ACKNOWLEDGMENTS

First of all, I would like to pay my deepest gratitude to my **Parents** who always encourage, support and taking care of me for all the time. Especially, they give even more support and consider my busy time while I am writing this thesis.

I would like to express my gratitude and thanks to **Mr. PAN Sovanna**, head of Industrial and Mechanical Engineer Department, who has made his effort to lead the department and has looked after every student. Meanwhile, I also would like to thank all of my lectures for teaching me during academic years.

I wish to express my sincere thanks to **Dr. SRANG Sarot**, my advisor, who plays a crucial role during my research at dynamics and control laboratory. I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me. He is open-minded who always love to exchange ideas with students.

Finally, I would like to thank my **Seniors** and **Colleagues** in Dynamics and Control Laboratory who always support and help in many other problems.

អត្ថបទសង្ខេប

Simultaneous Localization and Mapping (SLAM) គឺជារបៀបមួយដែលត្រូវបានប្រើប្រាស់យ៉ាងទូលំទូលាយសម្រាប់ការកំណត់ទីតាំងរបស់រូប៉ូតនិងបង្កើតជាផែនទីនៅក្នុងបរិស្ថានជុំវិញដូចជាលំនៅដ្ឋាន អាគារ សាលារៀន រោងចក្រ ជាដើម។ បើទោះបីជារបៀបមួយនេះត្រូវបានប្រើប្រាស់និងអភិវឌ្ឍន៍យ៉ាងក៏ដោយក៏ទិន័យដែលទទួលបានមានភាពល្អៀងដោយសារតែភាពមិនសុក្រិត (Noise) នៃឧបករណ៍និងបរិស្ថាន។ ដូចនេះហើយទើបឧបករណ៍ (Sensor) ផ្សេងៗត្រូវបានជ្រើសរើសមកប្រើប្រាស់បញ្ចូលគ្នា (Fusion) ដើម្បីធ្វើអោយទិន័យដែលទទួលបានមានភាពប្រសើរដែលអាចទទួលយកបាន។ទិន័យដែលទទួលបានអាចអនុញ្ញាតឱ្យយើងប្រើប្រាស់និងអនុវត្តជាមួយប្រាជ្ញាសិប្បនិម្មិត (AI) ដូចជា ការគ្រង់នៃគន្លងដំណើរ (Path Planning), ការធ្វើដំណើរក្នុងបរិស្ថានដែលមានភាពមិនបិតថេរ (Dynamic Environment Navigation), ការចតយានយន្តស្វ័យប្រវិត្ត(Autonomous Parking), ល។ សារណាមួយនេះនិយាយអំពីការអនុវត្ត Simultaneous Localization and Mapping (SLAM) ដោយប្រើប្រាស់ Light Detection and Ranging Sensor (Lidar)។ MATLAB, Robotic Operation System (ROS), GAZEBO Simulation ត្រូវបានប្រើសំរាប់ធ្វើគំរូ និង Simulate។

RESUME

Simultanés Localisation et Mapping (SLAM) est une méthode qui a été utilisé par tout le monde pour localiser la location de robot et en même temp créer the plan de l'environnement au round de robot comme la maison, le bâtiment, l'université ou l'usine. Malgré le fait que cette méthode a été utilisé est développé de temp a temp, le data qui est obtient par cette méthode n'est pas précis cause par bruit d'appareil de senseur et l'environnement. De cette manière de plus en plus appareils de senseur sont choisi pour faire la fusion de senseur à la suite d'obtenir le data qui est optimise et plus acceptable. Le data qui nous obtenons par le SLAM méthode permet-nous d'utiliser and exécuter l'artificiel intelligent comme Trajectoires Planification (Path Planning), Navigation de Dynamique Environnement (Dynamic Environment Navigation), Autonome Parking (Autonomous Parking), etc. Ce mémoire présente l'application de Simultanés Localisation et Mapping (SLAM) par utilise Light Detection and Ranging Senseur (Lidar). MATLAB, Robotic Operation System (ROS), GAZEBO Simulation sont utilisé pour modéliser et simuler.

ABSTRACT

Simultaneous Localization and Mapping (SLAM) is a method that widely used for localizing the location of the robot and at the same time, create Occupancy Map of the environment such as residence, building, university/school, and factory. Despite the fact that this method is used and developed from time to time, the data acquired is not accurate that cause by noise produced by sensor and the environment surrounding. Thus, multiple sensors were chosen for sensor fusion in pursuit of obtaining the synthesized data that is optimized and acceptable. The data acquired from SLAM method allow us to use and implement the Artificial intelligent (AI) such as Robot Path Planning, Dynamic Environment Navigation, Autonomous Parking, etc. This thesis presents the Implementation of Simultaneous Localization and Mapping (SLAM) using Light Detection and Ranging Sensor (LIDAR). MATLAB, Robotic Operation System (ROS), Gazebo Simulation is used for modeling and simulation.

ABBREVIATION AND SYMBOLS

SLAM	Simultaneous Localization and Mapping
IMU	Inertial Measurement Unit
ROS	Robotic Operation System
WMR	Wheeled Mobile Robot
ICR	Instantaneous center of rotation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Background

Simultaneous Localization and Mapping (SLAM) has been known to be one of the methods that has been used to localize the position of Robot in Global Frame at the same time creating the map out of that environment to further and enhance its autonomous functionality. To obtain the most accurate map data, multiple devices have been used in sensor fusion method to produce the best possible result. SLAM method has been widely used with Wheeled Mobile Robot (WMR) in order to acquire the map of the environment that the robot or vehicle currently in. The map of the environment that acquired from this method could be further in used of Path Planning and Autonomous Navigation.

1.2. Objectives

In this project, we present SLAM algorithm to build Occupancy Grid Map of surrounding environment using Lidar by the implementation of Hector SLAM ROS package, Gmapping SLAM ROS package and MATLAB SLAM. Gazebo 3D Simulation Software will be used to simulate the model of WMR that equipped with simulate sensor (Lidar, IMU, Odometry) and the surrounding environment. RVIZ will be used as the visualization tool for Occupancy Map, WMR model, robot trajectories and sensor data.

1.3. Scope

The Occupancy Grid Map is represented in two dimensional (2D) as (x, y). The Two Wheels Differential drive WMR is designed in two dimensional (2D) planar motion (x, y) and the orientation of the robot is represented by the rotation angle ϕ from the global map frame. All the work has been done inside the Gazebo 3D Simulation. The environment is considered to be Indoor. We are using the sensor data that is obtained from the simulation 2D Lidar, IMU, Odometry.

2. LITERATURE REVIEWS

Navigating of robots around the unknown environment has been a challenging problem for the robotic community for the past years. The majority of mobile robot require map of the environment so that it can move inside that place. Building the map from the bottom up is very essential because it reduces the amount of work that involves the installation of mobile robots. In addition, it enables the mobile robot to easily adapt to change without human intervention. As the matter of fact, mapping is one of the core competencies of a truly autonomous robot.(Thrun, 2002)

SLAM is a method that is used for building and updating the map of the environment while the robot is moving inside the unknown environment and localizing itself in the map. SLAM takes all the available sensors that are equipped on the robot to estimate its position and collects scans from any ranging sensor, builds up the occupancy map and determines the location of itself in that map. The trajectory of the mobile robot and landmark are all being estimated from the input data of the sensor online without pre-knowledge of location.

In SLAM, we use a probabilistic approach in order to solve the SLAM problem. We called it Probabilistic SLAM. In this approach the probabilistic distribution is required to be computed during the process. There are two main forms of the SLAM problem: Online SLAM and Full SLAM.

Online SLAM is the process of estimating the posterior map from a recently collected data from sensors at time t

$$p(x_t, m|\, z_{1:t}, u_{1:t}) \qquad \textbf{(Eq 2.1.)}$$

Where x_t is the position at time t

m is the map

$z_{1:t}$ is the measurement

$u_{1:t}$ is the control vector

Full SLAM is the process of calculating the posterior map from the full trajectories of the mobile robot.

$$p(x_{1:t}, m|\, z_{1:t}, u_{1:t}) \qquad \textbf{(Eq 2.2.)}$$

Where $x_{1:t}$ is the position from the initial time to time t

m is the map

$z_{1:t}$ is the measurement

$u_{1:t}$ is the control vector

3. SIMULTANEOUS LOCALIZATION AND MAPPING USING LIDAR

3.1. Occupancy Grid

3.1.1. Introduction / Properties

Occupancy grid map is one of many pieces of information that are required for the mobile robot for navigation tasks such as path planning, driving a round, mapping the environment, and localizing. It is a 2D representation of the environment.

Occupancy grid map consists of many occupancy grids cells. Each of grid cells is represented as an occupied cell or a free cell according to the calculation of binary probability value. In short, a 2D occupancy map is a large set that contains a probability value in every cell.

$$m_{x,y} = \{free, occupied\} = \{0,1\} \quad \textbf{(Eq 3.1.)}$$

The occupied cell is represented by a probability value of (1) with black color and the free cell is represented by a probability value of (0) with white color. On the Assumption of that the cell is either occupied or free, each probability value contain in each cell are independent, and the surrounding environment is static.

Occupancy grid maps are fine-grained grids defined over the continuous space of locations and often used after solving the SLAM problem by some other means, and taking the resulting path estimates for granted.

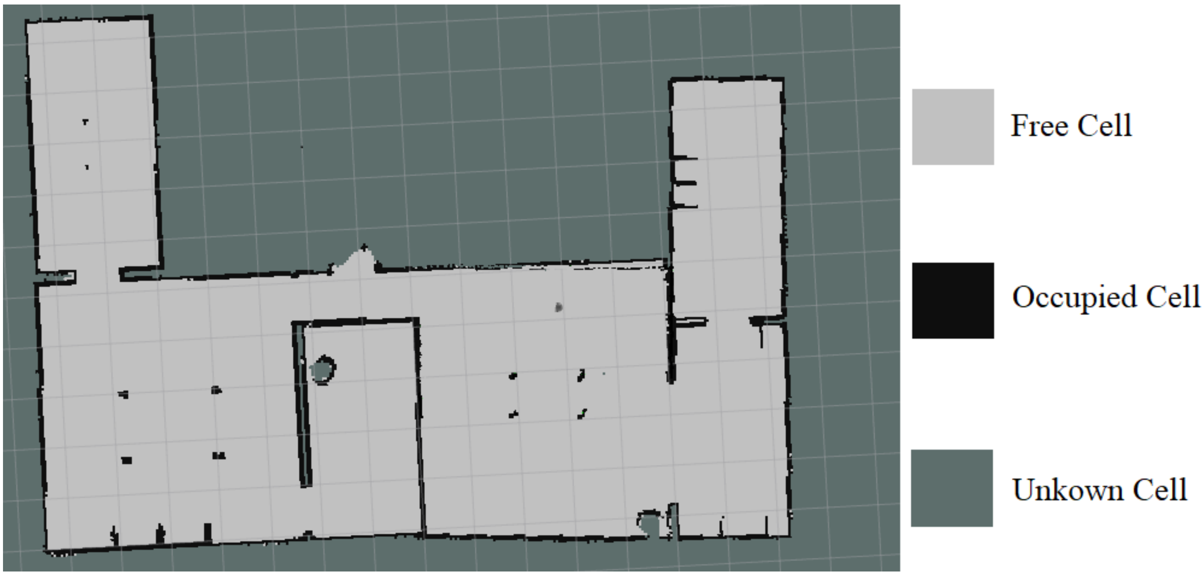


Figure 3.1. Occupancy Grid Map

3.1.2. Occupancy Grid Mapping Algorithm

In Occupancy Grid Mapping Algorithm also known as mapping with known pose the control $u_{1:t}$ is omitted. The main goal of any occupancy grid mapping algorithm is to calculate the posterior over maps given the data represented in probability value

$$p(m \mid z_{1:t}, x_{1:t}) \tag{Eq 3.2.}$$

Where m is the map

$z_{1:t}$ the set of all measurements up to time t

$x_{1:t}$ is the path of the robot, that is, the sequence of all its poses.

In the occupancy grid, each grid cells address is represented with index. The size of grid cell is expressed with map resolution m_r[m/cell].

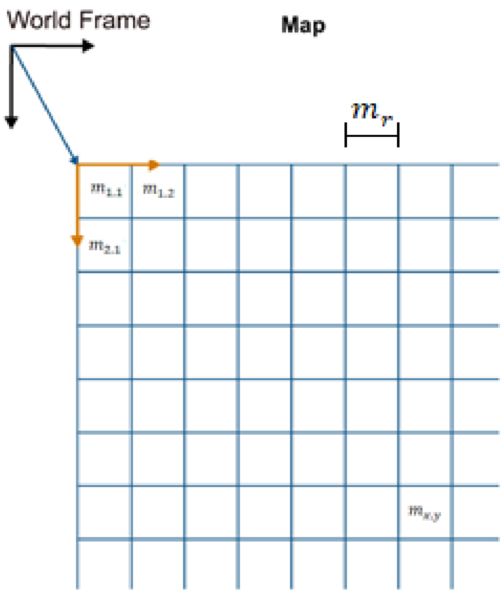


Figure 3.2. Occupancy Grid Map Cell

Let m_i denote the grid cell with index i. An occupancy grid map partitions the space into finitely many grid cells

$$m = \sum_i m_i \tag{Eq 3.3.}$$

The notation $p(m_i = 1)$ or $p(m_i)$ will refer to a probability that a grid cell is occupied.

The standard occupancy grid approach breaks down the problem of estimating the map into a collection of separate problems, namely that of estimating

$$p(m_i \mid z_{1:t}, x_{1:t}) \tag{Eq 3.4.}$$

for all grid cell m_i. Each of these estimation problems is now a binary problem with static state.

This decomposition is convenient but not without problems. In particular, it does enable us to represent dependencies among neighboring cells; instead, the posterior over maps is approximated as the product of its marginals:

$$p(m \mid z_{1:t}, x_{1:t}) = \prod_i p(m_i \mid z_{1:t}, x_{1:t}) \qquad \textbf{(Eq 3.5.)}$$

As the estimation of Occupancy Grid cell has become a static state binary problem, we are using Static State Binary Bayes Filters.

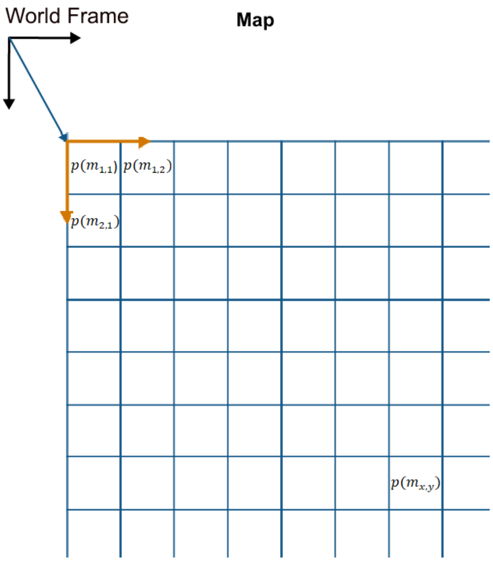


Figure 3.3. Occupancy Grid Map Probability Cell

When using occupancy grids with probability values, the goal is to estimate the probability of obstacle locations for use in real-time robotics applications. The Occupancy Map uses a *log-odds* representation of the probability values for each cell. Each probability value is converted to a corresponding *log-odds* value for internal storage. The value is converted back to probability when accessed. This representation efficiently updates probability values with the fewest operations. Thus, integrate sensor data into the map can be calculate quickly.
(The MathWorks, 2019)

Log-odd Notation

$$odd(x\ event) = \frac{probability\ of\ x\ event\ happen}{probability\ of\ x\ event\ not\ happen} = \frac{p(x)}{p(-x)} = \frac{p(x)}{1 - p(x)} \qquad \textbf{(Eq 3.6.)}$$

The *Log-odd* is the logarithm of above equation

$$l(x) := \log \frac{p(x)}{1 - p(x)} \qquad \textbf{(Eq 3.7.)}$$

$$p(x) = 1 - \frac{1}{1 + expl(x)} \qquad \textbf{(Eq 3.8.)}$$

Substitute in $p(m_i \,|z_{1:t}, x_{1:t})$ with $p(x)$, We get

$$l_{t,i} = \ log \ \frac{p(m_i \,|z_{1:t}, x_{1:t})}{1 \ - \ p(m_i \,|z_{1:t}, x_{1:t})} \quad \begin{matrix} (I) \\ (II) \end{matrix} \qquad \textbf{(Eq 3.9.)}$$

On (I) Using Bayes rule, we get

$$p(m_i \,|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, z_{1:t-1}, x_{1:t}) \, p(m_i|z_{1:t-1}, x_{1:t})}{p(z_t|z_{1:t-1}, x_{1:t})} \qquad \textbf{(Eq 3.10.)}$$

Using Markov Assumption on **Eq 3.10.**, we get

$$p(m_i \,|z_{1:t}, x_{1:t}) = \frac{p(z_t|m_i, x_t) \, p(m_i|z_{1:t-1}, x_{1:t-1})}{p(z_t|z_{1:t-1}, x_{1:t})} \qquad \textbf{(Eq 3.11.)}$$

We have

$$p(z_t|m_i, x_t) = \frac{p(m_i|z_t, x_t) p(z_t|x_t)}{p(m_i|x_t)} \qquad \textbf{(Eq 3.12.)}$$

Substitute **Eq 3.12.** to **Eq 3.11.**, we get

$$p(m_i \,|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t) \, p(z_t|x_t) \, p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i|x_t) \, p(z_t|z_{1:t-1}, x_{1:t})} \qquad \textbf{(Eq 3.13.)}$$

Using Markov Assumption on **Eq 3.13.**, We get

$$p(m_i \,|z_{1:t}, x_{1:t}) = \frac{p(m_i|z_t, x_t) \, p(z_t|x_t) \, p(m_i|z_{1:t-1}, x_{1:t-1})}{p(m_i) \, p(z_t|z_{1:t-1}, x_{1:t})} \qquad \textbf{(Eq 3.14.)}$$

On (II) We have

$$1 \ - \ p(m_i \,|z_{1:t}, x_{1:t}) = \ p(-m_i \,|z_{1:t}, x_{1:t}) \qquad \textbf{(Eq 3.15.)}$$

The same calculation above, we obtain

$$p(-m_i \,|z_{1:t}, x_{1:t}) = \frac{p(-m_i|z_t, x_t) \, p(z_t|x_t) \, p(-m_i|z_{1:t-1}, x_{1:t-1})}{p(-m_i) \, p(z_t|z_{1:t-1}, x_{1:t})} \qquad \textbf{(Eq 3.16.)}$$

Divide **Eq 3.14.** with **Eq 3.16.**

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

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

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

$$\frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} = \frac{p(m_i \mid z_t, x_t)}{p(-m_i \mid z_t, x_t)} \frac{p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(-m_i \mid z_{1:t-1}, x_{1:t-1})} \frac{p(-m_i)}{p(m_i)} \qquad \textbf{(Eq 3.17.)}$$

Apply Log on **Eq 3.17.**

$$log \ \frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} = \log \left(\frac{p(m_i \mid z_t, x_t)}{p(-m_i \mid z_t, x_t)} \frac{p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(-m_i \mid z_{1:t-1}, x_{1:t-1})} \frac{p(-m_i)}{p(m_i)} \right)$$

We get

$$log \ \frac{p(m_i \mid z_{1:t}, x_{1:t})}{1 - p(m_i \mid z_{1:t}, x_{1:t})} = \log \frac{p(m_i \mid z_t, x_t)}{p(-m_i \mid z_t, x_t)} + log \frac{p(m_i \mid z_{1:t-1}, x_{1:t-1})}{p(-m_i \mid z_{1:t-1}, x_{1:t-1})} + log \frac{p(-m_i)}{p(m_i)} \qquad \textbf{(Eq 3.18.)}$$

$$l_{t,i} = l(m_i \mid z_{1:t}, x_{1:t}) = l(m_i \mid z_t, x_t) + l(m_i \mid z_{1:t-1}, x_{1:t-1}) - l(m_i) \qquad \textbf{(Eq 3.19.)}$$

$$l_{t,i} = inverse\_sensor\_model(m_i, x_t, z_t) + l_{t-1,i} - l_0 \qquad \textbf{(Eq 3.20.)}$$

Where

$l(m_i \mid z_t, x_t) = inverse\_sensor\_model(m_i, x_t, z_t)$

$l(m_i \mid z_{1:t-1}, x_{1:t-1}) = l_{t-1,i}$

$l(m_i) = l_0$

Table 3.1. The Occupancy Grid Algorithm, with Binary Bayes Filter (Thrun, 2002)

Occupancy Grid Mapping Algorithm	Line
Algorithm occupancy_grid_mapping$(\{l_{t-1,i}\}, x_t, z_t)$:	1
For all cell m_i do	2
If m_i in perceptual field of z_t then	3
$l_{t,i} = l_{t-1,i} + inverse\_sensor\_model(m_i, x_t, z_t) - l_0$	4
else	5
$l_{t,i} = l_{t-1,i}$	6
endif	7
Endfor	8
return $\{l_{t,i}\}$	9

Eq 3.20. is used to update the occupancy grid cell. In **Table 3.1.**Line4, if the grid cell falls inside the scan area of lidar, the algorithm will return a new grid cell that its probability value change according to the scan if it is occupied or free. In **Table 3.1**.Line 6, if the grid cell does not fall inside the scan area of lidar, its probability value remains unchanged.

3.1.3. Inverse Sensor Model for Lidar

In **Eq 3.20.**, to update the occupancy grid, we incorporate a new the measurement from the lidar and update the existing probability value inside each grid with new calculation from lidar. Given the current WMR position on a cell grid and the measurement from a lidar scan region, we assign the probability value to the cell that lidar beam past through. If the detection happens in cells grid, the probability value of that cells grid will be calculated with a new assign probability value from lidar output represented with l_{occ}, and the cells which shorter from occupied cell will be assign as free cells lidar output represented with l_{free}. If there is not any occupied detection from lidar outside the lidar max scan range, the cells will be considered to be unknown represented with l_0 prior cells.

In this thesis, we assign l_{occ} =0.9

l_{free}=0.4

l_0 =0.5

Table 3.2. Inverse 2D Lidar Sensor Model Algorithm (Steven Waslander, n.d.)

Inverse Lidar Sensor Model Algorithm
Algorithm inverse_lidar_sensor_model(i, x_t, z_t):
Let x_i, y_i be the center of mass of m_i
$r^i = \sqrt{\left(m_x^i - x_{1,t}\right)^2 + \left(m_y^i - x_{2,t}\right)^2}$
$\phi^i = \tan^{-1}\frac{(m_x^i - x_{1,t})}{(m_y^i - x_{2,t})} - x_{3,t}$
$k = argmin_j \lvert \phi^i - \phi_j^s \rvert$
If $r > \min(r_{\max}^s)$
return l_0
If $r_k^s < r_{\max}^s$
return l_{occ}
If $r^i \leq r_k^s$
return l_{free}
endif

Where r is the range from lidar to grid cell

m_x, m_y is the center of cell

x_1, x_2 is sensor location

3.2. Simultaneous Localization and Mapping (SLAM) Algorithm

There is numerous SLAM algorithm that has been develop over the year. One of them is Scan Matching. Scan matching is the process of aligning laser scans with each other (Scan to Scan Matching) or with an existing map (Scan to Map Matching). Using Scan Matching algorithm, the main idea is to find the rigid transformation in robot pose from two different scans. Scan Matching algorithm can be corporate with many different kinds of range finder sensors. Thus, in this thesis we use lidar as our range finder sensor. As the lidar scans data being subscribe with ROS and publish to the algorithm, scans get aligned with the existing map, or another scan and the matching is implicitly performed with all preceding scans.(Kohlbrecher et al., 2011)

To find the rigid transformation of robot pose that make the best lidar scan alignment, we have to find transformation $\xi = \left(p_x, p_y, \phi\right)^T$ that minimizes

$$\xi^* = argmin_\xi \sum_{i=1}^{n} \left[1 - M\left(S_i(\xi)\right)\right]^2 \qquad \textbf{(Eq 3.21.)}$$

Where

$S_i(\xi)$ are the world coordinates of scan endpoint $s_i = \begin{pmatrix} s_{i,x} \\ s_{i,x} \end{pmatrix}$

$S_i(\xi)$ is the function of ξ , the pose of the robot in the world coordinate.

$$S_i(\xi) = \begin{pmatrix} cos\phi & -sin\phi \\ sin\phi & cos\phi \end{pmatrix} \begin{pmatrix} s_{i,x} \\ s_{i,x} \end{pmatrix} + \begin{pmatrix} p_x \\ p_y \end{pmatrix} \qquad \textbf{(Eq 3.22.)}$$

The function $M\left(S_i(\xi)\right)$ return the map value at the coordinate given by $S_i(\xi)$. Given some starting estimate of ξ, we want to estimate $\Delta\xi$ which optimize the error measure according to

$$\sum_{i=1}^{n} \left[1 - M\left(S_i(\xi)\right)\right]^2 \to 0 \qquad \textbf{(Eq 3.23.)}$$

Using the first order of Taylor expansion of $M(S_i(\xi + \Delta\xi))$ we get

$$\sum_{i=1}^{n} \left[1 - M\left(S_i(\xi)\right) - \nabla M\left(S_i(\xi)\right) \frac{\partial S_i(\xi)}{\partial \xi} \Delta\xi)\right]^2 \to 0 \qquad \textbf{(Eq 3.24.)}$$

This equation in minimized by setting the partial derivative with respect to $\nabla\xi$ to zero

$$2 \sum_{i=1}^{n} \left[\nabla M\left(S_i(\xi)\right) \frac{\partial S_i(\xi)}{\partial \xi}\right]^T \left[1 - M\left(S_i(\xi)\right) - \nabla M\left(S_i(\xi)\right) \frac{\partial S_i(\xi)}{\partial \xi} \Delta\xi)\right] = 0 \qquad \textbf{(Eq 3.25.)}$$

Solving for $\Delta\xi$ yields the Gauss-Newton equation for the minimization problem

$$\Delta\xi = H^{-1}\sum_{i=1}^{n}\left[\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\right]^T [1 - M(S_i(\xi))] \qquad \textbf{(Eq 3.26.)}$$

Where

$$H = \left[\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\right]^T \left[\nabla M\big(S_i(\xi)\big)\frac{\partial S_i(\xi)}{\partial \xi}\right]$$

An approximation for the map gradient $\nabla M\big(S_i(\xi)\big)$ is provided is section IV-A in Hector SLAM (Kohlbrecher et al., 2011)

With **Eq 3.22.**, we get

$$\frac{\partial S_i(\xi)}{\partial \xi} = \begin{pmatrix} 1 & 0 & -sin\phi s_{i,x} - cos\phi s_{i,y} \\ 0 & 1 & cos\phi s_{i,x} - sin\phi s_{i,y} \end{pmatrix} \qquad \textbf{(Eq 3.27.)}$$

3.3. Robotic Operating System (ROS)

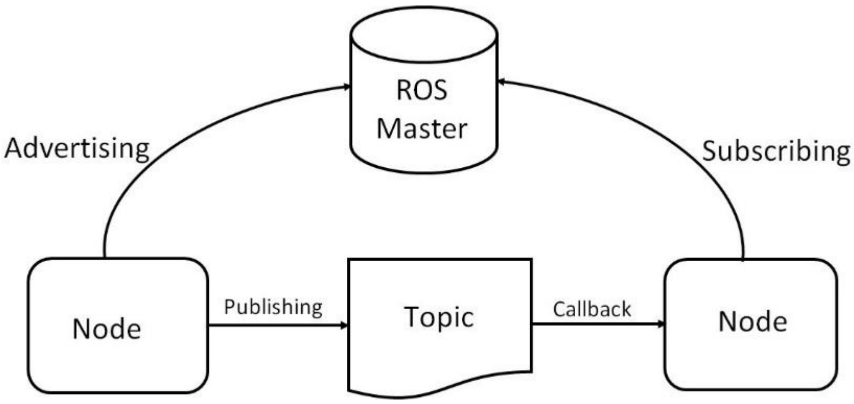


Figure 3.4. ROS Framework

Robotic operating system (ROS) is an open source framework develop for robotic purpose. It contains libraries and package that are already build and ready to use for robot. ROS is a peer-to-peer network of process that could run on multiple devices that connected via network.

ROS center of communication is ROS Master. ROS master act as a keeper of topics and services registration and information of ROS node. ROS nodes are the process of performing the computation. It publishes or subscribes ROS messages with other nodes via ROS topics. ROS message is data that have been simplified into a structure.

3.3.1. Gazebo Simulation

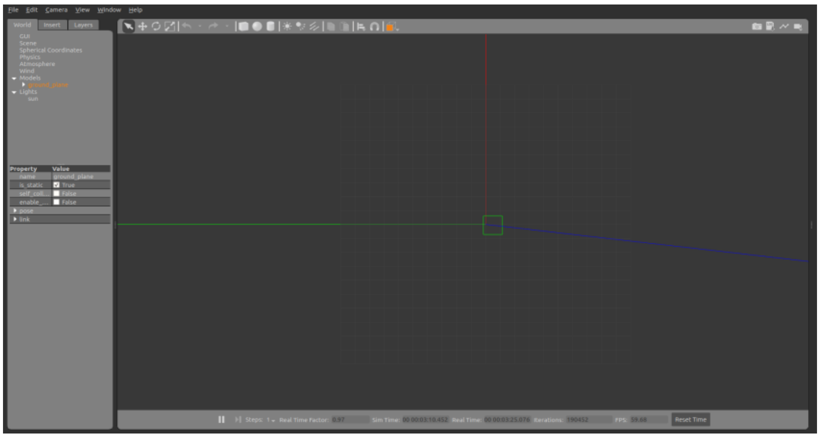


Figure 3.5. Gazebo Simulation Window

Gazebo is an open source physic 3D simulator that allow user to simulate and design the robot and the environment. Gazebo has the ability to simulate a highly accurate robot and sensor with the noise from the environment similarly to a game engine. In this thesis, we use Gazebo simulator a main software for our robot, sensor, and the environment simulation.

3.3.1.1 Gazebo Environment

In Gazebo environment, user can create their own robot accordingly to their design as well as the scenario of the surrounding environment for the experiment. In this thesis, we use Turtlebot3 Burger as a main simulate WMR that are equipped with Lidar, IMU, Odometry sensor. In addition to that, multiple scenarios will be used for SLAM implementation.

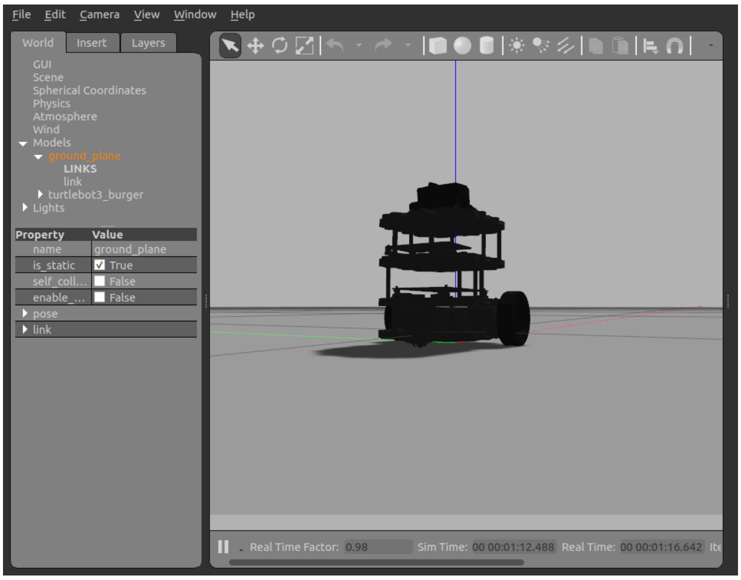


Figure 3.6. Turtlebot3 Burger Simulation Window

3.3.1.2 *Subscribed / Published Topic*

In Gazebo 3D Simulator, the WMR (Turtlebot3 Burger) subscribed to a controller keyboard teleoperation topic in order to move via the command from the user. Gazebo publish multiple ROS massage data such as robot state, link state, lidar, IMU and Odometry sensor, to the ROS topic from the simulator onto ROS master. By using the ROS message from the simulation, we can use it as a simulate data source for SLAM.

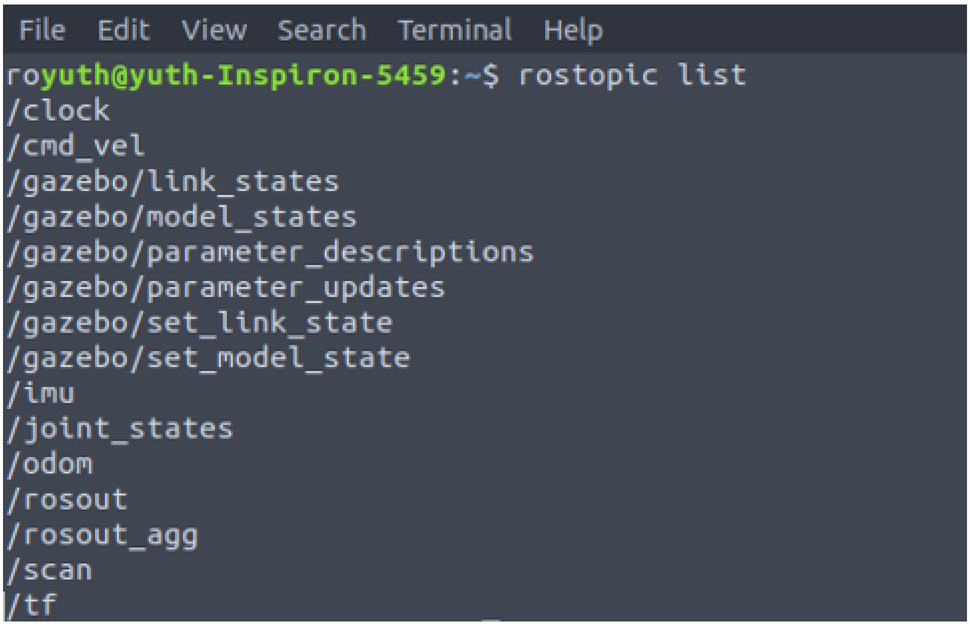


Figure 3.7. Gazebo Simulation ROS Topic

Table 3.3. Publish and Subscribe Simulation Topic

Topic	Message type	Frame ID	Description	Function
/joint_states	/sensor_msgs /JointState		Status of joint in robot	Publish
/cmd_vel	/geometry_msgs /Twist		Command WMR	Subscribe
/imu	/sensor_msgs /Imu	/base_imu	Data from IMU	Publish
/odom	/nav_msgs /Odometry	/odom	Data from Odometry	Publish
/scan	/sensor_msgs /Laserscan	/base_laser	Data from lidar	Publish
/tf	/tf/tfMessage		Transform package	Publish

3.3.2. Coordinate Frame

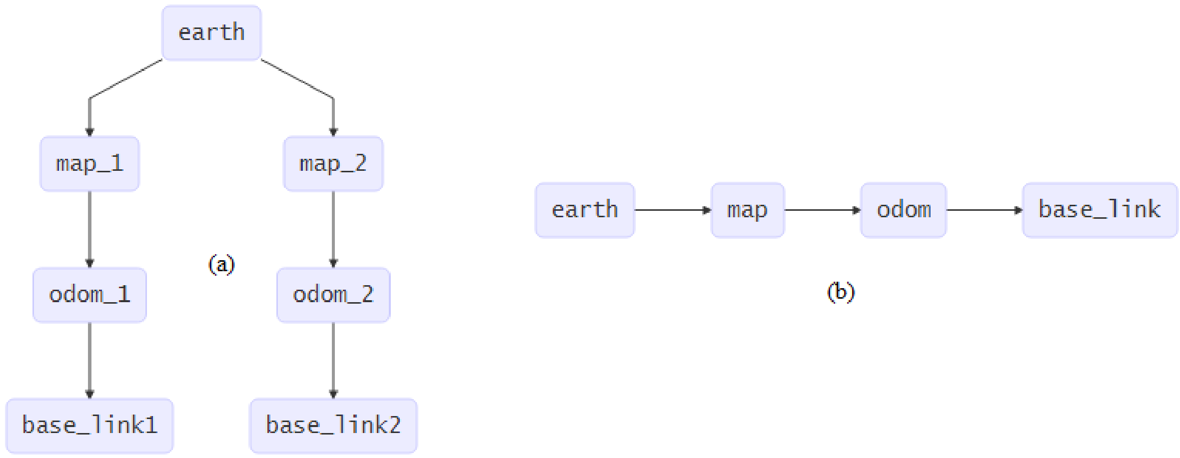


Figure 3.8. Coordinate Frame Relationship (a) Multiple Map; (b) Single Map

In SLAM, there are multiple coordinate frames that we need to utilize. Thus, we must define the base reference of those coordinates. According to ROS community guideline REP[105] (Wim Meeuseen, 2010) Coordinate Frames for Mobile Platform, we define

- **Base_link** or **Base_frame** as the coordinate that attached to mobile robot base. We define it as the point of reference of the WMR. Base_link is main link that WMR sensor is attached to.
- **Odom** as the world-fixed frame that is continuous without discrete jumps. This frame is drift over time that can accumulate the error in long term or in a large-scale map.
- **Map** as the world-fixed frame that is not continuous. In this frame the pose of WMR can be change with discrete jump.
- **Earth** as the coordinate the allow interaction between multiple maps.
- **Laser_frame** as the coordinate that assign to Lidar.

In the Gazebo Simulation, Lidar sensor is attached on top of the WMR. Thus, the when the WMR move, the coordinate of the Lidar move along with it. Map frame and Odom frame, is the static coordinate, while Base frame and Lidar coordinate move it Map frame and Odom frame.

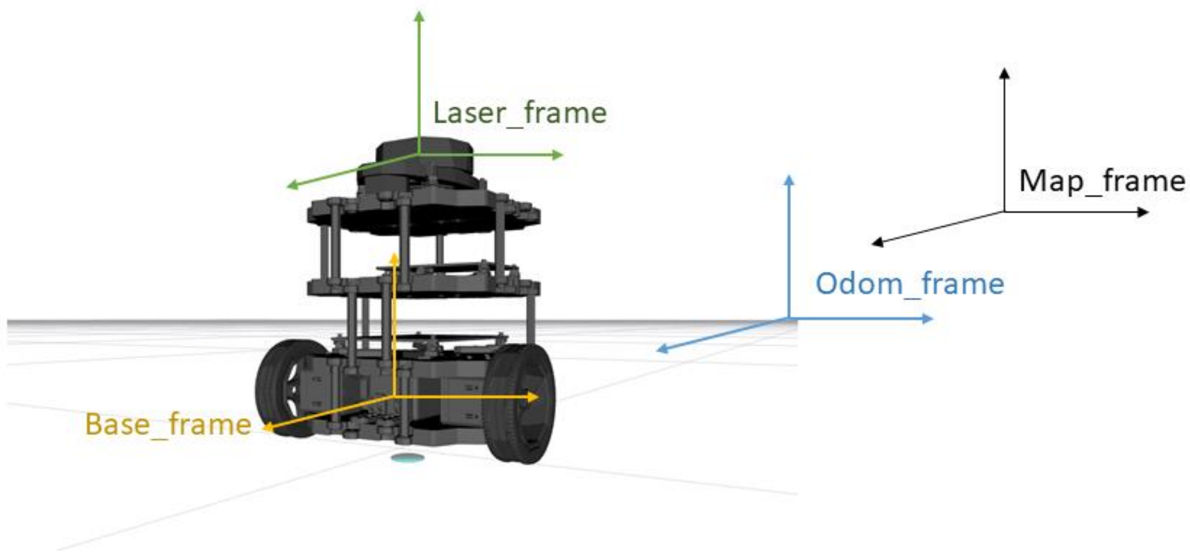


Figure 3.9. Simulation Coordinate Frame

3.3.3. tf Tree

In any physical system there tend to be more than one coordinate frame. Thus, keeping track of all the coordinate frame is very important in our work.

tf the one of the most important ROS packages that enable user to track all coordinate frame as well as maintaining the relationship between each frame and publish it during the operation time, we can compute and transform one frame to another frame with this simple tf package. tf defined the robot with position and orientation. Position is expressed as vector (x, y, z) and the orientation is expressed as quaternion vector form with (x, y, z, w).

In this thesis, tf package for static transform publisher has been used for transform coordinate system. In order to using this package, the following syntax has been used

static_transform_publisher x y z yaw pitch roll frame_id child_frame_id period_in_ms

where

static_transform_publisher is the name of tf package	
x/y/z	is offset [m]
yaw/pitch/roll	is rotation about x, y, z in [rad]
frame_id	is the name of main frame
childframe_id	is the name of the frame that will be transform to
period	is how often to send a transform [ms]

3.3.4. RVIZ

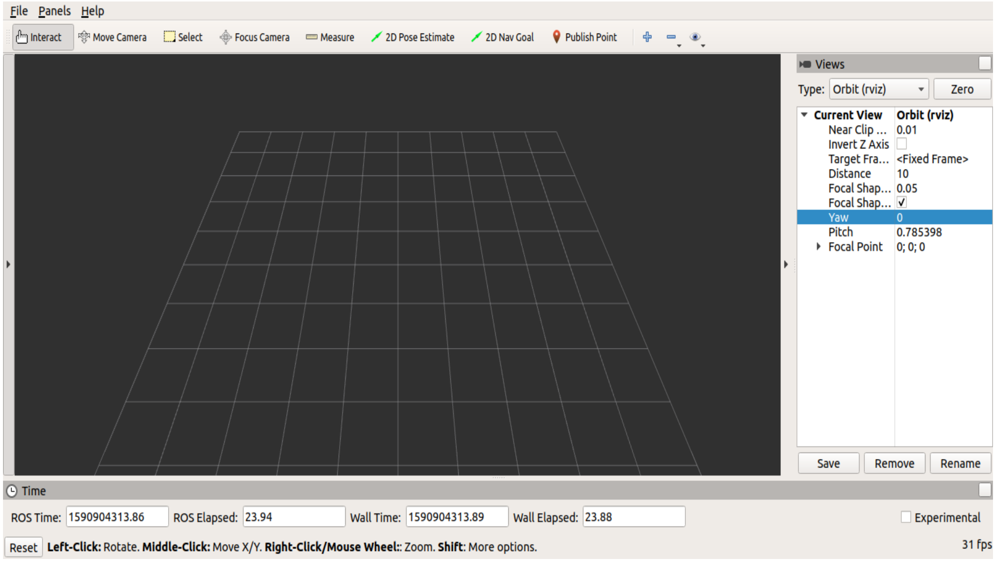


Figure 3.10. RVIZ Window

RVIZ is one of the 3D visualizations tools in ROS. RVIZ allow us to visualize and verify the incoming data from the ROS messages. In this thesis, RVIZ is used to visualize the Occupancy Grid Map, Robot Model, Sensor Data.

3.4. SLAM Packages

There are numerous Open source SLAM packages that developed by many robotic communities over the year. Most notable are Hector SLAM, Gmapping SLAM, and MATLAB SLAM which are used in this thesis for the implementation of SLAM.

3.4.1. Hector SLAM Package (ROS)

Hector SLAM Package is one of many SLAM packages in ROS develop by Team Hector. Hector SLAM use hector_mapping ROS node for SLAM algorithm, provide the map of the environment. Scan Matching SLAM approach has been implemented in order to determine the displacement of WMR in 2D from previous and current consecutive scan. Hector SLAM can be use with or without the odometry data.(Stefan Kohlbrecher, 2012)

Table 3.4. Publish and Subscribe Hector SLAM Topic

Topic	Message Type	Description	Function
/scan	/sensor_msgs /LaserScan	Data from Lidar	Subscribe
/syscommand	/std_msgs /String	User command Reset map	Subscribe
/map_metadata	/nav_msgs /MapMetaData	Map Data	Publish
/map	/nav_msgs /OccupancyGrid	Map Data	Publish
/slam_out_pose	/geometry_msgs /PoseStamped	Robot pose estimated without covariance	Publish
/poseupdata	/geometry_msgs /PoseWithCovarianceStamped	Robot pose estimated with Gaussian estimation of uncertainty	Publish

Table 3.5. Hector SLAM Parameters

Parameters Name	Description
base_frame	Main frame attached to robot for localization
map_frame	Frame of the map
odom_frame	Frame attached to Odometry
map_resolution	Size of grid cell in [m]
map_size	Number of cells per axis
map_start_x	Map origin on x axis
map_start_y	Map origin on y axis
map_update_distance_tresh	Threshold for performing map update in [m]
map_update_angle_tresh	Threshold for performing map update in [m]
map_pub_period	The map publish period in [s]
map_multi_res_levels	The number of map multi-resolution grid levels
updata_factor_free	The map update modifier for free cell l_{free}
update_factor_occupied	The map update modifier for occupied cell l_{occ}

laser_min_dist	The minimum distance for laser scan endpoint to be used in system in [m]
laser_max_dist	The maximum distance for laser scan endpoint to be used in system in [m]
laser_z_min_value	The minimum height relative to the laser_frame
laser_z_max_value	The maximum height relative to the laser_frame
pub_map_odom_transform	Publish map_frame to odom_frame
output_timing	Output timing information for processing of every laser scan
scan_subscriber_queue_size	The queue size of the scan subscriber
pub_map_scanmatch_transform	Publish map_frame to scanmatcher_frame
tf_map_scanmatch_transform_frame_name	Scanmatcher_frame name

Hector SLAM Package Provide tf Transform /map_frame to /odom_frame that estimate the current robot's pose in /map_frame. We have to provide our own tf transform from robot link (/base_link) to lidar link (/laser_frame) using tf static_transform_publisher ROS package.

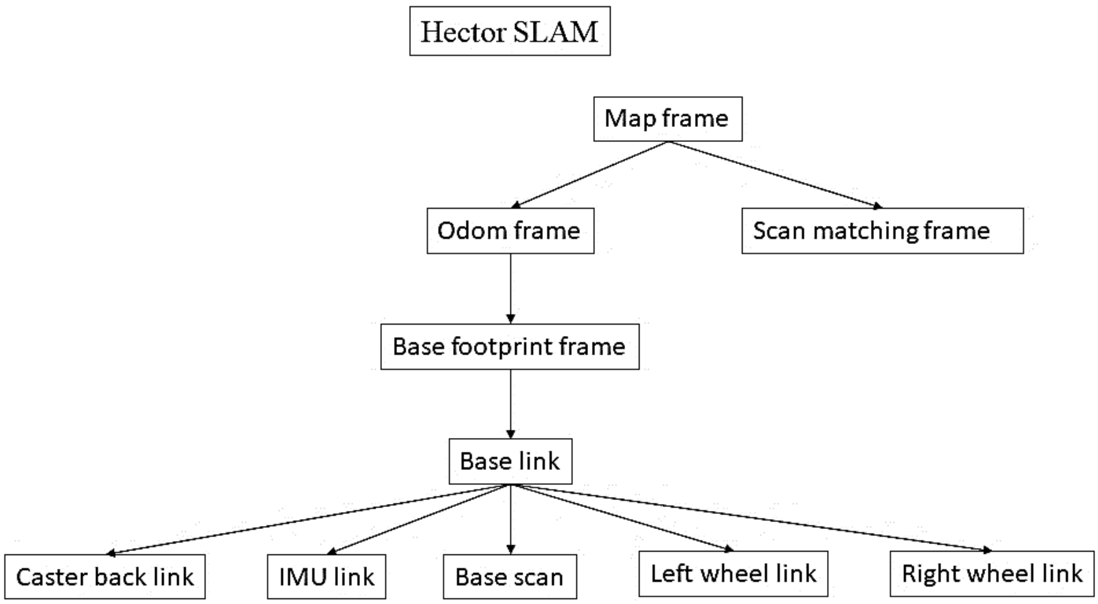


Figure 3.11. Hector SLAM tf frame

3.4.2. Gmapping SLAM Package (ROS)

Gmapping SLAM is another SLAM ROS package which is developed by OpenSlam. Gmapping SLAM provide a laser-based SLAM by using slam_gmapping ROS node. Gmapping SLAM use Scan Matching algorithm for matching the income scans just like Hector SLAM. Using Gmapping SLAM required an odometry information in order to localize the robot pose. (Brian Gerkey, 2019)

Table 3.6. Publish and Subscribe Gmapping SLAM Topic

Topic	Message Type	Description	Function
/scan	/sensor_msgs /LaserScan	Data from Lidar	Subscribe
/tf	/tf/tfMessage	Frame transformation	Subscribe
/map_metadata	/nav_msgs /MapMetaData	Map Data	Publish
/map	/nav_msgs /OccupancyGrid	Map Data	Publish
/entropy	/std_msgs/Float64	Entropy of the distribution estimation over robot's pose (higher is more uncertain)	Publish

Table 3.7. Gmapping SLAM Parameters

Parameters Name	Description
inverted_laser	Laser CCW or CW
throttle_scans	Process 1 out of this many scan (Skip scan)
base_frame	Main frame attached to robot for localization
map_frame	Frame of the map
odom_frame	Frame attached to Odometry
map_update_interval	Duration between map update [s]
maxurange	Maximum usable range of the laser
sigma	The greedy endpoint matching
kernelsize	Look for a correspondence
lstep	The optimization step in translation

astep	The optimization step in rotation
interation	The number of iterations of the scan matcher
lsigma	The sigma of beam used for likelihood computation
ogain	Gain used while evaluating the likelihood for smoothing resampling effects
lskip	Number of beams to skip
minimumscore	Minimum score for considerating the outcome of the scan matching good
srr	Odom error in translation as a function of translation (ρ/ρ)
srt	Odom error in translation as a function of rotation (ρ/θ)
str	Odom error in rotation as a function of translation (θ/ρ)
stt	Odom error in rotation as a function of rotation (θ/θ)
linearupdate	Process a scan each time robot translates
angularupdate	Process a scan each time robot rotates
temporalupdate	Process a scan if the last scan processed is older that update time [s]
resamplethreshold	The Neff based resampling threshold
particles	Number of Particle in the filter
xmin	Initial map size in x axis [m]
xmax	Initial map size in x axis [m]
ymin	Initial map size in y axis [m]
ymax	Initial map size in y axis [m]
delta	Size of grid cell in [m]
llsamplerange	Translational sampling range for the likelihood
llsamplestep	Translational sampling step for the likelihood
lasamplerange	Angular sampling range for the likelihood
lasamplestep	Angular sampling step for the likelihood
transform_publish_period	Duration between transform publish [s]
occ_thresh	Threshold on gmapping's occupancy value.
maxrange	Lidar maximum range

Gmapping SLAM Package Provide tf Transform /map_frame to /odom_frame that estimate the current robot's pose in /map_frame. We have to provide our own tf transform from lidar link (laser_frame) to robot link (base_link) and robot link (base_link) to odometry link (odom) using tf static_transform_publisher.

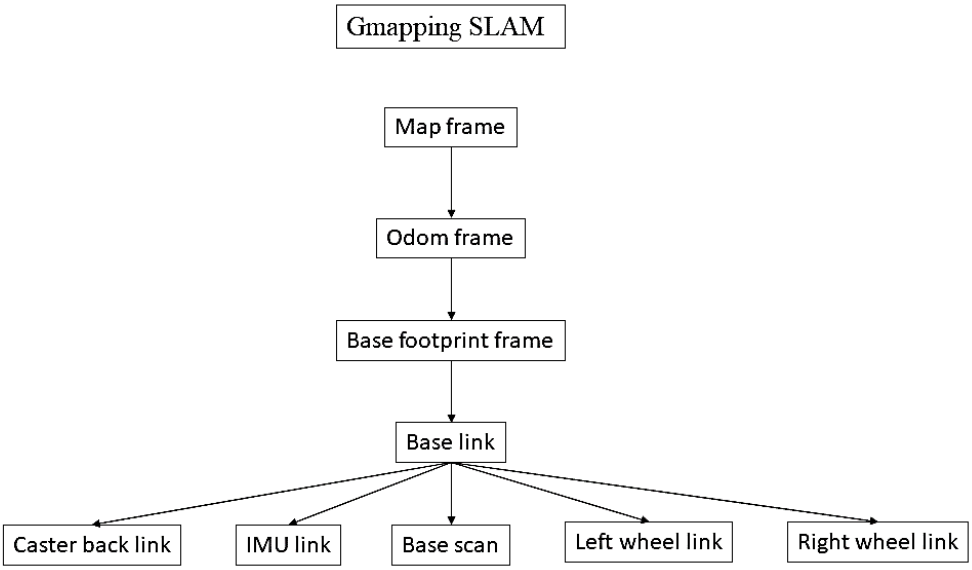


Figure 3.12. Gmapping SLAM tf frame

3.4.3. MATLAB SLAM Package

Using MATLAB SLAM in Navigation Toolbox, we create an object that waiting for the series of incoming scan data from ROS Lidar subscribe function. We use **scansAndPoses** function to retrieve the collected scan and the estimated poses from the class the stored the scans information, then build the occupancy map using **buildMap** function.

Table 3.8. MATLAB SLAM Function

Function	Parameters		Description
	Input	Output	
lidarSLAM	mapResolution maxLidarRange	Lidar Slam Class	Create an object to store lidarscan
lidarScan	Ranges Angles	Lidar data	Create lidarscan object from lidar
addScan	lidarSLAM lidarScan	Collected scans	Add scan to lidarSLAM object

scansAndPoses	lidarSLAM	scans optimizedPoses	Extract scans and poses from lidarSLAM object
buildMap	Scans OptimizedPoses mapResolution maxLidarRange	OccupancyMap	Create an occupancy map

3.5. Wheel Mobile Robot

Wheel Mobile Robot is one of the ground robots that move via the wheel that attached to robot's body frame. There are different types of WMR that can be named after the number of wheels, drive mode, etc. WMR usually has been used as moving platform to move another object. In Robotic Community, WMR are equipped with sensors for further its functionality.

In this thesis, we use the Turtlebot3 Burger two wheels differential drive robot as a moving platform for our simulation.

Figure 3.13. Turtlebot3 Burger

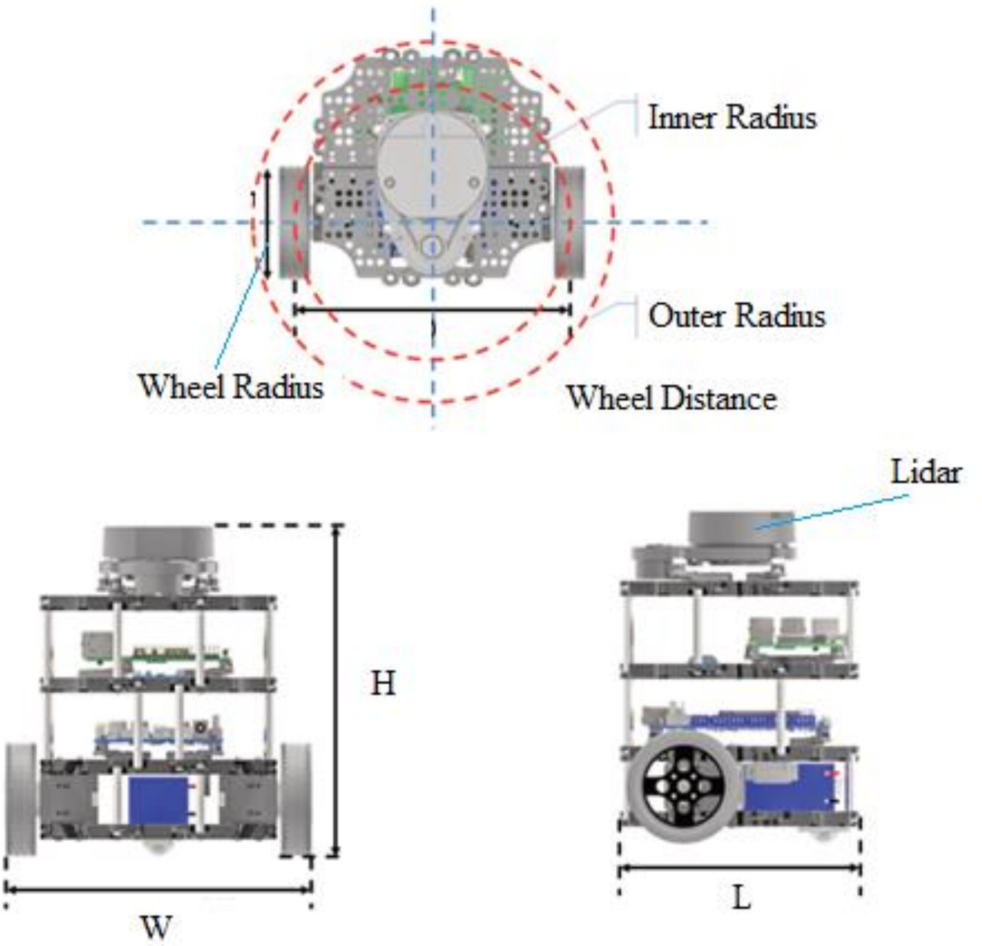


Figure 3.14. Turtlebot3 Burger Specification

Table 3.9. Hardware Specification

Parameters	Value	Unit
Wheel Radius	66	mm
Robot Width (W)	178	mm
Robot Length (L)	138	mm
Robot Height (H)	192	mm
Distance between wheel	160	mm
Robot Outer Radius	105	mm
Robot Inner Radius	80	mm
Maximum Translational velocity	0.22	mm/s
Maximum Rotational velocity	2.84	rad/s

Table 3.10. WMR Sensor

Sensor	Description
Lidar	360 degrees scanner
	Gyroscope 3 Axis
IMU	Accelerometer 3 Axis
	Magnetometer 3 Axis
Odometer	2 Wheels Odometer

3.5.1. Differential Drive WMR Kinematic Model

The WMR pose in 2D plane is defined in state vector as

$$\xi(t) = \begin{bmatrix} p_{x(t)} \\ p_{y(t)} \\ \phi(t) \end{bmatrix} \quad \textbf{(Eq 3.28.)}$$

The Global coordinate frame is represented as (X_g, Y_g), and Robot frame is represented as (X_m, Y_m). Using ROS coordinate system representation as Global Frame is /map_frame and WMR frame is /base_link. The relation between the global frame and robot frame is defined by a translation vector $[x, y]^T$ and rotation matrix about z axis

$$R_z(\phi) = \begin{bmatrix} cos\phi & sin\phi & 0 \\ -sin\phi & cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(Eq 3.29.)}$$

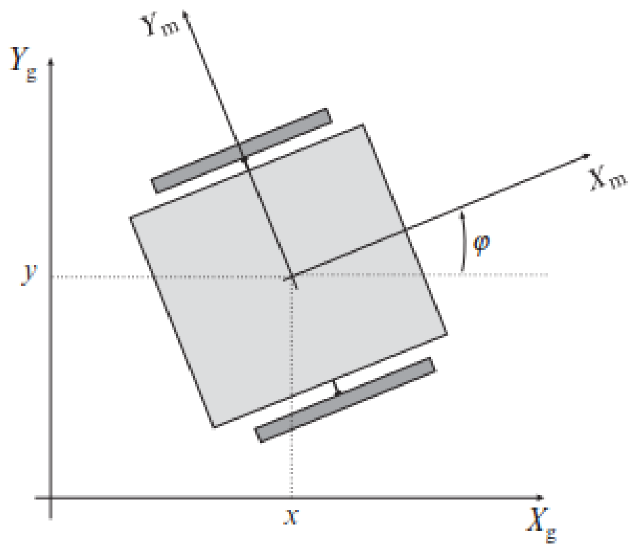


Figure 3.15. 2D Coordinate of Robot

For Two Wheels Differential drive WMR, its movement is depending on the rotation velocity of each wheel. These two wheels is independent to each other, meaning one can rotate faster or slower than the other. To make the rotation, each wheel could rotate in the same or different direction of each other at the different rotation rate. During the rotation of WMR in circular motion, there exist a common point that is intersect of 2-wheel axes called Instantaneous center of rotation (ICR).(Klančar et al., 2017)

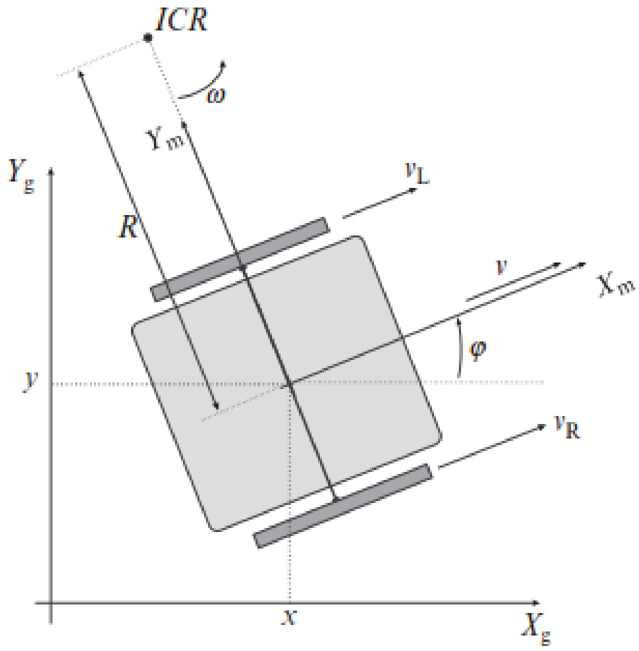


Figure 3.16. Differential Drive Kinematics

Let $v_R(t)$ is the velocity of the right wheel

$v_L(t)$ is the velocity of the left wheel

r is the wheel radius

L is the distance between wheel

$R(t)$ is the ICR

$\omega(t)$ is the angular velocity in which both wheel rotate in the instance of time

$v(t)$ is the WMR tangential velocity

$\omega(t)$ is expressed as

$$\omega(t) = \frac{v_L(t)}{R(t) - L/2} \quad \textbf{(Eq 3.30.)}$$

$$\omega(t) = \frac{v_{R(t)}}{R(t) - L/2} \quad \textbf{(Eq 3.31.)}$$

From **Eq 3.30.** and **Eq 3.31.**

$$\omega(t) = \frac{v_R(t) - v_L(t)}{L} \quad \textbf{(Eq 3.32.)}$$

$$R(t) = \frac{L}{2}\frac{v_R(t) + v_L(t)}{v_R(t) - v_L(t)} \quad \textbf{(Eq 3.33.)}$$

$v(t)$ is expressed as

$$v(t) = \omega(t)R(t) = \frac{v_R(t) + v_L(t)}{2} \quad \textbf{(Eq 3.34.)}$$

Each wheel tangential velocities are expressed as

$$v_L(t) = r\omega_L(t) \quad \textbf{(Eq 3.35.)}$$

$$v_R(t) = r\omega_R(t) \quad \textbf{(Eq 3.36.)}$$

In Local coordinate frame, WMR kinematic

$$\begin{bmatrix} \dot{p_{x_m(t)}} \\ \dot{p_{y_m(t)}} \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} v_{x_m}(t) \\ v_{y_m}(t) \\ \omega(t) \end{bmatrix} = \begin{bmatrix} \frac{r}{2} & \frac{r}{2} \\ -\frac{r}{L} & \frac{r}{L} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix} \quad \textbf{(Eq 3.37.)}$$

In Global coordinate frame, WMR kinematic

$$\begin{bmatrix} \dot{p_{x(t)}} \\ \dot{p_{y(t)}} \\ \dot{\phi}(t) \end{bmatrix} = \begin{bmatrix} \cos(\phi(t)) & 0 \\ \sin(\phi(t)) & 0 \\ 0 & 1 \end{bmatrix} \begin{matrix} v(t) \\ \omega(t) \end{matrix} \quad \textbf{(Eq 3.38.)}$$

Eq 3.38. can be written in discrete Euler integration form with discrete time instant $t = kT_s$ where T_s is the sampling interval and $k = 0,1,2,\ldots$

$$\begin{aligned} p_{x(k+1)} &= p_{x(k)} + v(k)T_s\cos(\phi(k)) \\ p_{y(k+1)} &= p_{y(k)} + v(k)T_s\sin(\phi(k)) \\ \phi(k+1) &= \phi(k) + \omega(k)T_s \end{aligned} \quad \textbf{(Eq 3.39.)}$$

Forward Kinematics

Odometry of the robot is the pose of the robot at time t that is obtained by integration of kinematic model. Odometry is explored more in 3.6.2.

$$\begin{aligned} p_{x(t)} &= \int_0^t v(t)\cos\big(\phi(t)\big)\,dt \\ p_{y(t)} &= \int_0^t v(t)\sin\big(\phi(t)\big)\,dt \\ \phi(t) &= \int_0^t \omega(t)dt \end{aligned} \quad \textbf{(Eq 3.40.)}$$

Case 1: If v and ω is assumed to be constant during sample time $x(k+1), y(k+1), \phi(k+1)$ is expressed in **Eq 3.39.**

Case 2: If trapezoidal numerical integration is used, a better approximation is:

$$p_{x(k+1)} = p_{x(k)} + v(k)T_s\cos\left(\phi(k) + \frac{\omega(k)T_s}{2}\right)$$
$$p_{y(k+1)} = p_{y(k)} + v(k)T_s sin\left(\phi(k) + \frac{\omega(k)T_s}{2}\right) \quad \textbf{(Eq 3.41.)}$$
$$\phi(k+1) = \phi(k) + \omega\ (k)T_s$$

Case 3: If exact integration is applied

$$p_{x(k+1)} = p_{x(k)} + \frac{v(k)}{\omega(k)}(\sin(\phi(k) + \omega(k)T_s) - \sin(\phi(k)))$$
$$p_{y(k+1)} = p_{y(k)} - \frac{v(k)}{\omega(k)}(\cos(\phi(k) + \omega(k)T_s) - \cos(\phi(k))) \quad \textbf{(Eq 3.42.)}$$
$$\phi(k+1) = \phi(k) + \omega\ (k)T_s$$

3.6. Sensor

3.6.1. Light Detection and Ranging Sensor (LIDAR)

Lidar is a remote sensing device that use light pulses laser to detect the distance from an object and has been widely used for multi-purposes including navigation and mapping. Lidar usually contain of laser scanner and DC motor. DC motor rotates the laser scanner in 360 degrees circle in order to obtain a full 360 degrees of the environment. In this thesis, we use the simulation from 2D lidar that scan a 2D slice of a surrounding 3D environment.

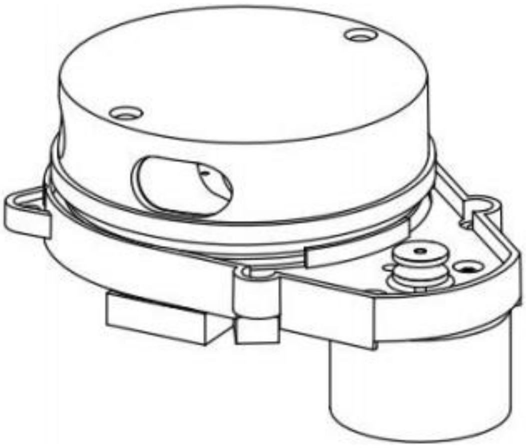


Figure 3.17. Lidar

3.6.1.1 *Coordinate Frame*

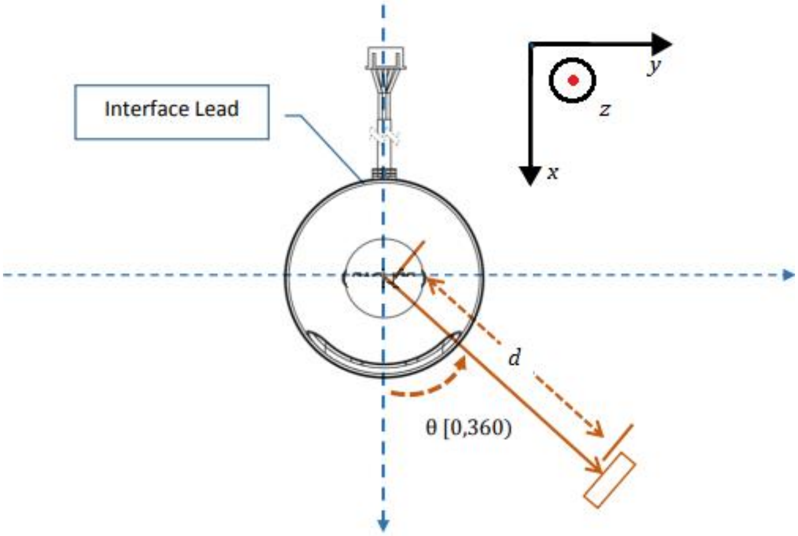


Figure 3.18. Lidar Coordinate System

Lidar Coordinate in 3D Cartesian coordinate system is (x, y, z) where z is pointing upward. But we only use the 2D Lidar simulation data, thus the Lidar Coordinate will be represented in 2D Cartesian coordinate system (x, y) and the rotation angle of θ. The distance from the center of the laser scan to the object is d. When the DC motor rotate, different angle θ and distance d will be recorded, each θ and d are correspond to each other.

$$\theta = [\theta_1, \theta_2, \dots \dots \dots, \theta_{360}]_{(1\times360)}$$

$$d = [d_1, d_2, \dots \dots \dots, d_{360}\,]_{(1\times360)}$$

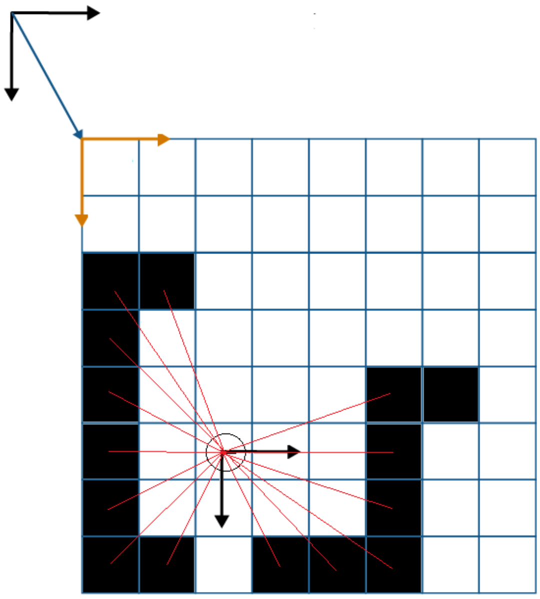


Figure 3.19. Lidar Laser Beam

3.6.1.2 Published ROS Topic

In ROS, Lidar node launch and publish on /scan topic with ROS message type sensor_msg/LaserScan.msg. The Lidar coordinate frame is /laser_frame.

Table 3.11. Lidar Properties

ROS message	Definition	Unit
header	Timestamp and frame_id	
angle_min	Started angle of scan	rad
angle_max	End angle of scan	rad
angle_increment	Angular distance between scan to scan	rad
time_increment	Time between one full scan to one full scan	s
scan_time	Time between scan to scan	s
range_min	Minimum range	m
range_max	Maximum range	m
ranges	Range data in one full scan	m
intensities	Intensity data	

3.6.2. Odometry

Odometry is the estimation of robot position and velocity within the environment using the calculation from the sensor data. Odometry can be computed from odometry source such as wheel odometry, visual odometry (camera), or an IMU.

In Odometry 3D coordinate system the robot position is represented as (P_x, P_y, P_z) and the orientation of the robot is represented as ϕ. As our robot is in 2D planar motion, thus we only interested in $\begin{pmatrix} p_x \\ p_y \\ \phi \end{pmatrix}$ as written in state space form.

When using ROS message, the odometry directly publish with the position and the orientation of the robot from the simulated environment. The position is expressed as (x, y, z) and the orientation in quaternion form (x, y, z, w).

3.6.2.1 Published ROS Topic

In ROS, Odometry node launch and publish on /odom topic with ROS message type nav_msgs/Odometry.msg. The Odometry coordinate frame is /odom_frame.

Table 3.12. Odometry Properties

ROS message	Definition
Header	Timestamp and frame_id
Child_frame_id	
Geometry_msgs/PoseWithCovariance	Estimation of WMR Position in free space with uncertainty
Geometry_msgs/Pose	Contain the information of the position (x, y, z) and orientation in quaternion form (x, y, z, w)
Geometry_msgs/TwistWithCovariance	Estimation of WMR Velocity in free space with uncertainty
Geometry_msgs/Twist	Contain the information of velocity in linear and angular

4. SIMULATION AND DISCUSSION

4.1. Simulation

In this thesis, we use Gazebo Simulation as our data source. Three of SLAM package, Hector SLAM, Gmapping SLAM and MATLAB SLAM, are carrying out. Using keyboard command to control robot movement in Gazebo simulation. During the operation time, the simulation output the sensor data and we subscribe to the data to use in each of SLAM packages. In ROS the Occupancy Grid will be viewed in RVIZ while in MATLAB it will be view in MATLAB Figure.

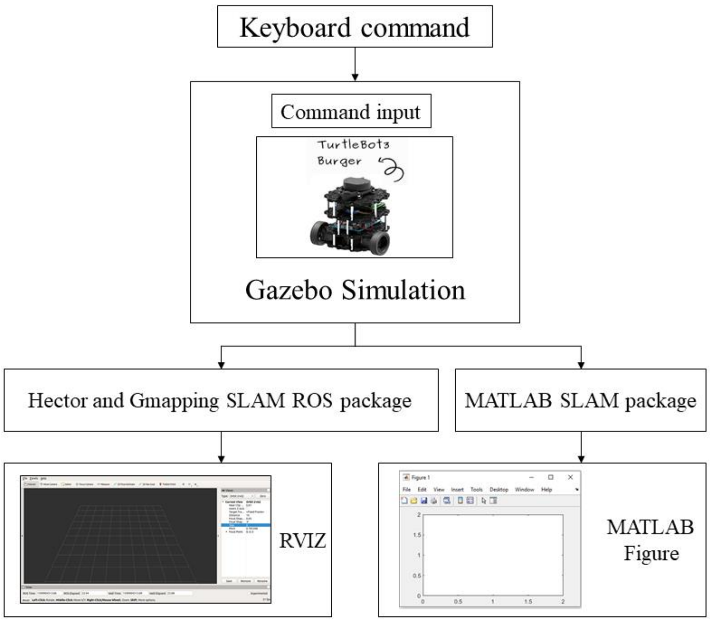


Figure 4.1. Simulation Flow

4.2. Preparation Simulation

Before the simulation start, we have to configure the parameters of each package as well as our WMR parameters to meet our desire and as close as possible to the real-world condition. The file that required to modify are:

For Hector SLAM Package configuration main SLAM file Mapping_default.launch, we set odom_frame to /odom, base_frame to /base_footprint, scan_topic to /scan, map_frame to /map.

For Gmapping SLAM Package configuration main SLAM file gmapping_default.launch, we set odom_frame to /odom, base_frame to /base_footprint, map_frame to /map.

For MATLAB SLAM Package configuration script, we initiate MATLAB and ROS communication with rosinit function and set subscriber to ROS scan topic /scan.

4.3. Data Collection and Analysis

MATLAB SLAM

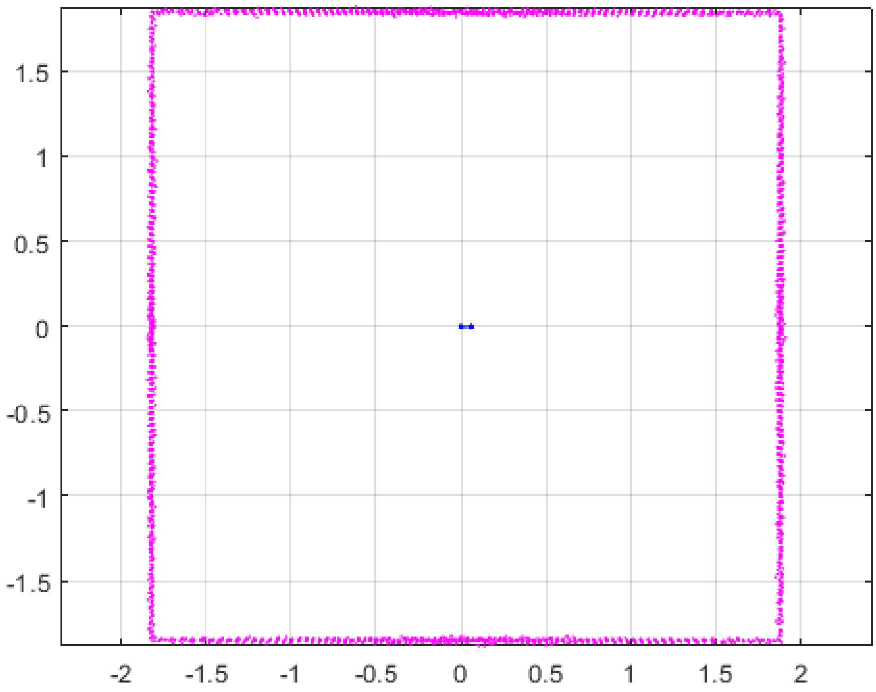


Figure 4.2. MATLAB Pose Graph SLAM gazebo_stage_1

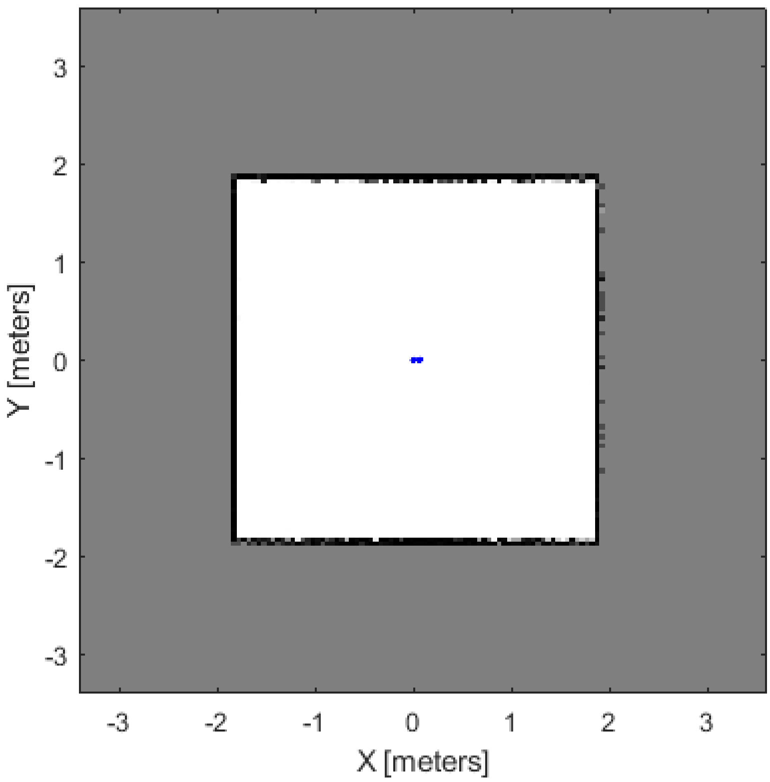


Figure 4.3. MATLAB Occupancy Map gazebo_stage_1

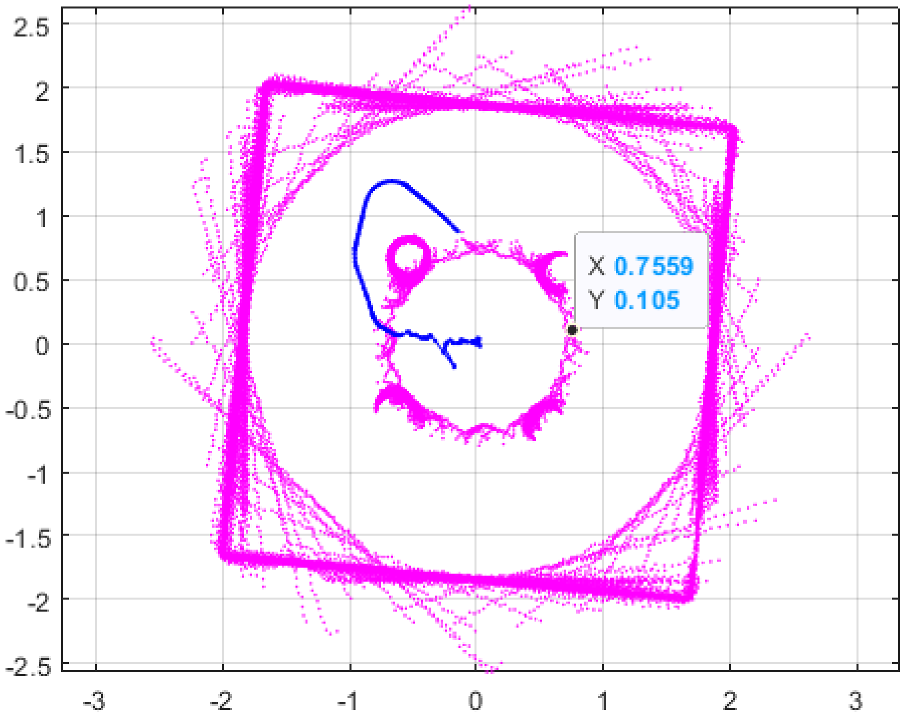


Figure 4.4. MATLAB Pose Graph SLAM gazebo_stage_2

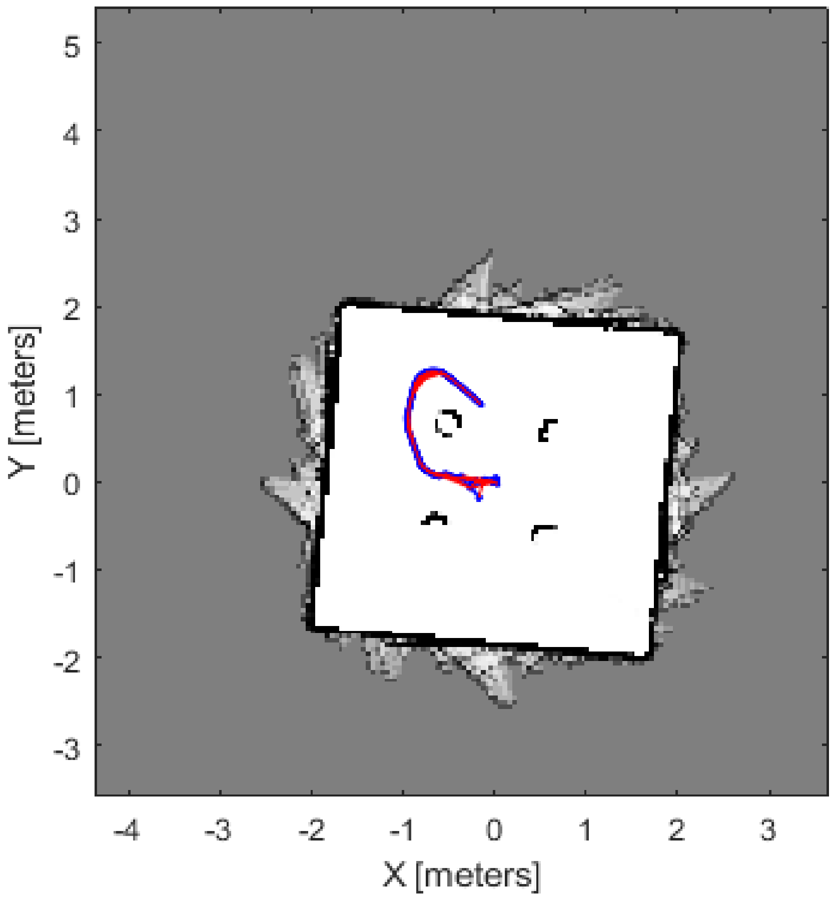


Figure 4.5. MATLAB Occupancy Map gazebo_stage_2

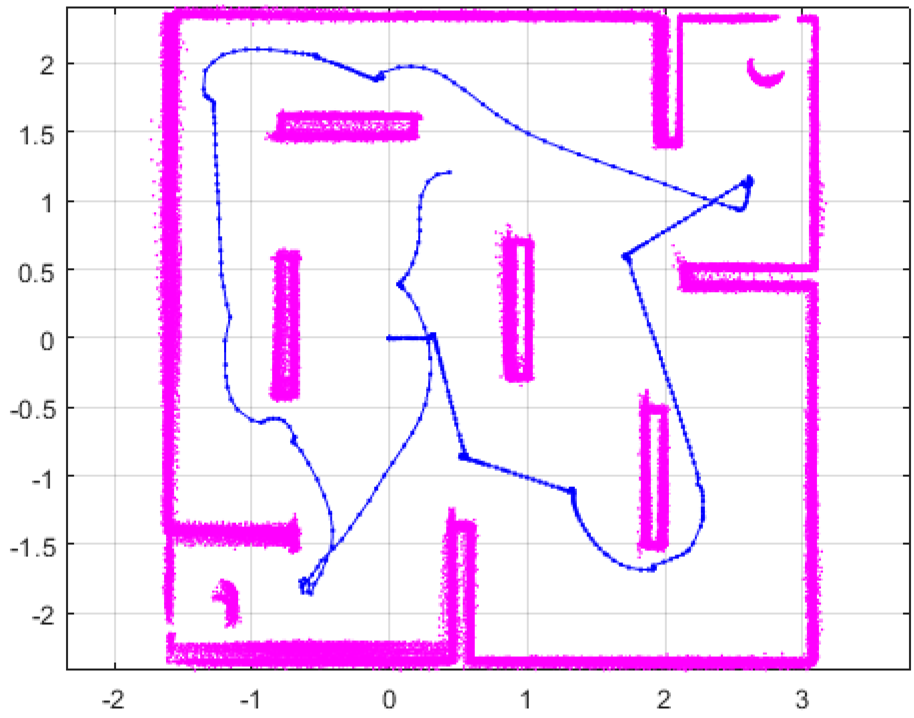


Figure 4.6. MATLAB Pose Graph SLAM gazebo_stage_3

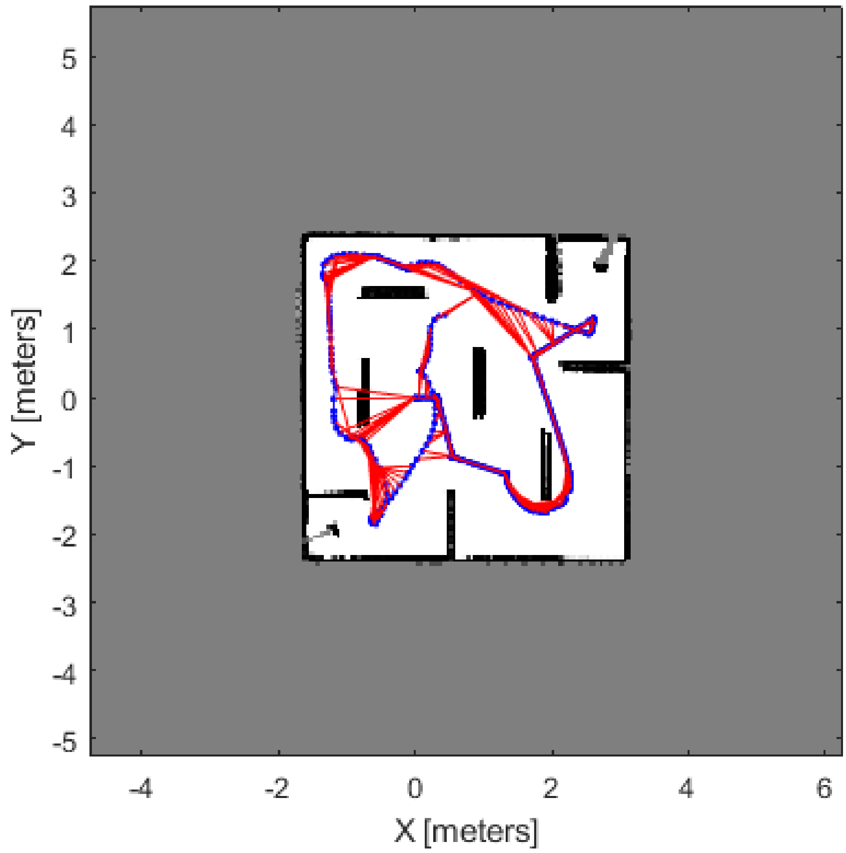


Figure 4.7 MATLAB Occupancy Map gazebo_stage_3

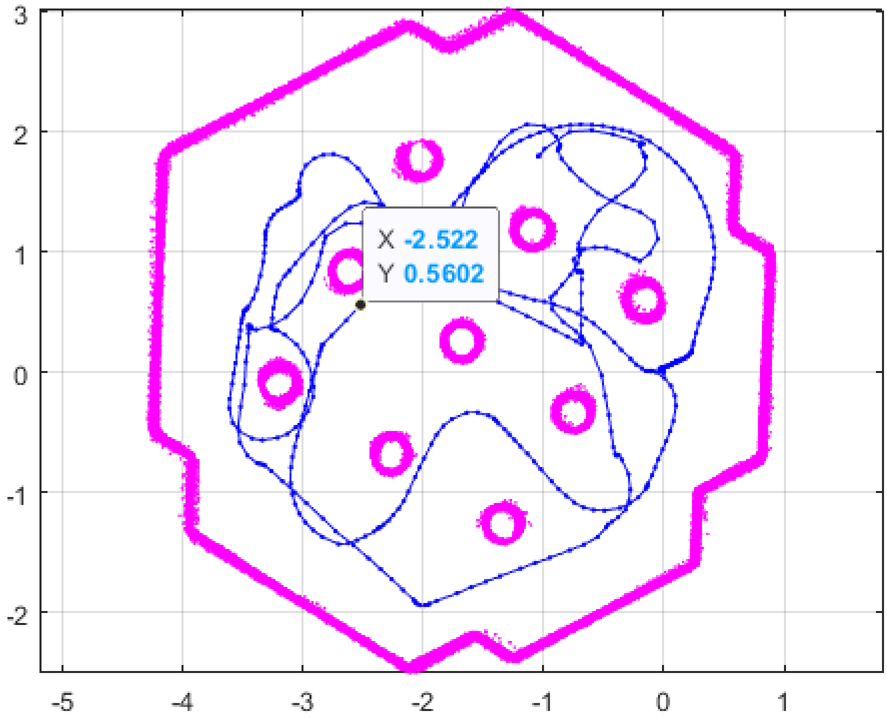


Figure 4.8. MATLAB Pose Graph SLAM gazebo_world

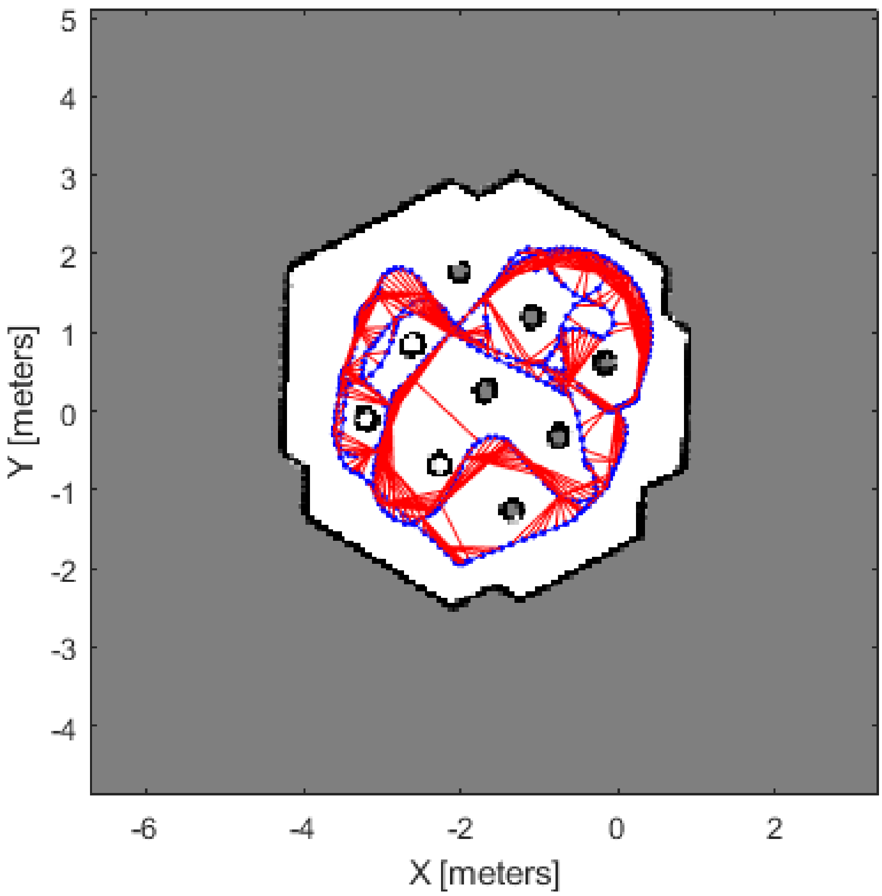


Figure 4.9. MATLAB Occupancy Map gazebo_world

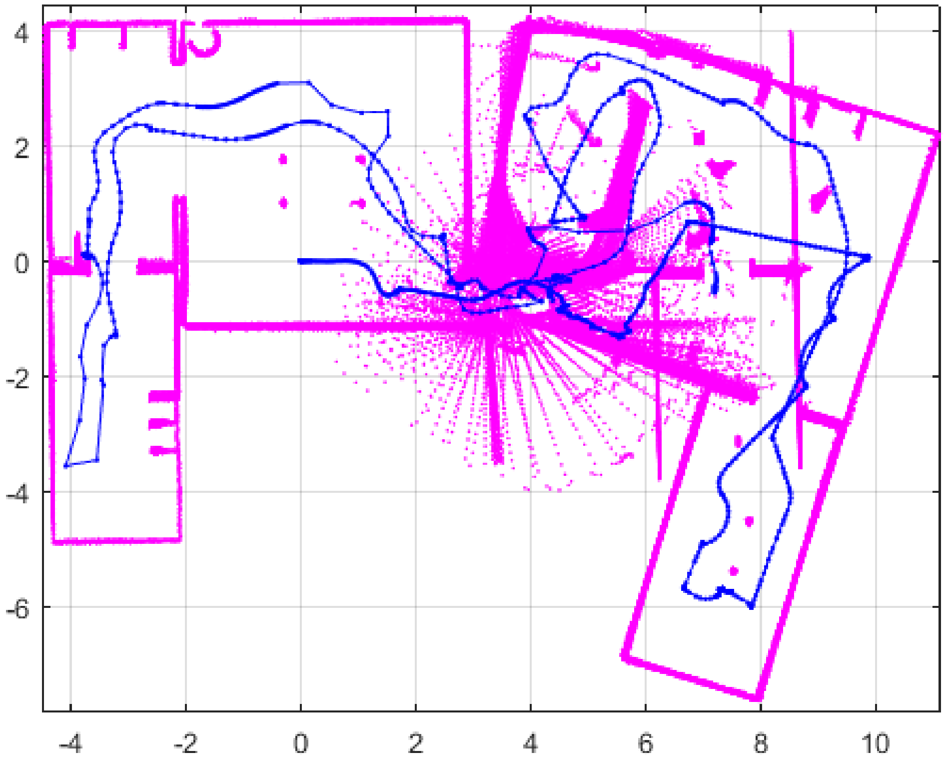


Figure 4.10. MATLAB Pose Graph SLAM gazebo_house

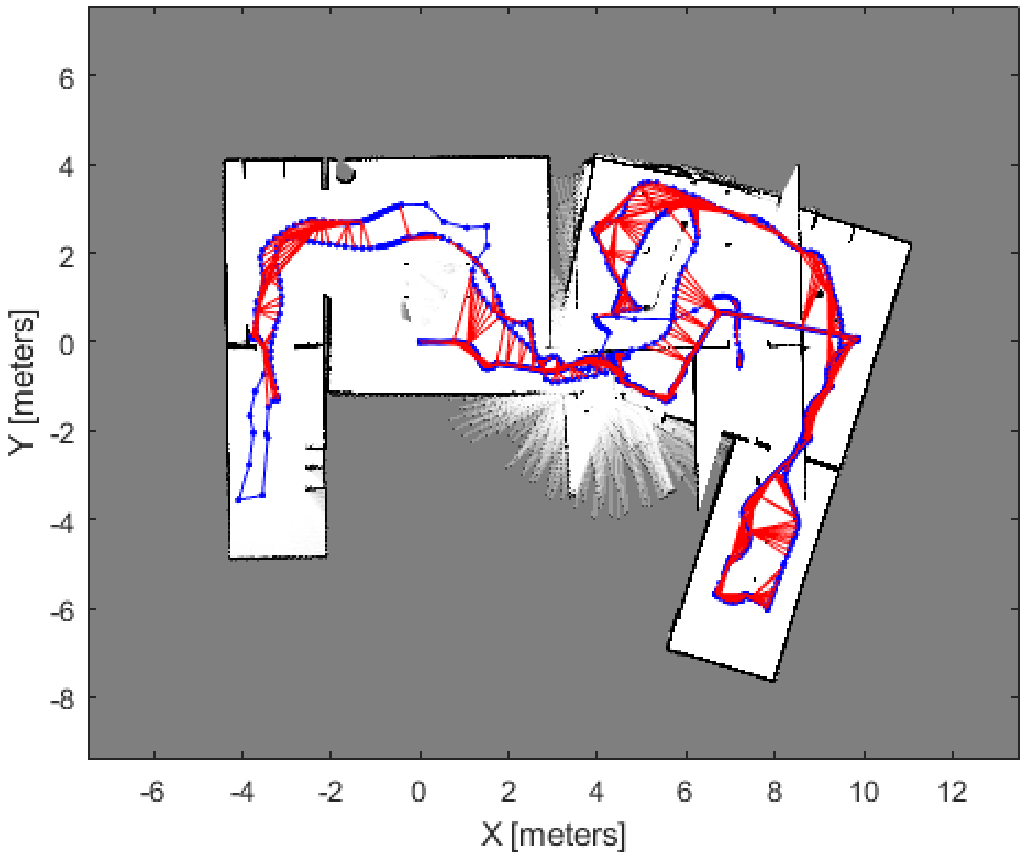


Figure 4.11. MATLAB Occupancy Map gazebo_house

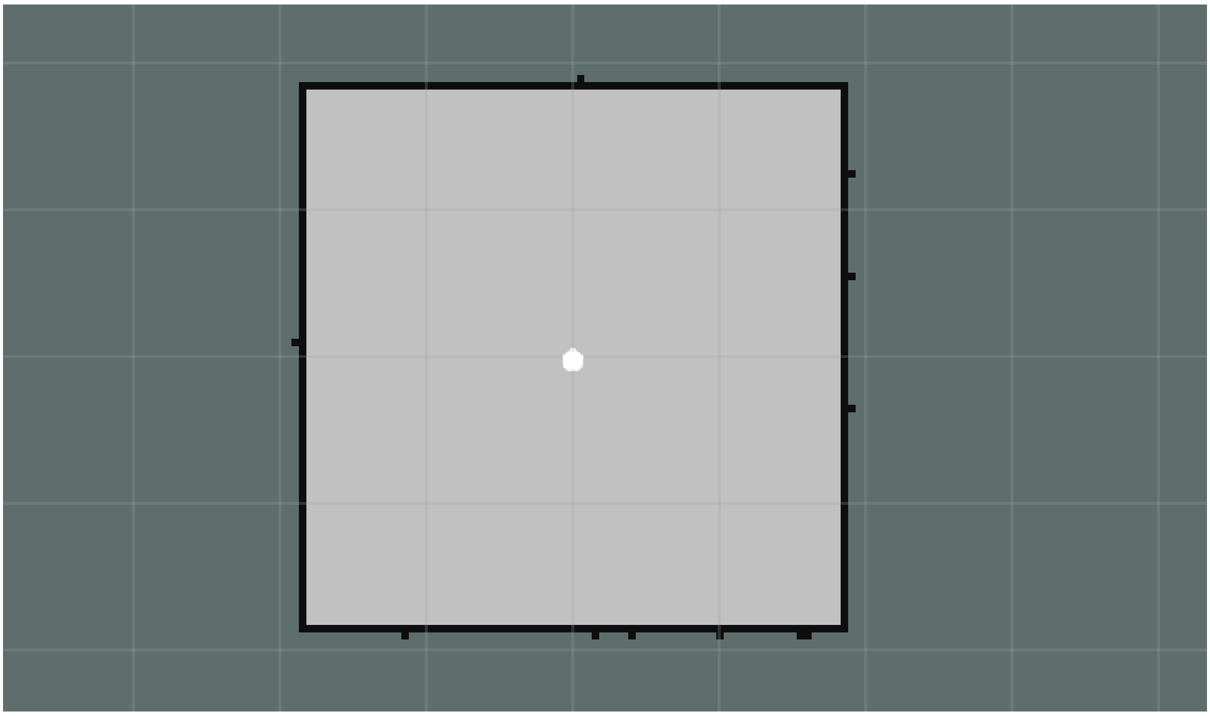


Figure 4.12. Gmapping SLAM Occupancy Map gazebo_stage_1

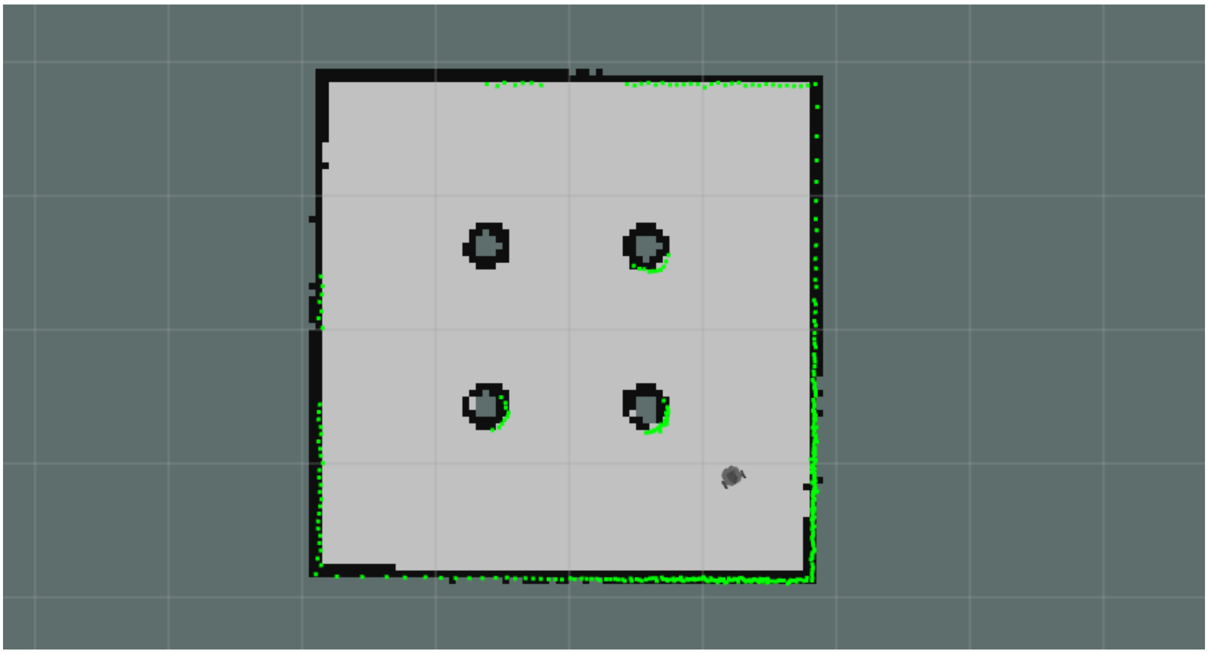


Figure 4.13. Gmapping SLAM Occupancy Map gazebo_stage_2

Figure 4.14. Gmapping SLAM Occupancy Map gazebo_stage_3

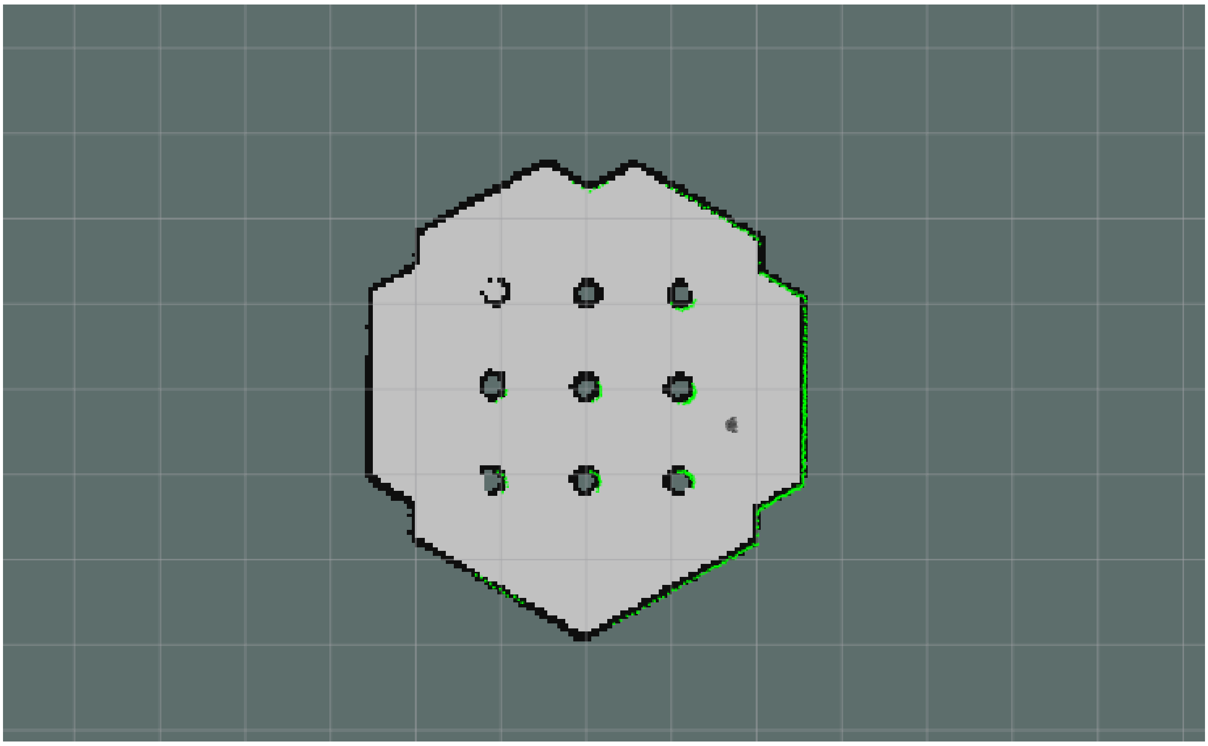


Figure 4.15. Gmapping SLAM Occupancy Map gazebo_world

Figure 4.16. Gmapping SLAM Occupancy Map gazebo_house

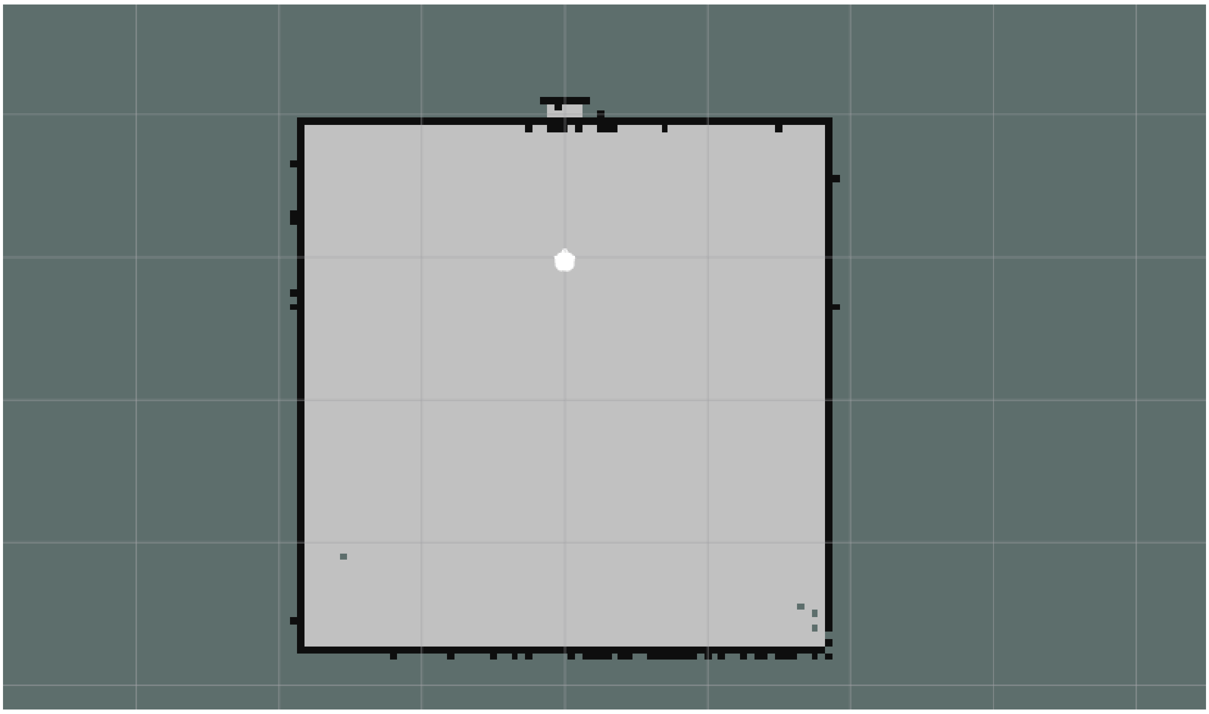


Figure 4.17. Hector SLAM Occupancy Map gazebo_stage_1

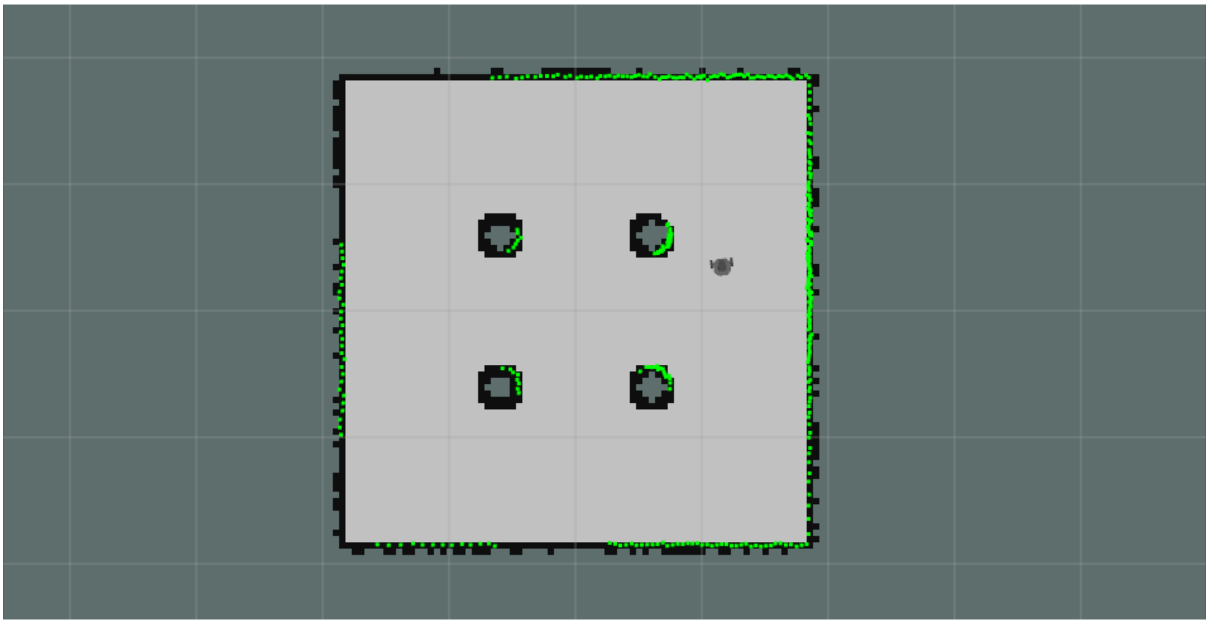


Figure 4.18. Hector SLAM Occupancy Map gazebo_stage_2

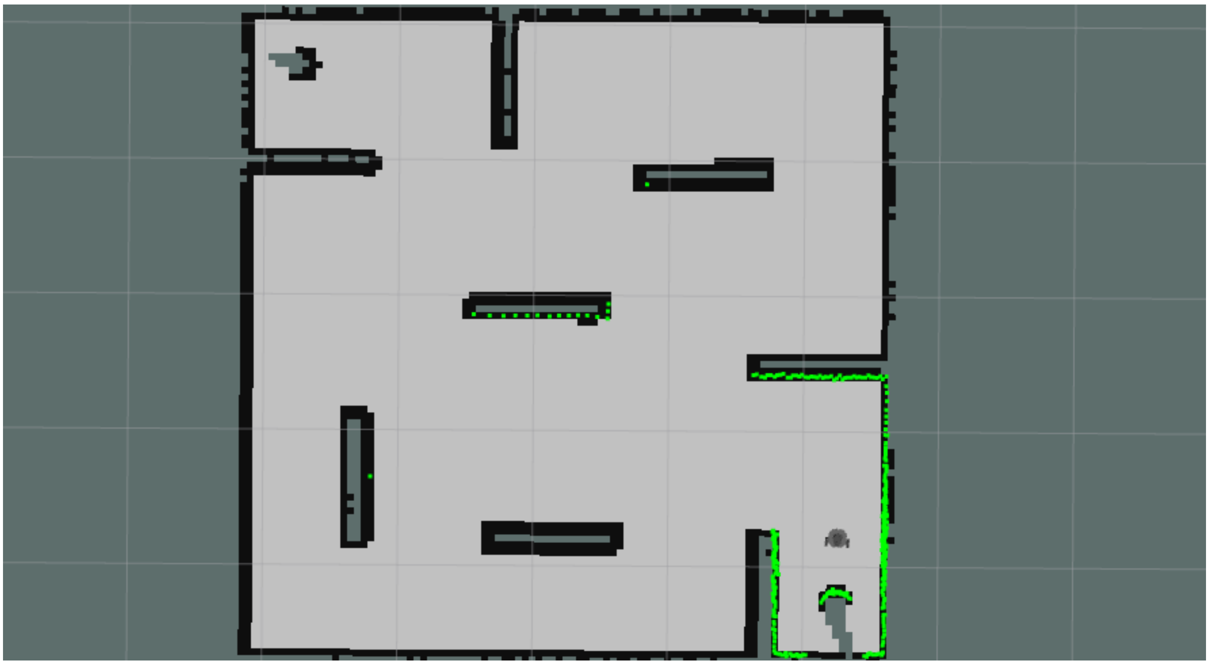


Figure 4.19. Hector SLAM Occupancy Map gazebo_stage_3

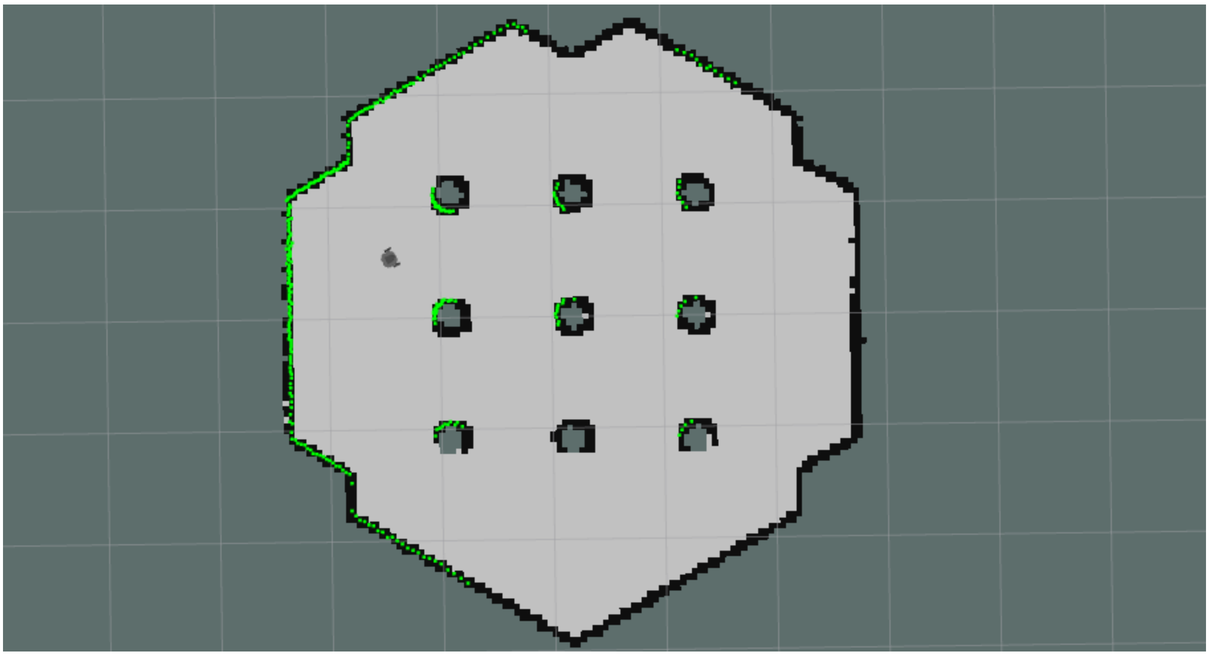


Figure 4.20. Hector SLAM Occupancy Map gazebo_world

Figure 4.21. Hector SLAM Occupancy Map gazebo_house

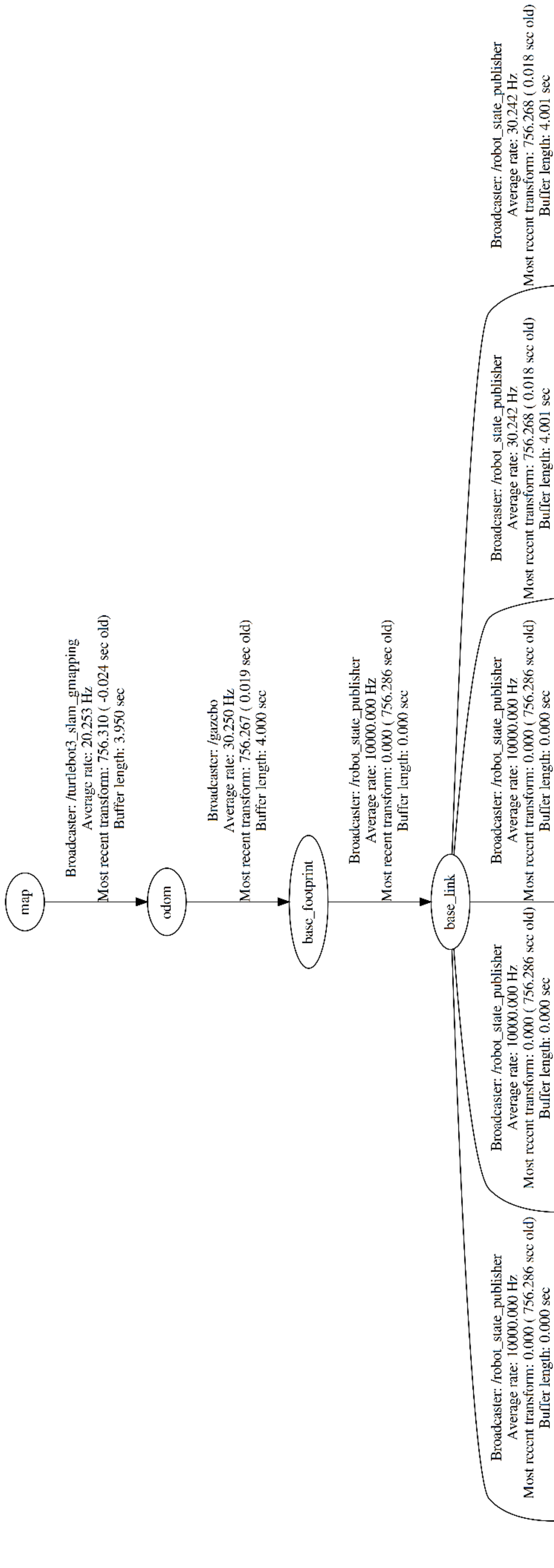


Figure 4.22. Gmapping SLAM tf Tree

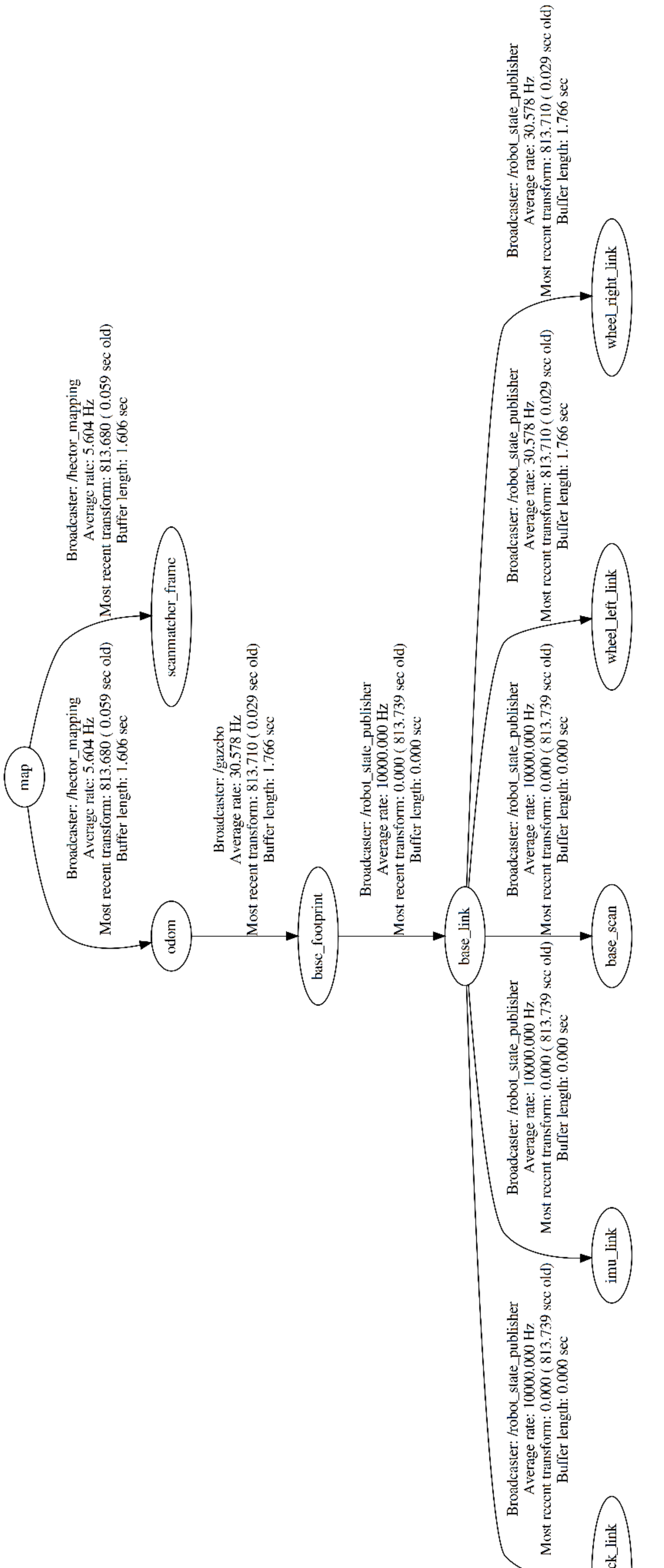


Figure 4.23. Hector SLAM tf Tree

5. CONCLUSION AND RECOMMENDATION

5.1. Conclusion

To conclude in this thesis, SLAM algorithm is presented. By using the simulation software, we are able to simulate the two-wheel differential drive WMR equipped with Lidar sensor and Odometry that publish data to implement SLAM. ROS is used as the main operation system to carry out the SLAM packages such as Hector SLAM, Gmapping SLAM, and MATLAB SLAM to create the Occupancy Grid Map of the simulated environment. While RVIZ and MATLAB are used for displaying the Occupancy Grid Map and sensor data. Comparing the obtained map from SLAM to the simulation environment, we can see the map represents the environment well in small scale map but induces error for a bigger scale one because of the accumulated error from odometry data. As the matter of fact, these maps can be used for robot navigation field.

5.2. Recommendation Future Work

Using this SLAM approach with the data from the simulation give an acceptably good result. For the future work, this approach will be using in real world condition with the real data from the WMR and sensor. As we know, the incoming data from the real sensor will be corrupted by noise that effect on the obtained occupancy grid map. Thus, the sensor data filter can be used to correctly create an acceptable map for real use purpose such as path planning, autonomous driving and dynamic environment navigation with the help from other type of sensors.

REFERENCE

Brian Gerkey. (2019). *gmapping - ROS Wiki*. http://wiki.ros.org/gmapping

Klančar, G., Zdešar, A., Blažič, S., & Škrjanc, I. (2017). Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems. In *Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems*.

Kohlbrecher, S., Stryk, O. Von, Meyer, J., & Klingauf, U. (2011). *[scan match in hector cartographer] A Flexible and Scalable SLAM System with Full 3D Motion Estimation*. 0–5.

Stefan Kohlbrecher. (2012). *hector_mapping - ROS Wiki*. http://wiki.ros.org/hector_mapping

Steven Waslander. (n.d.). *Lesson 2: Populating Occupancy Grids from LIDAR Scan Data (Part 2) - Module 2: Mapping for Planning | Coursera*. Retrieved June 28, 2020, from https://www.coursera.org/lecture/motion-planning-self-driving-cars/lesson-2-populating-occupancy-grids-from-lidar-scan-data-part-2-VcH67

The MathWorks, I. (2019). *Fuzzy Logic Toolbox ™ User ' s Guide R 2019 b*. https://www.mathworks.com/help/pdf_doc/fuzzy/fuzzy_ug.pdf

Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, *45*(3), 52–57. https://doi.org/10.1145/504729.504754

Wim Meeuseen. (2010). *REP 105 -- Coordinate Frames for Mobile Platforms (ROS.org)*. https://www.ros.org/reps/rep-0105.html

APPENDIX A

MATLAB SLAM

MATLAB SLAM script

```
%clear
close all
clc
%%
laser = rossubscriber('/scan'); %subscribe to /scan topic
maxLidarRange = 3.4;
mapResolution = 20;
slamAlg = lidarSLAM(mapResolution, maxLidarRange);
slamAlg.LoopClosureThreshold = 210;
slamAlg.LoopClosureSearchRadius = 8;
for i = 1:1500
    laserdata = receive(laser,1);
    Angles = double([0:0.0175:6.2832]');
    Ranges = double(laserdata.Ranges);
    lidarScanNew=lidarScan(Ranges,Angles);
    [isScanAccepted, loopClosureInfo, optimizationInfo] =
addScan(slamAlg, lidarScanNew);
    if isScanAccepted
        fprintf('Added scan %d \n', i);
    end
end
%%
figure
show(slamAlg);
title({'Map of the Environment Pose Graph'});

%% occupancy grid map built
[scans, optimizedPoses]  = scansAndPoses(slamAlg);
map = buildMap(scans, optimizedPoses, mapResolution,
maxLidarRange);
```

```
figure;
show(map);
hold on
show(slamAlg.PoseGraph, 'IDs', 'off');
hold off
title('Occupancy Grid Map Built Using Lidar SLAM');
```

APPENDIX B

Hector SLAM

Hector SLAM configuration file:

hector_ws/src/hector_slam/hector_mapping/launch/mapping_default.launch

```
<?xml version="1.0"?>

<launch>

  <arg name="tf_map_scanmatch_transform_frame_name"
default="scanmatcher_frame"/>
  <arg name="base_frame" default="base_footprint"/>
  <arg name="odom_frame" default="odom"/>
  <arg name="pub_map_odom_transform" default="true"/>
  <arg name="scan_subscriber_queue_size" default="5"/>
  <arg name="scan_topic" default="scan"/>
  <arg name="map_size" default="2048"/>

  <node pkg="hector_mapping" type="hector_mapping"
name="hector_mapping" output="screen">

    <!-- Frame names -->
    <param name="map_frame" value="map" />
    <param name="base_frame" value="$(arg base_frame)" />
    <param name="odom_frame" value="$(arg odom_frame)" />

    <!-- Tf use -->
    <param name="use_tf_scan_transformation" value="true"/>
    <param name="use_tf_pose_start_estimate" value="false"/>
    <param name="pub_map_odom_transform" value="$(arg
pub_map_odom_transform)"/>

    <!-- Map size / start point -->
    <param name="map_resolution" value="0.050"/>
```

```
    <param name="map_size" value="$(arg map_size)"/>
    <param name="map_start_x" value="0.5"/>
    <param name="map_start_y" value="0.5" />
    <param name="map_multi_res_levels" value="2" />

    <!-- Map update parameters -->
    <param name="update_factor_free" value="0.4"/>
    <param name="update_factor_occupied" value="0.9" />
    <param name="map_update_distance_thresh" value="0.4"/>
    <param name="map_update_angle_thresh" value="0.06" />
    <param name="laser_z_min_value" value = "-1.0" />
    <param name="laser_z_max_value" value = "1.0" />

    <!-- Advertising config -->
    <param name="advertise_map_service" value="true"/>

    <param name="scan_subscriber_queue_size" value="$(arg
scan_subscriber_queue_size)"/>
    <param name="scan_topic" value="$(arg scan_topic)"/>

    <!-- Debug parameters -->
    <!--
      <param name="output_timing" value="false"/>
      <param name="pub_drawings" value="true"/>
      <param name="pub_debug_output" value="true"/>
    -->
    <param name="tf_map_scanmatch_transform_frame_name"
value="$(arg tf_map_scanmatch_transform_frame_name)" />
  </node>

  <node pkg="tf" type="static_transform_publisher"
name="map_nav_broadcaster" args="0 0 0 0 0 0 map odom 100"/>

</launch>
```

Hector SLAM launch file:

hector_ws/src/hector_slam/hector_slam_launch/launch/tutorial.launch

```
<?xml version="1.0"?>

<launch>

  <arg name="geotiff_map_file_path" default="$(find
hector_geotiff)/maps"/>

  <param name="/use_sim_time" value="true"/>

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find
hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>

  <include file="$(find
hector_mapping)/launch/mapping_default.launch"/>

  <include file="$(find
hector_geotiff)/launch/geotiff_mapper.launch">
    <arg name="trajectory_source_frame_name"
value="scanmatcher_frame"/>
    <arg name="map_file_path" value="$(arg
geotiff_map_file_path)"/>
  </include>

</launch>
```

APPENDIX C

Gmapping SLAM

Gmapping SLAM configuration file:

gmapping_ws/gmapping/launch/slam_gmapping_pr2.launch

```
<launch>

  <!-- Arguments -->
  <arg name="set_base_frame" default="base_footprint"/>
  <arg name="set_odom_frame" default="odom"/>
  <arg name="set_map_frame"  default="map"/>

  <!-- Gmapping -->
  <node pkg="gmapping" type="slam_gmapping"
name="turtlebot3_slam_gmapping" output="screen">
    <param name="base_frame" value="$(arg set_base_frame)"/>
    <param name="odom_frame" value="$(arg set_odom_frame)"/>
    <param name="map_frame"  value="$(arg set_map_frame)"/>
    <param name="map_update_interval" value="2.0"/>
    <param name="maxUrange" value="3.0"/>
    <param name="sigma" value="0.05"/>
    <param name="kernelSize" value="1"/>
    <param name="lstep" value="0.05"/>
    <param name="astep" value="0.05"/>
    <param name="iterations" value="5"/>
    <param name="lsigma" value="0.075"/>
    <param name="ogain" value="3.0"/>
    <param name="lskip" value="0"/>
    <param name="minimumScore" value="50"/>
    <param name="srr" value="0.1"/>
    <param name="srt" value="0.2"/>
    <param name="str" value="0.1"/>
    <param name="stt" value="0.2"/>
    <param name="linearUpdate" value="1.0"/>
```

```
    <param name="angularUpdate" value="0.2"/>
    <param name="temporalUpdate" value="0.5"/>
    <param name="resampleThreshold" value="0.5"/>
    <param name="particles" value="100"/>
    <param name="xmin" value="-10.0"/>
    <param name="ymin" value="-10.0"/>
    <param name="xmax" value="10.0"/>
    <param name="ymax" value="10.0"/>
    <param name="delta" value="0.05"/>
    <param name="llsamplerange" value="0.01"/>
    <param name="llsamplestep" value="0.01"/>
    <param name="lasamplerange" value="0.005"/>
    <param name="lasamplestep" value="0.005"/>

  </node>

</launch>
```

Gmapping SLAM launch file: gmapping_ws/src/tutorial.launch

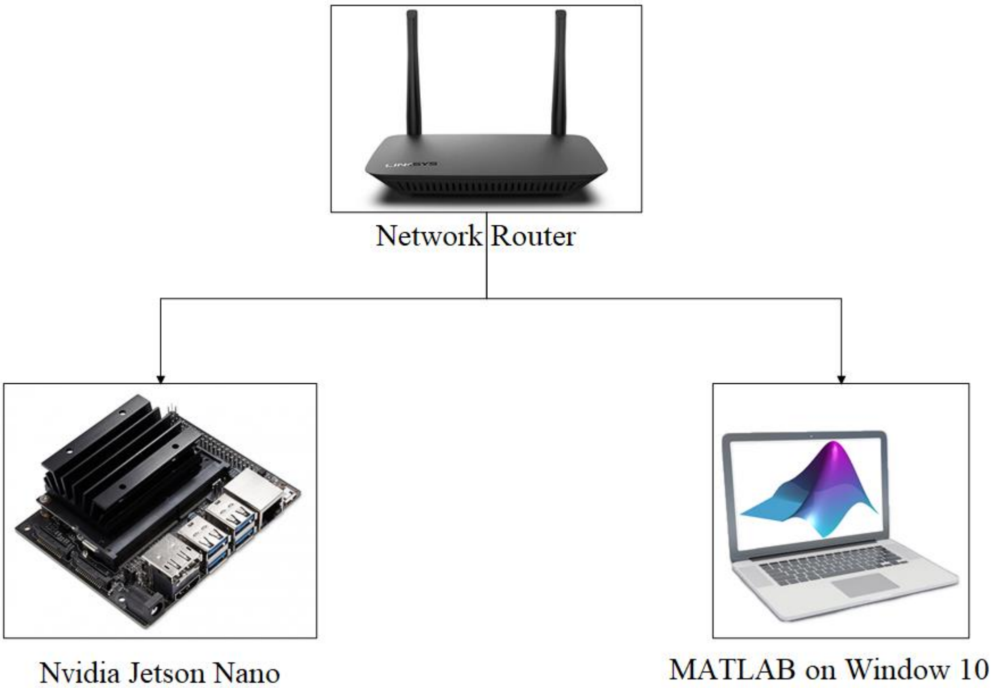
```
<?xml version="1.0"?>

<launch>

  <param name="use_sim_time" value="true" />

  <node pkg="rviz" type="rviz" name="rviz"
    args="-d $(find
gmapping_launch)/rviz_cfg/mapping_test.rviz"/>

<node pkg="gmapping" type="slam_gmapping"
name="gmapping_thing" output="screen" >
```

```
    <param name="scan" value="scan" />
    <param name="odom_frame" value="odom" />
    <param name="base_frame" value="base_footprint" />
    <param name="map_frame" value="map" />

</node>

</launch>
```

APPENDIX D

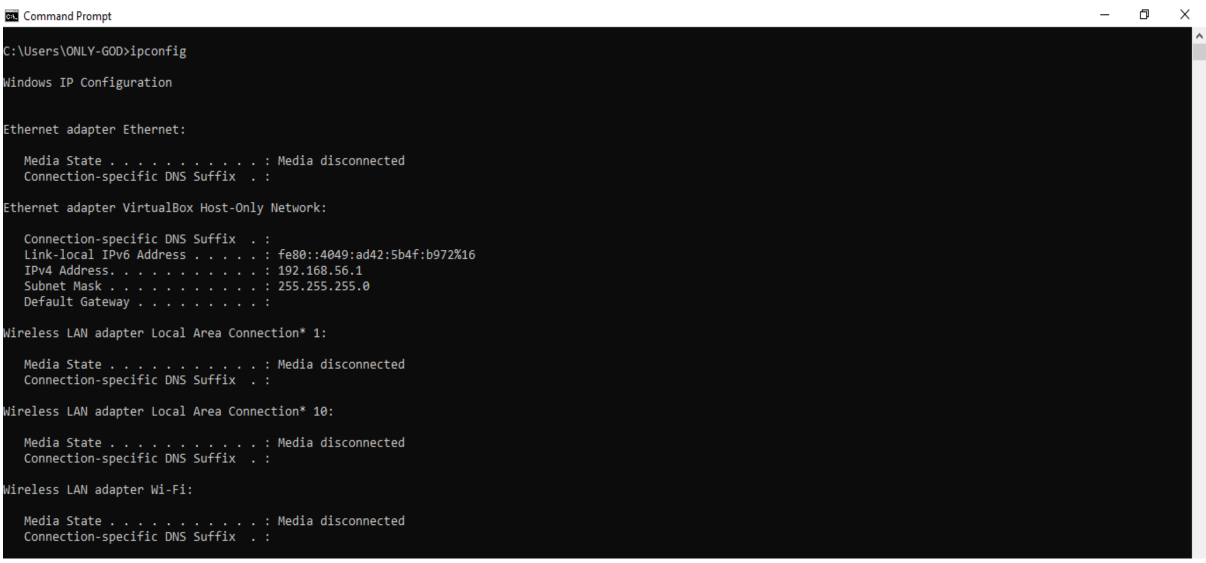
Interfacing ROS with MATLAB

We are using ROS melodic on Nvidia Jetson Nano running Ubuntu 18.04 LTS (Bionic Beaver) with MATLAB 2019b running on Window10 OS. Nvidia Jetson Nano connect to Window 10 OS via Network Router



To establish connection between ROS on Ubuntu and ROS on MATLAB, we have to find IP address of both devices.

- For Window 10, open cmd window and enter 'ipconfig'



- For Ubuntu, open terminal and enter ‘ip addr’

```
yuth@yuth-VirtualBox: ~
yuth@yuth-VirtualBox:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group defaul
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:2f:78:0e brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic enp0s3
       valid_lft 555sec preferred_lft 555sec
    inet6 fe80::9f35:338e:2b86:ce59/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:51:bf:ee brd ff:ff:ff:ff:ff:ff
    inet 10.0.3.15/24 brd 10.0.3.255 scope global dynamic enp0s8
       valid_lft 86355sec preferred_lft 86355sec
    inet6 fe80::1579:a938:a30e:9bd0/64 scope link
       valid_lft forever preferred_lft forever
yuth@yuth-VirtualBox:~$
```

Testing Connection between device via pinging

- For Window 10, in cmd window, enter ‘ping <ubuntu ip address>’

```
Command Prompt
Microsoft Windows [Version 10.0.17763.1282]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ONLY-GOD>ping 192.168.56.101

Pinging 192.168.56.101 with 32 bytes of data:
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64
Reply from 192.168.56.101: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.56.101:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

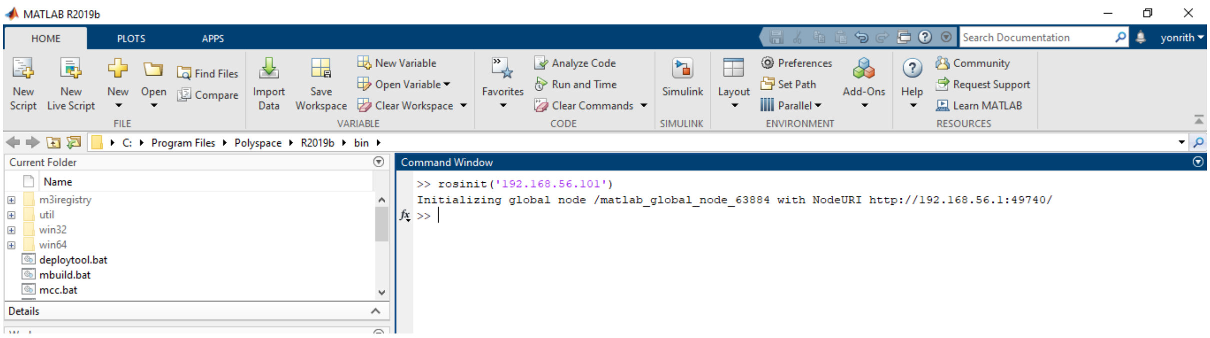
- For Ubuntu, in terminal, enter ‘ping <window ip address>’

```
yuth@yuth-VirtualBox: ~
yuth@yuth-VirtualBox:~$ ping 192.168.56.1
PING 192.168.56.1 (192.168.56.1) 56(84) bytes of data.
^C
--- 192.168.56.1 ping statistics ---
7 packets transmitted, 0 received, 100% packet loss, time 6147ms

yuth@yuth-VirtualBox:~$ ping 192.168.56.101
PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.020 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.060 ms
64 bytes from 192.168.56.101: icmp_seq=4 ttl=64 time=0.043 ms
^C
--- 192.168.56.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3067ms
rtt min/avg/max/mdev = 0.020/0.046/0.061/0.016 ms
```

On Jetson Nano, Initiate command ‘roscore’ in ubuntu terminal to begin ROS Master

```
yuth@yuth-VirtualBox:~$ roscore
... logging to /home/yuth/.ros/log/a8f791d0-b9fc-11ea-9da2-0800272f780e/roslaunc
h-yuth-VirtualBox-1910.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://yuth-VirtualBox:36905/
ros_comm version 1.12.14


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.14

NODES

auto-starting new master
process[master]: started with pid [1921]
ROS_MASTER_URI=http://yuth-VirtualBox:11311/

setting /run_id to a8f791d0-b9fc-11ea-9da2-0800272f780e
process[rosout-1]: started with pid [1934]
started core service [/rosout]
```

Establish connection from MATLAB to ROS machine via command ‘rosinit <ubuntu ip address>’



Testing the ROS connection by calling ROS topic in MATLAB via ‘rostopic list’