COOPERATIVE LOCALIZATION AND MAPPING OF AUTONOMOUS ROBOTS

Submitted in partial fulfilment
of the requirements of the degree of

BACHELOR OF SCIENCE (HONOURS)

of Rhodes University

Lynton Dicks

Grahamstown, South Africa
November 2011
Abstract

Cooperative Localization and Mapping (CLAM) of Autonomous Robots is an extension to the Simultaneous Localization and Mapping problem (SLAM), in the field of robotics, that provides a team of robots with the ability to create a global map of an environment while, at the same time, using that map to localize (position) themselves within that environment. This research paper is concerned with the implementation of a framework, which is written in the Python 3 programming language, that provides a team of robots with the tools that are necessary to explore an environment and construct a single, combined map of an environment quickly and efficiently. These tools include: an implementation of the RANSAC algorithm for fitting the landmark data to a least squares model so as to clean the data of any outliers, an occupancy grid map building module for constructing a map from landmark data that was captured relative to a compass as a common reference point, and an occupancy grid map merging module that combines the map segments constructed by each individual robot into their respective positions in a single, combined map. Results show how the RANSAC algorithm outperforms older parameter estimation techniques, such as the least squares model, in differentiating correct sensor data from incorrect sensor data. It was concluded that merging occupancy grid maps can be done relatively easily, with minimum computation, by using compass readings as a common reference point during landmark extraction and map building. Using the RANSAC algorithm on the landmark data should result in an occupancy grid that requires less storage due to the removal of outliers from the occupancy grid map.
ACM Computing Classification System Classification


**D.3.3** [Language Constructs and Features]: Frameworks

**I.2.9** [Robotics]: Autonomous vehicles, Kinematics and dynamics, Sensors

**General-Terms**: Algorithms, Design
Acknowledgements

This thesis has taken many months to produce with support from many people. I would like to thank all those people who have contributed and assisted me in any way to make this research a success. I would like to thank a number of people specifically.

I would like to acknowledge the financial and technical support of Telkom, Tellabs, Stortech, Eastel, Bright Ideas Project 39, and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

I would also like to acknowledge my friends, family, and, especially, my fiance for all the support, funding, and opinions that they have provided during this time.

Finally, I would like to thank my project supervisor, Dr. Karen Bradshaw, for the guidance and knowledge provided during this project, which, without, this project would not have been realized.
2.6.1 Multi-Hypothesis Tracking with Pruning ................................. 20
2.6.2 Joint Compatibility Branch and Bound test ............................. 21

2.7 Localization Algorithms .......................................................... 21
  2.7.1 Kalman Filter (EKF) ............................................................. 22
  2.7.2 Monte Carlo Localization ...................................................... 22
  2.7.3 Markov Localization ............................................................. 23

2.8 Map Building .......................................................... 23
  2.8.1 Topological Maps ............................................................... 23
  2.8.2 Occupancy Grid Maps ......................................................... 24
  2.8.3 Hybrid Maps ................................................................. 24

2.9 Map Merging ................................................................. 26

2.10 Robotics Frameworks ............................................................... 26

2.11 Applications of SLAM and CLAM ............................................. 29

2.12 Summary ................................................................. 29

3 Design Decisions .......................................................... 31
  3.1 Landmark Extraction ................................................................. 31
  3.2 Map Building ................................................................. 32
  3.3 CLAM ................................................................. 33
    3.3.1 Role ................................................................. 34
    3.3.2 Centralization ............................................................... 34
    3.3.3 Aggregation ............................................................... 35
    3.3.4 Communication ............................................................. 35
4 Implementation

4.1 Hardware Set-up .................................................. 38
4.2 Programming Frameworks and Languages .......................... 39
4.3 XBoxUtils .............................................................. 40
4.4 DatabaseUtils .......................................................... 41
4.5 RansacUtils ........................................................... 42
4.6 MapBuildUtils .......................................................... 43
4.7 MapMergeUtils ........................................................ 44

5 Results ................................................................. 45

5.1 Introduction ............................................................ 45
5.2 Results ................................................................. 46

6 Conclusion .............................................................. 48

6.1 Project Summary ....................................................... 48
6.2 Outcomes of Objectives ............................................... 48
6.3 Extensions .............................................................. 49
   6.3.1 Unique versus Non-unique Landmark Observations ............. 49
6.3.2 Communication ...................................................... 50
6.3.3 Mapping ............................................................. 50
6.3.4 Localization ......................................................... 51
## List of Figures

2.1 The Fischertechnik robot that is used in this project complete with Controller, two ultrasonic sensors, two encoder motors, and Bluetooth. (This image is taken from [4]). ................................. 11

2.2 The Fischertechnik Encoder Motor used to record the rotation of the motor’s output shaft to use in odometry. (This image is taken from [4]). . . . 12

2.3 The Fischertechnik ultrasonic range finder with an arc width of 60 degrees, maximum range of 4 meters, and a scan resolution of 1 centimetre (This image is taken from [4]). .................................................. 13

2.4 Summary of the RANSAC Algorithm [6] ........................................ 16

2.5 Example range scan data (This image taken from [4]). ..................... 17

2.6 Least Squares approximation of the range scan data (This image modified from [4]). ............................................................... 18

2.7 RANSAC approximation, using the Least Squares model, of the range scan data (This image taken from [4]). ................................. 19

2.8 Grid Map overlaid on data points ............................................... 25

2.9 Occupancy Grid Map computed from data points .......................... 25

2.10 Maps created from a dynamic, populated environment without people filtering. (Image taken from [8]) ............................... 27

2.11 Maps created from a dynamic, populated environment using people filtering. (Image taken from [8]) ............................... 27
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.12</td>
<td>Framework Module Hierarchy (Image taken from [14]).</td>
<td>28</td>
</tr>
<tr>
<td>2.13</td>
<td>SLAM Framework Overview (Image taken from [4]).</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>RANSAC calculated on example range data overlaid on an occupancy grid map.</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Occupancy grid map built after the RANSAC approximation model has been run on the example data.</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>Rotated occupancy grid map with RANSAC calculated least squares lines.</td>
<td>36</td>
</tr>
<tr>
<td>5.1</td>
<td>RANSAC algorithm run on example data points. The red line represents the least squares model and the blue dashed line represents the RANSAC algorithm using the least squares model.</td>
<td>47</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Cooperative Localization and Mapping (CLAM) of Autonomous Robots is an extension to the Simultaneous Localization and Mapping (SLAM) problem in the field of robotics that provides a team of robots with the ability to create a global map of an environment while, at the same time, using that map to localize (position) themselves within that environment.

SLAM is a framework of algorithms that provides a robot with the ability to build a map of an environment for which no prior information exists and, at the same time, determine its location within that environment [3].

CLAM allows larger areas to be mapped in parallel, which makes exploration more efficient since robots do not have to map areas that have been mapped already by other robots [22]. It also gives robots the ability to map environments that have limited prior information available efficiently and reliably [23].

Autonomous robots with the ability to map an environment and localize themselves within that environment could be used in a variety of situations that could prove difficult or dangerous for humans, such as military reconnaissance or exploration of a remote planet. Reference [24] presents an application of SLAM that is designed to increase the safety of travelling in motor vehicles and reduce the traffic injuries caused by human factors such as speeding and distraction. Reference [24] also lists automated city mapping as another application of SLAM. CLAM can be applied to the problem of automated ocean exploration as done by the authors in [23]. The authors in [13], as well as the authors in [18], have successfully used CLAM in the RoboCup Soccer game, which is a soccer game played by robots.
When using the CLAM framework, the robots will move through the environment that they are exploring and capture landmark data (visible objects in the environment) and odometry data (data relating to the distance that the robot has travelled since it was started). The landmark and odometry data is used in a localization algorithm, which will attempt to correct for any errors in sensor readings, and then valid landmark data is added to a map if it has not already been added, which is determined in the data-association phase. The data-association phase of SLAM checks to see if the landmark already exists in the map or not. Since the main purpose of CLAM is cooperative mapping, the map segments that each robot has collected will need to be merged to form a single, combined map that represents the entire environment as explored by the robots. This combining of map segments to form a single map is referred to as map merging or aggregation.

1.1 Problem Statement and Research Goals

The goal of this research project is to implement a CLAM framework that could eventually be used with a team of robots to allow them to work together to explore and map an environment. The research project will delve into the details of the components required for a functional CLAM implementation, but will focus more on the aspects of map building and map merging for the implementation in this research project.

This research project is not concerned with the actual interfacing and control of the robot’s hardware so this functionality is provided by the generic robotic programming framework, which was developed in [14], that abstracts away the details of working with different robot platforms.

Since the SLAM framework, which was developed in [4], that is being used for the SLAM components of the project does not contain a localization algorithm, the data collected from the environment, via the landmark sensors, will have to be run through the RANSAC algorithm. The RANSAC algorithm will remove incorrect data readings (outliers) and match straight lines to the correct data readings (inliers). Since correct map data is essential to this research project, it will provide an implementation of the RANSAC algorithm. The SLAM framework does, however, provide landmark extraction and odometry information with support classes for both of them.

The implementation of this research project will be written in the Python 3 programming language, which allows it to integrate well with the two frameworks mentioned previously.
Map building and map merging of occupancy grid maps will be implemented and the results of these algorithms will be stored in a database, which will also be implemented.

A method of interfacing with a Microsoft Wireless XBox Controller, in order to provide test input to the robot, will be implemented.

The functionality for this research project will be implemented in a series of modules and classes using sound object-oriented methodology to allow ease of maintenance and ease of extensibility.

\section*{1.2 Thesis Organization}

The rest of this thesis is divided into 5 chapters, which are summarized in this section.

Chapter 2 provides background knowledge on the topic of localization and mapping of Autonomous Robots. Both SLAM and CLAM will be discussed, along with descriptions of the components that are necessary to create a working implementation. Current research in the field of localization and mapping will be investigated and ideas compared.

Chapter 3 details the design of the framework and explains the rationale behind design decisions. These design decisions are closely related to the topics discussed in the background chapter.

Chapter 4 describes the implementation of the framework along with any libraries used. The chapter begins with an explanation of the robotic platform used in the project, along with the programming frameworks and programming language used in the development of the project. Descriptions of the various modules and classes, within those modules, are provided.

Chapter 5 outlines the tests used and the results obtained, followed by an analysis of the results.

Chapter 6 concludes the research project with a summary of the project, the outcomes of the objectives, and possible extensions to the project.
Chapter 2

Background

This chapter reviews various literature concerning Cooperative Localization and Mapping (CLAM) of Autonomous Robots in order to garner the knowledge necessary to implement and test a working framework that provides a team of robots with the ability to accurately map an environment by working together. The chapter begins with a discussion of Simultaneous Localization and Mapping (SLAM) and how the framework for a single robot to position itself and, at the same time, map its environment can be extended to allow a team of robots to do the same. During the course of this discussion, the various steps required during the process are expanded upon, along with the available algorithms and their respective advantages and disadvantages.

2.1 Simultaneous Localization and Mapping

SLAM is a framework of algorithms that provides a robot with the ability to build a map of an environment for which no prior information exists and, at the same time, determine its location within that environment [3]. Using a map with other means of recording movement, such as odometry and environmental sensing devices, gives the robot the ability to detect and recover from incorrect sensor data and more accurately plot its position within an environment [4]. SLAM has produced significant results in the past decade, but mainly for single robots [13]. The problem of single robot localization has been solved in many different ways by posing it in a probabilistic state estimation framework which combines the position and bearing (pose) of the robot, relative to its starting position, with its environmental observations (landmarks) to estimate the current robot’s position and bearing in absolute world co-ordinates [23].
2.2 Cooperative Localization and Mapping

A substantial amount of effort has been expended on solving the problem of localization of a single robot, but localization of a team of robots is still a relatively new field [23].

Cooperative Localization and Mapping introduces additional problem on top of SLAM such as each robot’s role, centralization, aggregation, and communication methods [4]. Regarding the robot’s role in Cooperative Localization and Mapping, each robot in a team can be an independent entity running its own instance of a localization algorithm or, as in the case of [23], there can be master and slave robots where the master robots perform SLAM and the slave robots act as extended eyes and ears. The slave robots feed sensor data back to the master robots to be used in the localization algorithm, but do not perform SLAM themselves. The master robots have better odometry and environment sensors and must have the ability to make inter-robot measurements as well as a means of communicating with the other robots in the team [23].

Centralization/Convergence concerns the question of how the robots will converge upon each other once they have completed mapping their respective sections of the environment.

Aggregation is concerned with how to combine the different sections of the map, which have been gathered by each of the robots, so that each section is in its correct place.

The communication methods used will depend on what roles are performed by each of the robots in a team: whether all the robots communicate with only a single, controlling robot (the master robot as mentioned in [23]) or whether each robot is independent and communicates with every robot in its team to build its portion of the map if and when it observes a robot.

Cooperative Localization and Mapping allows larger areas to be mapped in parallel, which makes exploration more efficient since robots do not have to map areas that have already been mapped by other robots [22]. It also gives robots the ability to map environments with limited prior information available, efficiently and reliably [23].

2.3 Hardware

The robot used for the development and testing of the framework is a Fischertechnik robotic kit with a Fischertechnik ROBO TX Controller. The controller consists of a 32-bit ARM 9 processor operating at 200 MHz, a display, and an integrated Bluetooth radio
2.3. HARDWARE

Figure 2.1: The Fischertechnik robot that is used in this project complete with Controller, two ultrasonic sensors, two encoder motors, and Bluetooth. (This image is taken from [4]).

interface operating at 2.4 GHz with a range of 10 m [5]. The robot has two ultrasonic sensors used to make observations of the environment and this sensor data forms part of the Measurement Model and is used to extract objects (Landmarks) from the environment [12]. The Plant Model contains measurements of how the robot’s position changes over time relative to its starting position by using encoders to measure the rotations of the motors (odometry) [12]. Such relative position data is known as odometry data. The robot is equipped with two such encoders that provide it with the necessary odometry data that it needs to localize itself (determine its position).

2.3.1 Encoders

Encoders are specialized motors that are used to capture odometry data by recording the rotations performed by a motor. The robot’s controller operates the motor via the controller’s internal PWM (Pulse Width Modulation) circuitry to control its forward or backward speed. Relying on the amount of time that the motor’s output shaft turns is unreliable since the speed that the motor turns is dependent on the battery’s voltage levels as well as the resistance that the motor encounters through surface friction on attached
2.3. HARDWARE

Figure 2.2: The Fischertechnik Encoder Motor used to record the rotation of the motor’s output shaft to use in odometry. (This image is taken from [4]).

wheels [4]. A decrease in battery voltage or increase in resistance on the motor will cause the motor to spin slower and thus overestimate the distance traversed by the robot.

There are many methods that encoders use to record the rotation of the motor’s output shaft. Mechanical encoders use magnets or metal brushes around the shaft in regular positions that generate a tick each time they pass a stationary contact in the motor casing. The number of ticks per second are recorded and represent the amount of rotation in terms of angular ticks [17].

Another method mentioned in [17] is the use of optical encoders. These encoders have a patterned/punctured disc attached to the output shaft of the motor and a stationary contact in the motor encasing will record the number of ticks.

2.3.2 Sensors

Observations of the environment are captured by sensors which provide range and bearing data for observed landmarks (objects in the environment). These landmark observations
2.4 Odometry Data

Odometry data is captured by Dead reckoning sensors. These sensors provide data about the robot’s current position relative to its starting position. There are many types of sensors that provide dead reckoning data. For ground based robots encoders are used, and for aerial vehicles GPS (Global Positioning System) is used, whereas for AUVs (Autonomous Underwater Vehicles) Gyros, INS (Inertial Navigation Systems, and DVL (Doppler Velocity Logs) are used [11].
2.5 Landmark Extraction

Landmarks are observed objects in the environment whose co-ordinates have been recorded by environmental sensing devices. A localization algorithm will use this data to match landmarks against the map if they exist or add the landmarks to the map if they have not been previously observed [19].

2.5.1 RANSAC

RANSAC (Random Sample Consensus) is a parameter estimation algorithm, developed by the authors in [6] for fitting a model to data in the presence of many outliers. Outliers are incorrect data readings that do not follow the correct pattern of the data set. In this project, for example, outliers are the result of inaccuracies in the ultrasonic sensors or wheel slippage in the odometry readings. RANSAC can be used, as a method of Landmark Extraction, to identify lines from range scans [4].

As explained by [6], most scientific pursuits deal with the interpretation of captured data in terms of a set of predefined models. In order to interpret data, a model that accurately matches the data must be chosen and then the data values, which have the best fit to the chosen model, must be computed [6].

Older parameter estimation techniques, such as least squares, have no internal methods for detecting or rejecting outliers, but rather rely on there being large amounts of correct data to average out the results [6]. It is the opinion of the authors in [6] that averaging the data is not an effective method to use on data sets containing high amounts of outliers since a single datum with an excessively incorrect value can cause these older parameter estimation techniques to fail.

Unlike these older parameter estimation techniques, which use as much data as possible to find an initial solution and then attempt remove the outliers, RANSAC uses the smallest set of data that is possible in order to form its initial solution and then attempts to add additional correct data that fits within a specified error tolerance [6]. RANSAC then uses a smoothing technique, such as least squares, in order to improve the accuracy of the parameter estimation [6].

Figure 2.4 summarizes the main steps involved in the RANSAC algorithm. The initial starting condition is that the data set is bigger than the minimum number of data values
that is necessary to solve for the model’s parameters [6]. The minimum number of data values necessary to solve for the model parameters are randomly chosen from this data set and used to solve for the selected model [6]. The model is then used to determine the subset of data values that lie within a predefined error tolerance of the selected model [6]. The subset of data values that fit the model within a predefined error tolerance is called the consensus set of the model’s parameter data [6]. If the consensus set divided by the total number of data values in the data set is greater than a predefined threshold then compute a new model using the consensus set as the model’s parameters, else repeat the process from the start a predefined number of times, at which point, one can either exit with a failure status or solve the model with the largest consensus set found [6].

The authors in [6] point out that the algorithm can be improved by using a deterministic method of choosing the model parameters from the data set instead of randomly choosing them.

The authors in [6] state the three predefined parameters that are used in the RANSAC algorithm as:

1. The error tolerance that is used to determine whether a point fits a particular model. The error tolerance is calculated, via experiment, as being one or two standard deviations from the measured average error.

2. The threshold, which specifies the number of correct data points necessary to say the correct model has been found. This threshold must be large enough to find the correct model and help with the computation of the improved model parameter estimates [6].

3. The number of times to repeat the algorithm to find a valid model can be based upon the expected number of attempts required to select a subset of good data points.

Figure 2.5 shows the example range scan data that will be used to illustrate the difference in effectiveness of the Least Squares approximation model versus the RANSAC approximation model, which is using the least squares approximation model as its base model.

Figure 2.6 illustrates how the Least Squares approximation model interprets the example range scan data and figure 2.7 illustrates how the RANSAC approximation model, which is using the Least Squares approximation model as its base model, interprets the example
2.5. LANDMARK EXTRACTION

Figure 2.4: Summary of the RANSAC Algorithm [6]

1. Randomly select the minimum number of data values necessary for the model parameters
2. Solve for the model using the selected data values
3. Use the model to determine the subset (consensus set) of the data set that lie within a predefined error tolerance of the model
4. If the consensus set divided by the total number of data values in the data set is greater than a predefined threshold, compute a new model using the consensus set as the model’s parameters.
5. If the consensus set divided by the total number of data values in the data set is not greater than a predefined threshold, go to step 1 and repeat the process until a predefined maximum attempts has been reached and, at which point, exit with a failure status or solve the model with the largest consensus set found

range scan data. As can be seen from figure 2.6, the least squares approximation model does a poor job of interpreting the data in the presence of a high number of outliers whereas, as shown in figure 2.7, the RANSAC approximation model accurately represents the example range data according to the least squares approximation model.

2.5.2 Unique versus Non-unique Landmark Observations

Normally, as described in [19], landmarks should be unique so that they do not get incorrectly associated with other landmarks in the map and they should also be adequately spaced to prevent them being interpreted as a single landmark. However, as described in [18], it is possible to localize with non-unique landmark observations by using an importance sampling based approximate solution and implicit hypothesis pruning. This works by removing as many ambiguities, caused through non-unique landmarks, as possible before the landmark data is used by the localization algorithm. A simpler method of removing landmark ambiguities mentioned in [18], is by using a compass to provide directional context to non-unique landmarks.
Figure 2.5: Example range scan data (This image taken from [4]).
Figure 2.6: Least Squares approximation of the range scan data (This image modified from [4]).
Figure 2.7: RANSAC approximation, using the Least Squares model, of the range scan data (This image taken from [4]).
2.5.3 Time between Landmark Observations

Another requirement for successful SLAM, is that there must not be a long time period between observations of landmarks since this would cause the algorithm to reach high levels of uncertainty and get lost [19].

2.5.4 Static versus Dynamic Environments

Reference [19] mentions that another requirement for successful SLAM is that landmarks should remain stationary since a moving landmark would cause uncertainty in the localization algorithm, however, the authors in [8] explain how they implemented a probabilistic method of tracking multiple people in a populated environment and how they incorporated the results of the tracking into the mapping process. They explain that the resulting maps are more accurate since the number of random or non-landmark objects in the environment are reduced and corrupted readings are flagged.

2.6 Data Association

Data Association is the ability of a robot to match a set of measurements, made by itself or other robots, against other robots and known features already in the map in order to determine new landmarks [23].

2.6.1 Multi-Hypothesis Tracking with Pruning

The authors in [18] use a method called Multi-Hypothesis Tracking (MHT), where the possible data associations form hypotheses, which are tracked with a Kalman Filter after first being pruned to reduce complexity. This allows non-unique landmarks to be used in the environment since it reduces the ambiguities caused through non-unique landmarks, such as the poles of a goal post, before the landmark data is used in the localization algorithm.
2.6.2 Joint Compatibility Branch and Bound test

Reference [16] proposes the use of a data association method called the Joint Compatibility Branch and Bound test (JCBB). JCBB is more robust to incorrect pairings even if the pose uncertainty is high and JCBB takes the compatibility of the observations made by more than one robot into account. It does this by performing a depth first search for the correct hypothesis [16].

2.7 Localization Algorithms

While a robot is moving, the odometry and landmark data that is collected is used by the localization algorithm, which maintains the uncertainties of landmark and position estimates to update the robots position in absolute world co-ordinates and map the locations of those landmarks if they have not already been seen [19]. The odometry data and the landmark data must be collected simultaneously or time stamped in order for it to be useful as input to the localization algorithm [19]. Localization algorithms combine odometry data with environment observations (Landmarks) to determine the robot’s position and orientation in absolute world coordinates (the robot’s pose). Localization algorithms can be divided into those that do not use Landmarks commonly observed by other team robots to improve their localization calculations and those that take advantage of environmental observations to improve each robot’s pose estimates, while simultaneously mapping the environment [13]. The localization algorithm can be distributed over all the robots in a team as demonstrated in [20] or each individual robot can run its own instance of the localization algorithm.

Reference [10] uses a stereo-vision based method that uses landmarks extracted by one robot to help other robots localize themselves. Reference [13] claims that the stereo-vision based method is focused on a specific application (stereo-vision-based) and is therefore unsuitable as a general localization method that makes use of landmarks in its pose estimation, but they retain the idea of using landmarks extracted by one robot to help another robot to localize itself. Their method is described in further detail in the Monte Carlo Localization section below.
2.7. LOCALIZATION ALGORITHMS

2.7.1 Kalman Filter (EKF)

Reference [20] presents a method which use a single Kalman Filter distributed among the robots in the group. This approach does not use commonly observed landmarks to improve the accuracy of its localization calculations and addresses multi-robot localization as the relative localization of a robot based on the inter-robot distance measurements [13].

Similarly, the authors in [23] investigate a method of Cooperative Mapping and Localization, known as Moving Baseline Navigation, in which only one robot is responsible for maintaining estimates of the map and poses for each robot. Inter-robot measurements are combined with observations of the environment made by each robot in the team to determine each robot’s pose [23].

Reference [21] implements cooperative localization within a Kalman Filter Framework using the relative positions of the robots as the observations of the filtering part of the algorithm, and the state includes the positions of all the robots. This algorithm does not use commonly observed landmarks to improve the accuracy of its localization calculations [13].

2.7.2 Monte Carlo Localization

Reference [7] uses the Monte Carlo Localization algorithm to represent the probability distributions of the robots in a group and uses inter-robot observations to improve state estimation, but claims that the Monte Carlo Localization algorithm ignores the inter-robot correlation which result in the algorithm suffering from overconfidence.

A modification of the Monte Carlo Localization (MCL) algorithm is demonstrated in [13] where particles are not spread uniformly in the state space when a robot is lost, but rather according to information on the location of a commonly observed object whose distance and bearing is measured by the lost robot and whose absolute world coordinates are provided by other robots via direct wireless communication. This algorithm does use commonly observed landmarks to improve the accuracy of its localization calculations and takes advantage of the features exposed by particle filter algorithms and provides global robot and object localization [13]. The modification is said to decrease the time taken for a lost robot to recover and, due to the extra landmark information provided by other robots in the team, is robust to perceptual aliases, which occurs when environments have symmetries [13]. Each robot runs the Monte Carlo Localization algorithm to
localize itself and is able to determine when the uncertainty about its localization drops below a given threshold and that uncertainty is determined through the existence of an observation model that enables the level of confidence on the position of the object to be determined [13]. This modification to the MCL algorithm relies on a common object being observed by many robots in the team, which is, in this case, a soccer ball that all robots of the team can observe and provide position estimates to be used to localize the lost robot.

### 2.7.3 Markov Localization

Reference [7] uses an extended version of the general Markov Localization algorithm where the relative distance and bearing of two robots is inserted as an extra step in the belief update algorithm, which is based on the Bayes filter. The Monte Carlo Localization sampled version of the Markov Localization algorithm was used to influence the weights of the observed robot’s particles over the observing robot’s particle sampling of the inter-robot distance and bearing measurements.

### 2.8 Map Building

Map building refers to the process where a robot constructs a map out of the raw data points collected from the environment after the Landmark Extraction phase. There are two main types of mapping: metric mapping and topological mapping [1].

Metric maps are well suited to representing the geometry of the environment and are able to provide a detailed representation of that environment, but they require more storage and are sensitive to sensor errors [1].

Topological maps, on the other hand, are well suited for representing navigable paths through the environment, require less storage and are more robust to sensor error, but cannot represent open spaces [1].

#### 2.8.1 Topological Maps

Topological maps represent the navigability of the environment in a graph structure [1]. Vertices in topological maps represent distinct features of the environment such as door-
ways and junctions [1]. Edges in topological maps represent paths through the environment or the steps required to get to a specific vertex [1]. Attempts have been made to extract topological information from occupancy grid maps, but most topological maps are generated during the exploration phase [1].

2.8.2 Occupancy Grid Maps

An occupancy grid maps is a method of storing metric information about an environment as a 2-dimensional grid of cells [1]. Each cell in the occupancy grid contains a probability that the cell contains an object [1]. This probability can be computed by searching all the cells for the cell containing the maximum number of data points and then setting all cell probabilities relative to this maximum probability. Any cell that contains a probability higher than a predefined threshold value will be considered occupied.

Figure 2.8 illustrates an occupancy grid map overlaid on example data points. The resolution of an occupancy grid map can be increased by increasing the number of rows and columns that the grid contains, which allows an occupancy grid map to represent the positions of smaller objects more accurately.

Figure 2.9 illustrates an occupancy grid map with the probabilities calculated for the example data points from figure 2.8. The probability of each cell can be calculated relative to the cell with the maximum number of data points. Using the example data points in figure 2.8, the maximum number of data points in any one cell is 4 therefore all the cells that contain 4 data points will have a probability of 100% and all other cells will have their probabilities set relative to these cells.

2.8.3 Hybrid Maps

As mentioned in [1], attempts have been made to combine topological and metric mapping techniques in order to obtain both their benefits. Reference [1] mentions authors that used a topological map to represent a network of hallways and metric maps to represent the rooms.
Figure 2.8: Grid Map overlaid on data points

Figure 2.9: Occupancy Grid Map computed from data points
2.9 Map Merging

Each robot builds its own map of the section of the environment that it is exploring and, when a robot observes another robot, attempts to combine those individual sections of the environment into a single complete map which works in absolute world co-ordinates.

The authors in [2] present an algorithm for merging the occupancy grid maps produced by the multiple robots exploring the environment. Their algorithm produces a set of weighted possible transformations (rotations and translations) needed to merge two maps by extracting spectral information from the maps. Since each possible transformation is weighted, the algorithm is able to continue when ambiguities arise. The authors make the claim that their approach is deterministic, non-iterative, and fast (merging 250 thousand cells in 500 ms on common desktop PCs).

The authors in [8] discuss an algorithm to build maps in dynamic, populated environments using a probabilistic method to track people and to incorporate the tracking technique into the mapping process. Their algorithm could be useful as a first step in co-operating localization and mapping since it can be modified to track other robots in the environment and thus filter them out of the mapping process and use their positions as reference points for merging their map sections.

Figure 2.10 illustrates a map that was built of an environment populated with people. This map contains many spurious readings caused by the moving people and, therefore, does not accurately represent the environment. Figure 2.11 illustrates a map that was built of an environment populated with people using the mapping algorithm mentioned in [8]. This algorithm has tracked and removed the moving objects from the generated map.

2.10 Robotics Frameworks

Various programming frameworks can be used to simplify the task of implementing co-operative mapping and localization functionality on different robotic platforms.

Leslie Luyt, in [14], developed a generic robotic programming framework to allow multiple differing robotic platforms to be used in a project without having to alter the project code. As shown in Figure 2.12, the framework includes modules for movement, sensor management, and artificial intelligence. Base classes, which contain the specifics about
Figure 2.10: Maps created from a dynamic, populated environment without people filtering. (Image taken from [8])

Figure 2.11: Maps created from a dynamic, populated environment using people filtering. (Image taken from [8])
sensor and motor interfaces, must be implemented for each new robotic platform in order for the framework to work with that respective robotic platform.

Shaun Egan, in [4], developed an on-line and an off-line SLAM framework, providing common functionality to both such as odometry, landmark extraction, data association, and localization. The on-line section of the framework runs in real-time, concurrently capturing odometry and landmark data and then using it immediately. The odometry and landmark extraction phases run concurrently using separate threads of execution so as to avoid having to time-stamp the data. The off-line section of the framework captures odometry and landmark data and stores the time-stamped data in a serialized binary file for later use. After data capture, the off-line section of the framework processes the time-stamped data from the serialized binary file. The off-line SLAM framework allows different approaches to the SLAM problem to be used in order to compare their results [4].

Figure 2.13 shows the flow of data in the SLAM framework. Landmark data is first extracted from a set of range scans. This landmark data then goes through the Data Association phase, which determines whether the landmark has been seen before or not. If it has been seen before, the certainty of that landmark position is increased in the EKF localization method. If the landmark has not been seen before, it is added to the map with an observation count of 1. The new landmark, together with the odometry data collected at the same time as that landmark, are fed into the EKF odometry update method in order to localize the robot according to the new landmark.
2.11 Applications of SLAM and CLAM

Autonomous robots with the ability to map an environment and localize themselves within that environment could be used in a variety of situations that could prove difficult or dangerous for humans, such as military reconnaissance or exploration of a remote planet.

Reference [24] presents an application of SLAM that is designed to increase the safety of travelling in motor vehicles and reduce the traffic injuries caused by human factors such as speeding and distraction. Reference [24] also lists automated city mapping as another application of SLAM.

CLAM can be applied to the problem of automated ocean exploration as done by the authors in [23]. The authors in [13], as well as the authors in [18], have successfully used CLAM in the RoboCup Soccer game, which is a soccer game played by robots.

2.12 Summary

This chapter presented the background knowledge necessary to understand and implement a working CLAM framework. The chapter began with general definitions and explanations of SLAM and CLAM. A list of problems that need to be solved in order to solve the CLAM
problem was presented and includes topics such as: each robot’s role, centralization, aggregation, and communication methods. Each of these issues were defined and discussed in that section.

The issues that relate to the set-up of the landmarks in the environment are mentioned as well. These include problems such as unique versus non-unique landmarks, distance between landmarks, elapsed time between landmark observations, and static versus dynamic environments.

Methods of data association, such as the multi-hypothesis tracking with pruning and the joint compatibility branch and bound test are looked at.

The Kalman filter, the Monte Carlo particle filter, and the Markov localization algorithms are discussed.

In the map building section, topological maps and occupancy grid maps are discussed and compared, and hybrid maps are brushed over.

The map merging sections, explains map merging in static and map merging in dynamic environments.

The generic robotic programming framework and the SLAM framework, which could be used in this research project, are investigated.

Finally, this chapter ends with list of real-world applications of the SLAM and CLAM problems.
Chapter 3

Design Decisions

The design of the cooperative mapping and localization framework requires many decisions to be made and constraints to be put in place in order to obtain a working implementation in a limited time frame. This section lists those decisions along with the issues that were encountered during the project implementation and the actions taken to correct those issues.

3.1 Landmark Extraction

The first of these issues deals with the landmark extraction phase where the robot’s sensors record landmark co-ordinates in the environment. The landmarks that will be used to test the project have to be unique in order to avoid miss-matching with other landmarks in the environment. This is the simplest and quickest option to implement, but, as mentioned in the Background chapter, there are methods that allow one to work with non-unique landmarks in the environment. The distance between landmarks must be above a specific minimum, which can be set in the landmark extraction phase, to avoid multiple landmarks being interpreted as one.

Nevertheless, the landmarks will be placed as close as possible so as to minimize the amount of time between observations in order to avoid high levels of uncertainty in the localization algorithm. High levels of uncertainty in the localization algorithm could cause a robot to get lost.

To avoid further uncertainty in the localization algorithm, the landmarks in the environment will be kept stationary. However, this adds a complication to the co-operating
mapping and localization part of the project: if the landmarks in the environment must be kept stationary, how does one account for multiple moving robots in the environment? The authors in [9] use a coloured strip around each robot’s base, which is detected by cameras mounted on each robot. If the robot detects another robot in its line-of-site using the camera system, it will disregard any landmark observations that relate to the observed robot’s position. This gives the robots the ability to not only detected each other, but also to determine whether they have seen that specific robot before and to keep track of which map segments came from which robots. Since adding this ability requires the use of cameras with an image recognition system, this method is beyond the scope of this project. Instead of using this method, we will only be working with one robot, which performs separate slightly overlapping runs of the environmental mapping procedure in order to simulate using multiple robots in the environment. This avoids the problem of needing to detect a moving robot in the environment.

3.2 Map Building

To simplify the process of map merging, it is assumed that each robot is equipped with a compass for directional readings. Each robot will build its segment of the map relative to the North compass reading so that the y-axis of the map is aligned to the North direction. This is done, as explained in the map merging section, in order to reduce the amount of computation necessary to merge occupancy grid maps since no rotations are required if all map segments are aligned to the North direction.

The RANSAC algorithm is used to clean the landmark data, which is collected relative to the North direction, of any outliers and to find the data points that fit the least squares model. This cleaned data is then used to build an occupancy grid map.

Before the occupancy grid map can be built, the resolution of the map must first be chosen by choosing the number of columns and rows to use in the grid. Higher resolution results in more detailed representations of the environment, but at the cost of higher storage requirements.

The minimum and maximum x and y values are determined from the cleaned data and these values are used to initialize the grid so that the grid’s rows and columns are equally spaced over the range of x and y values in the cleaned data set.

Since the data has been run through the RANSAC parameter estimation algorithm using the least squares model, all the outliers have been removed and a single line of inliers
for that subset of the data is available to be added to the occupancy grid. Since the outliers have been removed by the RANSAC algorithm, it is no longer necessary to use probabilities and a threshold value to determine whether each cell in the occupancy grid is occupied or not since, with RANSAC, if there is a data point in a cell then the cell is occupied and if there is no data point in a cell then the cell is empty.

Figure 3.1 illustrates an occupancy grid map with the lines that the RANSAC algorithm would choose using example range data and figure 3.2 shows the occupancy grid map with the occupancy for each cell calculated after the RANSAC algorithm has been executed. From contrasting the RANSAC approach in figure 3.2 with the pure occupancy grid map approach in figure 2.9, it can be seen that using the RANSAC method with occupancy grid maps results in less cells containing data, which could lower the high storage requirements for occupancy grid maps. This is due to the RANSAC algorithm removing the outliers from the data set.

3.3 CLAM

As mentioned in the background chapter, the additional issues that CLAM has to solve on top of SLAM are each robot’s role, centralization, aggregation, and communication methods [4].
3.3. CLAM

Figure 3.2: Occupancy grid map built after the RANSAC approximation model has been run on the example data.

3.3.1 Role

For this project, each robot’s role will be that of an independent entity running its own instance of the map building and map merging procedure. Each robot will be able to communicate its segment of the map to another robot following the rules described in the map building section and communication section of this chapter. Each robot will be completely independent of any other robot and will communicate at specified communication points in order to perform map merging.

3.3.2 Centralization

Centralization/Convergence concerns the question of how the robots will converge upon each other once they have completed mapping their respective sections of the environment.

It will be assumed that the robots will map their local environment and, after both robots have performed map merging, co-ordinate their actions to map any areas of the global environment that they have no landmark data for. Once the robots are certain that there are no more unexplored areas of the environment then they can cease their mapping operation. They could reach this certainty by, for example, realizing that they are enclosed by four walls with no further avenue for exploration. The coordinated exploration feature
is left as a future extension to the project and is listed in the extensions section of the conclusion chapter.

### 3.3.3 Aggregation

Aggregation is concerned with how to combine the different sections of the map, which have been gathered by each of the robots, so that each section is in its correct place in a global map. This is more commonly referred to as map merging.

As mentioned in the map building section of this chapter, each robot will build a map of its local section of the environment using a compass direction as a common reference point to all the robots in the team. This allows all the collected map segments to be axis-aligned so that the process of map merging will be more efficient and simpler to implement.

Map merging without a common reference point is a computationally expensive operation due to the large number of rotations and translations that have to be performed in order to align the map segments. Figure 3.3 shows 3 rotations of an occupancy grid map, containing RANSAC calculated least squares lines. In order to merge two occupancy grid maps, the second map is translated over the first map, iteratively, for each cell in the map from top left to bottom right. For each cell translation, the second map will need to be rotated 360 degrees, counting the number of matching cells in both maps for each rotation. The number of rotations per translation is directly proportional to the resolution of the maps (the number of rows and columns). Therefore, removing the need for rotations in the map merging process can significantly reduce the amount of computation required to align maps.

### 3.3.4 Communication

Since a single robot will be used in this project, communication points will need to be established in order to facilitate map merging. In a test environment with multiple robots, these communication points would represent places in the environment where a robot comes within line-of-site of another robot and is able to identify that robot. At these communication points, the two robots involved in the current communication will transfer their respective local map segments to each other in order to perform map merging. After
map merging both robots will have a single local map segment, which is a combination of both maps, and will be able to continue their exploration of the environment.

Transferring the entirety of a robot’s map segment to another robot would work in the case of a small environment, but with a large environment the amount of map data to be transferred might take a long time to transfer, especially when using occupancy grid maps as mentioned in the map building section of the background chapter. This project will side-step this problem by using a small environment for testing, but this could be a possible extension to the project and is listed in the extensions section of the conclusion chapter.

If the maps of two robots at a communication point are the same then there is no need for map merging to occur. In order to check for this, each robot could run their respective map segments through a hashing function and then, when communication takes place, these hash values can be compared to determine whether map merging should occur or not. If the hash values are equal then there is no need for map merging to take place since the maps will be the same. This is simple future extension to the project and is listed in the extensions section of the conclusion chapter.

Another communication issue, which could lead to improved mapping performance if solved, is whether the robots should collaborate in order to more efficiently map sections of the environment that they both know to be unexplored from the map merging process. If both robots know that two sections of the environment are yet to be explored then
there is little point for both robots to go to one of those points. Instead, to improve mapping efficiency, each robot should, after performing map merging and identifying the unexplored areas, collaborate to map the unexplored areas separately. This is an interesting future extension to the project and is included in the extensions section of the conclusion chapter, but will not be implemented in this current version.

Since a single robot will be used in this project, instead of it communicating directly with another robot, the robot will explore and collect landmark data representing a small section of the environment, which will then be saved to the database with timestamps, which keep track of when the data was recorded. The robot will then explore and collect landmark data for a slightly overlapping section of the environment and this will also be saved to the database with timestamps in order to keep track of when the data was recorded so that it can be extracted at the right time later by using a timer. When map merging occurs, the timestamped data will be read from each database concurrently and used to build separate map segments, which will then be used for map merging where both generated maps will be combined into one.

In a test scenario with multiple robots, the robots would need some means of communicating with each other. Networking sockets could be used with some form of networking technology or a messaging queue such as ZMQ can be used with some form of networking technology to easily and reliably transfer map segments between robots.
Chapter 4

Implementation

This chapter explains the implementation of each of the components in this research project. It begin with a discussion of the hardware set-up used, followed by an explanation of the programming language and frameworks used. Finally each module, with containing classes, is discussed.

4.1 Hardware Set-up

For the development and testing of the framework, we decided to use the FischerTechnik robotic kit, with the Fischertechnik ROBO TX Controller, which was used by Shaun Egan in his SLAM project, in [4] the year before. The FischerTechnik robotic kit consists of a set of lego-like pieces, which are used to build the frame that the other components will be attached to.

The Fischertechnik ROBO TX Controller is stated to have an integrated Bluetooth radio interface operating at 2.4 GHz with a range of 10 m [5], but during testing it was discovered that it had an effective range of 4 m. We contributed this decrease in range to the Bluetooth stack on the Bluetooth adapter that was being used and decided that it should not cause any problems since testing of the mapping capabilities of the project should still be possible within the 4 m range. The main benefit of using Bluetooth in this project is that Bluetooth is capable of keeping several connections alive concurrently, which allows many robots to be controlled from the same Bluetooth adapter.

The Bluetooth radio interface allows programs running on a computer to interact with the robot’s controller in order to receive sensor input or control the robot’s motors. Programs
4.2. PROGRAMMING FRAMEWORKS AND LANGUAGES

can be written that make use of the Fischertechnik supplied DLL to send commands to
the robot’s controller via the Bluetooth link.

According to reference [4], the DLL checks the robot every 12 milliseconds in order to
update its local copy of the robot’s state and to issue commands to the robot.

On the front of the robot there are two Fischertechnik ultrasonic range finders, which
have an arc width of 60 degrees, a range of 5 cm to 4 meters, and a scan resolution of 1
cm. The two ultrasonic range finders are separated by a gap of 4.6 cm [4].

The two encoder motors are mounted in a differential drive set-up with a gear ratio of
2:1 and are fitted with tracks [4]. Due to the poor clipping of the lego-like pieces, the
robot could not be used on any carpeted surfaces as the friction between the tracks and
the carpet would cause the gears to slip out of place.

The battery used by the robot is a 8.4 V 1500 mAh battery [4].

A D-Link DBT-122 Bluetooth adapter was used on the computer to communicate with
the Bluetooth radio interface on the Fischertechnik ROBO TX Controller.

To control the movement of the robot, a Microsoft Wireless XBox Controller was attached
to the computer and the input was routed through the DLL to the robot. Since the
two analogue sticks on the XBox Controller serve different purposes, the Point-Of-View
analogue stick, which is the left analogue stick when holding the XBox Controller, was
used to control the differential drive system of the robot by combining the “x” and “y”
axis of the XBox Controller’s point-of-view analogue stick.

4.2 Programming Frameworks and Languages

The goal of this project is to implement and test a cooperative mapping and localization
framework that will work with two robots to begin with, but can easily be extended to
work with more. Even though the project is designed to work with two robots, only one
robot will be used, which will simulate the running of two robots by building separate
maps on separate runs of the project.

To simplify the task of interacting with the robot, the generic robotic programming frame-
work that was created by Leslie Luyt in [14], which is summarized in the Robotics Frame-
works Section of the Background chapter, will be used. This framework includes modules
for movement, sensor management, and artificial intelligence, which simplify the task of running programs on additional robot platforms without any modification.

The odometry, landmark extraction, and data association components of this research project are provided by the SLAM framework that was created by Shaun Egan in [4] and summarized in the Robotics Frameworks section of the Background chapter.

The programming language that is used for this research project is the Python programming language, which, at the time of this writing, is at version 3.2. The frameworks described above were created in Python 3.1, which does not pose a problem since Python 3.2 is backwards compatible with Python 3.1. The choice of Python as the programming language is, to some degree, influenced by the availability of the aforementioned frameworks, but more so by Python’s intuitive language syntax and intuitive data structures, which speed development and reduce the possibility of error since programs written in python are significantly shorter than programs with similar functionality written in another language such as C, C++, C#, or Java [14].

### 4.3 XBoxUtils

This module consists of a class called XBox. This class is responsible for capturing user input from the Microsoft Wireless XBox Controller that is used in this project and for sending that input to the SLAMController class. The SLAMController class uses that input to control the robot through the motor manager interface of the generic robotic programming framework from [14].

To interact with the Xbox Controller, the Python Pygame module is used. Pygame is a cross-platform Python library that was created to simplify the task of creating multimedia software, such as games, in Python. Pygame uses the SDL (Simple Directmedia Library) multimedia library, which is a cross-platform C library for performing multimedia functions. It has modules for input handling using the keyboard, mouse, and joystick. It has a display concept called surfaces, which can be used to draw simple shapes and to rotate and scale pictures.

The XBox class uses the Pygame joystick module to capture input from the XBox Controller and uses surfaces to draw a visual representation of the XBox Controller’s input to a window. The left analogue stick on the XBox Controller is used for input since the right analogue stick was found, during testing, to be inadequate for the purpose of controlling
the robot due to it serving a different purpose on the controller. The left and right axis of the left analogue stick were combined to retrieve correct readings to control the left and right tracks of the robot. The y-axis determines the maximum forwards or backwards speed to be used and the x-axis determines the values to be added or subtracted from the maximum forwards or backwards speed for input to the left and right robot tracks. So, for example, if the Y-axis had a value of 0.6 units (positive values representing the forward direction) and the x-axis had a value of -0.2 (negative values representing the left direction), then the left robot track would receive an input of 0.4 (0.6 - 0.4) and the right robot track would receive an input of 0.8 (0.6 + 0.2). In the SLAMController class, the input to the tracks is multiplied by the maximum speed value, which is set in this class, and then sent to the motor manager interface of the generic robotic programming framework. In order to use the Pygame module from within Python, the Pygame package must first be installed.

Pygame was chosen to read input from the XBox Controller over the Microsoft XNA framework because Pygame is cross-platform and allows the XBox class to be written entirely in Python. If Microsoft XNA was used, the XBox class would have to be written in C#.

The XBox class uses the Python ZMQ (Zero Messaging Queue) module to send the XBox input to the SLAMController class over a network. ZMQ in the XBox and SLAMController classes is set up in the publish/subscribe model, which requires that the XBox class publish data using a topic string and the SLAMController class subscribe to that topic string in order to receive the input data. The topic string used for this purpose is the string “Controller”. The SLAMController class will block until input is available on the queue. This does not mean the robot will stop moving though since the robot will keep following the last command issued to it until a new command is available. In order to use the Python ZMQ module from within Python, the pyzmq package must first be installed.

4.4 DatabaseUtils

This module contains the SLAM_Database class. This class contains methods to read and write the data that is used in the project to a database file. The SLAM_Database class creates a new database, with an auto-incremented file-name, for each run of the project. These database files are stored in the SLAM_Database_data subdirectory from the directory where the SLAM_Database class is stored. Each database file consists of the “data” table, the “ransac” table, and the “map” table.
4.5. RANSACUTILS

The time-stamped raw data from the Landmark Extraction phase is written to the “data” table.

The cleaned data from the RANSAC phase, using the RansacUtils module described in the RansacUtils section, is written to the “ransac” table.

The occupancy grid map data from the map building phase, using the MapBuildUtils module described in the MapBuildUtils section, is written to the “map” table.

The database used for this purpose is the SQLite database. SQLite is a C library that provides a minimum resources, disk-based database that does not require a separate server process and allows accessing the database using the SQL query language.

In order to use the SQLite database from within Python, the SQLite3 module must be imported into the SLAM Database file. The SQLite3 module is packaged as a standard part of the Python 3 distribution.

4.5 RansacUtils

This module contains the RANSAC class, which is an implementation of the RANSAC algorithm discussed in the background chapter. The Python code for the RANSAC class is adapted from the code made freely available, under the GNU General Public License, by Luis J. Manso, available at [15].

The values of the three predefined parameters, mentioned in [6], that are used on the RANSAC algorithm are as follows:

1. The error tolerance that is used to determine whether a point fits a particular model is set to a value of 35
2. The threshold, which specifies the number of correct data points necessary to say the correct model has been found, is set to a value of 11.
3. The number of times to repeat the algorithm to find a valid model is set to a value of 50.

The class starts by reading the raw data from the database, which was populated by the landmark extraction phase, and passing it to the RANSAC method. The RANSAC
method removes the outliers and estimates the parameters that fit the least squares model. The data points, inliers, which fit this point, are written to the “ransac” table in the SLAM_Database.

The RANSAC method takes, as its parameters: the points upon which to perform parameter estimation, the number of times to repeat the algorithm, the minimum number of points necessary for the smoothing (averaging) model, the error tolerance for determining whether a point fits the model, and the threshold value for determining whether the correct model has been found. The RANSAC method returns: the starting point of the least squares line, the ending point of the least squares line, the inlier points, and the outlier points.

The least squares model, which is used by the RANSAC algorithm, takes a list of points as its parameter and returns the starting point and ending point of the averaged line.

4.6 MapBuildUtils

The MapBuildUtils module contains classes for creating and using maps. It contains the OccupancyGridMap class, which is an implementation of the occupancy grid map and contains methods for creating, managing, and using occupancy grid maps.

The resolution of the occupancy grid map can be set with the “rows” and “columns” variables in the OccupancyGridMap class. These variables are used, together with the maximum and minimum data points that will be added to the map, to produce the occupancy grid. Private variables are used to hold the widths and heights of the columns and rows, which are calculated from the number of rows and columns, and the maximum and minimum data points. Since all the grid cells have the same size, there is only one set of width and height values for the columns and rows.

The OccupancyGridMap class contains the private CalculateProbabilities method, which is used to calculated the probability that each cell is occupied. This method is called automatically when data is added to or deleted from the occupancy grid map, either through the class constructor or through the Add or Remove methods. The only probabilities that will be used in this project are 100% for occupied and 0% for unoccupied due to the data points being cleaned by the RANSAC algorithm and therefore containing no outliers.

Data points can be added and removed through the Add and Remove methods respectively and both of these take the data points that will be operated on as parameters.
Individual occupancy grid cells can be addressed via column and row indexes and there are methods for retrieving and deleting the entire occupancy grid.

In a future implementation, interfaces will be used so that different mapping techniques besides occupancy grid maps, such as topological maps, can be used and programs can interact with these maps through the standard interface.

4.7 MapMergeUtils

The MapMergeUtils module contains the OccupancyGridMapMerge class, which is used to merge two occupancy grid maps.

The OccupancyGridMapMerge class has a MergeMaps static method that takes the two maps to be merged as parameters and returns the merged map as the result. This method works, conceptually, by leaving the first map as it is and, iteratively, translating the second map over the first map from left to right and top to bottom while counting the number of matching cells at each iteration. At the end of the translations, the iteration with the highest number of matching cells is chosen as the correct match. The two maps are combined, at the iteration with the highest number of matching cells, into a single map that is big enough to hold the combined data points from both maps and this single map is returned as the result of this method.
Chapter 5

Results

This chapter introduces the tests that will be used and the results obtained, along with a discussion of the results obtained in this research project.

5.1 Introduction

To begin working with the robot used in this project, a method of input had to be devised. As explained in the design and implementation chapters, a Microsoft Wireless XBox Controller is used to provide test input to the robot and the research project. A graphical display, made using the Pygame library in Python, was used to monitor the input from the XBox Controller. The initial results were not very promising as the controller did not seem to be providing valid input for the right analogue stick. Eventually, after more testing, it was found that the left analogue stick and the right analogue stick serve different purposes, therefore the left analogue stick was used as the only input to the project by combining its x and y axis. This provided valid input data that was then used to drive the simple, lego-like robot through a series of simple obstacle courses.

The performance of the SQLite3 database was tested by, iteratively, sending data to it as fast as possible. This was tested on a 4 year old laptop. On linux, the SQLite3 database could keep up with the data that was being sent to it, even without a time delay. On windows, however, it was discovered that the SQLite3 database could not keep up with the speed that data was being sent to it unless a time delay was in place. It was assumed that this result is due to the timer on windows having a lower resolution than the timer.
used on linux. In practical use of this project, however, this should not be a problem since
the FischerTechnik supplied dll, for interfacing with the robot, has a 12 ms poll time [4].

Landmark data and odometry data was then captured by controlling the robot in this
way and saved to the database under the “data” table.

The collected landmark data was then used to test the RANSAC algorithm, using the
least squares model, and the cleaned landmark data that was generated by the RANSAC
algorithm was saved to the database under the “ransac” table.

This cleaned landmark data was then used to build the occupancy grid map, which was
then written to the database as well.

Finally, the occupancy grid map was merged with another occupancy grid map in order
to generate a single, combined map.

5.2 Results

As can be expected, the landmark data that was captured using the robot’s sonar sensors
contained a high amount of errors and these had to be removed, by using the RANSAC
algorithm, before map building and map merging could take place.

The RANSAC algorithm was able to determine the correct line of data, even with a high
amount of outliers. The outliers were discarded and the inliers were saved to the database
under the “map” table.

Figure 5.1 illustrates the graphical output of the RANSAC algorithm implemented in
the code adapted from [15]. The example data points consist of a consistent line of
points starting from the bottom left and extending to the top right. There are multiple
outliers placed randomly on the image to simulate real data captured from robot sensors.
The red line in figure 5.1 represents the least squares model that, as can be seen from
the deviation from the expected line of data, poorly estimates the correct parameters.
The blue dashed line in figure 5.1 shows, on the other hand, that the RANSAC algorithm
performs exceptionally at estimating the correct parameters since the line does not deviate
from the correct data line, which starts at the bottom left of the image and extends to
the top right of the image.
Figure 5.1: RANSAC algorithm run on example data points. The red line represents the least squares model and the blue dashed line represents the RANSAC algorithm using the least squares model.
Chapter 6

Conclusion

This chapter concludes the research project with a summary of the project, followed by the outcomes of the objectives, and, finally, a list of possible future extensions to the research project are given.

6.1 Project Summary

The goal of this research project is to implement a CLAM framework that could eventually be used with a team of robots to allow them to work together to explore and map an environment. The research project will delve into the details of the components required for a functional CLAM implementation, but will focus more on the aspects of map building and map merging for this specific implementation.

In order to build a map of the environment, the robot was assumed to have a compass so that the maps could be built relative to a common directional reference point. This was done in order to minimize the amount of computation required to merge maps since, if all the map segments are already axis-aligned, then no rotations are required when trying to determine how the various map segments fit together.

6.2 Outcomes of Objectives

Since the framework, from [4], that is being used for the SLAM components of the project does not contain a localization algorithm, the data collected from the environment via
6.3. EXTENSIONS

the landmark sensors had to be run through the RANSAC algorithm in order remove incorrect data readings (outliers) and match straight lines to the correct data readings (inliers), therefore the RANSAC algorithm, using the least squares model as its base model, was implemented and tested in this research project. Results comparing the RANSAC algorithm to other, older parameter estimation techniques, such as the least squares model by itself, show that RANSAC accurately represents the true fit of the correct data whereas the least squares model does a very poor job of estimating the correct parameters.

The research project was implemented in the Python 3 programming language and, as can be seen from the implementation chapter, a sound object-oriented methodology, containing modules of classes, was used so that the project could easily be maintained and extended whenever needed.

The implementation of the map building and map merging components use occupancy grid maps as their underlying data structure. The map merging process is simpler to implement and more efficient since the use of a compass to provide directional context to the map building process removed the need to perform any rotations on the maps due to the maps are already being axis-aligned.

A method of interfacing with a Microsoft Wireless XBox Controller, in order to provide test input to the robot, was implemented using the Pygame multimedia library for accessing the XBox Controller and for displaying a graphical representation of the input to the screen. The pyzmq module was used for the communication of the input from the XBox Controller to the SLAMController class, which provide control instructions to the robot via the generic robotic programming framework that was used.

6.3 Extensions

There are many areas of this project that can be extended and improved and these extensions are listed below.

6.3.1 Unique versus Non-unique Landmark Observations

One of the assumptions that his project makes is that landmarks should be unique so that they do not get miss-associated with other landmarks already in the map. It would be
much more useful if the framework were able to detect chair legs as being separate rather than confusing them as one and thereby increasing the uncertainty of the localization algorithm.

This can be solved by either, implementing an Importance Sampling based approximate solution and implicit hypothesis pruning, as used in reference [18], or, as mentioned in [18], by removing landmark ambiguities by using a compass to provide directional context to non-unique landmarks. Since this project is already using a compass on each robot for building maps relative to a common reference point, the latter option might be easier to implement than the former.

6.3.2 Communication

1. This extension deals with the amount of map data to transfer between the two robots at communication points. Only the map data that is different from the other robot should be transferred at communication points in order to keep the amount of time necessary to transfer the map data to a minimum.

2. Only transfer map data if the two maps are dissimilar since map merging is not necessary if the maps are the same.

3. Collaborating to explore sections of the environment both robots know to be unexplored in order to improve mapping efficiency.

6.3.3 Mapping

1. Implementing and comparing different mapping techniques, such as topological maps. It will be interesting to compare the merging performance of different mapping techniques, such as occupancy grid maps and topological maps.

2. Implement map merging without a common reference point and compare the performance to the current implementation. The maps in this project are built according to a common compass reference point and this should increase the speed of map merging since no rotations need to be performed, but how much of an improvement this provides over map merging without a common reference point still needs to be measured.
3. The algorithm discussed in [8] to build maps in a dynamic, populated environment can be implemented in order to track robots in the environment and use their positions as reference points in the map merging process, as well as to track and remove other moving objects such as people from the localization algorithm.

### 6.3.4 Localization

Since the framework, created in [4] and used for SLAM in this project, does not contain a localization algorithm, it would be a very good future extension to the project to add one so as to add the ability to recover from errors in the landmark sensors or the odometry sensors. Reference [4] suggests that the Extended Kalman Filter or the Monte Carlo particle filter be implemented in order to obtain a fully functional SLAM framework.
Bibliography


