OPTIMIZATION OF MAPPING ALGORITHM FOR MOBILE ROBOT

SHARIFAH AIDA AFIQAH BINTI SYED ABD RAHMAN ALZAWAWI

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

OPTIMIZATION OF MAPPING ALGORITHM FOR MOBILE ROBOT

SHARIFAH AIDA AFIQAH BINTI SYED ABD RAHMAN ALZAWAWI

This report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Electronic Engineering with Honours

Faculty of Electronic and Computer Engineering
Universiti Teknikal Malaysia Melaka

2019/2020

DECLARATION

I declare that this report entitled “Optimization of Mapping Algorithm for Mobile Robot” is the result of my own work except for quotes as cited in the references.

Signature : ……

Author : Sharifah Aida Afiqah Bnti Syed Abd Rahman Alzawawi

Date : 25 June 2020

APPROVAL

I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in terms of scope and quality for the award of Bachelor of Electronic Engineering with Honours.

Signature : ……

Supervisor name : Dr. Norhidayah Binti Mohamad Yatim

Date : 25 June 2020

DEDICATION

To all my beloved family and friends, I couldn’t have done this without you. Thank you for all of your support along the way.

ABSTRACT

Autonomous mobile robots can safely explore their environments without hit or crashing with anything. Simultaneous localization and mapping (SLAM) helps the robot to gain such independence by answering questions about how its environment map looks like and also where is it location. Building maps is one of mobile robots' most fundamental tasks. Actually, the robot can build maps from the information and data collected by a mobile robot by applying a sensor like a laser sensor. The autonomous mobile robots can be located and navigate in the practical environment with the maps. The goal of this research is to investigate parameters such as grid size of the occupancy grid map algorithm and to stimulate the mapping algorithm and investigate performance based on map accuracy. Occupancy grid map is used as a map representation in this study because it can generate a discrete grid representation. This means, the robot can run on a given region in a grid to speed up the computation process. In addition, Gmapping is the SLAM algorithm used in this research as it is the most accurate method compared to other methods and it can provide a high quality performance output map.

ABSTRAK

Robot mudah alih autonomi boleh menjelajahi persekitarannya dengan selamat tanpa bertembung dengan orang atau berlanggar dengan objek. Penyetempatan dan pemetaan serentak (SLAM) membantu robot mencapai kebebasan tersebut dengan menjawab persoalan tentang bagaimana ia kelihatan dan juga di mana ia berada.. Pemetaan bangunan adalah salah satu tugas paling penting robot mudah alih. Sebenarnya, robot boleh membina peta daripada maklumat dan data yang dikumpul oleh robot mudah alih dengan menggunakan sensor seperti sensor laser. Robot boleh ditempatkan dan mengemudi dalam persekitaran praktikal dengan peta. Matlamat penyelidikan ini adalah untuk mengkaji parameter seperti saiz grid algoritma peta penghuni grid dan untuk merangsang algoritma pemetaan dan menyiasat prestasi berdasarkan ketepatan peta. Peta grid penghunian digunakan sebagai perwakilan peta dalam kajian ini kerana ia boleh menjana perwakilan grid diskret. Ini bermakna, robot boleh berjalan di kawasan tertentu dalam grid untuk mempercepat proses perhitungan. Di samping itu, Gmapping adalah algoritma SLAM yang digunakan dalam penyelidikan ini kerana ia merupakan cara yang paling tepat berbanding dengan kaedah lain dan ia dapat memberikan peta keluaran prestasi yang berkualiti tinggi.

ACKNOWLEDGEMENTS

I would like to thank Allah S.W.T. for his abundant blessings and unfailing love for me. Here, I express my deepest appreciation for supporting me throughout my degree study to several individuals and organizations. First of all, I wish to express my sincere thanks to my supervisor, Dr. Norhidayah Binti Mohamad Yatim for her patience, enthusiasm, insightful feedback, helpful suggestions useful information, practical advice and unceasing ideas that have supported me tremendously in my research and writing of this dissertation at all times. Her deep knowledge, vast experience and professional expertise have enabled me to successfully complete this research. I am truly thankful to her for the time and energy she has spent in guiding me, answering my questions, correcting and improving my thesis report. This thesis would not have been possible without her guidance and help. I never dreamed having a great supervisor in my research.

Furthermore, a special thank you goes to the Universiti Teknikal Malaysia Melaka (UTeM) for accepting me as a graduate. I am also deeply indebted to the Higher Education Ministry, Malaysia for awarding me the JPA scholarship. This financial support has allowed me to finish my degree study effectively. Besides, I would like to extend my sincere thanks to Professor Dr Mohd Shakir Md Saat, Dean of the Faculty

of Electronic and Computer Engineering (FKEKK), UTeM for his consistent support and help in my undergraduate study. Next, I also appreciate the kindness, hospitality and technical support of FKEKK's lecturers and staff. In addition, I would like to acknowledge the FKEKK for organizing numerous seminars, which led me to develop my ability to write the report of the thesis.

Additionally, I also wish to express my deepest gratitude to my parents and family. My source of strength is their unwavering support and encouragement. I am also grateful to my friend, Husna. She's always there to take care of me, cheering me up and standing by me through my life's peaks and valleys. Last but not least, I give my thanks to all my friends for giving me their encouragement, friendship, moral support and advice.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF GRAPHS

LIST OF SYMBOLS AND ABBREVIATIONS

SLAM	:	Simultaneous Localization and Mapping
AMRs	:	Autonomous Mobile Robots
AGVs	:	Automatic Guided Vehicles
OGM	:	Occupancy grid maps
GPS	:	Global Positioning System
2-D	:	2-Dimensional
3-D	:	3-Dimensional
ROS	:	Robot Operating System
ML	:	Maximum Likelihood
EM	:	Expectation Maximization
KF	:	Kalman Filter
EKF	:	Extended Kalman Filter
SBCs	:	Single Board Computers
LIDAR	:	360 Laser Distance Sensor Lds-01
LRF	:	Laser Range Finder
RADAR	:	Radio Detection And Ranging

PF	:	Particle Filter
RBPF	:	Rao-Blackwellized Particle Filter
IoTs	:	Internet of Technologies
UNIX	:	Uniplexed Information and Computer System
OS	:	Operating System
LISP	:	List Processing
RAM	:	Random Access Memory
Rviz	:	Ros Visualization

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

This chapter covers significant background studies, problem statements, goals, project scope and the environment and sustainability of the project. In this part, it also provides an outline of the chapter.

1.1 Background Studies

Autonomous robotics has been an area of interest for researchers for a long time. Autonomous robots have the potential to decide their own alternative and move properly. An autonomous robot is one that can observe its surroundings act based on what it sees and is programmed to identify and after that respond on a gesture or control in that situation. For example, by regard to flexibility, this decision is including acts but not bound to the specified element such as starting, stopping, and maneuvering

around obstacles in their path. It is highly desirable to have a programmable robot with the ability to multi-task and traverse difficult terrains where a human being would otherwise be risky to cross. A large number of accurate sensors installed by the robot will measure and process information in fractions of the time it would take a person to do. Over the past few decades, this has become an ambitious research area.

As technology evolves, autonomous robots in the manufacturing, health care, security, military and similar sectors have become increasingly popular. The primary benefit of this technology is that, under certain circumstances, it can function without human intervention and still perform better. In other words, in activities that are risky, repetitive and require high levels of concentration over a long time, autonomous robots may replace humans. Unlike the robot for manual control, autonomous robot does not rely on live input commands. Autonomous robot makes decisions on its own based on inputs from sensors and pre-defined behaviors. It can therefore operate with minimal human oversight and correction. It therefore acts in its environment as an autonomous individual.

Autonomous mobile robots (AMRs) are a type of automatic guided vehicles (AGVs) that can be deployed without supporting any framework such as markers, wires or magnets embedded in the ground or laser targets that are precisely located. There are two types of AMRs that depend on choosing optimization based on fleet management and systems. Usually, fleet management systems work with larger payloads and guide the robots to a location from a source. In addition, pick optimization robots combine machine and human movement into an operation progress designed to improve choosing input. Pick robots for optimization support the picking of cartons and totes and therefore have a small payload.

To move through an indoor environment, autonomous indoor mobile robots are required. In order to be considered independent of an indoor mobile robot, it must have at least the following capabilities. First, within the indoor environment, it should be able to locate itself. Second, in order to reach its destination, it should be able to plan its path and create the environmental map. Third, if it deviates from the intended course, it should be able to correct itself. Fourth, new obstacles should be found and avoided. An autonomous mobile robot also needs many other capabilities, depending on its function and how the designer defines the word 'autonomous'. Simultaneous Localization and Mapping (SLAM) is a method to determine their current position, direction and the map of robot's environment. SLAM technology is well known, but still faces some limitations [1].

In this research project, the focus is about robotic mapping. This is because in the robotics and automation industries, robot mapping has develop actively to become an area of research [2]. Robotic mapping addresses the issue of using mobile robots to build spatial representations of physical environments. The problem of mapping is generally considered to be the most critical issues in pursuing the construction of autonomous mobile robots. It still presents great challenges, despite significant progress in this area. Currently, the robust methods for fixed, structured and narrow range mapping environments, mapping unstructured, dynamic, or large scale environments are the reasons why this problem become an area of research.

Mapping the environment using mobile robot has been done previously by multiple researchers. In these works, three types of map representation are studied such as occupancy grid map, topological mapping and features map representation. Occupancy grid maps (OGM) divide the map into cells which each construct a grid with a binary random variable showing whether the cell is occupied or not [3]. The

benefit of OGM is that due to moving objects, it can function with dynamically changing environments and still allow accurate modelling of it.

Feature maps use unique environmental features, each of which is marked by its position in the global map. To act as a landmark for the mapping process, these features must be static and distinguishable from the rest of the environment. This form of mapping makes localization effective but the data association process is not a simple job.

Topological maps are capable of representing a compact form of the map using a series of nodes and arcs. It only showing the abstract model. Topological maps need less storage and computation time compared to metric or grid maps. However, topological maps are more difficult to construct and may not be valid for map matching and may suffer from perceptual aliasing in recognition of the same location.

1.2 Problem Statements

The first issue of robotic mapping is the acquisition of a spatial model of the environment of a robot [2]. The maps for robot navigation (localization) are widely used. Robots must have sensors to help them to see the farther world in order to earn its own map. Cameras, range finders using sonar, laser and infrared technology, radar, tactile sensors, compasses and Global Positioning System (GPS) are widely used for this mission. All of these sensors are the reason why the errors, occurs and called measuring noise. Besides, the sensors of the robot also the reasons to rigid limitations of the distance. For instance, light and sound cannot pass through the walls. Such limitations of the scope make it necessary for a robot to have a mapping algorithm.

The second complicated problem of robot mapping begin from the large dimensionality of the map that has been create [2]. The reader can assume the numbers that maybe will be used to explain the environment in order to understand the dimensionality of the problem. An exact 2-dimensional (2D) plan often requires thousands of numbers. However, a comprehensive 3-dimensional (3D) graphical map of a house can take large of numbers quickly. The problem of mapping can therefore be extremely high-dimensional.

Third, the difficult robotic mapping problem is the issue of correspondence or known as the problem of data association [2]. The problem of correspondence is analyzing whether sensor measurements taken at various points in time correspond to the same physical object in the world. When the cycle is closed, the robot must find out where it is relative to its map that had been built before.

1.3 Objectives

This project focused on developing a mapping of the mobile robot's indoor environment to ensure that it can work well without any obstacles. This project's objective is as follows:

1) To investigate the parameters of occupancy grid map algorithm such as grid size.
2) To stimulate the mapping algorithm and investigate the performance based on map accuracy.

1.4 Scope of Project

Limitation or scope is important for producing a good report to enable us to achieve the objectives that have been set. There are few criteria to be met in this project to ensure the project will be complete within stipulated time and resources. First, to model the map environment for mobile robot, this project uses Ubuntu and Robot Operating System (ROS) technology. The TurtleBot3 in the ROS software will be selected as the robot because this robot has a various function to use. In ROS code, the programming language is C++.

Furthermore, as the method for mapping algorithms, this project focuses on an occupancy grid map because this method can produce a discrete representation. The robot will work in a grid on a selected region to speed up the process of computation. In this project, compared to a multi-robot, it will only focus on a single robot. This is because a single robot can increase productivity as it is programmed to accomplish the task by performing repetitive movements. Last but not least, only a static indoor setting was involved in this research. There is no use of the dynamic environment as it is a process that requires further steps to set up and deploy.

1.5 Environment and Sustainability

The autonomous mobile robot can be deployed anywhere like factory, hospital, and library that can help to reduce human jobs. This mobile robot, for instance, can move items from one location to another without following assembly line.

This autonomous mobile robot can increase flexibility as it is primarily used to control on-board sensors, not wires or magnetic tape. It can automatically create their

own paths within a facility from one stage to another to help them avoid obstacles. This may not cause any problems for mobile robot if there are a changing occur in the environment as the mobile robot can adapt it easier.

Besides, as we know, autonomous mobile robot is equipped with a sensor which will increase the safety. This help this mobile robot to perceive and understand its environment, allowing it to move effectively with any obstacles in collision. The robot also has the power to evaluate the steps to be done to perform a mission via a method of interpretation that helps it. It often includes a cognition unit or a control panel to manage all of the robot's subsystems. Autonomous mobile robots need to have a source of input information, a way of processing the information and a method of taking action to respond to an environment world including their own motion.

1.6 Project Outline

Chapter 1 shows how to generate ideas to carry out the project. A short introduction to the idea of autonomous mobile robot. Also included in this chapter are background studies and the problem that contributes to the project title proposal. In addition, the project's priorities are defined in order to determine the scope of the project. The significant of the project is also mentioned in this section.

Chapter 2 includes a review of historical literature and basic information on an occupancy grid map, as well as a previous observation. This chapter is critical for obtaining information from the analysis being conducted.

The methodology was the focus of Chapter 3. A set of research and flowcharts are included in this chapter. This chapter also discusses the steps to be taken in the simulation.

Chapter 4 explained more about the outcome and discussed the implementation of the software using software for Ubuntu and Robot Operating System (ROS). The simulation data sequence and analysis described in this part.

Chapter 5 concludes the comprehensive finding of the proposals for research tasks and prospects. Recommendation is also intended for further studies or research.

1.7 Summary

Chapter 1 is the first step in the implementation of a study or research project history and objectives. The explanation of the issue with the title must be relevant and sufficient so that we can effectively achieve the goals. The following sections are about reviewing the literature and the cases for performing the entire case analysis.

CHAPTER 2

LITERATURE REVIEW

This chapter deals with the review of literature. Reviewing literature is the act of reading, reviewing and summarizing specific topics picked. In this section, literature review can identify the project's benefits and disadvantages, identify discussions and assist with questions that require further research. This chapter will explore and analyze the study of mobile robot mapping algorithms.

2.1 Mapping Algorithm

The primary properties of some of the most popular algorithms are summarized in Table 2.1. In the field representation, the map representation is summarized and was defined clearly in depth discussion for each of the algorithms. In Table 2.1, it states how the resulting map shows uncertainty. The map is characterized by a Bayesian

posterior including its uncertainty. Since an estimate of Maximum Likelihood (ML) develop only one map, so it is fewer of an informative.

The convergence mention what is understood, under reasonable assumptions and the algorithm's convergence properties. For each method, it has a different properties of convergence. The convergence's notion in Expectation Maximization (EM) in particular is a basic one. A locally optimal map could be characterized by the resulting solution. A weak convergence in the table is to differentiate it from good results regarding the estimate's optimality.

In the next row, it is specified algorithm is subject to local minima or not. The field of incremental show if the maps able to created incremental or need multiple data passes. Incrementality is generally a logic property, particularly for robots to explore and built a map autonomously. It's an important field that requires poses. Only a subset of mapping algorithms where the robot poses are unknown, attack the complete mapping problem above. Other algorithms require information on the exact posture.

The type of the sensor noise is recorded as the map's dimensionality can be generated in practical implementation. The correspondence item function to determine whether an algorithm can cope with unknown correspondence issues and can accommodate similar looking features in the situation. Next, raw data represent the requirement in realistic implementations data pre-processing and filtering. An algorithm may create maps from the data of raw sensor. Calculated maps often have more information from raw sensor data. Lastly, the dynamic environments show the approach to dynamic environments is acceptable or not because there are approaches that can accommodate limited types of dynamics as indicated which was been stated as 'limited' in the table.

Table 2.1: Characteristics of the major mapping algorithms [2].

	Kalman Filter (KF)	**Hybrid**	**Occupancy Grids**
Representation	Landmark locations	Point obstacles	Occupancy grids
Uncertainty	Posterior poses And map	Maximum Likelihood map	Posterior map
Convergence	Strong	No	Strong
Local Minima	No	Yes	No
Incremental	Yes	Yes	Yes
Requires Poses	No	No	Yes
Sensor Noise	Gaussian	Any	Any
Can map cycles	Yes	Yes, but not nested	N/a
Map dimensionality	$\sim 10^3$	Unlimited	Unlimited
Correspondence	No	Yes	Yes
Handles raw data	No	Yes	Yes
Dynamic environment	Limited	No	Limited

Besides, Table 2.2 summarizes the advantages and disadvantages of occupancy grid map, topological mapping and features map. From Table 2.2, it can be concluded that an occupancy grid map is the best method to be used as the mobile robot mapping algorithm due to its flexibility to make no assumption about the environmental feature [4]. Therefore, in the application, the mapping algorithm can be used such as inspection where the actual environmental condition is needed for evaluation.

Table 2.2: Comparison of occupancy grid mapping, topological mapping and features map.

Map representation	Advantages	Disadvantages
Occupancy grid mapping	• Can be used to map effectively. It is possible to label the cell in an occupancy grid with the total distance to the goal. • A discrete representation of a grid. The robot will work in a grid on a selected region to incease the speed process of computation. • The mapping algorithm termination criterion is explicitly defined as each grid cell having a certain state assigned to it. • The histogram of the vector field can be used on the occupancy grid to avoid obstacles.	• The method is only suitable for mapping the local environment. The robot sensor's maximum range must be considered. • The map's scale in the memory of the robot increases with the exploration of the surroundings.
Topological mapping	• Topological mapping requires a small amount of memory space compared to a more complex map. • It is easy to combine two or more topological maps to construct a more complete map.	• Only good for a narrow area of utilization at the real world when geometry is not required. • Over-specified in the path planner perspective and with difficulty in finding the shortest route.
Features map	• Features map provides a means to assess both location and orientation of the robot. • Features map can be deployed in the environment or an agent can map features to improve navigation.	• The target must have the same appearance and the robot must be conscious of that appearance. • Map recognition among other information acquired is a challenging problem due to random object noisy perception.

2.1.1 Kalman Filter (KF) Approaches

A conventional method to map generation is based on filters from Kalman [2]. This technique was introduced by Smith, Self, and Cheeseman, who create a mathematical version of the approach, which is commonly used today. This method was further explored by a number of researchers in the following years. KF based mapping algorithms are used in the literature as Simultaneous Localization and Mapping (SLAM). SLAM is a concept which closely associated with the algorithms that use KF to estimate the map and robot location.

KF is one of Bayes filters which is most common implementations. There are two distinct phases of the KF such as prediction and update [5]. The phase of prediction estimates the state space from a previous iteration, while in the update phase the estimated state is combined with sensor observations. The outcome of the update phase is called posterior. The Extended Kalman Filter (EKF), which arises from the KF's prior development, solves the nonlinearity issue in the pose model of the robot. KF represent the Bayes filters covering posteriors $p(s_t, m \mid z^t, u^t)$ with Gaussians. In Gaussians, a small number of parameters can represent in a compact way because it is unimodal distributions. The Gaussian model is the full state vector x in the context of the robotic mapping problem, which includes the pose s of the robot and the map m:

$$x_t = (s_t, m)^T \tag{2-1}$$

In the equation above, ^{T} is assign to a vector or matrix being transposed. The location of the robot, s was modeled by three variables for robots operating. Let's denote these coordinates, respectively, by s_x, s_y and s_θ. The Cartesian coordinates of

feature sets are usually expressed by maps in the KF method. Appropriate feature in the environment may be landmarks, distinguishing objects or forms. The corresponding vector status given as 2K+3-dimensional vector, indicating the number in the map by K:

$$x_t = (s_{x,t}, s_{y,t}, s_{\theta,t}, m_{1,x,t}, m_{1,y,t}, m_{2,x,t}, m_{2,y,t}, \ldots, m_{K,x,t}, m_{K,y,t})^T \qquad (2\text{-}2)$$

Figure 2.1 displays a map picture obtained using the KF method. The route of an area measurements using a pencil sonar are shown in this image. The map itself is made up of fourteen point features which has been extracted from the data of the sonar and five of that features are thin and vertical artificial landmarks. Besides, the other corresponds to reflective objects in the environment. The covariance matrix Σ, has define the ellipses around these landmarks by shown the remaining of the residual uncertainty after mapping. The ellipses which of a single Gaussian (μ, Σ) show the posterior joint over all landmark locations and the pose of the robot. Multiple dots in Figure 2.1 which highlight theories about the position of multiple landmarks whose proof is too poor to be included in the map.

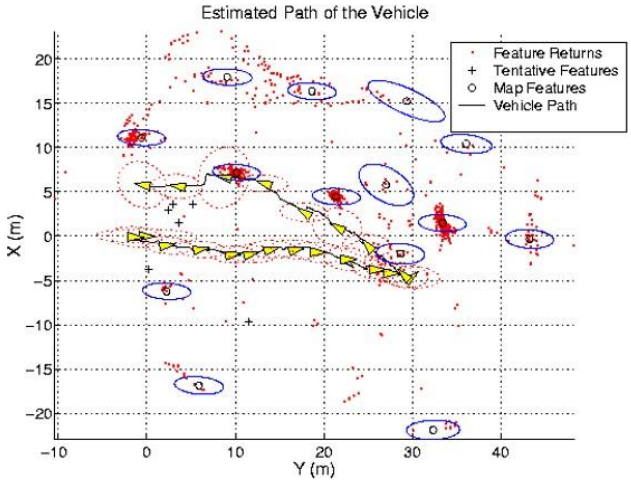


Figure 2.1: Kalman Filter estimation of the vehicle pose and the map [2].

Figure 2.2 illustrates the result from the example of simulation after successful mapping. In Figure 2.2b, the correlation (normalized covariance matrix) between the three-dimensional pose of the robot and the two-dimensional location of all 20 landmarks, is of great importance. The presence of the checkerboard indicates that there are strong correlations between the x-dimensions and y-dimensions of all position estimates [2]. The measurements only transmit data about the robot's proximity to landmarks and through integration among the landmarks. The absolute coordinates do not include in this measurement. The final result of the map is still ambiguous as shown in Figure 2.2a.

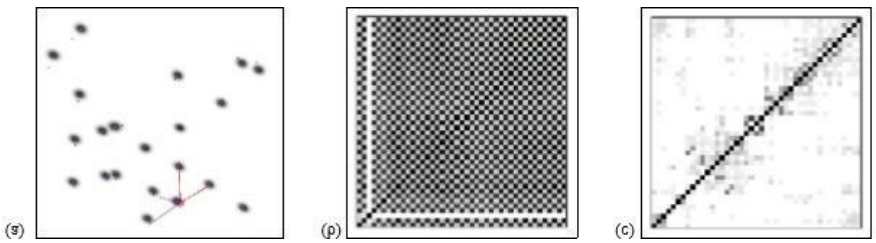


Figure 2.2: (a) Landmarks map obtained in simulation. (b) A correlation matrix after 278 iteration of Kalman Filter mapping. (c) The same estimate's standardized inverse covariance matrix.

2.1.1.1 Lu/Milios algorithms

The Lu / Milios algorithm is known as a recent extension of the basic paradigm [2]. Gutmann has successfully implemented this algorithm. The laser range data is somewhat specific to the Lu / Milios algorithm. This incorporates two simple phases of estimation, a phase in which Kalman Filters (KF) are used to quantify posteriors over charts and a phase in which distance measurements are combined with each other in multiple range scans.

The correspondence is obtained through the association of maximum probability data, example is the algorithm simply pairs up measurements nearby. However, the correspondence is calculated repeatedly by iterating both phases, allowing the approach to recover from incorrect correspondence.

In action, this method is capable of mapping unknown communication from raw data. Nevertheless, the fact that it uses the maximum likelihood of 'guess' correspondence rather than estimating the full back over correspondence and maps creates significant limitations. In reality, the initial pose of error measurements are low (less than 2 meters) the algorithm works incredibly well. It is not possible to accommodate larger poses errors, such as usually found when mapping a cyclic environment. This method is not an algorithm in real time as its need multiple passes through the data. Figure 2.3c shows a map generated from the range data. The final map is very precise and shows a detailed structure with more details.

For the Lu / Milios algorithm, the raw data is too inaccurate as shown in Figure 2.3a. it is from the robot's odometry for pose estimation. The phase’s failure of the Maximum Likelihood (ML) when using Kalman Filtering, in turn leads to wrong maps. This result illustrates the basic approach's strength and weakness. Note that Figure 2.3a to Figure 2.3b pre-alignment was performed using an algorithm.

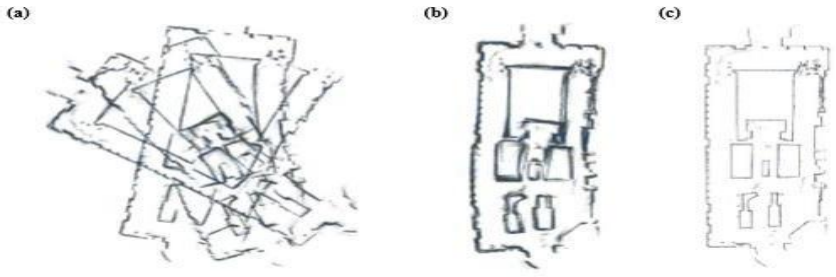


Figure 2.3: (a) Raw range data of large museum hall. (b) Artificial but indistinguishable information associated with Expectation Maximization (EM). (c) Application of the Lu / Milios algorithm to the pre-aligned data output.

2.1.1.2 Expectation Maximization (EM) algorithms

Expectation Maximization (EM) is a statistical algorithm developed by Dempster, Laird and Rubin [2]. A recent book on this subject reveals the abundance of literature on the EM algorithm that currently exists. The EM algorithm applied to the problem of robotic mapping has quite orthogonal features. EM algorithms used to solve the problem of mapping correspondence. In particular, EM algorithms function is to generate a consistent maps of a large-scale cyclic environment even cannot be perceptually distinguished. Indeed, EM algorithms cannot complete a concept of uncertainty. Alternatively, to find the most possible map, they conduct hill climbing in the space of all charts. To do so, multiple processing of the data is required. Therefore, maps cannot be created incrementally by EM algorithms.

EM aim is to determining a map and it is relatively simple if the robot's direction is known. EM includes two main steps. First step is the expectation or an E-step in which the posterior over robot poses are measured and second step is the maximization or an M-step in which EM calculates similar map provided these assumption of the pose. The outcome is an increasingly detailed sequence of maps, $m^{[0]}, m^{[1]}, m^{[2]}$. An empty map is the initial map, $m^{[0]}$.

The data set that has been mapped by using EM is shown in Figure 2.4a. The robot tests about 28 landmarks which correspond to corners, intersections and distinctive locations. However, no perceptual information is given to the robot for exercising mapping with unknown communication that would help disambiguate them. The error is too high in the pose calculation using odometry. So it is not suitable to use when traversing the wide loop in the field to solve the problem of correspondence. It is understood that such wide loops are difficult to chart. The consequence of applying

EM to this data set is shown in Figure 2.4c. The map and path that results a topologically right. To demonstrate the map's accuracy, Figures 2.4b and Figure 2.4d display occupancy grid maps based on sonar range measurements without and with poses calculated by EM of the raw range data.

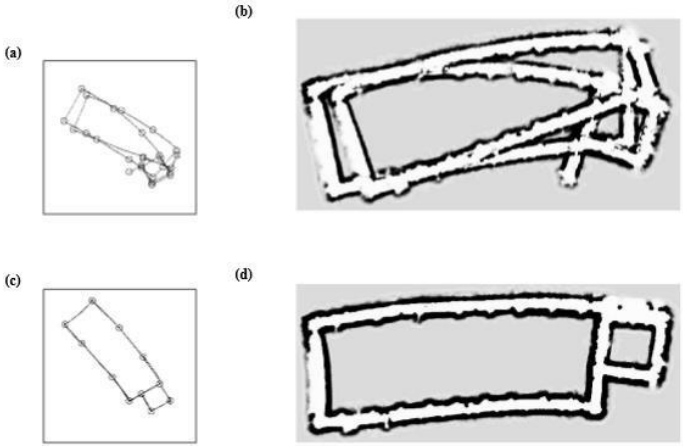


Figure 2.4 (a) Raw data with indistinguishable landmarks on a large-scale cyclic environment. (b) Occupancy grid map using sonar sensors built from raw data. (c) EM aligned map and path, proving the potential of EM to solve problems of hard correspondence. (d) Occupancy grid map create based on the EM mapping algorithm result.

2.1.2 Hybrid Approaches

Hybrid methods is the example of the famous approaches. It is the incremental Maximum Likelihood (ML) model from a statistical point of view inferior to both Kalman Filters and EM [2]. The purpose of this method is to build a single map incrementally when the data's sensor arrives without monitoring any residual uncertainty. A technique without an E-step can be interpreted as an M-step in EM. This framework has the advantage of being plain, which accounts for its popularity.

Figure 2.5 shows an example to map a cyclic system using the incremental ML method, including the equipment of the robot such as 2-Dimensional (2D) laser range finder. When the map is reasonably consistent until closing the loop, the large residual error contributes to discrepancies that the maximum incremental probability method cannot overcome. The weakness of this algorithms is it do not recognize ambiguity when constructing maps and it also do not have a method for using future information to change past decisions.

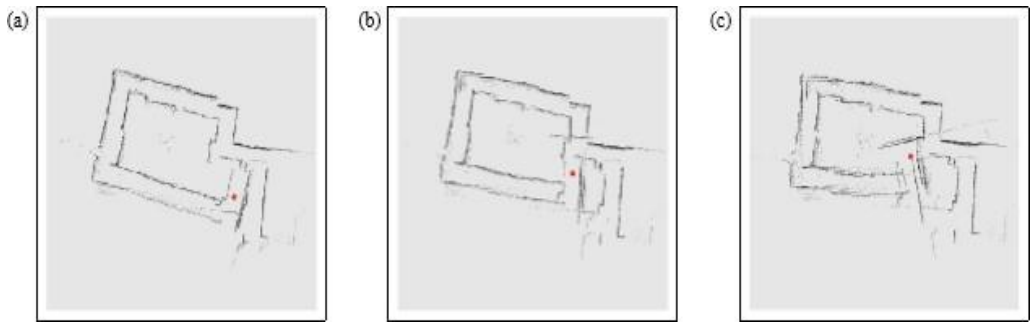


Figure 2.5: Incremental mapping of maximum likelihood, the map is developed at each stage by finding the most possible continuation, this non-probabilistic approach works well in non-cyclical conditions but is usually unable to manage cycles [2].

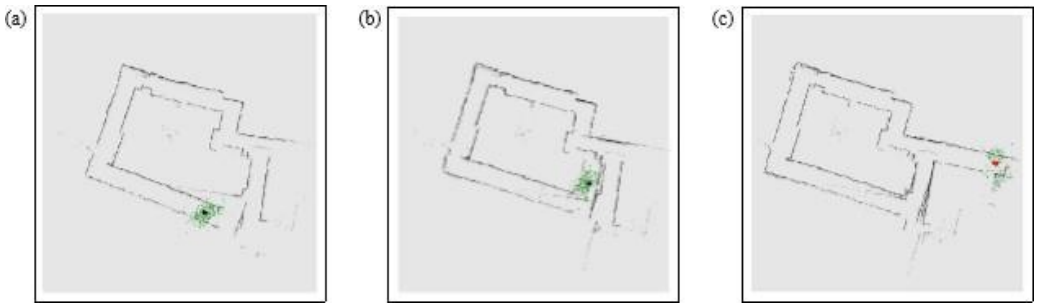


Figure 2.6: Hybrid approach which retains a later approximation over the poses of the robot, defined by a series of particles, these samples are used when closing the loop to move the robot in the map and correct the map accordingly [2].

For the same data used to produce Figure 2.5, Figure 2.6 displays a series of step for map estimation by using Hybrid. The pose posterior approximation $p(s_t, m \mid z^t, u^t)$ is implemented using particle filters, which is a sample representing variant of the Bayes filter. Figure 2.6 displays the specimens in all the diagrams. If the robot moves through a cyclic area, the samples is used to locate itself in the created map. It transmitted the error result when throughout this process in the map after the high probability of pose has been calculated. As a result, the solution still only retains a single map which has the benefit in calculation. This method is different from incremental Maximum Likelihood (ML) methods because when there is an error, it able to correct the map back. Mathematically, the differences are observed. Hybrid algorithm is simplistic approximation with EM algorithm which represent the E-step and the M-step.

However, the hybrid approach has many drawbacks. Firstly, and foremost, it can lead to catastrophic failure if it is incorrect and if want to reverse the map in time is a discrete. In addition, the complex ambiguities cannot be solved by this method. The example of it disadvantage is the uncertainty that arises while the robot passes through multiple nested cycles. Next, the hybrid approach is not real time algorithm because the time that it will takes to fix a loop is according on the loop's size. Nonetheless, when used in office-building style settings, practical implementation seems to work well in real time. The hybrid mapping algorithm was expanded to manage several robots obtaining a single map together. Figure 2.7 shows a map that was obtained by three autonomous robots, coordinating their exploration activities during the creation of the map.

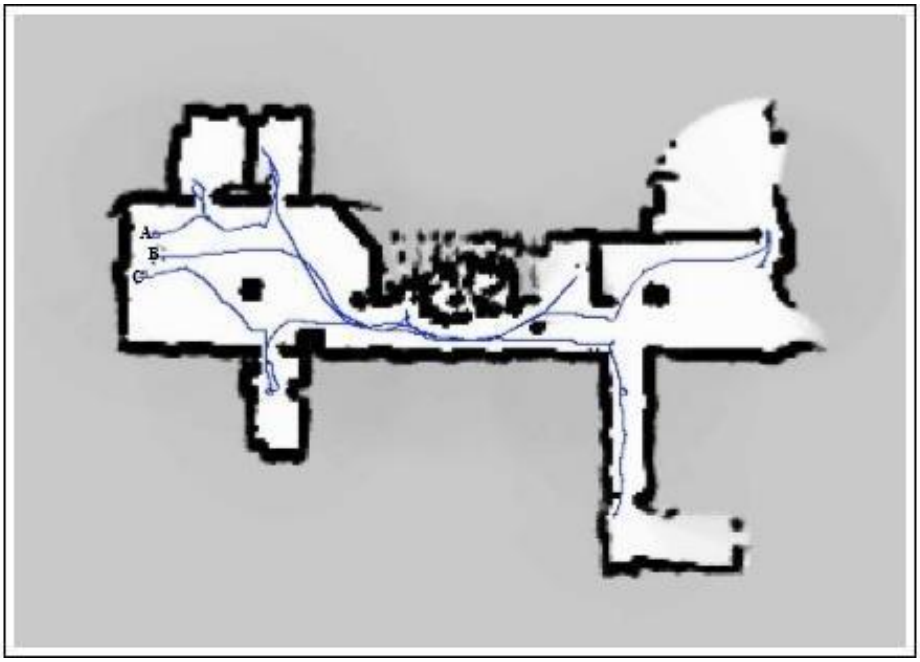


Figure 2.7: The initial robot poses, designed by three autonomous robots are on the left as indicated by the letters A, B, and C [2].

2.1.3 Occupancy Grid Map

Another of the mapping algorithms known as occupancy grid maps developed in the mid-eighties by Elfes and Moravec enjoyed considerable popularity [2]. A variety of autonomous robots use this algorithm, usually in conjunction with one of the above-mentioned algorithms. The central issue this method tackle is how the generating a reliable metric map from noisy or incomplete data's sensor. If the poses of the robot are identified because of ambiguities in the sensor data, it is sometimes difficult to say that a position is occupied in the area or not.

Occupancy grid maps are robot ecosystems' spatial representations. We describe environments that indicate the occupation of the system by fine-grained, metric grids

of variables. It also can allow the function's key that mobile robot navigation need. Example are position, path planning, avoidance of collisions and finding people [6].

An occupancy grid map used a block of cells to describe the environment. There are two conditions that has been mention in this method. First, the block cell is occupied so that the robot is unable to pass through it. Second, the block cell is unoccupied which mean that the robot can move through it [7]. Even if the world is entirely composed of cubes, maps of the occupancy grid cannot be absolutely accurate. However, it can provide the data by selecting a small enough cell volume. Grid cells status is done without comparing it to any map that uses sensors.

By generating probabilistics maps, the occupancy grid maps can solve this problems. Occupancy grid maps are defined by two- dimensional (2D) grid but also able to occupy the three-dimensional (3D). The regular occupancy grid mapping algorithm, like any other popular mapping algorithm, is a variant of Bayes filters.

The posterior over each grid cell's occupancy is measure by Bayes filters. The (x, y) is assume to be a grid cell's co-ordinates and $m_{x,y}$ is its occupancy of the grid cell. It is a binary variable. It also can occupy the cell or they are free. Therefore, the problem is to measure a posterior over a collection of binary variables, which is a single numerical likelihood $p(m_{x,y} \mid z^t, x^t)$. Bayes filters also function to measure the basis for this posteriors. Odds are often used to write the binary Bayes filter. The probability $p(x)$ odds of an event x is defined as $\frac{p(x)}{1-p(x)}$. In odds notation, the binary Bayes filter works as follows for a static map and with known poses, s^t:

$$\frac{p(m_{x,y} \mid z^t, s^t)}{1-p(m_{x,y} \mid z^t, s^t)} = \left[\frac{p(m_{x,y} \mid z_t, s_t)}{1-p(m_{x,y} \mid z_t, s_t)}\right]\left[\frac{1-p(m_{x,y})}{p(m_{x,y})}\right]\left[\frac{p(m_{x,y} \mid z^{t-1}, s^{t-1})}{1-p(m_{x,y} \mid z^{t-1}, s^{t-1})}\right] \quad (2\text{-}3)$$

The simple update formula is often logarithmically applied, which is useful in mathematical terms and prevents statistical instabilities that occur when probabilities are nearest to zero:

$$\log \frac{p(m_{x,y} \mid z^t, s^t)}{1-p(m_{x,y} \mid z^t, s^t)} = \log \left[\frac{p(m_{x,y} \mid z_t, s_t)}{1-p(m_{x,y} \mid z_t, s_t)} \right] + \log \left[\frac{1-p(m_{x,y})}{p(m_{x,y})} \right] + \log \left[\frac{p(m_{x,y} \mid z^{t-1}, s^{t-1})}{1-p(m_{x,y} \mid z^{t-1}, s^{t-1})} \right] \quad (2\text{-}4)$$

It is straightforward to see that the representation of the log odds can be recover the probability of occupancy. In addition, this method is recursive, enabling the individual grid cells to be incrementally updated when the new data sensor is arrives. Lastly, occupancy grid maps need two densities of probability, $p(m_{x,y} \mid z^t, s^t)$ and $p(m_{x,y})$.

2.2 Simultaneous Localization and Mapping (SLAM) Algorithm

The Robot Operating System (ROS) is a rapidly growing platform for the development of smart robotic applications. Its function is to support the sensors and efficient implementation for various SLAM, route planning and image processing algorithms [8]. ROS provides many SLAM algorithms such as:

- HectorSLAM
- Gmapping
- KartoSLAM
- CoreSLAM
- LagoSLAM

KartoSLAM, HectorSLAM and Gmapping establish the best algorithm to use because it has high accuracy according to the findings of [5]. All this three algorithms are in fact conceptually different. The similarity for this three algorithms is about the performance from the view's point of map accuracy. HectorSLAM is based on Extended Kalman Filter (EKF). However, Gmapping and KartoSLAM are depend on occupancy grid mapping and graph mapping based on Rao-Blackwellized Particle Filter (RBPF). HectorSLAM uses an inertial sensing platform to combine a 2D SLAM system based on rigorous scan matching and a 3D navigation technique [5].

KartoSLAM is one of a graph-based SLAM solution. Then, Robot Operating System (ROS) has been applied it by using highly optimized and non-iterative cholesky matrix decomposition. Its function as a solver for sparse linear systems. CoreSLAM is a ROS replacement for the previous 200-line-of-code [9]. A simple SLAM algorithm is designed to be clear and to understand with reduced performance loss easily.

The function of nonlinear non-convex cost function is to origin of graph-based SLAM algorithms. Specifically, in order to update the configuration of the map initial, the problem at each iteration need to be solve first. It will happen until the cost function is reached at a local minimum. Indeed, this optimization process depends heavily on an initial assumption of convergence. Carlone et al. have introduced a new method called LagoSLAM in which there is no initial guess for the optimization process.

2.3 TurtleBot

TurtleBot is a wheeled robot [10] and a regular robot on the ROS system. Turtle comes from the robot Turtle, which was motivated in 1967 by the language of

education computer programming Logo. However, the turtlesim node appears in the Robot Operating System (ROS) basic tutorial. It is a program which mimics the Logo Turtle program's command system. It can also be used as representation of ROS to build the Turtle logo. The nine dots in the logo of ROS represent the turtle's back shell. TurtleBot is derived by the Turtle of Logo which has a function to introduce that ROS can be used easily by using TurtleBot and to teach people about the computer programming language by using Logo.

2.3.1 TurtleBot3

There are three iterations of the series TurtleBot.TurtleBot1 was created by Tully and Melonee. It was created in 2010 and since 2011 has been on sale. In 2012, Yujin Robot built TurtleBot2 based on iClebo Kobuki. Then in 2017, TurtleBot3 was built with capabilities to balance its predecessor's lack of usability and user needs. ROBOTIS smart actuator (Dynamixel) is function to drive by the TurtleBot3. TurtleBot3 is a compact, inexpensive, programmable, ROS-based robot which can be used in learning, testing and product’s prototyping. TurtleBot3's can reduce the platform's size and lower the price without losing its usability and reliability while providing expandability at the same time.

SLAM, navigation and manipulation are the core technology of the TurtleBot3. This making it suitable to function as home service robots. The TurtleBot3 can run SLAM algorithms in order to create a map and to move in their environment. Besides, this robot can be remotely controlled from any gadget. Example are laptop, joypad or Android-based smart phone. When a person walk in the environment, the TurtleBot3 can also follow the legs that person.

A single board computers is used for both TurtleBot3 models (Burger and Waffle) to keep costs down in existing TurtleBot versions. Both models of this TurtleBot3 can run the new Ubuntu Linux (16.04.2 LTS) and ROS (Kinetic) versions. Both models has 360°. Hence, it allowing the TurtleBot3 to do SLAM and navigate autonomously. In addition, the single-board computer communicates with a control board in both versions, driven by an ARM Cortex-M7, linking the servos and the battery. With the Arduino software development environment, this board is a programmable and called OpenCR. The additional behaviors can be program using Arduino's C / C++ functions and libraries. ROS also can be used to command the robot. The contrast between the TurtleBot3 Burger, Waffle and Waffle Pi were listed in Table 2.3.

Table 2.3: Hardware Specifications of TurtleBot3.

Items	**Burger**	**Waffle**	**Waffle Pi**
Maximum translationa velocity	0.22 m/s	0.26 m/s	0.26 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)	1.82 rad/s (104.27 deg/s)	1.82 rad/s (104.27 deg/s)
Maximum payload	15kg	30kg	30kg
Dynamixel ports	5V / 4A	5V / 4A	5V / 4A
Programmable LEDs	GPIO 18 pins	GPIO 18 pins	GPIO 18 pins
Buttons and Switches	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
Battery	RS485 x 3, TTL x 3	RS485 x 3, TTL x 3	RS485 x 3, TTL x 3

2.3.2 Dimension and Mass

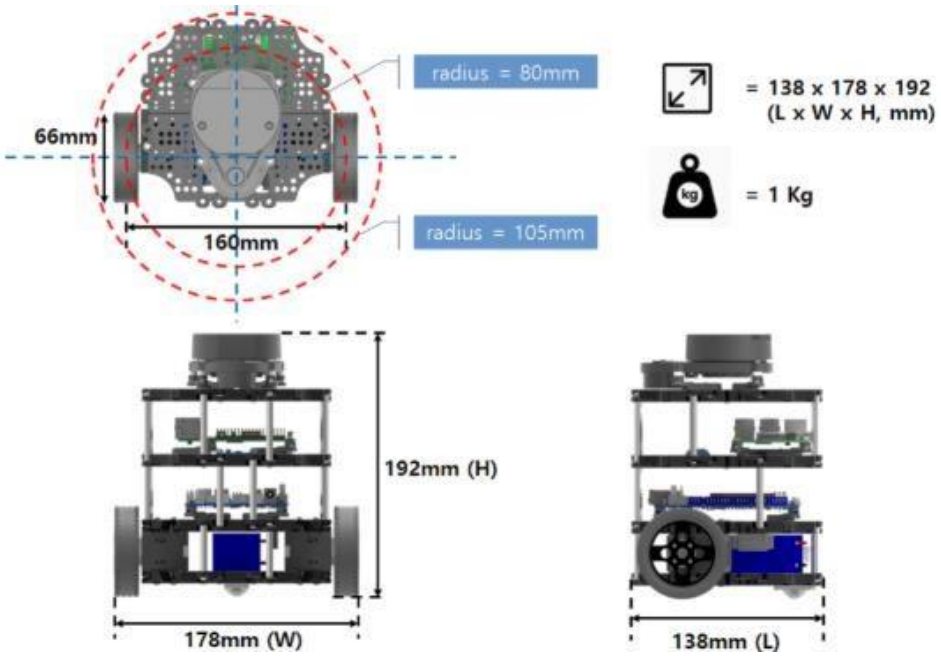


Figure 2.8: Dimension of TurtleBot3 Burger.

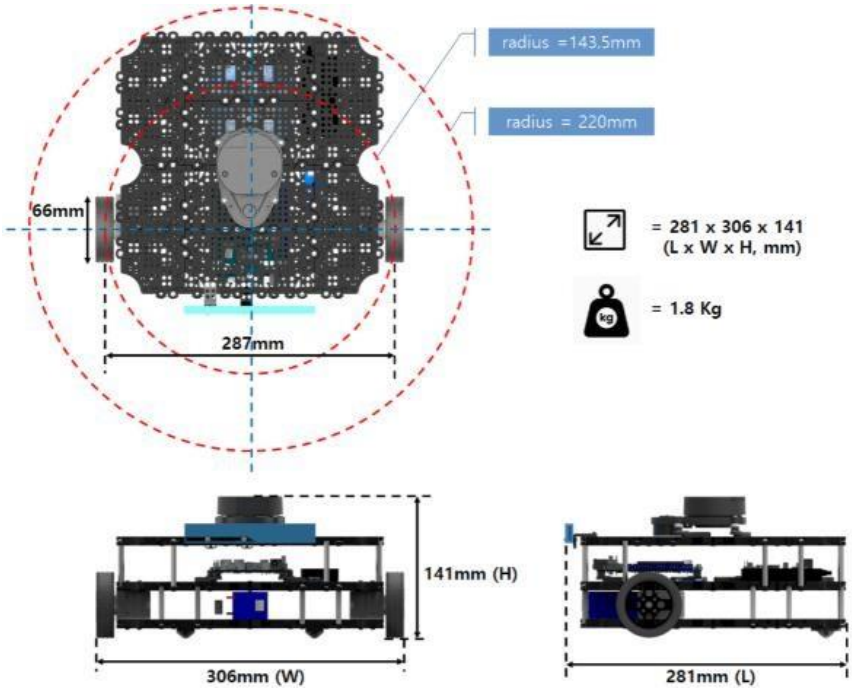


Figure 2.9: Dimension of TurtleBot3 Waffle.

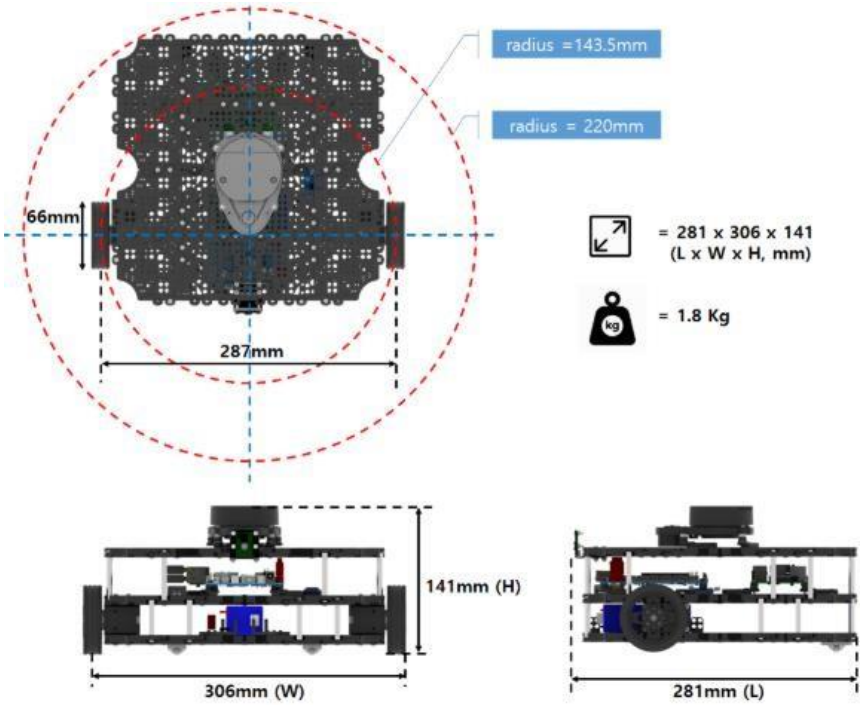


Figure 2.10: Dimension of TurtleBot3 Waffle Pi.

2.3.3 Components

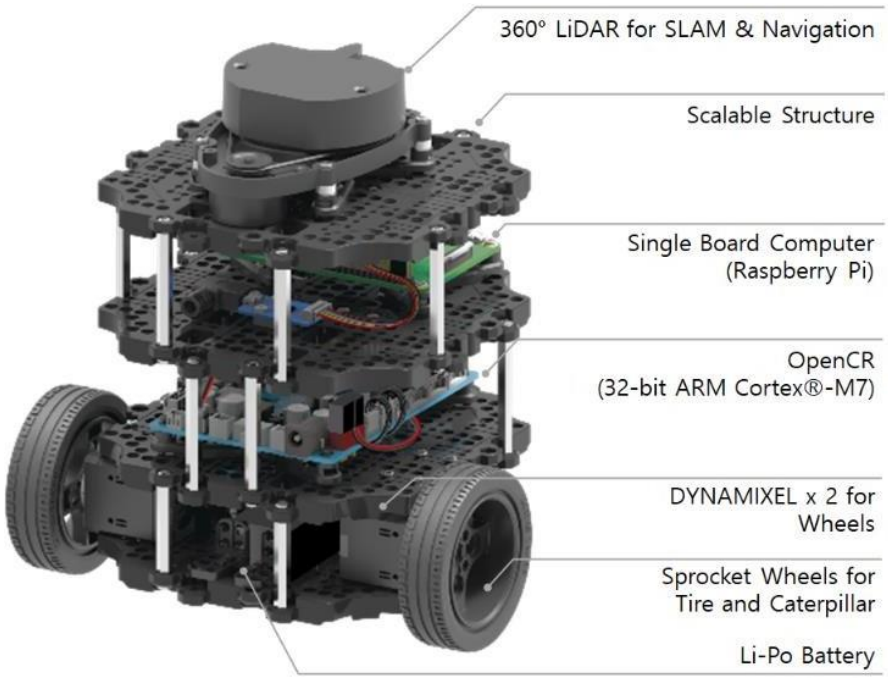


Figure 2.11: The TurtleBot3 Burger.

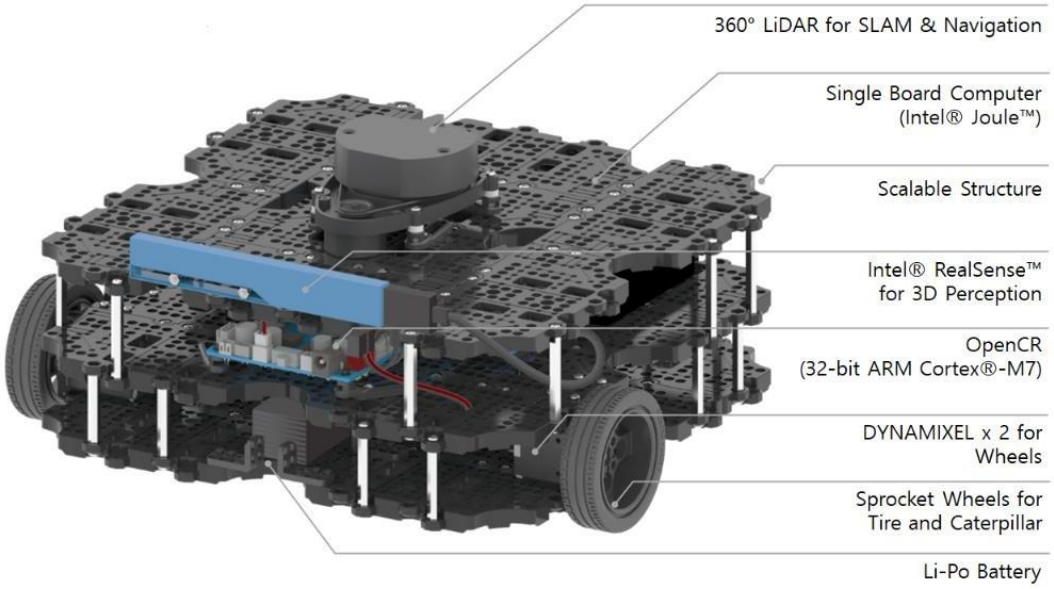


Figure 2.12: The TurtleBot3 Waffle.

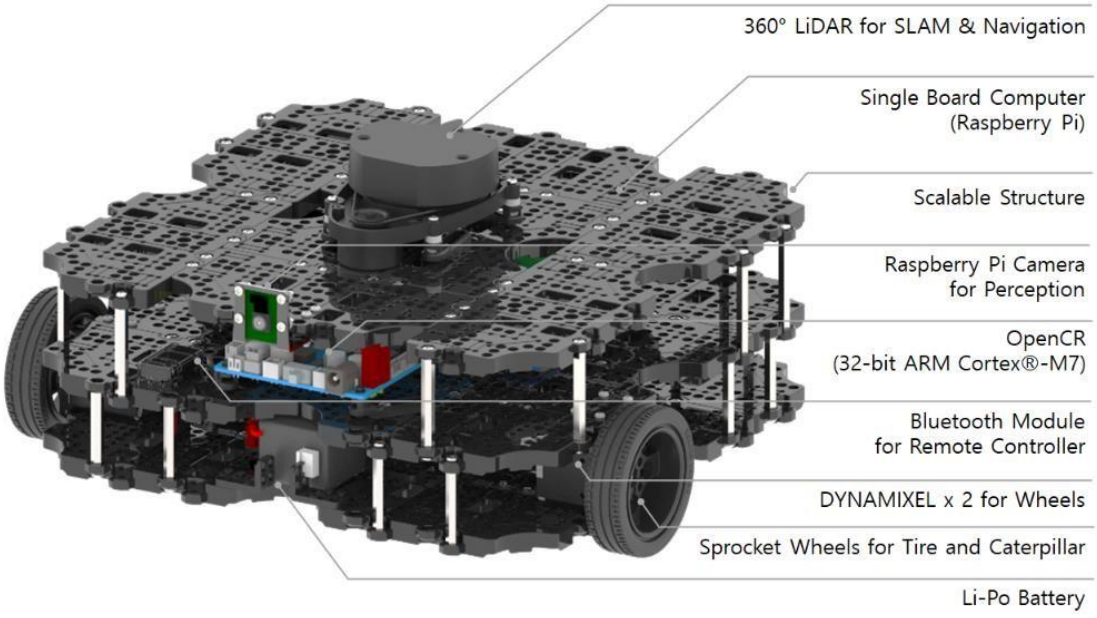


Figure 2.13: The TurtleBot3 Waffle Pi.

2.3.3.1 Single Board Computers (SBCs)

The main computer of TurtleBot3 is Raspberry Pi 3 which is for TurtleBot3 Burger and Waffle Pi, while the Intel ® JouleTM 570x is for TurtleBot3 Waffle. These SBCs are sufficient to utilize TurtleBot3's basic features, but users need to increase CPU performance, use GPU or add RAM size for other uses. In order to keep the costs down, both TurtleBot3 models (Burger and Waffle) use SBCs rather than netbooks which has been used in previous TurtleBot versions. Then, all of these TurtleBot3 run the new Ubuntu Linux (16.04.2 LTS) and ROS (Kinetic).

The TurtleBot3 Waffle is a larger and substantially better computing of an Intel ® JouleTM 570x instead of a Raspberry Pi 3 Model B, more sensing of an Intel RealSense 3D sensor and high powerful Dynamixel servos to drive the wheels and handle more payload. The SBCs for each TurtleBot3 are summarized in Table 2.4.

Table 2.4: Single Board Computers (SBCs) of the TurtleBot3.

TurtleBot3	**Single Board Computers (SBCs)**
TurtleBot3 Burger	• Raspberry Pi 3 Model B • Raspberry Pi 3 Model B+
TurtleBot3 Waffle	• Intel® Joule™ 570x
TurtleBot3 Waffle Pi	• Raspberry Pi 3 Model B • Raspberry Pi 3 Model B+ (Applied from products shipped in 2019)

2.3.3.2 Sensor

TurtleBot3 Burger uses 360 ° LIDAR, 9-Axis Inertial Measurement Unit and accurate encoder for research and development. Besides, TurtleBot3 Waffle also comes with an identical 360 ° LIDAR but also offers a powerful Intel ® RealSenseTM with the software development kit recognition. TurtleBot3 Waffle Pi uses Raspberry Pi Camera which is used widely. This will be the best hardware solution for making a mobile robot. Table 2.5 shows the sensor that has been used for each TurtleBot3.

Table 2.5: Type of sensor used of the TurtleBot3.

TurtleBot3	**Sensor**
TurtleBot3 Burger	• 360 Laser Distance Sensor LDS-01
TurtleBot3 Waffle	• 360 Laser Distance Sensor LDS-01 • Intel® RealSense™ R200
TurtleBot3 Waffle Pi	• 360 Laser Distance Sensor LDS-01 • The Raspberry Pi Camera Module v2.1

2.3.3.3 Embedded board and actuator

The SBCs interfaces with a control board, operated by an ARM Cortex-M7 that links the servos and battery are used for both TurtleBot3 (Burger and Waffle). This board was called OpenCR which has been developed by Robotis. It is a programmable with the development environment for the Arduino software. Besides using ROS, the program additional behaviors using Arduino's C/C++ functions and libraries also can

be used to control the robot. Table 2.6 summarize the embedded board and the actuator for each TurtleBot3.

Table 2.6: Embedded board and actuator of the TurtleBot3.

TurtleBot3	Embedded board	Actuator
TurtleBot3 Burger	OpenCR1.0	Dynamixel XL430
TurtleBot3 Waffle	OpenCR1.0	Dynamixel XL430
TurtleBot3 Waffle Pi	OpenCR1.0	Dynamixel XL430

2.4 Sensor Implementation for Grid Map

2.4.1 Range Sensor

Autonomous mobile robots are not a new development and they have introduced several types of scope sensors. Beginning from the most basic level, inexpensive ultrasonic and infrared distance sensors can be used. Mobile robots can figure out the distance between themselves and the obstacle in front of one dimension.

Figure 2.14: Ultrasonic and Infra-red Range Sensor.

These sensors are quite simple and straightforward to construct, but in some respect they have poor performance. The infra-red range sensor has weak sensing area whereas the ultrasonic range sensor has wide sensing but limited resolution of the room. Both are appropriate for educational purposes for beginners and are not suitable for advance autonomous mobile robot in precise space measurement.

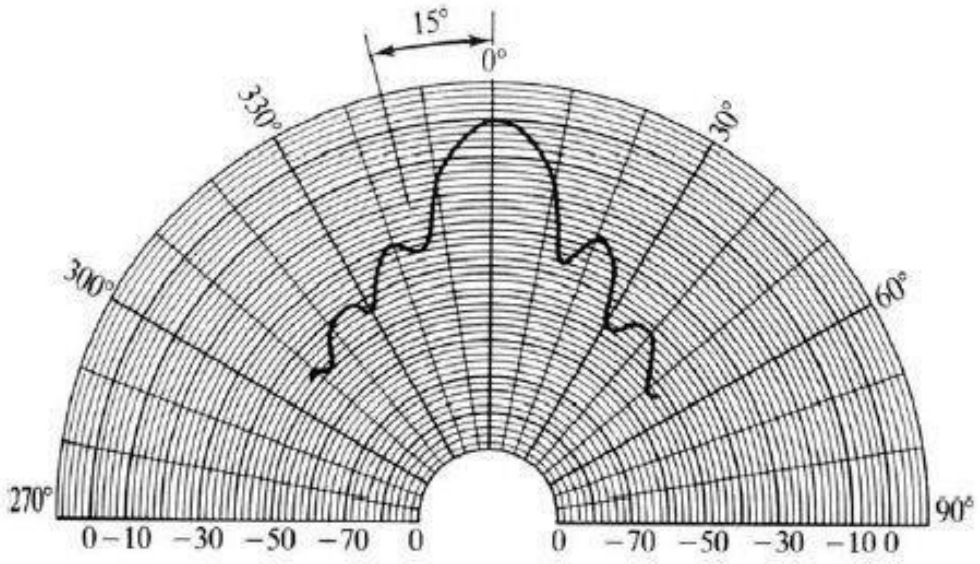


Figure 2.15: Ultrasonic Sensor's Sensing Ranges.

2.4.2 Laser Range Finder

Laser range finders and flight time cameras are probably the highest distance measurement accuracy sensors [11]. Typically, they are used in autonomous mobile robots or vehicles that need accurate measurement of the distance to facilitate algorithms such as position and mapping. Under certain conditions, stereo vision performance may be as good as laser range finders, but the computational power required to execute stereo algorithms is significant [12]. In other meanings, the data of the laser range finder does not require another step of matching process and

therefore, besides its accuracy, its sensing speed is another advantage. Certain range sensors are typically some order of magnitude quite costly than other range sensors due to their high performance.

Laser range finder was widely used in the field of robotic growth. Due to its speed and accuracy of data, holds the crown among other navigator devices. This is a popular way for obtaining 3D data and creating local maps. To create a 3D map, a 2D laser range finder was deployed on an autonomous robot [13].Their solution is to rotate with horizontal axis of rotation the 2D laser range finder. Thus, a 3D range finder with a servo motor is built based on a 2D range finder. The rotation offers vertical scanning and repels the need to upgrade the sensor. The accuracy of the map produced is therefore largely dependent on the resolution of the servo motor.

Figure 2.16: Rotational Laser Range Finder.

2.4.3 Light Detection and Ranging (LIDAR)

Light Detection and Ranging (LIDAR) is a sensor that utilize Laser Range Finder (LRF) [14]. This sensor have a characteristic reliable and has been used for many

researches which is related to robotics. LRF sensor is essential to supporting process reading environment, as an eye to robot. For example, LIDAR has been apply in household appliances such as vacuum to move automatically and can clear the house without human control.

To measure the distance from sensor to target, LIDAR will operates with emitting laser beam. After that, by using a delay, LIDAR will calculates the distance between laser emitting and laser bouncing back to sensor. LIDAR operating frequency is higher compared to Radio Detection and Ranging (RADAR). This is because LIDAR has hundreds of Tera Hertz (light pulse) while RADAR only has Giga Hertz (electronic waves) [15].

Data generated by LIDAR are usually stored in binary form, consisting of 2D/3D coordinate and laser intensity. Due to the LIDAR data burst, the file size that was produced has an impact. Ying research has found that 3D mapping area in rural locations as large as 2.79 square kilometers generates 7 million 3D point [16], which is caused by LIDAR data being highly precision. The number of LIDAR data produced can increase the map forest area with more complex object.

CHAPTER 3

METHODOLOGY

This part's aim is to evaluate the project's techniques. This chapter clarified the approach used in this research involving the cover system throughout the entire study context. The description in this chapter is the design flowchart structure from the start to the end of the project.

3.1 Introduction

There are four stages in the technique or procedure. The first stage is the preliminary study. In this step, all the research about the problems and methods which related to this project was studied. Then, the second stage is the installation of the software. For this project, ROS was installed in Ubuntu operating system because all

the commands will be run in this platform. The next stage is the development of the environment. The simulation environment for the robot was created in the Gazebo. The last stage is the examination of the algorithm performance and the analysis of the data. The accuracy of the map by using Gmapping technique was measured in order to know which speed has a better performance and suitable to be used for the mobile robot. Figure 3.1 shows the flowchart for this project.

3.2 Flowchart

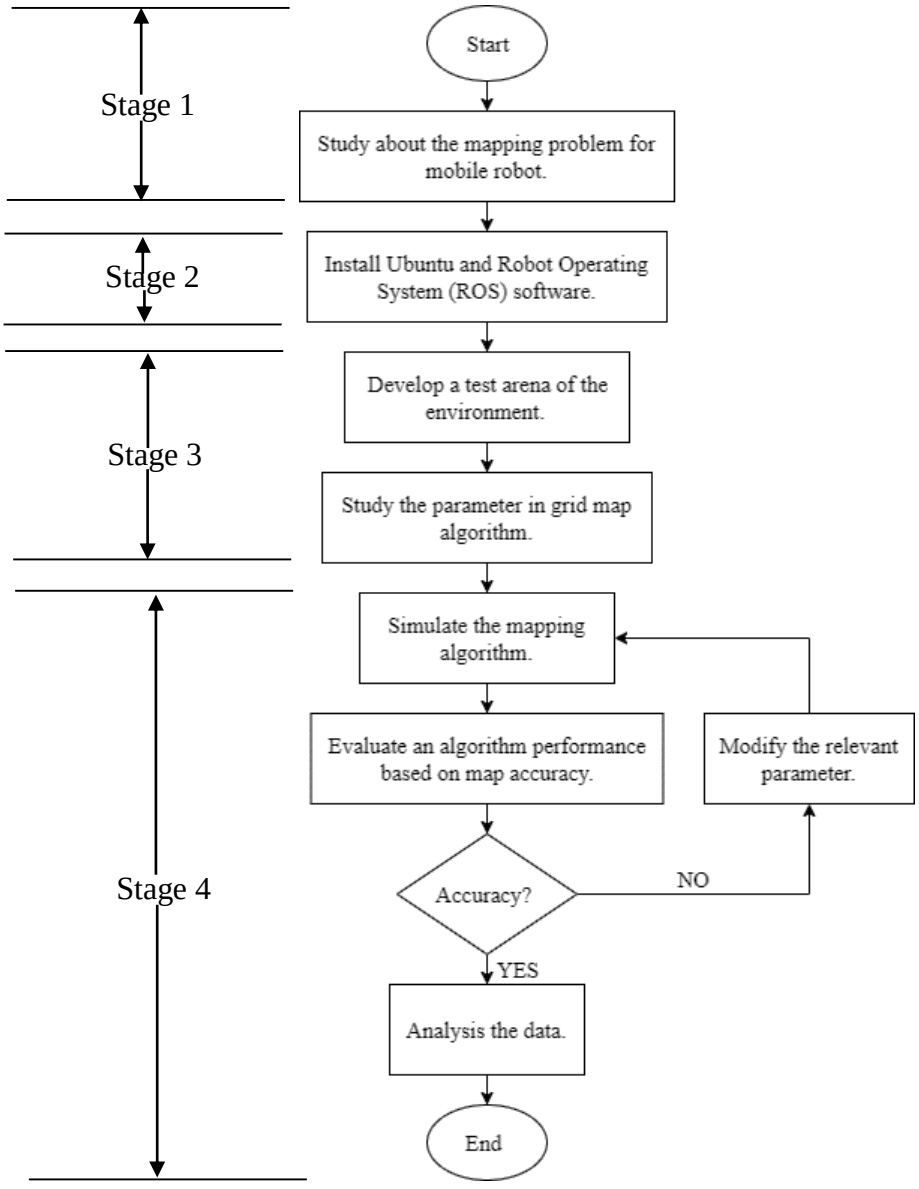


Figure 3.1: Flowchart.

This project is to create the simulation of the mapping algorithm for mobile robot by using Ubuntu and Robot Operating System (ROS) software. First of all, to start this project, the details about the mapping problems that the mobile robot faced nowadays need to study clearly.In order to understand about this, the researches from others paper and journal which are related to this topic was done.

After doing a research, the next step which is installing the Ubuntu and Robot Operating System (ROS) sofware in the laptop. This takes about one week to finish because of the internet limitation and the performance of the laptop. The laptop's RAM was upgraded to support this two software. Ubuntu is one of the open source operating system (OS) based on the Linux distribution. In Ubuntu, ROS is used because of it is a flexible framework for writing a robot system.

Then, after the software was succesfully installed, the map environment in the simulation was developed. In this case, Gazebo is used in ROS to create the mapping environment. Besides, Gmapping SLAM algorithm was applied in Rviz simulation. This is because, Gmapping algorithm is the best method to use for the calculate of accurancy map. In this step, the performances of the mobile robot based on map accuracy ware measured. Two different speed which are $0.5ms^{-1}$ and $1.0ms^{-1}$ was used for this mission. For each speed, the performances of the mobile robot at five different times was recorded. Then, if the accuracy of the map is accepted, so the data analysis will be proceeded. If not, the relevent parameter need to be modified until its success.

For data analysis, the details of the calculation as in Chapter 4 has been done. The comparision between speed $0.5ms^{-1}$ and $1.0ms^{-1}$ was shown clearly. Not only that, the ratio of free space and ocupied space also been calculated. Besides, the fitness score also be measured in order to know the accuracy of the map.

3.3 Occupancy Grid Map Algorithm

By producing a probabilistic map, the problem can be solved by residential grid maps. Occupancy grid maps are characterized by 2D grids, but some of them have 3D space. Just like any other main mapping algorithm, a Bayesian filter version is the regular occupancy grid mapping algorithm. The posterior above the occupancy of each grid cell can be calculated by using Bayes filter.

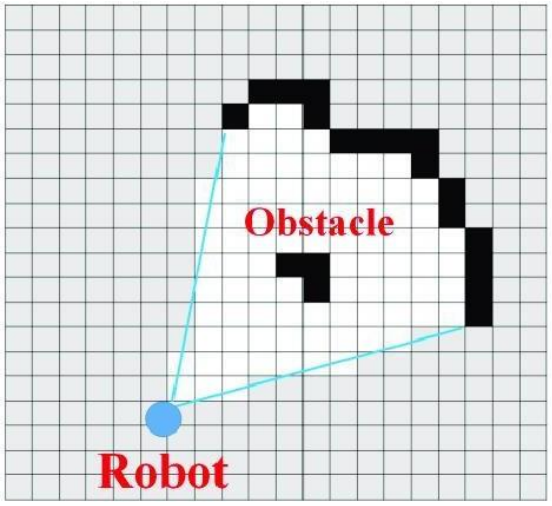


Figure 3.2: Grid cell.

3.3.1 Gmapping

Gmapping is a SLAM algorithm based on a range sensor. It always been used as SLAM program in robotics fields. This algorithm is a Rao-Blackwellized Particle Filter SLAM solution proposed by Grisetti et al. The algorithm Particle Filter (PF) family usually need a large number of particles to achieve good performance. This can increases the difficulty of its computation. The depletion problem associated with the process of PF resampling also reduces the accuracy of the algorithm. This happen due

to the significance of particle weights can become insignificant. The algorithm requires a limited particles' number to describe the posterior SLAM. It also can reduces the effort to resample and create a precise maps successfully. The Gmapping provides the easier way to optimize the mapping process by adjusting certain mapping parameters. The example of mapping parameter are the particle's number used by Rao-Blackwellized Particle Filter (RBPF), the displacement stage for processing new scanning and the resampling threshold to match the application specific needs.

3.4 Experiment Platform

3.4.1 Ubuntu Operating System

Ubuntu is a free and open-source operating system developed by Canonical Ltd. based on the Linux distribution and Debian operating system that is similar to Uniplexed Information and Computer Systems (UNIX) [17]. It is available on three versions which are the desktop, server, and core format. It runs on personal computers, server or cloud computing platforms, and Internet of Technologies (IoTs).

Ubuntu is one of the Linux operating systems for great and complete desktops. It has community and professional support and is freely available. The Ubuntu community is focused on the principles expressed in the Ubuntu Manifesto that the software must be freely available, that the software must be open to users in their local language even if disabled and that individuals are free to adjust their software and change it in whatever way they think fit. "Ubuntu" is inspired by a traditional African term, meaning "humanity to others" Ubuntu distribution carries Ubuntu's spirit to the world of technology.

The Ubuntu's benefits is its being a free-to-download and open-source operating system. Unlike Microsoft Windows and Apple's macOS, the people and organizations can own and keep personal devices without the need to pay for software licenses or buy exclusive devices. Another strength of Ubuntu is that it can compete fairly against Windows and macOS, especially in offering a complete desktop computing experience for users. For instance, the Desktop version comes with applications for office productivity from LibreOffice. Additionally, Ubuntu can be installed in multiple devices such as Windows and Mac computers. It operates via a virtual machine or containers on network servers, IoTs devices and robots, and in emulated or virtualized computer environments.

Ubuntu Operating System is a software system which has been operating for a number of years and is capable of performing well to minimize and increase the rate of failure caused by constantly increasing error conditions that can cause the system to fail. This is the software aging phenomenon. Critical security programs are stated to be age-related failures as the aging assets of the technology can be concealed in multiple layers of complex software systems, from the Operating System (OS) to the user device level. Ubuntu in the latest version is more than two million lines of code and is not planned and managed as much as most industrial computer processes due to its development model.

3.4.2 Robot Operating System (ROS)

Robotics Operating System or ROS, is an open-source robot software writing framework [18]. ROS is an operating system identical to Windows or Linux but is an open source operating system for robots that includes of a library and a platform for

controlling robots, including drivers of different devices and algorithms. ROS used to build complex robot systems on a large scale. The SLAM approach is used by the ROS survey robot to guide the robot. Also, ROS can allow researchers to do work more quickly. Research teams can also use ROS to conduct simulations and experiments in the real world. However, ROS is known for its modular and decentralized architecture.

ROS supports C++, Python, Octave, and List Processing (LISP) languages [18]. Ubuntu Linux is better used because it includes a range of open- source software. Requirement of vast amounts of open source software.

The ROS framework has several aims. The first is thin. ROS is built to be as lightweight as possible, such that ROS-written code can be combined with other implementations of robot software. A benefit of this is that ROS can be conveniently combined with other robot program frameworks. OpenRAVE, Orocos, and Player have already incorporated with ROS. Next, write ROS-agnostic libraries with clean functional interfaces is the reason of preferred development model. In modern programming language, the ROS framework is easy to accomplish. It is already implemented in Python, C++, Lisp and for the experimental library are Java and Lua. In addition, the ROS is easy to test. This is because ROS has a builtin unit or integration test framework named rostest which makes it simpler for test fixtures to be brought up and ripped down. For scaling, ROS is suitable for large runtime systems and for huge development processes.

3.4.3 Gazebo

Gazebo is a simulator that can be used to model a robot. In a three-dimensional world, Gazebo effectively simulate and imagine robotic acts [10]. Robot simulation in

indoor and outdoor settings using various types of robot models is necessary in Gazebo. Gazebo allows robotic and sensor systems to be replicated in 3D indoor and outdoor environments. It has a server or client design and has an interprocess communication model based on the subject Publish or Subscribe.

Besides native interface, Gazebo also has a regular player interface [19]. Gazebo clients can use a shared memory to access their data. That Gazebo simulation object able to connect with more than one controllers that process the object control commands and generate the object's state. The controller generated data is released in the shared memory using Gazebo interfaces (Ifaces). A simple simulator environment has been developed in Gazebo. This simulator tests the proposed algorithm.

3.4.4 TurtleBot3

Turtlebot3 is a redesign of the original Turtlebot, an open-ended hardware built by Willow Garage in 2010 and operating on a motorized wheelbase [18]. Figure 3.3 shows the list of components for TurtleBot3 Burger.

Figure 3.3: Turtlebot3 (Burger).

This robot has many features, such as low-cost [20], personal robot package with open-source software, and it is lightweight, portable and customizable. The TurtleBot3 package is composed of a mobile platform, a 2D/3D distance sensor (Microsoft's Kinect), a Single Board Computer (Raspberry Pi), microcontrollers (Arduino UNO), sensors (ultrasonic and accelerometer gyroscope) and components to allow to move. TurtleBot3 is simple to purchase, build and install from regular materials, utilizing shelf market goods and parts that can be quickly made. It also utilized the most common ROS platform in the world.

3.4.5 360 Laser Distance Sensor LDS-01 (LIDAR)

LIDAR is a remote optical sensing technology that measures distance and angle between the sensor and the target [21]. The LIDAR sensors provide autonomous ground robots with navigation and localization. The distance from the sensor to the target is determined by calculating the time period between the transmitted laser pulse and the reflected pulse receipt [15]. It system allows the creation of high-resolution maps for various applications. One major application is SLAM, in which the purpose is to create a map of an unknown environment while at the same time keeping track of the location of the robot. Figure 3.4 shows the LDS-01 Laser Distance Sensor.

Figure 3.4: 360 Laser Distance Sensor LDS-01 (LIDAR).

The LDS-01 is a 2D laser scanner sensors that measure 360 degrees that collects data around the robot for Simultaneous Localization and Mapping (SLAM) applications. It support the USB interface and can be easily installed on a PC. The specifications of LDS-01 Requirements are listed in Table 3.1.

Table 3.1: Specifications of LDS-01.

Items	Specifications
Operating supply voltage	5V DC ±5%
Light source	Semiconductor Laser Diode (λ=785nm)
Distance Range	120 ~ 3,500mm
Angular Range	360°
Angular Resolution	1°
Current consumption	400mA or less (Rush current 1A)
Interface	3.3V USART (230,400 bps) 42bytes per 6 degrees, Full Duplex option
Ambient Light Resistance	10,000 lux or less
Sampling Rate	1.8kHz
Dimensions	69.5(W) X 95.5(D) X 39.5(H)mm
Mass	Under 125g

3.5 Experiment Setup

The code was tested on a 2.4GHz i5-core laptop with Ubuntu and Robot Operating System (ROS) Kinetic edition running 10GB Random Access Memory (RAM). The simulation environment has been referred to as the Turtlebot3 World which available in Gazebo models. The robot is running the SLAM Gmapping approaches. The detector used was the Lds-01 (LIDAR) laser distance sensor with 360° and 30m range.

To evaluate Gmapping, the simulation was run on the test environment. Then, the robot need to discover and captured the map environment in that simulation to determine either the performance is accurate or not.

3.5.1 Simulation Command

3.5.1.1 Ros Visualization (Rviz)

Ros Visualization (Rviz) is a 3D visualization tool for Robot Operating System (ROS). Rviz will allows the user to simulate robot model, log sensor information from the robot's sensors and to replay the logged sensor information. If an actual robot is communicating with a workstation that is running Rviz, Rviz will display the robot's current configuration on the virtual robot model.

3.5.1.2 Installing TurtleBot3 package

The packages installed to run the simulation are shown in Figure 3.5. The sudo command lets programs run with another's security privileges. It requests an individual password and approves a request to perform a command by testing a script, named sudoers that is set up by the system administrator. By using the sudoers folder, system administrators may provide access to any or all commands to other users without needing to remember the root password for certain accounts. It often records all commands and statements such that a database of who used it for what and when is accessible.

```
sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-twist-joy roskinetic-teleop-twist-keyboard ros-kinetic-laser-proc ros-kinetic-rgbdlaunch ros-kinetic-depthimage-to-laserscan ros-kinetic-rosserialarduino ros-kinetic-rosserial-python ros-kinetic-rosserial-server roskinetic-rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-urdf roskinetic-xacro ros-kinetic-compressed-image-transport ros-kinetic-rqtimage-view ros-kinetic-gmapping ros-kinetic-navigation ros-kineticinteractive-markers
```

Figure 3.5: ROS released packages command.

A source package collects the program code and fixes the way they were at the time of creation. Source packages are highly helpful to programs and libraries for debugging problems and regenerating binary packages. It also functions to change current programs and incorporate extra monitoring and to check that a particular security patch has been added to the source or not. A catkin workspace (catkin ws) is a file in which current catkin packages are generated or updated. The catkin arrangement makes the cycle of constructing and installing your ROS packages easier. Figure 3.6 shows the source packages command.

```
cd ~/catkin_ws/src/
git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
git clone https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
cd ~/catkin_ws && catkin_make
```

Figure 3.6: Source packages command.

A new terminal window needs to be opened to start Gazebo for the first time. Then, the command as in Figure 3.7 was typed. Launching for the first time probably takes long time. By running this command, the Gazebo will be open and ready to use. In Gazebo, the environment map will be created.

```
~/catkin_ws/src/turtlebot3_simulations/turtlebot3_gazebo/launch
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

Figure 3.7: Command for launch Gazebo.

Figure 3.8 until Figure 3.13 shows the commands that were used to run the simulation, to control the mobile robot and to save the environment map. Each of the command was run in different terminal in ROS. For movement, the mobile robot able to move automatically or it can be controlled manually by using the keyboard. To adjust the speed of the robot, it also needs to press the keyboard. By running these commands, the simulation was successful done because the mobile robot able to build its own map by using Gmapping algorithm.

```
roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

Figure 3.8: Command for TurtleBot3 movement.

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```

Figure 3.9: Command to execute Rviz.

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Figure 3.10: Command for launch Gmapping.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Figure 3.11: Command for control TurtleBot3.

```
sudo apt-get install ros-kinetic-teleop-twist-keyboard
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

Figure 3.12: Command for control the speed of TurtleBot3.

```
rosrun map_server map_saver -f ~/map
```

Figure 3.13: Command for control the speed of TurtleBot3.

3.5.2 Test Environment

To test the performance of the robot in narrow and flat area, the simulated environment was simulated. The simulation environment used in Robot Operating System (ROS) is TurtleBot3 World. The simulated environment's area is $25m^2$. Figure 3.14 shows the ROS mapping environment.

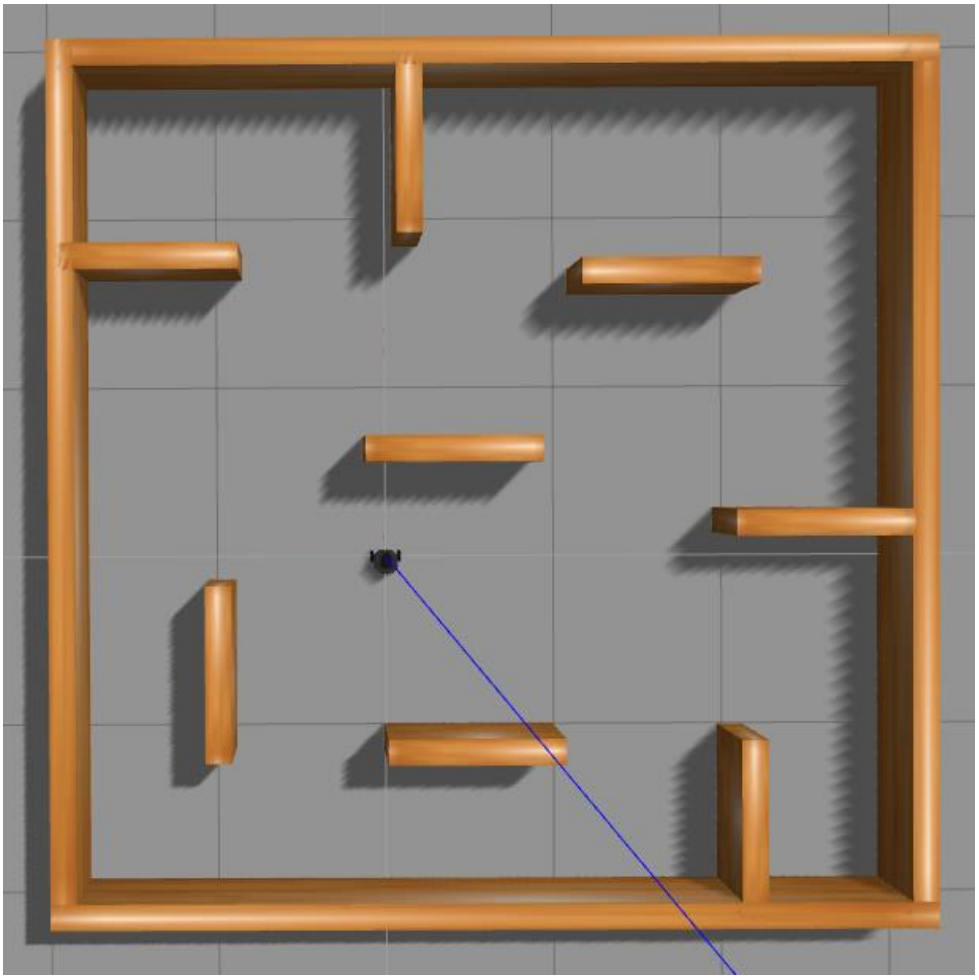


Figure 3.14: Simulated environment in Gazebo platform.

3.5.3 Robot Operation

The robot's exploration operation consists of two parts. First part is the computer keyboard system control process that controls the robot to build map through the user. The laser detector scans the objects or obstacles that surround the robot in the simulation. Then, the Ros Visualization (Rviz) software in the Robot Operating System (ROS) was used to build a map, also known as the SLAM which controls the keyboard of the computer.

3.5.4 Evaluation Metrics

3.5.4.1 Occupancy binary random variable

In occupancy binary random variable, there are two conditions which are first when the grid cell is in a free space and second is when the grid cell is in an occupied space. Free space means there are no obstacle in the grid cell and the robot able to pass through it while occupied space means there are an obstacle in the grid cell and the robot unable to pass through it [7]. The value for free space is one and the value for occupied space is zero. The equation for an occupancy binary random variable is:

$$m_{x,y} : (\text{free, occupied}) \rightarrow (1, 0) \tag{3-1}$$

3.5.4.2 Occupancy grid map

In order to calculate the occupancy grid map, the Bayes' Rule equation was used. First, the odd equation needs to be derived. Then, the log for the odd equation was calculated by using the previous equation. The equations and derivations are shown below.

$$p\left(m_{x,y}\middle|z\right) = \frac{p(z|m_{x,y})p(m_{x,y})}{p(z)} \tag{3-2}$$

$$\begin{aligned} \text{Odd}\left[\left(m_{x,y} = 1\right)\text{given } z\right] &= \frac{p(x)}{p(x')} \\ &= \left[\frac{p\left(m_{x,y} = 1\middle|z\right)}{p\left(m_{x,y} = 0\middle|z\right)}\right] \end{aligned}$$

$$= \left[\frac{\frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z)}}{\frac{p(z|m_{x,y}=0)p(m_{x,y}=0)}{p(z)}}\right]$$

$$= \left[\frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)}\right] \quad (3\text{-}3)$$

$$\log \text{odd} = \log\left[\frac{p(m_{x,y}=1|z)}{p(m_{x,y}=0|z)}\right]$$

$$= \log\left[\frac{p(z|m_{x,y}=1)p(m_{x,y}=1)}{p(z|m_{x,y}=0)p(m_{x,y}=0)}\right]$$

$$= \log\left[\frac{p(z|m_{x,y}=1)}{p(z|m_{x,y}=0)}\right] + \log\left[\frac{p(m_{x,y}=1)}{p(m_{x,y}=0)}\right] \quad (3\text{-}4)$$

There are two conditions in this case. First when z is zero, the equation is:

$$\log \text{odd} \downarrow \text{occ} = \log\left[\frac{p(z=0|m_{x,y}=0)}{p(z=0|m_{x,y}=1)}\right] \quad (3\text{-}5)$$

Second when z is one, the equation is:

$$\log \text{odd} \downarrow \text{free} = \log\left[\frac{p(z=1|m_{x,y}=1)}{p(z=1|m_{x,y}=0)}\right] \quad (3\text{-}6)$$

For constant measurement model, the values for free space and occupied space are:

$$\log \text{odd} \downarrow \text{free} = 0.9 \quad (3\text{-}7)$$

$$\log \text{odd} \downarrow \text{occ} = 0.7 \quad (3\text{-}8)$$

At initial map, the log odd is always zero as state in equation below.

$$\log \text{odd} = 0 \text{ for all (x,y)} \tag{3-9}$$

Then, for condition one is equal to zero, the value is 0.5 as shown in equation below.

$$p(m_{x,y} = 1) = p(m_{x,y} = 0) = 0.5 \tag{3-10}$$

3.5.4.3 Fitness score

Fitness score focuses on comparing the performance that integrates with the occupancy grid map algorithm [4]. The value of the cell at position (x, y) in test map of the simulation environment are compared with the value of the cell at position (x,y) in reference map of the simulation environment. The sum of difference is divided by the number of cell used in test map, m. The most accurate map will obtain a score of 1 while a less accurate map will result in a score of 0. The fitness score is determined by the following equation:

$$f(m, n) =)1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N} \tag{3-11}$$

Where:

- $m_{x,y}$ = value of the cell at position (x,y) in test map.
- $n_{x,y}$ = value of the cell at position (x,y) in reference map.
- $N = \text{number of cell used in test map, m.}$

3.5.4.4 Occupied / free cells ratio

The occupied or free cell ratio is when the map was examined with the occupied and free cells in the ground truth map respectively using a very intuitive method in the generated graph. The nearest value of free cells ratio to 1, the better performances will obtain. The equations to find the ratios are:

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y,\text{free,true}})}{\sum(n_{x,y,\text{free}})} \quad (3\text{-}12)$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y,\text{occ,true}})}{\sum(n_{x,y,\text{occ}})} \quad (3\text{-}13)$$

3.5.5 Run Gmapping Algorithms

To measure the accuracy of map performance, the simulation environment map needs to be built. Figure 3.15 shows the environment map created in the simulation by using Gmapping algorithm. From this map, it is shown that the total numbers of grid cell that has been used in this simulation are 25. In this simulation, only a single robot and static indoor environment are involved. The green line in the figure below shows the laser range finder from the TurtleBot3 Burger, which function to detect the obstacle while create its own map. Figure 3.16 shows the save mapping environment by using the command.

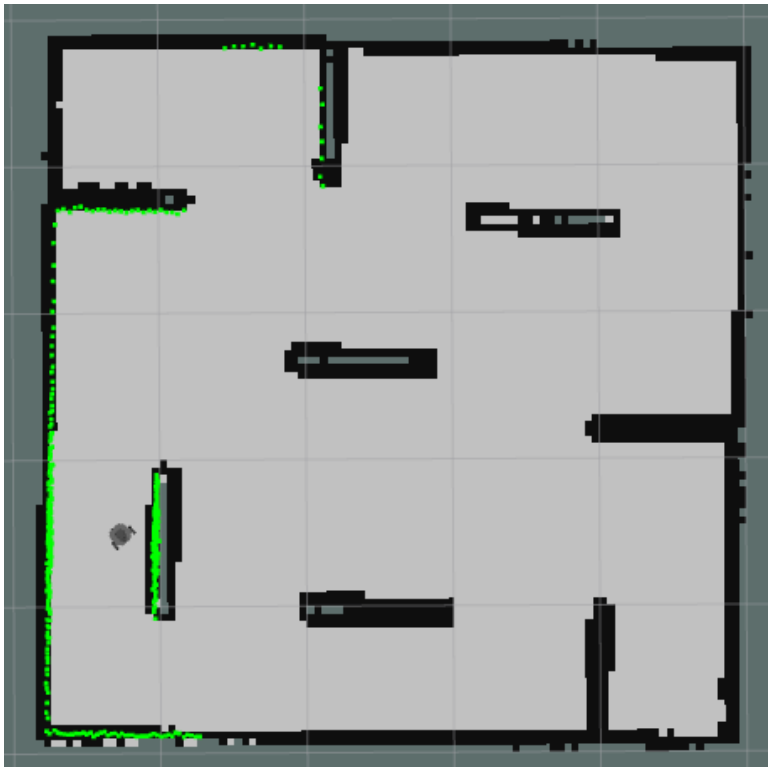


Figure 3.15: Mapping environment from the TurtleBot3.

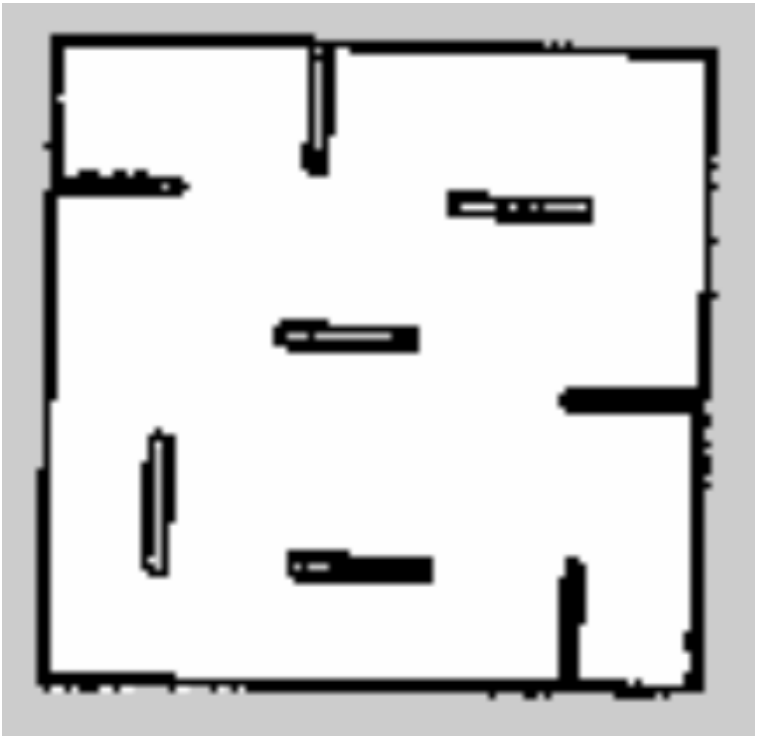


Figure 3.16: The save mapping environment using command.

CHAPTER 4

RESULTS AND DISCUSSION

This chapter discusses the collected details, the outcomes of the statistical research carried out, and the explanation of the findings for the research problem regarding map accuracy by using a Gmapping SLAM algorithm. All the results are presented in tables and graphs.

4.1 Area for Free Grid Cell

The simulation environment was created in ROS by using Gmapping SLAM algorithm. The total numbers of grid cell that has been used in this simulation are 25. For each grid cell, the area is $1m^2$. Then, to calculate the total area of this environment map, the area for each grid cell ($1m^2$) need to be multiplied with the total numbers of

grid cell that has been used the simulation (25). So the value of the area for this environment map is $25m^2$. Figure 4.1 shows the environment map from the simulation.

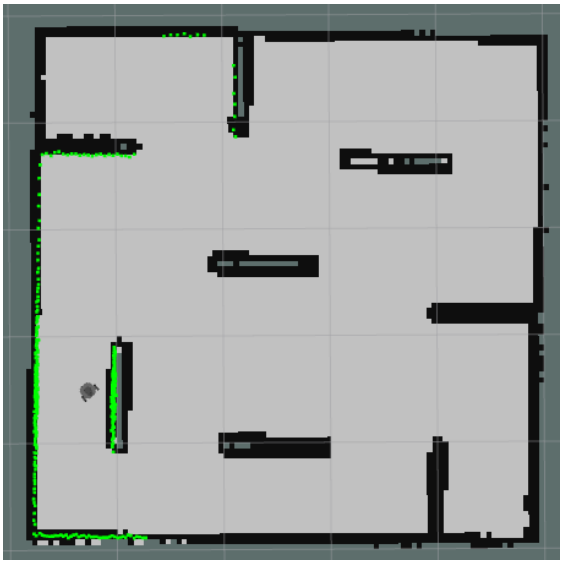


Figure 4.1: Environment map from the simulation.

Next, to calculate the area for each grid cell of mapped environment, the grid cell was divided into 100 small grid cells as shown in Figure 4.2. The area of each small grid cell is $0.01m^2$. This is because as stated before, the area for each grid cell is $1m^2$, so to know the area of this small grid cell, it need to be divided into 100.

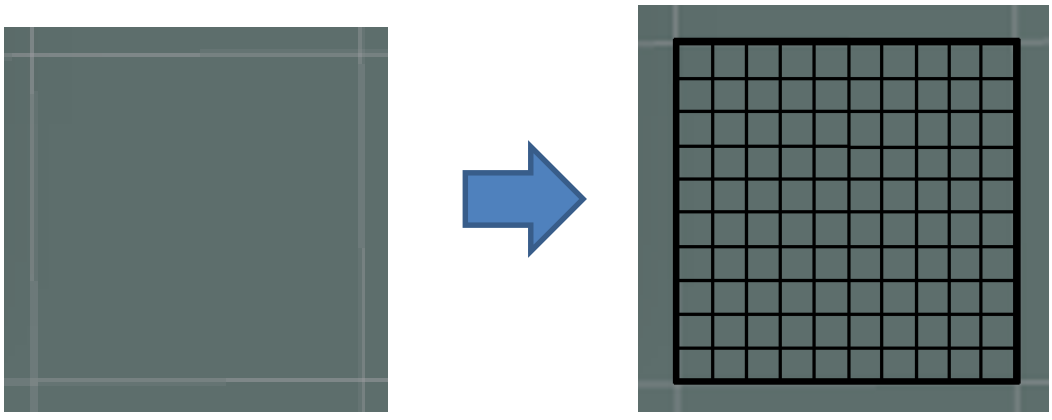


Figure 4.2: Small grid cell.

For example, one of the grid cell was taken from the environment map to calculate its area. First, this grid cell was divided into 100 small grid cells. The total number of free small grid cell was calculated and the value is 70. Then, to calculate the area of the free small grid cell, the total number of free small grid cell (70) was multiplied with the area of each small grid cell ($0.01m^2$) and the result is $0.7m^2$. Figure 4.3 shows the free small grid cell for part 1.

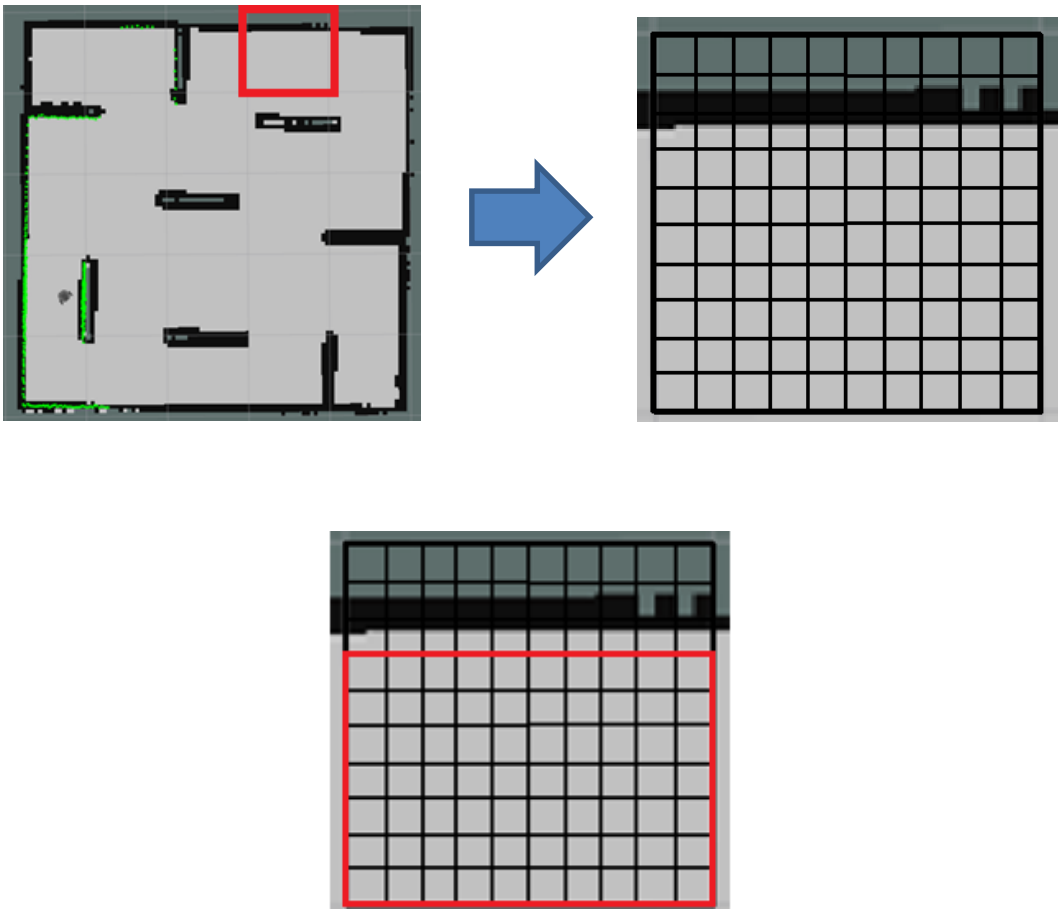


Figure 4.3: Free small grid cell for part 1.

Next, the area for the balance free small grid cell was calculated. In this step, only nine small grid cell was involved. The area of each small grid cell is $0.01m^2$. First, the small grid cell was divided into 25. Then, to calculate the area of each free small grid

cell, the value of $0.01m^2$ has been multiplied with 25, so the answer is $0.0004m^2$. As shown in Figure 4.4, the numbers of free small grid cell are 20. So, to know the total numbers of free small grid cell, 20 need to multiply with 9. The result is 180. Lastly, to calculate the area of the free small grid cell, the total numbers of free small grid cell (180) were multiplied with the area of each free small grid cell ($0.0004m^2$) and the value is $0.072m^2$. Figure 4.4 shows the free small grid cell for part 2.

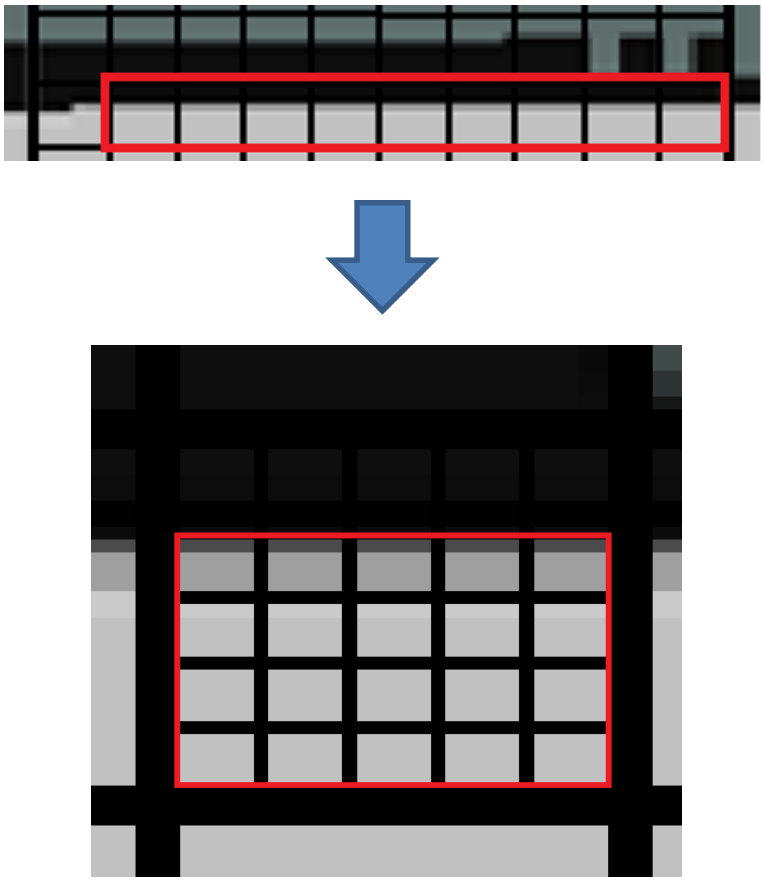


Figure 4.4: Free small grid cell for part 2.

Besides, in this step, only one small grid cell was involved. The area of small grid cell is $0.01m^2$. The small grid cell was divided into 25. Then, to calculate the area of the free small grid cell, $0.01m^2$ was multiplied with 25, so the answer is $0.0004m^2$. As shown in Figure 4.5, the numbers of free small grid cell are 18. Lastly, to calculate the

area of the free small grid cell, the numbers of free small grid cell (18) were multiplied with the area of each free small grid cell ($0.0004m^2$) and the value is $0.0072m^2$. Figure 4.5 shows the free small grid cell for part 3.

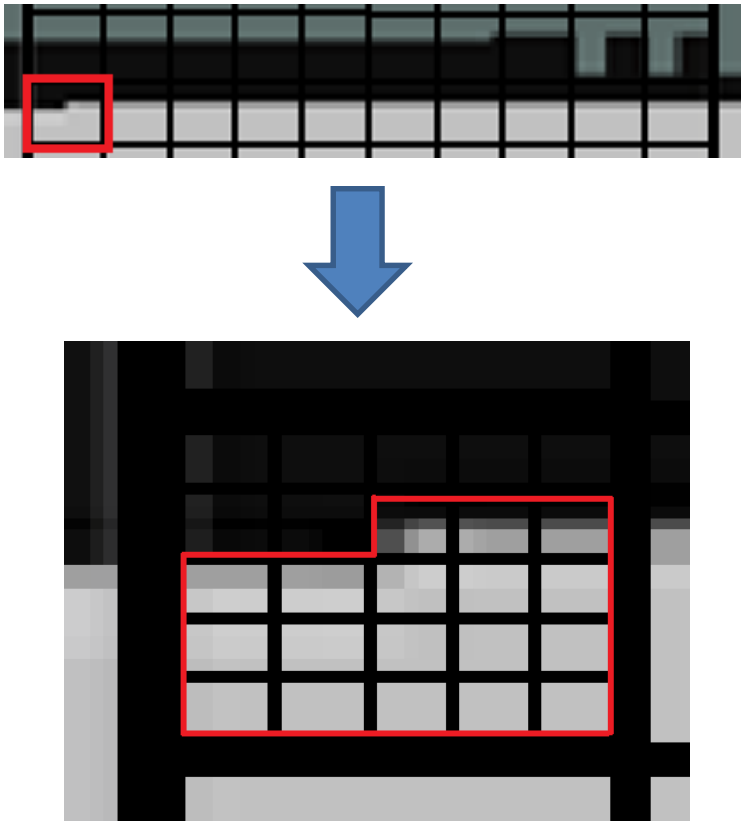


Figure 4.5: Free small grid cell for part 3.

Lastly, to obtain the area of this grid cell, all the calculated values of free small grid cell from the previous step was added together. The total area of this free grid cell is $0.7792m^2$ and was rounded off to $0.78m^2$ ($0.7m^2 + 0.072m^2 + 0.0072m^2$). Then, the same steps were repeated to calculate the area for each free grid cell. Figure 4.6 shows the free small grid cell.

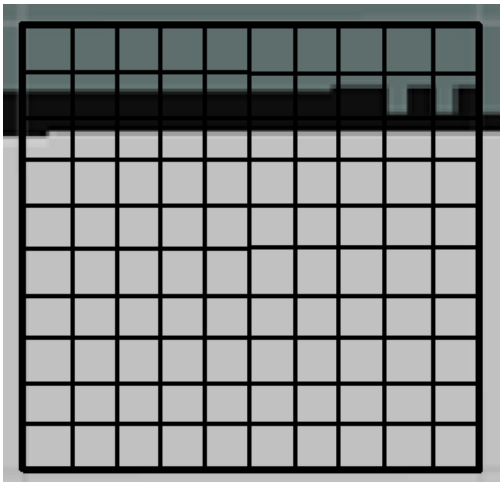


Figure 4.6: Free small grid cell.

4.2 Result and Calculation

4.2.1 Full Environment Map

The full environment for the simulation was shown in Figure 4.7. The area for each grid cell was calculated by using the same method which has been explained in 4.1. Table 4.1 was summarized all the areas of each grid cell for the environment map as shown in Figure 4.7.

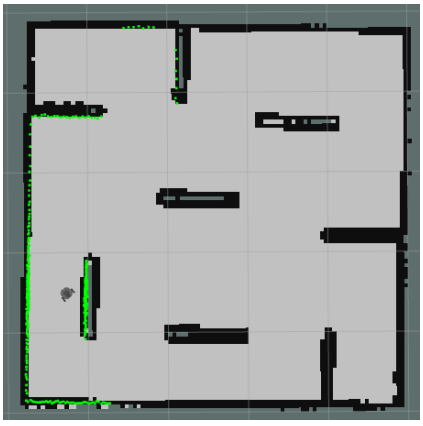


Figure 4.7: Full environment map created.

Table 4.1: Area of each grid cell for full environment map created.

0.75	0.80	0.75	0.78	0.76
0.80	0.98	0.96	0.90	0.90
0.80	0.95	0.85	0.98	0.80
0.80	0.95	0.90	1.00	0.90
0.80	0.90	0.80	0.80	0.80

The total of cell $n_{x,y}$, free and $n_{x,y}$, occupied were calculated for this environment map. For $n_{x,y}$, free, the value was multiplied with the log odd for free space which is 0.9 while for $n_{x,y}$, occupied, the value was multiplied with the log odd occupied space which is 0.7. The equation used for these calculations are:

Total number of cell $n_{x,y}$, free = log odd ↓ free x total area of the environment
= 0.9 x (0.75 + 0.80 + 0.75 + 0.78 + 0.76 + 0.80 + 0.98 + 0.96 + 0.90 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.80 + 0.98 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.80)
= 0.9 x 21.41
= 19.269

Total number of cell $n_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)
= 0.7 x [25 − (0.75 + 0.80 + 0.75 + 0.80 + 0.75 + 0.80 + 0.98 + 0.95 + 0.90 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.80 + 0.98 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.80)]
= 0.7 x 3.59
= 2.513

4.2.2 Speed 0.5ms^{-1} at 20 Seconds

Figure 4.8 shows the environment map created for 20 seconds when speed is 0.5ms^{-1}. The area for each grid cell was calculated by using the same method as stated in 4.1. Table 4.2 was recorded all the areas of each grid cell for the environment map as shown in Figure 4.8.

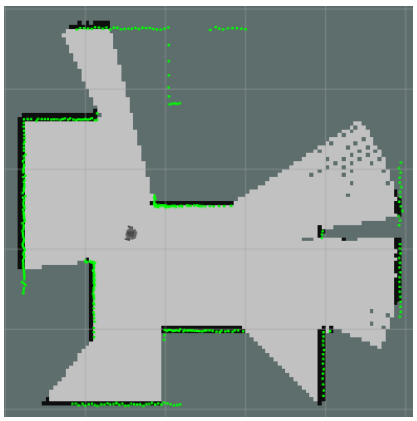


Figure 4.8: Environment map created for 20 seconds when speed is 0.5ms^{-1}.

Table 4.2: Area of each grid cell for environment map created at 20 seconds when speed is 0.5ms^{-1}.

0.10	0.20	0	0	0
0.60	0.65	0	0.05	0.20
0.80	0.90	0.60	0.95	0.75
0.15	0.95	0.90	1.00	0.90
0.15	0.90	0	0.50	0.05

Same as 4.2.1, the total of cell $n_{x,y}$, free and $n_{x,y}$, occupied were calculated for environment map created at 20 seconds when speed is 0.5ms-1. The value was

multiplied with the log odd for free space which is 0.9 for $n_{x,y}$, free while for $n_{x,y}$, occupied, the value was multiplied with the log odd occupied space which is 0.7. The equations used for these calculations are:

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

= 0.9 x (0.10 + 0.20 + 0.60 + 0.65 + 0.05 + 0.20 + 0.80 + 0.90 + 0.60 + 0.95 + 0.75 + 0.15 + 0.95 + 0.90 + 1.00 + 0.90 + 0.15 + 0.90 + 0.50 + 0.05)
= 0.9 x 11.3
= 10.17

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)
= 0.7 x [25 − (0.10 + 0.20 + 0.60 + 0.65 + 0.05 + 0.20 + 0.80 + 0.90 + 0.60 + 0.95 + 0.75 + 0.15 + 0.95 + 0.90 + 1.00 + 0.90 + 0.15 + 0.90 + 0.50 + 0.05)]
= 0.7 x 13.7
= 9.59

Then, the free and occupied ratios were calculated by using the equation below.

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$

$$= \frac{10.17}{19.269}$$
$$= 0.5278$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$

$$= \frac{9.59}{2.513}$$
$$= 3.8162$$

For fitness score, f(m, n), the equation is:

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \begin{vmatrix} (0.75-0.10)+(0.80-0.20)+(0.75-0)+(0.78-0)+(0.76-0)+ \\ (0.80-0.60)+(0.98-0.65)+(0.96-0)+(0.90-0.05)+(0.90-0.20)+ \\ (0.80-0.80)+(0.95-0.90)+(0.85-0.60)+(0.98-0.95)+(0.80-0.75)+ \\ (0.80-0.15)+(0.95-0.95)+(0.90-0.90)+(1.00-1.00)+(0.90-0.90)+ \\ (0.80-0.15)+(0.90-0.90)+(0.80-0)+(0.80-0.50)+(0.80-0.05) \end{vmatrix}}{20}$$

$$f(m,n) = 1 - \frac{10.11}{20}$$

$$f(m,n) = 1 - 0.5055$$

$$f(m,n) = 0.4945$$

4.2.3 Speed 0.5ms^{-1} at 30 Seconds

Figure 4.9 shows the environment map created for 30 seconds when speed is 0.5ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.3. All of the values for each equation were state below and the details of the calculations were shown in Appendix M.

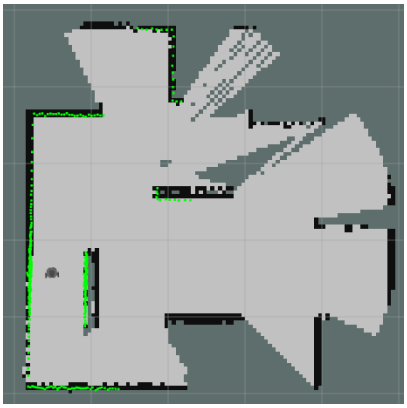


Figure 4.9: Environment map created for 30 seconds when speed is 0.5ms^{-1}.

Table 4.3: Area of each grid cell for environment map created at 30 seconds when speed is 0.5ms^{-1}.

0.10	0.80	0.20	0.10	0
0.60	0.98	0.80	0.20	0.25
0.80	0.95	0.75	0.95	0.75
0.80	0.95	0.90	1.00	0.90
0.80	0.90	0.10	0.50	0.05

Total number of cell $m_{x,y}$, free = 13.617

Total number of cell $m_{x,y}$, occ = 6.909

Free cells ratio = 0.7067

Occupied cells ratio = 2.7493

Fitness score, f(m, n) = 0.7383

4.2.4 Speed 0.5ms^{-1} at 40 Seconds

Figure 4.10 shows the environment map created for 40 seconds when speed is 0.5ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.4. All of the values for each equation were state below and the details of the calculations were shown in Appendix N.

Figure 4.10: Environment map created for 40 seconds when speed is 0.5ms^{-1}.

Table 4.4: Area of each grid cell for environment map created at 40 seconds when speed is 0.5ms^{-1}.

0.10	0.80	0.20	0.10	0
0.60	0.98	0.80	0.20	0.25
0.80	0.95	0.75	0.95	0.75
0.80	0.95	0.90	1.00	0.90
0.80	0.90	0.80	0.80	0.05

Total number of cell $m_{x,y}$, free = 14.517

Total number of cell $m_{x,y}$, occ = 6.209

Free cells ratio = 0.7534

Occupied cells ratio = 2.4708

Fitness Score, f(m, n) = 0.78

4.2.5 Speed 0.5ms^{-1} at 50 Seconds

Figure 4.11 shows the environment map created for 50 seconds when speed is 0.5ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.5. All of the values for each equation were state below and the details of the calculations were shown in Appendix O.

Figure 4.11: Environment map created for 50 seconds when speed is 0.5ms^{-1}.

Table 4.5: Area of each grid cell for environment map created at 50 seconds when speed is 0.5ms^{-1}.

0.10	0.80	0.65	0.10	0.10
0.60	0.98	0.95	0.60	0.70
0.80	0.95	0.80	0.98	0.75
0.80	0.95	0.90	1.00	0.90
0.80	0.90	0.80	0.80	0.80

Total number of cell $m_{x,y}$, free = 16.659

Total number of cell $m_{x,y}$, occ = 4.543

Free cells ratio = 0.8645

Occupied cells ratio = 1.8078

Fitness Score, f(m, n) = 0.8844

4.2.6 Speed 0.5ms^{-1} at 60 Seconds

Figure 4.12 shows the environment map created for 60 seconds when speed is 0.5ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.6. All of the values for each equation were state below and the details of the calculations were shown in Appendix P.

Figure 4.12: Environment map created for 60 seconds when speed is 0.5ms^{-1}.

Table 4.6: Area of each grid cell for environment map created at 60 seconds when speed is 0.5ms^{-1}.

0.10	0.80	0.75	0.80	0.75
0.75	0.98	0.95	0.90	0.90
0.80	0.95	0.85	0.98	0.80
0.80	0.95	0.90	1.00	0.90
0.80	0.90	0.80	0.80	0.80

Total number of cell $m_{x,y}$, free = 18.639

Total number of cell $m_{x,y}$, occ = 3.003

Free cells ratio = 0.9673

Occupied cells ratio = 1.1950

Fitness Score, f(m, n) = 0.9684

4.2.7 Speed 1.0ms^{-1} at 20 Seconds

Figure 4.13 shows the environment map created for 20 seconds when speed is 1.0ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.7. All of the values for each equation were state below and the details of the calculations were shown in Appendix Q.

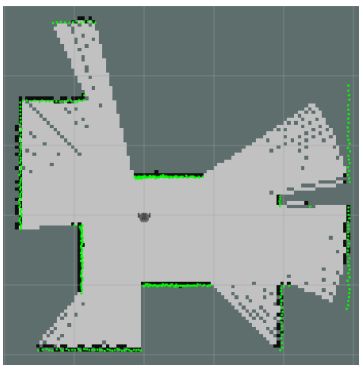


Figure 4.13: Environment map created for 20 seconds when speed is 1.0ms^{-1}.

Table 4.7: Area of each grid cell for environment map created at 20 seconds when speed is 1.0ms^{-1}.

0.10	0.15	0	0	0
0.50	0.50	0	0.10	0.25
0.75	0.85	0.60	0.85	0.60
0.15	0.90	0.90	0.98	0.85
0.15	0.85	0	0.40	0.05

Total number of cell $m_{x,y}$, free = 9.432

Total number of cell $m_{x,y}$, occ = 10.164

Free cells ratio = 0.4895

Occupied cells ratio = 4.0446

Fitness Score, f(m, n) = 0.4535

4.2.8 Speed 1.0ms^{-1} at 30 Seconds

Figure 4.14 shows the environment map created for 30 seconds when speed is 1.0ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.8. All of the values for each equation were state below and the details of the calculations were shown in Appendix R.

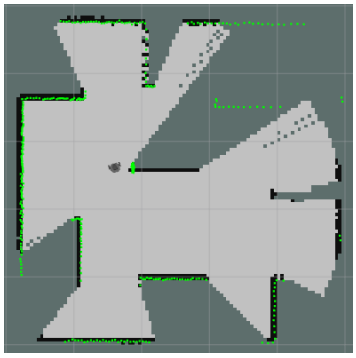


Figure 4.14: Environment map created for 30 seconds when speed is 1.0ms^{-1}.

Table 4.8: Area of each grid cell for environment map created at 30 seconds when speed is 1.0ms^{-1}.

0.10	0.80	0.40	0.05	0
0.55	0.98	0.45	0.10	0.30
0.80	0.90	0.60	0.85	0.70
0.35	0.90	0.90	1.00	0.88
0.20	0.88	0.10	0.50	0.10

Total number of cell $m_{x,y}$, free = 12.051

Total number of cell $m_{x,y}$, occ = 8.127

Free cells ratio = 0.6254

Occupied cells ratio = 3.2340

Fitness Score, f(m, n) = 0.6658

4.2.9 Speed 1.0ms^{-1} at 40 Seconds

Figure 4.15 shows the environment map created for 40 seconds when speed is 1.0ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.9. All of the values for each equation were state below and the details of the calculations were shown in Appendix S.

Figure 4.15: Environment map created for 40 seconds when speed is 1.0ms^{-1}.

Table 4.9: Area of each grid cell for environment map created at 40 seconds when speed is 1.0ms^{-1}.

0.10	0.80	0.50	0.65	0.05
0.55	0.98	0.92	0.55	0.35
0.80	0.95	0.85	0.98	0.75
0.45	0.90	0.90	1.00	0.88
0.20	0.88	0.10	0.50	0.10

Total number of cell $m_{x,y}$, free = 14.121

Total number of cell $m_{x,y}$, occ = 6.517

Free cells ratio = 0.7328

Occupied cells ratio = 2.5933

Fitness Score, f(m, n) = 0.7752

4.2.10 Speed 1.0ms^{-1} at 50 Seconds

Figure 4.16 shows the environment map created for 50 seconds when speed is 1.0ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.10. All of the values for each equation were state below and the details of the calculations were shown in Appendix T.

Figure 4.16: Environment map created for 50 seconds when speed is 1.0ms^{-1}.

Table 4.10: Area of each grid cell for environment map created at 50 seconds when speed is 1.0ms^{-1}.

0.65	0.80	0.65	0.70	0.65
0.70	0.98	0.95	0.80	0.90
0.80	0.95	0.85	0.98	0.80
0.50	0.90	0.90	1.00	0.88
0.20	0.88	0.20	0.65	0.15

Total number of cell $m_{x,y}$, free = 16.578

Total number of cell $m_{x,y}$, occ = 4.606

Free cells ratio = 0.8603

Occupied cells ratio = 1.8329

Fitness Score, f(m, n) = 0.8804

4.2.11 Speed 1.0ms^{-1} at 60 Seconds

Figure 4.17 shows the environment map created for 60 seconds when speed is 1.0ms^{-1}. The area for each grid cell was calculated and was recorded in Table 4.11. All of the values for each equation were state below and the details of the calculations were shown in Appendix U.

Figure 4.17: Environment map created for 60 seconds when speed is 1.0ms^{-1}.

Table 4.11: Area of each grid cell for environment map created at 60 seconds when speed is 1.0ms^{-1}.

0.70	0.80	0.70	0.78	0.76
0.70	0.98	0.96	0.80	0.90
0.80	0.95	0.85	0.98	0.80
0.50	0.90	0.90	1.00	0.90
0.20	0.88	0.50	0.70	0.80

Total number of cell $m_{x,y}$, free = 17.766

Total number of cell $m_{x,y}$, occ = 3.682

Free cells ratio = 0.9220

Occupied cells ratio = 1.4652

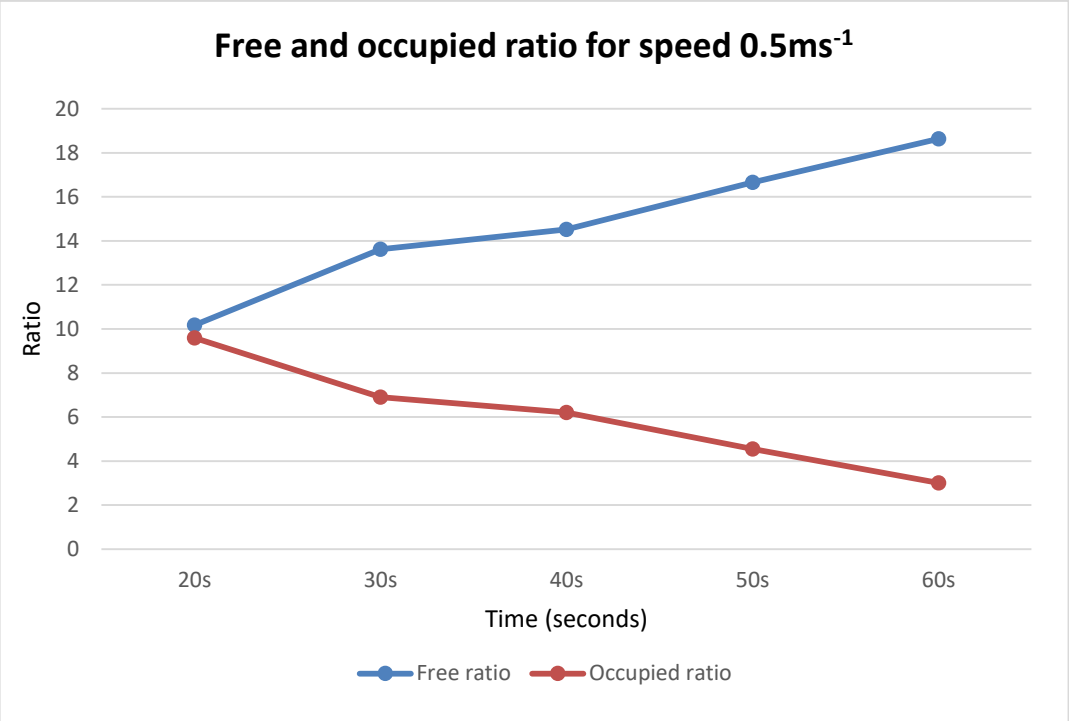
Fitness Score, f(m, n) = 0.9332

4.2.12 Comparison between Free and Occupied Ratio for Speed 0.5ms^{-1}

The comparison between free and occupied ratio for speed 0.5ms^{-1} was recorded in Table 4.12. All the results were obtained from the previous calculations. Then, from the table, the Graph 4.1 was created. In the graph, its shown that the values of free ratio at each seconds are higher compared to the occupied ratio for speed 0.5ms^{-1}. Based on the graph, it can be concluded that, if the free ratio is higher, so the occupied ratio will be lower.

Table 4.12: Free and occupied ratio for speed 0.5ms^{-1}.

Time (seconds)	Speed 0.5ms^{-1}	
	Free ratio	Occupied ratio
20	10.170	9.590
30	13.617	6.909
40	14.517	6.209
50	16.659	4.543
60	18.639	3.003



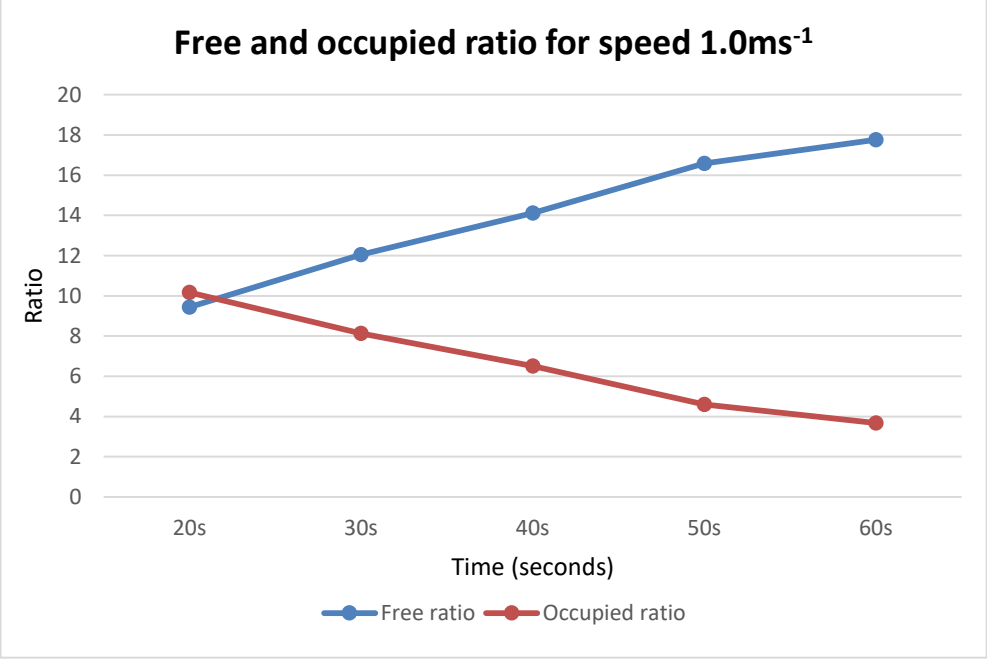
Graph 4.1: Free and occupied ratio for speed 0.5ms^{-1}.

4.2.13 Comparison between Free and Occupied Ratio for Speed 1.0ms^{-1}

Besides, Table 4.13 shows the comparison between free and occupied ratio for speed 1.0ms^{-1}. Graph 4.2 was designed from the values in the Table 4.13. For speed 1.0ms-1, the values of free ratio is also higher compared to the occupied ratio except for 20 seconds which the free ratio has a lower value compared to the occupied ratio. These can be seen clearly in the Graph 4.2.

Table 4.13: Free and occupied ratio for speed 1.0ms^{-1}.

Time (seconds)	Speed 1.0ms^{-1}	
	Free ratio	Occupied ratio
20	9.432	10.164
30	12.051	8.127
40	14.121	6.517
50	16.578	4.606
60	17.766	3.682



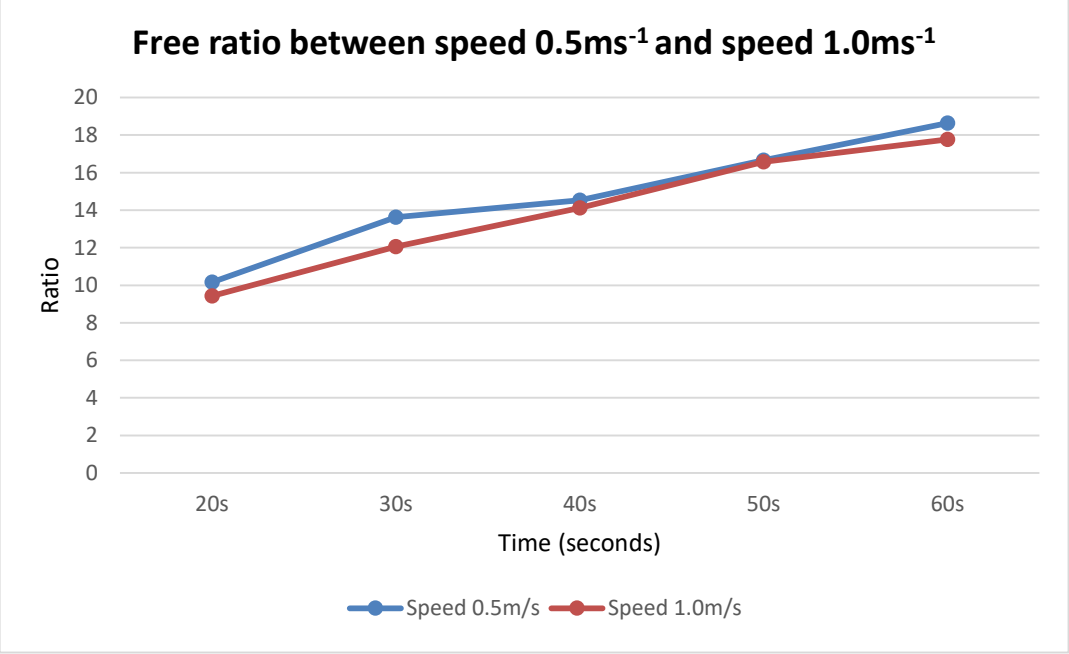
Graph 4.2: Free and occupied ratio for speed 1.0ms^{-1}.

4.2.14 Comparison of Free Ratio between Speed 0.5ms^{-1} and 1.0ms^{-1}

Next, the comparison of free ratio between speed 0.5ms^{-1} and speed 1.0ms^{-1} was summarized in Table 4.14 and the Graph 4.3 was constructed based on this table. From the graph, it can be concluded that the free ratio for speed 0.5ms^{-1} is higher compared to speed 1.0ms^{-1}. In other word, the speed 0.5ms^{-1} was detected less obstacle in the grid cell compared to speed 1.0ms^{-1}.

Table 4.14: Free ratio between speed 0.5ms^{-1} and speed 1.0ms^{-1}.

Time (seconds)	Free ratio for each speed	
	Speed 0.5ms^{-1}	Speed 1.0ms^{-1}
20	10.170	9.432
30	13.617	12.051
40	14.517	14.121
50	16.659	16.578
60	18.639	17.766



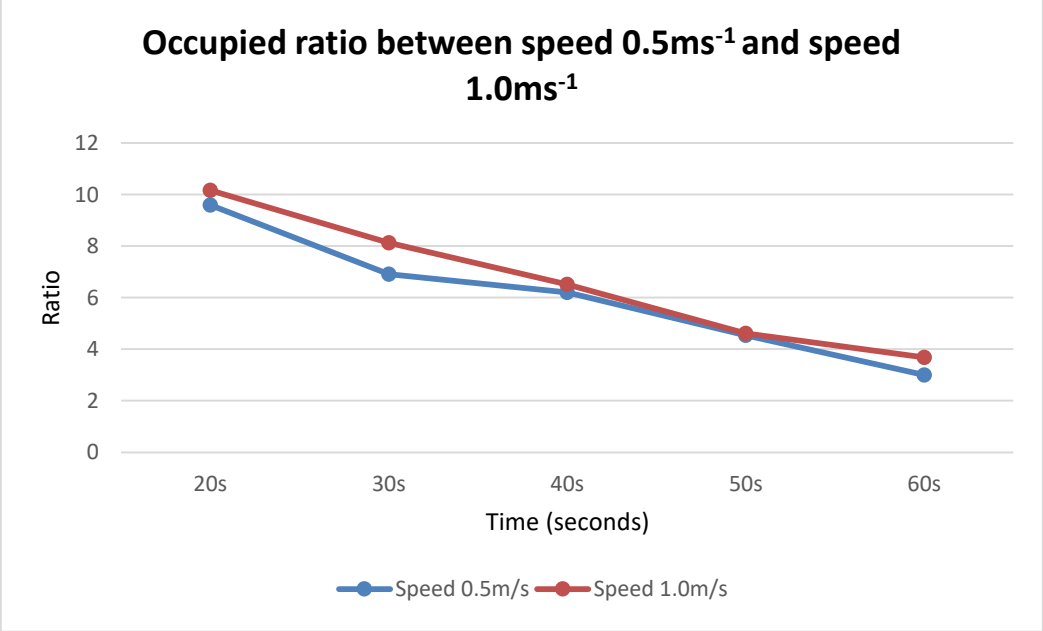
Graph 4.3: Free ratio between speed 0.5ms^{-1} and speed 1.0ms^{-1}.

4.2.15 Comparison of Occupied Ratio between Speed 0.5ms^{-1} and 1.0ms^{-1}

Then, Table 4.15 was recorded the comparison of the occupied ratio between speed 0.5ms^{-1} and speed 1.0ms^{-1}. The Graph 4.4 also was built from Table 4.15. Otherwise from the comparison in 4.2.6, the occupied ratio for speed 0.5ms^{-1} is lower compared to speed 1.0ms^{-1}. In this case, the speed 1.0ms^{-1} was detected more obstacle in the grid cell compared to speed 0.5ms^{-1}.

Table 4.15: Occupied ratio between speed 0.5ms^{-1} and speed 1.0ms^{-1}.

Time (seconds)	Occupied ratio for each speed	
	Speed 0.5ms^{-1}	Speed 1.0ms^{-1}
20	9.590	10.164
30	6.909	8.127
40	6.209	6.517
50	4.543	4.606
60	3.003	3.682



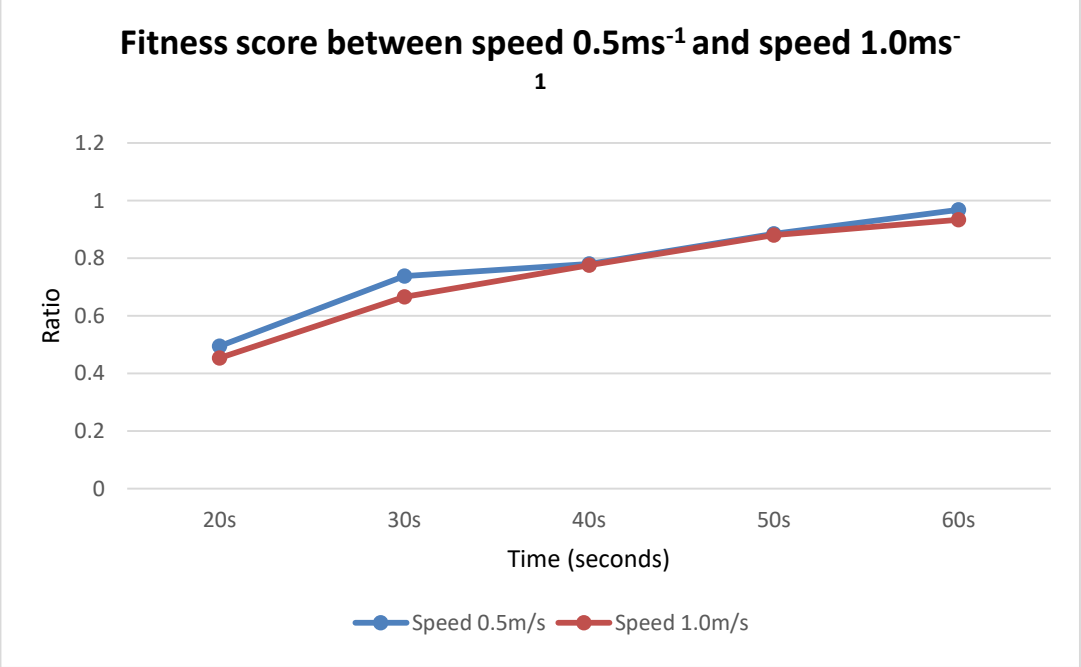
Graph 4.4: Occupied ratio between speed 0.5ms^{-1} and speed 1.0ms^{-1}.

4.2.16 Comparison of Fitness Score between Speed 0.5ms^{-1} and 1.0ms^{-1}

Lastly, the contrast of fitness score between speed 0.5ms^{-1} and 1.0ms^{-1} listed in Table 4.16. Then, the Graph 4.5 was generated. From the graph, the values of fitness score for speed 0.5ms^{-1} is higher compared to speed 1.0ms^{-1}. So, based on the result obtained from the graph, it proved that speed 0.5ms^{-1} has a better accuracy of the map as it fitness value is nearer to one.

Table 4.16: Fitness score between speed 0.5ms^{-1} and speed 1.0ms^{-1}.

Time (seconds)	Fitness score for each speed	
	Speed 0.5ms^{-1}	Speed 1.0ms^{-1}
20	0.4945	0.4535
30	0.7383	0.6658
40	0.7800	0.7752
50	0.8844	0.8804
60	0.9684	0.9332



Graph 4.5: Fitness score between speed 0.5ms^{-1} and speed 1.0ms^{-1}.

4.2.17 Discussion

Based on the study that has been carried out, it can be concluded that the robot's speed must be slow in order to achieve a good result in terms of mapping accuracy. It was found that the speed at which the robot is moving affects the precision of the mapping. In other words, the robot has to be performed slowly to detect the corners. However, it can be operated faster when at a straight direction. An adaptive speed control technique can be built to integrate these findings to vary the speed automatically and thus reduce the overall mapping time.

The results that obtained showed the lower speed ($0.5ms^{-1}$) is more suitable for Turtlebot3 to undergo mapping process and create a more accurate and perfect map compare with the higher speed ($1.0ms^{-1}$). This simulation analysed the accuracy of G-mapping at two different speed. From the simulation, it proved that the differences of the Turtlebot3 speed can influence the accuracy of the SLAM mapping.

The ratio of free and occupied space in the created environment has been calculated for every speed. By using speed $0.5ms^{-1}$, the ratio of free space was 18.639 and it occupied space was 3.003. Whereas the free space for speed $1.0ms^{-1}$ was 17.766 while it occupied space was 3.682.

Besides, from the fitness score calculation, it also shown that the speed $0.5ms^{-1}$ is more accurate compared to speed $1.0ms^{-1}$ as it value was 0.9684 and 0.9332 respectively. This is because, as we know the accuracy of the fitness score is better if it value is nearest to 1. Below shown the equation used to calculate the percent of map accurate for speed $0.5ms^{-1}$:

$$\text{percent of the map accuracy} = \frac{\text{fitness score for speed } 0.5\text{ms}^{-1} - \text{fitness score for speed } 1.0\text{ms}^{-1}}{\text{fitness score for speed } 1.0\text{ms}^{-1}} \times 100\%$$

From the equation above, to calculate the percent of map accuracy for speed 0.5ms^{-1}, the value of 0.9684 has to be minus with 0.9332 and then divided by 0.9332. After that, the answer need to be multiplied with 100% and the result of this calculation is 3.77%. This mean, by using speed 0.5ms^{-1}, the accuracy increase by 3.77%.

Based on the result from the calculation, it can be summarize that the highest free space, the lowest occupied space and the highest fitness score will obtain the better accuracy of mapping process. Therefore, the speed 0.5ms^{-1} is much more suitable to use as the speed for Turtlebot3 to carry out the mapping process for the mapping simulation.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Conclusion

The aim of this project was to investigate the occupancy grid map algorithm parameters, such as grid size. In addition, the final objective of this project was to stimulate the mapping algorithm and investigate the performance based on the accuracy of the maps. In this project, the Ubuntu Operating System was used to present a successful implementation of the ROS robotic platform. By using the commands, the Gazebo and Turtlebot3 Burger were launched. ROS and Gazebo software were function to create a suitable environment for simulating and controlling the mobile robots. It has been shown that after properly designing the robot platform models and

their environments, the device used in the simulation can be used effectively to handle the real robots.

Then, to achieve the project's objectives, the Gmapping SLAM algorithm was launched in Rviz by running the command in ROS. Gmapping is the method that has been used to build the environment map for this project. During the process of creating the map, the robot also able to detect the obstacle in the environment as it is already attached with the sensor. To measure the performance based on the map accuracy, the robot was set into two different speeds which are $0.5ms^{-1}$ and $1.0ms^{-1}$. For each speed, the observation of creating the map was taken at five different seconds (20, 30, 40, 50 and 60 seconds). All of the related calculations for each speed were calculated clearly. In the calculation, the occupancy grid map parameter was used to calculate the total number of cell $m_{x,y}$, free and the total number of cell $m_{x,y}$, occupied. Lastly, to know which speed created the most accurate map, the details of the comparison between these two speeds were also being recorded. From the calculation, it can be concluded that the slower the speed, the better the performance of map accuracy. In other word, for this project, the best speed of mobile robot to build the better map accuracy is $0.5ms^{-1}$.

5.2 Future Works

The benefits of the occupancy grid mapping technique are that the forward models are more realistic than the inverse models. This is because forward models define the physical processes that underlie the generation of data. Next, this method yields more accurate maps, since it is based on fewer assumptions of freedom. However, this technique also has its own drawbacks, such as an obvious increased sensitivity to

changes in the environment and a more times need to go through the data which inhibits its real-time application. The extension of this algorithm into an algorithm online is the subject of future research.

A further potential for future research comes from the fact that surroundings have structure. In this occupancy grid mapping method, the prior probability assumes independence between various grid cells. This is only a simplistic approximation, in real life. Surroundings are typically composed of wider items such as furniture and wall. However, acquiring adequate priors which characterize indoor environments is mainly a range of experiments. This approach focuses on an option way to build maps with mobile robots regardless of this barriers.

REFERENCES

[1] P. Kim, J. Chen, and Y. K. Cho, “Autonomous mobile robot localization and mapping for unknown construction environments,” *Constr. Res. Congr. 2018 Constr. Inf. Technol. - Sel. Pap. from Constr. Res. Congr. 2018*, vol. 2018-April, pp. 147–156, 2018, doi: 10.1061/9780784481264.015.

[2] S. Thrun, “Robotic Mapping: A Survey,” *Science (80-.).*, vol. 298, no. February, pp. 1–35, 2002, doi: 10.1126/science.298.5594.699f.

[3] S. Seong, C. Lee, and J. Kim, *Multi-robot SLAM: An Overview and Quantitative Evaluation of MRGS ROS Framework for MR-SLAM*, no. January. Springer International Publishing, 2019.

[4] N. M. Yatim and N. Buniyamin, “Indoor mapping with machine learning algorithm using Khepera III mobile robot,” *J. Telecommun. Electron. Comput. Eng.*, vol. 8, no. 9, pp. 61–66, 2016.

[5] J. M. Santos, D. Portugal, and R. P. Rocha, “An evaluation of 2D SLAM techniques available in Robot Operating System,” *2013 IEEE Int. Symp. Safety, Secur. Rescue Robot. SSRR 2013*, 2013, doi: 10.1109/SSRR.2013.6719348.

[6] S. Thrun, “Learning occupancy grid maps with forward sensor models,” *Auton. Robots*,

vol. 15, no. 2, pp. 111–127, 2003, doi: 10.1023/A:1025584807625.

[7] M. Drahansky *et al.*, “Occupancy Grid Maps for Localization and Mapping,” *Intech*, vol. i, no. tourism, p. 13, 2016, doi: http://dx.doi.org/10.5772/57353.

[8] Y. Abdelrasoul, A. B. S. H. Saman, and P. Sebastian, “A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM,” *2016 2nd IEEE Int. Symp. Robot. Manuf. Autom. ROMA 2016*, 2017, doi: 10.1109/ROMA.2016.7847825.

[9] B. Steux and O. El Hamzaoui, “tinySLAM: A SLAM algorithm in less than 200 lines C-language program,” *11th Int. Conf. Control. Autom. Robot. Vision, ICARCV 2010*, no. December, pp. 1975–1979, 2010, doi: 10.1109/ICARCV.2010.5707402.

[10] N. Kumar, Z. Vamossy, and Z. M. Szabo-Resch, “Robot obstacle avoidance using bumper event,” *SACI 2016 - 11th IEEE Int. Symp. Appl. Comput. Intell. Informatics, Proc.*, pp. 485–490, 2016, doi: 10.1109/SACI.2016.7507426.

[11] A. Stoyanov, T Louloudi, A Andreasson, H Lilienthal, “Comparative evaluation of range sensor accuracy in indoor environments,” *Proc. 5th Eur. Conf. Mob. Robot. ECMR 2011*, pp. 19–24, 2011.

[12] M. Antunes, J. P. Barreto, C. Premebida, and U. Nunes, “Can stereo vision replace a Laser Rangefinder?,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 5183–5190, 2012, doi: 10.1109/IROS.2012.6385844.

[13] H. Surmann, K. Lingemann, a. Nüchter, and J. Hertzberg, “A 3D laser range finder for autonomous mobile robots,” *Proc. 32nd ISR (International Symp. Robot.*, vol. 19, no. 21, pp. 153–158, 2001.

[14] H. A. Sidharta, S. Sidharta, and W. P. Sari, “2D Mapping and boundary detection using

2D LIDAR sensor for prototyping Autonomous PETIS (Programable Vehicle with Integrated Sensor)," *Kinet. Game Technol. Inf. Syst. Comput. Network, Comput. Electron. Control*, vol. 4, no. 2, pp. 107–114, 2019, doi: 10.22219/kinetik.v4i2.731.

[15] G. Adamo and A. Busacca, "Time of Flight measurements via two LiDAR systems with SiPM and APD," *AEIT 2016 - Int. Annu. Conf. Sustain. Dev. Mediterr. Area, Energy ICT Networks Futur.*, 2016, doi: 10.23919/AEIT.2016.7892802.

[16] D. Lv, X. Ying, Y. Cui, J. Song, K. Qian, and M. Li, "Research on the technology of LIDAR data processing," *1st Int. Conf. Electron. Instrum. Inf. Syst. EIIS 2017*, vol. 2018-Janua, pp. 1–5, 2018, doi: 10.1109/EIIS.2017.8298694.

[17] B. Al Housani, B. Mutrib, and H. Jaradi, "The Linux review - Ubuntu desktop edition - Version 8.10," *Proc. 2009 Int. Conf. Curr. Trends Inf. Technol. CTIT 2009*, pp. 67–72, 2009, doi: 10.1109/CTIT.2009.5423142.

[18] D. Singh, E. Trivedi, Y. Sharma, and V. Niranjan, "TurtleBot: Design and hardware component selection," *2018 Int. Conf. Comput. Power Commun. Technol. GUCON 2018*, pp. 805–809, 2019, doi: 10.1109/GUCON.2018.8675050.

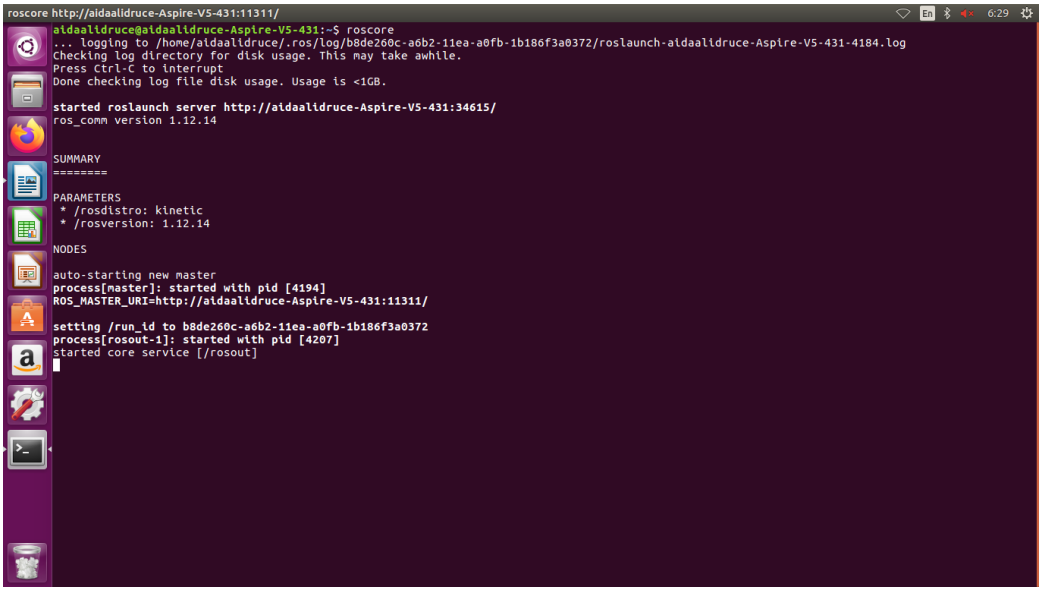
[19] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, "Simulation environment for mobile robots testing using ROS and Gazebo," *2016 20th Int. Conf. Syst. Theory, Control Comput. ICSTCC 2016 - Jt. Conf. SINTES 20, SACCS 16, SIMSIS 20 - Proc.*, pp. 96–101, 2016, doi: 10.1109/ICSTCC.2016.7790647.

[20] H. Aagela, M. Al-Nesf, and V. Holmes, "An Asus-xtion-probased indoor MAPPING using a Raspberry Pi with Turtlebot robot Turtlebot robot," *ICAC 2017 - 2017 23rd IEEE Int. Conf. Autom. Comput. Addressing Glob. Challenges through Autom. Comput.*, no. September, pp. 7–8, 2017, doi: 10.23919/IConAC.2017.8082023.

[21] H. Deilamsalehy and T. C. Havens, “Sensor fused three-dimensional localization using IMU, camera and LiDAR,” *Proc. IEEE Sensors*, pp. 7–9, 2017, doi: 10.1109/ICSENS.2016.7808523.

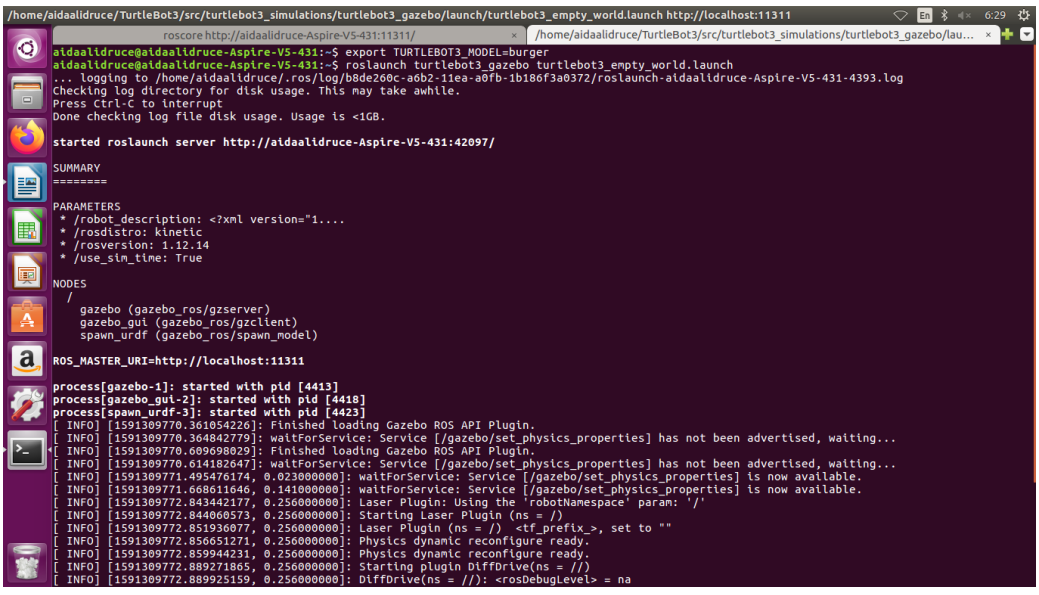
APPENDIX A

Run roscore in ROS.



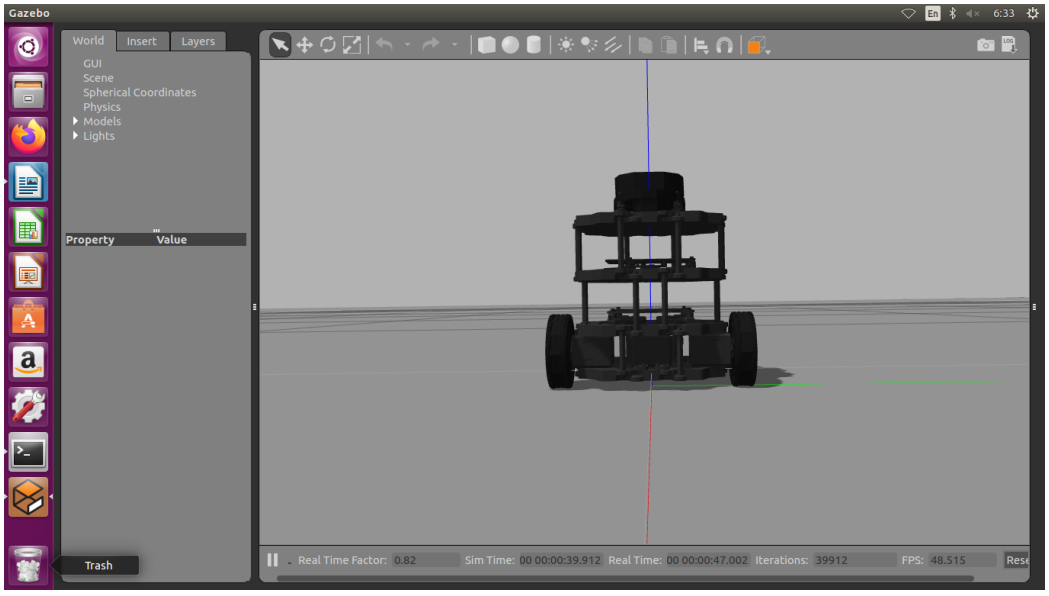
APPENDIX B

Launch TurtleBot Burger and simulation environment in gazebo.



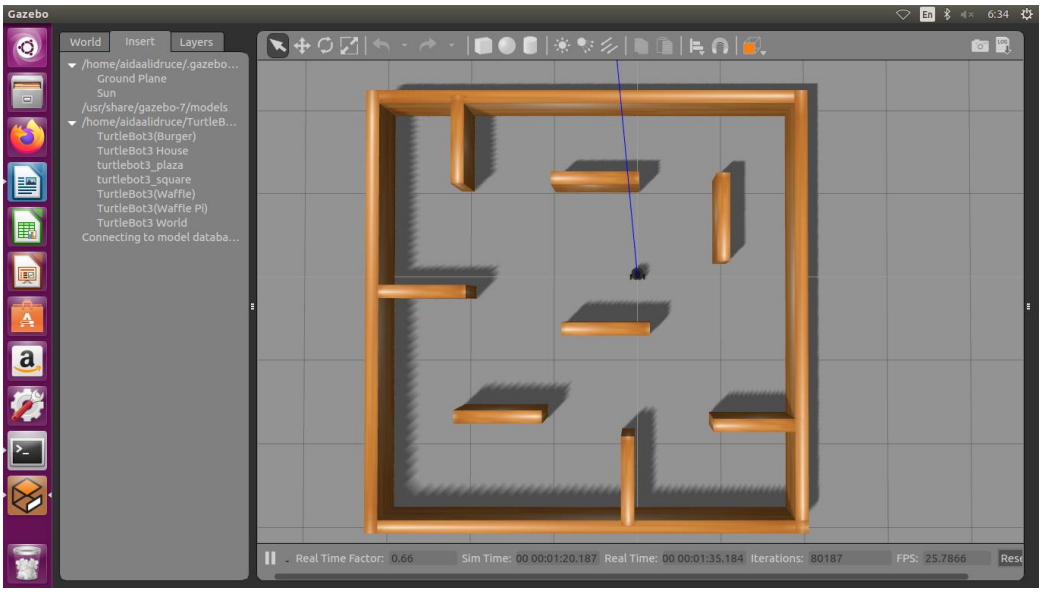
APPENDIX C

TurtleBot Burger in gazebo.



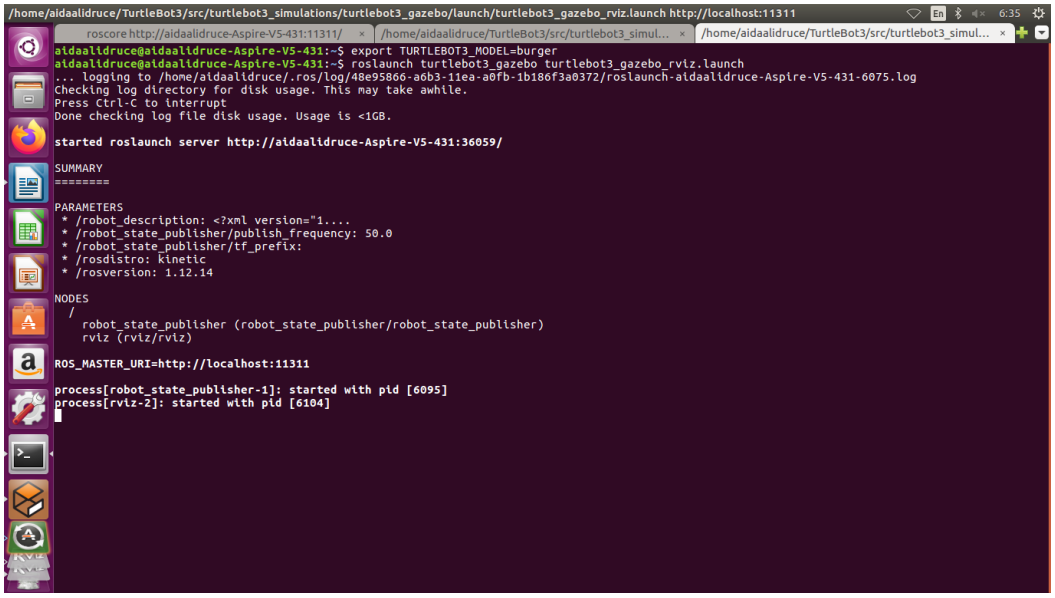
APPENDIX D

Simulation environment in gazebo.



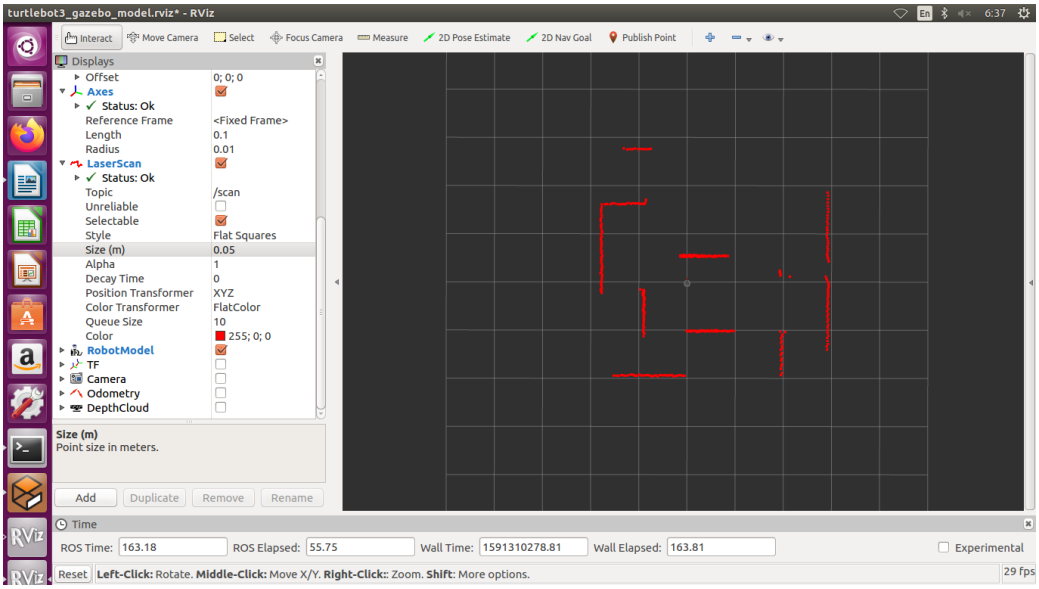
APPENDIX E

Execute Rviz in ROS.



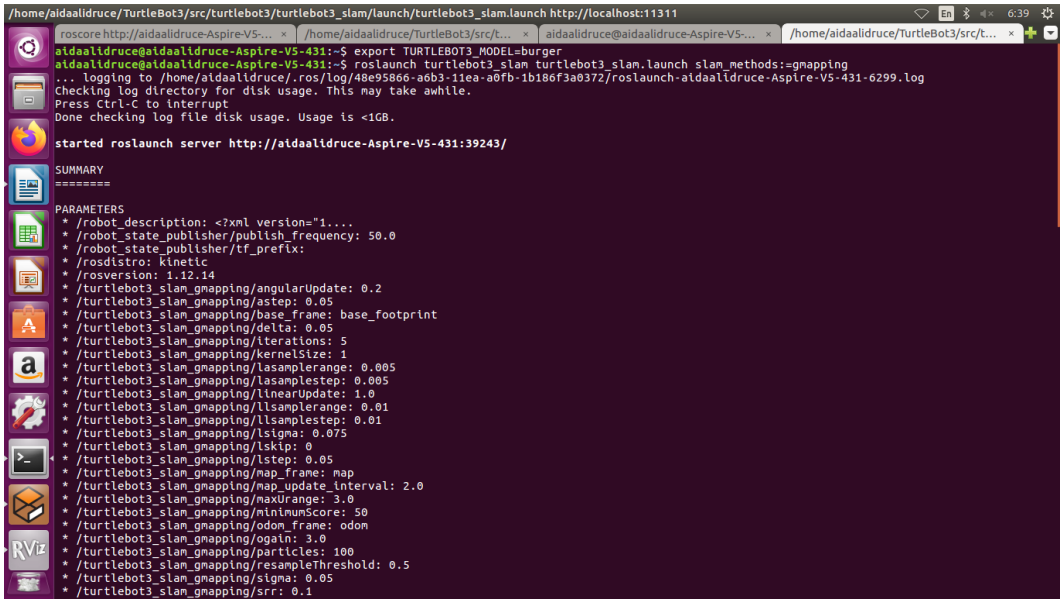
APPENDIX F

Rviz in ROS.



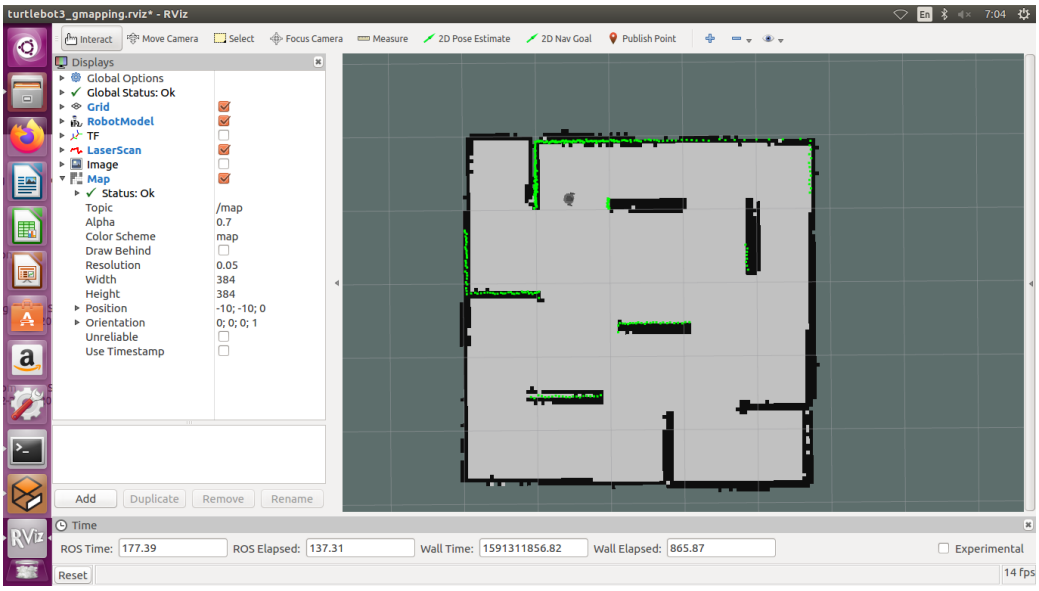
APPENDIX G

Run Gmapping SLAM in ROS.



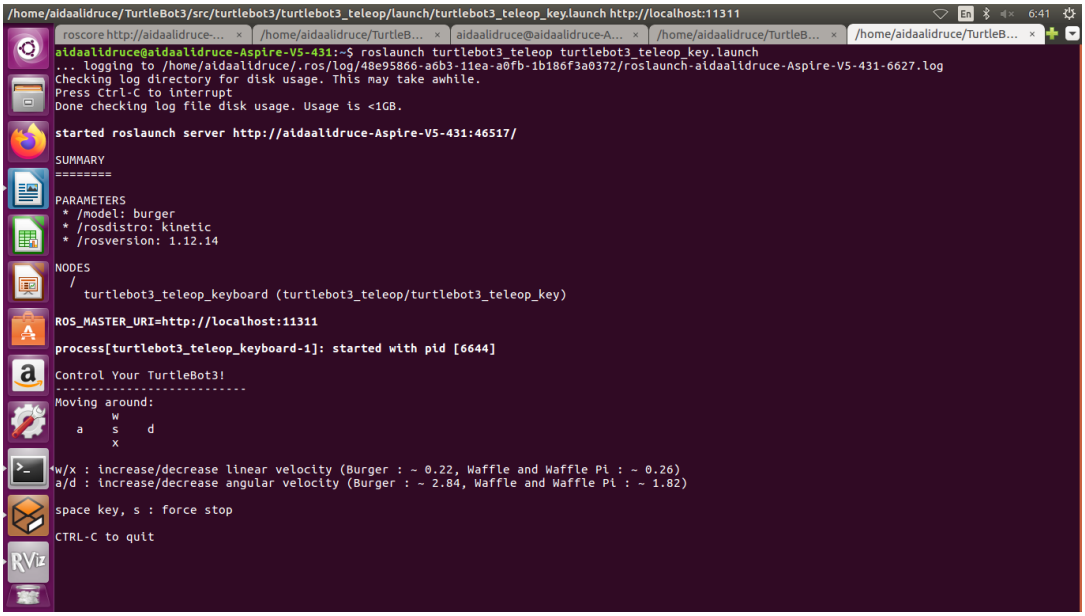
APPENDIX H

Gmapping SLAM in ROS.



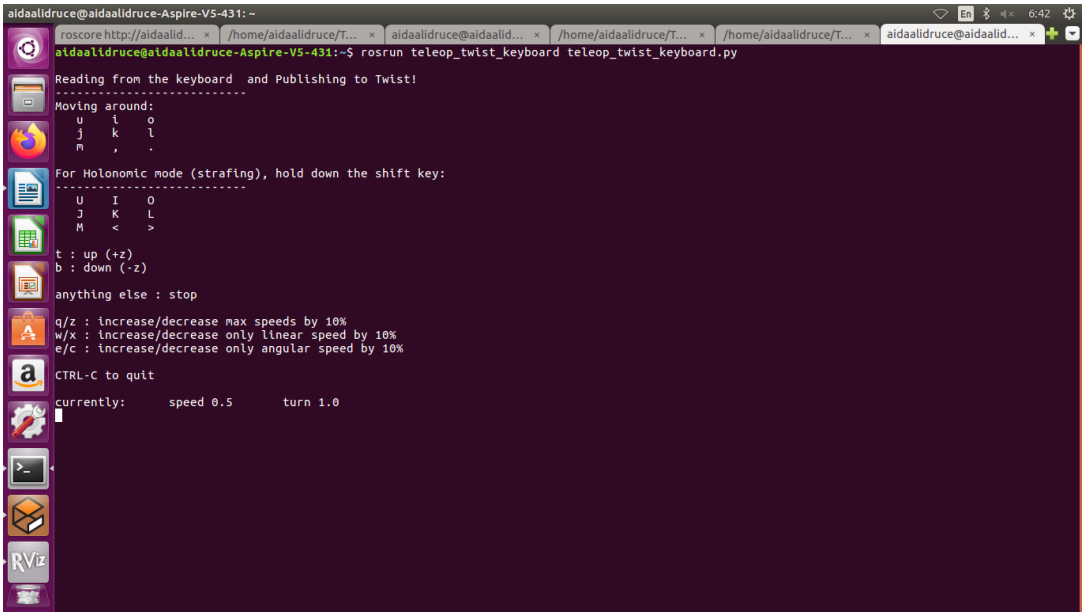
APPENDIX I

Run teleop key to control the keyboard for movement of the TurtleBot Burger.



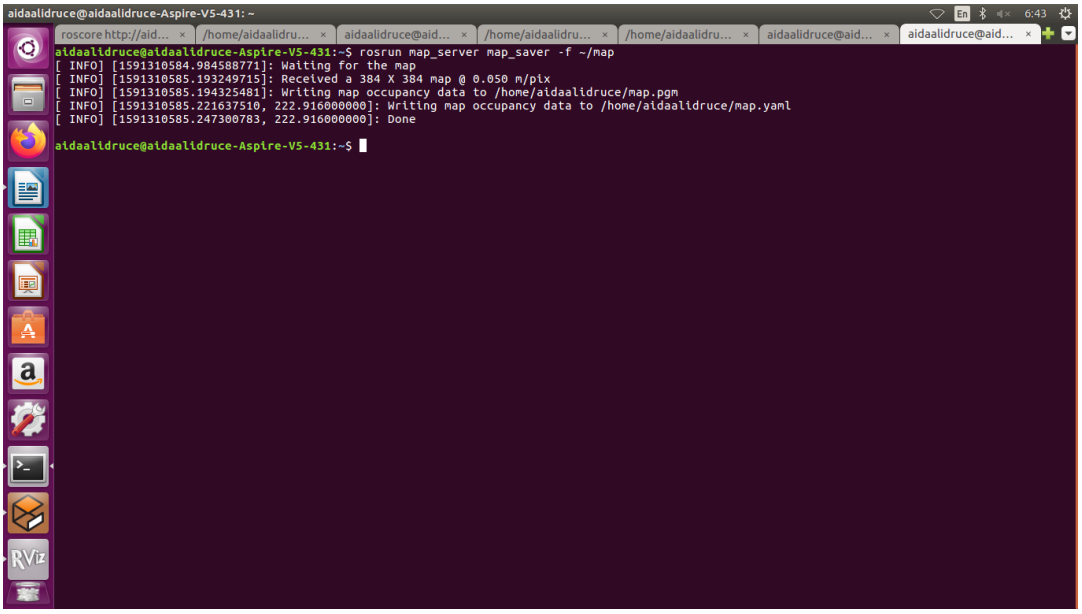
APPENDIX J

Run the teleop twist to control the speed of the TurtleBot Burger.



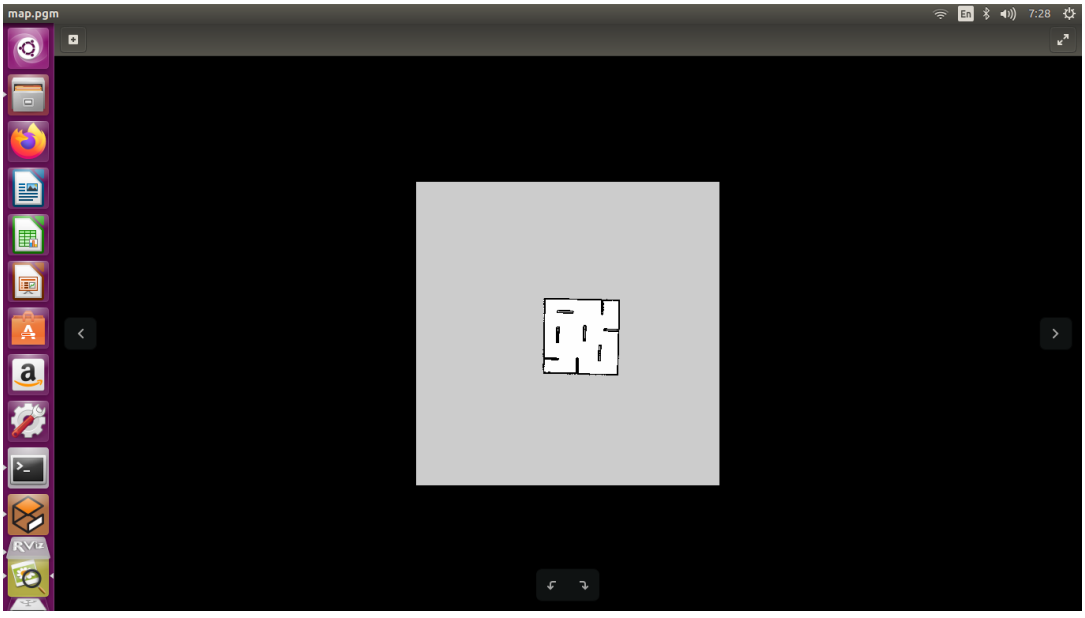
APPENDIX K

Run map server to save the map.



APPENDIX L

The save map.



APPENDIX M

The calculation for speed 0.5ms^{-1} at 30 seconds.

Calculation For Speed 0.5ms^{-1} At 30 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.10 + 0.80 + 0.20 + 0.10 + 0.60 + 0.98 + 0.80 + 0.20 + 0.25 + 0.80 + 0.95 + 0.75 + 0.95 + 0.75 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.10 + 0.50 + 0.05)$$
$$= 0.9 \text{ x } 15.13$$
$$= 13.617$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.10 + 0.80 + 0.20 + 0.10 + 0.60 + 0.98 + 0.80 + 0.20 + 0.25 + 0.80 + 0.95 + 0.75 + 0.95 + 0.75 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.10 + 0.50 + 0.05)]$$
$$= 0.7 \text{ x } 9.87$$
$$= 6.909$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{13.617}{19.269}$$
$$= 0.7067$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{6.909}{2.513}$$
$$= 2.7493$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \begin{vmatrix} (0.75 - 0.10) + (0.80 - 0.80) + (0.75 - 0.20) + (0.78 - 0.10) + (0.76 - 0) + \\ (0.80 - 0.60) + (0.98 - 0.98) + (0.96 - 0.80) + (0.90 - 0.20) + (0.90 - 0.25) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.85 - 0.75) + (0.98 - 0.95) + (0.80 - 0.75) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.90) + \\ (0.80 - 0.80) + (0.90 - 0.90) + (0.80 - 0.10) + (0.80 - 0.50) + (0.80 - 0.05) \end{vmatrix}}{24}$$

$$f(m,n) = 1 - \frac{6.28}{24}$$
$$f(m,n) = 1 - 0.2617$$
$$f(m,n) = 0.7383$$

APPENDIX N

The calculation for speed 0.5ms^{-1} at 40 seconds.

Calculation For Speed 0.5ms^{-1} At 40 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \times (0.10 + 0.80 + 0.20 + 0.10 + 0.60 + 0.98 + 0.80 + 0.20 + 0.25 + 0.80 + 0.95 + 0.75 + 0.95 + 0.75 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.05)$$
$$= 0.9 \times 16.13$$
$$= 14.517$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \times [25 - (0.10 + 0.80 + 0.20 + 0.10 + 0.60 + 0.98 + 0.80 + 0.20 + 0.25 + 0.80 + 0.95 + 0.75 + 0.95 + 0.75 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.05)]$$
$$= 0.7 \times 8.87$$
$$= 6.209$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{14.517}{19.269}$$
$$= 0.7534$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{6.209}{2.513}$$
$$= 2.4708$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{matrix} (0.75-0.10)+(0.80-0.80)+(0.75-0.20)+(0.78-0.10)+(0.76-0)+ \\ (0.80-0.60)+(0.98-0.98)+(0.96-0.80)+(0.90-0.20)+(0.90-0.25)+ \\ (0.80-0.80)+(0.95-0.95)+(0.85-0.75)+(0.98-0.95)+(0.80-0.75)+ \\ (0.80-0.80)+(0.95-0.95)+(0.90-0.90)+(1.00-1.00)+(0.90-0.90)+ \\ (0.80-0.80)+(0.90-0.90)+(0.80-0.80)+(0.80-0.80)+(0.80-0.05) \end{matrix} \right|}{24}$$

$$f(m,n) = 1 - \frac{5.28}{24}$$
$$f(m,n) = 1 - 0.22$$
$$f(m,n) = 0.78$$

APPENDIX O

The calculation for speed 0.5ms^{-1} at 50 seconds.

Calculation For Speed 0.5ms^{-1} At 50 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.10 + 0.80 + 0.65 + 0.10 + 0.10 + 0.60 + 0.98 + 0.95 + 0.60 + 0.70 + 0.80 + 0.95 + 0.80 + 0.98 + 0.75 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.80)$$

$$= 0.9 \text{ x } 18.51$$

$$= 16.659$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.10 + 0.80 + 0.65 + 0.10 + 0.10 + 0.60 + 0.98 + 0.95 + 0.60 + 0.70 + 0.80 + 0.95 + 0.80 + 0.98 + 0.75 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.80)]$$

$$= 0.7 \text{ x } 6.49$$

$$= 4.543$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$

$$= \frac{16.659}{19.269}$$

$$= 0.8645$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$

$$= \frac{4.543}{2.513}$$

$$= 1.8078$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \begin{vmatrix} (0.75 - 0.10) + (0.80 - 0.80) + (0.75 - 0.65) + (0.78 - 0.10) + (0.76 - 0.10) + \\ (0.80 - 0.60) + (0.98 - 0.98) + (0.96 - 0.95) + (0.90 - 0.60) + (0.90 - 0.70) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.85 - 0.80) + (0.98 - 0.98) + (0.80 - 0.75) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.90) + \\ (0.80 - 0.80) + (0.90 - 0.90) + (0.80 - 0.80) + (0.80 - 0.80) + (0.80 - 0.05) \end{vmatrix}}{25}$$

$$f(m,n) = 1 - \frac{2.89}{25}$$

$$f(m,n) = 1 - 0.1156$$

$$f(m,n) = 0.8844$$

APPENDIX P

The calculation for speed 0.5ms^{-1} at 60 seconds.

Calculation For Speed 0.5ms^{-1} At 60 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.10 + 0.80 + 0.75 + 0.80 + 0.75 + 0.75 + 0.98 + 0.95 + 0.90 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.80)$$

$$= 0.9 \text{ x } 20.71$$

$$= 18.639$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.10 + 0.80 + 0.75 + 0.80 + 0.75 + 0.75 + 0.98 + 0.95 + 0.90 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.80 + 0.95 + 0.90 + 1.00 + 0.90 + 0.80 + 0.90 + 0.80 + 0.80 + 0.80)]$$

$$= 0.7 \text{ x } 4.29$$

$$= 3.00$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$

$$= \frac{18.639}{19.269}$$

$$= 0.9673$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$

$$= \frac{3.003}{2.513}$$

$$= 1.1950$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{matrix} (0.75 - 0.10) + (0.80 - 0.80) + (0.75 - 0.75) + (0.78 - 0.80) + (0.76 - 0.70) + \\ (0.80 - 0.75) + (0.98 - 0.98) + (0.96 - 0.95) + (0.90 - 0.90) + (0.90 - 0.90) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.85 - 0.85) + (0.98 - 0.98) + (0.80 - 0.80) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.90) + \\ (0.80 - 0.80) + (0.90 - 0.90) + (0.80 - 0.80) + (0.80 - 0.80) + (0.80 - 0.80) \end{matrix} \right|}{25}$$

$$f(m,n) = 1 - \frac{0.79}{25}$$

$$f(m,n) = 1 - 0.0316$$

$$f(m,n) = 0.9684$$

APPENDIX Q

The calculation for speed 1.0ms^{-1} at 20 seconds.

Calculation For Speed 1.0ms^{-1} At 20 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.10 + 0.15 + 0.50 + 0.50 + 0.10 + 0.25 + 0.75 + 0.85 + 0.60 + 0.85 + 0.60 + 0.15 + 0.90 + 0.90 + 0.98 + 0.85 + 0.15 + 0.85 + 0.40 + 0.05)$$
$$= 0.9 \text{ x } 10.48$$
$$= 9.432$$

Total number of cell $m_{x,y}$, occ = log odd ↓occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.10 + 0.15 + 0.50 + 0.50 + 0.10 + 0.25 + 0.75 + 0.85 + 0.60 + 0.85 + 0.60 + 0.15 + 0.90 + 0.90 + 0.98 + 0.85 + 0.15 + 0.85 + 0.40 + 0.05)]$$
$$= 0.7 \text{ x } 14.52$$
$$= 10.164$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{9.432}{19.269}$$
$$= 0.4895$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{10.164}{2.513}$$
$$= 4.0446$$

Fitness Score, f(m , n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{array}{c} (0.75 - 0.10) + (0.80 - 0.15) + (0.75 - 0) + (0.78 - 0) + (0.76 - 0) + \\ (0.80 - 0.50) + (0.98 - 0.50) + (0.96 - 0) + (0.90 - 0.10) + (0.90 - 0.25) + \\ (0.80 - 0.75) + (0.95 - 0.85) + (0.85 - 0.60) + (0.98 - 0.85) + (0.80 - 0.60) + \\ (0.80 - 0.15) + (0.95 - 0.90) + (0.90 - 0.90) + (1.00 - 0.98) + (0.90 - 0.85) + \\ (0.80 - 0.15) + (0.90 - 0.85) + (0.80 - 0) + (0.80 - 0.40) + (0.80 - 0.05) \end{array} \right|}{20}$$

$$f(m,n) = 1 - \frac{10.93}{20}$$
$$f(m,n) = 1 - 0.5465$$
$$f(m,n) = 0.4535$$

APPENDIX R

The calculation for speed 1.0ms^{-1} at 30 seconds.

Calculation For Speed 1.0ms^{-1} At 30 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \times (0.10 + 0.80 + 0.40 + 0.05 + 0.55 + 0.98 + 0.45 + 0.10 + 0.30 + 0.80 + 0.90 + 0.60 + 0.85 + 0.70 + 0.35 + 0.90 + 0.90 + 1.00 + 0.88 + 0.20 + 0.88 + 0.10 + 0.50 + 0.10)$$
$$= 0.9 \times 13.39$$
$$= 12.051$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \times [25 - (0.10 + 0.80 + 0.40 + 0.05 + 0.55 + 0.98 + 0.45 + 0.10 + 0.30 + 0.80 + 0.90 + 0.60 + 0.85 + 0.70 + 0.35 + 0.90 + 0.90 + 1.00 + 0.88 + 0.20 + 0.88 + 0.10 + 0.50 + 0.10)]$$
$$= 0.7 \times 11.61$$
$$= 8.127$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{12.051}{19.269}$$
$$= 0.6254$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{8.127}{2.513}$$
$$= 3.2340$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{array}{l} (0.75 - 0.10) + (0.80 - 0.80) + (0.75 - 0.40) + (0.78 - 0.05) + (0.76 - 0) + \\ (0.80 - 0.55) + (0.98 - 0.98) + (0.96 - 0.45) + (0.90 - 0.10) + (0.90 - 0.30) + \\ (0.80 - 0.80) + (0.95 - 0.90) + (0.85 - 0.60) + (0.98 - 0.85) + (0.80 - 0.70) + \\ (0.80 - 0.35) + (0.95 - 0.90) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.88) + \\ (0.80 - 0.20) + (0.90 - 0.88) + (0.80 - 0.10) + (0.80 - 0.50) + (0.80 - 0.10) \end{array} \right|}{24}$$

$$f(m,n) = 1 - \frac{8.02}{24}$$
$$f(m,n) = 1 - 0.3342$$
$$f(m,n) = 0.6658$$

APPENDIX S

The calculation for speed 1.0ms^{-1} at 40 seconds.

Calculation For Speed 1.0ms^{-1} At 40 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.10 + 0.80 + 0.50 + 0.65 + 0.05 + 0.55 + 0.98 + 0.92 + 0.55 + 0.35 + 0.80 + 0.95 + 0.85 + 0.98 + 0.75 + 0.45 + 0.90 + 0.90 + 1.00 + 0.88 + 0.20 + 0.88 + 0.10 + 0.50 + 0.10)$$
$$= 0.9 \text{ x } 15.69$$
$$= 14.121$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.10 + 0.80 + 0.50 + 0.65 + 0.05 + 0.55 + 0.98 + 0.92 + 0.55 + 0.35 + 0.80 + 0.95 + 0.85 + 0.98 + 0.75 + 0.45 + 0.90 + 0.90 + 1.00 + 0.88 + 0.20 + 0.88 + 0.10 + 0.50 + 0.10)]$$
$$= 0.7 \text{ x } 9.31$$
$$= 6.517$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{14.121}{19.269}$$
$$= 0.7328$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{6.517}{2.513}$$
$$= 2.5933$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{matrix} (0.75 - 0.10) + (0.80 - 0.80) + (0.75 - 0.50) + (0.78 - 0.65) + (0.76 - 0.05) + \\ (0.80 - 0.55) + (0.98 - 0.98) + (0.96 - 0.92) + (0.90 - 0.55) + (0.90 - 0.35) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.85 - 0.85) + (0.98 - 0.98) + (0.80 - 0.75) + \\ (0.80 - 0.45) + (0.95 - 0.90) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.88) + \\ (0.80 - 0.20) + (0.90 - 0.88) + (0.80 - 0.10) + (0.80 - 0.50) + (0.80 - 0.10) \end{matrix} \right|}{25}$$

$$f(m,n) = 1 - \frac{5.62}{25}$$
$$f(m,n) = 1 - 0.2248$$
$$f(m,n) = 0.7752$$

APPENDIX T

The calculation for speed 1.0ms^{-1} at 50 seconds.

Calculation For Speed 1.0ms^{-1} At 50 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.65 + 0.80 + 0.65 + 0.70 + 0.65 + 0.70 + 0.98 + 0.95 + 0.80 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.50 + 0.90 + 0.90 + 1.00 + 0.88 + 0.20 + 0.88 + 0.20 + 0.65 + 0.15)$$
$$= 0.9 \text{ x } 18.42$$
$$= 16.578$$

Total number of cell $m_{x,y}$, occ = log odd ↓ occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.65 + 0.80 + 0.65 + 0.70 + 0.65 + 0.70 + 0.98 + 0.95 + 0.80 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.50 + 0.90 + 0.90 + 1.00 + 0.88 + 0.20 + 0.88 + 0.20 + 0.65 + 0.15)]$$
$$= 0.7 \text{ x } 6.58$$
$$= 4.606$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{16.578}{19.269}$$
$$= 0.8603$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{4.606}{2.513}$$
$$= 1.8329$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{array}{l} (0.75 - 0.65) + (0.80 - 0.80) + (0.75 - 0.65) + (0.78 - 0.70) + (0.76 - 0.65) + \\ (0.80 - 0.70) + (0.98 - 0.98) + (0.96 - 0.95) + (0.90 - 0.80) + (0.90 - 0.90) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.85 - 0.85) + (0.98 - 0.98) + (0.80 - 0.80) + \\ (0.80 - 0.50) + (0.95 - 0.90) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.88) + \\ (0.80 - 0.20) + (0.90 - 0.88) + (0.80 - 0.20) + (0.80 - 0.65) + (0.80 - 0.15) \end{array} \right|}{25}$$

$$f(m,n) = 1 - \frac{2.99}{25}$$
$$f(m,n) = 1 - 0.1196$$
$$f(m,n) = 0.8804$$

APPENDIX U

The calculation for speed 1.0ms^{-1} at 60 seconds.

Calculation For Speed 1.0ms^{-1} At 60 Seconds

Total number of cell $m_{x,y}$, free = log odd ↓ free x total area of the environment

$$= 0.9 \text{ x } (0.70 + 0.80 + 0.70 + 0.78 + 0.76 + 0.70 + 0.98 + 0.96 + 0.80 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.50 + 0.90 + 0.90 + 1.00 + 0.90 + 0.20 + 0.88 + 0.50 + 0.70 + 0.80)$$
$$= 0.9 \text{ x } 19.74$$
$$= 17.766$$

Total number of cell $m_{x,y}$, occ = log odd ↓occ x (total area − total area of the environment)

$$= 0.7 \text{ x } [25 - (0.70 + 0.80 + 0.70 + 0.78 + 0.76 + 0.70 + 0.98 + 0.96 + 0.80 + 0.90 + 0.80 + 0.95 + 0.85 + 0.98 + 0.80 + 0.50 + 0.90 + 0.90 + 1.00 + 0.90 + 0.20 + 0.88 + 0.50 + 0.70 + 0.80)]$$
$$= 0.7 \text{ x } 5.26$$
$$= 3.682$$

$$\text{Free Cells Ratio} = \frac{\sum(m_{x,y}, \text{free}, \text{true})}{\sum(n_{x,y}, \text{free})}$$
$$= \frac{17.766}{19.269}$$
$$= 0.9220$$

$$\text{Occupied Cells Ratio} = \frac{\sum(m_{x,y}, \text{occ}, \text{true})}{\sum(n_{x,y}, \text{occ})}$$
$$= \frac{3.682}{2.513}$$
$$= 1.4652$$

Fitness Score, f(m, n):

$$f(m,n) = 1 - \frac{\Sigma_{m_{x,y} \in m} |n_{x,y} - m_{x,y}|}{N}$$

$$f(m,n) = 1 - \frac{\Sigma \left| \begin{matrix} (0.75 - 0.70) + (0.80 - 0.80) + (0.75 - 0.70) + (0.78 - 0.78) + (0.76 - 0.76) + \\ (0.80 - 0.70) + (0.98 - 0.98) + (0.96 - 0.96) + (0.90 - 0.80) + (0.90 - 0.90) + \\ (0.80 - 0.80) + (0.95 - 0.95) + (0.85 - 0.85) + (0.98 - 0.98) + (0.80 - 0.80) + \\ (0.80 - 0.50) + (0.95 - 0.90) + (0.90 - 0.90) + (1.00 - 1.00) + (0.90 - 0.90) + \\ (0.80 - 0.20) + (0.90 - 0.88) + (0.80 - 0.50) + (0.80 - 0.70) + (0.80 - 0.80) \end{matrix} \right|}{25}$$

$$f(m,n) = 1 - \frac{1.67}{25}$$
$$f(m,n) = 1 - 0.0668$$
$$f(m,n) = 0.9332$$