ANALYSIS ON 2D MAPPING FOR MOBILE ROBOT ON THE SHARP EDGES AREA

DESMOND LING ZE YEW

B. ENG(HONS.) AUTOMOTIVE ENGINEERING (COLLABORATION PROGRAMME WITH HKA, GERMANY)

UNIVERSITI MALAYSIA PAHANG

AL-SULTAN ABDULLAH

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name	: ______________________________
Date of Birth	: ______________________________
Title	: ______________________________
	______________________________
	______________________________
Academic Session	: ______________________________

I declare that this thesis is classified as:

☐	CONFIDENTIAL	(Contains confidential information under the Official Secret Act 1997)*
☐	RESTRICTED	(Contains restricted information as specified by the organization where research was done)*
☑	OPEN ACCESS	I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

_______________________ (Student's Signature)

_______________________ (Supervisor's Signature)

____
New IC/Passport Number
Date:

____
Name of Supervisor
Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Perpustakaan Universiti Malaysia Pahang,
Universiti Malaysia Pahang,
Lebuhraya Tun Razak,
26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name
Thesis Title

Reasons	(i)
	(ii)
	(iii)

Thank you.

Yours faithfully,

____
(Supervisor's Signature)

Date:

Stamp:

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.

Universiti Malaysia PAHANG
Engineering • Technology • Creativity

SUPERVISOR'S DECLARATION

I/We* hereby declare that I/We* have checked this thesis/project* and in my/our* opinion, this thesis/project* is adequate in terms of scope and quality for the award of the degree of *Doctor of Philosophy/ Master of Engineering/ Master of Science in …………………………..

____

(Supervisor's Signature)

Full Name	:
Position	:
Date	:

____

(Co-supervisor's Signature)

Full Name	:
Position	:
Date	:

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

____

(Student's Signature)

Full Name	: DESMOND LING ZE YEW
ID Number	: HB19024
Date	: 21 OCTOBER 2024

ANALYSIS ON 2D MAPPING FOR MOBILE ROBOT ON THE SHARP EDGES AREA

DESMOND LING ZE YEW

Thesis submitted in fulfillment of the requirements
for the award of the
Bachelor Degree in Automotive Engineering

Faculty of Mechanical and Automotive Engineering Technology

UNIVERSITI MALAYSIA PAHANG

FEB 2024

ACKNOWLEDGEMENTS

I would like to extend my heartfelt appreciation to Dr. Mohamad Heerwan Bin Peeie, my esteemed supervisor, for his unwavering guidance, invaluable insights, and continuous support throughout the entire duration of my final year project. His expertise and encouragement played a pivotal role in shaping the trajectory of my research and contributed significantly to its successful completion.

A special note of gratitude goes to my dearest friend, Ang Suzanne, whose unwavering support and comforting presence were a beacon of strength during moments of challenge and uncertainty. Her encouragement and positivity were a source of motivation, making this academic journey more manageable and fulfilling.

I also wish to express my thanks to the incredible individuals in the Autonomous Lab. Their willingness to share knowledge, provide technical assistance, and foster a collaborative environment significantly enhanced the quality and depth of my project. The sense of camaraderie within the lab contributed to a positive and enriching experience.

I am deeply grateful to everyone who contributed to this project's success, be it through shared expertise, encouragement, or simply a helping hand. Each interaction and collaboration played a crucial role in the development and execution of this endeavor.

In conclusion, I extend my sincere appreciation to Dr. Mohamad Heerwan Bin Peeie, Ang Suzanne, and the Autonomous Lab team for their invaluable contributions to the realization of this project.

ABSTRAK

Simultaneous Localization and Mapping (SLAM) adalah teknik asas dalam sistem navigasi dalaman bagi kebanyakan kenderaan autonom dan robot. Salah satu isu dalam SLAM ialah kelajuan robot yang mungkin mempengaruhi kualiti pemetaan. Oleh itu, distorsi pergerakan sendiri LiDAR merupakan cabaran umum bagi pelbagai algoritma SLAM, terutamanya dalam persekitaran dengan tepi yang tajam. Disebabkan oleh isu ini, projek ini bertujuan untuk menganalisis impak distorsi pergerakan sendiri LiDAR terhadap tiga algoritma SLAM: GMapping, Hector SLAM, dan Google Cartographer. Algoritma-algoritma ini dilaksanakan pada robot TurtleBot3 Burger untuk menjalankan pemetaan 2D di bawah keadaan kelajuan yang berbeza (0.07m/s, 0.14m/s, dan 0.22m/s) di Control System Lab di UMPSA. Kualiti peta yang dihasilkan dinilai dengan mengukur panjang dinding yang telah ditakrif dan sudut penjuru yang telah ditakrif, dan membandingkannya dengan dimensi sebenar dalam dunia sebenar. Metrik kesilapan mutlak dan statistik (MAE, MSE, RMSE, dan MAPE) dikira untuk setiap titik data dan setiap algoritma. Hasil menunjukkan bahawa Hector SLAM adalah algoritma paling tebal di bawah kelajuan tinggi, semua dinding dan penjuru dapat dipetakan dengan tepat, dengan nilai MAPE terendah, disebabkan oleh kebebasannya daripada data odometri. Hasil juga mendedahkan bahawa kesan distorsi pergerakan sendiri LiDAR meningkat dengan kelajuan, seperti yang ditunjukkan oleh nilai kesilapan yang lebih tinggi untuk semua algoritma. Kajian ini menyumbang kepada pemahaman bagaimana distorsi pergerakan sendiri LiDAR mempengaruhi prestasi pelbagai algoritma SLAM, dan memberikan pandangan untuk memilih algoritma yang sesuai untuk skenario kelajuan yang berbeza.

ABSTRACT

Simultaneous localization and mapping (SLAM) is a fundamental technique block in the indoor-navigation system for most autonomous vehicles and robots. One of the issues in SLAM is that the speed of the robot may affect the mapping quality. Therefore, LiDAR self-motion distortion is a common challenge for different SLAM algorithms, especially in environments with sharp edges. Due to this issue, this project aims to analyze the impact of LiDAR self-motion distortions on three SLAM algorithms: GMapping, Hector SLAM, and Google Cartographer. These algorithms are implemented on a TurtleBot3 Burger robot to perform 2D mapping under different speed conditions (0.07m/s, 0.14m/s, and 0.22m/s) in the Control System Lab at UMPSA. The quality of the generated maps is evaluated by measuring the length of predefined walls and the angle of predefined corners, and comparing them with the actual dimensions in the real world. The absolute error and statistical error metrics (MAE, MSE, RMSE, and MAPE) are computed for each data point and each algorithm. The results show that Hector SLAM is the most robust algorithm under high speed, all the walls and corners can be accurately mapped, with the lowest MAPE value, due to its independence of odometry data. The results also reveal that the effect of LiDAR self-motion distortion increases with speed, as indicated by the higher error values for all the algorithms. This study contributes to the understanding of how LiDAR self-motion distortions affect the performance of different SLAM algorithms, and provides insights for choosing the appropriate algorithm for different speed scenarios.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

MAE	Mean Average Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAPE	Mean Absolute Percentage Error
cm	centimetre
m	metre
$^\circ$	degree

LIST OF ABBREVIATIONS

SLAM	Simultaneous Localization and Mapping
LiDAR	Light Detection and Ranging
OCM	Occupancy Grid Map
ROS	Robot Operating System
IMU	Inertial Measurement Unit
ICP	Iterative Closest Point
RBPF	Rao-Blackwellized Particle Filter
LRF	Laser Range Finder

CHAPTER 1

INTRODUCTION

1.1 Introduction

With the remarkable advancement of information and communication technology paving the way for the digital transformation of industry, the use of advanced robots is being emphasised more and more as one of the key drivers of the fourth industrial revolution [1].

The future of manufacturing and logistics, according to a number of experts, lies in open networks of dynamic and changeable cyber-physical systems of cooperating autonomous entities [2][3]. Numerous benefits come with these methods, including lower costs and gains in performance, resilience, adaptability, and flexibility. However, these dispersed methods present a number of issues that need to be resolved. Decentralised information, decision myopia, security and confidentiality, network stability, local autonomy, and communication overload are a few examples of these [3].

In the aforementioned (cooperative) autonomous cyber-physical systems paradigm, autonomous mobile robots (AMRs) are a significant component and are essential to the development of intricate, flexible, distributed logistic systems [4].

One of the primary challenges for Autonomous Mobile Robots (AMRs) involves accurately perceiving their surroundings and efficiently navigating within it to reach a specific destination. Simultaneous Localization and Mapping (SLAM) [5][6] techniques address a portion of this problem by continuously collecting data from sensors, constructing an internal representation of the environment, and providing precise information about the robot's position and orientation within it. The initial step is accomplished by processing sensor data, which is then transformed and integrated to create an occupancy map. At the same time, any new sensor measurements are compared to the map to determine the robot's current position and orientation in the environment.

1.2 Problem Statement

Simultaneous Localization and Mapping (SLAM) algorithms play a crucial role in the navigation and mapping capabilities of mobile robots. However, the performance of these algorithms can be significantly affected by various factors, especially the speed of the robot, which can lead to the distortion of sharp edges in the resulting map. This is due to the LiDAR self-motion distortion, which is also known as point cloud distortion. It is a significant challenge in the field of robotics and autonomous driving. This distortion occurs when the LiDAR sensor and the objects in the environment are moving while the LiDAR is collecting data, causing the resulting point cloud to appear distorted. In other words, the relative motion between the spinning LiDAR sensor and objects leads to a distortion of the point cloud [7], [8], [9], [10]. This distortion can potentially affect the performance of various SLAM algorithms, which are critical for tasks such as navigation and obstacle avoidance [9], [10].

Based on Kristian Wahlqvist in his paper entitled “A Comparison of Motion Priors for EKF-SLAM in Autonomous Race Cars”, he stated that LiDAR odometry can be affected by motion distortion when the motion is fast relative to the scan rate. This means that the LiDAR measurements can become distorted due to the fact that scan points are captured at different points in time, and thus at different poses. The author suggests that aiding sensors such as cameras or IMUs are typically required for velocity estimates in order to cancel the motion distortion [11]. Another paper entitled “Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth” by Rauf Yagfarov stated that the mapping quality decreases when the robot moves with high speed [12]. This is because the lidar measurements become less accurate and the odometry errors increase. The author also says that the Hector SLAM algorithm is more robust to high speed than the other two algorithms [12].

Despite the importance of this issue, there is a need for investigating the impact of robot speed on the performance of different SLAM algorithms in environments with sharp edges. Therefore, the aim of this experiment is to systematically evaluate how different speeds of the robot and different SLAM algorithms affect the quality of the 2D map, particularly around sharp edges.

1.3 Objectives

The objectives of this project are as follows:

a. To analyze how LiDAR self-motion distortions, caused by different robot speeds, affect the performance of various SLAM algorithms.

b. To evaluate the performance of three specific SLAM algorithms (GMapping, Hector SLAM, and Google Cartographer) under different robot speeds (0.07m/s, 0.14m/s, and 0.22m/s) using a Turtlebot3 Burger.

c. To calculate the map accuracy by measuring the length of predefined walls and the angle of predefined corners in the generated 2D LiDAR maps, and compare these measurements with the real-world dimensions.

1.4 Scope of the Project

The scope of this project is as follows:

a. SLAM Algorithms: The project will focus on three specific SLAM algorithms: GMapping, Hector SLAM, and Google Cartographer.

b. Robot and Speeds: The project will use a Turtlebot3 Burger and will test three different robot speeds: 0.07m/s, 0.14m/s, and 0.22m/s.

c. Environment: The project will be conducted in the Control System Lab, located in the Faculty of Manufacturing and Mechatronic Engineering Technology at UMPSA.

d. Map Comparison: The project will involve a visual comparison of all generated maps, as well as a comparison of measurements taken from the maps with real-world dimensions.

e. Error Metrics: The project will calculate several error metrics, including absolute error, MAE, MSE, RMSE, and MAPE, to evaluate the accuracy of the maps.

CHAPTER 2

Literature Review

2.1 LiDAR Self-Motion Distortion

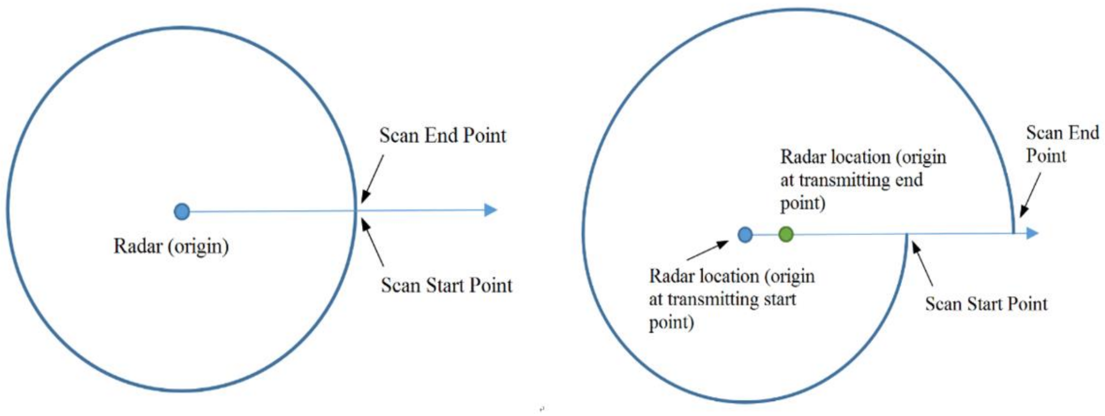

Scan End Point
Radar (origin)
Scan Start Point
Radar location (origin at transmitting end point)
Scan End Point
Radar location (origin at transmitting start point)
Scan Start Point

Figure 2.1: 360-degree mechanical LiDAR self-motion distortion schematic.
Source: https://www.livoxtech.com/showcase/211220

LiDAR, which stands for Light Detection and Ranging, is a method that measures the distance and direction to nearby objects. It works by emitting laser beams and then measuring the time it takes for the beams to bounce back after hitting an object. This information is used to determine the relative location of a vehicle to an obstacle [4].

When a sufficient number of laser beams are emitted, the points at which they bounce back form what is known as a point cloud [13]. This point cloud is used to create a three-dimensional representation of the surrounding environment. However, for most LiDAR systems, even though the laser beams are transmitted and received rapidly, each point in the point cloud is not generated at the exact same moment [10]. Typically, the data accumulated within a 100ms period (which corresponds to a typical frequency of 10Hz) is output as one frame of the point cloud.

Based on Figure 2.1, if the absolute location of the LiDAR device, or the vehicle on which the LiDAR is mounted, changes during this 100ms period, the coordinate system of each point in this frame of the point cloud will be different [10]. This means that the frame of point cloud data will appear “distorted” and will not accurately represent the detected environmental information [9]. This can be likened to taking a photo with a shaky hand, which results in a blurred image. This phenomenon is referred to as the self-motion distortion of LiDAR.

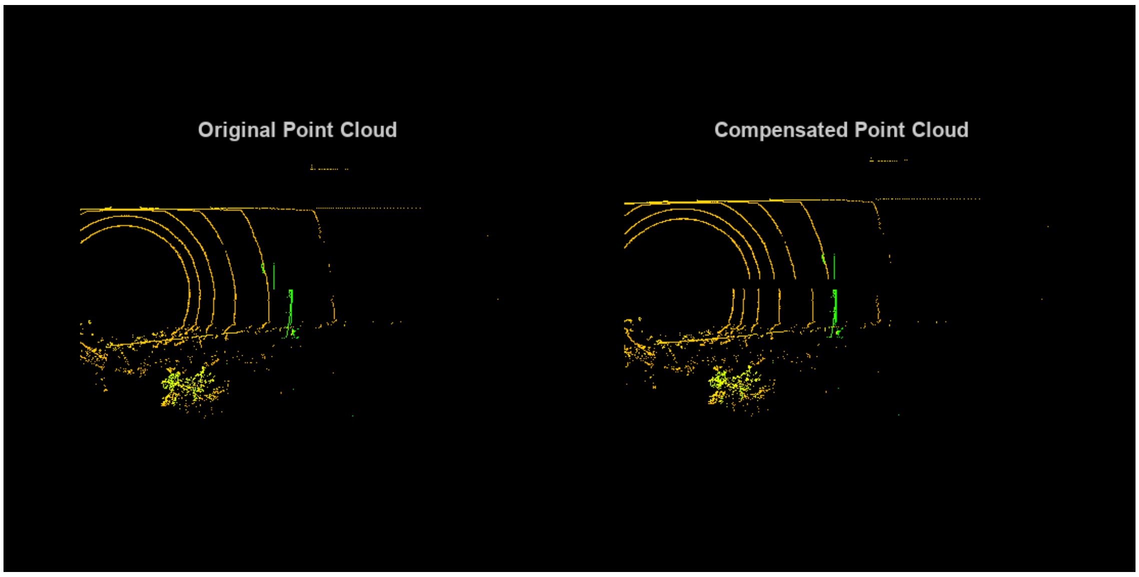


Figure 2.2: Motion compensation in 3D LiDAR point clouds.
Source: https://www.mathworks.com/help/lidar/ug/motion-compensation-in-lidar-point-cloud.html

Figure 2.2 shows the effect of motion distortion in 3D LiDAR point clouds. The self-motion distortion in a LiDAR point cloud is closely tied to the scanning method used. For instance, a traditional 360-degree mechanical LiDAR scans points around its centre for each frame, typically over a period of 100ms. When the LiDAR device or the vehicle it’s mounted on is stationary, the starting and ending points of the scan align well because the coordinate origin remains the same [8]. However, when the LiDAR or vehicle is moving, self-motion distortion occurs [7]. This results in a distortion of the data for one round of scanning, causing the surrounding points to no longer match, as different points have different coordinate origins [7], [8].

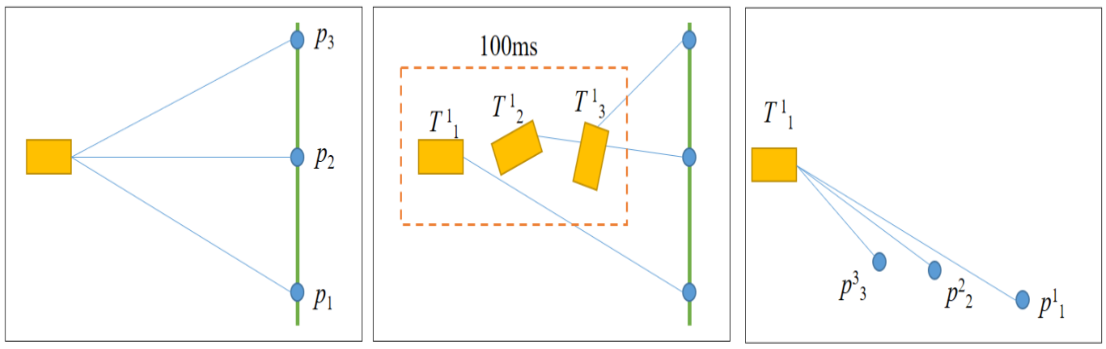


Figure 2.3: Point cloud coordinate system changes.

Source: https://www.livoxtech.com/showcase/211220

To see deeper into the concept, the self-motion distortion observed in the LiDAR point cloud fundamentally arises from each point in a frame having its own different coordinate system. Consider the Figure 2.3 where p1 to p3 represent three points scanned consecutively by a LiDAR device. These points are aligned in a straight line in the actual environment. However, if the LiDAR device experiences significant motion within a single frame, it scans these three points at three different attitudes, as shown in the middle figure. As a result, in the final point cloud that is obtained (shown in the rightmost figure), these three points exist in different coordinate systems and no longer appear to be aligned in a straight line. This is the core of the phenomenon known as LiDAR self-motion distortion.

2.2 Literature Review

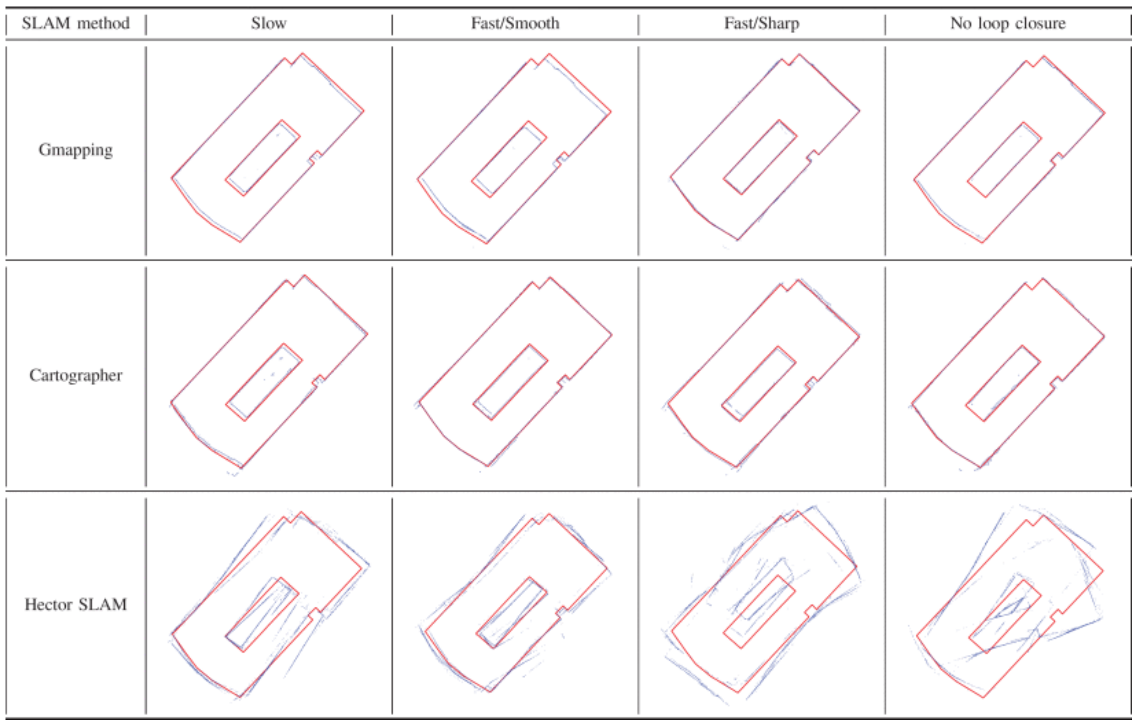
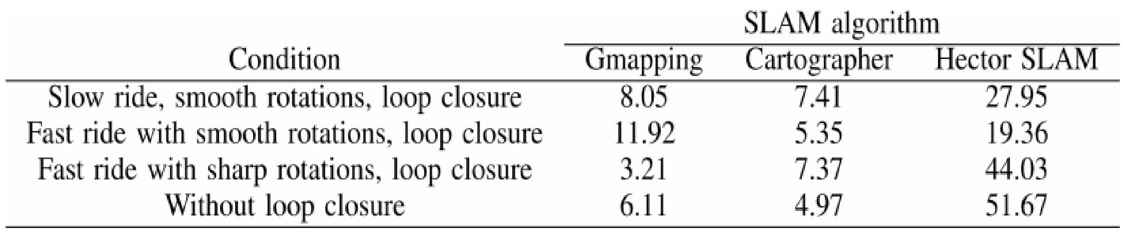


Figure 2.4: Map comparison constructed from SLAM algorithms (in blue colors) and ground truth (in red color).

Source: Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth | IEEE Conference Publication | IEEE Xplore (ump.edu.my)

Based on the paper "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth" written by Rauf Yagfarov, Mikhail Ivanou and Ilya Afanasyev, who are researchers from Innopolis University, Russia, they conducted an experiment on comparative analysis of three ROS-based 2D SLAM algorithms: Google Cartographer, GMapping and Hector SLAM, using a metric of average distance to the nearest neighbour (ADNN). The authors used a mobile robot equipped with a 2D lidar and a high-precision laser tracker to construct and evaluate maps in a static indoor environment [12].

The paper shows that Google Cartographer produces the most accurate maps in most scenarios, followed by GMapping and Hector SLAM, as shown in Figure 2.4. The authors explain that the reason for this is that Cartographer and GMapping use odometry and loop closure to correct the localization and mapping errors, while Hector SLAM relies only on lidar data [12].



Condition	SLAM algorithm		
	Gmapping	Cartographer	Hector SLAM
Slow ride, smooth rotations, loop closure	8.05	7.41	27.95
Fast ride with smooth rotations, loop closure	11.92	5.35	19.36
Fast ride with sharp rotations, loop closure	3.21	7.37	44.03
Without loop closure	6.11	4.97	51.67

Figure 2.5: Map comparison: Error calculated with ADNN metrics for SLAM methods relative to the ground truth, in cm.

Source: Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth | IEEE Conference Publication | IEEE Xplore (ump.edu.my)

Furthermore, this paper also shows that the speed of the robot affects the map quality, as faster and sharper movements introduce more noise and distortion in the lidar scans, resulting in larger ADNN errors, as shown in the Figure 2.5 [12]. The speed of the robot affects the map result because it influences the quality of the LiDAR data. When the robot moves faster and turns sharper, the lidar scans become noisier and more distorted, which makes it harder for the SLAM algorithms to match them with the previous scans or the submaps [10]. This leads to larger errors in the robot pose estimation and the map construction, which are measured by the ADNN metric. Therefore, slower and smoother movements result in more accurate maps [12].

The paper concludes that Google Cartographer is the best algorithm for generating 2D maps with a lidar mounted on a mobile robot. The paper also proposes to extend the comparison method to 3D SLAM algorithms and to add more metrics for a deeper analysis [12]

The paper titled "Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review" and was published in the 2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST). The authors are Yi Kiat Tee and Yi Chiew Han from Curtin University Malaysia, Miri, Malaysia. The authors aim to review and compare the common 2D SLAM systems in an indoor static environment, using the ROS-based SLAM libraries on an experimental mobile robot equipped with a 2D LIDAR module, IMU and wheel encoders. [14]

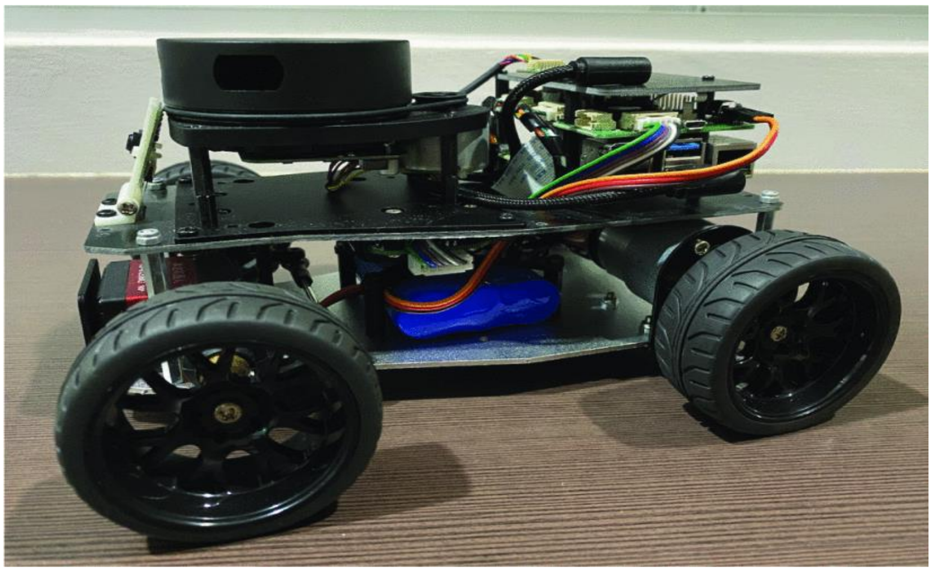


Figure 2.6: Overview of the experimental robot.

Source: [Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review | IEEE Conference Publication | IEEE Xplore (ump.edu.my)]

Figure 2.6 shows the robot that is used by the authors in their experiment. The mobile robot, which utilizes an Ackermann steering system, is equipped with a Raspberry Pi 4 (8GB RAM) that operates on Ubuntu 16.04 and comes with ROS-Melodic pre-installed as the primary controller. It also features a 2D LIDAR module (A1M8) from SLAMTEC, a power distribution board with an IMU soft-mounted on it, wheel encoders on the rear wheels, and a Raspberry Pi camera for environmental visualization. Additionally, it has a receiver for remote control via a mobile controller, as shown in Figure 2.6. The robot is remotely operated over WiFi using another Ubuntu device running ROS. Furthermore, the maps created are displayed using RViz (ROS Visualization). The paper presents a review and comparison of three common 2D SLAM (Simultaneous Localization and Mapping) systems in an indoor static environment, namely GMapping, Hector-SLAM, and Google Cartographer. The paper uses a ROS-based experimental mobile robot to test the performance of the SLAM systems in both simulated environment (Simple Two-Dimension Robot, STDR) and real-world scenarios. The paper evaluates the accuracy, computational complexity, and applicability of the SLAM systems under different circumstances. [14]

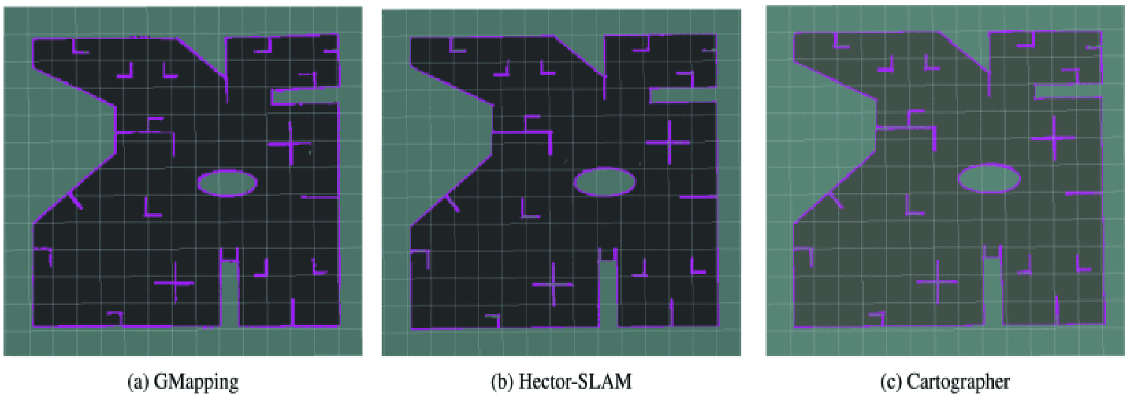


Figure 2.7: Mapping results of (a) GMapping (b) Hector-SLAM (c) Cartographer in simulated environment.

Source: Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review | IEEE Conference Publication | IEEE Xplore (ump.edu.my)

All of the constructed maps from simulation in this paper as in Figure 2.7 have given the ideal results, and it is mainly due to the reason that they are free of errors from the sensors and environment. However, it is obvious that both Hector and Cartographer outperform GMapping by observing the maps in grid view. The accuracy for simulated environment in this paper is determined in terms of location of scan points on the grid points as compared to the truth map.

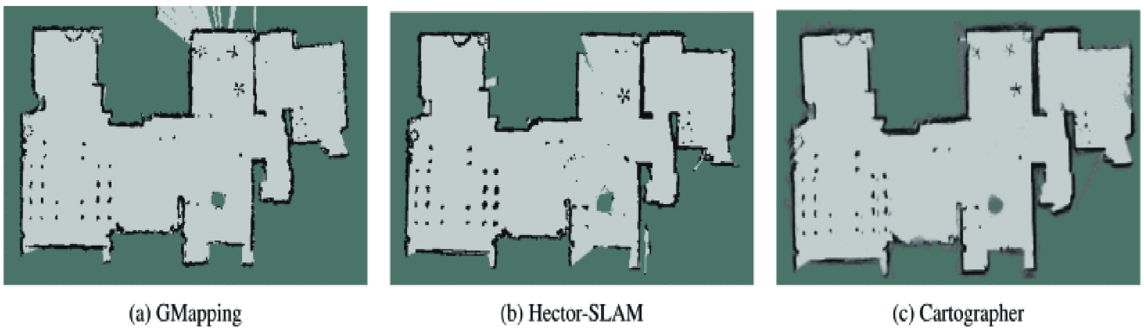
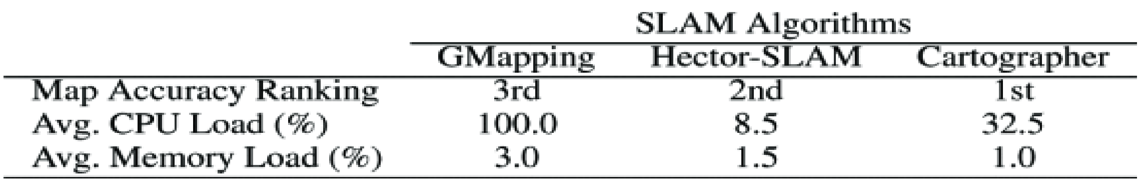


Figure 2.8: Mapping results of (a) GMapping (b) Hector-SLAM (c) Cartographer in real-world environment.

Source: [Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review | IEEE Conference Publication | IEEE Xplore (ump.edu.my)]

As illustrated in Figure 2.8, the paper shows that all three SLAM systems can construct accurate maps in a real flat indoor environment, but they have their own different strengths and weaknesses. GMapping has high map precision but suffers from high computational cost and particle depletion in large and complex environments. Hector-SLAM does not rely on odometry data but is sensitive to wrong LIDAR information and performs poorly in low scan-rate situations. Google Cartographer has slightly higher accuracy overall and can handle large and complex environments with real-time loop closure detection, but has lower map resolution and requires an IMU for orientation estimation. [14]

Table 2.1: Map comparison: Accuracy ranking and CPU load of each algorithms.



	SLAM Algorithms		
	GMapping	Hector-SLAM	Cartographer
Map Accuracy Ranking	3rd	2nd	1st
Avg. CPU Load (%)	100.0	8.5	32.5
Avg. Memory Load (%)	3.0	1.5	1.0

Source: [Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review | IEEE Conference Publication | IEEE Xplore (ump.edu.my)]

Moreover, the CPU and memory load while running each of the SLAM algorithms in real-world environment were examined from the experimental robot car by the authors, and summarised in Table 2.1. The average readings were determined in a small area of a typical house by considering only the computational power required by the algorithms, excluding all the other robot processes, such as sensors processing, robot controlling, etc. The results from the Table 2.1 showing that Cartographer requires the least computational power, following by Hector-SLAM and GMapping, due to the fact that GMapping utilized particle filter algorithm to localize itself, which need a lot of computational resources to carry out. [14]

The paper concludes that the selection of the most appropriate SLAM system depends on the intended application and the environment characteristics. The paper also suggests some potential directions for future optimization and improvement of the SLAM systems, such as tuning the system parameters, removing the motion distortion, and simplifying the source code.

Another research paper entitled "Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique". It was published in the 2019 International Electronics Symposium (IES) and is available on IEEE Xplore. The authors of the paper are Syahrul Fajar Andriawan Eka Wijaya, Didik Setyo Purnomo, Eko Budi Utomo, and Muhammad Akbaryan Anandito. They are affiliated with the Department of Mechanical and Energy, Universitas Negeri Surabaya, Indonesia. [15]

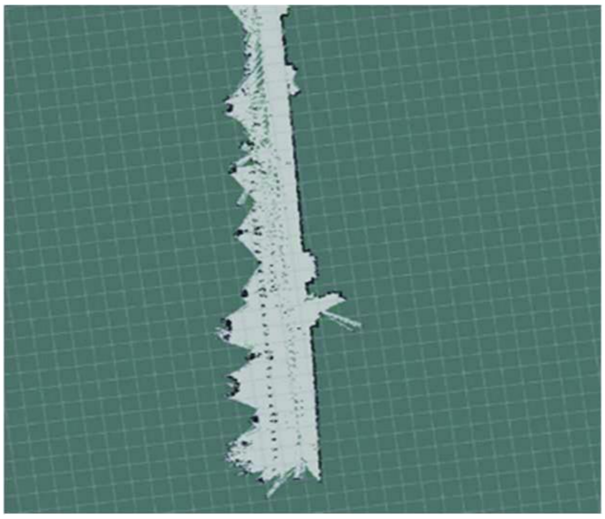
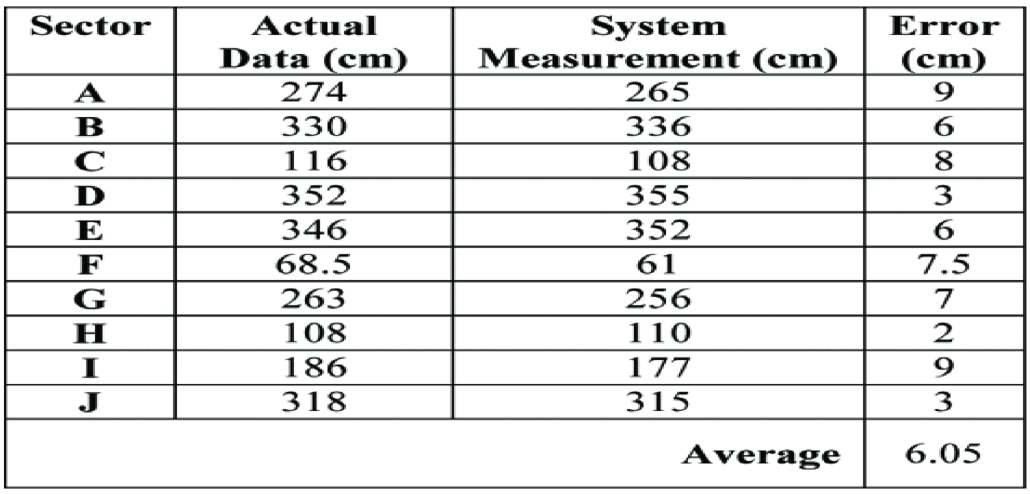


Figure 2.9: Example of Mapping Results Using the Gridmap Occupancy Method.

Source: [Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique | IEEE Conference Publication | IEEE Xplore (ump.edu.my)]

The paper proposes a method for mapping an indoor environment using a differential drive mobile robot equipped with a LIDAR sensor. The method uses the occupancy grid map algorithm and the Hector SLAM technique to generate a 2D map of the environment in real time, as illustrated in Figure 2.9.

Table 2.2: Mapping Result.



Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	274	265	9
B	330	336	6
C	116	108	8
D	352	355	3
E	346	352	6
F	68.5	61	7.5
G	263	256	7
H	108	110	2
I	186	177	9
J	318	315	3
		Average	6.05

Source: Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique | IEEE Conference Publication | IEEE Xplore (ump.edu.my)

The paper reports the results of an experiment conducted in a corridor inside a building. The paper compares the map generated by the proposed method with the actual environment and calculates the mapping error for different sectors. The paper shows that the proposed method successfully mapped the environment with an average mapping error of 6.05 cm, as shown in Table 2.2. The paper also explains the factors that affect the mapping accuracy, such as the sensor resolution, the robot motion, and the environmental features. The paper discusses the advantages and limitations of the proposed method, such as the low computational cost, the high accuracy, and the inability to detect transparent objects.[15]

Lastly, the paper concludes that the proposed method is effective and efficient for mapping an unknown indoor environment using a LIDAR sensor. The paper suggests some possible improvements for future work, such as using a 3D LIDAR sensor, incorporating other sensors, and applying the method to outdoor environments. [15]

Furthermore, the paper “Analysis of Mobile Robot Indoor Mapping using GMapping Based SLAM with Different Parameter” was published in the IOP Conference Series: Materials Science and Engineering in 2019. The authors are W.A.S Norzam, H.F.Hawari, and K.Kamarudin from Malaysia. The authors use the GMapping algorithm, which is a SLAM method based on Rao-Blackwellized Particle Filter, to create a 2D grid map of two different indoor environments using a mobile robot equipped with a laser range finder sensor. The paper investigates the effects of three parameters: robot speed, map update interval, and particle filter, on the mapping quality and time. The map accuracy is calculated when the robot finished the mapping. [16]

The equation below was used to determine the accuracy of the map based on the size of the real map layout, which x represents the total length of the map created by the robot, in m, while y is the total length of the real map, in m.

$$Accuracy\ (\%) = \frac{x}{y} \times 100$$

The results in this paper were obtained from the SLAM mapping of two different indoor areas, which is Lab A and Lab B. Each of the area mappings is done in three trials with the different parameter of mobile robot speed, mapping delay and particle filter as shown in table 1.

Table 2.3: Mapping parameter for three trial.

Trial	1	2	3
Robot speed (m/s)	0.1	0.133	0.161
Map update interval (s)	5	1	0.1
Particle filter	30	30	15

Source: MSEM7051037.pdf (iop.org)

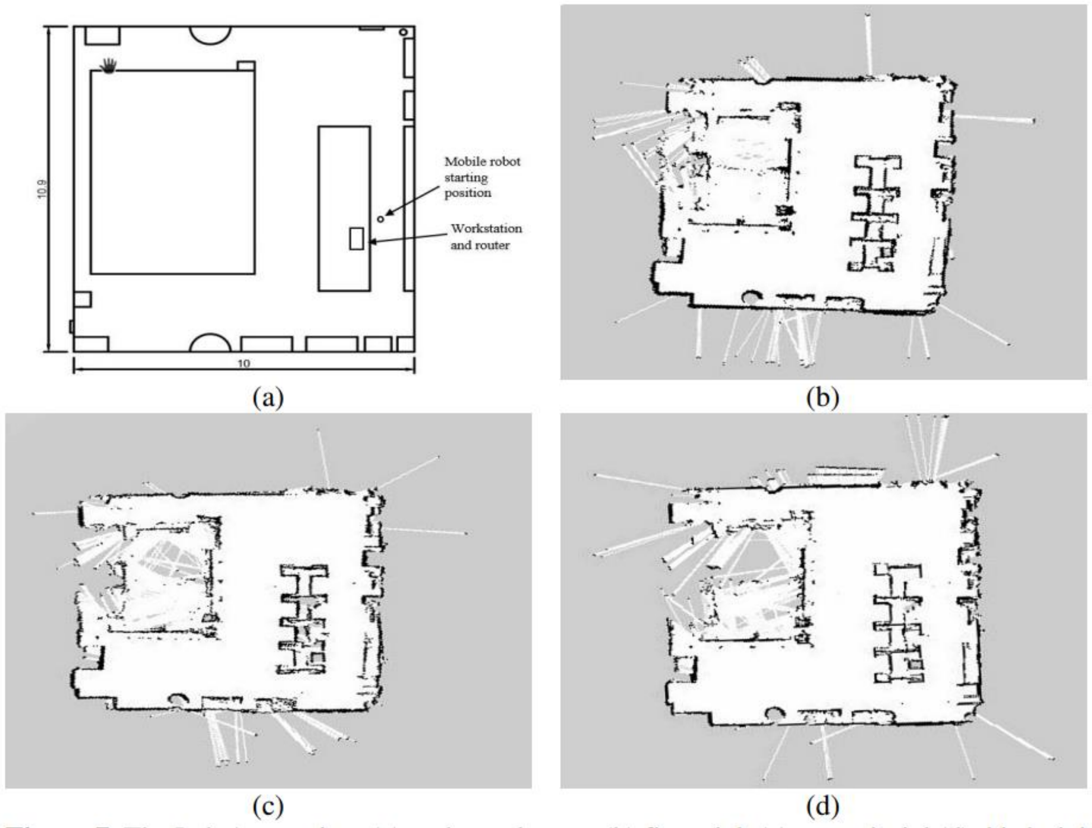


Figure 2.10: The Lab A mapping, (a) real map layout, (b) first trial, (c) second trial (d) third trial

Source: MSEM7051037.pdf (iop.org)

Table 2.4: Mapping result for Lab A.

Trial	1	2	3
Map size parameter (m)	38.8592	38.1266	36.1965
Time taken to complete mapping (min : s)	24 : 33	12 : 19	6 : 14
Map accuracy (%)	92.96	91.21	86.59

Source: MSEM7051037.pdf (iop.org)

Three trials were conducted for the mapping of Lab A. The characterization lab, which was the area of mapping, measures 108m^2 with a total length of 41.8m. The starting point for the robot was kept consistent across all three trials. Figure 2.10 presents the actual layout of the map alongside the three trial mappings of Lab A. A visual comparison of the three results with the actual map reveals an increase in unmapped areas, making the maps more incomplete as the trials progress. The outcomes of the mapping are documented in Table 2.4. It was observed that the accuracy of the mapping declined, despite a reduction in the time taken for mapping. The second trial produced the best map for Lab A, as the time required to complete the map was nearly half of that in the first trial, even though the accuracy was lower by 1%. [16]

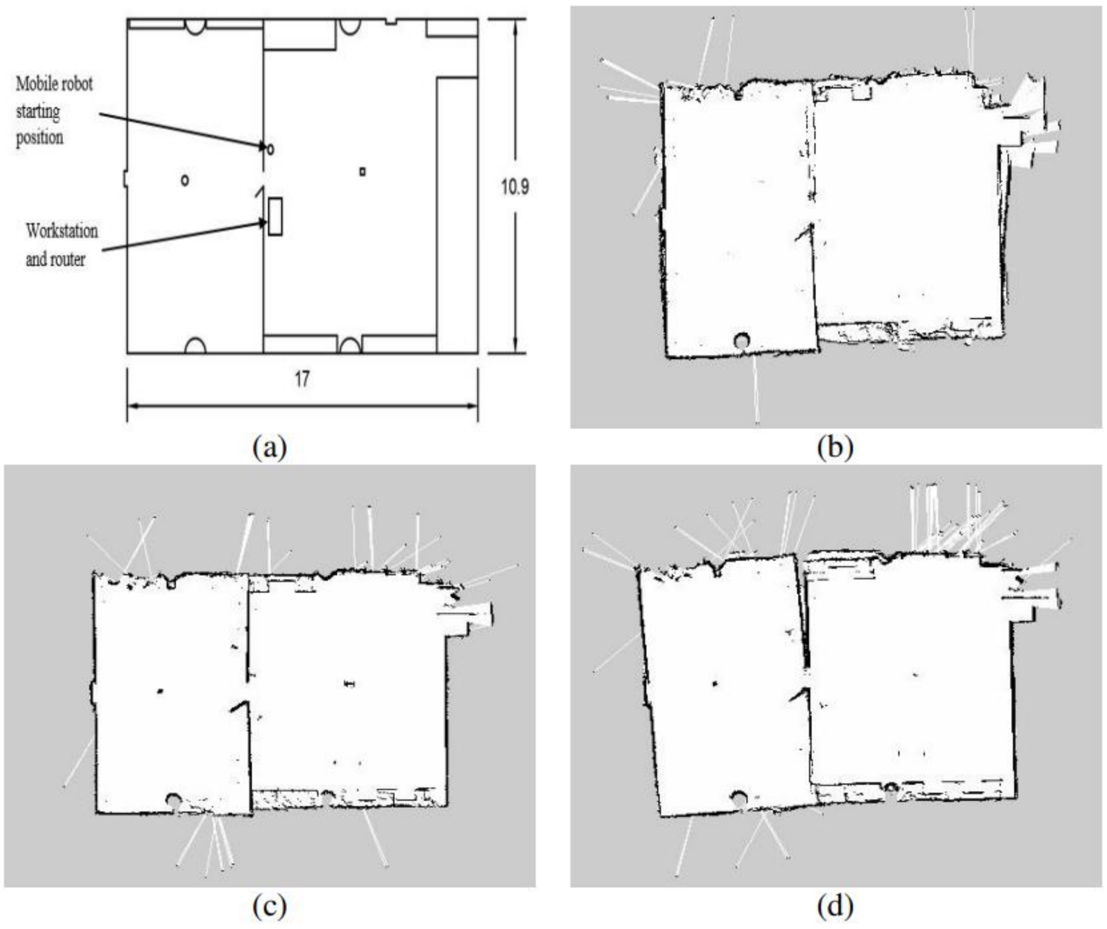


Figure 2.11: The Lab B mapping, (a) real map layout, (b) first trial, (c) second trial (d) third trial

Source: MSEM7051037.pdf (iop.org)

Table 2.5: Mapping result for Lab B.

Trial	1	2	3
Map size parameter (m)	54.5963	54.0231	51.8253
Time taken to complete mapping (min : s)	40 : 22	19 : 35	15 : 01
Map accuracy (%)	98.19	97.16	93.21

Source: MSEM7051037.pdf (iop.org)

The mapping area for Lab B is 184m^2 with a total length of 55.6m. This lab comprises two rooms, divided by a wall with an open door in the centre. A significant distinction from Lab A is that Lab B is larger in size but has fewer features within the area. Figure 2.11 illustrates the actual map layout and the three trials of Lab B mapping. The outcomes of the Lab B mapping are detailed in Table 2.5.

The authors explain that the higher robot speed and lower map update interval result in higher computational load for the mobile robot, which causes some data loss and errors in the map. The authors also explain that the faster robot speed means that the sensor has less time to scan the environment, which leads to more holes and incomplete areas in the map. The paper concludes that the GMapping algorithm is able to produce a good quality map for indoor environments, but the parameters need to be adjusted according to the size and features of the environment. [16]

2.3 Proposed Methodology

In this study, three of the most widely used SLAM algorithms, namely GMapping, Hector SLAM, and Google Cartographer, will be employed. The choice of these algorithms is motivated by their current popularity in the field. For example, the paper "Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth" written by Rauf Yagfarov, comparing these three SLAM algorithms on different speed of the mobile robot [12]. The Turtlebot3 Burger, which is readily available in the lab, will be utilized for the mapping task. This approach not only saves time but also avoids the unnecessary effort of constructing a new robot, adhering to the principle of not "reinventing the wheel".

The study aims to investigate the effect of LiDAR self-motion distortion on the quality of mapping. To achieve this, the robot will be operated at three different speeds: 0.7m/s (low), 0.14m/s (medium), and 0.22m/s (high). The maximum speed of 0.22m/s corresponds to the top speed of the Turtlebot3 Burger. By applying these varying speeds across the three different SLAM algorithms, the impact of LiDAR self-motion distortion on each algorithm can be thoroughly examined. This methodology allows for a comprehensive understanding of how changes in robot speed influence the performance of different SLAM algorithms in mapping tasks.

In this project, upon completion of the mapping process, measurements will be taken of the lengths of ten predefined walls and corners in both the RViz and the real world. These measurements will then be compared, and the absolute error will be calculated. This approach is inspired by the methodologies proposed by Yi Kiat Tee and Yi Chiew Han in their paper "Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review" [14], as well as Syahrul Fajar Andriawan Eka Wijaya and Didik Setyo Purnomo in their paper "Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique" [15].

The Mean Absolute Error (MAE) will be computed for all the proposed methods, following the approach outlined by Kristian Wahlqvist in the paper "A Comparison of Motion Priors for EKF-SLAM in Autonomous Race Cars" [11]. The MAE is a measure of the average magnitude of the errors in a set of predictions, without considering their direction. It’s the average over the test sample of the absolute differences between prediction and actual observation where all individual differences have equal weight [11].

In addition to MAE, the results will be further analysed using other statistical measures such as the Mean Squared Error (MSE), the Root Mean Squared Error (RMSE), and the Mean Absolute Percentage Error (MAPE). These additional metrics will provide a more comprehensive understanding of the accuracy of the SLAM methods. MSE is the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. RMSE is the square root of the mean of the square of all of the error. MAPE, on the other hand, expresses accuracy as a percentage, and is defined as the mean absolute percentage deviation of the actual values from the predictions. These metrics will offer more nuanced insights into the accuracy and performance of the SLAM methods under investigation.

2.4 Occupancy Grid Map

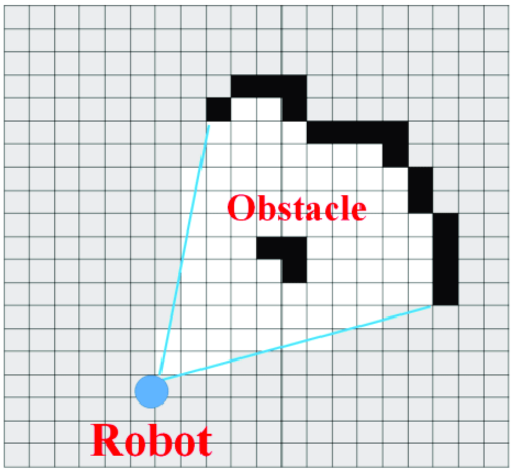


Figure 2.12: Occupancy Grid Map (OGM).

Source: https://www.researchgate.net/publication/321326800_A_25D_Map-Based_Mobile_Robot_Localization_via_Cooperation_of_Aerial_and_Ground_Robots

Occupancy Grid Maps (OGMs), as shown in Figure 2.12, are a popular method in robotics for representing the environment. In an OGM, the environment is divided into a grid, and each cell in the grid corresponds to a small area in the real world [17].

Each cell in the grid stores a value representing the probability that the corresponding area in the environment is occupied by an obstacle [18]. This provides a probabilistic representation of the environment, which is particularly useful in situations where the sensor data is uncertain or noisy [17], [18]. The most common type of OGMs are 2D maps that represent a slice of the 3D world. These maps are often used in mobile robots for tasks such as navigation and obstacle avoidance [19]. In summary, OGMs provide a way for robots to understand and navigate their environment, even when the sensor data is uncertain or noisy.

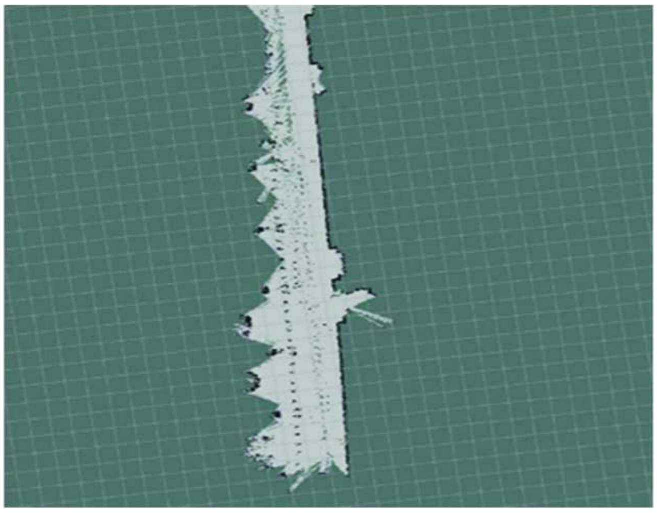


Figure 2.13: An example of an OGM.

Source: https://ieeexplore-ieee-org.ezproxy.ump.edu.my/document/8901657

In Figure 2.13, spaces detected by a sensor as empty are represented as white areas, while spaces identified as occupied or containing obstacles are shown as black areas. For the creation of an occupancy grid map, an occupancy probability is assigned to each cell, under the simplifying assumption that each cell on the map is independent. This assumption, while not entirely accurate, particularly for adjacent cells representing the same physical object, allows for the simplification of the occupancy grid map algorithm without a significant increase in errors [20]. The probability assigned to the map is influenced by the location history of the robot, and the sensor's data. Given sensor data from t_1 to t, $z_{1:t}$ and the poses from t_1 to t, $x_{1:t}$ of the sensor, the map of the environment can be estimated by

$$p(m|z_{1:t}, x_{1:t}) = \prod_i p(m_i|z_{1:t}, x_{1:t})$$

Since the probability of the whole map is broken up into the product of probability that a certain cell is occupied or free, given the sensor data and the pose information, then the individual cell is represented by a binary random variable. Then, in order to estimate the state of this variable, the binary Bayes Filter can be used, which is optimized for estimating the probability of a binary random variable. In order to use binary Bayes Filter, a static state must be assumed, which means that the state doesn't change over time. Something which is once free will stay free, something which is occupied will stay occupied. Therefore,

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, z_{1:t-1}, x_{1:t})\ p(m_i|z_{1:t}, x_{1:t})}{p(z_t|z_{1:t-1}, x_{1:t})} \quad Bayes\ rule$$

For the next step, the Markov assumption can be exploited. Markov assumption is something which says if the state of the world at a given point in time is known, then what happened in the past is independent from what happened in the future. Therefore, the equation becomes:

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, x_t)\ p(m_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \quad Markov\ rule$$

Then, the equation involves three elements now. apply Bayes rule to the first element and the equation will become:

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t)\ p(z_t|x_t)\ p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i|x_t)\ p(z_t|z_{1:t-1}, x_{1:t})}$$

The term $p(m_i|x_t)$ is the probability of a cell being occupied given the robot pose, but it doesn't really help since we have no idea on what the robot sees or senses, unless the robot is standing in that cell. Therefore, by independence assumption, the x_t in this term can be ignored.

$$p(m_i|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t)\ p(z_t|x_t)\ p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i)\ p(z_t|z_{1:t-1}, x_{1:t})}$$

Then, for the opposite event, it can be done by using exactly the same method as above. Therefore, the probability of an individual cell that is not being occupied becomes:

$$p(-m_i|z_{1:t}, x_{1:t}) = \frac{p(-m_i|z_t, x_t)\ p(z_t|x_t)\ p(-m_i|z_{1:t-1}, x_{1:t-1})}{p(-m_i)\ p(z_t|z_{1:t-1}, x_{1:t})}$$

By computing the ratio of both probabilities, the equation becomes:

$$\frac{p(m_i|z_{1:t}, x_{1:t})}{p(-m_i|z_{1:t}, x_{1:t})} = \frac{\frac{p(m_i|z_t, x_t)\ p(z_t|x_t)\ p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i)\ p(z_t|z_{1:t-1}, x_{1:t})}}{\frac{p(-m_i|z_t, x_t)\ p(z_t|x_t)\ p(-m_i|z_{1:t-1}, x_{1:t-1})}{p(-m_i)\ p(z_t|z_{1:t-1}, x_{1:t})}}$$

From this equation, the terms that are independent of m_i and $-m_i$ can be ignored, and the equation becomes:

$$\frac{p(m_i|z_{1:t},x_{1:t})}{p(-m_i|z_{1:t},x_{1:t})}=\frac{p(m_i|z_t,x_t)\,p(m_i|z_{1:t-1},x_{1:t-1})\,p(-m_i)}{p(-m_i|z_t,x_t)\,p(-m_i|z_{1:t-1},x_{1:t-1})\,p(m_i)}$$

$$=\frac{p(m_i|z_t,x_t)}{1-p(m_i|z_t,x_t)}\frac{p(m_i|z_{1:t-1},x_{1:t-1})}{1-p(m_i|z_{1:t-1},x_{1:t-1})}\frac{1-p(m_i)}{p(m_i)}$$

From the equation above, the first term, $\frac{p(m_i|z_t,x_t)}{1-p(m_i|z_t,x_t)}$ is the term that uses the current observation, so it can be clearly seen that all the equations here contain the current observation z_t, which means what is the robot seeing right now. The second term, $\frac{p(m_i|z_{1:t-1},x_{1:t-1})}{1-p(m_i|z_{1:t-1},x_{1:t-1})}$ is the recursive term which is only related to the state estimate of that same cell but using only the old data, up to t-1. The third term, $\frac{1-p(m_i)}{p(m_i)}$ is just prior information, which means what can be said about the map without having seen anything or having been there.

From here, since the equation above is in ratio form, so called the odds ratio. Odds ratio is simply the probability of the event divided by the probability of the opposite event. In order to change the equation above from ratio form above back to the probability form, the method below can be used:

$$Odds(x)=\frac{p(x)}{1-p(x)}$$

$$p(x)=Odds(x)-Odds(x)\,p(x)$$

$$p(x)\big(1+Odds(x)\big)=Odds(x)$$

$$p(x)=\frac{Odds(x)}{1+Odds(x)}$$

$$p(x)=\frac{1}{1+\frac{1}{Odds}}$$

By using this equation, the probability of an individual cell that is being occupied becomes:

$$p(m_i|z_{1:t},x_{1:t})=1+\frac{1-p(m_i|z_t,x_t)}{p(m_i|z_t,x_t)}\frac{1-p(m_i|z_{1:t-1},x_{1:t-1})}{p(m_i|z_{1:t-1},x_{1:t-1})}\frac{p(m_i)}{1-p(m_i)}$$

This is the formula used to estimate the state of an individual cell, whether it is free or occupied. But, for the reasons of efficiency, one performs the calculations in the log odds notation. The log-odds notation computes the logarithm of the ratio of probabilities. Log-odds ratio is defined as:

$$l(x) = \log \frac{p(x)}{1 - p(x)}$$

And with the ability to retrieve $p(x)$,

$$p(x) = 1 - \frac{1}{1 + \exp l(x)}$$

By using the log-odds, the equation can become like this:

$$\frac{p(m_i|z_{1:t}, x_{1:t})}{p(-m_i|z_{1:t}, x_{1:t})} = \frac{p(m_i|z_t, x_t)\, p(m_i|z_{1:t-1}, x_{1:t-1})\, p(-m_i)}{p(-m_i|z_t, x_t)\, p(-m_i|z_{1:t-1}, x_{1:t-1})\, p(m_i)}$$

$$l(m_i|z_{1:t}, x_{1:t}) = l(m_i|z_t, x_t) + l(m_i|z_{1:t-1}, x_{1:t-1}) - l(m_i)$$

Or in short:

$$l_{t,i} = inv\_sensor\_model(m_i, x_t, z_t) + l_{t-1,i} - l_0$$

This equation can greatly reduce the computational resources since it only involves plus and minus. By estimating the occupancy status of each grid cell, a whole OCM can be obtained by using the sensor’s observation and robot pose. This is used in various applications such as autonomous navigation and mapping. Several papers in the provided abstracts discuss different approaches to building occupancy grid maps. Rogers et al. propose a collaborative exploration strategy using resource-constrained robots to build an occupancy grid map [21]. Ye et al. present a method that uses LiDAR and visual inputs to obtain surrounding occupancy information and build a local occupancy grid map [19]. Sunil et al. on the other hand, introduce a feature-based map fusion approach for collaborative mapping using probabilistic occupancy grid maps [22]. Lastly, a paper by the Yunfan Ren presents ROG-Map, a uniform grid-based occupancy grid map that integrates LiDAR technology for robotic navigation [23].

2.5 SLAM Algorithm

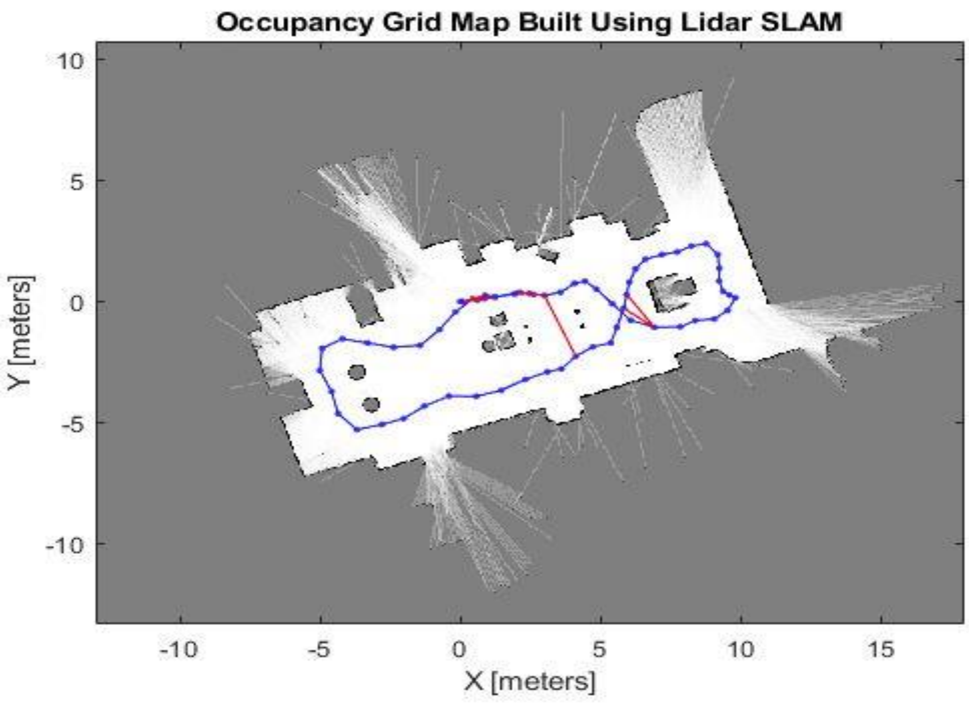


Figure 2.14: Mapping using SLAM algorithm.
Source: https://www.mathworks.com/discovery/slam.html

SLAM, short for Simultaneous Localization and Mapping, is a technique used in robotics and autonomous vehicles [6]. Its purpose is to create or update a map of an environment that's unknown while keeping track of the robot's location within that environment, as shown in Figure 2.14 [5], [6]. SLAM consists of two main parts:

a. Localization: This is about figuring out the robot's location within the map at any given moment. It's usually done using sensor data and a motion model.

b. Mapping: This involves creating a map of the environment. The map could take various forms such as grid maps, landmark-based maps, or even raw sensor readings.

Sensors like cameras, LiDAR, radar, sonar, GPS, and IMU (Inertial Measurement Unit) are used by SLAM algorithms to gather data [24]. One of the challenges with SLAM is that errors in localization and mapping can build up over time, leading to what's known as "drift". To address this issue, various SLAM algorithms have been developed, including Extended Kalman Filter (EKF) SLAM, FastSLAM, GraphSLAM, and others

[25]. SLAM is a crucial technology in many applications, including autonomous vehicles, robot navigation, augmented reality, and more [24], [25].

2.5.2 SLAM Algorithm: GMapping

The Gmapping algorithm is a reliable method based on a particle filter, particularly the Rao-Blackwellized particle filter (RBPF). This algorithm is good at handling data correlation issues and provides a better estimate of the map's likelihood. Gmapping, using a particle filter, effectively addresses nonlinear problems and is a widely used SLAM technique [26]. SLAM algorithms that rely on particle filtering typically demand substantial computational resources and memory. This is primarily because a larger quantity of particles is needed to attain more accurate results. The Rao-Blackwellized Particle Filter (RBPF) based SLAM method was developed as an efficient solution to the SLAM problem. It was subsequently optimized to decrease the number of particles needed, significantly reducing the uncertainty about the robot's pose during the prediction phase of the Particle Filter [14]. This optimization enhances the efficiency and accuracy of the SLAM process.

According to the key idea of RBPF-SLAM, the method is to estimate the probability of joint posterior $p(x_{1:t}, m|z_{1:t}, u_{1:t-1})$ of the robot trajectory $(x_{1:t} = x_1, x_2, \dots, x_t)$ and grid map of the environment, m, where the sensor observations and odometry measurements are denoted by $(z_{1:t} = z_1, z_2, \dots, z_t)$ and $(u_{1:t} = u_1, u_2, \dots, u_t)$respectively [27]. The posterior probability makes use of the following factorization using the Bayes' Rule:

$$\begin{aligned} &p(x_{1:t}, m|z_{1:t}, u_{1:t-1}) \\ &= p(m|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, u_{1:t-1}) \end{aligned}$$

The technique commonly referred to as Rao-Blackwellization enables the estimation of the robot's trajectory prior to the computation of the map based on that trajectory. This process can be understood as a two-part problem, where $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ represents the localization problem and $p(m|x_{1:t}, z_{1:t})$ signifies the mapping problem [27]. Moreover, the Rao-Blackwellized Particle Filter (RBPF) method incorporates the widely used particle filtering algorithm known as Sample Importance Resampling (SIR). The procedure of the SIR algorithm can be encapsulated in four key steps, which are sampling, importance weighting, resampling, and update of map. This combination enhances the efficiency and accuracy of the SLAM process [27].

A. Sampling: New particles, denoted as $(x_t^{(i)})$, are generated based on the previous set of particles, $(x_{t-1}^{(i)})$, which are sampled from the proposal distribution, π. The latest observations are then integrated into this process to enhance the sampling process.

B. Importance Weighting: The importance weight $w_t^{(i)}$ of each current particle $x_t^{(i)}$ is computed by referring to the importance sampling principle as follow:

$$w_t^{(i)} = \frac{p\left(x_{1:t}^{i} \middle| z_{1:t}, u_{1:t-1}\right)}{\pi\left(x_{1:t}^{i} \middle| z_{1:t}, u_{1:t-1}\right)}$$

Where π(.) is the proposal distribution (probabilistic odometry motion model).

C. Resampling: Particles with lower weight scores are selectively eliminated and substituted with resampled particles. Despite this process, the number of particles remains constant, all the particles will have equal weight after resampling.

D. Update of Map: The trajectory $x_{1:t}^{(i)}$ and history of observations $z_{1:t}$ are brought into $p\left(m^i \middle| x_{1:t}^{l}, z_{1:t}\right)$ to compute the map estimate.

The Rao-Blackwellized particle filter SLAM method, while effective, has several drawbacks that can significantly impact its accuracy over time. The primary issues arise from the increasing number of particles and the resampling cycles, leading to a degradation in the quality of particles, inaccurate laser scan models, and reduced practicality [27]. Specifically, a large number of particles require high computational power, which can be inefficient. Additionally, frequent resampling can increase the chance of eliminating good particles from the filter, resulting in a scarcity of particles [27].

Finally, as referenced in [28], the GMapping method is most effective in planar environments and is heavily dependent on the presence and adequacy of precise odometry data. As a result, this approach is not suitable for unstructured environments that lead to extreme pitch and roll motions of the robot or carrier [28].

2.5.3 SLAM Algorithm: Hector SLAM

Hector SLAM utilizes a scan matching algorithm to align consecutive LiDAR scans, which is crucial for accurate mapping and localization. This scan matching process is independent of odometry data, making Hector SLAM robust against situations where the robot moves at high speed or experiences wheel slippage [14].

The scan matching algorithm in Hector SLAM is based on the Gauss-Newton method. The Gauss-Newton method is an optimization algorithm used to solve non-linear least squares problems [15]. In the context of scan matching, the goal is to find the optimal pose (a combination of translation and rotation) of the robot, which corresponds to the rigid body transformation $\xi = (p_x, p_y, \psi)^T$from the robot to the prior map [15]. In other words, the rigid body transformation is determined to fit the laser beam optimally, that minimizes ξ as computed below:

$$\xi^* = argmin_\xi \sum_{i=1}^{n} [1 - M(S_i(\xi))]^2$$

Where robot pose, $\xi = (p_x, p_y, \psi)^T$, n = total number of scans, $S_i(\xi)$ is the impact coordinates of i^{th} scan in world frame, and M is the map value at coordinates given by S_i. In addition to the scan matching algorithm, Hector SLAM also uses a trajectory matching algorithm that employs the Iterative Closest Point (ICP) method and a reference frame to evaluate and correct the drifting of the system [20]. This helps to improve the error accumulation problem, further enhancing the accuracy and robustness of Hector SLAM.

The implementation of Hector-SLAM on mobile robots necessitates the use of modern LiDAR or Laser Range Finders (LRF). These devices are advantageous due to their low measurement noise and high update rate [20]. Moreover, Hector-SLAM offers added versatility and adaptability for aerial robots, such as quad-copters, and for ground robots navigating uneven terrains, as it does not rely on odometry data [20]. However, the absence of high-rate scans and dependable odometry information can lead to issues such as map distortion and poor localization.

2.5.4 SLAM Algorithm: Google Cartographer

As the trajectory of robots expands and the volume of data to process escalates, the particle-based approach becomes less feasible due to its high computational demands. Consequently, graph-optimization algorithms emerge as a more reliable and efficient alternative. Google's Cartographer, a SLAM method based on graph-optimization introduced in 2016, exemplifies this approach [29]. This graph-based method operates over a collection of nodes and edges, representing the robot's poses and features, and constraints derived from onboard observations, respectively. The algorithm can be viewed as two distinct yet interconnected approaches to the SLAM problem, comprising the front-end (Local SLAM) and back-end (Global SLAM) [30]. The front-end performs scan-matching (scan-to-submap) for loop closure and estimates the robot's pose. In contrast, the back-end regularly optimizes the pose based on the constructed graph to minimize loop-closing constraints, thereby ensuring minimal successive pose constraints or pose errors [30].

Local SLAM generates a series of submaps which are meant to be locally consistent but drift over time. It also creates the local trajectory. Global SLAM runs in separate threads and finds loop closure constraints between submaps generated by local SLAM [30]. It executes loop closure by scan matching scans of sensor data against the submaps. It also incorporates other sensors to attempt to get the most consistent global solution. It includes loop closing and removes the errors that have accumulated in each submap that are part of a loop, by using post graph optimization [31].

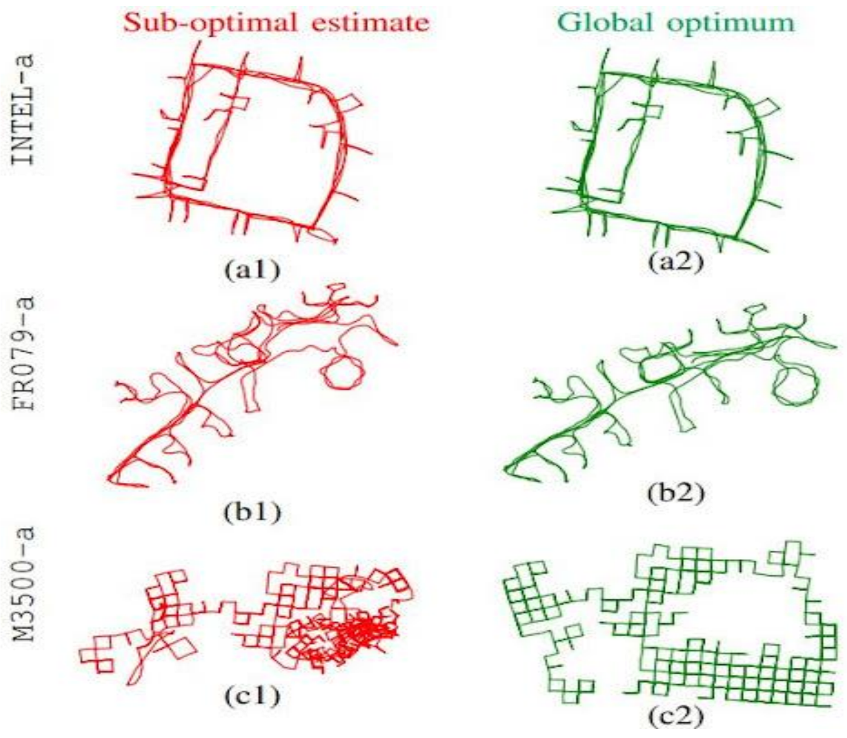


Figure 2.15: Post graph optimization.

Source: http://sidra2016.dia.uniroma3.it/towards-robust-2d-pose-graph-optimization-via-convex-relaxation/

Figure 2.15 shows the post graph optimization which is a technique for solving the SLAM problem, which involves estimating the pose (position and orientation) of a robot and the structure of the environment from sensor data, and thus the distorted map can be adjusted as shown in Figure 2.15. The primary functions of the global approach in SLAM involve performing loop closure detection and loop closing. This is achieved by identifying constraints between non-successive poses in real time and optimizing the pose graph to minimize loop closing errors. In the context of robotics SLAM, loop closure refers to the event where a robot returns to a location it has previously visited [30]. When this occurs, the relative transformation between the current scan and the previously visited scan is estimated to update the robot's position. Post graph optimization represents the SLAM problem as a graph, where the nodes are the robot poses and the edges are the constraints between them [31]. The constraints can be derived from odometry, loop closures, or other sources of information. The goal of post graph optimization is to find the optimal configuration of the nodes that minimizes the error of the constraints [29]. Post graph optimization can handle large-scale problems with thousands of nodes and edges, and can produce accurate and consistent maps of the environment [29].

Prior to the introduction of Google's Cartographer in 2016, real-time detection of loop-closing constraints was a challenging task in SLAM methods. This was largely due to the fact that loops are closed immediately upon revisiting a location, while optimization is completed every few seconds [29]. Consequently, loop closure scan-matching must occur faster than the addition of new scans. The real-time processing was made possible by using a branch-and-bound method and several precomputed grids for each completed submap, as described by the author in [29].

CHAPTER 3

METHODOLOGY

3.1 Project Flow Chart

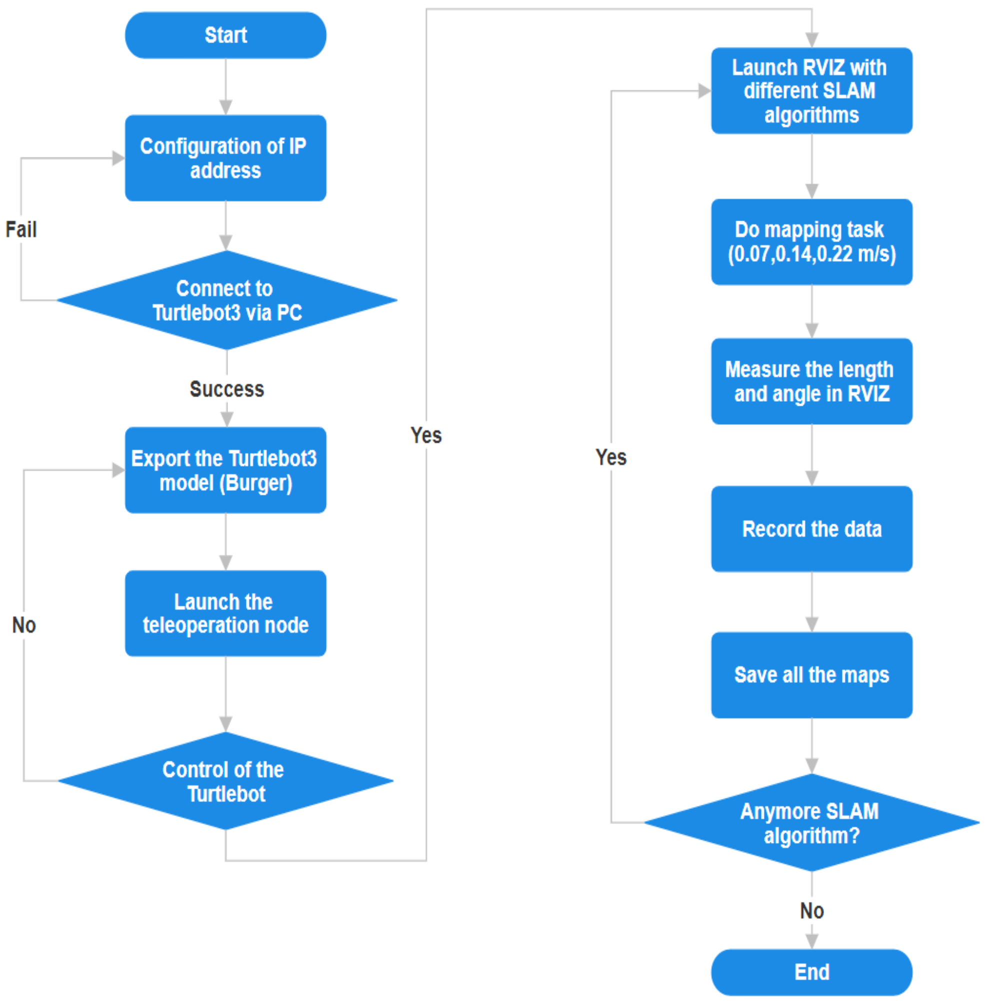

Start
Configuration of IP address
Fail
Connect to Turtlebot3 via PC
Success
Export the Turtlebot3 model (Burger)
No
Launch the teleoperation node
Control of the Turtlebot
Yes
Launch RVIZ with different SLAM algorithms
Yes
Do mapping task (0.07,0.14,0.22 m/s)
Measure the length and angle in RVIZ
Record the data
Save all the maps
Anymore SLAM algorithm?
No
End

Figure 3.1: Project flow chart.

The project overall methodology, as shown in Figure 3.1, begins with the configuration of the IP addresses of both the TurtleBot3 Burger and the remote PC. The TurtleBot3 Burger is then connected to the remote PC using the correct IP address. If the connection fails, the process is repeated until a successful connection is established. Once connected, the correct TurtleBot3 model, in this case, the ‘Burger’, is exported. The TurtleBot3 teleoperation node for the TurtleBot3 robot is then launched using the turtlebot3_teleop_key. The robot’s movements are controlled using the WASD keys on the PC keyboard. If the robot fails to move, the process is repeated from the model exportation step. The RVIZ is then launched to run the SLAM node for the TurtleBot3 robot, with the first SLAM algorithm being GMapping. The robot is maintained at a speed of 0.07m/s, and the process is repeated at speeds of 0.14m/s and 0.22m/s respectively. The ‘Measure’ tool in RVIZ is used to calculate the length of predefined features in the 2D LiDAR map, and the data is recorded. All angles in the 2D LiDAR map are measured and the data is recorded. All maps are saved using the save map node. The step of launching RVIZ to the map saving step are repeated using different SLAM algorithms, namely Hector SLAM and Google Cartographer. This methodology ensures a comprehensive and thorough approach to the project.

3.2 Light Detection and Ranging (LiDAR)

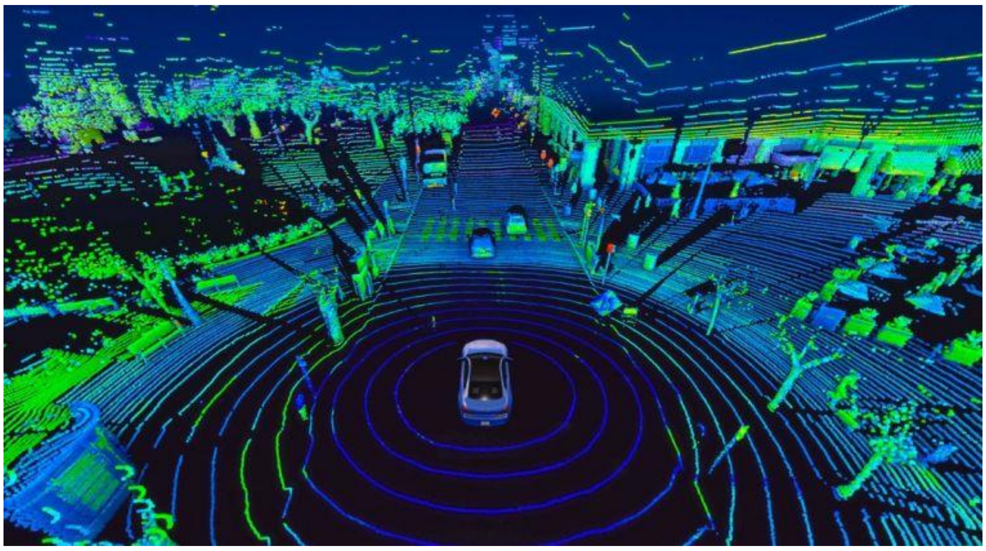


Figure 3.2: LiDAR used in autonomous car.

Source: https://www.gim-international.com/content/article/multibeam-lidar-for-mobile-mapping-systems

As outlined in the initial section, the Light Detection and Ranging sensor (LiDAR) sensor, as shown in Figure 3.2, is mainly used for autonomous vehicle (AV) nowadays. LiDAR is a 2D or 3D imaging system capable of collecting distance measurements primarily through time-of-flight (ToF) methods [32]. It operates on a principle similar to radar, but with a distinction: instead of relying on radio waves, it utilizes light-based remote sensing, typically in the form of lasers. In its fundamental configuration, a laser serves as the energy source, emitting light in all directions. Subsequently, the sensor records and assesses the reflected signals, along with their distinctive properties, to derive information about the surroundings and distances. Lidar scanning components have the capability to discharge numerous pulses, sometimes numbering in the hundreds of thousands per second, and leverage these pulses to obtain measurements [13]. Time-of-flight techniques involve measuring the duration it takes for a signal to leave the sensor, interact with a surface, and return to the sensor. By calculating the time taken, the system determines the distance covered by the signal until it encounters a reflective surface, as illustrated in Figure 3.2.

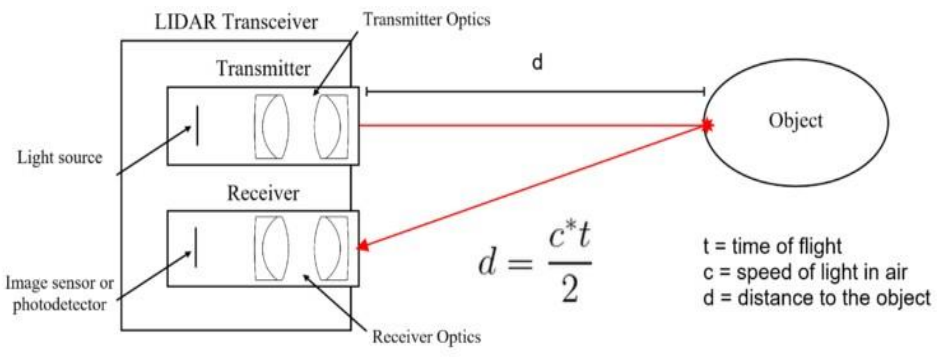


Figure 3.3: Working principle of the LiDAR sensor.

Source: https://how2electronics.com/how-to-use-tfmini-s-lidar-distance-sensor-with-arduino/

Figure 3.3 illustrates the ToF technique used by the LiDAR sensor, in which the sensor emits a laser beam from its light source and initiates a time counter. Subsequently, when the photodetector detects the incoming signal, the system records the entire duration of the laser's flight [33]. The distance then can be found in most typical LiDAR applications, as shown below:

$$d = \frac{c \times t}{2}$$

where d is the distance travelled by the LiDAR signal, t is the measured time of flight, and c is the speed of light [32].

By precisely measuring the time it takes for the emitted laser beam to travel to objects in the environment and back to the LIDAR sensor, the sensor can determine the distances to these objects [32]. These distance measurements are collected at various angles or directions, allowing the LIDAR sensor to create a detailed and accurate map of the surroundings. The resulting map is essentially a representation of the objects, obstacles, and structures in the environment, allowing for tasks such as obstacle avoidance, mapping of indoor or outdoor spaces, and localization of autonomous systems like robots and self-driving vehicles [6]. LIDAR's ability to provide accurate distance information is a fundamental factor in its effectiveness for a wide range of applications that require precise mapping and environmental awareness [13].

3.3 Robot Operating System (ROS)

Figure 3.4: Logo of Robot Operating System (ROS).
Source: https://www.pilz.com/en-INT/products/robotics/ros-modules

In response to the growing complexity of tasks in robot applications, there is an undeniable and substantial rise in computational challenges and hardware abstractions. To tackle these issues, the Robot Operating System (ROS), as shown in Figure 3.4, has emerged as a widely adopted software platform in the field of robotics. ROS serves as an environment that is widely employed to streamline and support the development of robot projects [34][35]. ROS simplifies the process of creating modules for autonomous robots, offering more than just a framework; it's often referred to as a meta-operating system for robots. This is because ROS encompasses a comprehensive range of services, including hardware abstraction, low-level device control, implementation of commonly-used functions, inter-process communication, and package management. The ROS platform is at the forefront of providing a complete suite of communication and control mechanisms for robots, thus reducing the significant time and effort traditionally required for programming. ROS has found widespread use in designing control systems for various objects, including self-driving cars and autonomous robots, as demonstrated in references [36], [37], and [38].

The Simultaneous Localization and Mapping (SLAM) algorithm is poised to be a crucial element in robot navigation and is gaining prominence in the field of robot applications. It serves as a method that enables the creation of a step-by-step map while simultaneously recognizing obstacles in unfamiliar surroundings. (See, for example, references [5], [6], and [34]). Utilizing data from an Inertial Measurement Unit (IMU) or encoders via the ROS platform [39], the mobile robot creates an incremental map of its surroundings. ROS seamlessly incorporates GAZEBO, a 3D dynamic simulator, along with RVIZ, a ROS visualization tool, to enable simulation and real-time monitoring of the robot's operations. Additionally, a virtual environment in GAZEBO is meticulously constructed based on calculations derived from diverse environmental data, ensuring the direct observation of robot activities across multiple environmental scenarios. Research illustrated succinctly in references [34] and [37] underscores the significance of ROS in the realms of mapping and navigation for autonomous robots.

3.4 TurtleBot3

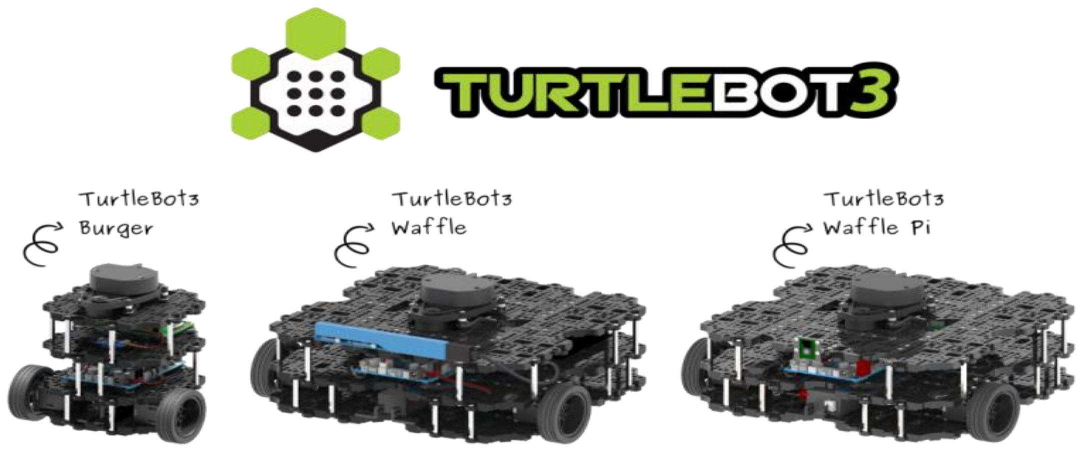


Figure 3.5: Turtlebot3 Burger, Waffle, and Waffle Pi.
Source: https://www.turtlebot.com/turtlebot3/

TurtleBot3 is a popular open-source mobile robot platform developed by the Open Source Robotics Foundation (OSRF) in collaboration with the Robotis Company, which comes with three models, namely Turtlebot3 Burger, Turtlebot3 Waffle, and Turtlebot3 Waffle Pi, as shown in Figure 3.5. It is designed for education, research, and hobbyist purposes, and it's part of the broader TurtleBot family [40]. Turtlebot3 is a mobile robot that can move using wheels. It has two cameras that can be used for object detection and depth measurement. The colour selection method, HSV, is used for object detection, while the binocular disparity method is used for measuring object distances using stereo cameras [41]. It is also equipped with a distance sensor (LIDAR) for object detection and localization [40]. Turtlebot3 is designed to provide a platform for learning and working with the Robot Operating System (ROS). It is built around low-cost computing and sensing hardware, making it an affordable option for those interested in ROS. TurtleBot3 can be simulated in various environments using tools like Gazebo. Simulations enable users to test and develop algorithms before deploying them on the physical robot. TurtleBot3 supports SLAM techniques, allowing the robot to navigate and map its environment autonomously [42].

3.4.1 TurtleBot3 Burger

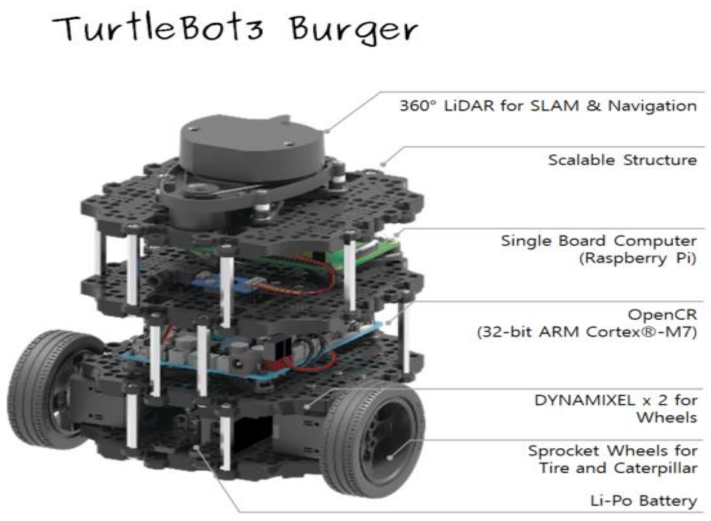

TurtleBot3 Burger
360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer (Raspberry Pi)
OpenCR (32-bit ARM Cortex®-M7)
DYNAMIXEL x 2 for Wheels
Sprocket Wheels for Tire and Caterpillar
Li-Po Battery

Figure 3.6: Turtlebot3 Burger Model.
Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/

Figure 3.6 shows a Turtlebot3 Burger which is used for mapping in this project. The TurtleBot3 Burger model is one of the TurtleBot3 variants, and it's known for its compact and lightweight design, making it highly portable and suitable for a variety of environments. Its small footprint allows it to navigate through confined spaces [43]. As shown in Figure 3.6, the features of Turtlebot3 Burger are as below:

a. 360^o LiDAR for SLAM and Navigation: The Burger model typically comes equipped with a 360-degree LiDAR sensor. LiDAR (Light Detection and Ranging) is a sensing technology that uses laser light to measure distances, providing a 360-degree view of the robot's surroundings. This sensor is crucial for Simultaneous Localization and Mapping (SLAM) and navigation tasks, allowing the robot to create a map of its environment and navigate within it [43].

b. Scalable Structure: The term "scalable structure" implies that the TurtleBot3 Burger is designed to be easily adaptable and expandable. Users can potentially add or modify components to suit their specific needs or experiment with different sensors and configurations [41].

c. Single Board Computer (Raspberry Pi): The Burger model often features a single-board computer, commonly a Raspberry Pi. The single-board computer serves as the brain of the robot, running the Robot Operating System (ROS) and providing computational power for various tasks, including sensor data processing, decision-making, and communication [42].

d. OpenCR (32-bit ARM Cortex-M7): The OpenCR (Open-source Control Module for ROS) is a 32-bit ARM Cortex-M7 microcontroller board designed for use in robotics projects. It is often used in TurtleBot3 for low-level control tasks, interfacing with sensors and actuators, and managing the robot's motion [42].

e. Dynamixel x 2 for Wheels: Dynamixel servos are smart actuators often used in robotics applications. In the TurtleBot3 Burger, Dynamixel servos are typically employed as motor actuators for the wheels. These servos are known for their precision and controllability, making them suitable for robotic mobility [40].

f. Sprocket Wheels for Tire and Caterpillar: Sprocket wheels are components that engage with the chain or belt to transfer motion. In the context of TurtleBot3 Burger, sprocket wheels are likely used in conjunction with tires or caterpillar tracks to facilitate the movement of the robot. The specific choice of wheels depends on the model and variant [40].

g. Li-Po Battery: The TurtleBot3 Burger is powered by a Lithium Polymer (Li-Po) battery. Li-Po batteries are commonly used in robotics due to their high energy density, lightweight design, and ability to deliver sufficient power for the robot's operation. The battery provides the necessary electrical energy to power the motors, electronics, and sensors on the TurtleBot3 [40].

TurtleBot3 Burger, with its features and open-source nature, serves as an excellent platform for learning about robotics, exploring ROS, and conducting research in various robotic applications. Users can find detailed documentation and community support on the official TurtleBot website and related forums. The hardware specifications and the dimension and mass of Turtlebot3 Burger are shown in Table 3.1 and Figure 3.7 below respectively.

Table 3.1: Hardware specifications of Turtlebot3 Burger.

Maximum translational velocity	0.22 m/s
Maximum rotational velocity	2.84 rad/s (162.72 deg/s)
Maximum payload	15kg
Size (L×W×H)	138mm × 178mm × 192mm
Weight (+ SBC + Battery + Sensors)	1kg
Threshold of climbing	10mm or lower
Expected operating time	2h 30m
Expected charging time	2h 30m
Single Board Computers	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216MHz, 462 DMIPS)
Actuator	XL430-W250
IMU	Gyroscope 3 Axis Accelerometer 3 Axis
LDS (Laser Distance Sensor)	360 Laser Distance Sensor LDS-01 or LDS-02
Power connectors	3.3V / 800mA 5V / 4A 12V / 1A

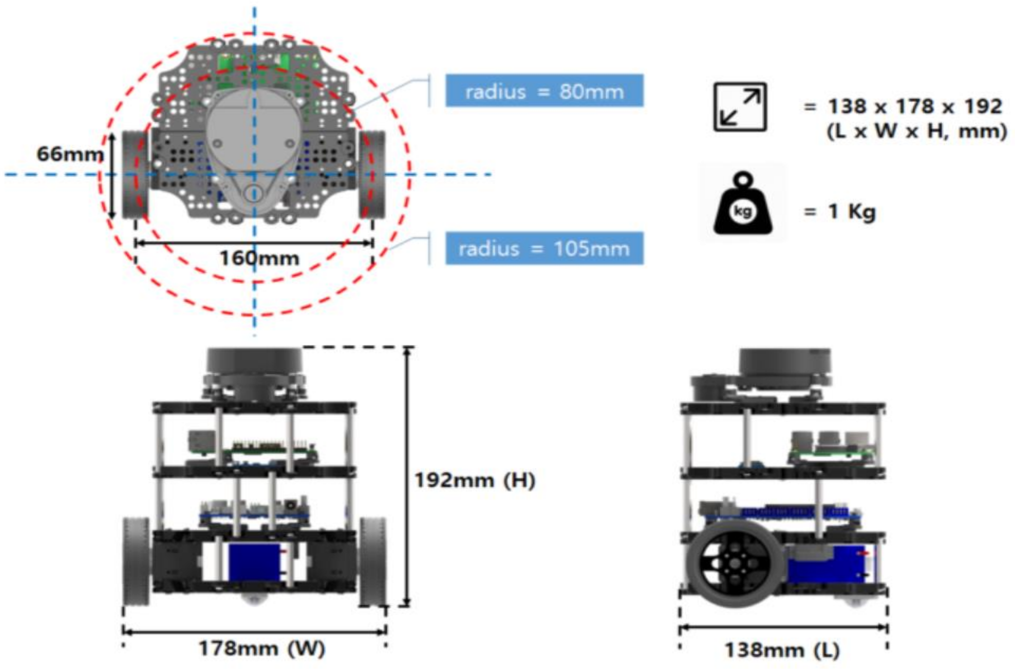


Figure 3.7: Dimension and mass of Turtlebot3 Burger.

Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#features

3.4.2 LDS-01 LiDAR

Figure 3.8: LDS-01 LiDAR sensor.
Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

The LiDAR LDS-01, as shown in Figure 3.8, is a 2D laser scanner which was mounted on the Turtlebot3 models, and capable of sensing 360 degrees [44] [45]. It collects a set of data around the robot to use for SLAM (Simultaneous Localization and Mapping) and Navigation [44], [45]. It's used in various applications, including robotics and autonomous navigation. The LDS-01 is used for TurtleBot3 Burger, Waffle, and Waffle Pi models1. It supports USB interface (USB2LDS) and is easy to install on a PC [44]. It also supports a UART interface for embedded boards1. In addition to ROS, the LDS-01 supports Windows, Linux, and MacOS development environments for general purposes [44]. LiDAR is more accurate than cameras, which means it sticks to virtual boundaries better than camera-based systems. Vacuums using Lidar are much faster at creating the initial map which will usually be completed after one run of user's home [45]. Table 3.2 and Table 3.3 below shows the general specifications and measurement performance specifications of LDS-01 LiDAR sensor respectively.

Table 3.2: General specifications of LDS-01.

Items	**Specifications**
Operating supply voltage	5V DC ±5%
Light source	Semiconductor Laser Diode(λ=785nm)
LASER safety	IEC60825-1 Class 1
Operating current	400mA or less (Rush current 1A)
Scan distance	120mm to 3,500mm
Interface	3.3V USART (230,400 bps) 42bytes per 6 degrees, Full Duplex option
Sampling rate	1.8kHz
Ambient Light Resistance	10,000 lux or less
Dimensions	69.5(W) × 95.5(D) × 39.5(H) in mm
Mass	Under 125g

Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#features

Table 3.3: Measurement Performance Specifications of LDS-01.

Distance Range	120 ~ 3,500mm
Distance Accuracy (120mm ~ 499mm)	±15mm
Distance Accuracy (500mm ~ 3,500mm)	±5.0%
Distance Precision (120mm ~ 499mm)	±10mm
Distance Precision (500mm ~ 3,500mm)	±3.5%
Scan Rate	300±10 rpm
Angular Range	360°
Angular Resolution	1°

Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

3.5 Configuration of IP Address

3.5.1 Connecting TurtleBot3 to External Devices

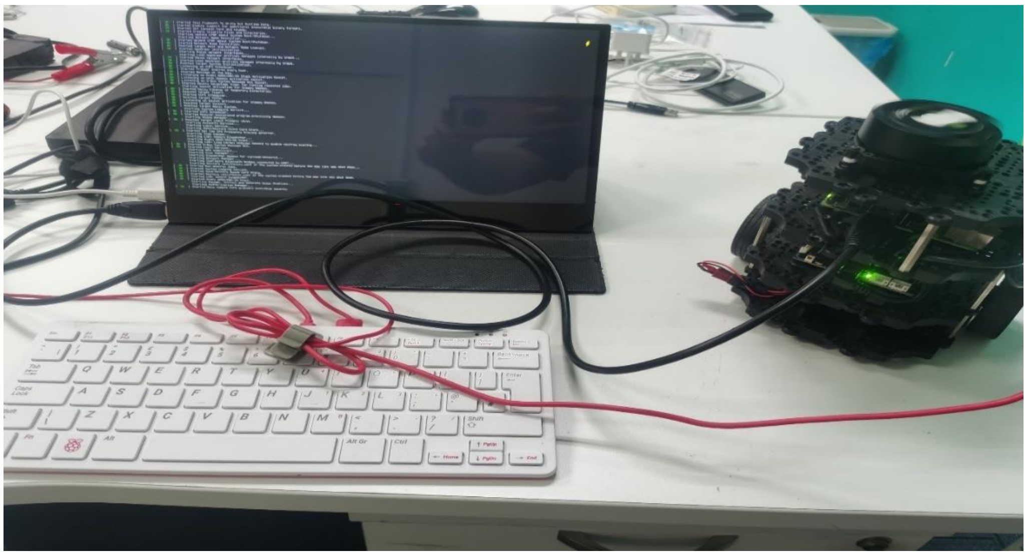
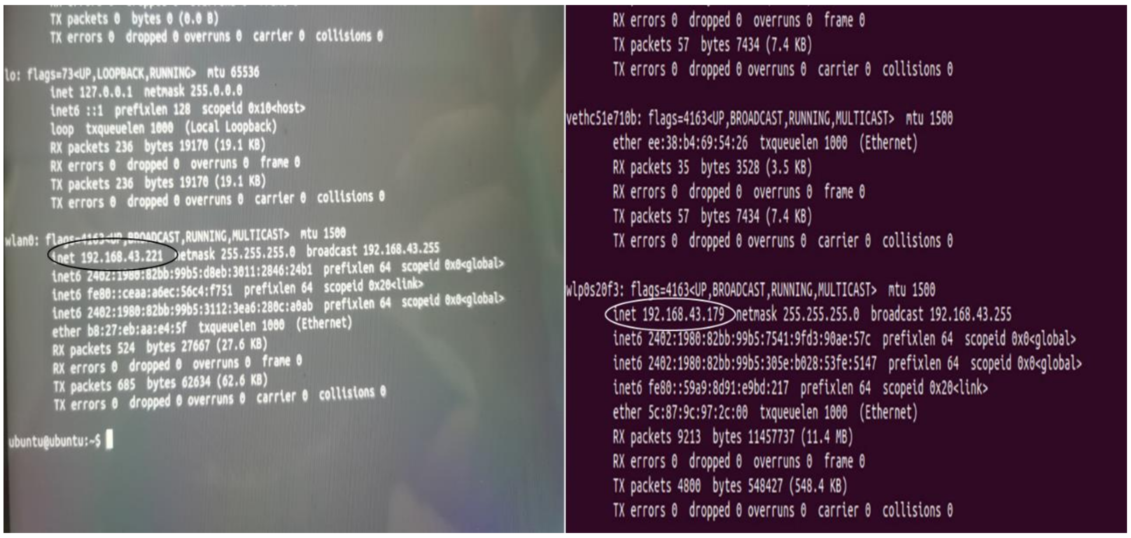


Figure 3.9: Connection of TurtleBot to external monitor and keyboard.

The TurtleBot3 is a highly versatile platform that is equipped with a Raspberry Pi, a compact yet powerful computer. This Raspberry Pi is a fully functional computer, which means it can be connected to a variety of external devices. This includes a monitor, keyboard, and mouse, which can be used to set up and program the robot, as shown in Figure 3.9. This setup process is made even more straightforward due to the Raspberry Pi’s compatibility with these common devices, allowing for easy interaction and programming.

The Raspberry Pi on the TurtleBot3 runs the Ubuntu operating system. Ubuntu is a popular choice for many robotics applications due to its robustness and the large number of compatible software packages available. This means that a wide range of robotics software can be easily installed and run on the TurtleBot3, making it a flexible platform for a variety of robotics applications. The Raspberry Pi on the TurtleBot3 Burger can also be used to run more complex software, such as the Robot Operating System (ROS). Running ROS on the Raspberry Pi allows the TurtleBot3 to perform a variety of tasks, from basic movement and obstacle avoidance to more complex behaviours like mapping and navigation. This makes the TurtleBot3 a powerful tool for exploring a wide range of robotics concepts and technologies.

3.5.2 Obtaining the IP Address



```
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 236  bytes 19170 (19.1 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 236  bytes 19170 (19.1 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.221  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 2402:1980:82bb:99b5:d8eb:3011:2846:24b1  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::ceaa:a6ec:56c4:f751  prefixlen 64  scopeid 0x20<link>
        inet6 2402:1980:82bb:99b5:3112:3ea6:280c:a0ab  prefixlen 64  scopeid 0x0<global>
        ether b8:27:eb:aa:e4:5f  txqueuelen 1000  (Ethernet)
        RX packets 524  bytes 27667 (27.6 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 685  bytes 62634 (62.6 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ubuntu@ubuntu:~$
```

```
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 57  bytes 7434 (7.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

vethc51e710b: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        ether ee:38:b4:69:54:26  txqueuelen 1000  (Ethernet)
        RX packets 35  bytes 3528 (3.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 57  bytes 7434 (7.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlp0s20f3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.43.179  netmask 255.255.255.0  broadcast 192.168.43.255
        inet6 2402:1980:82bb:99b5:7541:9fd3:90ae:57c  prefixlen 64  scopeid 0x0<global>
        inet6 2402:1980:82bb:99b5:305e:b028:53fe:5147  prefixlen 64  scopeid 0x0<global>
        inet6 fe80::59a9:8d91:e9bd:217  prefixlen 64  scopeid 0x20<link>
        ether 5c:87:9c:97:2c:00  txqueuelen 1000  (Ethernet)
        RX packets 9213  bytes 11457737 (11.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 4800  bytes 548427 (548.4 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 3.10: IP address of a) TurtleBot3 and, b) Remote PC

The TurtleBot3 can be connected to external devices such as a monitor, keyboard, and mouse. This allows for direct interaction with the TurtleBot3's onboard computer, the Raspberry Pi. This setup is similar to a traditional desktop computer setup, making it easier to configure and program the robot. Once the TurtleBot3 is connected to the external devices, its IP address can be obtained, as shown in Figure 3.10, which is done using the 'ifconfig' command. This is a standard command in Unix-like operating systems such as Ubuntu. This command displays the network configuration of all network interfaces on the system, including their IP addresses. Based on Figure 3.10, the IP address of the TurtleBot3 is 192.168.43.221, and the IP address of the remote PC is 192.168.43.179.

3.5.3 Changing the IP Address of ROS Master and ROS Hostname

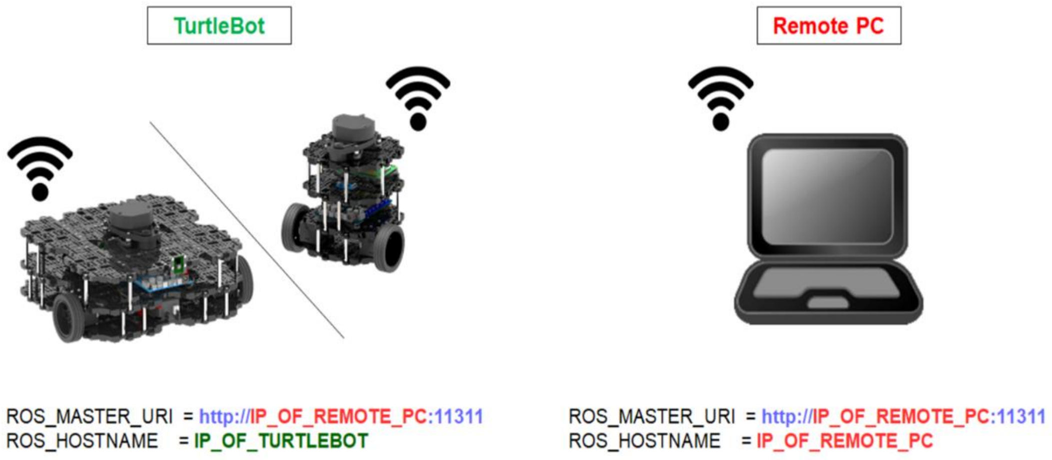
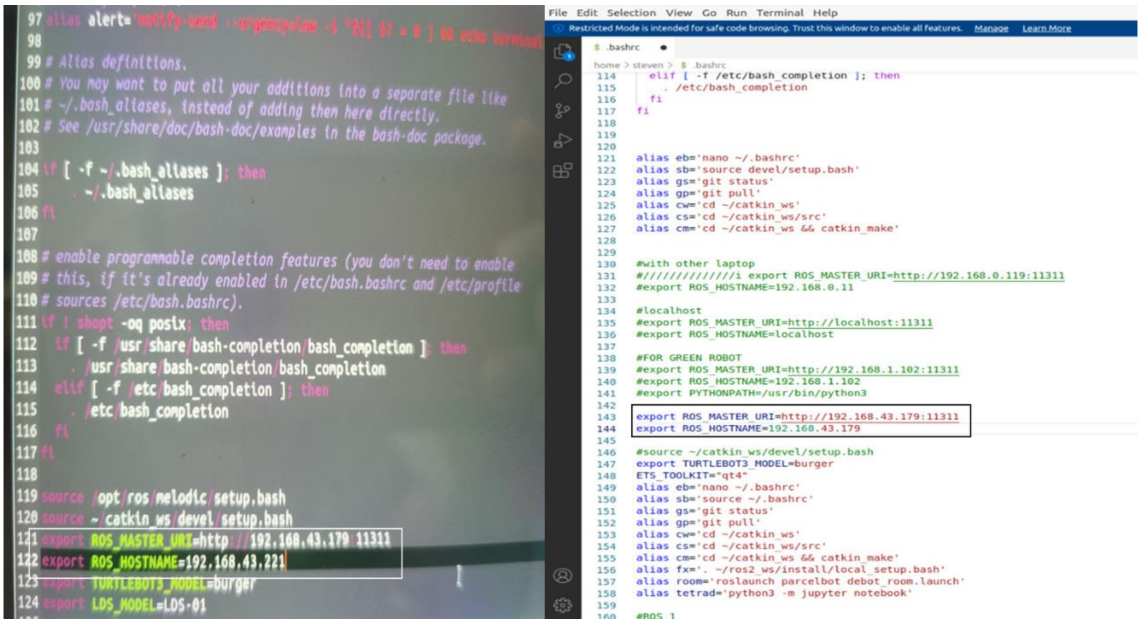

TurtleBot
Remote PC
ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_TURTLEBOT
ROS_MASTER_URI = http://IP_OF_REMOTE_PC:11311
ROS_HOSTNAME = IP_OF_REMOTE_PC

Figure 3.11: Network configuration of TurtleBot and PC.
Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/quick-start/#pc-setup

The process of changing the IP address of the ROS Master and ROS Hostname is typically guided by the Figure 3.11, which shows what the edited '.bashrc' file should look like. This figure serves as a reference, ensuring that the '.bashrc' file is edited correctly. The '.bashrc' file is a crucial component in a Unix-like operating system. It's a script file that gets executed every time a user logs into the system. This file contains a series of configurations that instruct the behaviour and appearance of the terminal session. These configurations can include a variety of features such as colouring, completion, shell history, command aliases, and more. By launching the '.bashrc' file, these configurations are essentially initialized, ensuring that the terminal session behaves according to the predefined settings.

In the context of the TurtleBot3 and the Remote PC, the '.bashrc' file plays an additional role. It's used to set the IP address of the ROS Master and ROS Hostname. The ROS Master is essentially the main hub of a ROS system, and the ROS Hostname is the name of the system in the network. These are set using the ROS_MASTER_URI and ROS_HOSTNAME environment variables respectively. By editing these values in the '.bashrc' file, the TurtleBot3 and the Remote PC are properly configured to communicate with each other.



```
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

source /opt/ros/melodic/setup.bash
source ~/catkin_ws/devel/setup.bash
export ROS_MASTER_URI=http://192.168.43.179:11311
export ROS_HOSTNAME=192.168.43.221
export TURTLEBOT3_MODEL=burger
export LDS_MODEL=LDS-01
```

File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
$ .bashrc
home > steven > $ .bashrc

```
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi



alias eb='nano ~/.bashrc'
alias sb='source devel/setup.bash'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'


#with other laptop
#//////////////i export ROS_MASTER_URI=http://192.168.0.119:11311
#export ROS_HOSTNAME=192.168.0.11

#localhost
#export ROS_MASTER_URI=http://localhost:11311
#export ROS_HOSTNAME=localhost

#FOR GREEN ROBOT
#export ROS_MASTER_URI=http://192.168.1.102:11311
#export ROS_HOSTNAME=192.168.1.102
#export PYTHONPATH=/usr/bin/python3

export ROS_MASTER_URI=http://192.168.43.179:11311
export ROS_HOSTNAME=192.168.43.179

#source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
ETS_TOOLKIT="qt4"
alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'
alias fx='. ~/ros2_ws/install/local_setup.bash'
alias room='roslaunch parcelbot debot_room.launch'
alias tetrad='python3 -m jupyter notebook'

#ROS 1
```

Figure 3.12: Changing of the IP address in '.bashrc' file in a) TurtleBot3, and b) Remote PC.

The ‘.bashrc’ file in both the TurtleBot3’s Raspberry Pi and the Remote PC has been modified to set up the ROS networking environment, as shown in Figure 3.12. This is done by setting the ROS_MASTER_URI and ROS_HOSTNAME environment variables.

For Raspberry Pi of TurtleBot3:

export ROS_MASTER_URI: This line sets the URI (Uniform Resource Identifier) where the ROS Master can be reached. In this case, it’s set to the IP address of the Remote PC. This means that the TurtleBot3’s Raspberry Pi will look for the ROS Master on the Remote PC.

export ROS_HOSTNAME: This line sets the hostname that the TurtleBot3’s Raspberry Pi should use when communicating over the network. It’s set to the IP address of the TurtleBot3 itself. This means that when the TurtleBot3 communicates with other devices on the network (like the Remote PC), it will identify itself using its own IP address.

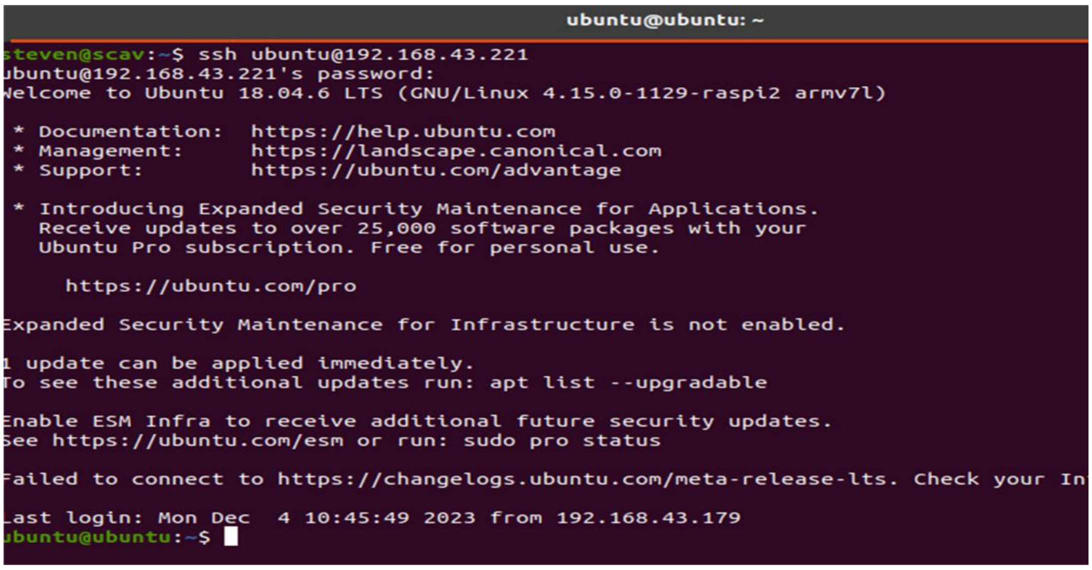
For Remote PC:

export ROS_MASTER_URI: Similar to the TurtleBot3's Raspberry Pi, this line sets the URI where the ROS Master can be reached. However, since the Remote PC is where the ROS Master is running, it's set to the IP address of the Remote PC itself.

export ROS_HOSTNAME: This line sets the hostname that the Remote PC should use when communicating over the network. It's also set to the IP address of the Remote PC itself. This means that when the Remote PC communicates with other devices on the network (like the TurtleBot3), it will identify itself using its own IP address.

These configurations ensure that the TurtleBot3 and the Remote PC can communicate with each other correctly over the network. They're essential for running ROS applications where the ROS Master and the ROS Nodes may be distributed across different devices on the network. By correctly setting the ROS_MASTER_URI and ROS_HOSTNAME variables, all the ROS Nodes can know where to find the ROS Master and how to identify themselves on the network. This is a fundamental part of setting up a distributed ROS system.

3.6 Connecting to the TurtleBot



```
ubuntu@ubuntu: ~
steven@scav:~$ ssh ubuntu@192.168.43.221
ubuntu@192.168.43.221's password:
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-1129-raspi2 armv7l)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Introducing Expanded Security Maintenance for Applications.
   Receive updates to over 25,000 software packages with your
   Ubuntu Pro subscription. Free for personal use.

     https://ubuntu.com/pro

Expanded Security Maintenance for Infrastructure is not enabled.

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

Enable ESM Infra to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Failed to connect to https://changelogs.ubuntu.com/meta-release-lts. Check your In

Last login: Mon Dec  4 10:45:49 2023 from 192.168.43.179
ubuntu@ubuntu:~$ █
```

Figure 3.13: Connecting to the TurtleBot.

A Secure Shell (SSH) connection to the TurtleBot3 is established by executing the command “ssh ubuntu@{IP address of the TurtleBot}” from the current machine. SSH is a cryptographic network protocol that enables secure remote login from one computer to another over an insecure network. The ‘ubuntu’ user’s password on the TurtleBot3 is required once the SSH connection has been initiated. This security measure ensures that access to the TurtleBot3 is granted only to authorized users. If the correct password is entered, the SSH connection is established. However, this is dependent on the accuracy of the IP address and the network reachability of the TurtleBot3. If either of these conditions is not met, the SSH connection will not be successful. Successful login into the TurtleBot3 is achieved if the setup is correct and the TurtleBot3 is reachable over the network. Once logged in, commands can be run directly on the TurtleBot3, its movements can be controlled, and complex robotics software like ROS can be run.

3.7 Specify the Model

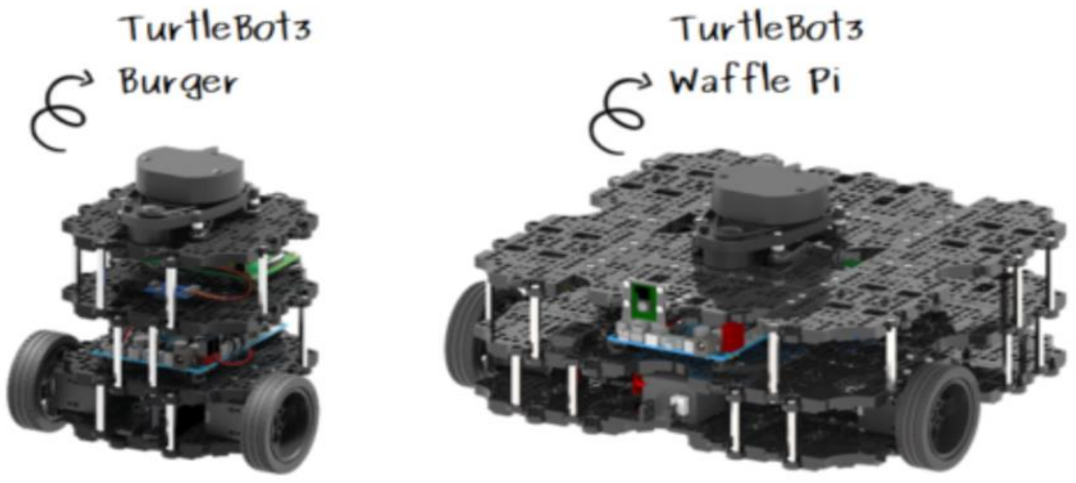


Figure 3.14: TurtleBot3 models

Source:https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#specifications

While working with the TurtleBot3 robot, an environment variable named 'TURTLEBOT3_MODEL' is set to the value 'burger' by executing the command "export TURTLEBOT3_MODEL=burger". This is due to the fact that the TurtleBot3 robot comes in two models: 'burger' and 'waffle_pi', as shown in Figure 3.14. The setting of the TURTLEBOT3_MODEL environment variable to 'burger' signifies that the model of the TurtleBot3 robot being worked with is specified as 'burger'. This environment variable plays a crucial role as it is utilized by the ROS nodes to access the specifications of the robot model. It is imperative to note that the model of the robot must be specified first before any turtlebot3 command is executed. This is to ensure that the ROS nodes have the correct information about the robot's model, which is necessary for the correct execution of the commands. This process is a fundamental step in working with the TurtleBot3 robot and forms the basis for any further interactions with the robot.

3.8 TurtleBot3 Bringup

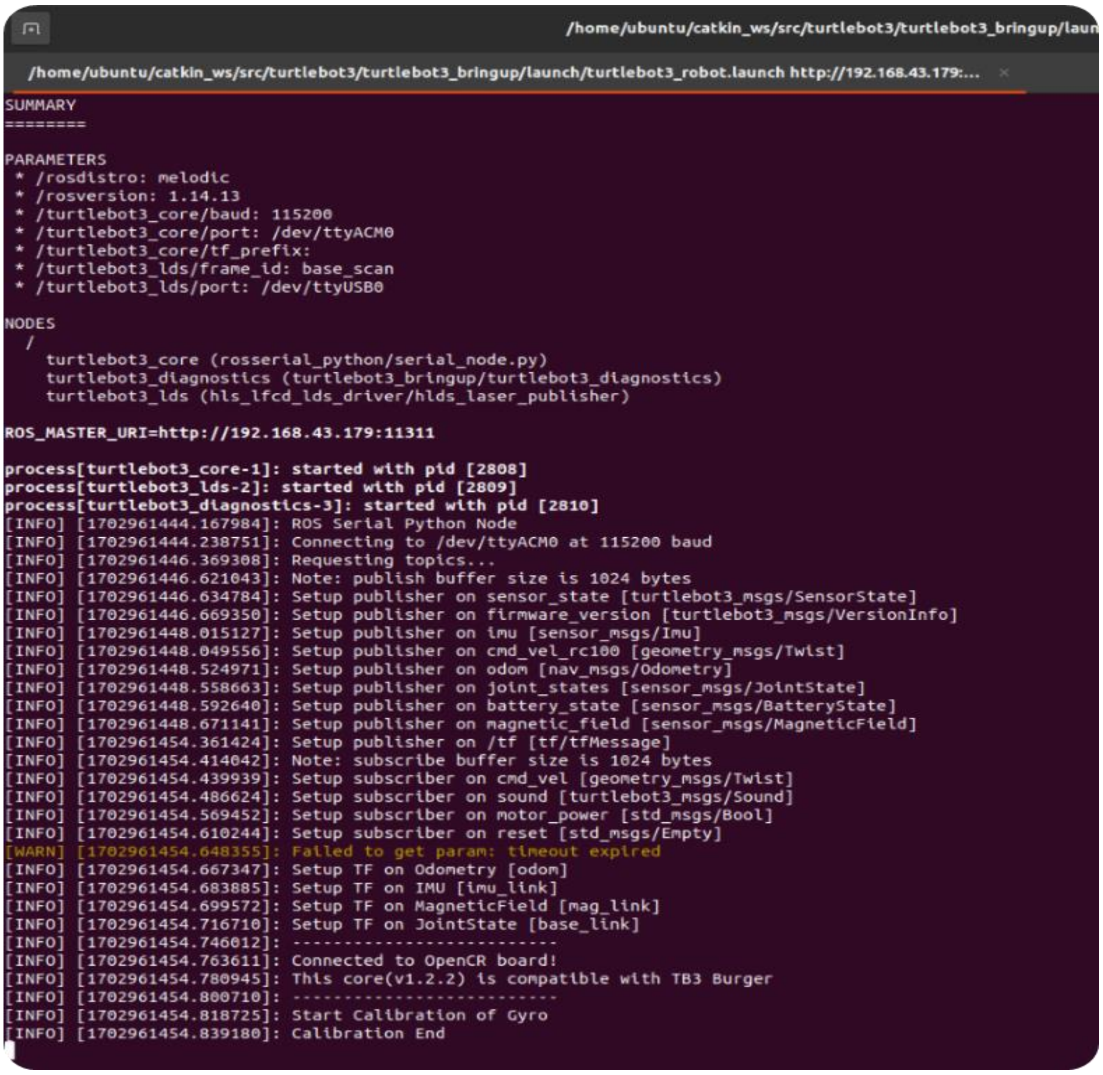
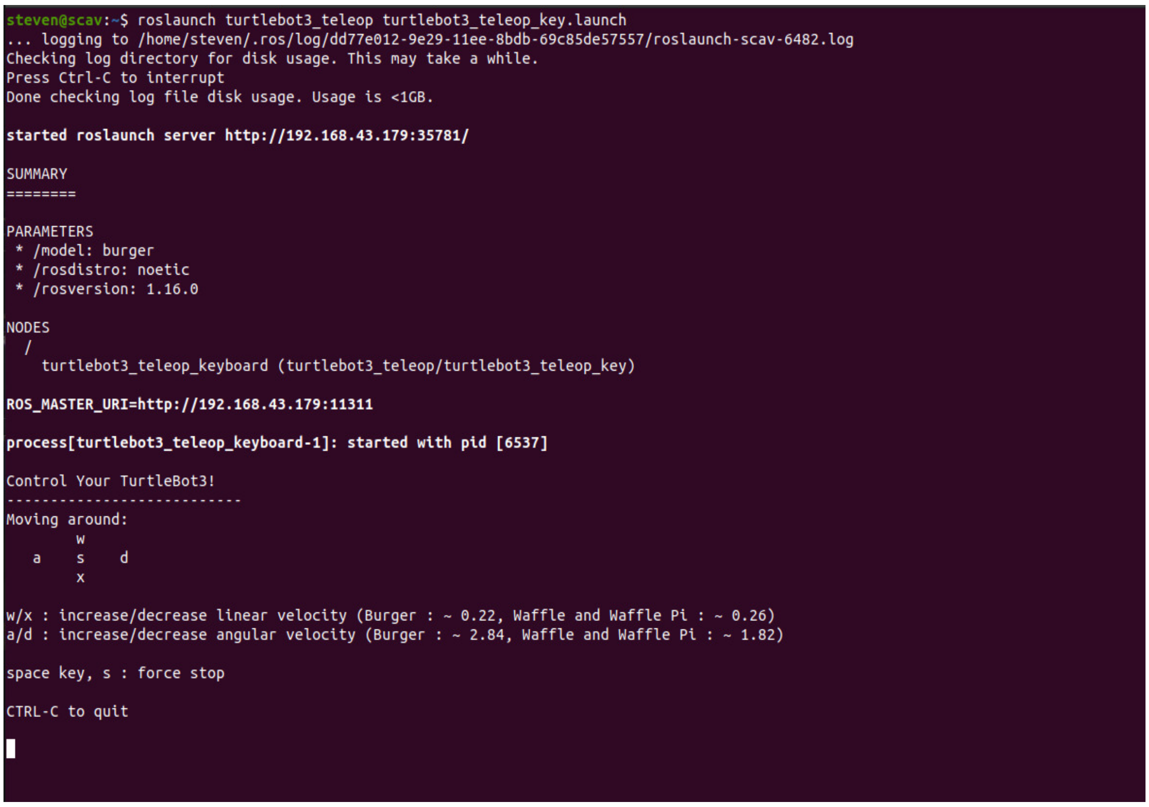

```
/home/ubuntu/catkin_ws/src/turtlebot3/turtlebot3_bringup/laun
/home/ubuntu/catkin_ws/src/turtlebot3/turtlebot3_bringup/launch/turtlebot3_robot.launch http://192.168.43.179:...
SUMMARY
========

PARAMETERS
 * /rosdistro: melodic
 * /rosversion: 1.14.13
 * /turtlebot3_core/baud: 115200
 * /turtlebot3_core/port: /dev/ttyACM0
 * /turtlebot3_core/tf_prefix:
 * /turtlebot3_lds/frame_id: base_scan
 * /turtlebot3_lds/port: /dev/ttyUSB0

NODES
  /
    turtlebot3_core (rosserial_python/serial_node.py)
    turtlebot3_diagnostics (turtlebot3_bringup/turtlebot3_diagnostics)
    turtlebot3_lds (hls_lfcd_lds_driver/hlds_laser_publisher)

ROS_MASTER_URI=http://192.168.43.179:11311

process[turtlebot3_core-1]: started with pid [2808]
process[turtlebot3_lds-2]: started with pid [2809]
process[turtlebot3_diagnostics-3]: started with pid [2810]
[INFO] [1702961444.167984]: ROS Serial Python Node
[INFO] [1702961444.238751]: Connecting to /dev/ttyACM0 at 115200 baud
[INFO] [1702961446.369308]: Requesting topics...
[INFO] [1702961446.621043]: Note: publish buffer size is 1024 bytes
[INFO] [1702961446.634784]: Setup publisher on sensor_state [turtlebot3_msgs/SensorState]
[INFO] [1702961446.669350]: Setup publisher on firmware_version [turtlebot3_msgs/VersionInfo]
[INFO] [1702961448.015127]: Setup publisher on imu [sensor_msgs/Imu]
[INFO] [1702961448.049556]: Setup publisher on cmd_vel_rc100 [geometry_msgs/Twist]
[INFO] [1702961448.524971]: Setup publisher on odom [nav_msgs/Odometry]
[INFO] [1702961448.558663]: Setup publisher on joint_states [sensor_msgs/JointState]
[INFO] [1702961448.592640]: Setup publisher on battery_state [sensor_msgs/BatteryState]
[INFO] [1702961448.671141]: Setup publisher on magnetic_field [sensor_msgs/MagneticField]
[INFO] [1702961454.361424]: Setup publisher on /tf [tf/tfMessage]
[INFO] [1702961454.414042]: Note: subscribe buffer size is 1024 bytes
[INFO] [1702961454.439939]: Setup subscriber on cmd_vel [geometry_msgs/Twist]
[INFO] [1702961454.486624]: Setup subscriber on sound [turtlebot3_msgs/Sound]
[INFO] [1702961454.569452]: Setup subscriber on motor_power [std_msgs/Bool]
[INFO] [1702961454.610244]: Setup subscriber on reset [std_msgs/Empty]
[WARN] [1702961454.648355]: Failed to get param: timeout expired
[INFO] [1702961454.667347]: Setup TF on Odometry [odom]
[INFO] [1702961454.683885]: Setup TF on IMU [imu_link]
[INFO] [1702961454.699572]: Setup TF on MagneticField [mag_link]
[INFO] [1702961454.716710]: Setup TF on JointState [base_link]
[INFO] [1702961454.746012]: -------------------------
[INFO] [1702961454.763611]: Connected to OpenCR board!
[INFO] [1702961454.780945]: This core(v1.2.2) is compatible with TB3 Burger
[INFO] [1702961454.800710]: -------------------------
[INFO] [1702961454.818725]: Start Calibration of Gyro
[INFO] [1702961454.839180]: Calibration End
```

Figure 3.15: TurtleBot3 bringup command.

As shown in Figure 3.15, by executing the command “roslaunch turtlebot3_bringup turtlebot3_robot.launch”, the nodes specified in the turtlebot3_robot.launch file are started, allowing the TurtleBot3 robot to perform various tasks as defined by these nodes. This process is an integral part of operating the TurtleBot3 robot within the ROS framework, enabling the robot to interact with its environment and carry out complex operations. The command “roslaunch turtlebot3_bringup turtlebot3_robot.launch” is a specific instruction used in the ROS framework. This command utilizes ‘roslaunch’, a tool within ROS, to initiate a set of nodes. These nodes are specified within a file named ‘turtlebot3_robot.launch’. The ‘turtlebot3_robot.launch’ is an XML file is located within a package named ‘turtlebot3_bringup’. In the context of ROS, a package is a structured directory that contains code, libraries, and other resources, and it serves as the primary unit for organizing software in ROS. In other words, when the “roslaunch turtlebot3_bringup turtlebot3_robot.launch” command is executed, ‘roslaunch’ looks for the ‘turtlebot3_robot.launch’ file within the ‘turtlebot3_bringup’ package. This launch file contains instructions for starting up a group of nodes.

3.9 Controlling the Robot



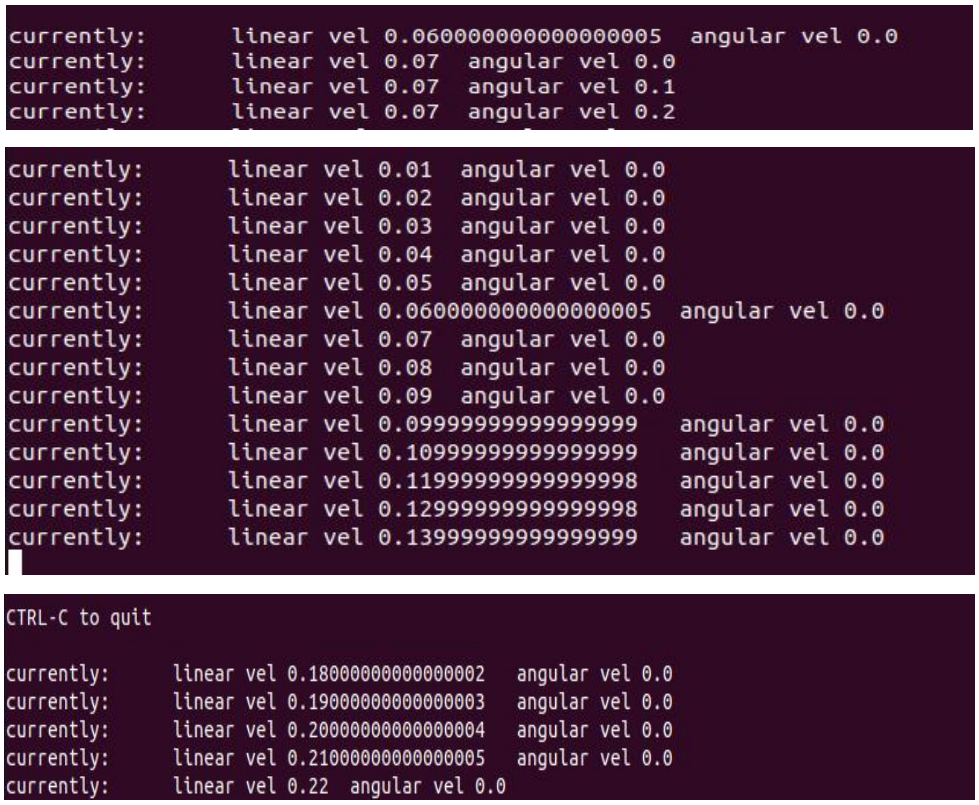
```
steven@scav:~$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
... logging to /home/steven/.ros/log/dd77e012-9e29-11ee-8bdb-69c85de57557/roslaunch-scav-6482.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://192.168.43.179:35781/

SUMMARY
========

PARAMETERS
 * /model: burger
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES
  /
    turtlebot3_teleop_keyboard (turtlebot3_teleop/turtlebot3_teleop_key)

ROS_MASTER_URI=http://192.168.43.179:11311

process[turtlebot3_teleop_keyboard-1]: started with pid [6537]

Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit
```

Figure 3.16: Teleoperation node for the TurtleBot3.

In Figure 3.16, the command “roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch” is also utilized within the ROS framework to initiate the teleoperation node for the TurtleBot3 robot. Teleoperation refers to the process of remotely controlling a robot from a computer or another device. This is particularly useful in scenarios where direct human control is either unsafe, impractical, or inconvenient. The ‘turtlebot3_teleop’ package in ROS provides several launch files for teleoperation using different input devices. The specific launch file used in this command, ‘turtlebot3_teleop_key.launch’, is designed for teleoperation using a keyboard as the input device. This allows the user to control the TurtleBot3 robot using simple keyboard commands.

The keyboard commands for controlling the robot are straightforward which is also illustrated in Figure 3.16. The ‘w’ key is used to move the robot forward, the ‘a’ key to turn it left, and the ‘d’ key to turn it right. The ‘x’ key moves the robot backward, and the ‘s’ key is used to stop the robot. These commands provide a simple and intuitive way to control the robot’s movements. The “roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch" command provides a user-friendly interface for controlling the TurtleBot3 robot. By mapping the robot's movements to keyboard keys, it allows users to easily control the robot, making it a powerful tool for a wide range of applications.

3.9.1 Controlling the Robot's Velocity



```
currently:      linear vel 0.060000000000000005  angular vel 0.0
currently:      linear vel 0.07  angular vel 0.0
currently:      linear vel 0.07  angular vel 0.1
currently:      linear vel 0.07  angular vel 0.2
```

```
currently:      linear vel 0.01  angular vel 0.0
currently:      linear vel 0.02  angular vel 0.0
currently:      linear vel 0.03  angular vel 0.0
currently:      linear vel 0.04  angular vel 0.0
currently:      linear vel 0.05  angular vel 0.0
currently:      linear vel 0.060000000000000005  angular vel 0.0
currently:      linear vel 0.07  angular vel 0.0
currently:      linear vel 0.08  angular vel 0.0
currently:      linear vel 0.09  angular vel 0.0
currently:      linear vel 0.09999999999999999   angular vel 0.0
currently:      linear vel 0.10999999999999999   angular vel 0.0
currently:      linear vel 0.11999999999999998   angular vel 0.0
currently:      linear vel 0.12999999999999998   angular vel 0.0
currently:      linear vel 0.13999999999999999   angular vel 0.0
```

```
CTRL-C to quit

currently:      linear vel 0.18000000000000002   angular vel 0.0
currently:      linear vel 0.19000000000000003   angular vel 0.0
currently:      linear vel 0.20000000000000004   angular vel 0.0
currently:      linear vel 0.21000000000000005   angular vel 0.0
currently:      linear vel 0.22  angular vel 0.0
```

Figure 3.17: The linear velocity of the robot at a) 0.07m/s, b) 0.14m/s, and c) 0.22m/s.

As mentioned earlier, the teleoperation node plays a crucial role in controlling the robot's movements. This node listens to the keyboard input from the user and translates it into corresponding velocity commands, as shown in Figure 3.17. These commands are then sent to the robot's motor controllers. The communication between the teleoperation node and the motor controllers happens over a topic named /cmd_vel, a standard communication channel in ROS for transmitting velocity commands. The motor controllers on the TurtleBot3 robot interpret these velocity commands and adjust the robot's velocity accordingly. This process allows the user to control the robot's movements in real-time, providing a direct and easier way to operate the robot.

In the experiment, three different linear velocities are used: 0.07m/s, 0.14m/s, and 0.22m/s. The maximum velocity of the TurtleBot3 Burger model is 0.22m/s, making it the upper limit for this experiment. By varying the linear velocity, the experiment explores the robot's mapping performance under different speed conditions. While maintaining the same linear velocity, the angular velocity of the TurtleBot3 can be adjusted manually to perform good cornering. This means that even as the robot moves at a constant speed, it can change its direction smoothly. This ability to adjust the angular velocity while maintaining a constant linear velocity is crucial for navigating the robot when the mapping task is carried out.

The teleoperation node, the /cmd_vel topic, and the motor controllers work together to provide a robust and flexible system for controlling the TurtleBot3 robot. By adjusting the linear and angular velocities, the robot can be made to move in a variety of ways, enabling it to carry out mapping task while maintaining a same linear velocity.

3.10 Mapping Task

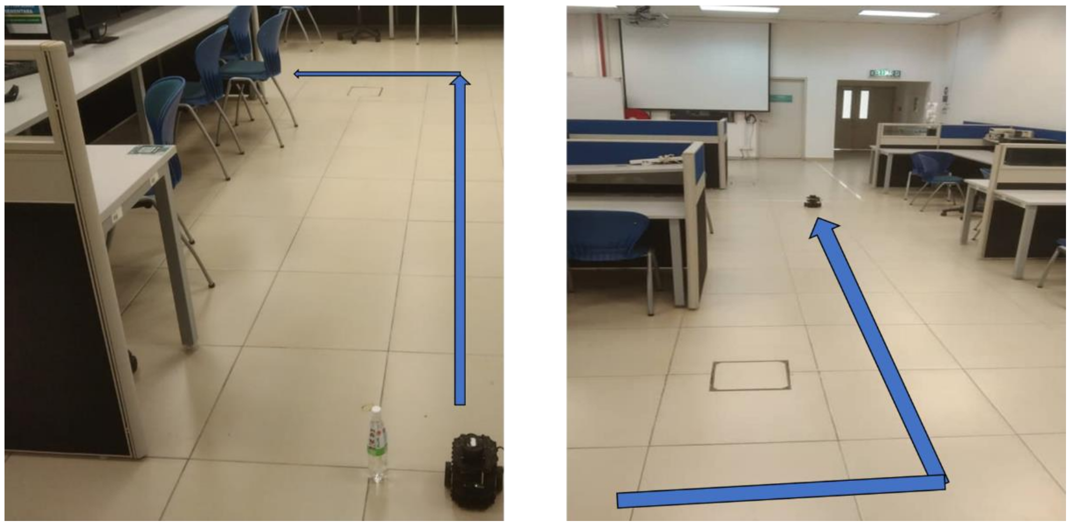


Figure 3.18: Mapping tasks carried out by the Turtlebot3 Burger.

The mapping process for the TurtleBot3 robot is conducted in the Control System Lab, located in the Faculty of Manufacturing and Mechatronic Engineering Technology at UMPSA, as illustrated in Figure 3.18. The choice of this location is due to its simple layout and the fact that it is not frequently used, which creates a static environment that is conducive for mapping.

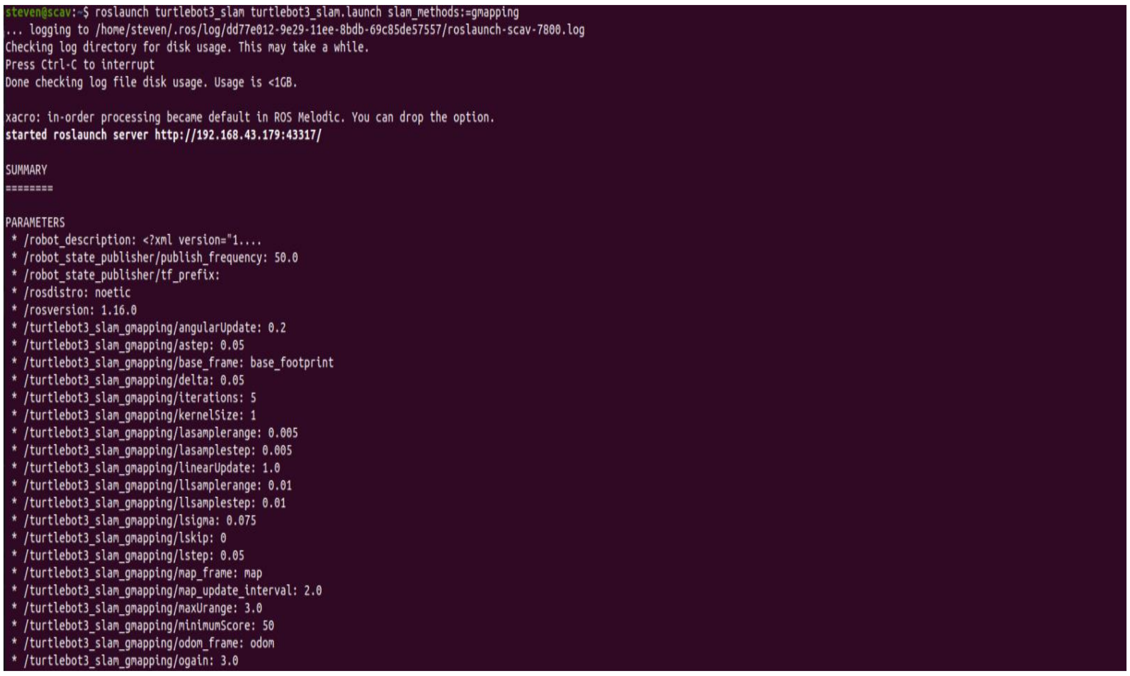
In the initial setup, the robot is placed at a predetermined starting position. A route for the robot to follow is marked out, as indicated by the blue arrow in the accompanying figure. This route is carefully designed to ensure that the robot covers the entire area of the lab while avoiding any obstacles. The experiment begins with the use of GMapping as the first Simultaneous Localization and Mapping (SLAM) algorithm. GMapping is a well-known SLAM algorithm that uses a particle filter to track the robot’s path and update the map. The robot is set to move at three different linear velocities: 0.07m/s, 0.14m/s, and 0.22m/s. These velocities are chosen to test the robot’s mapping capabilities under different speed conditions. The maximum velocity of the TurtleBot3 Burger model is 0.22m/s, making it the upper limit for this experiment.

Once the mapping tasks with GMapping are completed, the experiment continues with Hector SLAM as the next SLAM algorithm. Hector SLAM is another popular SLAM algorithm that is known for its robustness and accuracy. The same three linear velocities are used for this part of the experiment. Finally, the experiment concludes with

Google Cartographer as the last SLAM algorithm. Google Cartographer is a modern SLAM algorithm that uses a combination of 2D laser scans and IMU data to build maps. Again, the robot is set to move at the same three linear velocities.

At the end of the experiment, nine maps are generated, each corresponding to a different combination of SLAM algorithm and linear velocity. These maps are saved in the pgm format, a common format for grayscale images. The maps are then compared to evaluate the performance of the different SLAM algorithms and the effect of different linear velocities on the mapping process. This experiment provides a comprehensive evaluation of the TurtleBot3 robot's mapping capabilities under different speed conditions. The findings from this experiment can provide valuable insights for future research and development in the field of robotics.

3.10.1 Mapping by using GMapping

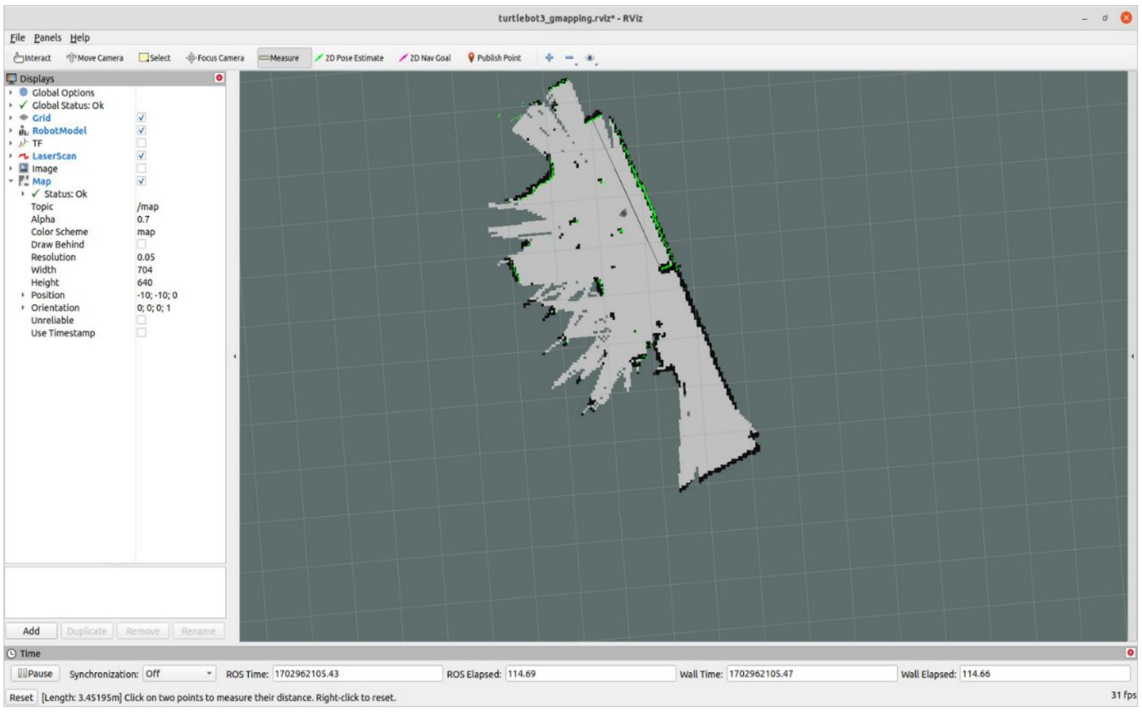


```
steven@scav:~$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
... logging to /home/steven/.ros/log/dd77e012-9e29-11ee-8bdb-69c85de57557/roslaunch-scav-7800.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://192.168.43.179:43317/

SUMMARY
========

PARAMETERS
 * /robot_description: <?xml version="1....
 * /robot_state_publisher/publish_frequency: 50.0
 * /robot_state_publisher/tf_prefix:
 * /rosdistro: noetic
 * /rosversion: 1.16.0
 * /turtlebot3_slam_gmapping/angularUpdate: 0.2
 * /turtlebot3_slam_gmapping/astep: 0.05
 * /turtlebot3_slam_gmapping/base_frame: base_footprint
 * /turtlebot3_slam_gmapping/delta: 0.05
 * /turtlebot3_slam_gmapping/iterations: 5
 * /turtlebot3_slam_gmapping/kernelSize: 1
 * /turtlebot3_slam_gmapping/lasamplerange: 0.005
 * /turtlebot3_slam_gmapping/lasamplestep: 0.005
 * /turtlebot3_slam_gmapping/linearUpdate: 1.0
 * /turtlebot3_slam_gmapping/llsamplerange: 0.01
 * /turtlebot3_slam_gmapping/llsamplestep: 0.01
 * /turtlebot3_slam_gmapping/lsigma: 0.075
 * /turtlebot3_slam_gmapping/lskip: 0
 * /turtlebot3_slam_gmapping/lstep: 0.05
 * /turtlebot3_slam_gmapping/map_frame: map
 * /turtlebot3_slam_gmapping/map_update_interval: 2.0
 * /turtlebot3_slam_gmapping/maxUrange: 3.0
 * /turtlebot3_slam_gmapping/minimumScore: 50
 * /turtlebot3_slam_gmapping/odom_frame: odom
 * /turtlebot3_slam_gmapping/ogain: 3.0
```

Figure 3.19: Run the GMapping SLAM node.

The command "roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping" as in Figure 3.19 is used to initiate the Simultaneous Localization and Mapping (SLAM) node for the TurtleBot3 robot. SLAM is a computational problem in robotics that involves constructing or updating a map of an unknown environment while simultaneously keeping track of the robot's location within it. The 'turtlebot3_slam' package in ROS provides several launch files for SLAM, with GMapping being one of the mapping methods available.

When this command is executed, the SLAM node is launched, and the TurtleBot3 robot begins its exploration of the environment. The robot utilizes its onboard sensors to gather data about its surroundings. This data is then processed by the SLAM node to create a map of the environment. Simultaneously, the robot localizes itself within this map. This dual process of mapping and localization allows the robot to navigate effectively within its environment, even if the environment is previously unknown to the robot. The map created by the SLAM process is published on the '/map' topic in ROS. Topics in ROS are named buses over which nodes exchange messages. In this case, the '/map' topic serves as the communication channel for transmitting the map data from the SLAM node to any other nodes that may need it.



Global Options	
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Image	☐
Map	✓
Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	704
Height	640
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐

Synchronization: Off ROS Time: 1702962105.43 ROS Elapsed: 114.69 Wall Time: 1702962105.47 Wall Elapsed: 114.66

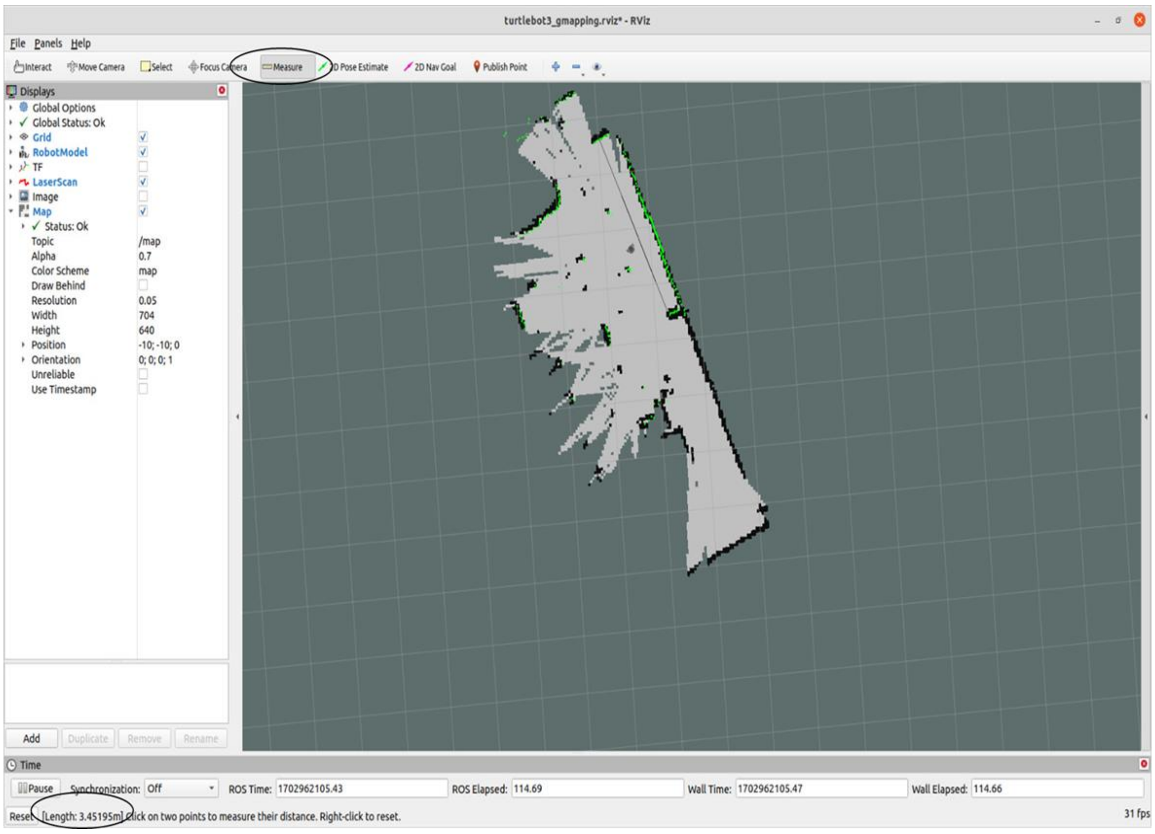
[Length: 3.45195m] Click on two points to measure their distance. Right-click to reset.

Figure 3.20: RVIZ interface.

RViz is a 3D visualization tool that is part of the ROS. It provides users with the ability to visualize the state of a robot in a three-dimensional space. This is particularly useful in robotics applications as it allows users to visually monitor and understand the robot's interactions with its environment. One of the key features of RViz is its ability to display the map created by the SLAM node. By visualizing this map in RViz, the robot's understanding of its environment and its position within that environment can be seen. In addition to visualizing the robot's state, RViz can also be used in conjunction with the teleoperation node to control the robot's movements. The teleoperation node, as discussed in Chapter 3.9, is a component of ROS that allows users to remotely control a robot. For TurtleBot3, the teleoperation node is used to move the robot around its environment, enabling it to explore and create a map of its surroundings.

The mapping process is influenced by several parameters, including the map resolution, the LiDAR scanning rate, the angular range, the angular resolution, and the range. In this case, the map resolution is set to 0.05, indicating the size of each cell in the map grid. The LiDAR scanning rate is set to 300 revolutions per minute (rpm), determining how frequently the robot's LiDAR sensor scans its environment. The angular range is set to 360 degrees, allowing the robot to scan its entire surroundings. The angular resolution is set to 1 degree, determining the granularity of the LiDAR scan. Finally, the range is set between 120 and 3500mm, indicating the minimum and maximum distances that the LiDAR sensor can detect. These parameters collectively influence the quality and accuracy of the map created by the TurtleBot3.

3.11 Measure the Length of Features in RVIZ



Global Options	
Global Status: Ok	
Grid	✓
RobotModel	✓
TF	☐
LaserScan	✓
Image	☐
Map	✓
Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	☐
Resolution	0.05
Width	704
Height	640
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	☐
Use Timestamp	☐

Synchronization: Off ROS Time: 1702962105.43 ROS Elapsed: 114.69 Wall Time: 1702962105.47 Wall Elapsed: 114.66

[Length: 3.45195m] Click on two points to measure their distance. Right-click to reset. 31 fps

Figure 3.21: The 'Measure' tool in RVIZ.

Figure 3.21 illustrates the “Measure” tool in RVIZ, a 3D visualization tool in ROS, is utilized for measuring the distance between corresponding points in the 2D LiDAR map. Activation of the tool is achieved by clicking on the “Measure” button located in the toolbar on the left side of the RViz window. The starting point of the measurement is set by a single click on the desired location on the 2D LiDAR map. The cursor is then moved to the desired end point of the measurement, and a second click sets this end point. The distance between the two points is calculated by the tool, taking into account the scale of the map. The measured distance is displayed at the bottom of the RViz window for easy reference, and the data are recorded.

Comparison of the results with measurements taken in the real world is possible, which can be useful for verifying the accuracy of the map or for calibrating the robot's sensors. By comparing the measurements from the "Measure" tool with real-world distances, the accuracy of the map's representation of the physical environment and the robot's perception of its surroundings can be ensured.

3.12 Save the Map

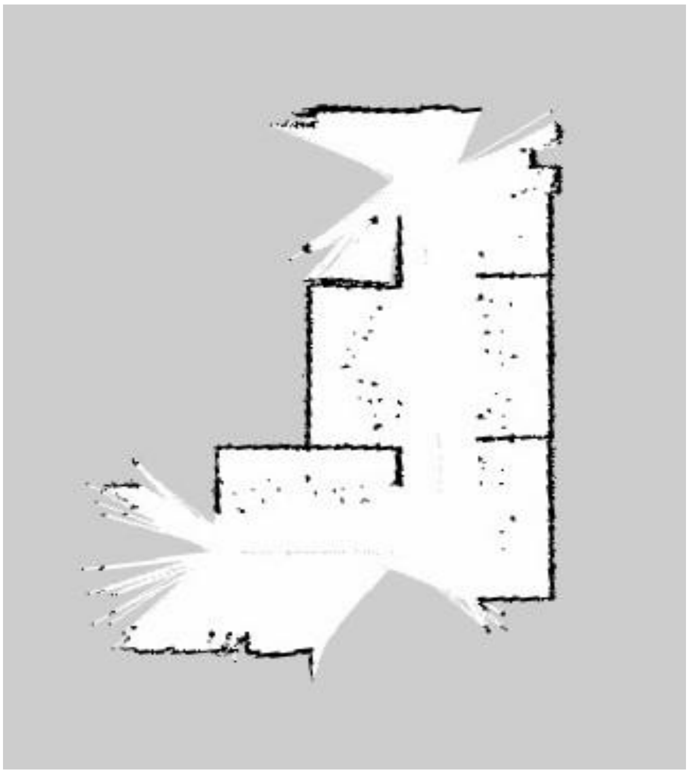


Figure 3.22: Map generated in RViz.

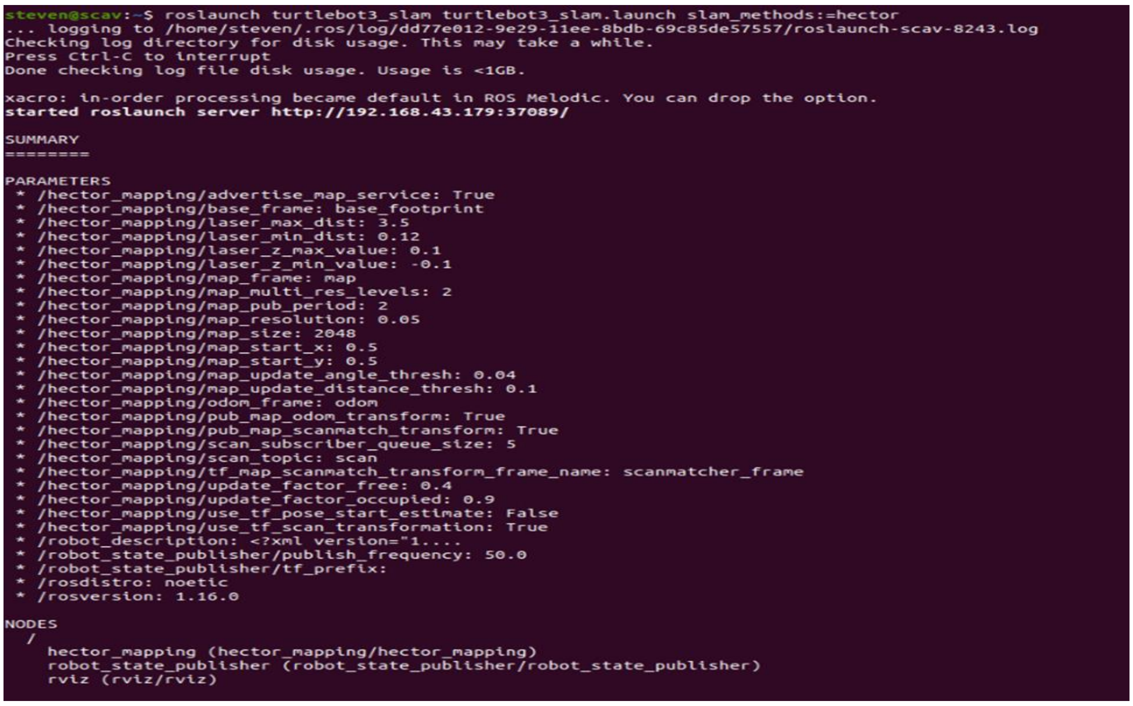
The creation of the map is based on the robot's odometry, tf, and scan information. As the TurtleBot3 travels, this data is visualized in the RViz window, resulting in the formation of the map. Once a complete map of the desired area has been created, as in Figure 3.22, the map data is saved to the local drive for future use.

The 'map_saver' node, which is part of the 'map_server' package, is launched to generate map files. The location where the 'map_saver' node is launched serves as the directory where the map file is saved. The command "rosrun map_server map_saver -f ~/map" is executed to save the map.

The map utilizes a two-dimensional Occupancy Grid Map (OGM), a format commonly used in ROS. In the saved map, different areas are represented by different colours: white represents collision-free areas, black represents occupied and inaccessible areas, and grey represents unknown areas. This map is subsequently used for navigation purposes.

In conclusion, the process involves the collection of robot’s sensor data, visualization of this data into a map, and the saving of this map for future navigation tasks. This procedure is a fundamental part of operating the TurtleBot3 robot within the ROS framework, enabling the robot to navigate effectively within its environment.

3.13 Repeat the Steps

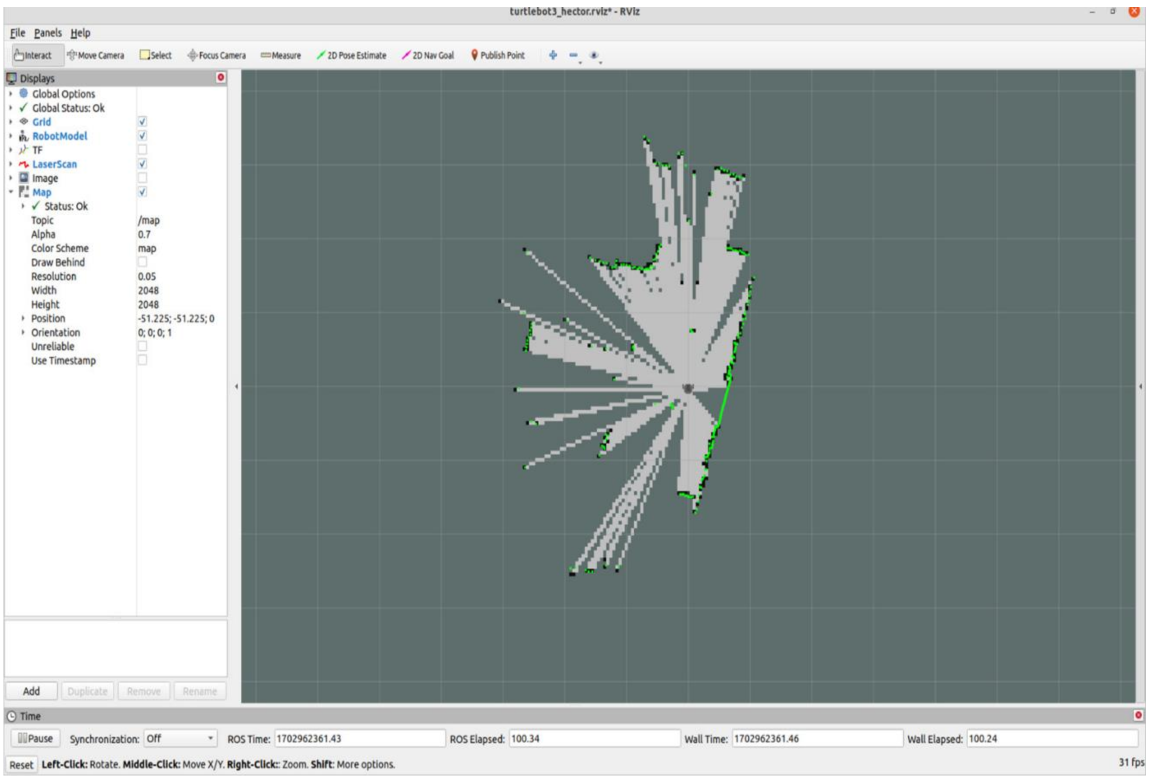


```
steven@scav:~$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=hector
... logging to /home/steven/.ros/log/dd77e012-9e29-11ee-8bdb-69c85de57557/roslaunch-scav-8243.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://192.168.43.179:37089/

SUMMARY
========

PARAMETERS
 * /hector_mapping/advertise_map_service: True
 * /hector_mapping/base_frame: base_footprint
 * /hector_mapping/laser_max_dist: 3.5
 * /hector_mapping/laser_min_dist: 0.12
 * /hector_mapping/laser_z_max_value: 0.1
 * /hector_mapping/laser_z_min_value: -0.1
 * /hector_mapping/map_frame: map
 * /hector_mapping/map_multi_res_levels: 2
 * /hector_mapping/map_pub_period: 2
 * /hector_mapping/map_resolution: 0.05
 * /hector_mapping/map_size: 2048
 * /hector_mapping/map_start_x: 0.5
 * /hector_mapping/map_start_y: 0.5
 * /hector_mapping/map_update_angle_thresh: 0.04
 * /hector_mapping/map_update_distance_thresh: 0.1
 * /hector_mapping/odom_frame: odom
 * /hector_mapping/pub_map_odom_transform: True
 * /hector_mapping/pub_map_scanmatch_transform: True
 * /hector_mapping/scan_subscriber_queue_size: 5
 * /hector_mapping/scan_topic: scan
 * /hector_mapping/tf_map_scanmatch_transform_frame_name: scanmatcher_frame
 * /hector_mapping/update_factor_free: 0.4
 * /hector_mapping/update_factor_occupied: 0.9
 * /hector_mapping/use_tf_pose_start_estimate: False
 * /hector_mapping/use_tf_scan_transformation: True
 * /robot_description: <?xml version="1....
 * /robot_state_publisher/publish_frequency: 50.0
 * /robot_state_publisher/tf_prefix:
 * /rosdistro: noetic
 * /rosversion: 1.16.0

NODES
  /
    hector_mapping (hector_mapping/hector_mapping)
    robot_state_publisher (robot_state_publisher/robot_state_publisher)
    rviz (rviz/rviz)
```

Figure 3.23: Initiate SLAM node using Hector SLAM.

Upon the successful generation and saving of the three occupancy grid maps in RViz using the GMapping algorithm, the experiment progresses to the next phase. This involves returning the TurtleBot3 to its original position to ensure a consistent starting point for the next set of mapping tasks.

The next step involves launching the SLAM node again, but this time employing the Hector SLAM algorithm. This is achieved by executing the command “roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=hector”, as shown in Figure 3.23. Before the launching of SLAM node, the Hector SLAM package is installed first in the ROS first from the TurtleBot3 website. With the Hector SLAM algorithm in place, the mapping process is repeated. The TurtleBot3 is set to move at three different linear velocities: 0.07m/s, 0.14m/s, and 0.22m/s, which is the same as in GMapping. These velocities are chosen to test the robot’s mapping capabilities under different speed conditions, with Hector SLAM as SLAM algorithm.



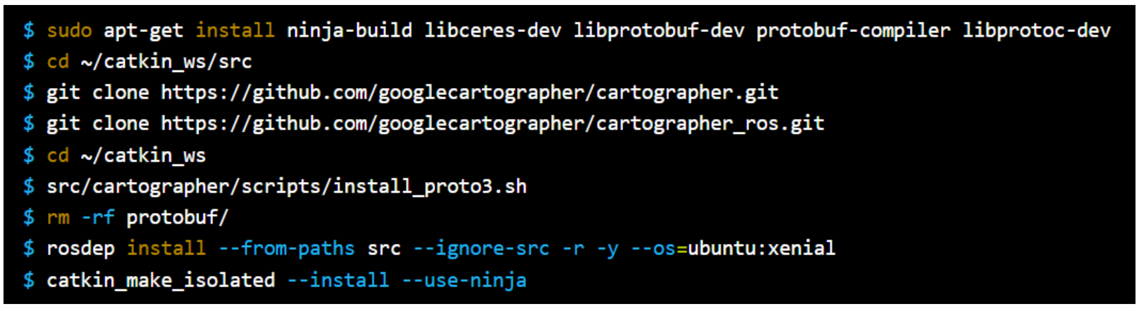
```
Topic             /map
Alpha             0.7
Color Scheme      map
Draw Behind
Resolution        0.05
Width             2048
Height            2048
Position          -51.225; -51.225; 0
Orientation       0; 0; 0; 1
Unreliable
Use Timestamp
```

```
Synchronization: Off   ROS Time: 1702962361.43   ROS Elapsed: 100.34   Wall Time: 1702962361.46   Wall Elapsed: 100.24
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.   31 fps
```

Figure 3.24: RViz with Hector SLAM.

After the Rviz is launched, the TurtleBot3 explores its environment. Then, the 'Measure' tool in RViz is used to measure predefined features in the map, the same step as describes in Chapter 3.11. All the data from these measurements are saved for further analysis. Finally, the three occupancy grid maps generated using the Hector SLAM algorithm are saved. These maps will be visually compared later on to evaluate the performance of the different SLAM algorithms and the effect of different linear velocities on the mapping process.

3.13.1 Google Cartographer

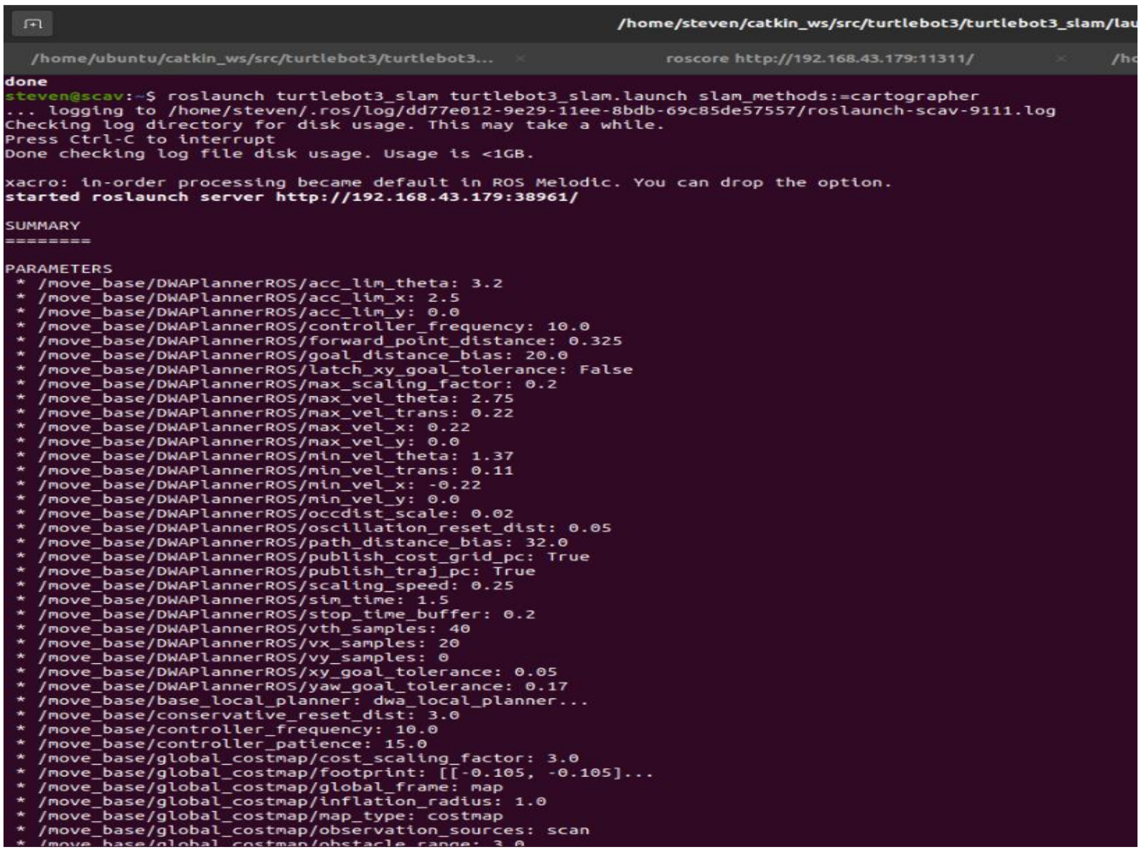


```
$ sudo apt-get install ninja-build libceres-dev libprotobuf-dev protobuf-compiler libprotoc-dev
$ cd ~/catkin_ws/src
$ git clone https://github.com/googlecartographer/cartographer.git
$ git clone https://github.com/googlecartographer/cartographer_ros.git
$ cd ~/catkin_ws
$ src/cartographer/scripts/install_proto3.sh
$ rm -rf protobuf/
$ rosdep install --from-paths src --ignore-src -r -y --os=ubuntu:xenial
$ catkin_make_isolated --install --use-ninja
```

Figure 3.25: The source code of installing Cartographer.
Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/slam/#save-map

The installation of Google Cartographer package is mentioned here due to its difficulty to be installed in ROS1 Noetic, which is worth noting of. The source code of installing Cartographer in ROS1 Noetic, as shown in Figure 3.25, is necessary to be downloaded and built. The binary installation method of Cartographer for ROS1 Noetic currently is not provided. The Cartographer package is not included with ROS1 Noetic "out of the box" because it has not been officially released for this version. The decision to release a package for a specific ROS version is up to the developers/maintainers of that package. As of the last update, the maintainers of Cartographer have stated that they plan to release it for Noetic, but the timeline and method for this release have not been finalized. Therefore, until an official release is made, Cartographer must be installed from source when using ROS Noetic.

This process involves manually downloading and building the Cartographer source code and its dependencies, including the 'cartographer_ros' package, which provides the ROS integration for Cartographer. The commands listed in Figure 3.25 are necessary for installing Cartographer on ROS Noetic.



/home/steven/catkin_ws/src/turtlebot3/turtlebot3_slam/lau
/home/ubuntu/catkin_ws/src/turtlebot3/turtlebot3...
roscore http://192.168.43.179:11311/
/hc

```
done
steven@scav:~$ roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=cartographer
... logging to /home/steven/.ros/log/dd77e012-9e29-11ee-8bdb-69c85de57557/roslaunch-scav-9111.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

xacro: in-order processing became default in ROS Melodic. You can drop the option.
started roslaunch server http://192.168.43.179:38961/

SUMMARY
========

PARAMETERS
 * /move_base/DWAPlannerROS/acc_lim_theta: 3.2
 * /move_base/DWAPlannerROS/acc_lim_x: 2.5
 * /move_base/DWAPlannerROS/acc_lim_y: 0.0
 * /move_base/DWAPlannerROS/controller_frequency: 10.0
 * /move_base/DWAPlannerROS/forward_point_distance: 0.325
 * /move_base/DWAPlannerROS/goal_distance_bias: 20.0
 * /move_base/DWAPlannerROS/latch_xy_goal_tolerance: False
 * /move_base/DWAPlannerROS/max_scaling_factor: 0.2
 * /move_base/DWAPlannerROS/max_vel_theta: 2.75
 * /move_base/DWAPlannerROS/max_vel_trans: 0.22
 * /move_base/DWAPlannerROS/max_vel_x: 0.22
 * /move_base/DWAPlannerROS/max_vel_y: 0.0
 * /move_base/DWAPlannerROS/min_vel_theta: 1.37
 * /move_base/DWAPlannerROS/min_vel_trans: 0.11
 * /move_base/DWAPlannerROS/min_vel_x: -0.22
 * /move_base/DWAPlannerROS/min_vel_y: 0.0
 * /move_base/DWAPlannerROS/occdist_scale: 0.02
 * /move_base/DWAPlannerROS/oscillation_reset_dist: 0.05
 * /move_base/DWAPlannerROS/path_distance_bias: 32.0
 * /move_base/DWAPlannerROS/publish_cost_grid_pc: True
 * /move_base/DWAPlannerROS/publish_traj_pc: True
 * /move_base/DWAPlannerROS/scaling_speed: 0.25
 * /move_base/DWAPlannerROS/sim_time: 1.5
 * /move_base/DWAPlannerROS/stop_time_buffer: 0.2
 * /move_base/DWAPlannerROS/vth_samples: 40
 * /move_base/DWAPlannerROS/vx_samples: 20
 * /move_base/DWAPlannerROS/vy_samples: 0
 * /move_base/DWAPlannerROS/xy_goal_tolerance: 0.05
 * /move_base/DWAPlannerROS/yaw_goal_tolerance: 0.17
 * /move_base/base_local_planner: dwa_local_planner...
 * /move_base/conservative_reset_dist: 3.0
 * /move_base/controller_frequency: 10.0
 * /move_base/controller_patience: 15.0
 * /move_base/global_costmap/cost_scaling_factor: 3.0
 * /move_base/global_costmap/footprint: [[-0.105, -0.105]...
 * /move_base/global_costmap/global_frame: map
 * /move_base/global_costmap/inflation_radius: 1.0
 * /move_base/global_costmap/map_type: costmap
 * /move_base/global_costmap/observation_sources: scan
 * /move_base/global_costmap/obstacle_range: 3.0
```

Figure 3.26: Initiate SLAM node with Cartographer.

Upon successful completion of the source code download and build process, the Cartographer SLAM node is initiated as in Figure 3.26. The command “roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=cartographer” is used for this purpose. In RViz, the same methodology as GMapping and Hector SLAM is employed. The TurtleBot3 is set to scan the environment at three different velocities (0.07, 0.14, 0.22 m/s), enabling the creation of a 2D LiDAR map using Cartographer. The ‘Measure’ tool in RViz is then utilized to measure predefined features. All data collected during this process are saved for future reference. The occupancy grid maps are stored in the PGM format. This allows for a visual comparison of all nine maps (three each from GMapping, Hector SLAM, and Cartographer). Finally, the mapping error is calculated and compared across all methods. This comparison provides valuable insights into the performance and accuracy of each SLAM method. The method with the smallest mapping error is typically considered the most accurate.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Map Generated (GMapping)

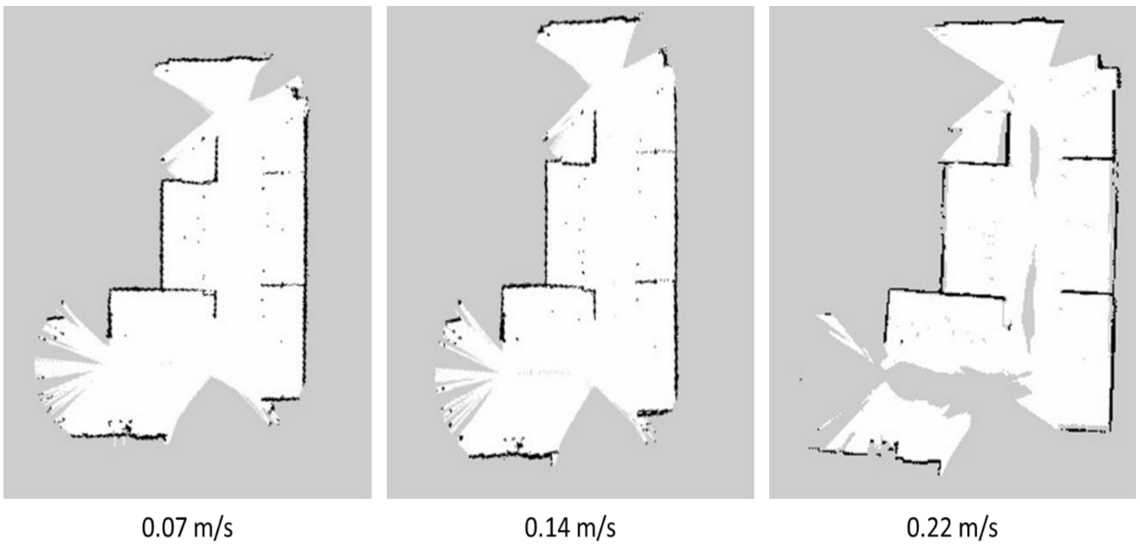


Figure 4.1: GMapping map generated at a) 0.07m/s, b) 0.14m/s, c) 0.22m/s.

Figure 4.1 illustrates the map generated by RViz with GMappping as the SLAM algorithm, with the Turtlebot3 at the velocities of 0.07m/s, 0.14m/s, and 0.22m/s respectively. At a lower speed of 0.07m/s, it is observed that the map generated by GMapping is of high quality with clear details. The structure of the map is well-defined, with corners and edges being distinctly visible. The walls appear straight, providing a faithful representation of the actual environment. This is in contrast to the maps generated at higher speeds of 0.14m/s and 0.22m/s. When the velocity is 0.14m/s, slight distortions start to appear in the map. Some corners and edges become rounded, and some lines, which should be straight, are not. This indicates that the accuracy of GMapping starts to decrease with an increase in speed. At a velocity of 0.22m/s, it has been observed that the map is heavily distorted. The layout of the room, as represented by the map, is significantly altered from its actual structure. A notable distortion is the presence of a large grey zone at the bottom left of the map. This grey zone represents an unknown area in the mapping system. However, this is a clear error in the mapping calculation by the system, as there is no such gap in reality.

The distortion and mapping errors can be attributed to the relatively high velocity of the TurtleBot3 and the resulting odometry error. At higher velocities, the TurtleBot3 has less time to accurately scan its surroundings, leading to inaccuracies in the data it collects. These inaccuracies are then reflected in the generated map. GMapping’s sensitivity to odometry, which is the use of data from motion sensors to estimate change in position over time, plays a significant role in this phenomenon. Higher speeds can lead to larger odometry errors, as the robot has less time to accurately sense its surroundings. These errors can then propagate into the map, leading to distortions and inaccuracies.

Therefore, the system’s inability to accurately map its surroundings under these conditions results in a distorted representation of the environment. This highlights the importance of considering the robot’s velocity and the accuracy of its odometry when using SLAM methods like GMapping. It also underscores the need for careful calibration and error correction to ensure accurate mapping.

4.2 Map Generated (Hector SLAM)

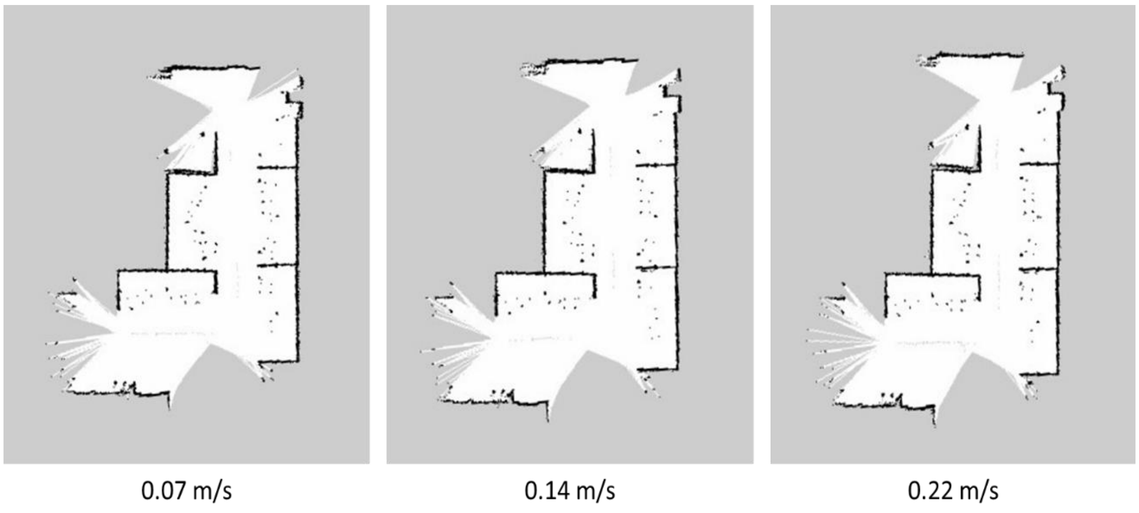


Figure 4.2: Hector SLAM map generated at a) 0.07m/s, b) 0.14m/s, c) 0.22m/s.

As illustrated in Figure 4.2, the map generated by Hector SLAM at 0.07m/s, 0.14m/s, and 0.22m/s are presented. In the case of Hector SLAM, it is observed that the quality of the maps remains high across different velocities. The maps exhibit clear details, with sharp edges and straight lines being distinctly visible. This indicates that Hector SLAM is able to accurately capture the structure of the environment, even at higher speeds. The layout of the maps generated by Hector SLAM closely resembles the actual environment, suggesting a high degree of accuracy in the mapping process. This is a significant advantage of Hector SLAM, as it allows for a more faithful representation of the environment, which is crucial for navigation and other robotic tasks.

However, at a high speed of 0.22m/s, a slight distortion is noticed at the top of the map. This suggests that, while Hector SLAM performs well at different speeds, the accuracy may slightly decrease at higher speeds. This is likely due to the increased difficulty in accurately capturing sensor data at higher speeds. Hector SLAM's ability to handle higher speeds better than GMapping, with less distortion observed in the maps, can be attributed to its independence from odometry data. Odometry data, which is the estimation of change in position over time using motion sensors, plays a significant role in many SLAM methods, including GMapping. Errors in odometry data, which can be more pronounced at higher speeds, can lead to distortions and inaccuracies in the generated maps. The independence of Hector SLAM from odometry data contributes to its ability to generate high-quality maps at higher speeds, distinguishing it from other SLAM methods like GMapping.

4.3 Map Generated (Google Cartographer)

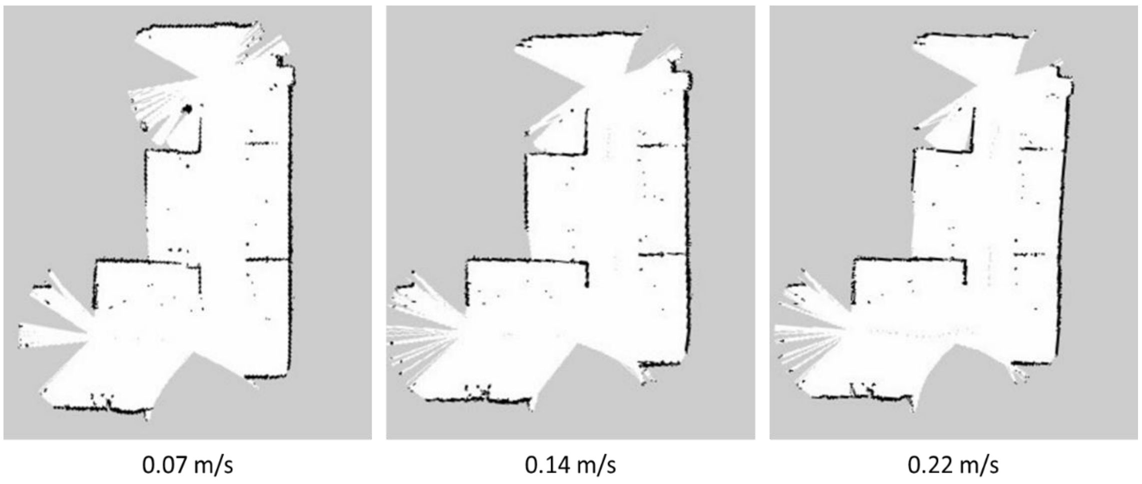


Figure 4.3: Cartographer map generated at a) 0.07m/s, b) 0.14m/s, c) 0.22m/s.

Figure 4.3 shows the maps generated by Google Cartographer, at the speed of 0.07m/s, 0.14m/s, and 0.22m/s. The performance of Google Cartographer in generating maps is observed to be satisfactory, but not without noticeable distortions, especially as the speed is increased. Compared to GMapping and Hector SLAM, the details in the maps generated by Cartographer are not as clear. At lower speeds (0.07 and 0.14 m/s), the maps generated by using Cartographer are well-defined and the layout is quite clear, although not as precise as those generated by Hector SLAM. Some lines in the map are straight, accurately representing the environment, while others are inaccurately represented with a certain slanted angle. Sharp edges in the environment are also clearly visible in the map.

However, at a higher speed (0.22m/s), the distortions in the map become more obvious. The upper side of the map appears to be “slanted to the right”, indicating a deviation from the actual layout of the environment. Despite this distortion, the overall layout of the map is still considered acceptable, and the performance of Cartographer at this speed is not as bad as GMapping.

These observations suggest that while Google Cartographer is capable of generating usable maps at different speeds, its performance and the accuracy of the maps it generates can be affected by the speed of the robot. This highlights the importance of selecting an appropriate speed for the robot when using SLAM methods like Cartographer. It also underscores the need for careful calibration and error correction to ensure accurate mapping. Despite these challenges, Cartographer’s performance is commendable, particularly when compared to other SLAM methods at higher speeds. However, for the most accurate mapping, it may be beneficial to operate the robot at lower speeds.

4.4 Predefined Features

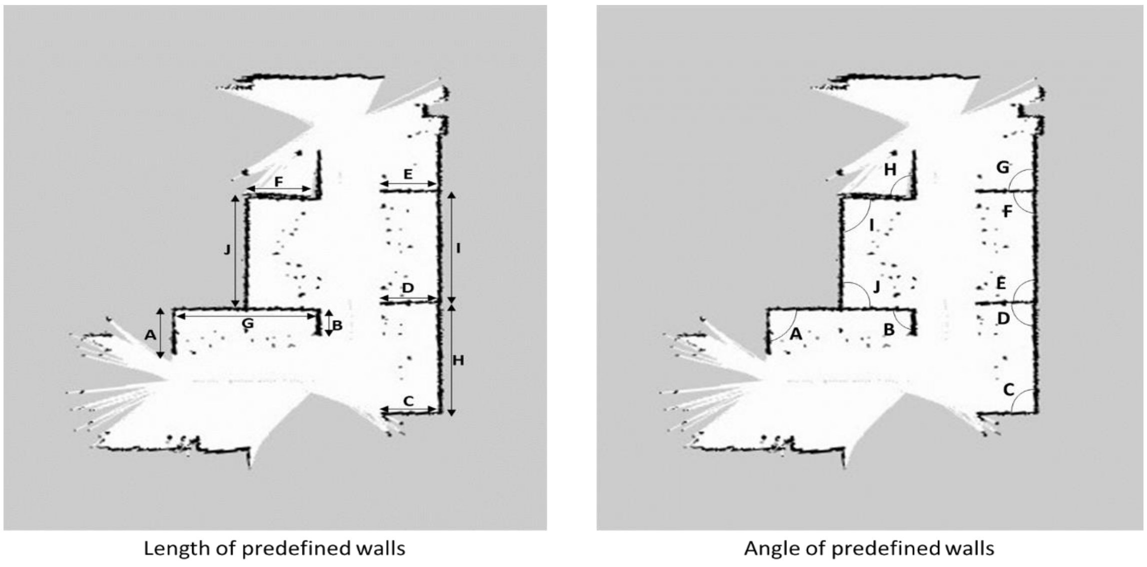


Figure 4.4: Predefined sectors for measuring length and angle.

Figure 4.4 shows the predefined sectors for measuring the length and the angle in the map which are used for calculating the mapping error. The process of calculating the mapping error involves several steps. Initially, ten sectors in the map that can be measured are identified. These sectors are then measured and the results are compared with the actual dimensions of these sectors in the real world. All the data for each map, including the measurements and the corresponding actual dimensions, are recorded in a table. The absolute error for each sector is then calculated by comparing the measured dimensions with the actual dimensions.

In addition to the sectors, ten corners in the map that can be measured are also identified and measured. The data from these measurements are further interpreted to calculate the mapping error.

The final calculation of the mapping error is based on two main factors: a. The dimensions of the map sectors, and b. The angles of the corners of the map, as shown in Figure 4.4.

By comparing the measured dimensions and angles with the actual dimensions and angles, the mapping error can be calculated. This error represents the difference between the map generated by the SLAM method with different speed, and the actual layout of the environment. It provides a quantitative measure of the accuracy of the SLAM method under different speed conditions, allowing for a more objective comparison between different methods.

4.5 Results Obtained From Experiment

The process of evaluating the accuracy of the maps generated by GMapping, Hector SLAM, and Google Cartographer involves comparing the measured dimensions and angles in the maps with their actual values in the real world. This is done at different speeds: 0.07m/s, 0.14m/s, and 0.22m/s.

Firstly, the lengths of various features in the maps are measured and recorded. These measurements are then compared with the actual dimensions of the same features in the real world. The results of these comparisons are presented in a tabular form in Chapters 4.5.1 to 4.5.3. Each row in the table represents a different feature, and the columns contain the measured length from the map and the actual length in the real world.

In addition to the lengths of features, the angles of various corners in the maps are also measured and compared with their actual angles in the real world. The results of these angle comparisons are presented in Chapters 4.5.4 to 4.5.6.

The absolute error for each feature and corner is then calculated. This is done by finding the difference between the measured dimension or angle in the map and the actual dimension or angle in the real world. The absolute value of this difference is taken to ensure that the error is always a positive number, regardless of whether the measured value is greater or less than the actual value.

This absolute error represents the discrepancy between the map and the real world. A smaller absolute error indicates a higher accuracy of the mapping algorithm, as it means that the map is a closer representation of the real world. Conversely, a larger absolute error suggests that the map is less accurate. By calculating the absolute error for multiple features and corners and comparing these errors, the overall accuracy of each SLAM method with different speed can be obtained. This can help to identify areas where each method performs well and areas where it might need improvement. It’s a valuable tool for evaluating and refining SLAM methods.

4.5.1 GMapping (length)

Table 4.1: Result of sector’s length by GMapping at 0.07m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	174	6
B	94	96	2
C	168	173	5
D	168	171	3
E	168	172	4
F	210	215	5
G	421	411	10
H	421	412	9
I	421	405	16
J	421	409	12

Table 4.2: Result of sector’s length by GMapping at 0.14m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	177	9
B	94	93	1
C	168	165	3
D	168	160	8
E	168	167	1
F	210	207	3
G	421	402	19
H	421	412	9
I	421	409	12
J	421	402	19

Table 4.3: Result of sector’s length by GMapping at 0.22m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	176	8
B	94	90	4
C	168	176	8
D	168	170	2
E	168	167	1
F	210	224	14
G	421	404	17
H	421	410	11
I	421	403	18
J	421	401	20

4.5.2 Hector SLAM (length)

Table 4.4: Result of sector's length by Hector SLAM at 0.07m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	170	2
B	94	100	6
C	168	170	2
D	168	170	2
E	168	164	4
F	210	216	6
G	421	412	9
H	421	417	4
I	421	419	2
J	421	417	4

Table 4.5: Result of sector's length by Hector SLAM at 0.14m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	172	4
B	94	93	1
C	168	170	2
D	168	170	2
E	168	165	3
F	210	217	7
G	421	417	4
H	421	416	5
I	421	415	6
J	421	415	6

Table 4.6: Result of sector's length by Hector SLAM at 0.22m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	164	4
B	94	95	1
C	168	170	2
D	168	170	2
E	168	171	3
F	210	216	6
G	421	419	2
H	421	413	8
I	421	414	7
J	421	411	10

4.5.3 Google Cartographer (length)

Table 4.7: Result of sector’s length by Google Cartographer at 0.07m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	167	1
B	94	94	0
C	168	168	0
D	168	168	0
E	168	171	3
F	210	219	9
G	421	398	23
H	421	425	4
I	421	420	1
J	421	400	21

Table 4.8: Result of sector’s length by Google Cartographer at 0.14m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	176	8
B	94	99	5
C	168	164	4
D	168	171	3
E	168	162	6
F	210	210	0
G	421	412	9
H	421	415	6
I	421	407	14
J	421	394	27

Table 4.9: Result of sector’s length by Google Cartographer at 0.22m/s

Sector	Actual Data (cm)	System Measurement (cm)	Error (cm)
A	168	173	5
B	94	90	4
C	168	167	1
D	168	173	5
E	168	168	0
F	210	217	7
G	421	392	29
H	421	411	10
I	421	405	16
J	421	383	38

4.5.4 GMapping (degree)

Table 4.10: Result of corners' angle by GMapping at 0.07m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	93	3
B	90	92	2
C	90	91	1
D	90	89	1
E	90	91	1
F	90	92	2
G	90	88	2
H	90	89	1
I	90	88	2
J	90	95	5

Table 4.11: Result of corners' angle by GMapping at 0.14m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	90	0
B	90	93	3
C	90	93	3
D	90	89	1
E	90	91	1
F	90	91	1
G	90	89	1
H	90	91	1
I	90	89	1
J	90	92	2

Table 4.12: Result of corners' angle by GMapping at 0.22m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	93	3
B	90	92	2
C	90	93	3
D	90	88	2
E	90	92	2
F	90	89	1
G	90	91	1
H	90	92	2
I	90	92	2
J	90	86	4

4.5.5 Hector SLAM (degree)

Table 4.13: Result of corners' angle by Hector SLAM at 0.07m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	90	0
B	90	90	0
C	90	90	0
D	90	88	2
E	90	92	2
F	90	90	0
G	90	90	0
H	90	88	2
I	90	91	1
J	90	89	1

Table 4.14: Result of corners' angle by Hector SLAM at 0.14m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	89	1
B	90	92	2
C	90	90	0
D	90	89	1
E	90	91	1
F	90	89	1
G	90	91	1
H	90	85	5
I	90	88	2
J	90	91	1

Table 4.15: Result of corners' angle by Hector SLAM at 0.22m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	90	0
B	90	92	2
C	90	91	1
D	90	90	0
E	90	90	0
F	90	91	1
G	90	91	1
H	90	83	7
I	90	89	1
J	90	93	3

4.5.6 Google Cartographer (degree)

Table 4.16: Result of corners' angle by Google Cartographer at 0.07m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	90	0
B	90	88	2
C	90	90	0
D	90	90	0
E	90	90	0
F	90	89	1
G	90	91	1
H	90	89	1
I	90	91	1
J	90	90	0

Table 4.17: Result of corners' angle by Google Cartographer at 0.14m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	93	3
B	90	92	2
C	90	90	0
D	90	90	0
E	90	90	0
F	90	89	1
G	90	91	1
H	90	91	1
I	90	89	1
J	90	92	2

Table 4.18: Result of corners' angle by Google Cartographer at 0.22m/s

Sector	Actual Data (°)	System Measurement (°)	Error (°)
A	90	91	1
B	90	91	1
C	90	88	2
D	90	93	3
E	90	89	1
F	90	93	3
G	90	88	2
H	90	90	0
I	90	91	1
J	90	92	2

4.6 Result Interpretation

While the calculation of absolute error provides a preliminary measure of the accuracy of the SLAM methods, it is not sufficient for a comprehensive interpretation of the results. The absolute error only provides a snapshot of the discrepancies between the generated maps and the real world at specific points. It does not provide a holistic view of the overall performance of the SLAM methods.

To gain a deeper understanding of the performance of the SLAM methods and to interpret the results more effectively, additional statistical tools will be introduced in the subsequent subchapters (4.2.1 to 4.2.5). These tools, which include Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE), offer more nuanced insights into the accuracy of the SLAM methods.

These statistical measures will be applied to the data collected from the maps generated by the SLAM methods. By analyzing these measures, a more detailed understanding of the performance of each SLAM method can be obtained. This will allow for a more informed comparison of the SLAM methods and a more accurate assessment of their suitability for different tasks and environments.

4.6.1 Mean Absolute Error (MAE)

MAE is a measure of errors between paired observations expressing the same phenomenon. It is calculated as the average of the absolute differences between the predicted and actual values. The formula is as below:

$$MAE = \frac{\sum_{i=1}^{n} |x_m - x_r|}{n}$$

Where x_m is the measured value in the Rviz, x_r is the real value, and n is the number of samples. MAE is particularly useful when the magnitude of errors is required without considering their direction. It is less sensitive to outliers compared to metrics that square the error. It gives an idea of how wrong the predictions were. The measure gives an equal weight to all errors, whether they are big or small. The closer the MAE to 0, the better the model.

- Advantages: It is robust to outliers and easy to interpret as it provides the average error in the same unit as the output.
- Disadvantages: It does not reflect the impact of the squared errors as all errors are weighted equally.

4.6.2 Mean Squared Error (MSE)

MSE measures the average of the squares of the errors — that is, the average squared difference between the measured and actual values. The formula of MSE is as below:

$$MSE = \frac{\sum_{i=1}^{n}(x_m - x_r)^2}{n}$$

Where x_m is the measured value in the Rviz, x_r is the real value, and n is the number of samples. MSE is used when larger errors are to be highlighted. It is more sensitive to outliers compared to MAE, because those errors get squared. So, the MSE will be larger than the MAE for the same dataset. The closer the MSE to 0, the better the model.

- Advantages: It penalizes larger errors that can be useful in many real-world problems.
- Disadvantages: The squaring makes it harder to interpret than MAE, and it may overemphasize the impact of outliers.

4.6.3 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE. It measures the standard deviation of the residuals. The formula of RMSE is as below:

$$RMSE = \sqrt{MSE} = \sqrt{\frac{\sum_{i=1}^{n}(x_m - x_r)^2}{n}}$$

Where x_m is the measured value in the Rviz, x_r is the real value, and n is the number of samples. This is the square root of the MSE. It’s in the same units as the measured and real value, which can make it easier to interpret than MSE. Like MSE, RMSE is more sensitive to large errors than MAE. The closer the RMSE to 0, the better the model.

- Advantages: It penalizes large errors and is in the same unit as the output, which makes it more interpretable than MSE.
- Disadvantages: It is also sensitive to outliers, and a low RMSE does not necessarily mean a good model if the model is biased.

4.6.4 Mean Absolute Percentage Error (MAPE)

MAPE measures the size of the error in percentage terms. It is calculated as the average of the absolute percent error for each time period. The formula of MAPE is as follow:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{|x_m - x_r|}{x_r}$$

Where x_m is the measured value in the Rviz, x_r is the real value, and n is the number of samples. MAPE is often used when the prediction error in terms of percentage is concerned, which makes it scale-independent. It gives an idea of the error rate. The MAPE can be more intuitive than other metrics, as it’s expressed in percentages. A lower MAPE value means that the measured values are more accurate, with fewer errors. Specifically, it means that the average difference between the measured and actual values (expressed as a percentage of the actual values) is smaller.

- Advantages: It is easy to interpret and provides a standardized measure of error.
- Disadvantages: It can lead to division by zero if the actual value is zero, and it puts a heavier penalty on negative errors when actual values are low.

4.6.5 Results Interpretation (Length)

Table 4.19: Results Interpretation for a) GMapping, b) Hector SLAM, and c) Cartographer, in cm.

GMapping	MAE (cm)	MSE (cm^2)	RMSE (cm)	MAPE (%)
0.07m/s	7.20	69.60	8.34	2.64
0.14m/s	8.40	111.20	10.55	2.90
0.22m/s	10.30	147.90	12.16	3.79

(a)

Hector SLAM	MAE (cm)	MSE (cm^2)	RMSE (cm)	MAPE (%)
0.07m/s	4.10	21.70	4.66	1.97
0.14m/s	4.00	19.60	4.43	1.59
0.22m/s	4.50	28.70	5.36	1.69

(b)

Cartographer	MAE (cm)	MSE (cm^2)	RMSE (cm)	MAPE (%)
0.07m/s	6.20	107.80	10.38	1.83
0.14m/s	8.20	119.20	10.92	3.11
0.22m/s	11.50	275.70	16.60	3.62

(c)

The results of the study, as presented in Table 4.19, clearly indicate that the speed of the robot significantly impacts the mapping accuracy of all the SLAM algorithms, as evidenced by the increase in the values of Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) with an increase in speed. This underscores the importance of the robot's speed in achieving accurate mapping.

A comparative analysis of the MAPE values, which represent the average difference between the measured and actual values, reveals that Hector SLAM consistently maintains a MAPE value of under 2%, even at higher speeds. This suggests that Hector SLAM is highly robust and capable of accurately mapping all the walls in the real world, irrespective of the speed of the robot.

In contrast, while Cartographer performs optimally at lower speeds, its performance decreased significantly as the speed increases, resulting in a high MAPE value. This indicates that Cartographer may struggle to map the surroundings accurately at higher speeds.

GMapping, on the other hand, exhibits the highest MAPE value among all the SLAM algorithms, indicating its relatively poor performance in terms of mapping accuracy.

4.6.6 Results Interpretation (Degree)

Table 4.20: Results Interpretation for a) GMapping, b) Hector SLAM, and c) Cartographer, in degree.

GMapping	**MAE (˚)**	**MSE (˚)^2**	**RMSE (˚)**	**MAPE (%)**
0.07m/s	2.00	5.40	2.32	2.22
0.14m/s	1.40	2.80	1.67	1.56
0.22m/s	2.20	5.60	2.37	2.44

(a)

Hector SLAM	**MAE (˚)**	**MSE (˚)^2**	**RMSE (˚)**	**MAPE (%)**
0.07m/s	0.80	1.40	1.18	0.89
0.14m/s	1.50	3.90	1.97	1.67
0.22m/s	1.60	6.60	2.57	1.78

(b)

Cartographer	**MAE (˚)**	**MSE (˚)^2**	**RMSE (˚)**	**MAPE (%)**
0.07m/s	0.60	0.80	0.89	0.67
0.14m/s	1.10	2.10	1.45	1.22
0.22m/s	1.60	3.40	1.84	1.78

(c)

Based on Table 4.20, it reveals a direct correlation between the speed of the robot and the quality of mapping, specifically in relation to the angles of corners. As the robot's speed increases, there is a corresponding rise in the values of Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). This indicates that higher speeds may compromise the accuracy of mapping.

When examining the performance of different algorithms, Hector SLAM and Cartographer stand out for their exceptional performance, particularly in terms of MAPE values. Even at the highest robot speeds, these algorithms maintain a low MAPE value of 1.78%. This suggests that they are able to map the angles of corners with high precision and accuracy, closely matching the actual layout.

On the other hand, GMapping exhibits the highest MAPE value, implying a lower level of accuracy in mapping corners. This suggests that the corners mapped by GMapping may not accurately represent the actual layout, particularly at higher robot speeds.

4.7 Discussions

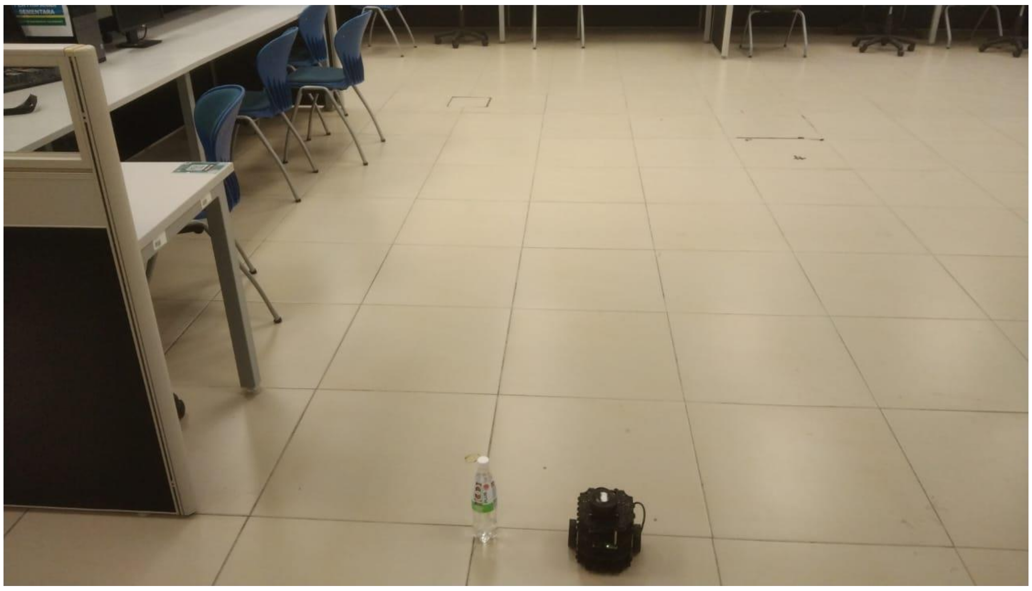


Figure 4.5: Presence of tiles grout between each tiles on the floor.

As shown in Figure 4.5, the floor is composed of tiles separated by grout. The presence of this grout can cause the robot to momentarily halt as it traverses over these lines, which in turn impacts the accuracy of the robot's odometry. This effect is obvious in the case of the Turtlebot3 Burger, which employs a ball caster as its rear wheel, as shown in Figure 4.6.

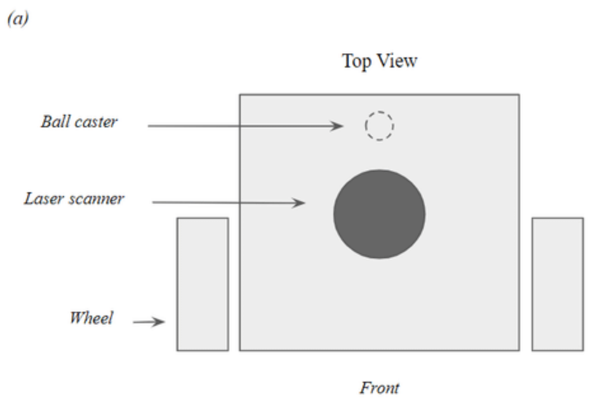


Figure 4.6: Front view of the TurtleBot3 Burger.
Source: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

Furthermore, the smoothness of the floor surface can exacerbate the degradation of the odometry data. Poor odometry, in essence, equates to poor localization, which subsequently leads to suboptimal mapping results. This is because in Simultaneous Localization and Mapping (SLAM), the processes of localization and mapping are interconnected.

In this experiment, GMapping demonstrated the least effective performance among the tested algorithms. At the highest speed of 0.22m/s, its Mean Absolute Percentage Error (MAPE) values soared to significant levels, reaching 3.79% in centimeters and 2.44% in degrees. This is notably higher compared to the other two algorithms.

GMapping's high sensitivity to odometry data proved to be a disadvantage in this context. The presence of tile grout on the road surfaces and the smoothness of the surface led to inaccuracies in the odometry data, which in turn resulted in a higher MAPE. This suggests that GMapping may not be the optimal choice for environments with these specific surface characteristics.

Therefore, when selecting a mapping algorithm for environments with similar surface features, the sensitivity of the algorithm to odometry data should be a key consideration. In this case, GMapping's high sensitivity resulted in less accurate mapping results, indicating that other algorithms may be more suitable for such conditions.

In the context of low-speed operations, Google Cartographer outperforms the other two algorithms. It exhibits the lowest Mean Absolute Percentage Error (MAPE) at 1.83%, which implies a high degree of accuracy in mapping the dimensions of predefined walls in the real world to the 2D LiDAR map. Furthermore, its MAPE for corner angles is also the lowest at 0.67%, indicating that the angles between the walls in the real world closely align with those represented in the 2D LiDAR map.

The superior performance of Google Cartographer in this scenario can be attributed to its utilization of Inertial Measurement Unit (IMU) sensors and pose graph optimization. These features provide additional data that enhance the accuracy of mapping, thereby improving the overall performance of the algorithm. This suggests that Google Cartographer is particularly effective in environments where operations are conducted at lower speeds.

Among the three SLAM methods that are evaluated, Hector SLAM demonstrated the best performance in terms of consistency and accuracy. Hector SLAM achieved relatively low MAPE scores in both the linear and angular dimensions (1.69% in cm and 1.78% in degree), indicating that it was able to estimate the robot pose with high precision. One of the main advantages of Hector SLAM is that it does not rely on odometry data, but instead uses a scan matching algorithm to align the lidar scans. This makes it robust to situations where the robot moves at high speed or experiences wheel slippage, which can degrade the quality of odometry data. Therefore, Hector SLAM is a suitable choice for a wide range of scenarios and environments.

4.7.1 Effect of LiDAR Self Motion Distortion

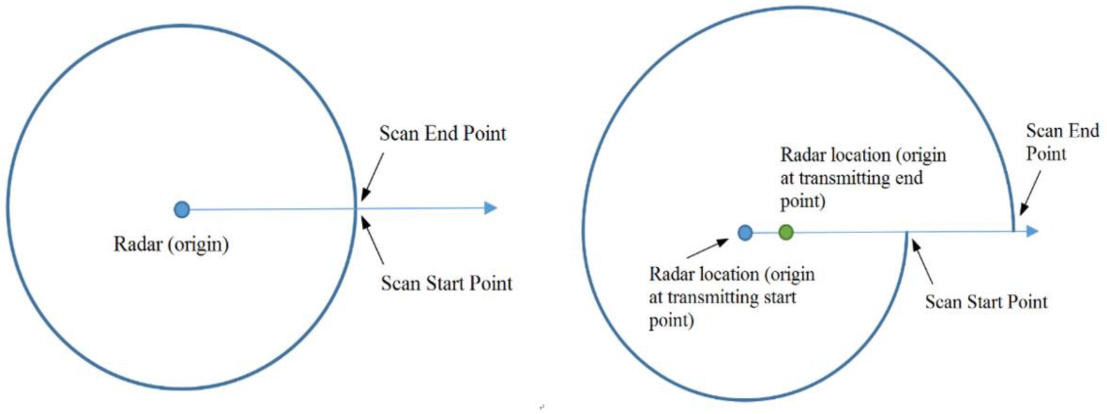

Scan End Point
Radar (origin)
Scan Start Point
Scan End Point
Radar location (origin at transmitting end point)
Radar location (origin at transmitting start point)
Scan Start Point

Figure 4.7: 360-degree mechanical LiDAR self-motion distortion schematic.
Source: https://www.livoxtech.com/showcase/211220

As delineated in Chapter 2.1, LiDAR self-motion distortion refers to the misalignment of the point cloud data captured by a LiDAR sensor during a single revolution. This misalignment, specifically between the first and last points of the scan, is a consequence of the sensor's movement during the scanning process, as shown in Figure 4.7.

In the context of this project, the LiDAR sensor is mounted on a TurtleBot3 Burger robot. The duration of a single revolution of the LiDAR sensor on this robot is 0.2 milliseconds. The robot operates at three different velocities: 0.07m/s, 0.14m/s, and 0.22m/s.

Given these velocities, the distortion in the measured angle position from the beginning to the end of a single LiDAR revolution is calculated to be 1.4cm, 2.8cm, and 4.4cm respectively. This means that the faster the robot moves, the greater the distortion in the LiDAR data, which can impact the accuracy of the mapping and localization tasks performed by the robot. The impact of LiDAR self-motion distortion towards different SLAM algorithms can be clearly seen through this experiment, in which the higher the speed of the robot, the higher the value of MAE, MSE, RMSE, and MAPE, which is due to the occurrence of LiDAR self-motion distortion and the odometry error accumulated. This is a crucial factor to consider when deploying SLAM algorithms in real-world applications.

CHAPTER 5

CONCLUSION

5.1 Introduction

Based on the project’s findings, it can be concluded that the speed of the robot and the choice of SLAM algorithm significantly impact the quality of mapping, particularly in environments with sharp edges. The study evaluated three SLAM algorithms: GMapping, Hector SLAM, and Google Cartographer, under three different speed conditions.

Hector SLAM emerged as the most robust and accurate algorithm, maintaining low Mean Absolute Percentage Error (MAPE) values even at higher speeds. This can be attributed to its independence from odometry data, making it less susceptible to LiDAR self-motion distortions. Google Cartographer performed optimally at lower speeds, but its performance declined as the speed increased. This suggests that while it can be effective in certain scenarios, its use may be limited in high-speed applications. GMapping, on the other hand, showed the highest sensitivity to odometry data and was the least accurate of the three algorithms. Its performance was particularly affected by higher speeds, leading to increased MAPE values.

For future work, it would be beneficial to explore the integration of additional sensors to improve odometry data or the application of these algorithms in different environmental conditions. Furthermore, the development of methods to mitigate the effects of LiDAR self-motion distortions could enhance the performance of SLAM algorithms, particularly in high-speed scenarios. This project serves as a valuable reference for selecting appropriate SLAM algorithms based on specific operational parameters and environmental conditions.

REFERENCES

[1] K. Schwab, *The fourth industrial revolution*. 2017. Accessed: Oct. 18, 2023. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=ST_FDAAAQBAJ&oi=fnd&pg=PR7&ots=DUoBbOyyZJ&sig=DZbtDlEziR86YGlCcH1gRosIXqM

[2] L. Monostori, B. Csáji, B. Kádár, … A. P.-A. reviews in, and undefined 2010, “Towards adaptive and digital manufacturing,” *Elsevier*, Accessed: Oct. 18, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578810000131

[3] L. Monostori, P. Valckenaers, A. Dolgui, … H. P.-A. R. in, and undefined 2015, “Cooperative control in production and logistics,” *Elsevier*, Accessed: Oct. 18, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578815000024

[4] R. Siegwart, I. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. 2011. Accessed: Oct. 18, 2023. [Online]. Available: https://books.google.com/books?hl=en&lr=&id=4of6AQAAQBAJ&oi=fnd&pg=PP1&ots=2x6d0ZxMLZ&sig=cine9WCuGLxCO9jhFDvn0xKpcss

[5] C. Cadena, L. Carlone, H. Carrillo, … Y. L.-I. T., and undefined 2016, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *ieeexplore.ieee.orgC Cadena, L Carlone, H Carrillo, Y Latif, D Scaramuzza, J Neira, I Reid, JJ LeonardIEEE Transactions on robotics, 2016•ieeexplore.ieee.org*, Accessed: Oct. 18, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7747236/

[6] S. Thrun, “Simultaneous localization and mapping,” *Springer Tracts in Advanced Robotics*, vol. 38, pp. 13–41, 2008, doi: 10.1007/978-3-540-75388-9_3.

[7] A. Haider *et al.*, “Modeling of Motion Distortion Effect of Scanning LiDAR Sensors for Simulation-based Testing,” *techrxiv.orgA Haider, L Haas, S KOYAMA, L ELSTER, MH KÖHLER, M SCHARDT, T ZEH, H INOUEAuthorea Preprints, 2023•techrxiv.org*, 2023, doi: 10.36227/techrxiv.24297064.v1.

[8] L. Haas *et al.*, “Velocity Estimation from LiDAR Sensors Motion Distortion Effect,” *Sensors 2023, Vol. 23, Page 9426*, vol. 23, no. 23, p. 9426, Nov. 2023, doi: 10.3390/S23239426.

[9] L. Gröll and A. Kapp, “Effect of fast motion on range images acquired by lidar scanners for automotive applications,” *IEEE Transactions on Signal Processing*, vol. 55, no. 6 II, pp. 2945–2953, Jun. 2007, doi: 10.1109/TSP.2007.893945.

[10] W. Yang, Z. Gong, B. Huang, and X. Hong, “Lidar With Velocity: Correcting Moving Objects Point Cloud Distortion From Oscillating Scanning Lidars by Fusion With Camera,” *IEEE Robot Autom Lett*, vol. 7, no. 3, pp. 8241–8248, Jul. 2022, doi: 10.1109/LRA.2022.3187506.

[11] K. Wahlqvist, “A Comparison of Motion Priors for EKF-SLAM in Autonomous Race Cars”.

[12] R. Yagfarov, M. Ivanou, and I. Afanasyev, “Map Comparison of Lidar-based 2D SLAM Algorithms Using Precise Ground Truth,” *2018 15th International Conference on Control, Automation, Robotics and Vision, ICARCV 2018*, pp. 1979–1983, Dec. 2018, doi: 10.1109/ICARCV.2018.8581131.

[13] Z. Chong, B. Qin, … T. B.-2013 I., and undefined 2013, “Mapping with synthetic 2D LIDAR in 3D urban environment,” *ieeexplore.ieee.orgZJ Chong, B Qin, T Bandyopadhyay, MH Ang, E Frazzoli, D Rus2013 IEEE/RSJ International Conference on Intelligent Robots and, 2013•ieeexplore.ieee.org*, Accessed: Oct. 26, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6697035/

[14] Y. K. Tee and Y. C. Han, “Lidar-Based 2D SLAM for Mobile Robot in an Indoor Environment: A Review,” *2021 International Conference on Green Energy, Computing and Sustainable Technology, GECOST 2021*, Jul. 2021, doi: 10.1109/GECOST52368.2021.9538731.

[15] S. F. Andriawan Eka Wijaya, D. Setyo Purnomo, E. B. Utomo, and M. Akbaryan Anandito, “Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique,” *IES 2019 - International Electronics Symposium: The Role of Techno-Intelligence in Creating an Open Energy System Towards Energy Democracy, Proceedings*, pp. 238–241, Sep. 2019, doi: 10.1109/ELECSYM.2019.8901657.

[16] Z. Liu *et al.*, “Analysis of Mobile Robot Indoor Mapping using GMapping Based SLAM with Different Parameter”, doi: 10.1088/1757-899X/705/1/012037.

[17] H. Thomas, M. Gallet De Saint Aurin, J. Zhang, and T. D. Barfoot, “Learning Spatiotemporal Occupancy Grid Maps for Lifelong Navigation in Dynamic Scenes”, Accessed: Jan. 08, 2024. [Online]. Available: https://github.com/utiasASRL/Deep-Collison-Checker

[18] S. Thrun, “Learning Occupancy Grid Maps With Forward Sensor Models”.

[19] X. Mu, H. Ye, D. Zhu, T. Chen, and T. Qin, “Inverse Perspective Mapping-Based Neural Occupancy Grid Map for Visual Parking,” *Proc IEEE Int Conf Robot Autom*, vol. 2023-May, pp. 8400–8406, May 2023, doi: 10.1109/ICRA48891.2023.10160849.

[20] S. F. Andriawan Eka Wijaya, D. Setyo Purnomo, E. B. Utomo, and M. Akbaryan Anandito, “Research Study of Occupancy Grid map Mapping Method on Hector SLAM Technique,” *IES 2019 - International Electronics Symposium: The Role of Techno-Intelligence in Creating an Open Energy System Towards Energy Democracy, Proceedings*, pp. 238–241, Sep. 2019, doi: 10.1109/ELECSYM.2019.8901657.

[21] A. Rogers, K. Eshaghi, G. Nejat, and B. Benhabib, “Occupancy Grid Mapping via Resource-Constrained Robotic Swarms: A Collaborative Exploration Strategy,” *Robotics*, vol. 12, no. 3, pp. 70–70, May 2023, doi: 10.3390/ROBOTICS12030070.

[22] S. Sunil, S. Mozaffari, R. Singh, B. Shahrrava, and S. Alirezaee, “Feature-Based Occupancy Map-Merging for Collaborative SLAM,” *Sensors*, vol. 23, no. 6, pp. 3114–3114, Mar. 2023, doi: 10.3390/S23063114.

[23] Y. Ren, Y. Cai, F. Zhu, S. Liang, and F. Zhang, “ROG-Map: An Efficient Robocentric Occupancy Grid Map for Large-scene and High-resolution LiDAR-based Motion Planning,” Feb. 2023, Accessed: Jan. 08, 2024. [Online]. Available: http://arxiv.org/abs/2302.14819

[24] E. Jharko, M. Mamchenko, and S. P. Khripunov, “Robot/UAV Indoor Visual SLAM in Smart Cities Based on Remote Data Processing,” *Proceedings - 2023 International Russian Smart Industry Conference, SmartIndustryCon 2023*, pp. 504–508, 2023, doi: 10.1109/SMARTINDUSTRYCON57312.2023.10110777.

[25] T. Lv, J. Zhang, and Y. Chen, “A SLAM Algorithm Based on Edge-Cloud Collaborative Computing,” *J Sens*, vol. 2022, pp. 1–17, Nov. 2022, doi: 10.1155/2022/7213044.

[26] E. Candes, T. T.-I. transactions on information theory, and undefined 2005, “Decoding by linear programming,” *ieeexplore.ieee.org*, Accessed: Jan. 08, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1542412/

[27] G. Grisetti, … C. S.-I. transactions on, and undefined 2007, “Improved techniques for grid mapping with rao-blackwellized particle filters,” *ieeexplore.ieee.orgG Grisetti, C Stachniss, W BurgardIEEE transactions on Robotics, 2007•ieeexplore.ieee.org*, Accessed: Feb. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/4084563/?casa_token=PVQyov9W4hMA

AAAA:J4FmT9dH86zixiIeVIpbG43OOOSnZNi5tXUD5sdj5-yrWn75BxPQ3XbG-ybu2gZjiOw3ebhaLw

[28] S. Kohlbrecher, O. Von Stryk, … J. M.-… symposium on safety, and undefined 2011, "A flexible and scalable SLAM system with full 3D motion estimation," *ieeexplore.ieee.orgS Kohlbrecher, O Von Stryk, J Meyer, U Klingauf2011 IEEE international symposium on safety, security, and rescue, 2011•ieeexplore.ieee.org*, Accessed: Feb. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6106777/?casa_token=fwl7HAPybGsAAAAA:nnfiO0QOTk6iz45ngZ9fhP_61LgDM0EsXvivdl3dDtUmcUYoA9C8XtpqdLojFVgFkkUMCYiPSQ

[29] K. Konolige, G. Grisetti, … R. K.-2010 I., and undefined 2010, "Efficient sparse pose adjustment for 2D mapping," *ieeexplore.ieee.orgK Konolige, G Grisetti, R Kümmerle, W Burgard, B Limketkai, R Vincent2010 IEEE/RSJ International Conference on Intelligent Robots and, 2010•ieeexplore.ieee.org*, Accessed: Feb. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5649043/?casa_token=5kgrtAG_XP8AAAAA:ZmQ0arOZ1g_S-Hl3rrhQYSU2svHFDLvBNiAr4URtPHCfyFDQat6-EHEzIvSQcr5U2H49vv65Hg

[30] W. Hess, D. Kohler, … H. R.-2016 I. international, and undefined 2016, "Real-time loop closure in 2D LIDAR SLAM," *ieeexplore.ieee.orgW Hess, D Kohler, H Rapp, D Andor2016 IEEE international conference on robotics and automation (ICRA), 2016•ieeexplore.ieee.org*, Accessed: Feb. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7487258/?casa_token=uf3GI2jJvPoAAAAA:I9bwXCHzuhos32Fclb61oqwCcqTBuY054Hp1M5b-MuxPXJtab0Ilq7CR6wHIpz4hVmOsy5mAFQ

[31] P. Renaud, N. Andreff, P. Martinet, and G. Gogu, ", C. Secchi, A. van der Schaft, and C. Fantuzzi 574 Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments................. C. Estrada, J. Neira …," *ieeexplore.ieee.org*, Accessed: Feb. 21, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1492468/

[32] J. C. K. Chow, "Multi-Sensor Integration for Indoor 3D Reconstruction," Feb. 2018, doi: 10.13140/RG.2.2.10534.42566.

[33] G. F. Gusmão, C. R. H. Barbosa, and A. B. Raposo, "Development and Validation of LiDAR Sensor Simulators Based on Parallel Raycasting," *Sensors 2020, Vol. 20, Page 7186*, vol. 20, no. 24, p. 7186, Dec. 2020, doi: 10.3390/S20247186.

[34] M. Galli, R. Barber, … S. G.-2017 I. I., and undefined 2017, "Path planning using Matlab-ROS integration applied to mobile robots," *ieeexplore.ieee.orgM Galli, R Barber, S Garrido, L Moreno2017 IEEE International Conference on Autonomous*

Robot Systems and, 2017•ieeexplore.ieee.org, Accessed: Oct. 28, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7964059/

[35] A. Ará ujo *et al.*, "Integrating Arduino-based educational mobile robots in ROS," *SpringerA Araújo, D Portugal, MS Couceiro, RP RochaJournal of Intelligent & Robotic Systems, 2015•Springer*, vol. 77, no. 2, pp. 281–298, Feb. 2015, doi: 10.1007/s10846-013-0007-4.

[36] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," *lars.mec.ua.pt*, Accessed: Oct. 28, 2023. [Online]. Available: http://lars.mec.ua.pt/public/LAR%20Projects/BinPicking/2016_RodrigoSalgueiro/LIB/ROS/icraoss09-ROS.pdf

[37] Y. Wu, Y. Zhang, H. Hu, and Y. Liu, "A Visual SLAM Navigation System Based on Cloud Robotics," 2010, Accessed: Oct. 28, 2023. [Online]. Available: https://www.cloudrobotics.info/files/papers/ICCR17_paper_7.pdf

[38] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, "Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo," Aug. 2016, Accessed: Oct. 28, 2023. [Online]. Available: http://arxiv.org/abs/1608.05742

[39] L. Zhi, M. X.-2018 I. 3rd A. Information, and undefined 2018, "Navigation and control system of mobile robot based on ROS," *ieeexplore.ieee.orgL Zhi, M Xuesong2018 IEEE 3rd Advanced Information Technology, Electronic and, 2018•ieeexplore.ieee.org*, Accessed: Oct. 28, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8577901/

[40] E. Guizzo and E. Ackerman, "The TurtleBot3 Teacher [Resources_Hands On]," *IEEE Spectr*, vol. 54, no. 8, pp. 19–20, Aug. 2017, doi: 10.1109/MSPEC.2017.8000281.

[41] A. S. Pramudyo, F. Muhammad, and A. D. Angkoso, "Sistem deteksi objek menggunakan metode hsv dan binocular disparity pada turtlebot3," *SETRUM (Sistem-KEndali-Tenaga-elektRonika-telekomUnikasi-koMputer)*, vol. 11, no. 1, Jun. 2022, doi: 10.36055/SETRUM.V11I1.15821.

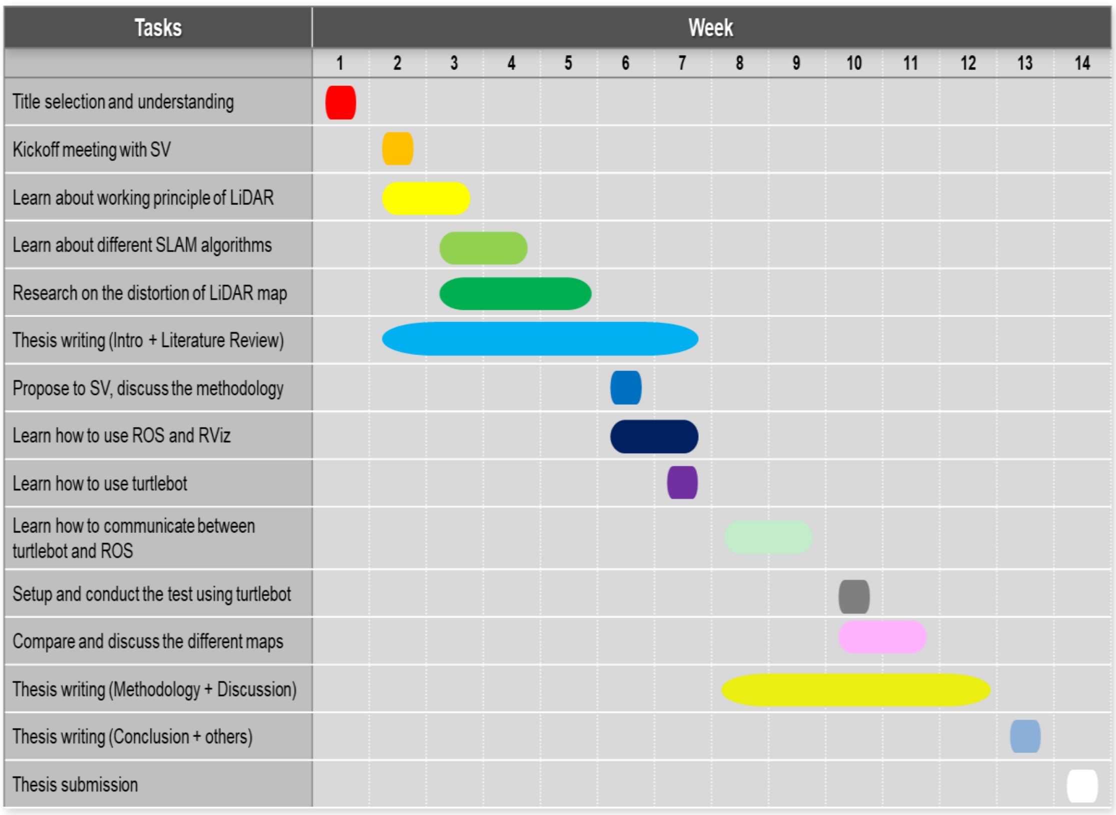
[42] "(PDF) Utilizing ROS 1 and the Turtlebot3 in a Multi-Robot System. (2020) | Corey Williams | 3 Citations." Accessed: Nov. 29, 2023. [Online]. Available: https://typeset.io/papers/utilizing-ros-1-and-the-turtlebot3-in-a-multi-robot-system-1ybm7cx11b

[43] S. Gattu and K. R. Penumacha, “Autonomous Navigation and Obstacle Avoidance using Self-Guided and Self-Regularized Actor-Critic,” *ACM International Conference Proceeding Series*, pp. 52–58, Nov. 2022, doi: 10.1145/3573910.3573914.

[44] “TurtleBot3.” Accessed: Dec. 18, 2023. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/

[45] “360 Laser Distance Sensor LDS-01 for SLAM | Tribotix.” Accessed: Dec. 18, 2023. [Online]. Available: https://tribotix.com/product/360-laser-distance-sensor-lds-01/

APPENDIX A
PROJECT GANTT CHART


Tasks
Week
Title selection and understanding
Kickoff meeting with SV
Learn about working principle of LiDAR
Learn about different SLAM algorithms
Research on the distortion of LiDAR map
Thesis writing (Intro + Literature Review)
Propose to SV, discuss the methodology
Learn how to use ROS and RViz
Learn how to use turtlebot
Learn how to communicate between turtlebot and ROS
Setup and conduct the test using turtlebot
Compare and discuss the different maps
Thesis writing (Methodology + Discussion)
Thesis writing (Conclusion + others)
Thesis submission