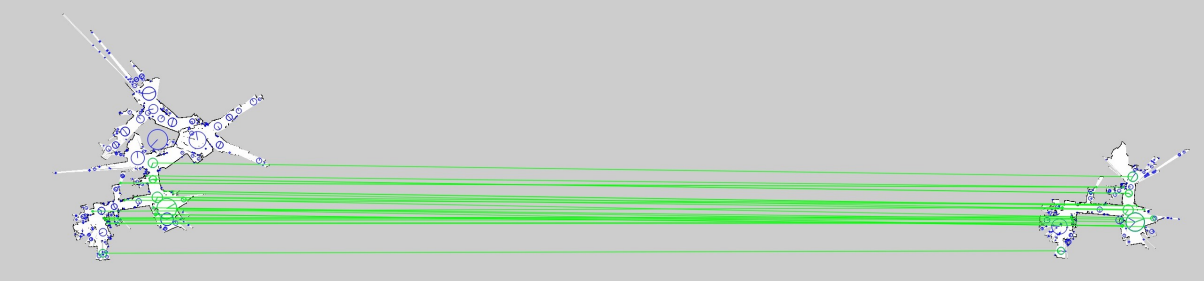


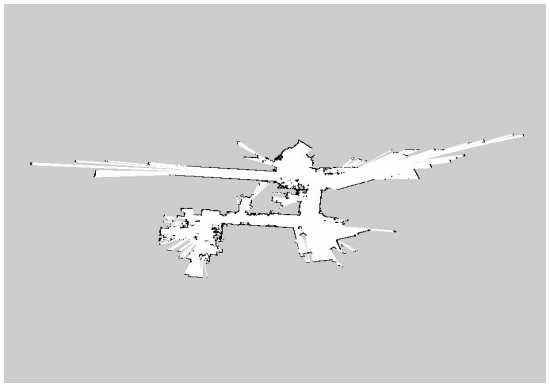
Figure 5.33: The matched features between partial map_1 and map_3 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

Table 5.9: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map_1 the results are relative to map_1, which has a resolution of 0.1 m/cell.

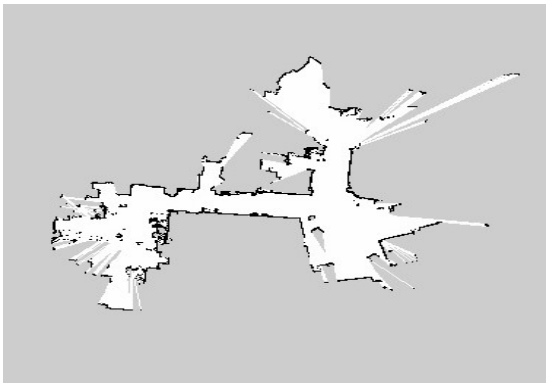
Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good Matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.09972303	-174.1462356	92	0.55	11.08
Partial map_3	0.1	0.10033173	-14.49865513	135	0.70	29.03

5.6.2.2 Experiment two

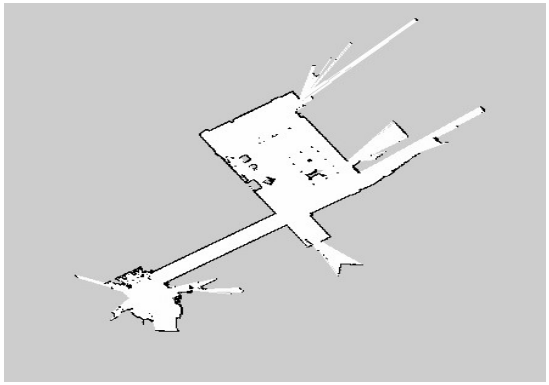
Figures 5.34a, 5.34b and 5.34c presented the partial maps before the alignment and merging operation, with resolutions of 0.1 m/cell, 0.1 m/cell and 0.11 m/cell respectively . Figure 5.34d presents the global map after alignment and merging of the three partial maps. The new resolution of partial_map_2 and partial_map_3 are within 1% of partial_map_1's resolution (shown in Table 5.10), therefore validating the success of the alignment operation.



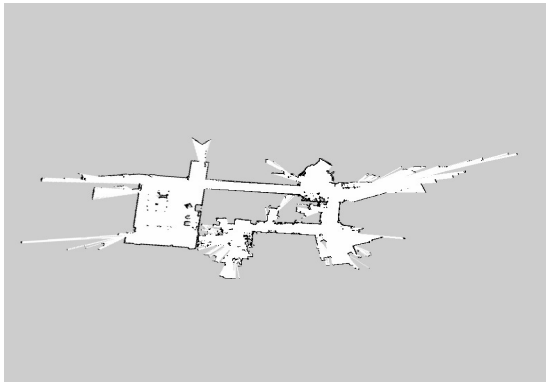
(a) Partial map_1 with resolution of 0.1 m/cell



(b) Partial map_2 with resolution of 0.1 m/cell



(c) Partial map_3 resolution of 0.11 m/cell



(d) Final map with resolution of 0.1 m/cell

Figure 5.34: Three partial maps and a global map are presented. The partial maps are generated from publicly available data. Since the maps are aligned relative to map_1, the resultant global map will have the same resolution as partial map_1

Figure 5.35 shows the feature matches between partial map_1 and map_2, with a match ratio of 0.70 in an overlap region of 39.63% (see Table 5.10). Figure 5.36 shows the feature matches between partial map_1 and map_3, with a match ratio of 0.85 in an overlap region of 13.59% (see Table 5.10). Here we observe another low percentage overlap of 13.59% with a high match ratio of 0.85.

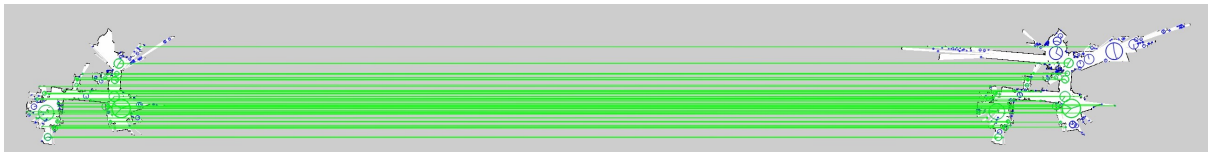


Figure 5.35: The matched features between partial map_1 and map_2 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

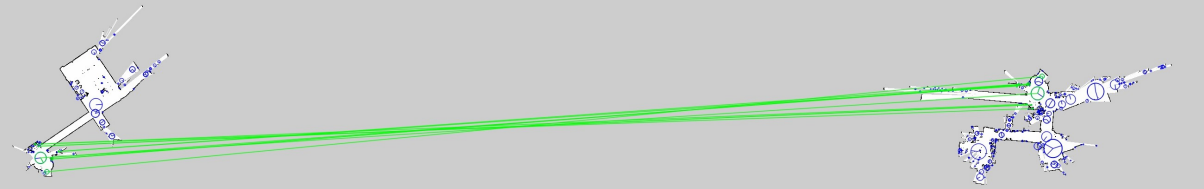


Figure 5.36: The matched features between partial map_1 and map_3 are presented in this figure. The features highlighted in green, are the features that have been matched between the two maps

Table 5.10: This table presents the results from the alignment and merging operation. Since the partial maps are merged relatively to map1 the results are relative to map1 which has a resolution of 0.1 m/cell.

Map	Resolution (m/cell)	New resolution (m/cell)	Angle of rotation (°)	Good matches	Match ratio	Percentage overlap
Partial map_2	0.1	0.100164329	-0.010020086	133	0.70	39.63
Partial map_3	0.11	0.105664826	139.9271825	94	0.85	13.59

5.6.2.3 Summary

This section shows that the solution can align and merge partial maps from other real-world data-sets produced by other authors. This result benchmarks the solution with other similar approaches. The results further demonstrate the algorithm ability to merge maps with a percentage overlap as low as 11.08%.

5.7 Summary

This chapter has tested the solution presented in Chapter 4. We found its performance acceptable, dealing with simulated and real-world scenarios. The technique can successfully align and merge maps gathered from multiple robots, with partial maps of different orientation, translation and resolution.

In the next chapter, we will reflect upon the work done in this dissertation and analyse the results in light of the requirements in Chapter 1. We will also explore the possible further work that can be done to expand on the work.

Chapter 6

Conclusion

In the previous chapter, results obtained in simulation and real-world experiments were presented and discussed. This chapter will present the conclusions based on this discussion. Also, suggestions for future research are made.

6.1 Approach overview

This thesis presents, a distributed, multi-robot SLAM solution. The work's primary focus is a map merging algorithm that merges partial maps to form a global map. Partial maps are standard; for example, multiple robots exploring an environment or a single robot map exploring multiple mapping sessions starting from different locations. Therefore the map merging algorithm in this work aims to merge occupancy grid maps from multiple robots, multiple sessions and at varying resolutions.

In this thesis a distributed approach implemented on ROS was presented, where each robot: can produce a local map and share it with other robots, can retrieve local maps from other robots in its network, and can produce a global map from its local map, previously mapped sessions and local maps from other robots.

Each robot was equipped with Hector SLAM for mapping and localisation. Hector SLAM was compared to Graph SLAM, EKF SLAM and Fast SLAM, and was selected because it can operate without odometry data and deals better with loop closure.

Experiments were conducted both in simulation and real-world environments. The real-world experiments were conducted on data produced for this work and a publicly available data-set. The results demonstrated that multi-robot, multi-session, multi-resolution map merging could be addressed using an image registration based algorithm. Features are extracted using SIFT from the maps then matched to find the relative transformation between maps. SIFT is orientation, translation and scale invariant, and when compared to alternative techniques, it was observed to be optimal for use on occupancy grid maps.

Results suggest that the match ratio between two maps is more significant than region overlap when matching two maps. It was observed that maps with an overlapping region

of as low as 11.08%, yield successful results.

There are, however, limitations identified, such as low resolution partial maps affect the quality of the global map. Alignment and merging are performed from multiple map sources, and map quality is not assessed. We assume that maps presented to the merging algorithm have a reasonable level of quality. Therefore, low quality is not rejected, and the results show that mapping errors are propagated into the global map.

6.2 Future work

Although the proposed solution performs well in structured and low noise environment, there some issues left open and correspond to future guidelines that can be followed to improve the work.

The maximum iterations described in equation 4.3, have been constraint to two minimum number of matched features to reduce computational requirements of the approach. Future work should consider four minimum number of matched features to account for the four degrees of freedom (x, y, orientation, and scale).

Currently, the algorithm continuously computes the transformation between maps on each merge cycle. In the future, to reduce computation time, the previously computed transformation can be used instead of recalculating. Also, noted in the results, was the algorithm's inability to reject or correct maps which have SLAM errors. This issue can cause problems such as incorrect navigation and path planning. Therefore the algorithm must be extended to reject or correct faulty maps.

This work has focused on two-dimensional maps. Although two-dimensional maps are still routinely used for navigation and path planning in three-dimensional SLAM algorithms, three-dimensional maps are becoming more common in robotics. In this case, a hybrid solution capable of working with two-dimensional and three-dimensional maps can be developed to extend the work in this thesis.

Bibliography

[1] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[2] R. Siegwart and I. R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*. USA: Bradford Company, 2004.

[3] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[4] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): part II," *IEEE Robotics Automation Magazine*, vol. 13, no. 3, pp. 108–117, 2006.

[5] K. Y. K. Leung, T. D. Barfoot, and H. H. T. Liu, "Decentralized cooperative simultaneous localization and mapping for dynamic and sparse robot networks," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3554–3561, 2010.

[6] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[7] S. Sukkarieh, E. M. Nebot, and H. F. Durrant-whyte, "A high integrity IMU/GPS navigation loop for autonomous land vehicle applications," vol. 15, no. 3, pp. 572–578, 2006.

[8] S. Kim, C. Roh, S. Kang, and M. Park, "Outdoor navigation of a mobile robot using differential gps and curb detection," in *2007 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3414–3419, 2007.

[9] W. Sheng, Q. Yang, J. Tan, and N. Xi, "Distributed multi-robot coordination in area exploration," *Robotics and Autonomous Systems*, vol. 54, no. 12, pp. 945–955, 2006.

[10] A. Howard, "Multi-robot mapping using manifold representations," in *2004 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, pp. 4198–4203 Vol.4, 2004.

[11] S. Carpin and G. Pillonetto, "Motion planning using adaptive random walks," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 543–548, 2005.

[12] M. Xin, G. Rui, L. Yibin, and C. Weidong, "Adaptive genetic algorithm for occupancy grid maps," *Intelligent Control and Automation*, pp. 5716–5721, 2008.

[13] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Autonomous Robots*, vol. 25, no. 3, pp. 305–316, 2008.

[14] S. Saeedi, L. Paull, M. Trentini, M. Seto, and H. Li, "Map merging for multiple robots using Hough peak matching," *Robotics and Autonomous Systems*, vol. 62, no. 10, pp. 1408–1424, 2014.

[15] S. Saeedi, L. Paull, M. Trentini, M. Seto, and H. Li, "Efficient map merging using a probabilistic generalized Voronoi diagram," *IEEE International Conference on Intelligent Robots and Systems*, pp. 4419–4424, 2012.

[16] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pp. 155–160, 2011.

[17] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag, 2007.

[18] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[19] J. Nieto, J. Guivant, E. Nebot, and S. Thrun, "Real time data association for fastslam," in *2003 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 412–418 vol.1, 2003.

[20] M. Labbé and F. Michaud, "Online global loop closure detection for large-scale multi-session graph-based SLAM," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2661–2666, 2014.

[21] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *Int. J. Rob. Res.*, vol. 25, p. 1181–1203, Dec. 2006.

[22] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. The MIT Press, 2005.

[23] M. Meyer, *SLAM Algorithms In Dynamic Environments*. PhD thesis, Technische Universitat Munchen, 2016.

[24] J. Li, L. Cheng, H. Wu, L. Xiong, and D. Wang, "An overview of the simultaneous localization and mapping on mobile robot," *Proceedings of 2012 International Conference on Modelling, Identification and Control*, pp. 358–364, 2012.

[25] Y. Chen, "Algorithms for simultaneous localization and mapping," 2013.

[26] Z. Kurt-Yavuz and S. Yavuz, "A comparison of EKF, UKF, FastSLAM2.0, and UKF-based FastSLAM algorithms," *INES 2012 - IEEE 16th International Conference on Intelligent Engineering Systems, Proceedings*, pp. 37–43, 2012.

[27] J. A. Castellanos, J. Neira, and J. D. Tardós, "Limits to the consistency of EKF-based SLAM," *IFAC Proceedings Volumes*, vol. 37, no. 8, pp. 716 – 721, 2004. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

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

[29] J. A. Castellanos, R. Martinez-Cantin, J. D. Tardós, and J. Neira, "Robocentric map joining: Improving the consistency of EKF-SLAM," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 21–29, 2007.

[30] B. Williams and I. Reid, "On combining visual SLAM and visual odometry," in *2010 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3494–3500, 2010.

[31] J. Civera, O. G. Grasa, A. J. Davison, and J. M. Montiel, "1-Point RANSAC for extended Kalman filtering: Application to real-time structure from motion and visual odometry," *Journal of Field Robotics*, vol. 27, no. 5, pp. 609–631, 2010.

[32] M. Balcilar, E. Uslu, F. Cakmak, N. Altuntas, S. Marangoz, M. Fatih Amasyali, and S. Yavuz, "An architecture for multi-robot hector mapping," *Proceedings of the 2016 International Symposium on Innovations in Intelligent Systems and Applications, INISTA 2016*, pp. 1–5, 2016.

[33] J. Neira and J. D. Tardos, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 890–897, 2001.

[34] L. M. Paz, J. D. Tardos, and J. Neira, "Divide and conquer: EKF SLAM in $O(n)$," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, 2008.

[35] V. Bonato, E. Marques, and G. A. Constantinides, "A floating-point extended kalman filter implementation for autonomous mobile robots," in *2007 International Conference on Field Programmable Logic and Applications*, pp. 576–579, 2007.

[36] J. J. Leonard and H. J. S. Feder, "A computationally efficient method for large-scale concurrent mapping and localization," in *Robotics Research* (J. M. Hollerbach and D. E. Koditschek, eds.), (London), pp. 169–176, Springer London, 2000.

[37] J. Knight, A. Davison, and I. Reid, "Towards constant time SLAM using post-ponement," in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, pp. 405–413 vol.1, 2001.

[38] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, 2001.

[39] T. A. Vidal-Calleja, C. Berger, J. Solà, and S. Lacroix, "Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain," *Robotics and Autonomous Systems*, vol. 59, no. 9, pp. 654 – 674, 2011.

[40] A. Quraishi, T. Cieslewski, S. Lynen, and R. Siegwart, "Robustness to connectivity loss for collaborative mapping," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2016-Novem, pp. 4580–4585, 2016.

[41] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," *Proc. of 8th National Conference on Artificial Intelligence/14th Conference on Innovative Applications of Artificial Intelligence*, vol. 68, no. 2, pp. 593–598, 2002.

[42] S. Thrun, "Robotic mapping: A survey," *Science*, vol. 298, pp. 1–35, 2002.

[43] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, *FastSLAM: A factored solution to the simultaneous localization and mapping problem with unknown data association*. PhD thesis, School of Computer Science Carnegie Mellon University Pittsburgh, 2003.

[44] A. Eliazar and R. Parr, "DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks.," in *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1135–1142, 2003.

[45] S. Thrun, M. Montemerlo, D. Koller, B. Wegbreit, J. Nieto, and E. Nebot, "FastSLAM: An efficient solution to the simultaneous localization and mapping problem with unknown data association," *Journal of Machine Learning Research*, vol. 4, no. 3, pp. 380–407, 2004.

[46] T. Bailey, J. Nieto, and E. Nebot, "Consistency of the FastSLAM algorithm," in *2006 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2006, pp. 424–429, 2006.

[47] C. Stachniss, D. Hahnel, and W. Burgard, "Exploration with active loop-closing for FastSLAM," *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, pp. 1505–1510.

[48] C. Stachniss, D. Hähnel, W. Burgard, and G. Grisetti, "On actively closing loops in grid-based FastSLAM," *Advanced Robotics*, vol. 19, no. 10, pp. 1059–1079, 2005.

[49] D. Balage, *Multi Robot FastSLAM*. PhD thesis, Memorial University of Newfoundland, 2010.

[50] D. Liu, G. Liu, and M. Yu, "An improved FastSLAM framework based on particle swarm optimization and unscented particle filter?," *Journal of Computational Information Systems*, vol. 8, no. 7, pp. 2859–2866, 2012.

[51] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 1, pp. 206–211 vol.1, 2003.

[52] D. Hähnel, S. Thrun, B. Wegbreit, and W. Burgard, "Towards lazy data association in slam," in *Robotics Research. The Eleventh International Symposium* (P. Dario and R. Chatila, eds.), (Berlin, Heidelberg), pp. 421–431, Springer Berlin Heidelberg, 2005.

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

[54] H. F. Durrant-Whyte, "Uncertain geometry in robotics," *IEEE Journal on Robotics and Automation*, vol. 4, pp. 23–31, Feb 1988.

[55] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333–349, Oct 1997.

[56] M. Golfarelli, D. Maio, and S. Rizzi, "Elastic correction of dead-reckoning errors in map building," in *1998 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 2, pp. 905–911 vol.2, 1998.

[57] Yufeng Liu and S. Thrun, "Results for outdoor-slam using sparse extended information filters," in *2003 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 1227–1233 vol.1, 2003.

[58] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework," *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, 2004.

[59] P. M. Newman, J. J. Leonard, and R. J. Rikoski, "Towards constant-time SLAM on an autonomous underwater vehicle using synthetic aperture SONAR," in *Robotics Research. The Eleventh International Symposium* (P. Dario and R. Chatila, eds.), (Berlin, Heidelberg), pp. 409–420, Springer Berlin Heidelberg, 2005.

[60] M. A. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, (San Francisco, CA, USA), p. 1157–1164, Morgan Kaufmann Publishers Inc., 2003.

[61] S. Thrun, *Simultaneous Localization and Mapping*, pp. 13–41. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[62] E. Nettleton, S. Thrun, H. Durrant-Whyte, and S. Sukkarieh, *Decentralised SLAM with Low-Bandwidth Communication for Teams of Vehicles*, pp. 179–188. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006.

[63] P. M. Newman, "On the Structure and Solution of the Simultaneous Localisation and Map Building Problem Australian Centre for Field Robotics," *Doctor*, 1999.

[64] P. Newman and H. Durrant-Whyte, "The geometric projection filter - an efficient solution to the SLAM problem," *Proceedings of SPIE - The International Society for Optical Engineering*, 10 2001.

[65] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[66] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), p. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[67] A. Burguera, Y. Cid, and G. Oliver, "On the use of likelihood fields to perform SONAR scan matching localization," *Autonomous Robots*, vol. 26, pp. 203–222, 05 2009.

[68] M. Habbecke and L. Kobbelt, "Iterative multi-view plane fitting," in *Computer Graphics Group, RWTH Aachen University*.

[69] M. Eliwa, A. Adham, I. Sami, and M. Eldeeb, "A critical comparison between Fast and Hector SLAM algorithms," *REST Journal on Emerging trends in Modelling and Manufacturing*, vol. 3, no. 2, pp. 44–49, 2017.

[70] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[71] S. Thrun, "Learning occupancy grids with forward models," in *2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 1676–1681 vol.3, 2001.

[72] B. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Journal of Robotics and Autonomous Systems*, vol. 8, pp. 47–63, 1991.

[73] "Machine vision," in *Machine Vision (Third Edition)*, Signal Processing and its Applications, p. i, Burlington: Morgan Kaufmann, third edition ed., 2005.

[74] J. Derenick, A. Speranzon, and R. Ghrist, "Homological sensing for mobile robot localization," in *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 572–579, 2013.

[75] D. Jung and A. Zelinsky, "Grounded symbolic communication between heterogeneous cooperating robots," *Autonomous Robots*, vol. 8, pp. 269–292, Jun 2000.

[76] J. De Hoog, S. Cameron, and A. Visser, "Role-based autonomous multi-robot exploration," *Computation World: Future Computing, Service Computation, Adaptive, Content, Cognitive, Patterns, ComputationWorld 2009*, pp. 482–487, 2009.

[77] A. Howard, "Multi-robot simultaneous localization and mapping using particle filters," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1243–1256, 2006.

[78] L. Carlone, M. K. Ng, J. Du, B. Bona, and M. Indri, "Rao-blackwellized particle filters multi robot SLAM with unknown initial correspondences and limited communication," *2010 IEEE International Conference on Robotics and Automation (ICRA)*, no. May, pp. 243–249, 2010.

[79] K. M. Wurm, *Techniques for Multi-Robot Coordination and Navigation*. PhD thesis, Technische Fakultät Albert-Ludwigs-Universität Freiburg im Breisgau, 2012.

[80] E. W. Nettleton, P. W. Gibbens, and H. F. Durrant-Whyte, "Closed form solutions to the multiple platform simultaneous localisation and map building (SLAM) problem," *Sensor Fusion: Architectures, Algorithms, and Applications IV*, pp. 428–437, 2000.

[81] S. Thrun and Y. Liu, "Multi-robot SLAM with sparse extended information filters," *Springer Tracts in Advanced Robotics*, vol. 15, pp. 254–265, 2005.

[82] J. Fenwick, P. Newman, and J. Leonard, "Cooperative concurrent mapping and localization," no. May, pp. 1810–1817, 2003.

[83] I. Rekleitis, G. Dudek, and E. Milios, "Multi-robot collaboration for robust exploration," *Annals of Mathematics and Artificial Intelligence*, vol. 31, pp. 7–40, Oct 2001.

[84] X. S. Zhou and S. I. Roumeliotis, "Multi-robot slam with unknown initial correspondence: The robot rendezvous case," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1785–1792, Oct 2006.

[85] L. A. Andersson and J. Nygårds, "C-SAM: Multi-robot SLAM using square root information smoothing," *2008 IEEE International Conference on Robotics and Automation (ICRA)*, no. June, pp. 2798–2805, 2008.

[86] S. Thrun, "A probabilistic online mapping algorithm for teams of mobile robots," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 335–363, 2001.

[87] T. A. Vidal-Calleja, C. Berger, J. Solà, and S. Lacroix, "Large scale multiple robot visual mapping with heterogeneous landmarks in semi-structured terrain," *Robotics and Autonomous Systems*, vol. 59, no. 9, pp. 654–674, 2011.

[88] J. Zhao, Z. Feng, F. Chu, and N. Ma, "Chapter 4 - free motion of the end effector of a robot mechanism," in *Advanced Theory of Constraint and Motion Analysis for Robot Mechanisms* (J. Zhao, Z. Feng, F. Chu, and N. Ma, eds.), pp. 113 – 157, Oxford: Academic Press, 2014.

[89] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *2000 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1, pp. 476–481, 2000.

[90] S. Carpin, "Fast and accurate map merging for multi-robot systems," *Autonomous Robots*, vol. 25, no. 3, pp. 305–316, 2008.

[91] W. Rone and P. Ben-Tzvi, "Mapping, localization and motion planning in mobile multi-robotic systems," *Robotica*, vol. 31, no. 1, pp. 1–23, 2013.

[92] J. Ko, B. Stewart, D. Fox, K. Konolige, and B. Limketkai, "A practical, decision-theoretic approach to multi-robot mapping and exploration," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 4, pp. 3232–3238 vol.3, Oct 2003.

[93] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006.

[94] S. Carpin, A. Birk, and V. Jucikas, "On map merging," *Robotics and Autonomous Systems*, vol. 53, no. 1, pp. 1–14, 2005.

[95] A. Birk and S. Carpin, "Merging occupancy grid maps from multiple robots," *Proceedings of the IEEE*, vol. 94, no. 7, pp. 1384–1397, 2006.

[96] S. Saeedi, L. Paull, M. Trentini, and H. Li, "Neural network-based multiple robot simultaneous localization and mapping," 2011.

[97] P. Dinnissen, S. N. Givigi, and H. M. Schwartz, "Map merging of multi-robot SLAM using reinforcement learning," *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, pp. 53–60, 2012.

[98] J. L. Blanco, J. González-Jiménez, and J. A. Fernández-Madrigal, "A robust, multi-hypothesis approach to matching occupancy grid maps," *Robotica*, vol. 31, no. 5, pp. 687–701, 2013.

[99] A. Censi, L. Iocchi, and G. Grisetti, "Scan Matching in the Hough Domain," in *2005 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2739–2744, April 2005.

[100] W. H. Huang and K. R. Beevers, "Topological map merging," *International Journal of Robotics Research*, vol. 24, no. 8, pp. 601–613, 2005.

[101] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *ICRA 2010 workshop*, 2010.

[102] L. Ma, J. Zhu, L. Zhu, S. Du, and J. Cui, "Merging grid maps of different resolutions by scaling registration," *Robotica*, vol. 34, no. 11, pp. 2516–2531, 2016.

[103] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–110, Nov 2004.

[104] E. Karami, M. S. Shehata, and A. J. Smith, "Image identification using SIFT algorithm: Performance analysis against different image deformations," *CoRR*, vol. abs/1710.02728, 2017.

[105] M. S. Güzel, "A hybrid feature extractor using fast Hessian detector and SIFT," *Technologies*, vol. 3, no. 2, pp. 103–110, 2015.

[106] L. Chiu, T. Chang, J. Chen, and N. Y. Chang, "Fast SIFT design for real-time visual feature extraction," *IEEE Transactions on Image Processing*, vol. 22, no. 8, pp. 3158–3167, 2013.

[107] Yan Ke and R. Sukthankar, "PCA-SIFT: a more distinctive representation for local image descriptors," in *2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. II–II, 2004.

[108] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool], "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346 – 359, 2008. Similarity Matching in Computer Vision and Multimedia.

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

[110] Itseez, "Open source computer vision library." https://github.com/itseez/opencv, 2015.

[111] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, p. 381–395, June 1981.

[112] E. Karami, S. Prasad, and M. Shehata, "Image matching using SIFT, SURF, BRIEF and ORB: Performance comparison for distorted images," *ArXiv*, vol. abs/1710.02726, 2017.

[113] A. Filatov, A. Filatov, K. Krinkin, B. Chen, and D. Molodan, "2D SLAM quality evaluation methods," in *2017 21st Conference of Open Innovations Association (FRUCT)*, pp. 120–126, Nov 2017.

[114] A. Huletski, D. Kartashov, and K. Krinkin, "TinySLAM improvements for indoor navigation," in *2016 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, pp. 493–498, 2016.

[115] N. Carlevaris-Bianco, A. K. Ushani, and R. M. Eustice, "University of Michigan North Campus long-term vision and LIDAR dataset," *The International Journal of Robotics Research*, vol. 35, no. 9, pp. 1023–1035, 2016.

[116] M. Fallon, H. Johannsson, M. Kaess, and J. J. Leonard, "The MIT stata center dataset," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1695–1699, 2013.

Appendix A

Appendix

A.1 Repository for the code

Follow the read me file in the GitHub repository to replicate the work done in this thesis: https://github.com/dikokob/DikokoMScEng. The read me file also includes the process of setting up the turtle bot.

A.2 Raw data-sets

Raw data-sets used in Chapters 5 and 3 can be found in: https://drive.google.com/drive/folders/1c_2-T8FcrSmE0VDyHw20-jRLHKRuAvpE?usp=sharing

A.3 Hector mapping parameters

Parameter	Value	Description
map_resolution	Varies based on experiment	The map resolution m/cell. This is the length of a grid cell edge.
map_size	2048	The size (number of cells per axis) of the map. The map is square and has (map_size * map_size) grid cells.
map_start_x	0.5(default)	Location of the origin (0.0, 1.0) of the /map frame on the x axis relative to the grid map. 0.5 is in the middle.
map_start_y	0.5(default)	Location of the origin (0.0, 1.0) of the /map frame on the y axis relative to the grid map. 0.5 is in the middle.
map_update_distance_thresh	0.4 (default)	Threshold for performing map updates (m). The platform has to travel this far in meters or experience an angular change as described by the map_update_angle_thresh parameter since the last update before a map update happens.
map_update_angle_thresh	0.9(default)	Threshold for performing map updates (rad). The platform has to experience an angular change as described by this parameter of travel as far as specified by the map_update_distance_thresh parameter since the last update before a map update happens.
map_pub_period	2.0	The map publish period (s).
map_multi_res_levels	3(default)	The number of map multi-resolution grid levels.
update_factor_free	0.4(default)	The map update modifier for updates of free cells in the range (0.0, 1.0). A value of 0.5 means no change.
update_factor_occupied	0.9(default)	The map update modifier for updates of occupied cells in the range (0.0, 1.0). A value of 0.5 means no change.
laser_min_dist	0.1	The minimum distance (m) for laser scan endpoints to be used by the system. Scan endpoints closer than this value are ignored.
laser_max_dist	30(default)	The maximum distance (m) for laser scan endpoints to be used by the system. Scan endpoints farther away than this value are ignored.
laser_z_min_value	-1.0(default)	The minimum height (m) relative to the laser scanner frame for laser scan endpoints to be used by the system. Scan endpoints lower than this value are ignored.
laser_z_max_value	1.0(default)	The maximum height (m) relative to the laser scanner frame for laser scan endpoints to be used by the system. Scan endpoints higher than this value are ignored.