PERFORMANCE ANALYSIS OF LOCALIZATION ALGORITHM FOR MOBILE ROBOT.

NUR NABILA HUSNA BINTI MOHD SAHABUDDIN

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

PERFORMANCE ANALYSIS OF LOCALIZATION ALGORITHM FOR MOBILE ROBOT.

NUR NABILA HUSNA BINTI MOHD SAHABUDDIN

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

2020

DECLARATION

I declare that this report entitled "The Performance Analysis Of Localization Algorithm For Mobile Robot" is the result of my own work except for quotes as cited in the references.

Signature	:	
Author	:	Nur Nabila Husna Binti Mohd Sahabuudin
Date	:	26 Jun 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. Nor Hidayah Binti Mohamad Yatim
Date	:	26 Jun 2020

DEDICATION

This thesis is dedicated to my parents who have enriched and motivated my spirit through their eternal love, encouragement, and motivation to pursue and complete this research. Thanks to you for encouraging me to succeed and instilling in me the confidence that I am competent of doing anything.

ABSTRACT

The ability to navigate the mobile robots in the human environment is a need in order to keep a safe and reliable way to overcome the loss of the mobile robot position. To solve the localisation problem for a mobile robot, the Particle Filter algorithm is used to achieve the required robustness and accuracy. Particle Filter is a technique from estimation theory that used a set number of particles in order to detect the position of the mobile robot. There are two types of results that have been analysed in this paper. First is the Particle Filter localization algorithm will be simulated and the performance of the mobile robot will be investigated based on the accuracy of the robot's state. C++ coding is used to run the simulation of the approaches of the Particle Filters. The performance of the algorithm using a set number of particles as the parameters in which will be varied in order to determine the accuracy of the localisation method. It is important investigate the effect of the number of particles used on accuracy of robot's state.

ABSTRAK

Keupayaan untuk menavigasi robot mudah alih di persekitaran manusia adalah keperluan untuk memastikan cara yang selamat dan boleh dipercayai untuk mengatasi kehilangan kedudukan robot mudah alih. Untuk menyelesaikan masalah penempatan untuk robot bergerak, algoritma Penapis Partikel digunakan untuk mencapai kekukuhan dan ketepatan yang diperlukan. Penapis Partikel adalah teknik dari teori anggaran yang menggunakan bilangan zarah set untuk mengesan kedudukan robot mudah alih. Terdapat dua jenis hasil yang telah dianalisis dalam karya ini. Pertama, algoritma lokalisasi penapis zarah akan disimulasikan dan prestasi robot bergerak akan disiasat berdasarkan ketepatan keadaan robot. Pengekodan C ++ digunakan untuk menjalankan simulasi pendekatan penapis zarah. Prestasi algoritma menggunakan bilangan set zarah sebagai parameter yang akan diubah untuk menentukan ketepatan kaedah penyetempatan. Hal ini penting untuk menyiasat kesan bilangan zarah yang digunakan pada ketepatan keadaan robot.

ACKNOWLEDGEMENTS

Firstly, I am very grateful to Allah for the good health and wellbeing while completing the project. I have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals. I would like to extend my sincere thanks to all of them. Millions of thanks are given Dr. Nor Hidayah Binti Mohamad Yatim, a dedicated lecturer who patiently guide me to complete this project. Her time, guidance, and constant supervision as well as for providing necessary information regarding the project, also for her support in completing this subject in this semester and who never tired giving me guidance on ROS and Ubuntu software, support and let me explore the app beyond my expectations is appreciated.

I am very grateful to have classmates that are helpful to each other without a judge and blames. Their care and devotion to share their knowledge is something that I appreciate and would surely treasure. I would like to thank my parents for encouragement, support. My gratitude also goes to all lecturers in the Faculty of Electronics and Computer Engineering (FKEKK) for sharing their pearls of wisdom whilst completing this project. Finally, I would like to place my sense of gratitude to all, who directly or indirectly, have lent their hand in completing the project. I hope that the experiences gained for completing the project will be beneficial to everyone.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AMCL	:	Adaptive Monte Carlo Localization
AMRs	:	Autonomous Mobile Robots
EKF	:	Extended Kalman Filter
LDS	:	Laser Distance Sensor
ROS	:	Robot Operating Software

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background of Project

Nowadays, the use of the mobile robot in the routine life has obtained momentum. In the time ahead, the number of robots usage will be a demand and increased outstandingly [1]. For example, most of the mobile robots are applied to do a variety of works, especially in manufacturing industries. The application of mobile robots can be found in various scenes, either in harmful places or safe places, such as in the store and warehouse. The sizes are different depending on the implementation and the role [2]. There are many types of the autonomous robot were created since they were used widely nowadays especially in industrial and manufacturing fields.

For example, Automated Guided Vehicle (AGV) is one of the autonomous robots that is widely used in manufacturing industries. In the robotic system, the autonomous system can be divided into four types. There are:

i. Programmable Automatic Robot.
ii. Non-Programmable Automatic Robot.
iii. Adaptive Robot.
iv. Intelligence Robot.

Even though there are differences in types of robots, but the navigation system is a very important component in every autonomous system such as a mobile robot. Because of that, the ability to navigate the mobile robots in the human environment is a need in order to keep a safe and reliable way. Besides, in order to be able to localize, the mobile robot must have the ability to track the path and trajectories in the environment.

Due to this reason, a project entitled The Performance Analysis of Localization Algorithm for Mobile Robot was chosen as the Final Year Project (FYP) to analyze the localization algorithm for a mobile robot. Localization is the detection of the present coordinates of the robot. In the current mobile robot applications, there are several problems arise. One of the problems is there was a limit in order to track the location and discover the current position of the mobile robot in the absence of GPS connection. To optimize the localization algorithm, particle filter method is used to solve the problem. The particle filter uses particles as the representation of the robot's belief. By generating a ton of hypotheses, each particle is the hypothesis where the robot might be.

To overcome the problems arise, Ubuntu and Robot Operating System (ROS) software will be used in this project. The function of Ubuntu software is to support the Robot Operating System (ROS) software while ROS software is a need to create robotic simulation as it has the software libraries and tools to help to construct robot applications. In this project, the simulation of the localization algorithm for mobile robot helps to track the robot especially when the robot is lost of the track. At the end of the project, the understanding of the localization algorithm of the mobile robot can be developed. Besides, the performance of the localization algorithm based on the robot's position can be evaluated.

1.2 Problem Statement

Several problems arise in this project. One of the problems is there is a limitation in order to track the location of the mobile robot. Usually, a system that used to track the location of the mobile robot must have a GPS connection. Because of that, if the mobile robot located in the area with no GPS connection, they are unable to discover the current location of the mobile robot. Thus, there is a limitation where the connection of GPS is needed in order to track the location of the mobile robot in a certain indoor area. Besides, it will be difficult to discover the coordinate of the current position of the mobile robot. They will be facing difficulties to detect the current location of the mobile robot if there suddenly a lost connection of GPS or the building system having some connection problem. Thus, the current location of the mobile robot cannot be detected due to the situation. Another problem is lost from the track or the usual measurements of the route of the mobile robot. When the situation is happening, it will be difficult to find the location of the mobile robot as we will probably do not know where the mobile robot will go. To overcome these problems, the student will develop the localization algorithm that does not depends on GPS but by using the sensor measurements and map.

1.3 Objectives

There are two objectives listed for this project;

i. To simulate the particle filter localization algorithm and investigate the performance based on the accuracy of the robot's state.

ii. To investigate the effect of the number of particles used on the accuracy of the robot's state.

1.4 Scope of Work

This project focuses on the development of software that can track the location of the robot. The software used in this project Ubuntu and Robot Operating System (ROS) software. ROS is a framework for building a Robotics Application. Ubuntu is a software where the coding is created and then will sync with the ROS to perform the system. For this project, language programming used is C++.

Besides, this project also focuses on the indoor environment as it is easy to apply the localization algorithm. The indoor environment was chosen because it has a fixed area or environment compared to the outdoor environment. Other than that, this project will be simulated or applied in a static environment since it has less noise in the environment. In the ROS software, among the various types of robots, TurtleBot3 will be used as the mobile robot for this project. This is because TurtleBot3 is suitable as the project only focuses on the indoor environment. The robot itself not suitable for an outdoor environment.

There are several types of robotic systems which are a single robot and multi-robot. Therefore, the project only emphasizes on single robotic rather than multi-robotic. It is because of the characteristics of single-robot that can localize independently and a self-adaptive sample is used in the localization algorithm to overcome the position tracking, global localization, and kidnapped robot problem. Different from the multi-robot, it is cooperative localization which enables robots to exchange and share information through communication.

Besides, the sensor is one of the requirements as the tools in order to perform the activities of tracking or detection for the mobile robot in this project. Therefore, a laser range sensor is used as it is suitable to detect and track the location in the indoor environment compared to other sensors such as vision sensors that only used to monitor path planning and obstacle detection of the mobile robot while moving in the unknown environment.

1.5 Thesis Outline

The body of the contents in this thesis is divided into five chapters which are the introduction, background study, methodology, results and discussion, and conclusion and future works. References, list of publications and papers presented, and appendices were listed at the end of the report sequentially.

The first chapter briefly describes the background of the research work. Besides, this chapter also shows the problem statement, objectives and scope of this project.

The second chapter covers an important literature review related to this project. This chapter starts with the studies about the mobile robot, localization, and navigation, types of localization algorithm, the comparison between the algorithm used and Robotic Operating Software (ROS).

The third chapter is about the methodology of the project. In this chapter, each method used like Robotic Operating Software and other process flow are shown.

The fourth chapter is the results and discussions. This chapter will be analyzing the data based on the given scope and discussion about the result that observed. Every simulation data and result achieved were listed in this part.

The last chapter of this thesis is about the future work recommendation and conclusion about the overall achievement of this project.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Nowadays, the application of the mobile robot is widely implemented in a variety of fields, especially in industry and manufacturing fields. Thus, localization and navigation also play an important task to assure the movement of the mobile robot under control. Hence, several types of localization problems of the mobile robot will be explained in this chapter.

Besides, the types of localization algorithms that can be used to overcome the localization problems will be listed and explained in this chapter. Previous researches that have used the different types of method to overcome or improve the localization of the mobile robot will be compared to the overview of the benefits and limitations faced in the projects.

Lastly, the main tools that will be used in this project which is Robot Operating Software (ROS) will be described in the last part of this chapter. The explanation of the function and the requirement of the ROS application also will be listed.

2.2 Navigation and Localization

As the use of the mobile robot is increasing rapidly nowadays, the implementation of navigation and localization system in the mobile robot also play an important task [1]. It is a necessary task in the navigation of the field of mobile robotics that will make the mobile robot able to drive in an unsure environment. The accuracy of localization has a direct effect on the accuracy of navigation. With the high accuracy of localization, a mobile robot is able to perform the assigned task faster [2].

The navigation can be classified into two types; global navigation and local navigation. The previous knowledge of the environment should be available for global navigation. On the other hand, for the local navigation, by using provided sensors, the robot can decide or control the movement freely. Various types of algorithms can be used to overcome the local navigation problem [3].

2.3 Problems in Mobile Robot Localization

Despite the benefits gained from mobile robots, there are several problems arise in different flavors. One of the problems is the mobile robot localization problem. The mobile robot localization problem can be listed in three types. There are position tracking, global localization and kidnapped robot problem [4].

2.3.1 Position Tracking

The position tracking problem is the simplest localization problem that needs to correct the incremental errors in a robot’s odometry as the initial position of the mobile robot is recognized. It is updated dependent on the information on its past position which is detecting or tracking. The robot introductory area is required in this case. When if the uncertainly of a robot present is excessively enormous, then there is more failure to localize the robot. [4]

2.3.2 Global Localization

Different from the position tracking problem, the global localization has no information about the initial location of the mobile robot but it has to find if from scratch [5]. This implies that without the information or knowledge, the robot can move to anywhere and can localize globally within it [4].

2.3.3 Kidnapped Robot

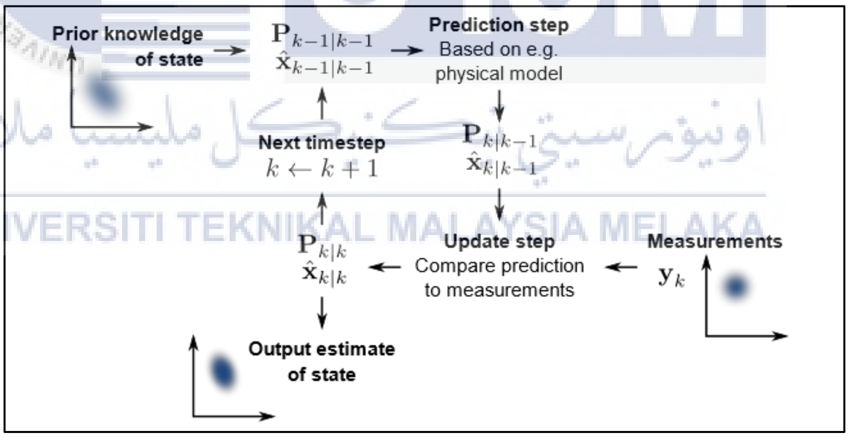
Next, the localization problem is the kidnapped robot problem. This problem is more challenging than others because the well-localized robot is teleported to another place without being informed [7]. The kidnapped robot problem is similar to global localization but the worst thing is the robot itself does not know where it has been moved and it believes it knows where it is but in fact does not. [6]

2.4 Types of Localization Algorithm

Based on the problems of mobile robots that were listed before, there is a reason to find the solution to overcome it. Because of that, there are several localization algorithms that can be used to solve the problems. In order to overcome the problem of localization for mobile robots, there are several types of localization algorithms that can be applied. There are Kalman Filter algorithm, Extended Kalman Filter algorithm (EKF) and Particle Filter algorithm.

2.4.1 Kalman Filter Algorithm

The function of the Kalman filter is to keep track of the approximated state of the system and the unsureness of the approximation. The application of Kalman filter is generally applied for guidance, navigation, and control of vehicles, particularly aircraft, spacecraft and dynamically positioned ships. Besides, the Kalman filter also used for modeling the control of the central nervous system of movement. Kalman filter helps a realistic model for forming an approximation of the current state of the motor system and problem updated commands. Figure 2.3 below shows the two-step process involved in Kalman Filter Algorithm.



Prior knowledge of state
$\mathbf{P}_{k-1|k-1}$ $\hat{\mathbf{x}}_{k-1|k-1}$
Prediction step
Based on e.g. physical model
Next timestep
$k \leftarrow k+1$
$\mathbf{P}_{k|k-1}$ $\hat{\mathbf{x}}_{k|k-1}$
$\mathbf{P}_{k|k}$ $\hat{\mathbf{x}}_{k|k}$
Update step
Compare prediction to measurements
Measurements
$\mathbf{y}_k$
Output estimate of state

Figure 2.1 : Two-step process in Kalman Filter algorithm.

By referring in Figure 2.3 above, it shows that there is a two-step procedure used in this type of algorithm. As shown above, estimates of the initial state of variables are constructed by the Kalman filter. The estimation gained by the observation of the outcome will be updated using the mean of the weight. Kalman filter algorithm is recursive that can run in real-time by using only the present input measurements and past calculated state and its unsureness or covariance matrix.

2.4.2 Extended Kalman Filter (EKF) Algorithm

The Extended Kalman Filter algorithm can be described as the estimator algorithm that can be used when the system is evaluated by a nonlinear function. One of the applications of EKF is to combine the approximate data from the odometer and landmark to enable the mobile robot to localize itself accurately. Besides, it is used to evaluate the state and apply the data of the system and sensor dynamics, the system and measurement noises, and any obtainable data for the initial values of the state. Basically, the EKF is used to keep the estimated position from drifting [6].

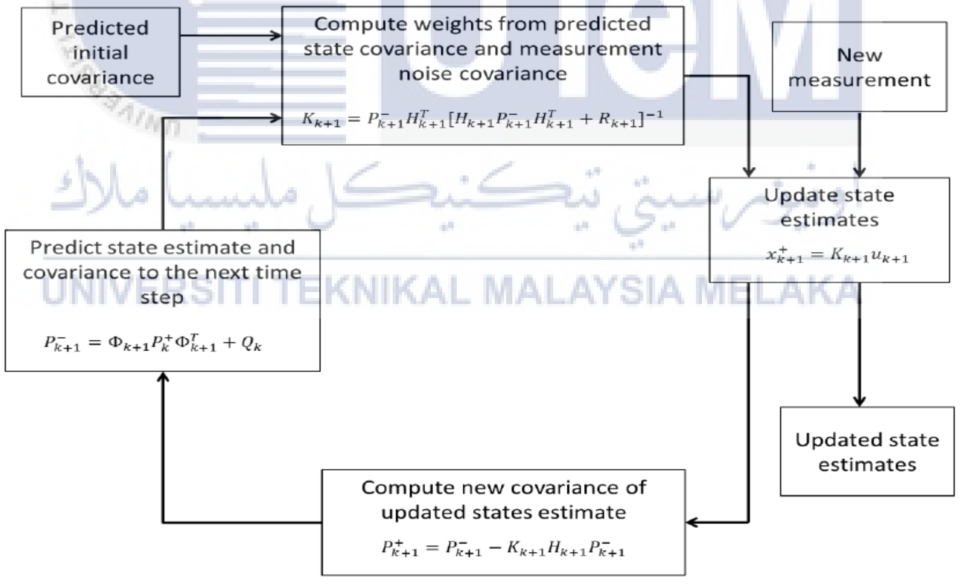

Predicted initial covariance
Compute weights from predicted state covariance and measurement noise covariance
$K_{k+1} = P_{k+1}^{-} H_{k+1}^{T} [H_{k+1} P_{k+1}^{-} H_{k+1}^{T} + R_{k+1}]^{-1}$
New measurement
Update state estimates
$x_{k+1}^{+} = K_{k+1} u_{k+1}$
Predict state estimate and covariance to the next time step
$P_{k+1}^{-} = \Phi_{k+1} P_{k}^{+} \Phi_{k+1}^{T} + Q_{k}$
Updated state estimates
Compute new covariance of updated states estimate
$P_{k+1}^{+} = P_{k+1}^{-} - K_{k+1} H_{k+1} P_{k+1}^{-}$

Figure 2.2 : Extended Kalman Filter algorithm.

Based on Figure 2.2 above, it shows that prediction and update algorithm is required for EKF. For the prediction algorithm, it will estimate the state vector and simulates the corresponding covariance matrix from the initial period to the next one using the state transition matrix presenting the process described. Then, the equation of the update algorithm will correct the state vector that was assumed. Besides, the corresponding covariance matrix also will be corrected by using the measurement model as shown in Figure 2.2 above [12].

2.4.3 Particle Filter Algorithm

The particle filter approach depends on the Monte Carlo algorithm. In this algorithm, particle filtering is implemented to track the state factors of a system with non-Gaussian. Besides, it can be multi-modal probability distribution functions [7].

In the Monte Carlo algorithm, it shows the distribution of a variant through a large number of samples. Particle filters can be categorized as Monte Carlo algorithms that are used to evaluate the states in partially observable Markov chains. Monte Carlo algorithms represent the distribution of a random variable through a large number of samples such as a set of random particles [8]. Figure 2.3 below shows the prediction and measurement updates in Particle Filter.

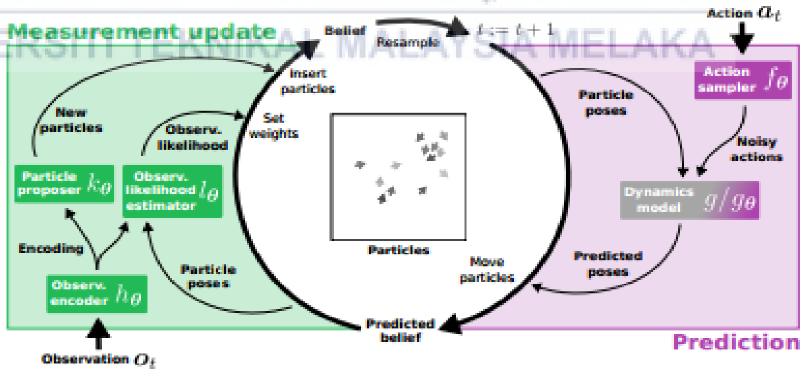


Figure 2.3 : The prediction and measurement updates in the particle filter.

The condition of the framework through a dataset made out of random samples which called the particle is represented in the particle filter. Three alternating steps can be applied in the particle filter. First is the prediction step where a new particle of each particle will be drawn based on the movement model given the action. The correction step is another step where an integration for a new observation will be created by allocating new weights to each particle based on the sensor model. The third step is the resampling step where a new generation of the particle will be constructed [10].

2.4.4 Comparison Between Kalman Filter, Extended Kalman Filter (EKF), and Particle Filter Algorithms.

Every type of localization algorithms such as Particle Filter, EKF, and Kalman Filter algorithms, they have their benefits and the drawbacks. Therefore, in Table 1 shows below, all the advantages and disadvantages were listed according to each algorithm.

Table 1 : Comparison between three types of localization algorithm.

Method	Advantages	Disadvantages
Kalman Filters	• Enable them to keep track of multiple hypotheses [9].	• Poorly robust if the initial hypothesis is bad or noise assumptions fail [9].
Extended Kalman Filter (EKF)	• It solves the nonlinear equations while providing optimal denoising [10]. • Its fast recursive algorithm needs a small memory and is thus low-cost [10]. • It practically does not demonstrate divergence under the normal conditions of triangulation [10].	• Biased estimates, because noise is nonadditive in the robot dynamics formulation [10]. • Divergence under the conditions of large nonlinearities and large noise [10]. • High sensitivity to noise statistics [10]. • Large errors under the industrial uncertainties [10].

Table 1 (continued).

Particle Filter	• Able to solve applications in the robotics domain for more than one reason [8]. • A fixed computational time is not necessary but the increase of computational resources can improve accuracy [8]. • Statistical robustness [9]. • Able to control the random noise distribution and non-linearities in the system and observation model [9].	• Bad implementation in higher dimension spaces [11].

As the details listed in Table 1, it shows the Particle Filter has more benefits and the advantages compare to EKF and Kalman Filter. Therefore, in this project, the Particle Filter algorithm is chosen as the localization algorithm for the mobile robots because it does not require a fixed computational time rather their accuracy increases with an increase in computational resources. Besides, it can manage the noise distributions and non-linearities in the system. Compare to another algorithm, Particle Filter can work for the high dimensional system. This is because it is independent of the size of the system [9],[13].

2.5 Development of Technology

In the robotics field, other researches have done much research or project to improve the system in localization, especially for mobile robots. Earlier research used particle filter algorithm [11], [12], [13] for their research. Table 2 below shows the comparison between the researches using the particle filter.

Table 2 : The comparison between the researches that using the particle filter.

Author / Date	Focus	Descriptions	Limitations / Gaps / Future Works
N. N. Bhat (2012)	To improvise the particle filter algorithm using the SIFT pattern recognition technique and image database processing for effective position tracking.	• [11] implemented a particle filter using a visual database that is more relevant in real-time environments.	• The particular database images must be stored and chose for SIFT recognition and Particle Filter analysis. • Have a high computation time.
V. Raudonis, R. Simutis, and A. Paulauskaite-Tarasevičiene (2009)	To track the non-rigid objects using dynamic particle filter.	• [12] used the dynamic particle filter. • This method changes dynamically the number of particles.	• Includes the topological information, detection of unexpected appearing objects and multiple related object tracking.
C. C. Hsu, S. S. Yeh, and P. Lo Hsu (2016)	To improve the accuracy of (RSSI)-based localization algorithms for localizing mobile robots that moving by using particle filter design.	• [13] used sensor fusion to improve the localization accuracy. • Used the trilateration localization method with the particle filter design and the RSSI measurements from the wireless communication.	• The accuracy of localization is limited in practical.

Based on Table 2 above, according to N. N. Bhat (2012), the research was focused on improvising the particle filter algorithm using the SIFT pattern recognition technique and image database processing for effective position tracking. In the research, the particle filter using a visual database that is more relevant in real-time environments was implemented [11]**.** But, in this research, in order to perform recognition tasks in a given map, particular database images must be stored and chose for SIFT recognition and Particle Filter analysis. Thus, the computation time is necessary to match enormous database images.

Next, tracking the non-rigid objects using dynamic particle filter is the main focus in the research by V. Raudonis, R. Simutis, and A. Paulauskaite-Taraseviĉiene (2009). In their research, the dynamic particle filter that enables the tracking of the non-rigid object in the complex background was used. The formalizes algorithm enables us to define the dynamics logically and mathematically. This method changes dynamically the number of particles. If fewer particles are used then the real position of the observable object can be found with a certain error. However, the limitation of this project is the integration with the extension of the algorithm can be performed for automatic selection of the template histogram that includes the topological information, detection of unexpected appearing objects and multiple related object tracking.

According to C. C. Hsu, S. S. Yeh, and P. Lo Hsu (2016), the research was focused on improving the accuracy of (RSSI)-based localization algorithms for localizing mobile robots that moving by using particle filter design. Besides, it develops a particle filter design with the fusion of the current states of the mobile robot to improve localization accuracy. [13] used sensor fusion to improve the localization accuracy by employing particle filter design. through sensor fusion. The trilateration localization method with the particle filter design and the RSSI measurements from wireless communication indicates that the proposed approach for the localization of mobile robots is feasible and provides satisfactory results. But still, the accuracy of localization is limited in practical applications as there was interference and reflection of transmission signals during wireless signal transmission.

Based on the limitations and gaps that occurred during the previous researches, this project was chosen to overcome the limitations and gaps. This project will be used only the laser range sensor measurement in order to detect the location of the mobile robot rather than the particular database images that must be stored in the system [11]. Hence, the computation time can be reduced compared to previous research. Besides, to avoid the use of the integration with the extension of the algorithm, this project will be focused on the static and indoor environment. This is to overcome the noise that appeared due to the detection of unexpected appearing objects and multiple related object tracking [12]. Moreover, this project will not use the wireless signal transmission but only by the laser range sensor measurement in order to do the localization of the mobile robot. This helps to overcome the limitation of the accuracy of localization in practical applications as there were interference and reflection of transmission signals during wireless signal transmission [13].

2.6 Robot Operating Software (ROS)

The Robot Operating Software (ROS) is an open-source framework for robotic that have been practically applied and used around the world. There is a variety of types of research that have been presented involved in the robotic by using ROS. It provides the service user that would predict from an operating system. Besides, tools and libraries for gaining, designing, and running code across manifold platforms also provided in the ROS. ROS also gives an incorporated foundation to robot simulation. The robot model alongside the joined sensors can be effectively coordinated with the gazebo test system in ROS to create a correspondence string between the user and test system comparable to the one between a real robot and user [14].

Several controller sensors, tools for the navigation system, mapping for the environment, path planning, and a 3D visualization system are included in the packages of the ROS. Several sensor controllers, routing software, environmental modeling, route preparation, interprocess collaboration simulation tools, and a 3D visualization system tool are included in the ROS kits. The ROS enables productive creation of a modern robotic device and a high-performance robotic control program is reduced considerably when used in combination with a simulation middleware such as Gazebo [15].

Figure 2.4 : Robot Operating Software (ROS) logo.

Besides, ROS is the role of a distributed framework of nodes or processes. It can practicable to be individually designed and flexible at runtime. Furthermore, in ROS, several main components will be used to operate the robotic simulation in ROS software. The several main components are node, package, message, and topic.

2.6.1 Node

The smallest processor unit running in ROS is a node. Note this as a software executable. For each purpose, ROS recommends the creation of one single node and is recommended for easy reusability. The robot operating system, for example, is divided into special functions for mobile robots. For each function, a special node such as sensor drive, data sensor, obstacle recognition, motor drive, input encoder and navigation is employed. A node registers information such as name, message type, URI and node port number when it is launched that can act as publisher, subscriber, service server or service client, and nodes can exchange information through subjects and services.

2.6.2 Package

The simple ROS system is the package The package-based ROS software includes either a setup directory for launching other programs or nodes. It also contains all the package files, including ROS dependency libraries, which allow different processes, data sets and configuration files to be performed. In fact, there are approximately 4,600 users packages established and released nowadays.

2.6.3 Message

A node transfers or collects information through a message between nodes. Messages are parameters like integer, boolean, and floating points. The messages can use a nested message layout including individual messages or a set of messages. The messaging protocol TCPROS and UDPROS is used for transmission of messages. The topic is used in unidirectionally delivering messages while the service is used to deliver two-way messages involving request and response.

2.6.4 Topic

The topic is basically like a topic in a dialogue. The node of the publisher first registers its subject with the master, then begins to submit a message on a topic. Subscriber nodes wishing to receive the publisher's node topic request information corresponding to the topic name registered in master. The subscriber node connects directly to the publisher node on the basis of this information to exchange messages as a topic.

CHAPTER 3

PROJECT METHODOLOGY

3.1 Introduction

To ascertain the project runs successfully as the plan, the methodology part is very critical. All the steps and procedures must be done by sequence full of discipline. The methods that were implemented in this project are very significant to tackle the project comprehensively and systematically and hence, produce expected results.

In the first part of this project, the study and findings of previous researches about the localization algorithm will be implemented. Followed by the installation of the Ubuntu and Robot Operating System (ROS) software that will be done and the test environment in ROS will be developed. The generating of the environment in ROS will be done by configuring the ROS software and creating the workspace. Besides, the installation of packages for Turtlebot also will be done.

Next, the parameters in the particle filter algorithm will be studied. Then, the simulation of the localization algorithm will be done and the performance of the system will be evaluated. The accuracy of the robot's position will be determined either it is acceptable or not during the process of simulation. If the accuracy is not acceptable, then the relevant parameters of the algorithm will be modified until the accuracy is accepted. After that, the project will be continued with the analysis of the simulation of the localization algorithm. Figure 3.1 shows the flowchart process for this project.

In this chapter, all methods and procedures used in this project have been described and arranged accordingly from the installation and development of Robot Operating Software (ROS) to the simulation of the localization algorithm of the mobile robot. In order to achieve the best result for this project, the methodology must be pursued systematically.

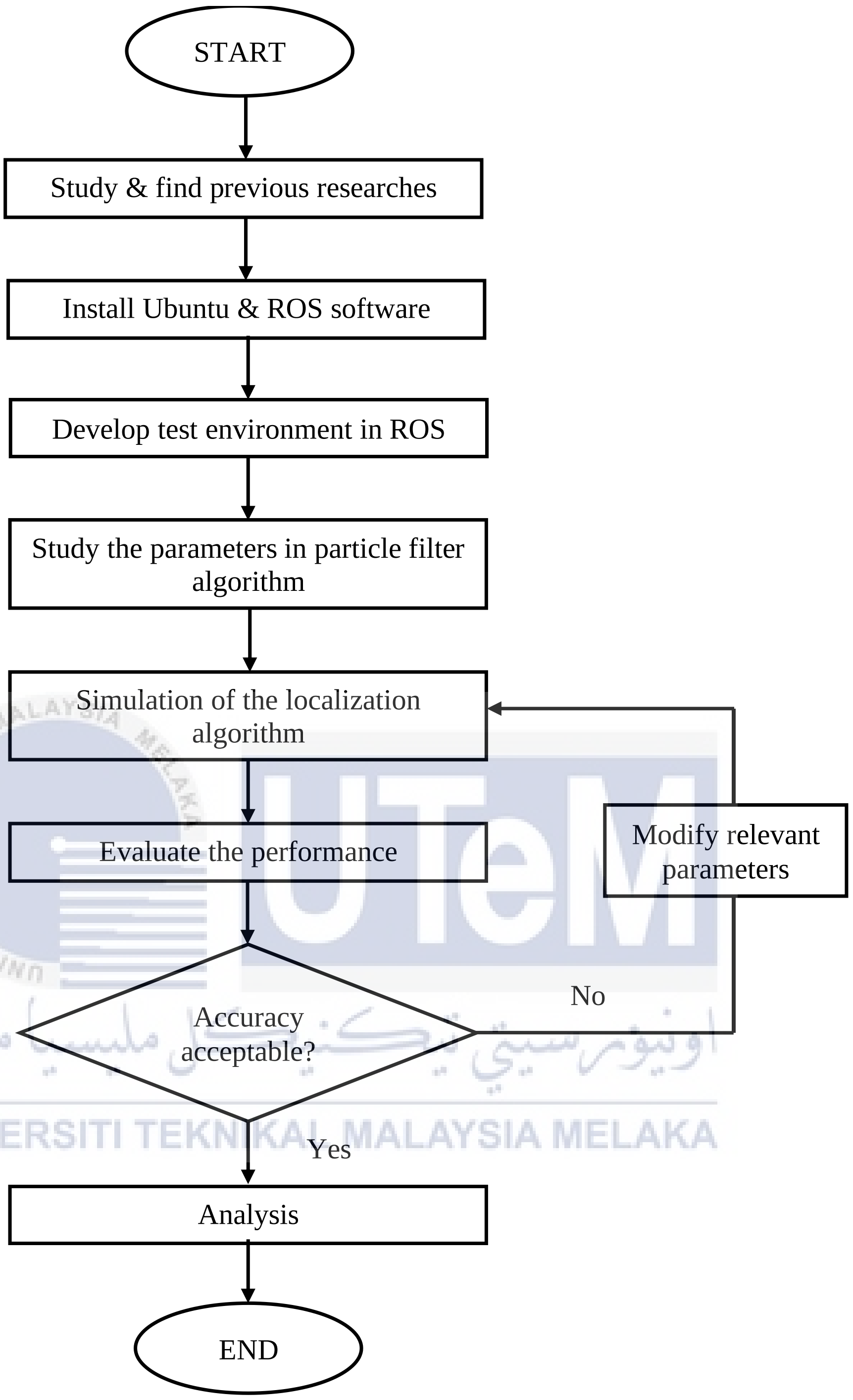

START
Study & find previous researches
Install Ubuntu & ROS software
Develop test environment in ROS
Study the parameters in particle filter algorithm
Simulation of the localization algorithm
Evaluate the performance
Accuracy acceptable?
No
Modify relevant parameters
Yes
Analysis
END

Figure 3.1 : Flowchart process of the project.

3.2 Robot Operating Software (ROS)

In this project, the software setup is implemented. The process of installing and configuring the operating system and ROS framework are required to develop the environment of the software. The installation of ROS must be done first by installing it on top of Ubuntu software. For Ubuntu software, there are several version of the software. Hence, for this project, Ubuntu 16.0 version is chosen to support ROS software available. Besides, ROS Kinetic is installed and used on the Ubuntu 18.4 version. After installing ROS, the configuration of the ROS environment must be done. The environment of the ROS software can be built with the command in Figure 3.2.

```
echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

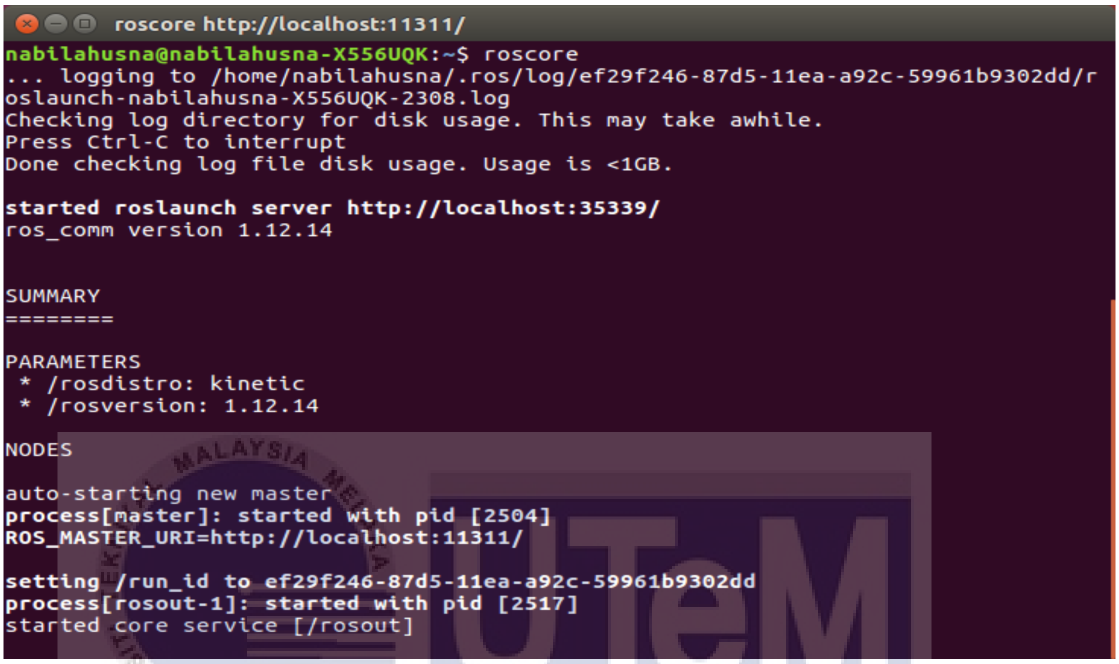
Figure 3.2 : Command for the configuration of ROS environment.

Then, the ROS workspace can be created by building the catkin workspace in the ROS. The catkin workspace is set up with the goal that ROS packages can be worked at this individual workspace and is simpler to oversee. The following command in Figure 3.3 must be run in the terminal provided in the ROS to create catkin workspace:

```
$ mkdir -p ~/catkin_ws/src

$ cd ~/catkin_ws/

$ catkin_make
```

Figure 3.3 : Command to create ROS catkin workspace.

To start up the ROS master, ROS parameter server and a `rosout` log node, the `roscore` command is needed. The `roscore` command is the mandatory support to run for ROS to work. Besides, `roscore` oversees correspondence between nodes that each of it register with the `roscore` during start-up. Figure 3.4 shows the simulation when running the `roscore` in the terminal of ROS.



```
roscore http://localhost:11311/
nabilahusna@nabilahusna-X556UQK:~$ roscore
... logging to /home/nabilahusna/.ros/log/ef29f246-87d5-11ea-a92c-59961b9302dd/r
oslaunch-nabilahusna-X556UQK-2308.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://localhost:35339/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [2504]
ROS_MASTER_URI=http://localhost:11311/

setting /run_id to ef29f246-87d5-11ea-a92c-59961b9302dd
process[rosout-1]: started with pid [2517]
started core service [/rosout]
```

Figure 3.4 : The simulation when running the roscore command.

3.3 TurtleBot3 Installation

TurtleBot is a ROS standard stage robot. In addition, the turtlesim node, which initially appears in the basic tutorial of ROS, is a program that copies the command system of the Logo turtle program. From that point, forward TurtleBot has become the standard foundation of ROS, which is the most famous stage among designers and understudies. There are several series of TurtleBot version. But for this project, TurtleBot3 was chosen. This is because it is a small, affordable, programmable, ROS-based portable robot for use in training, research, leisure activity, and item prototyping.

The TurtleBot3 can be redone into different ways relying upon the method for remaking the mechanical parts and utilize discretionary parts, for example, the computer and sensor. In addition, TurtleBot3 is advanced with financially savvy and small-sized SBC that is appropriate for robust embedded systems, 360-degree distance sensors, and 3D printing technology. Besides, TurtleBot3 is reasonable for this project as its one of the core technology navigation.

In the ROS software, after the configuration of the ROS is done, the Turtlebot3 package can be installed. To simulate the Turtlebot in the software, the following command must be run in the terminal in Figure 3.5:

```
$ rosrun turtlesim turtlesim_node
```

Figure 3.5 : Command to simulate the TurtleBot.

Then a window will show up as shown in Figure 3.7 to show the turtle icon where it can move or control by using the following command in Figure 3.6 :

```
$ rosrun turtlesim turtle_teleop key
```

Figure 3.6 : Command to move or control the TurtleBot.

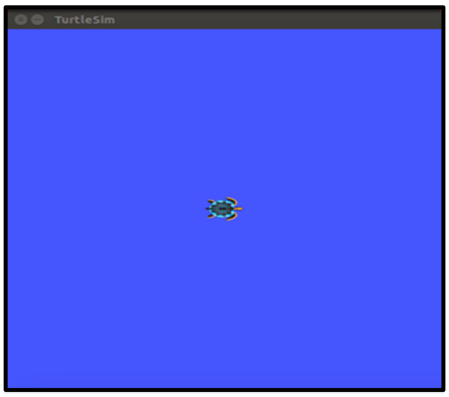


Figure 3.7 : TurtleSim window.

3.4 Gazebo Simulator

The Gazebo is a 3D simulator that delivers robots, sensors, and environmental models needed in the 3D simulation of a robot for its development. It is a well-known robotic simulation, even when it is an open-source due to its high performance. In addition, Open Robotics which is responsible for ROS and its community developed and distributed Gazebo. Therefore, Gazebo is highly compatible with ROS. The ROS and Gazebo-based simulation framework thus benefit from various state-of-the-art robotics algorithms and helpful ROS debugging tools. In terms of code reuse and project co-creation, it may also benefit or support active software communities like ROS and Gazebo [16].

In the Gazebo, libraries for physics simulation, rendering, user interface, communication, and sensor generation are provided in the application. The Gazebo simulator offers the practical sensing input and theoretically possible interaction between the object by counting the exact simulation of the rigid-body physic of the object. Besides, the user can make the Gazebo perform as a node to simulate the robot movement and communicate with other nodes to exchange the data [17].

Figure 3.8 : Gazebo logo.

3.4.1 TurtleBot3 Simulation Using Gazebo

After the installation of ROS and Gazebo have been made, the Gazebo can be launched. By opening the new terminal at the Ubuntu windows, the following command as shown in Figure 3.9 is inserted to launch the Gazebo :

```
$ roslaunch gazebo_ros empty_world.launch
```

Figure 3.9 : Command to launch Gazebo in ROS.

After running the coding above, the Gazebo window will appear. For the project, there are several types of the mobile robot can be chosen in the Gazebo. For example, in this project, TurtleBo3 Burger was chosen for the simulation as shown in Figure 3.10 below.

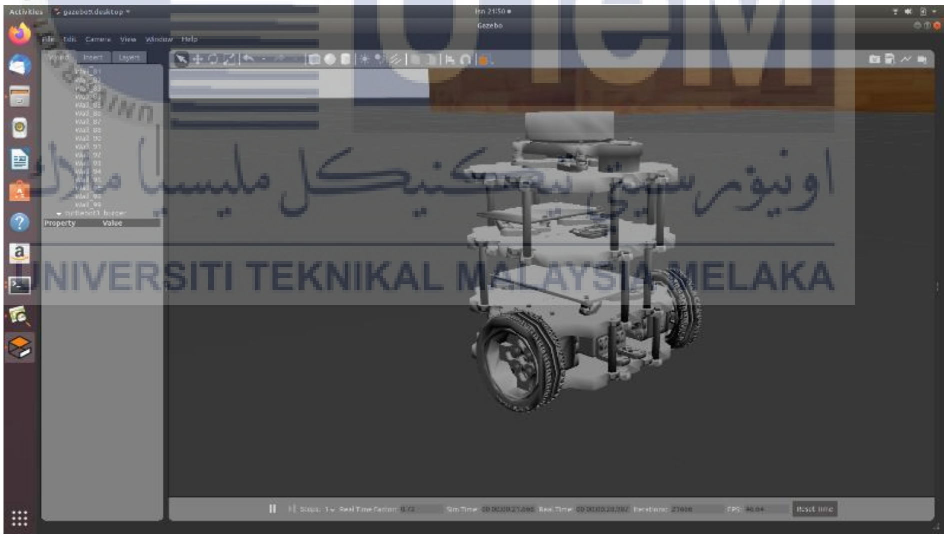


Figure 3.10 : TurtleBot3 Burger in Gazebo simulator.

3.4.2 Development of the Environment in Gazebo Simulation Using Gazebo

In the Gazebo simulator, it provides three-dimensional model importing and an environment editor for the user. Despite of this, there is no mesh editing provided and third-party software is required if any imported 3D model needs to be optimized [18].

As the Gazebo provides the 3D model importing and the scene editor, the user can develop and construct a test environment. For example, the user can build a house or a maze for the simulation of the mobile robot. The size of the test environment can be adjusted and edited by the user. For this project, a test environment as shown in Figure 3.11 was created.

After the test environment was successfully created, the mobile robot used such as TurtleBot3 Burger can be added into the environment. The test environment cannot be edited once the simulation of the project in the Gazebo is started.

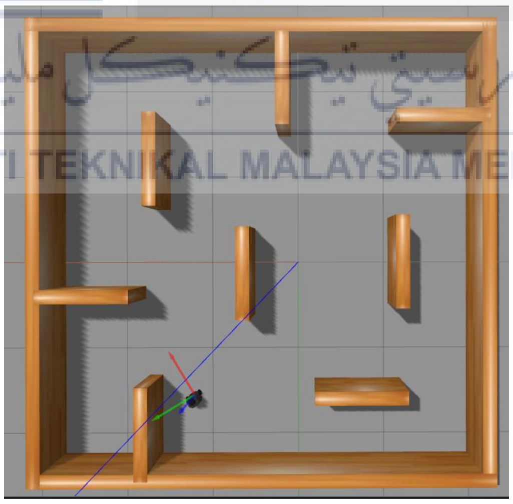


Figure 3.11 : Test environment.

3.5 Laser Distance Sensor (LDS)

To detect or to track the position of a mobile robot, there is a requirement of the use of the sensor. In this project, the sensor used to track the location of the mobile robot is the laser distance sensor (LDS). This is because the sensor itself acceptable and compatible with the indoor environment and it can give a range measurement for the analysis. Besides, the laser distance sensor can be utilized to quantify the geometry of objects in order to acquire the data of the environment and space around. It works by sweeping a laser over a scene and at each point estimating the range and the returned power. Furthermore, the laser distance sensor allows the robot to distinguish the various types and sizes of the obstacle in the surrounding [19].

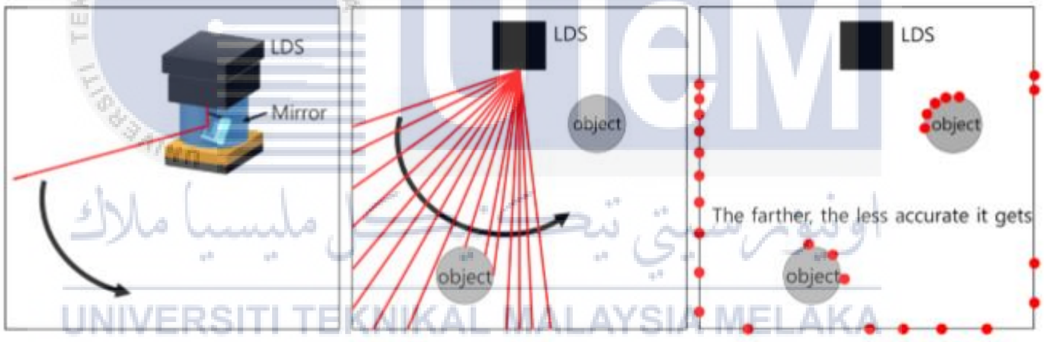


Figure 3.12 : Application of laser distance sensor.

Figure 3.12, shows that a laser distance sensor can detect the object placed around the environment. The accuracy of the laser distance sensor depends on the distance between the sensor and the object. The further the laser distance sensor is, the accuracy of the detection of the object will be decreased.

3.6 Particle Filter Algorithm

In the particle filter algorithm, an estimate variable of interest is obtained by the weighted sum of all the particles. A particle defined as an individual state estimate. It also can be characterized by the state values that decide its area in the arrangement space, for example [x y θ] and a likelihood that demonstrates its probability. The particle filters utilize numerous particles for representing the belief state. Figure 3.13 below shows the algorithm used for the particle filter :

```
1 : $\bar{X}_t = X_t = 0$
2 : $for\ m = 1\ to\ M\ do$
3 :         $sample\ x_t^{[m]} p\left(x_t | u_t, x_{t-1}^{[m]}\right)$
4 :         $w_t^{[m]} = p\left(z_t | x_t^{[m]}\right)$
5 :         $\bar{X}_t = \bar{X}_t < x_t^{[m]}, w_t^{[m]}$
6 : $for\ m = 1\ to\ M\ do$
7 :         $draw\ i\ with\ probability \approx w_t^{[m]}$
8 :         $add\ x_t^{[i]}\ to\ X_t$
9 : $return\ X_t$
```

Figure 3.13: Particle Filter algorithm.

Based on the coding listed as above, it shows that component of $\boldsymbol{\bar{X}_t}$ and $\boldsymbol{X_t}$ represent the state of the robot at time $\boldsymbol{t}$. The number of particles used is represented as M component. Besides, the prediction step of the particle filter algorithm is described as $\boldsymbol{p\left(x_t | u_t, x_{t-1}^{[m]}\right)}$ where component of $\boldsymbol{x_t}$ represent as the current state of the robot and $\boldsymbol{u_t, x_{t-1}^{[m]}}$ represent as the robot state due to the robot command. For update step, it is mentioned in the coding as $\boldsymbol{p}\left(\boldsymbol{z_t}|\boldsymbol{x_t^{[m]}}\right)$. After the particles are weighted, the highest weighted sample or weighted sum of particles can be used as the best-guess state or to get the mean equivalent. Then, the sensor measurement, $\boldsymbol{z_t}$ will be used to detect the robot's state. The flowchart of the particle filter algorithm is shown in Figure 3.14 below. Based on Figure 3.14, it shows that after the weight was assigned to the particles, the number of effective particles will be determined. The resampling process can be processed when the number of effective particles is less than the threshold value[20].

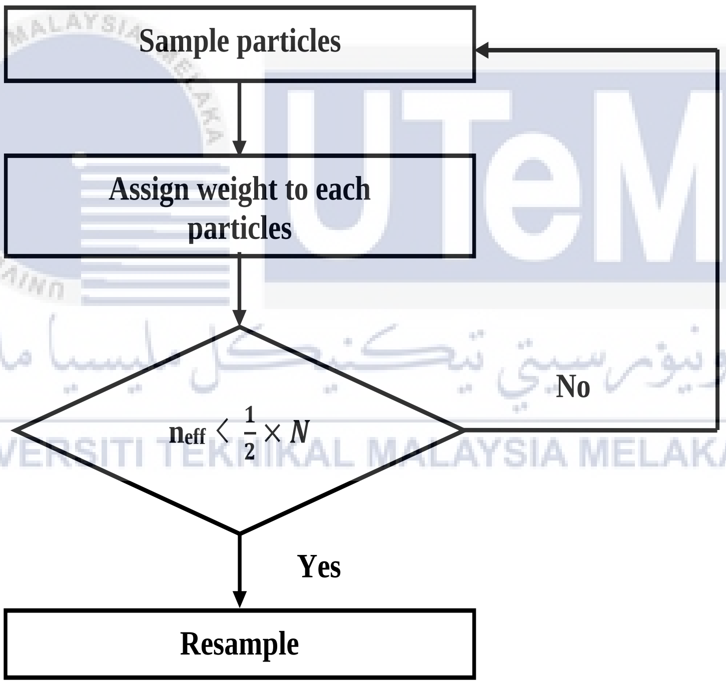

Sample particles
Assign weight to each particles
$n_{eff} < \frac{1}{2} \times N$
No
Yes
Resample

Figure 3.14 : The flowchart of Particle Filter algorithm.

3.6.1 Particle Filtering

The particle filter is repetition in nature and works in two steps. The two steps involve are prediction and update stage. After each activity, inclusive of the addition of random noise in order to simulate the effect of noise on the variable interest, every particle is altered by the existing model during the prediction step. At that point, every particle's weight is reexamined dependent on the most recent sensory data accessible during the update step.

Besides, at times, the particles with small weights are eliminated. The process involves in the elimination of the particles is called a resampling step [21]. The formula used for each step of the particle filter is mentioned as below :

i. **Prediction step that uses the state update model by using the formula:**

$$\boldsymbol{p(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_k|z_{1:k-1})dx_{k-1}} \tag{3.1}$$

ii. **Update step that updates the prior using Bayes' Rule with measurement by using the formula :**

$$p(x_k|z_{1:k}) = \frac{p(x_k|x_{k-1})p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})} \tag{3.2}$$

iii. **Resampling step that implemented when the particle weights degenerate over time by using the formula :**

$$\hat{n}_{eff} = 1/\sum_{i=1}^{n}\left(w_k^i\right)^2 \tag{3.3}$$

$$1 \leq \hat{n}_{eff} \leq n$$

The particle will be resampled whenever $\hat{n}_{eff} \leq n_{thr}$.

3.7 Performance Analysis : Absolute Trajectory Error

The absolute trajectory error (ATE) straightforwardly gauges the distinction between points of the true and the evaluated direction. The ATE also able to evaluate the translational errors and able to detect consequently the rotational errors that usually show in wrong translations [22]. As a pre-processing step, the estimated poses are combined with ground truth presents evaluating the timestamps. In view of this affiliation, the true and the estimated direction utilizing particular value decomposition will be adjusted.

By contrasting the absolute distances between the estimated and the ground truth trajectory, the global consistency of the estimated trajectory can be evaluated. The error between the estimated and the ground truth trajectory is calculated by using an equation in Equation (3.4). From the errors obtained, the cumulative error was calculated as shown in Equation (3.5). Both Equation (3.4) and (3.5) were shown as below:

i. **The error between estimated and the ground truth trajectory calculated by using the formula:**

$$Error = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{3.4}$$

ii. **The cumulative error calculated by using the formula:**

$$Cumulative\ Error = \Sigma\ Error \tag{3.5}$$

CHAPTER 4

RESULTS AND DISCUSSION

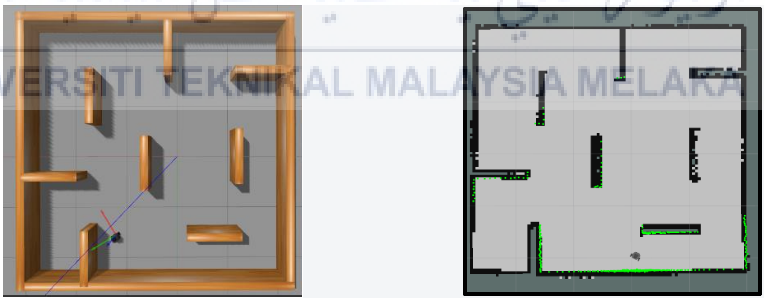
4.1 Introduction

This chapter highlights the result of the simulation of the particle filter localization algorithm. The project aimed to detect the position and coordinate of the mobile robot. Thus, all the simulated results will be shown in this chapter. The main objective of the simulation is to investigate the performance based on the accuracy of the robot's state and the effect of the number of particles used on the accuracy of the robot's state.

4.2 Map of the Test Environment

The map is the first significant element for navigation. An accurate map is provided from the beginning on the navigation system and the modified map can be periodically downloaded so that the map can be matched to the environment. A robot needs a map in the navigation system. Thus, the map has to be created and given to the robot, or the map can be created by the robot.

In robotics, SLAM is developed to allow the robot to create a map with or without the help of a human being. The robot identifies the unknown space and detects its environment and estimates its current location and creates a map. This is a method of creating a map. Thus, a test environment has to be developed first before creating the map as shown in Figure 4.1 (a). Next, a map of the test environment in Figure 4.1 (b) can be created by using the following command in Figure 4.2 :



(a) **(b)**

Figure 4.1 : The (a) test environment in Gazebo simulator and the (b) map created in RViZ simulation.

```
$ export TURTLEBOT3_MODEL=burger

$ roslaunch turtlebot3_slam turtlebot3_slam.launch
slam_methods:=gmapping
```

Figure 4.2 : Command to create the map of the test environment.

Figure 4.1 (b) above shows the map that was created by the simulation. After the map is finished, the map of the test environment can be saved by using the following command in Figure 4.3 :

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

Figure 4.3 : Command to save the map of the test environment.

After the map was saved successfully, the saved map of the test environment can be launch in the RViz simulation to start the particle filter localization simulation of the TurtleBot3.

4.3 Navigation of The TurtleBot3 Burger

After the map is completely created, the navigation of the mobile robot can be simulated. Navigation is to move the robot in a given environment from a starting point to a destination. A map containing information on the geometry of the furniture, items, and walls of the given environment will be needed for this purpose. The map was created using the distance data obtained from the sensor and the robot itself, as described in this last section of the SLAM. For this project, the navigation can be run by using the following command in Figure 4.4 :

```
$ export TURTLEBOT3_MODEL =Burger

$ roslaunch turtlebot3_navigation turtlebot3_
navigation.launch map_file:=$HOME/map.yaml
```

Figure 4.4 : Command to run the navigation of the mobile robot.

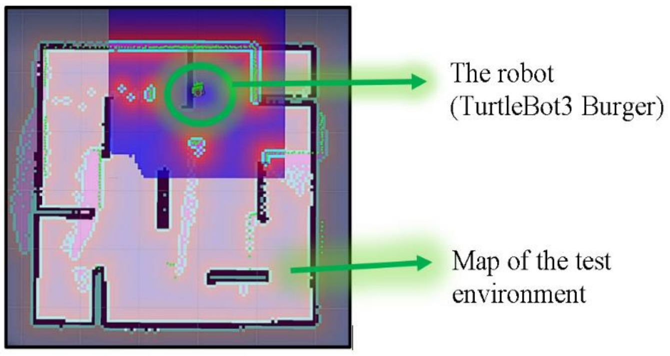


Figure 4.5: Navigation of the mobile robot in the map.

Based on Figure 4.5 shown above, the mobile robot which is TurtleBot3 Burger has moved around on the map. The blue region that was highlighted in the figure shows the area that was scanned by the laser sensor of the mobile robot. Besides, the object that in green color around the mobile robot represent the laser scan used in the simulation.

4.4 Particle Filter Localization on RViz Simulation

At the beginning of the simulation, the TurtleBot3 burger does not know its current location to start with. As a consequence, localization is needed to tells the robot where it is in the environment. Odometry, laser data, and a map are required for the localization [23]. Due to the problem, the Adaptive Monte Carlo Localization (AMCL) node must be launched to enable the particle filter algorithm on the simulation to track the pose of a robot against the saved map. The node uses the particle filter based localization method described in [24]. To run the AMCL launch file, the following command in Figure 4.6 is used :

```
$ rosrun amcl amcl
```

Figure 4.6 : Command to run the AMCL launch file.

Despite the AMCL launch file, there are other nodes that also necessary for the localization. The turtlebot3_gazebo_rviz.launch is one of the launch files. The turtlebot3_gazebo_rviz.launch file will visualize the position of the robot running in the Gazebo and the distance sensor details. This simulation performance is the same as the real robot in the Gazebo environment design.

Besides, provide_map.launch is another launch file that required for the localization. In the simulation of RViz, the file is used to provide the saved map of the test environment as shown in Figure 4.2 before. The command in Figure 4.7 is used to launch these files:

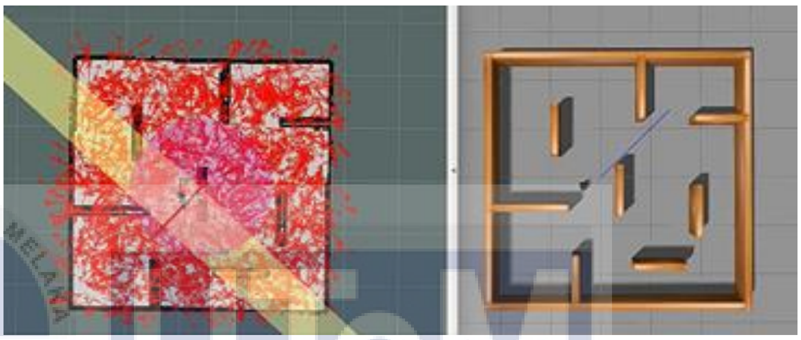
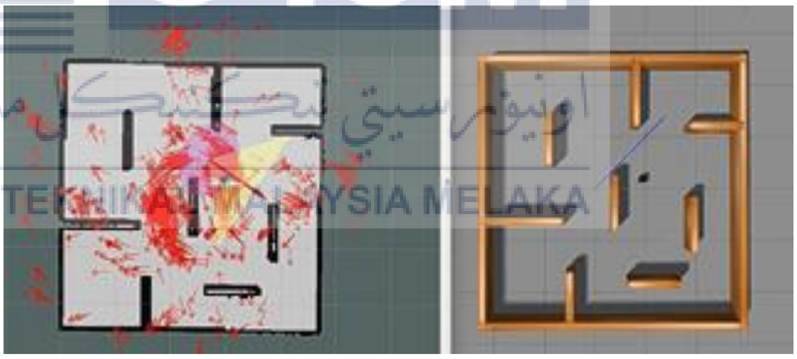
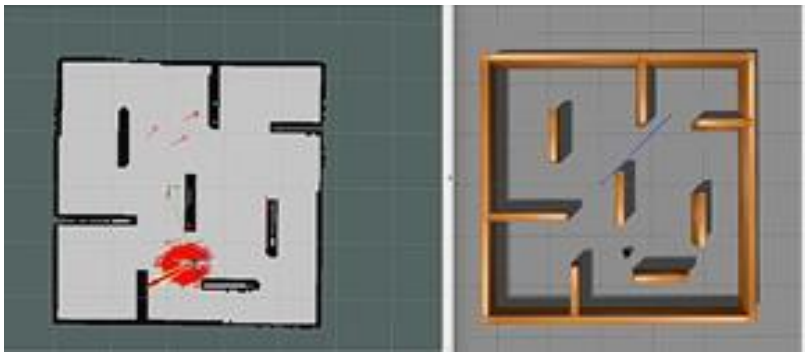
```
$ roslaunch turtlebot3_gazebo
turtlebot3_gazebo_rviz.launch

$ roslaunch provide_map provide_map.launch
```

Figure 4.7 : Command that necessary for the TurtleBot3 Burger robot localization.

Table 3 shows the simulation after the commands Figure 4.6 and 4.7 were launched in the terminal. Thus, the particle filter localization for the robot can be simulated. According to Table 3, three different diagrams represent the condition in each state of the particle filter. There are initial state, intermediate state, and final state of the particle filter.

In Table 3 (a), the TurtleBot3 burger robot is globally uncertain. The particles are also uniformly distributed in the test environment. The location of the robot along with the uncertainty state is shown. Instead, the particles that do not match the sensor measurements are thrown away until the actual position particles remain at the intermediate state of particle filter as shown in Table 3 (b). Now new particles will be added near the surviving particles to find out a changing position when the robot moves. As seen in Table 3 (c), the majority of the particles are now located nearly to the right location and the robot has successfully localized itself eliminating ambiguity.

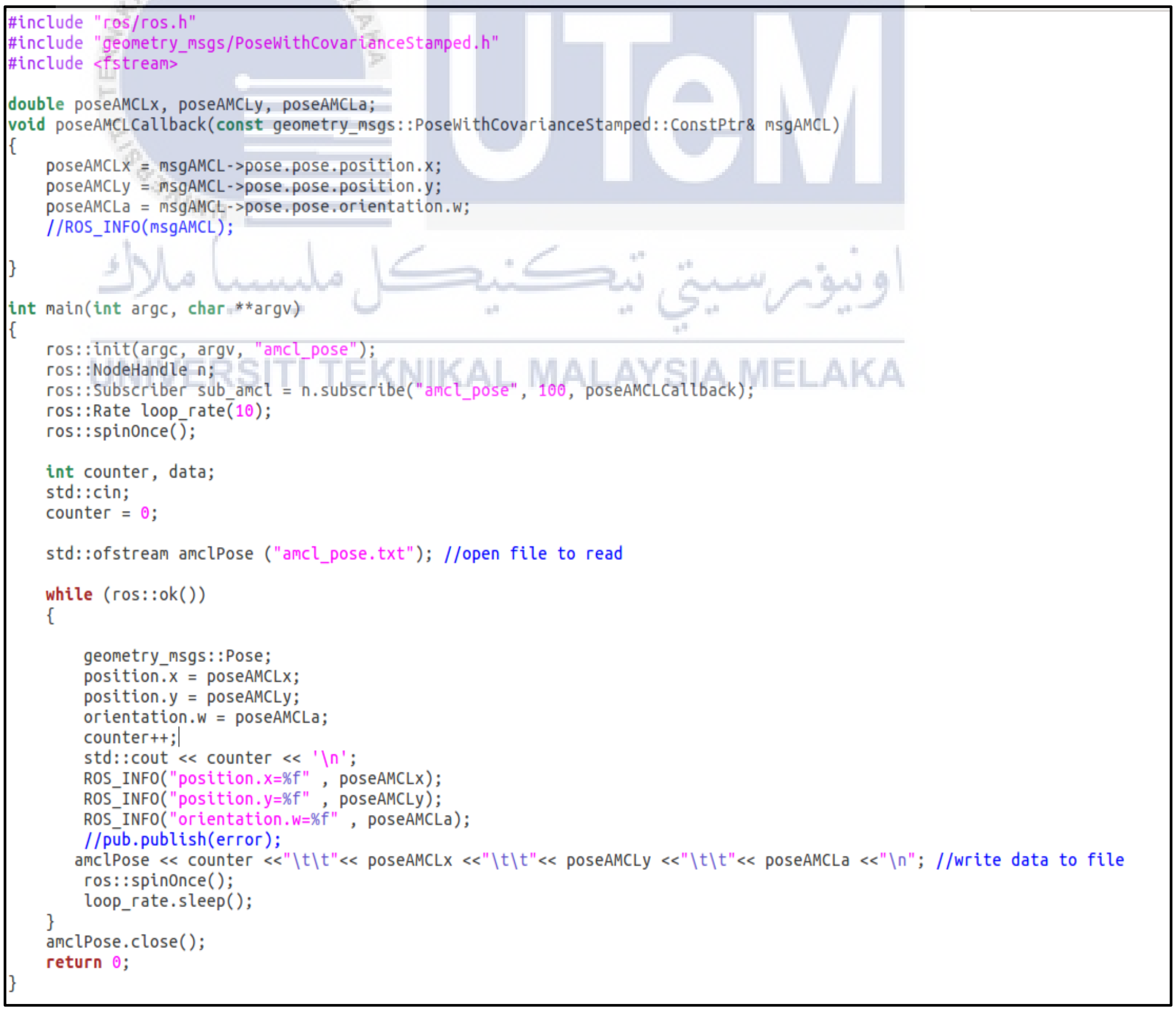
Table 3 : Particle filter localization simulation at different state simulation on Rviz and Gazebo simulator.

Explanation	Diagram
The initial state of the particle filter.	 (a)
The intermediate state of the particle filter.	 (b)
The final state of the particle filter.	 (c)

4.5 Localization Performance of Particle Filter Algorithm

To investigate the localization performance for each number of particles used, the value of the particle filter estimate pose and the ground truth pose of the TurtleBot3 Burger robot were recorded. The particle filter estimate pose and the ground truth pose of the robot will be used to calculate the error to investigate the accuracy of the robot’s state.

To obtain the particle filter estimate pose of the robot, a coding was created as shown in Figure 4.8. The file was named as sub_amcl.cpp file. It is used to record the pose of the robot at the position of x, y, and the orientation in the RViz simulation.



```
#include "ros/ros.h"
#include "geometry_msgs/PoseWithCovarianceStamped.h"
#include <fstream>

double poseAMCLx, poseAMCLy, poseAMCLa;
void poseAMCLCallback(const geometry_msgs::PoseWithCovarianceStamped::ConstPtr& msgAMCL)
{
    poseAMCLx = msgAMCL->pose.pose.position.x;
    poseAMCLy = msgAMCL->pose.pose.position.y;
    poseAMCLa = msgAMCL->pose.pose.orientation.w;
    //ROS_INFO(msgAMCL);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "amcl_pose");
    ros::NodeHandle n;
    ros::Subscriber sub_amcl = n.subscribe("amcl_pose", 100, poseAMCLCallback);
    ros::Rate loop_rate(10);
    ros::spinOnce();

    int counter, data;
    std::cin;
    counter = 0;

    std::ofstream amclPose ("amcl_pose.txt"); //open file to read

    while (ros::ok())
    {

        geometry_msgs::Pose;
        position.x = poseAMCLx;
        position.y = poseAMCLy;
        orientation.w = poseAMCLa;
        counter++;
        std::cout << counter << '\n';
        ROS_INFO("position.x=%f" , poseAMCLx);
        ROS_INFO("position.y=%f" , poseAMCLy);
        ROS_INFO("orientation.w=%f" , poseAMCLa);
        //pub.publish(error);
       amclPose << counter <<"\t\t"<< poseAMCLx <<"\t\t"<< poseAMCLy <<"\t\t"<< poseAMCLa <<"\n"; //write data to file
        ros::spinOnce();
        loop_rate.sleep();
    }
    amclPose.close();
    return 0;
}
```

Figure 4.8 : Coding for sub_amcl.cpp file.

Based on the coding in Figure 4.8, `amcl_pose` node subscribes the `amcl_pose` topic to obtain the particle filter estimate pose of the robot. To run this file, the command in Figure 4.9 shown below will be used to obtain the pose of the robot that simulates in the RViz simulation. All the data of the particle filter estimate pose of the robot is recorded in a file named amcl_pose.txt file. The data will be recorded once the simulation is started.

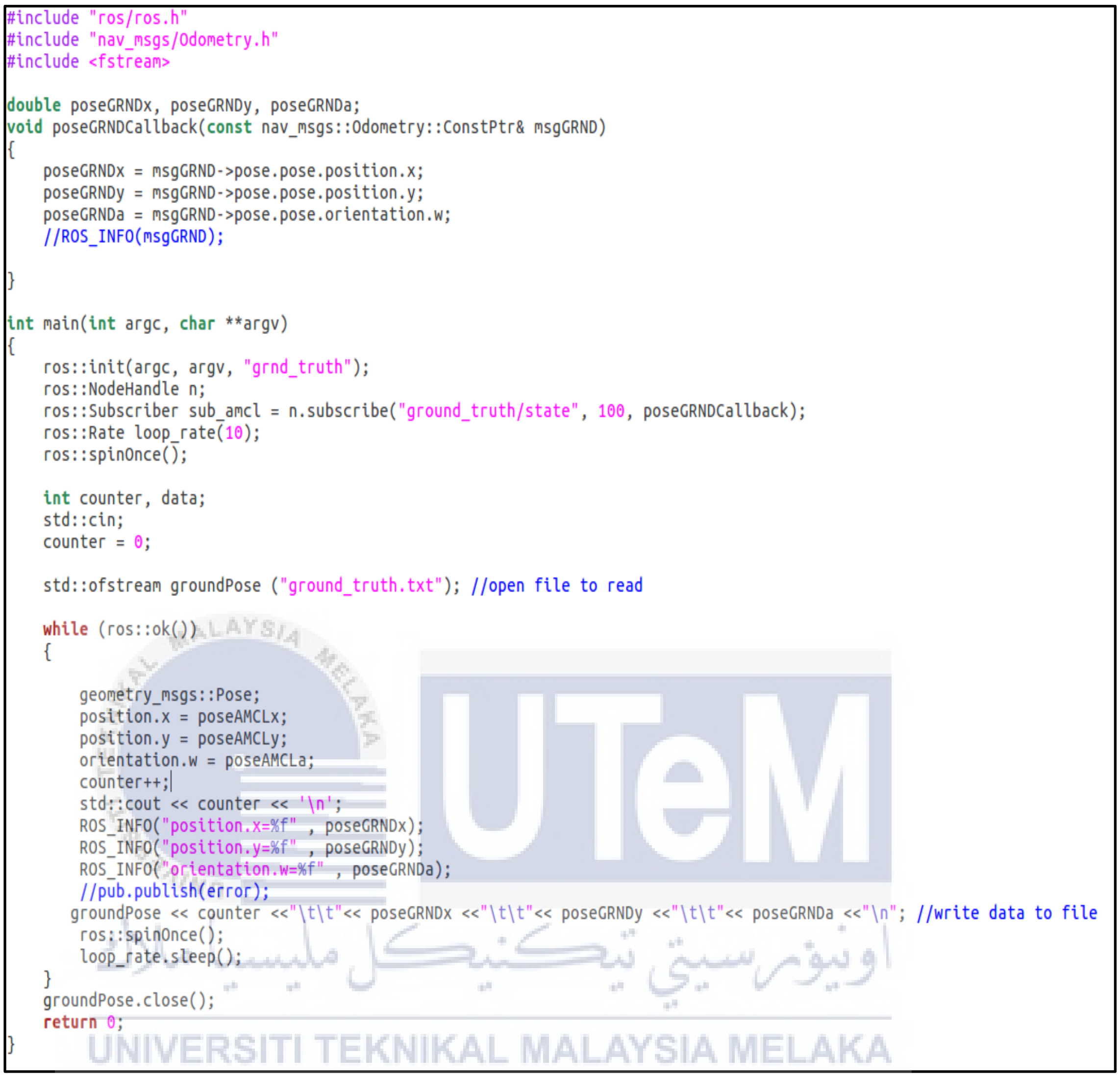
```
$ rosrun amcl sub_amcl
```

Figure 4.9 : Command to obtain particle filter estimate pose.

At the same time, to obtain the ground truth pose of the robot, a coding in Figure 4.11 was created. The coding is saved in a file that was named as sub_ground.cpp file. It is used to record the pose of the robot at the position of x, y, and the orientation in the Gazebo simulator. According to the coding in Figure 4.11, `grnd_truth` node subscribes to the `ground_truth/state` topic to obtain the ground truth pose of the robot. All the data of the ground truth pose of the robot is recorded in a file named ground_truth.txt file. To run the file, the command in Figure 4.10 shown below will be used to obtain the pose of the robot that simulates in the Gazebo simulator.

```
$ rosrun amcl sub_ground
```

Figure 4.10 : Command to obtain ground truth pose.



```
#include "ros/ros.h"
#include "nav_msgs/Odometry.h"
#include <fstream>

double poseGRNDx, poseGRNDy, poseGRNDa;
void poseGRNDCallback(const nav_msgs::Odometry::ConstPtr& msgGRND)
{
    poseGRNDx = msgGRND->pose.pose.position.x;
    poseGRNDy = msgGRND->pose.pose.position.y;
    poseGRNDa = msgGRND->pose.pose.orientation.w;
    //ROS_INFO(msgGRND);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "grnd_truth");
    ros::NodeHandle n;
    ros::Subscriber sub_amcl = n.subscribe("ground_truth/state", 100, poseGRNDCallback);
    ros::Rate loop_rate(10);
    ros::spinOnce();

    int counter, data;
    std::cin;
    counter = 0;

    std::ofstream groundPose ("ground_truth.txt"); //open file to read

    while (ros::ok())
    {

        geometry_msgs::Pose;
        position.x = poseAMCLx;
        position.y = poseAMCLy;
        orientation.w = poseAMCLa;
        counter++;
        std::cout << counter << '\n';
        ROS_INFO("position.x=%f" , poseGRNDx);
        ROS_INFO("position.y=%f" , poseGRNDy);
        ROS_INFO("orientation.w=%f" , poseGRNDa);
        //pub.publish(error);
       groundPose << counter <<"\t\t"<< poseGRNDx <<"\t\t"<< poseGRNDy <<"\t\t"<< poseGRNDa <<"\n"; //write data to file
        ros::spinOnce();
        loop_rate.sleep();
    }
    groundPose.close();
    return 0;
}
```

Figure 4.11 : Coding for sub_ground.cpp file.

During the simulation of the localization for the robot, there are several topics and nodes were listed. Thus, ROS provides the capabilities to receive any existing information on the current system. The structure of the computation graph that shows the nodes running in the system is one of the important information. In ROS, `rqt_graph` is one of a common tool for runtime visualization of the computation graph [25]. The graph can be visualized by running the `rqt_graph` command. Then, the rqt graph as shown in Figure 4.12 will appear.

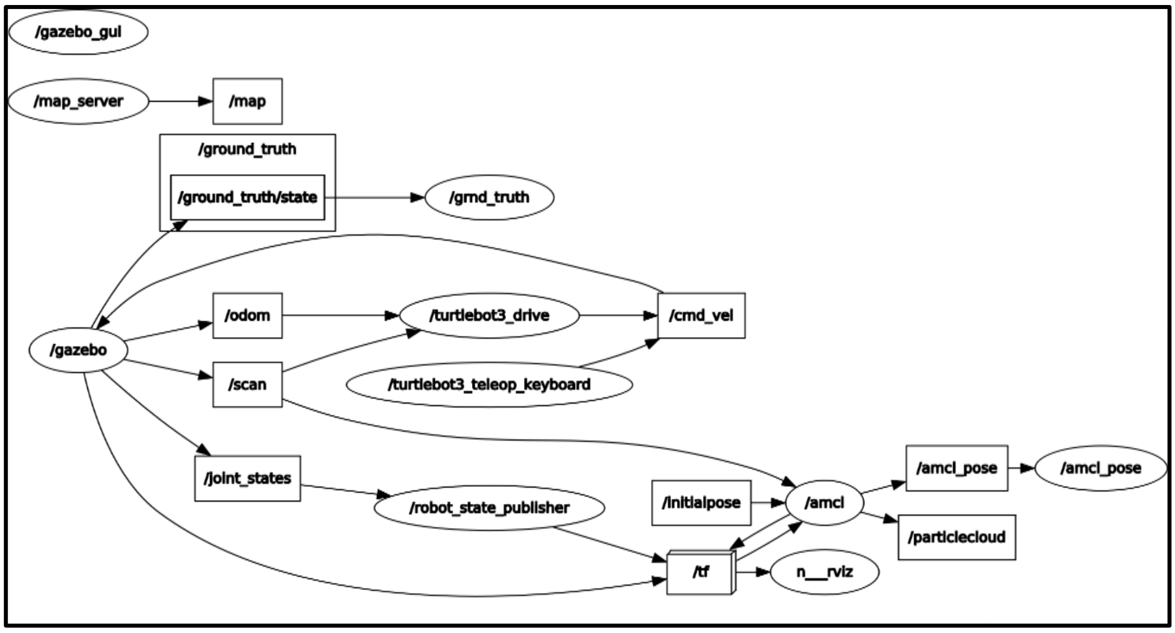


Figure 4.12: The rqt graph of all the topics and nodes running in the system.

The graphical view of the topics and nodes were shown in Figure 4.12 above. Figure 4.12 shows the rqt graph of the topics after all the nodes used for the localization of the robot are launched. The components in the rectangle shape represent the topic used for the simulation. While the components in oval shape represent the nodes launched in the ROS.

The */map_server* node shown in Figure 4.12 is launched to provide the saved map that was created in the RViz simulation as mentioned in Figure 4.1. For the localization, the starting node is */gazebo* that provides the test environment for the robot simulation. The other node which is */turtlebot3_teleop_keyboard* is launched to move the TurtleBot3 Burger in the Gazebo simulator. The */grnd_truth* node is launched to get the ground truth pose of the robot. Besides, */amcl* node subscribes the */scan* topic in order to track the position of the robot in the RViz simulation. Then, the */amcl* node publish messages to */amcl_pose* topic. The */amcl_pose* node is launched to get the particle filter estimate pose of the robot in the RViz simulation.

4.6 Results of Particle Filter Localization Simulation

The simulation of particle filter localization for the mobile robot is tested by using the different number of particles from 500 to 8000 particles. Figure 4.13 below shows the graph of error obtained by using the different numbers of particles. The value of the errors was obtained by using the value of particle filter estimate pose and the ground truth pose of the TurtleBot3 Burger.

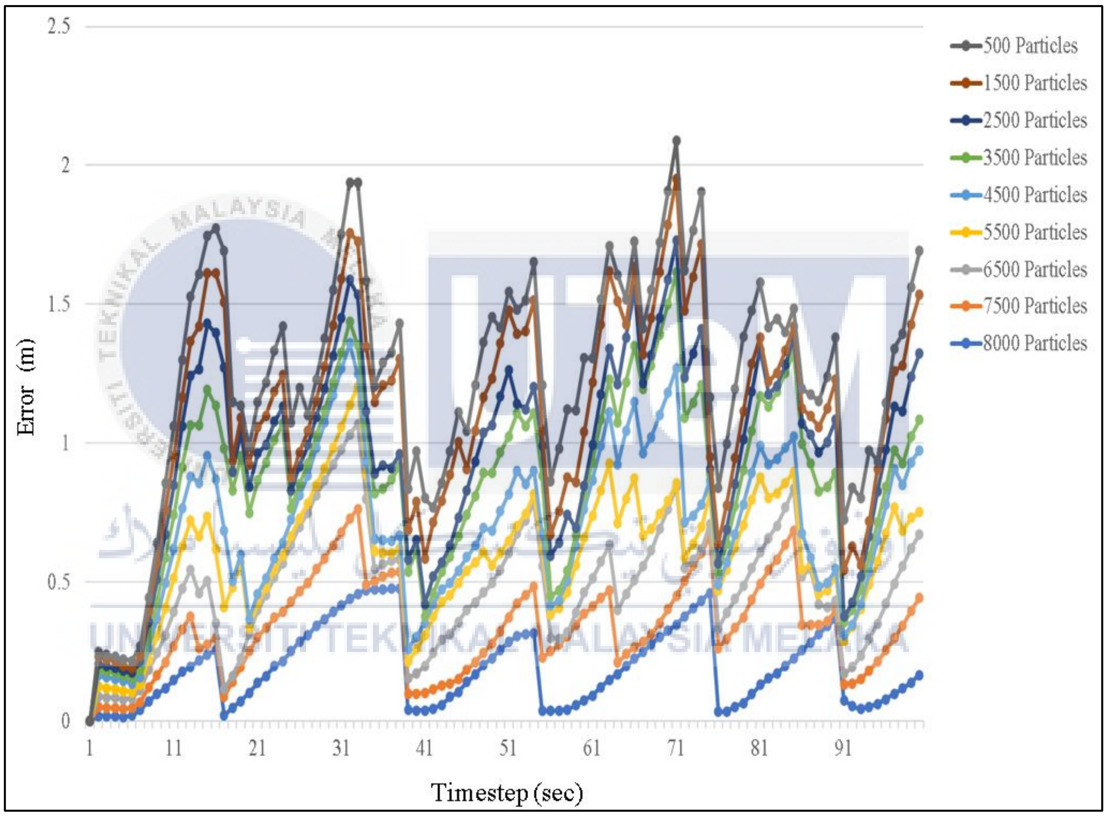


Figure 4.13 : The error obtained by using the different numbers of particles.

Based on Figure 4.13, it shows that the error for 500 particles is the highest value obtained compared to the other. While the lowest value of the is recorded when 8000 particles are used for the simulation. Besides, by referring to the figure, it shows that by using 8000 particles, the errors obtained is maintained within the bounds compared to when using 500 number of particles for the simulation. At timestep of 100 seconds, the error obtained by using 500 particles is 1.695 meters. While by using 800 particles, the error obtained to localize the robot is 0.872 meters at the timestep of 100 seconds. Therefore, the error obtained by using 8000 particles is 48% lesser than using 500 particles. This finding shows that by using 8000 particles, particle filter can localize the robot with less error obtained compared to 500 particles. Therefore, it can be concluded that the higher the number of particles used, the lesser the error obtained for the particles to localize the mobile robot to its position.

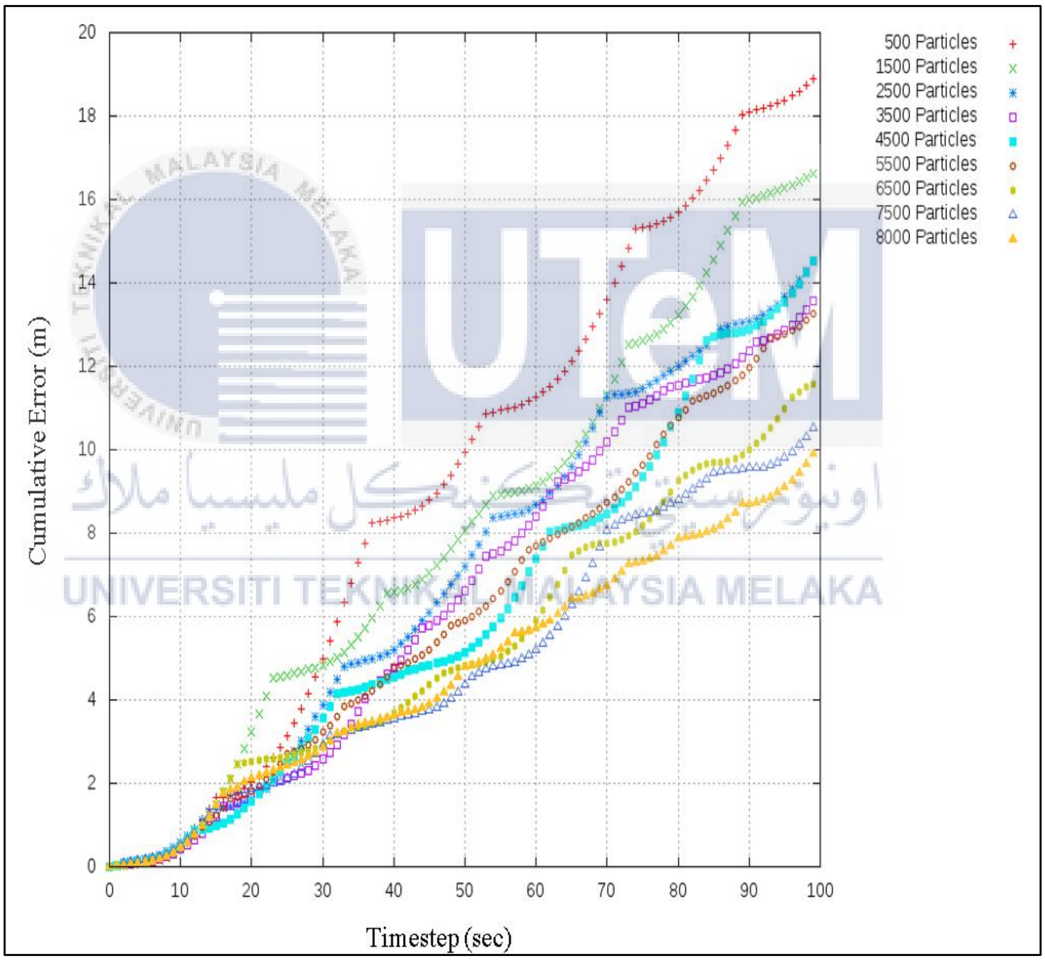


Figure 4.14 : Cumulative error by using the different number of particles.

Besides, Figure 4.14 above shows the cumulative error obtained by using the different number of particles. The simulation of particle filter localization by using

500 particles shows the highest cumulative error compared to the other number of particles. While using 8000 particles, it shows that the cumulative error obtained is lower compared to 500 particles. Figure 4.14 shows that the cumulative error obtained can be reduced as when the number of particles used for the particle filter localization increases.

To analyze the accuracy quantitatively, the error between the particle filter robot's state estimate and the ground truth robot's pose is calculated by using an equation in Equation (4.1). From the errors obtained, the cumulative error was calculated as shown in Equation (4.2) for each number of particles used. The significance of using cumulative error formula is to show clearly which of the number of particles used results the highest or the lowest number of the cumulative error as the timestep increases.

From that, the number of particles that have the lowest cumulative error is the best number of particles that can be used for the localization as it reflects the highest accuracy. Both Equation (4.1) and (4.2) were shown as below:

i. **The error between particle filter pose and the ground truth calculated by using the formula:**

$$Error = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{4.1}$$

ii. **The cumulative error calculated by using the formula:**

$$Cumulative\ Error = \Sigma\ Error \tag{4.2}$$

From the graph in Figure 4.14, the cumulative errors of the final pose of TurtleBot3 for the different number of particles used were recorded in Table 4. Table 4 shows that the value of the cumulative error of the final pose by using 500 particles is the highest compared to the others with 18.902 meters. Compared to the value obtained by using 8000 particles, it shows that the cumulative error of the final pose is decreased to 9.906 meters which is 47% lesser than using 500 particles.

Table 4 : The Cumulative error of the final pose.

Number of Particles Used	Cumulative Error of Final Pose (m)
500	18.902
1500	16.624
2500	14.512
3500	13.561
4500	14.511
5500	13.259
6500	11.579
7500	10.535
8000	9.906

4.7 Analysis of the Cumulative Error of the Final Pose of Turtlebot3 Burger According to The Number of Particles Used

From the simulation, it shows that the cumulative error of the final pose of TurtleBot3 Burger recorded in Table 4 decreased as the number of particles used increased. Based on Table 4, the cumulative error of the final pose according to the number of particles used is plotted in Figure 4.15. From the graph shown in Figure 4.15, the accuracy of the particle filter localization algorithm for the robot can be maximized as the error obtained is decreased when the number of particles used increases.

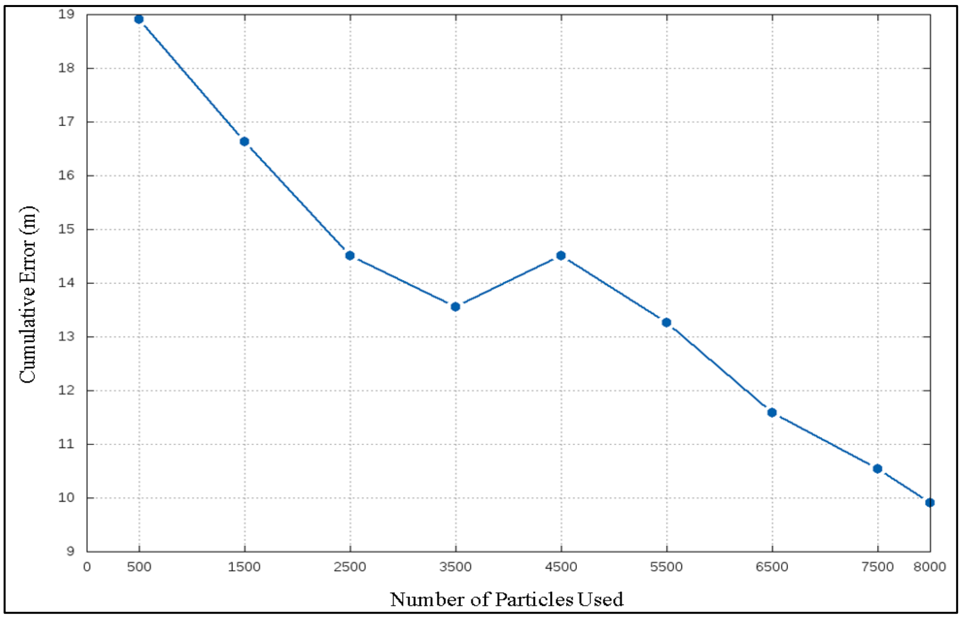


Figure 4.15 : Cumulative error of the final pose versus the number of particles used.

Referring to Figure 4.15, it shows that by using 500 particles, the highest cumulative error was obtained of the final pose and decreased by 2.278 meters when using 1500 particles. It continuously decreases until 8000 particles that shows the lowest cumulative error of the final pose is achieved compared to the others. From the graph in Figure 4.15, it shows the higher the number of particles used to localize the robot, the cumulative error of the final pose of the robot can be decreased. By using the particle filter algorithm, the higher the number of particles used is the best preference as it results the better to estimate the position of the robot. As the errors obtained can be reduced as the number of particles used is increasing, the accuracy to localize the robot can be maximized. Hence, it can be concluded that as the number of particles used increases, then the cumulative error of the final pose decreases and the accuracy of the estimation of the robot’s position can be maximized.

4.8 Project Significant

The environment and sustainability are represented as an ecosystem for the living things and the ability to control the production at a certain point or a certain behavior respectively. The combination of the environment and sustainable development mean maintaining long-term environmental quality factors and practices. When implementing a project, the environment and sustainability are significant to be considered because this can affect our nature and society directly.

First, the impact of this project will focus primarily on the development of the community. Robot Operating System (ROS) is an open-source software where it can be used as the localization software. The standard development of the localization software allows the use of many open source programs and the contribution to community development more efficiently. This would make the development of the software with more resources easier. Moreover, this project had shown the potential in development in terms of the autonomous system which can fully utilize on controlling by using the localization and detection of the mobile robot system. The software proposed gives the competency features especially to the persons or workers in manufacturing industries to detect and track the position of the mobile robot easily. Besides, this software helps them to localize or detect the position of the mobile robot even it was located in various scenes such as in the store and warehouse at any time by using the software.

Next, the impact of this project on social. This project will be needed especially for manufacturing industries. This is because manufacturing industries need a big help from the robotic services. The reason is that the robot can performs repeatable works. As a result of that, its benefits to the human where they can focus and work on more creative tasks. For example, Autonomous Mobile Robots (AMRs) are one of the most commonly used, especially in warehouses. In the AMRs, the sensor technology was developed in order to deliver or move the items from one place to another place around the warehouses. To understand and develop the environment such as warehouses, the use of computers, sensors, and the maps are required in the AMRs. The efficiency and accuracy were offered by the AMRs as it helps to minimize the repetitious works during the sorting process and allows workers to perform more useful works. Hence, the errors can be reduced by using the robot as it delivers a constant error compared to humans.

Furthermore, in the economy sector, this project gives the cost-saving method in reducing manufacturing costs. The manufacturing cost can be reduced as the factory or the manufacturer company does not require compensation since no manpower needed. In the starting, the cost needed to install the robots may seem daunting but it will benefit for a long term return on their investment. This is because the robots can work 24 hours per day without a break if needed compared to the humans that have the limitation of time and energy to work. Unlike humans, robots can do specific repeatable works or movements at the same time can reduce the time spent correcting human errors.

Last but not least, in the environment sector, the robot can perform in various scenes either in dangerous or harmful environments that humans cannot operate. This can help to reduce the number of accidents occur in the workplace among the workers. At the same time, as the robots are used especially in manufacturing industries for safety in factories and manufacturing plants, working conditions can be improved. Besides, this invention offers the energy and time saving to the users. Only with help on the software, they could detect and track the mobile robot from far or even when they are not around in a certain environment. Without this software, they have to monitor the mobile robot manually to ensure it is on the track. They have to manage it one by one. Thus, it requires more time and energy. Hence, with this localization of mobile robot software, more time and energy could be saved.

For this project, the only limitation of this software is that it only can be implemented in the indoor and static environment since less sensor noise involved during the localization of the robot. Besides, the map of the test environment needs to be updated if there is any changing of the environment as it affects the accuracy of the localization of the mobile robot.

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 Conclusion

In this report, the performance analysis of the localization algorithm for mobile robots is studied by using the particle filter method. The particle filter uses particles as the representation of the robot's belief. In particle filter, each particle is the hypothesis of where the robot might be.

The main idea was to track the location and discover the current position of the mobile robot in the absence of GPS coverage. To optimize the localization algorithm, particle filter method is used to solve the problem. The particle filter algorithm provides solutions for mobile robot localization problems such as position tracking, global localization, and kidnapped robot.

The first objective of this project is to simulate the particle filter localization algorithm and investigate the performance based on the accuracy of the robot’s state. The test environment used in this project for the simulation of the localization algorithm of the mobile robot is studied and developed by using Robot Operating Software (ROS). There are several main structures of ROS were studied such as topic, subscriber and publisher of ROS software. By using the rqt graph, all the nodes running in the current system used for the publish-subscribe mechanism were shown. Gazebo and RViz simulator in ROS were used to create the test environment and simulate the particle filter algorithm for the localization respectively. Furthermore, different numbers of particles were used in the simulation by using the ROS software. Even though it takes time to develop this project since it is a new development, the particle filter localization algorithm for the mobile robot was successfully simulated.

Next, the second objective of the project is to investigate the effect of the number of particles used on the accuracy of the robot’s state. For that, this project is simulated by using a different number of particles. Based on the simulation results shown in Chapter 4, the particle filter localization of TurtleBot3 Burger simulation is tested by using 500, 1500, 2500, 3500, 4500, 5500, 6500, 7500, and 8000 particles. The simulation shows that the cumulative error obtained by using 500 particles is the highest value compared to the higher number of particles. From the results, it shows that the accuracy of the mobile robot localization can be increased as the particles used are increasing.

Based on the results and analysis, the Performance Analysis Of The Localization Algorithm For Mobile Robots is a wise project made. However, there is a limitation for this particle filter localization algorithm. The algorithm only can be tested in the static environment. If the algorithm is tested in a dynamic environment, the localization of the mobile robot may be interrupted by a higher noise level. There are several improvements that can be made to create a better software system. In conclusion, the performance analysis of the localization algorithm for mobile robots is successfully developed. This project has provided an immersive experience through advanced technology for the student.

5.2 Future Works

For the current project of The Performance Analysis of The Localization Algorithm For Mobile Robots, it only can be applied in the static environment since there is less sensor noise involves during the localization. If the algorithm is tested in a dynamic environment, the localization of the mobile robot may be interrupted by other noise. In the next stage of this research, a proposed approach by applying the particle filter algorithm in the dynamic environment which involves sensor noise parameters that can be considered during the localization process.

Besides, according to localization, the mobile robot position is estimated based on the environment map. Therefore, the map needs to be updated when the environment is changing. The localization of the mobile robot may not precise or accurate if the map is not updated. Therefore, another approach with the presence or absence of the map information can be combined with this current technique of localization for particle filter to localize the mobile robot in the future. Hence, some improvements can be made to track or localize the mobile robot more efficiently.

REFERENCES

[1] P. Wozniak, H. Afrisal, R. G. Esparza, and B. Kwolek, “Scene recognition for indoor localization of mobile robots using deep CNN,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[2] H. Zhang, C. Zhang, W. Yang, and C. Y. Chen, “Localization and navigation using QR code for mobile robot in indoor environment,” *2015 IEEE Int. Conf. Robot. Biomimetics, IEEE-ROBIO 2015*, no. December 2015, pp. 2501–2506, 2015.

[3] A. Pandey, “Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review,” *Int. Robot. Autom. J.*, vol. 2, no. 3, 2017.

[4] I. Ashokaraj, A. Tsourdos, P. Silson, and B. White, “Sensor based robot localisation and navigation: Using interval analysis and nonlinear kalman filters,” *Trans. Can. Soc. Mech. Eng.*, 2005.

[5] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, “Particle Filters for Mobile Robot Localization,” *Seq. Monte Carlo Methods Pract.*, pp. 401–428, 2001.

[6] S. Nurmaini and S. Pangidoan, “Localization of Leader-Follower Robot Using Extended Kalman Filter,” *Comput. Eng. Appl. J.*, vol. 7, no. 2, pp. 95–108, 2018.

[7] F. G. Bravo, A. Vale, and M. I. Ribeiro, “Navigation strategies for cooperative localization based on a particle-filter approach,” *Integr. Comput. Aided. Eng.*, vol. 14, no. 3, pp. 263–279, 2007.

[8] S. L. Shue, N. S. Shetty, A. F. Browne, and J. M. Conrad, “Particle Filter Approach to Utilization of Wireless Signal Strength for Mobile Robot Localization in Indoor Environments,” *Int. J. Wirel. Mob. Networks*, vol. 10, no. 4, pp. 21–38, 2018.

[9] M. Mirkhani, R. Forsati, A. M. Shahri, and A. Moayedikia, “A novel efficient algorithm for mobile robot localization,” *Rob. Auton. Syst.*, vol. 61, no. 9, pp. 920–931, 2013.

[10] J. Pomárico-Franquiz, S. H. Khan, and Y. S. Shmaliy, “Combined extended FIR/Kalman filtering for indoor robot localization via triangulation,” *Meas. J. Int. Meas. Confed.*, vol. 50, no. 1, pp. 236–243, 2014.

[11] N. N. Bhat, “Robot Localization by Particle Filter using Visual Database .,” no. 1, pp. 22–27, 2012.

[12] V. Raudonis, R. Simutis, and A. Paulauskaite-Taraseviĉiene, “An efficient object tracking algorithm based on dynamic particle filter,” *Elektron. ir Elektrotechnika*, vol. 3, no. 3, pp. 93–98, 2009.

[13] C. C. Hsu, S. S. Yeh, and P. Lo Hsu, “Particle filter design for mobile robot

localization based on received signal strength indicator," *Trans. Inst. Meas. Control*, vol. 38, no. 11, pp. 1311–1319, 2016.

[14] A. Shukla, R. Singh, R. Agarwal, M. Suhail, S. K. Saha, and S. Chaudury, "Development of a Low-Cost Education Platform: RoboMuse 4.0," *ACM Int. Conf. Proceeding Ser.*, vol. Part F1320, pp. 0–5, 2017.

[15] 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.

[16] J. Xiao, D. Xiong, W. Yao, Q. Yu, H. Lu, and Z. Zheng, *Building Software System and Simulation Environment for RoboCup MSL Soccer Robots Based on ROS and Gazebo*. 2017.

[17] W. Qian *et al.*, "Manipulation task simulation using ROS and Gazebo," *2014 IEEE Int. Conf. Robot. Biomimetics, IEEE ROBIO 2014*, pp. 2594–2598, 2014.

[18] L. Pitonakova, M. Giuliani, A. Pipe, and A. Winfield, "Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10965 LNAI, pp. 357–368, 2018.

[19] J. H. Gao and L. S. Peh, "A smartphone-based laser distance sensor for outdoor environments," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2016-June, pp. 2922–2929, 2016.

[20] M. F. Measurements, X. Wang, S. Member, C. Zhang, F. Liu, and Y. Dong,

"Exponentially Weighted Particle Filter for Simultaneous Localization and Mapping Based on," pp. 1–10.

[21] I. Rekleitis, "A particle filter tutorial for mobile robot localization," *Cent. Intell. Mach. McGill Univ. Tech. …*, 2004.

[22] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IEEE International Conference on Intelligent Robots and Systems*, 2012.

[23] S. Zaman, W. Slany, and G. Steinbauer, "ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues," *Saudi Int. Electron. Commun. Photonics Conf. 2011, SIECPC 2011*, no. May 2014, 2011.

[24] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artif. Intell.*, vol. 128, no. 1–2, pp. 99–141, 2001.

[25] A. Santos, A. Cunha, and N. Macedo, "Static-Time Extraction and Analysis of the ROS Computation Graph," *Proc. - 3rd IEEE Int. Conf. Robot. Comput. IRC 2019*, pp. 62–69, 2019.

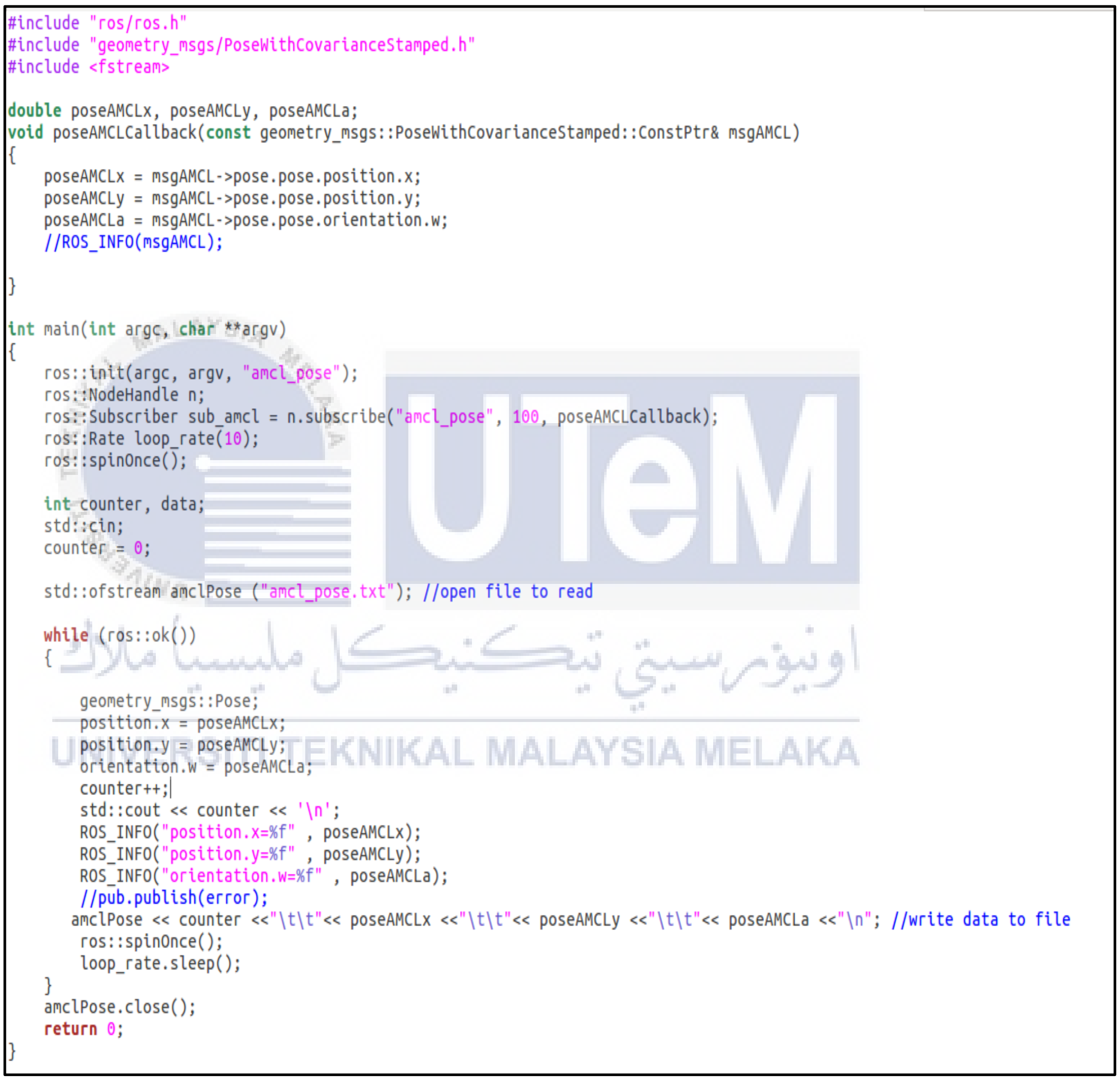
APPENDICES

APPENDIX A : GITHUB RESPIRATORY

Nur Nabila Husna, 2020, Github Respiratory, `https://github.com/hus-nabila/The-Performance-Analysis-of-Localization-Algorithm-For-Mobile-Robot`

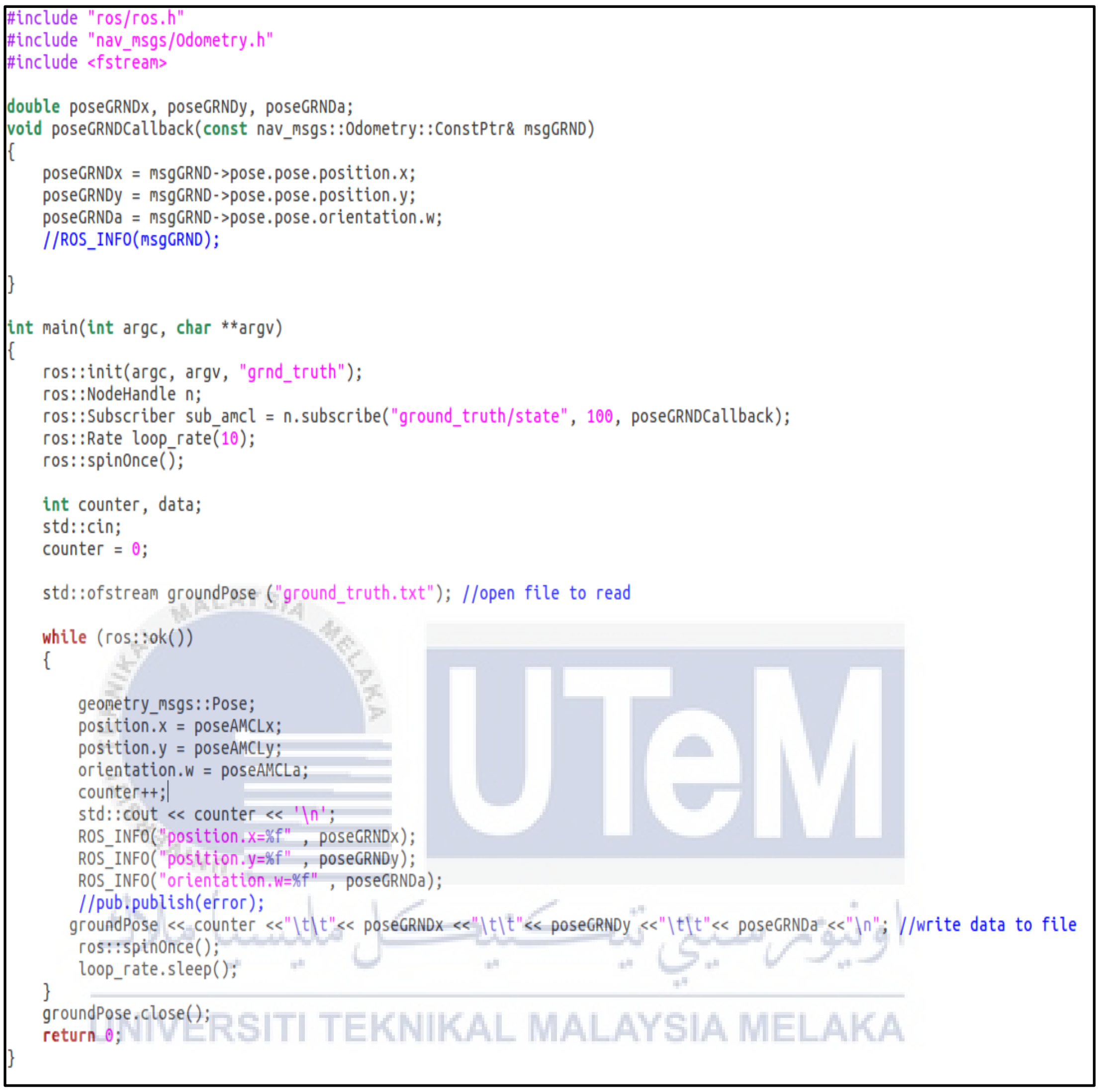
APPENDIX B : THE CODINGS

i. Coding for sub_amcl.cpp file.



```
#include "ros/ros.h"
#include "geometry_msgs/PoseWithCovarianceStamped.h"
#include <fstream>

double poseAMCLx, poseAMCLy, poseAMCLa;
void poseAMCLCallback(const geometry_msgs::PoseWithCovarianceStamped::ConstPtr& msgAMCL)
{
    poseAMCLx = msgAMCL->pose.pose.position.x;
    poseAMCLy = msgAMCL->pose.pose.position.y;
    poseAMCLa = msgAMCL->pose.pose.orientation.w;
    //ROS_INFO(msgAMCL);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "amcl_pose");
    ros::NodeHandle n;
    ros::Subscriber sub_amcl = n.subscribe("amcl_pose", 100, poseAMCLCallback);
    ros::Rate loop_rate(10);
    ros::spinOnce();

    int counter, data;
    std::cin;
    counter = 0;

    std::ofstream amclPose ("amcl_pose.txt"); //open file to read

    while (ros::ok())
    {

        geometry_msgs::Pose;
        position.x = poseAMCLx;
        position.y = poseAMCLy;
        orientation.w = poseAMCLa;
        counter++;
        std::cout << counter << '\n';
        ROS_INFO("position.x=%f" , poseAMCLx);
        ROS_INFO("position.y=%f" , poseAMCLy);
        ROS_INFO("orientation.w=%f" , poseAMCLa);
        //pub.publish(error);
       amclPose << counter <<"\t\t"<< poseAMCLx <<"\t\t"<< poseAMCLy <<"\t\t"<< poseAMCLa <<"\n"; //write data to file
        ros::spinOnce();
        loop_rate.sleep();
    }
    amclPose.close();
    return 0;
}
```

ii. Coding for sub_ground.cpp file.



```cpp
#include "ros/ros.h"
#include "nav_msgs/Odometry.h"
#include <fstream>

double poseGRNDx, poseGRNDy, poseGRNDa;
void poseGRNDCallback(const nav_msgs::Odometry::ConstPtr& msgGRND)
{
    poseGRNDx = msgGRND->pose.pose.position.x;
    poseGRNDy = msgGRND->pose.pose.position.y;
    poseGRNDa = msgGRND->pose.pose.orientation.w;
    //ROS_INFO(msgGRND);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "grnd_truth");
    ros::NodeHandle n;
    ros::Subscriber sub_amcl = n.subscribe("ground_truth/state", 100, poseGRNDCallback);
    ros::Rate loop_rate(10);
    ros::spinOnce();

    int counter, data;
    std::cin;
    counter = 0;

    std::ofstream groundPose ("ground_truth.txt"); //open file to read

    while (ros::ok())
    {

        geometry_msgs::Pose;
        position.x = poseAMCLx;
        position.y = poseAMCLy;
        orientation.w = poseAMCLa;
        counter++;
        std::cout << counter << '\n';
        ROS_INFO("position.x=%f" , poseGRNDx);
        ROS_INFO("position.y=%f" , poseGRNDy);
        ROS_INFO("orientation.w=%f" , poseGRNDa);
        //pub.publish(error);
       groundPose << counter <<"\t\t"<< poseGRNDx <<"\t\t"<< poseGRNDy <<"\t\t"<< poseGRNDa <<"\n"; //write data to file
        ros::spinOnce();
        loop_rate.sleep();
    }
    groundPose.close();
    return 0;
}
```