

Unmanned Ground Vehicle (UGV)

08/2017

EXECUTIVE SUMMARY:

INTRODUCTION/BACKGROUND:

The Unmanned Ground Vehicle (UGV) is one of the major topics in research where the robot must have several features such as navigation, mapping and obstacle avoidance. These features are still one of the main challenges that the robot encounters. The vehicle plays an important role where it can be used in hazardous and dangerous situations without risking the operator's life.

AIMS AND OBJECTIVES:

The aim of the project is to develop a UGV that can navigate a complex environment and can avoid obstacles. The basic objectives are the following:

- Demonstrate obstacle avoidance.
- Use SLAM algorithm to find the location of the vehicle while navigating the environment.
- Use mapping algorithm to map the terrain.

The advanced objectives are mentioned below:

- Apply power control system to distribute amount voltage evenly to motors.
- Apply positional error control to reduce the positional error of the robot between the position of the robot and the planned path towards the goal.

ACHIEVEMENTS:

The robot could navigate the terrain and showed its ability to avoid obstacles. Moreover, the robot managed to localise itself while navigating and developed a map of the terrain. Lastly, the robot includes a PID control where it managed to control the power distribution to motors and reduced the positional error between the path and goal.

CONCLUSIONS/RECOMMENDATIONS:

The navigation and obstacle avoidance tests were conducted in the simulation and the experiment of control systems was also conducted while the robot is not in touch with the ground. Therefore, further work to be done is to combine all the codes and systems together to be installed in the Nexus robot to observe if the behaviour of the robot is like the simulation.

ABSTRACT:

The dissertation portrays the development of Unmanned Ground Vehicle (UGV) that is capable of navigating through complex environments. The objectives of the project were accomplished where Robot Operating Systems (ROS) packages have been employed into Nexus robot and programming microcontroller which is connected to external devices such as sensors and motor controller to be integrated with ROS. An investigation was conducted to explore existing approaches and hardware devices to attain suitable methods that can be applied in the robot design. The project involves using Simultaneous Localisation And Mapping (SLAM) to map and localise the landscape and the Navigation_Stack was used to provide a target to reach, plans a path while avoiding obstacles. In addition, Nexus robot applied PD controller with the aim of controlling and regulating the amount of voltage going to the motors and reducing the positional error caused by the difference in speed of the motors. Nexus robot was able to partially control and regulate the voltage going to motors and maintained the positional error. However, during the testing phase, the laser finder was damaged which led to using Gazebo to simulate the robot. The robot in the simulation was able to navigate and map terrain while avoiding obstacles. Some challenges were faced regarding control system and movement of Nexus robot; several improvements are suggested.

Table of Contents:

Chapter 1 - Introduction:	1
1.1 Background information:	1
1.2 Motivation:	4
1.3 Aims and objectives:	4
1.3.1 Aims:	4
1.3.2 Objectives:	4
1.4 Project management:	5
1.5 Project report layout:	6
Chapter 2 – Literature Review:	7
2.1 Introduction:	7
2.2 Obstacle avoidance:	8
2.2.1 Infra-Red (IR) sensor:	8
2.2.2 Ultrasonic sensor:	8
2.2.3 Laser Range Finder sensor:	9
2.2.4 Vision sensor:	9
2.2.5 Related Work:	10
2.3 Mapping and localisation:	11
2.3.1 Feature-based SLAM:	12
2.3.2 Pose-based SLAM:	14
2.3.3 Related work:	15
2.4 Localisation:	17
2.4.1 Wheel encoders:	17
2.4.2 IMU:	18
2.4.3 Related Work:	18
2.5 Conclusion:	19
Chapter 3 – Hardware and Software description:	20
3.1 Hardware description:	20
3.1.1 Nexus Robot:	20
3.1.2 Nexus duino microcontroller:	22
3.1.3 Nexus duino I/O expansion shield:	23
3.1.4 Raspberry Pi 3:	24
3.1.5 DC motor:	25
3.1.6 Hokuyo Laser:	26
3.2 Software description:	28
3.2.1 Arduino Integrated Development Environment (IDE):	28

3.2.2	Robot Operating System (ROS):	29
Chapter 4 – Design methodology:		30
4.1	Software methodology:	30
4.1.1	Operating System:	30
4.1.2	System design:	31
4.1.3	SLAM mapping:	34
4.1.4	Navigation:	35
4.2	Hardware methodology:	38
4.2.1	Robot function block:	38
4.2.2	Laser power supply:	39
Chapter 5 – Experimental procedure and results:		41
5.1	Experimental procedure:	41
5.2	Results:	44
Chapter 6 – Discussion:		54
Chapter 7 – Conclusion and Further work:		63
7.1	Conclusion:	63
7.2	Further work:	63
References:		65
Appendices:		69
Arduino:		69
ROS:		78

Chapter 1 - Introduction:

1.1 Background information:

The Unmanned Ground Vehicles (UGV) can be either fully autonomous where the vehicle has sensors that can scan the environment and performs an action based on the situation or send the relevant information back to an operator who will be able to control the robot remotely. When the robot is being controlled remotely then it is a non-autonomous vehicle. Therefore, to decide whether the vehicle needs to be an autonomous system, it is important to know and classify the levels of autonomy operations. Autonomy levels is a concept that differentiates the level of decision making, actions performed and human intervention of autonomous system. Autonomy levels are composed of 5 levels (On-Road Automated Driving (ORAD) committee, 2014; Reese, 2016) and they are mentioned below:

- I. Level 1: the robot is capable to carry out uninterrupted tasks autonomously that the human operator specifies.
- II. Level 2: the robot suggests some options to the operator where he needs to evaluate the situation and decide what the robot should do.
- III. Level 3: the robot evaluates the situation, plans what to do, suggests some options and assesses their effect and consequence of using one of the options. However, the human operator has the final authority, whether the robot acts upon the action suggested or not.
- IV. Level 4: the robot judges the situation, executes an action, informs the operator about the action performed unless the operator disapproves of the action.
- V. Level 5: the robot chooses an action, performs it and informs the operator.

Hence, for a ground vehicle to be fully autonomous it must lie at level 5 and for a vehicle to be non-autonomous it should be in level 1. Whether the vehicle is autonomous or non-autonomous, one of the main advantages of the unmanned ground vehicle is that it can be used in dangerous environments and situations where it can be hazardous to humans. Figure 1 shows various unmanned ground vehicles with diverse sizes and weight.




















Over 30,000 lb	Prototype/Deployed		Prototype		
	 Panther w/SRS >40 tons	 Abrams Panther W/SRS >40 tons	 DEUCE w/SRS 35,500 lb	 D7G w/SRS 55,500 lb	 AOE 67,000 lb
2,501 to 20,000 lb	Fielded		Prototype		
	 ARTS 8000 lb	 DEMO III XUV 3,000 lb	 Smoke HMMWV w/SRS 11,500 lb	 T3 Dozer w/SRS 18,600 lb	
401 to 2,500 lb	Fielded	SDD/ Deployed	SDD		Prototype
	 RONS 600 lb	 Mini-FullRCSS 2500 lb	 MDARS -I 600 lb	 MDARS -E 1500 lb	 GLADIATOR 1600 lb
31 to 400 lb	Prototype/Deployed				
	 TALON 34 -80 lb	 URGOT 65 lb	 MATILDA 40 lb	 BUGS 45-60 lb	 ODIS 45 lb

Figure 1: Military unmanned ground vehicles (reproduced from (National Research Council, 2005)

Nevertheless, it is important that the robot must be safe so it does not cause any harm towards itself, environment or civilians. Another advantage of using unmanned ground vehicles is that it can perform tasks more efficiently and with precision because of an increase in processing capacity of embedded computers, which resulted in having high performance, the ability to process a large amount of data and capable of performing quick decision making. With these advantages, the economy can be improved because of the reduction in operational and capital costs in developing unmanned vehicles due to technological breakthroughs in cheap sensing and vision equipment. Figure 2 shows a number of ground robots being used in Iraq.

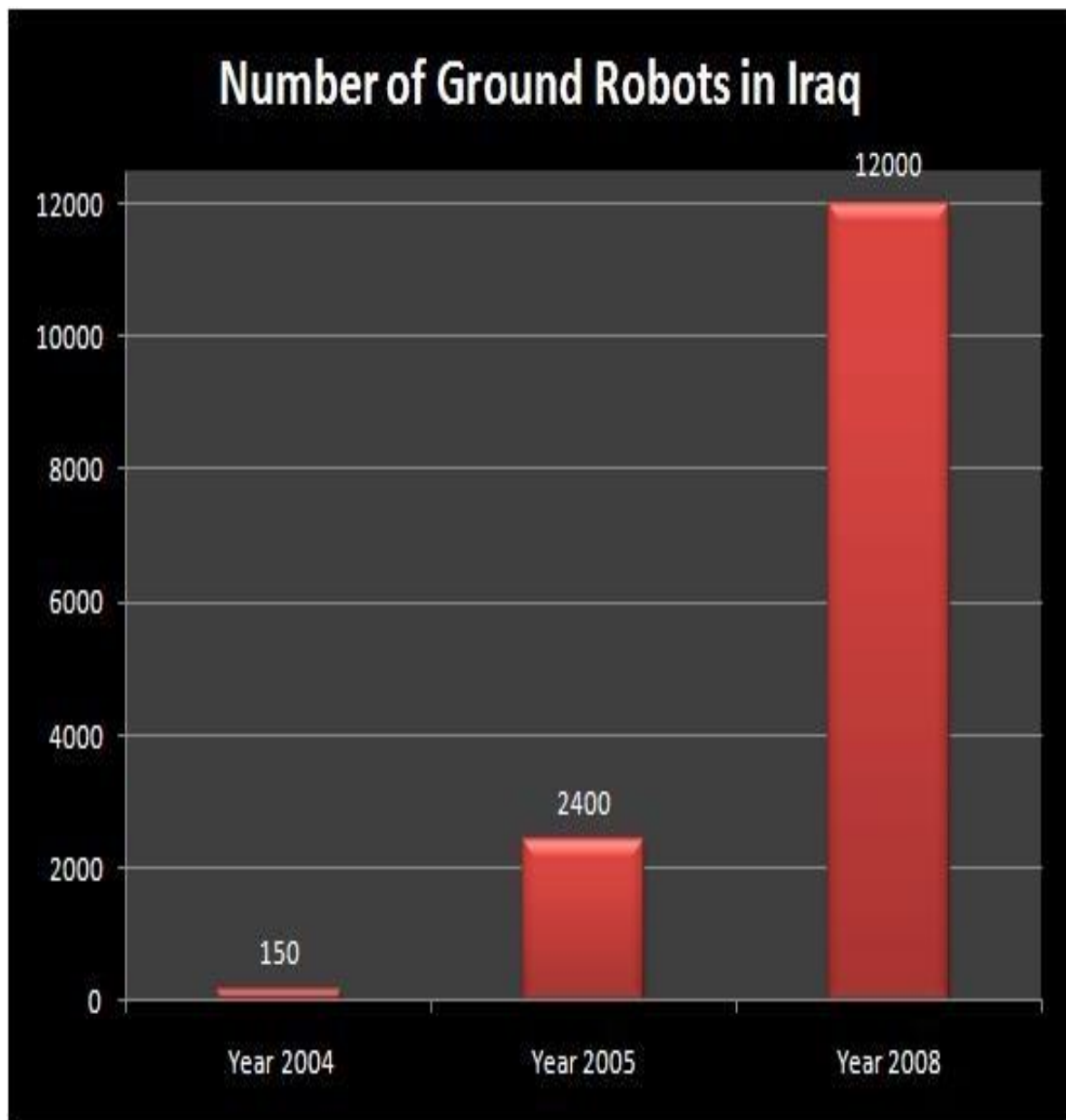


Figure 2: number of military ground robots utilised (reproduced from (Singer, 2009))

Because of the low operational and capital costs, it shows that there will be increase in the development of unmanned vehicles and it allows unmanned vehicles to have a large flexibility where it can be utilised in different applications, for instance, space exploration, civilian and commercial applications like mining or agriculture, emergency response, and military applications.

1.2 Motivation:

There are a number of challenges that the UGV faces, such as navigation, localisation, obstacle avoidance and map creation. These challenges are the main themes in, for instance, military, space exploration and emergency response. Power management of the vehicle is one of the difficulties that vehicle faces due to struggling to maintain some of the power that is being wasted in one or multiple motors. Considering the struggle in power management, this will cause errors in the path taken by the robot where it deviates from its course. Hence, it would be beneficial to add a control system to the robot where it will manage and regulate the amount of voltage being diverted towards the motor and control the deviation in the motion of the vehicle. Additionally, adding control systems would improve the vehicle's power management where it would extend the robot's operation and adjust the motion of the vehicle since it could provide accurate measurement of the vehicle's position and pose on the map.

1.3 Aims and objectives:

1.3.1 Aims:

The aim of the project is to develop a UGV that can navigate through the terrain and map the landscape. Moreover, the UGV has several characteristics: first, the robot constructs a map of the land and to be able to display the map on a Desktop Computer using an algorithm called Simultaneous Localization And Mapping (SLAM). Second, the autonomous robot will be able to avoid obstacles while navigating the terrain. Third, unmanned vehicle will contain a power control system that can distribute power evenly to motors to avoid the deviations in the path taken by the vehicle. Last, the vehicle will have a positional correction system that can reduce the difference between the path travelled by the robot and planned path to the goal.

1.3.2 Objectives:

Objectives are divided into two parts:

I. Basic objectives:

- Demonstrate obstacle avoidance.
- Use SLAM algorithm to find the location of the vehicle while navigating the environment.
- Use mapping algorithm to map the terrain.

II. Advanced objectives:

- Apply power control system to distribute amount of voltage evenly to motors
- Apply positional error control to reduce positional error of the robot between the position of the robot and planned path towards the goal

1.4 Project management:

The overall duration of the project is around 10 weeks, the initial plan was devised into the Gantt Chart as shown in Figure 3.

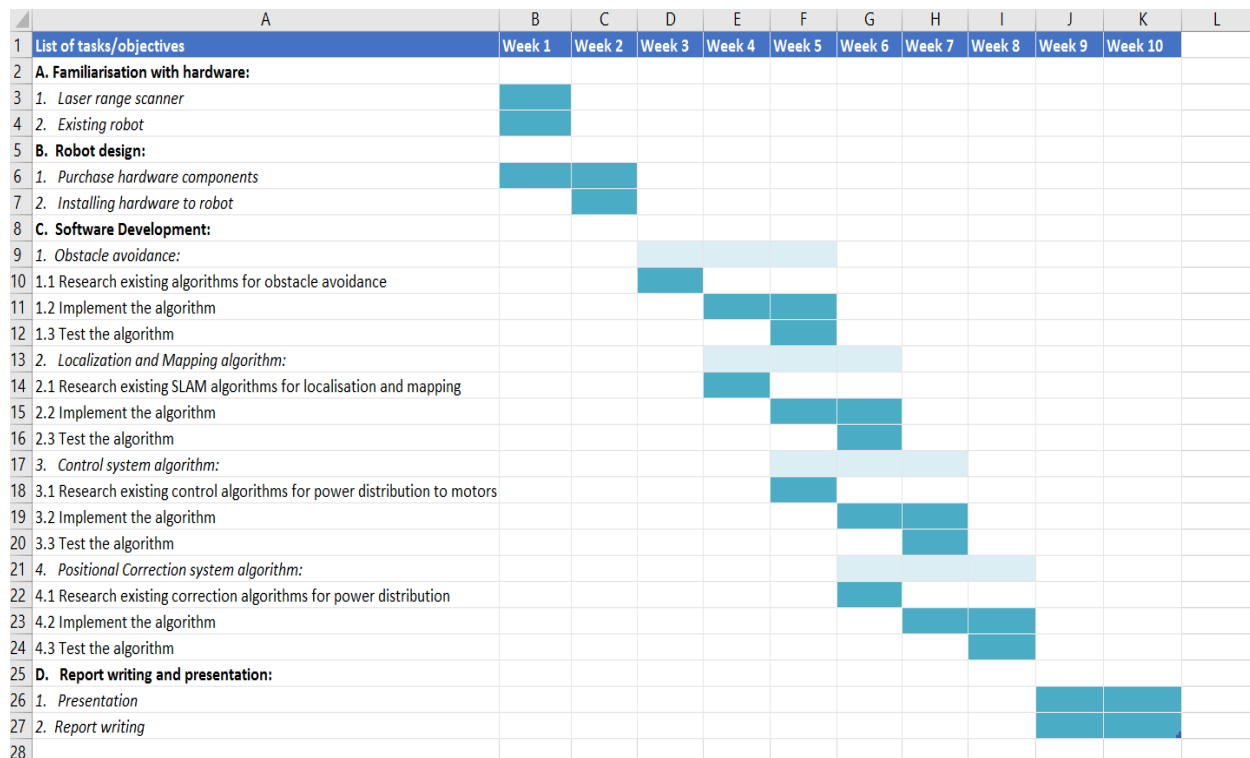


Figure 3: Preliminary Gantt Chart

Each objective takes around 3 weeks where the first week is doing some research into the objectives to find a suitable program/code/article that assists in solving the challenges of UGV. The second week considers the implementation of the process and the last week is to test the algorithm. However, all components were available so no purchase was needed. During the project, the laser got damaged during the testing phase and it would have taken longer to send it back to the manufacturer for the laser to be fixed. Hence, the simulation of the robot was carried out, so, the Gantt chart was modified to adapt to the change in plans. Figure 4 represents the modified Gantt chart.

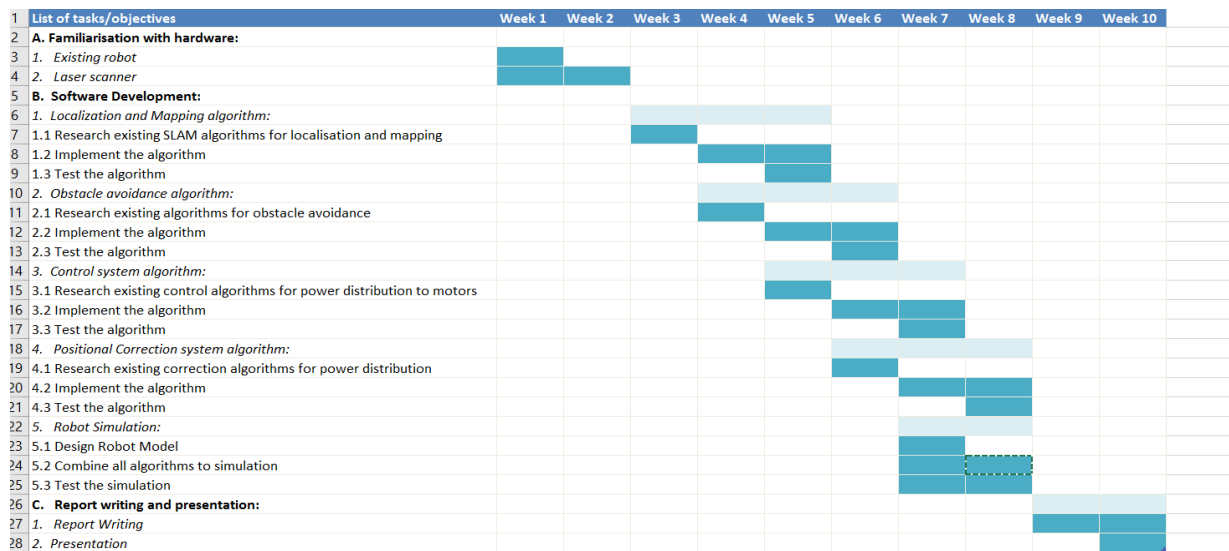


Figure 4: Actual Gantt Chart

1.5 Project report layout:

The dissertation will divide into several chapters:

- I. Chapter 2 investigates the literature review where it discusses the problems that UGV faces and previous researchers work.
- II. Chapter 3 inspects the hardware components and software program used in the project.
- III. Chapter 4 examines the design methodology used for the project.
- IV. Chapter 5 oversees the experimental procedure applied to achieve the mentioned objectives and results obtained from the experiments.
- V. Chapter 6 discusses the results obtained with an explanation.
- VI. Chapter 7 reviews the conclusion of the project and further work to be implemented to overcome the challenges faced during the project.

Chapter 2 – Literature Review:

2.1 Introduction:

There has been an increase in the amount of research into unmanned ground vehicles, most of the attention was directed towards obstacle avoidance, mapping and localisation, and navigation. Various research papers have considered these challenges and have published numerous articles where it looks into solutions to overcome them. This section of the dissertation will review existing methodologies as it might form a baseline for further research into improving the performance of the ground vehicle and the succeeding pages will involve several approaches that can be used to tackle these challenges.

The unmanned vehicle needs to be able to detect and avoid obstacles. To avoid obstacles, it requires sensors that can record the distance between the sensor and the obstacle. However, some of the obstacles could have different shapes or sizes, therefore, in order to identify an obstacle, it requires a sophisticated configuration system where it considers the appearances of the obstacle and how visible the obstacle is to the sensor (Discant *et al.*, 2007). Hence, section 2.2 will look into the sensors used to detect obstacles such as Laser range finder, Ultrasonic sensors, Infra-red sensors and vision sensor.

What's more is that the ground vehicle must be able to localise itself and map the terrain at the same time. This challenge involves trying to simultaneously find the vehicle position and orientation on the map by collecting several unknown landmarks that surround the vehicle. At the same time, it should map the surrounding environment from the vehicle's position (Osian Haines, 2016). Therefore, section 2.3 will consider using a technique called Simultaneous Localisation and Mapping (SLAM) where it divides into several types like HectorSLAM, Gmapping, KartoSLAM which implements SLAM algorithms, such as, Feature-based SLAM and Pose-based SLAM.

In addition, the unmanned vehicle should have the ability to find its location in a given terrain. Then, section 2.4 involves using sensors, for instance, encoders, Inertial Measurement Unit (IMU) and GPS to assist the robot in finding their location on the map.

2.2 Obstacle avoidance:

The autonomous vehicle which contains a distance sensor can convert incoming input (for example, laser ray, sound wave, infrared ray) into an electrical signal where an embedded computer records the data and execute a specific action. This signal could be used to detect surrounding environment and can be applied to create a map of the environment. There are several types of sensors that can be used on unmanned ground vehicle:

2.2.1 Infra-Red (IR) sensor:

The principle of the Infra-Red sensor is to emit an infrared ray of wavelength between 700nm and 1 μ m and the receiver detects if there was a change in the reflected wavelength as the wavelength is affected due to the absorption property of the object. If the wavelength lies within a certain value, then it means that the ray was reflected from the obstacle and triggers the signal pin. Figure 5 illustrates its process.

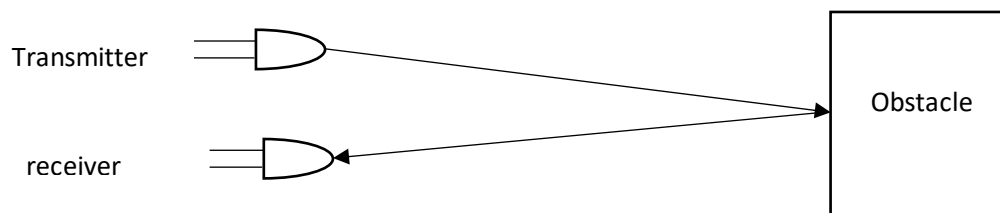


Figure 5: Infra-Red sensor process showing how the signals are being emitted and received by sensor where distance can be determined using difference in wavelength

2.2.2 Ultrasonic sensor:

Ultrasonic sensor implements a method called Time Of Flight (TOF) where it works by emitting an acoustic wave from the transmitter towards an obstacle and the receiver receives the reflected wave. The time duration between the transmission and reception determines the distance between the sensor and the obstacle. Figure 6 shows the TOF principle.

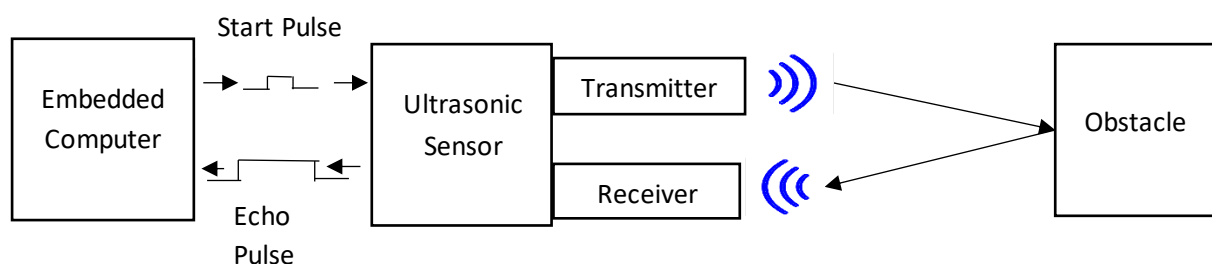


Figure 6: Acoustic sensor process showing how the signals are being emitted and received by sensor where distance can be determined using TOF principle

2.2.3 Laser Range Finder sensor:

Laser Range finder principle is similar to the ultrasonic sensor, but using laser beams. Figure 7 demonstrates its theory.

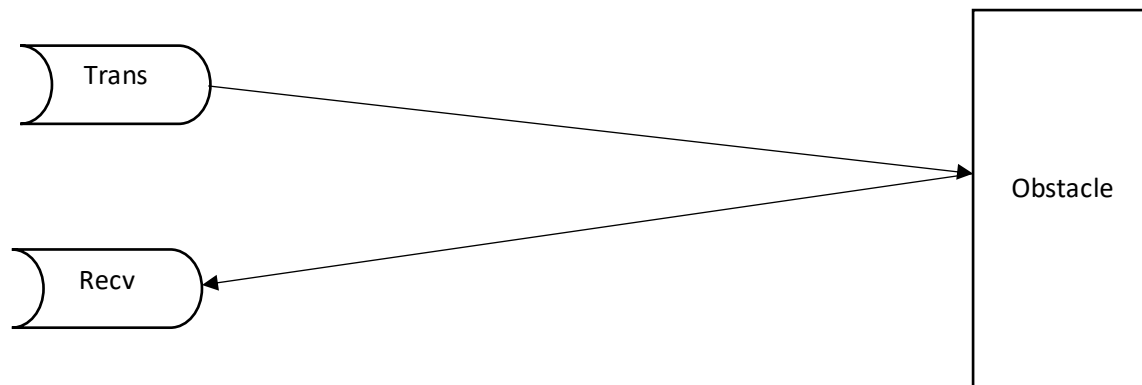


Figure 7: Laser range finder showing how the signals are being emitted and received by sensor where distance can be determined using several principles such as TOF and Phase delay.

2.2.4 Vision sensor:

There are two principal theories behind the vision sensor, Monocular vision approach and Stereo vision approach. Monocular vision manages to detect an obstacle by extracting its feature and obtain its appearance. Stereo vision approach looks into an obstacle from, for example, two positions which can generate 3D depth map (Park, Lee and Son, 2016). Figure 8 demonstrates stereo vision method.

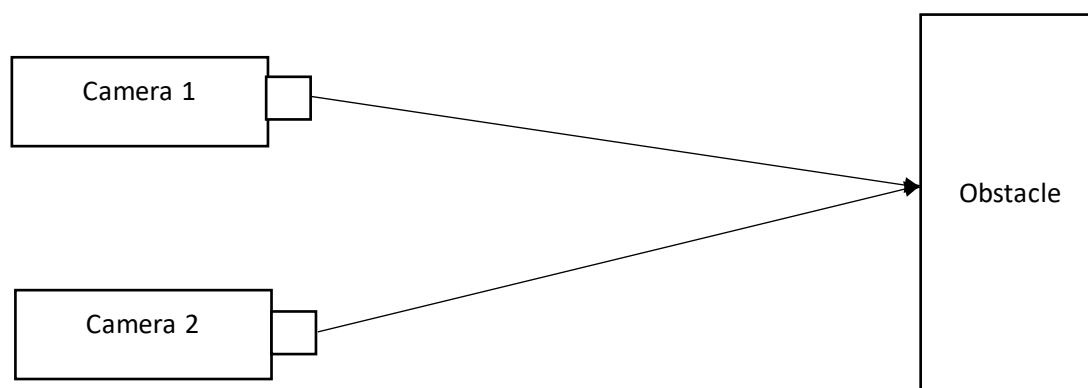


Figure 8: Stereo vision principle using parallel algorithm that extracts features of the obstacle and develops 3D depth map

Therefore, it is important to select an appropriate sensor as its implementation to create a map would affect the information of the environment surrounding the vehicle.

2.2.5 Related Work:

Adarsh *et al.* (2016) have researched into the performance of the ultrasonic sensor and infrared sensor in obstacle detection with obstacles of different materials. These sensor data are affected because of many factors, for instance, infrared sensor data can be altered because of the colour of the obstacle material (Mohamed Kassim *et al.*, 2016). Whereas, the ultrasonic sensor data can be modified since the sensor is sensitive to change in environment like temperature, humidity. The ultrasonic sensor is also sensitive to objects with highly reflective surfaces (Shrivastava, Verma and Singh, 2010). The investigation looked into distance measurement of the sensors from materials like cardboard, wood, rubber, plastic, paper, sponge and tile. The result of the experiment showed that the ultrasonic sensor is suitable with obstacle materials such as wood, tile, plastic, sponge, cardboard while the infrared sensor is suitable for detecting obstacle materials like sponge and paper sheet.

Panich (2010) research compared the distance measurement between the stereo vision and ultrasonic sensor. The study explored the effects of the sensor reading based on its update rate where the rate depends on the current orientation of the robot. The analysis showed that the ultrasonic distance error is about 3.6 cm where the stereo vision distance error is 36.9cm which is around ten times of the ultrasonic as the stereo vision requires double processing time because it analyses two images using a parallel algorithm.

The article that was published by Mohamed Kassim *et al.* (2016) have provided an examination where it compares a couple of sensors like ultrasonic sensor, infrared sensor and laser range finder sensor. The researchers used an equation to calculate which sensor to be used based on its features like range, accuracy, size, cost, weight and energy consumption. The analysis illustrated that the Laser range finder have accuracy and range which gave a weighting score of 1.801, while, the ultrasonic sensor has accuracy, range, size, weight, cost and energy consumption that gave a weighting score of 2.666. Whereas, the infrared sensor has size, weight, cost, and energy consumption where it gave 1.466 for weight score.

Chong *et al.* (2015) have published an article where it considers the sensor technologies used in SLAM. The investigation looked into sensors like laser range finder, acoustic sensors (ultrasonic sensor), stereo vision and researched each factor of the sensors and it demonstrated that ultrasonic sensor is most generally used in robots since its cheap and can work with most of the surface materials with acoustics reflective property. However, the sensor has a low spatial resolution, sensing range, slow speed response and sensitive to any change in environment. While the laser range finder is a popular choice for SLAM because it

can provide a robust result of the environment if it was placed indoor or outdoor environment. Moreover, the range finder is also known for its high speed and accuracy which can produce an accurate distance measurement.

2.3 Mapping and localisation:

It is quite challenging for an unmanned vehicle to navigate a terrain, create a map of the environment while localising itself in the map. This problem was initially investigated around 1986 by P.Cheeseman and R.C Smith in the representation and estimation of spatial uncertainty (Smith and Cheeseman, 1986). This work was improved by adding infinite data limit to SLAM by Hugh F. Durrant-Whyte in the 1990s (Leonard and Durrant-Whyte, 1991). This process involves updating the robot position based on extracting the surrounding environment features and re-observe the new environment as the robot moves to a new position. It can be done using some state estimations which will lead to some filters such as EKF, PF, KF. Figure 9 shows a general SLAM process using EKF.

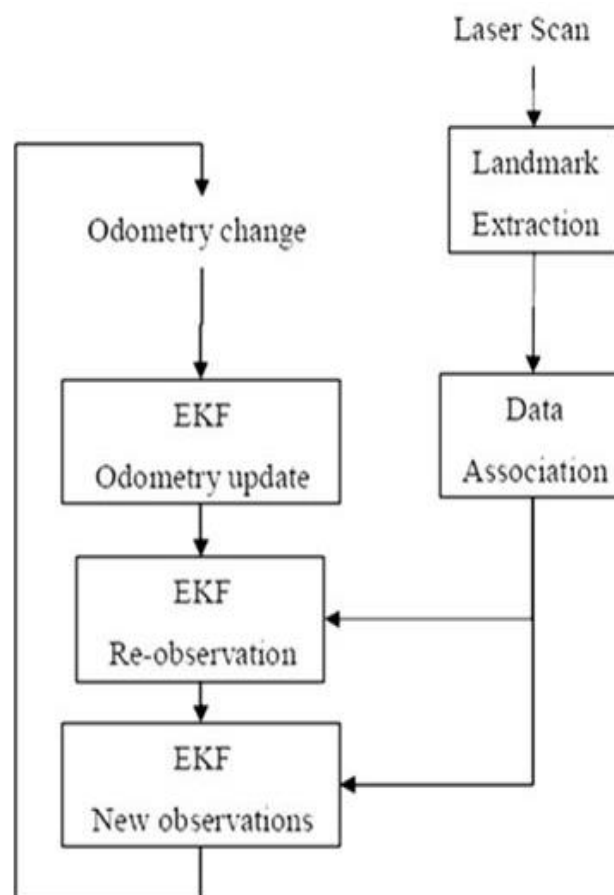


Figure 9: SLAM approach using EKF (Riisgaard and Blas, 2004)

Zamora and Yu (2013) researched into the advancement of SLAM for mobile robots. It reviewed that the solution to the SLAM problem is divided into several main classes, but this part of the section it will concentrate on common and related filter techniques:

2.3.1 Feature-based SLAM:

Feature-based SLAM uses some state estimation techniques where it utilises a model of the surrounding environment of the robot and pre-defined landmarks in order to find the approximate current pose of the robot on the map. Feature-based SLAM class looks into implementing several estimation techniques, such as KF-based SLAM, Graph-based SLAM, PF SLAM (Zamora and Yu, 2013).

Kalman-Filter SLAM uses Kalman-Filter based algorithms which divide into various algorithms and most common parts are Extended Kalman Filter (EKF), Information Filter (IF) or Extended Information Filter (EIF) algorithm and much more. This report will look into only KF and EKF.

Kalman-Filter SLAM is an online solution for SLAM problem where it uses the state space model of the robot and the environment surrounding it. The model uses Gaussian noise assumption as the estimated model contains the orientation and position of the robot as well as the position of the landmark. However, because of the assumption, it creates fake landmarks around the map and filter uses some techniques to eliminate this problem (Zamora and Yu, 2013). Moreover, EKF requires more computing time as covariance matrix needs to be updated because of the increase in the number of landmarks. On the other hand, this problem was investigated and to overcome these challenges, it can be done using the following steps:

- I. Limit the number of estimated landmarks.
- II. Update only the discovered landmarks and in due time fulfil full update of the landmarks.
- III. Consider making an approximate update of the covariance matrix.
- IV. Presume that environment lines are either parallel or perpendicular.

Chen, Samarabandu and Rodrigo (2007) investigated this filter and found out that Se, Lowe and Little (2001, 2002, 2005) have used EKF since it reduced the uncertainties of the currently stored environment features and compares the stored features with new features in case the robot revisited the same environment.

Madhavan, Fregene and Parker (2004) have also used EKF as the robots managed to localize itself and used elevation gradient and vision-based range with the robot's pose to produce a local map. However, through the authors' assessment between Sparse Extended Information Filter(SEIF) and EKF, EKF is slow and requires more memory space, instead, SEIF produces more error than EKF.

Aulinas *et al.* (2008) investigation looked into different various filters such as KF, PF, Information Filter(IF), Expectation Maximization(EM), Compressed Extended FK(CEKF). The analysis showed that using KF/EKF is beneficial because the filter offers high convergence and it can handle uncertainty. But, the disadvantages of using the filter is that it's slow in producing high dimensional maps and the filter uses Gaussian noise.

Yuan *et al.* (2017) have reviewed some of the filtering algorithms to be used in underwater robotic navigation. Their review showed some of the advantages and disadvantages of using KF/EKF and PF. The advantages of using KF/EKF over PF is that it can perform high convergence as the filter can estimate if the geometry of the environment is almost same as the true environment and KF/EKF can handle uncertainty in estimating the robot's pose. However, the disadvantage of KF/EKF is that it uses Gaussian noise assumption and the filter is slow in developing high dimensional maps. Whereas, the PF benefits over EKF/KF is that the filter does not use the Gaussian assumption and it can handle nonlinear system models. On the other hand, the disadvantage of the PF is that the computational complexity of the filter increases as the map environment increases.

Graph-based SLAM implements an optimisation system where it uses nonlinear quadratic programming. The graph-based method looks into the landmark positions in order to acquire the unmanned vehicle path constraints and it estimates the landmark states and vehicle states. Most common Graph-based SLAM is GraphSLAM where the operation extracts nonlinear constraints from the measurements of the distance from the landmark and the sensor, then it linearises the constraints and it tries to reduce it using elimination techniques (Zamora and Yu, 2013). Moreover, its operation uses Least-Square optimization to signify the information matrix and SLAM utilises conjugate gradient descent method. The advantage of using GraphSLAM is that the estimation accuracy is better than other SLAM. Nonetheless, the gradient descent of SLAM is slow to be used in real time (Zamora and Yu, 2013).

Particle-Filter SLAM uses Sequential Monte Carlo (SMC) method where it tries to approximate set of state through probability distribution. The method uses random point

clusters to be spread around the vehicle in order to identify the surrounding environment (Zamora and Yu, 2013). Using this technique, the newly detected landmarks would cause to increase the computation complexity, hence, this SLAM cannot be used for building a map, whereas, it can only be used for localisation to identify the robot's orientation and pose (Zamora and Yu, 2013).

Aulinas *et al.* (2008) review presented that this filter is common with FastSLAM and what makes this SLAM famous is that it encounters one of SLAM's problems which is the data association as it tries to match multiple scans of the landmarks observed by the robot sensor. This problem arises due to some reasons such as mixing up previously scanned landmark with the newly scan landmark, robot assumes that there was a landmark at a certain point, but in reality, there is no landmark, and difficulty in observing the same landmark again (Riisgaard and Blas, 2004). FastSLAM solution solves this problem by breaking it down into two parts: robot localisation and landmark estimation. Using PF, it assists the robot to analyse the path and the advantage of this filter is that it does not require a lot of memory to be used and computational power than any standard filter.

Zamora and Yu (2013) review showed that using this filter has an advantage over Kalman-Filter where it does not implement the assumption of Gaussian noise, also it does not require to linearize motion of the vehicle and can handle nonlinear systems. However, it has some drawbacks like the filter requires high computation power when it enters a high dimensional space because it tries to re-observe the landmarks and the filter is not suitable for long term use as there will be a drop in the filter's consistency (Bailey, Nieto and Nebot, 2006).

2.3.2 Pose-based SLAM:

Using Pose-based SLAM over feature-based SLAM is beneficial because Pose-based involves in estimating the landmark positions. In spite of this, as the number of robot pose increases, it will increase the computational power (Zamora and Yu, 2013). Pose-based SLAM uses a state estimation method where it divides into three types and for this dissertation, it will concentrate on main two types which are Graph-based SLAM and Particle-based SLAM.

Graph-based SLAM uses a graphical network that implements constraints between the landmark and the robot's pose. This method uses some assumptions, for example, it uses Gaussian noise which is similar to EKF but in reality, the map is accurate as the noise does

not exist (Zamora and Yu, 2013). Also, the method assumes that the world is static which could be beneficial in a situation where there is a dynamic environment.

The graph-based procedure involves using optimisation techniques that help in estimating the environmental states (Zamora and Yu, 2013) and the techniques are:

- I. Gradient descent: it reduces the graph map where it divides into several map components that show the operation in real time.
- II. Conjugate gradient: the gradient looks into the matrix inversion as it helps in reducing the computational power.
- III. Stochastic gradient descent: using Stochastic gradient over first two is beneficial because it looks for the global minimum pose by jumping from one local minimum to another. Also, it applies node reduction technique that can increase the map convergence rate.

Particle-based SLAM applies particle filter technique as to find the position of the robot on the map. This SLAM was used in Grisetti, Stachniss and Burgard (2007) research where the authors utilised scan matching technique that can estimate the position and pose of the robot using the resampling method as it helps in maintaining a reasonable amount of particles.

2.3.3 Related work:

Combining some of the classes and algorithms together will form a package where Santos, Portugal and Rocha (2013) evaluated some packages such as HectorSLAM, Gmapping, CoreSLAM, KartoSLAM, and LagoSLAM.

HectorSLAM uses 2D SLAM which contains scan matching algorithm and inertial sensing system for navigation (Kohlbrecher *et al.*, 2011).

Gmapping utilises Rao-Blackwellized PF which is one of the PF family group (Grisetti, Stachniss and Burgard, 2007)

CoreSLAM operates on tinySLAM that uses a simple PF algorithm (Bruno Steux, 2010).

KartoSLAM employs Graph-based SLAM with Sparse Pose Adjustment (SPA) solver where SPA contains scan matching and loop closure algorithm. The loop closure process involves combining previously scanned area with the newly scanned area (Konolige *et al.*, 2010).

LagoSLAM is Linear Approximation Graph Optimization SLAM (Carlone and Aragues, 2011) where its operation is based on Graph-based SLAM and contain 3 optimizer techniques.

Santos, Portugal and Rocha (2013) have tested each package in simulation and in a real arena using a laser sensor which is Hokuyo URG-04LX-UG01. The simulation maps are MRL arena and lr5map (where lr5map is a large map with few unique landmarks) and the real-world arena is MRL arena. The points below will discuss the results:

- Simulation results:

HectorSLAM, Gmapping, KartoSLAM and LagoSLAM produced a map that is almost identical to the original simulated MRL arena map. Whereas, map created by CoreSLAM is not the same as the original MRL simulated map. There is a slight difference between the maps produced by HectorSLAM and Gmapping and maps made by KartoSLAM and LagoSLAM where HectorSLAM and Gmapping are more accurate. The reason why HectorSLAM and Gmapping maps are accurate is that Gmapping package utilises PF algorithm which is optimized with better resampling rate. Moreover, HectorSLAM uses improved scan matching algorithm. On the other hand, KartoSLAM map is better than LagoSLAM because SPA is much better, faster and it has better convergence (Konolige *et al.*, 2010) than optimizer technique used in LAGOSLAM which is LAGO (Carlone and Aragues, 2011).

In the lr5map arena, Gmapping and KartoSLAM maps are almost similar to lr5map. HectorSLAM and LagoSLAM produced a map that is slightly shifted compared to original lr5map. CoreSLAM performance is still the same as MRL arena, the developed map is distorted to lr5map original map due to map not being converged and does not contain loop closure procedure. HectorSLAM map is different due to scan matching algorithm being heavily dependent on the number of landmarks and SLAM does not use odometry information. Hence, scan matching algorithm will find it difficult to match the previously scanned environment with the newly scanned environment as it has few unique landmarks and the robot position is not updated. As the position not being updated, then the current scan would be the same as the previous scan, hence, the map would be alike.

- Real results:

HectorSLAM, Gmapping and KartoSLAM shaped a map that is almost equivalent to the original real MRL Arena map.

2.4 Localisation:

For the robot to be able to navigate any terrain, it needs some sensors to identify its location and pose in the map. There are several sensors that can provide positioning of the robot. For this report, it will consider the first two sensors: encoders and IMU.

2.4.1 Wheel encoders:

The wheel encoder consists of a coding disc, optical detectors and a light source. Figure 10 shows an optical encoder inside a wheel.

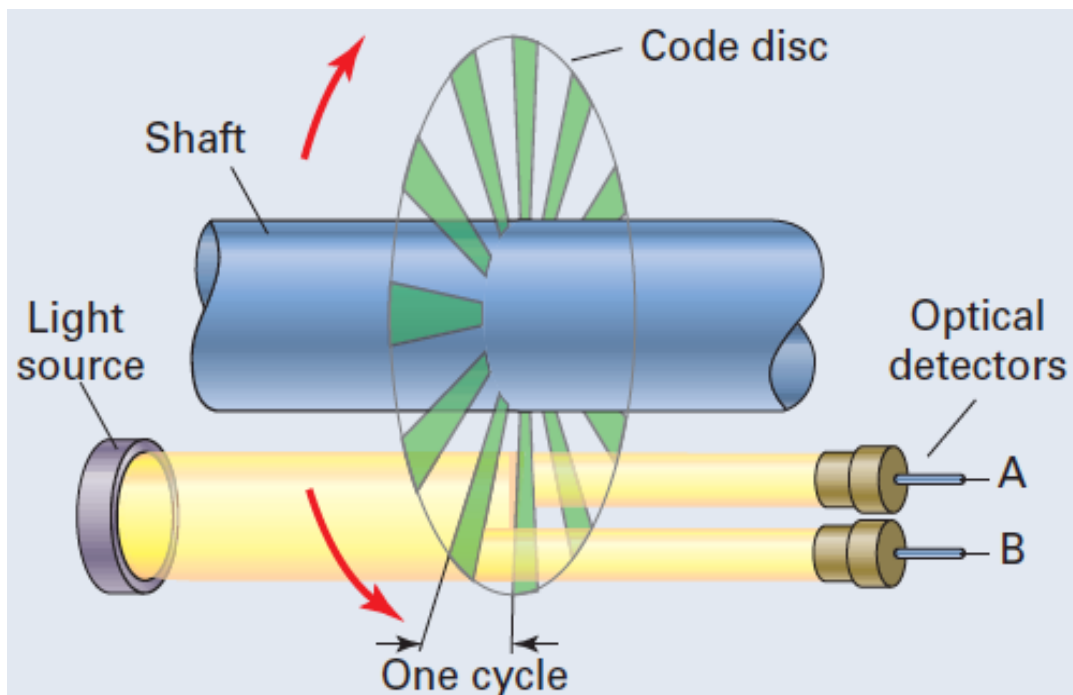


Figure 10: Optical encoder operation where signal pulse is created by light detection from detectors (reproduced from (Berardinis, 2004))

As the shaft of the wheel rotates, the code disc rotates with the shaft which will block the light source from reaching the detectors. This effect will create a series of pulses in which it will generate a signal that will record the movement of the shaft. The advantage of this sensor is that it can indicate the direction of the rotation based on which the optical detector responds to the change of light first.

2.4.2 IMU:

IMU is a single electronic module that is made up of 2 sensors 3-axis Gyroscope and Accelerometers. 3-axis Gyroscope is a sensor that measures the rate of angular change in the orientation of the robot by using deviations in non-Newtonian motion. An Accelerometers sensor measures the external forces which are being exerted on the robot. Some of these forces are Motion, Gravity and Vibration. Figure 11 will represent the integration between Gyroscope and Accelerometers.

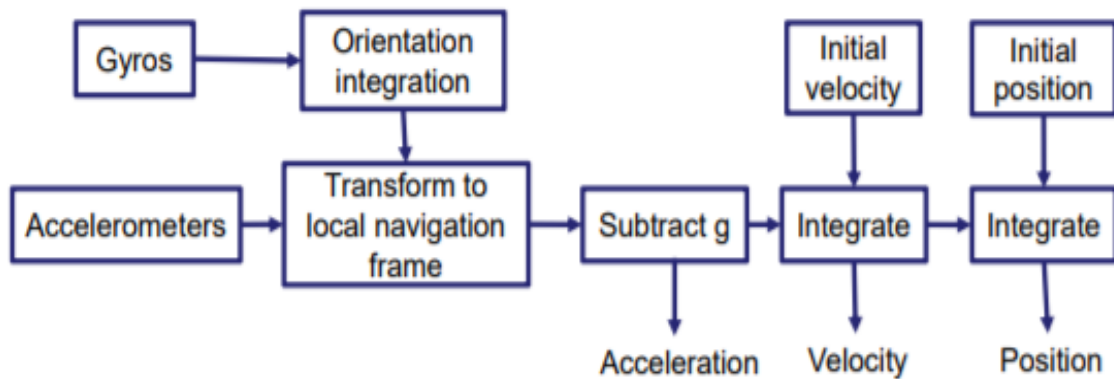


Figure 11: IMU Block Diagram (reproduced from (Dudek and Jenkin, 2008))

Hence, by finding a suitable localisation sensor it will help the robot to know its position and orientation and it will assist the SLAM algorithm as it can print the vehicle pose on the map.

2.4.3 Related Work:

Skog and Handel (2007) performed a survey about the in-car navigation system as it includes a review of the performance of each sensor like IMU and encoders from various articles.

Their review showed that using encoder, it estimates the distance travelled by the wheel and find its velocity to find the heading angle of the robot. The process uses an assumption that it is possible to translate the wheel revolution into linear displacement which is related to the ground (Skog and Handel, 2007). Utilising the encoder with this assumption will cause an inaccurate distance measurement due to some factors like wheel slips, skidding and sampling rate of the encoder (Borenstein and Feng, 1996; Carlson, Gerdes and Powell, 2004). Looking into IMU, the module would benefit the robot in localising itself in the map due to it not requiring an external reference which made it self-contained (Skog and Handel, 2007). On the other hand, the system is vulnerable to noise as a result of relative measurement of the vehicle's motion. These errors are known as drift and they come from both Gyroscope and Accelerometer. Gyroscope error is caused by error estimation on the vehicle attitude as

attitude calculates the gravity of the vehicle along the navigation axis. Error in accelerometer is caused because of the attitude and this error would affect acceleration estimation. To obtain the velocity and position of the vehicle it would require integrating the acceleration, but if the acceleration contains error, then when it undergoes integration the errors will be accumulated and cause errors in velocity and position estimation.

Dudek and Jenkin (2008) also evaluated the characteristics of sensors and among the review, it discusses IMU and encoder. The study also mentions the same error measurements in the encoders, however, despite pose errors, these can be eliminated by integrating the encoder estimates with another sensor system.

Aqel *et al.* (2016) reviewed various types of localisation sensors and technologies, including encoders and IMU. The encoders can provide accurate position of the vehicle over short distances. IMU is susceptible to drift and would require another positioning system for example GPS to correct the position of the robot over a certain period.

2.5 Conclusion:

After conducting the literature review, it is necessary to find suitable sensors and algorithms to be used and will be able to achieve the objectives of the project. The following bullet points will conclude the selection:

- Regarding obstacle avoidance, the sensor that will be suitable for the project is the Laser range finder due to the sensor having the ability to scan the environment at a high speed, can provide accurate distance measurement and has a high angular resolution (Aqel *et al.*, 2016).
- Concerning SLAM, HectorSLAM would be appropriate as maps created in both simulation and in the real world are within satisfactory range. During the lr5map simulation, it had a problem with the odometry; however, this can be resolved by fusing a localisation sensor in order to improve its performance since HectorSLAM uses EKF to merge both scan data with odometry data to provide accurate mapping with a precise location of the robot.
- On the topic of localisation, the encoder would fit well with the objectives of the project due to its accuracy and by utilising the information fusion of HectorSLAM it would eliminate the pose errors.

Chapter 3 – Hardware and Software description:

This chapter will look into the hardware and software components that will assist in achieving the objectives of the project.

3.1 Hardware description:

This section will consist of the robot used for the project and it will diverge into each individual components of the robot.

3.1.1 Nexus Robot:

The Nexus robot is a 4-wheel drive robot that uses mecanum wheels which allows the robot to move in any direction by just simply varying each wheel speed and its direction as shown in Figure 12.



Figure 12: Nexus Robot kit (reproduced from (Nexus Automation Limited, no date))

For the robot to move forward or backwards, it needs all the motors to be activated in the same direction. For the rotational movement, it requires each side motor to run on the opposite side. Moreover, the advantage of using mecanum wheels is that it can allow sideways movement and this can be achieved by running the front and back motors in the opposite direction. In addition, the robot consists of DC motors with built in encoders, a Nexus duino microcontroller with Input/Output (I/O) expansion board and the robot contains suspension structure that eases the motion between each wheel. Figure 13 demonstrates the robot movement and Table 1 represents the specification of the robot.

Table 1: Nexus robot specification (4WD 100mm Mecanum Wheel Robot Kit, no date)

Net Weight:	4.2Kg
Dimension:	400mm x 360mm x 100mm
Microcontroller:	Nexus duino
Battery:	12V NiMH battery
Max speed:	0.6 m/s

<u>Direction of Movement</u>	<u>Wheel Actuation</u>
Forward	All wheels forward same speed
Reverse	All wheels backward same speed
Right Shift	Wheels 1, 4 forward; 2, 3 backward
Left Shift	Wheels 2, 3 forward; 1, 4 backward
CW Turn	Wheels 1, 3 forward; 2, 4 backward
CCW Turn	Wheels 2, 4 forward; 1, 3 backward

To the right: This is a top view looking down on the drive platform. Wheels in Positions 1, 4 should make X- pattern with Wheels 2, 3. If not set up like shown, wheels will not operate correctly.

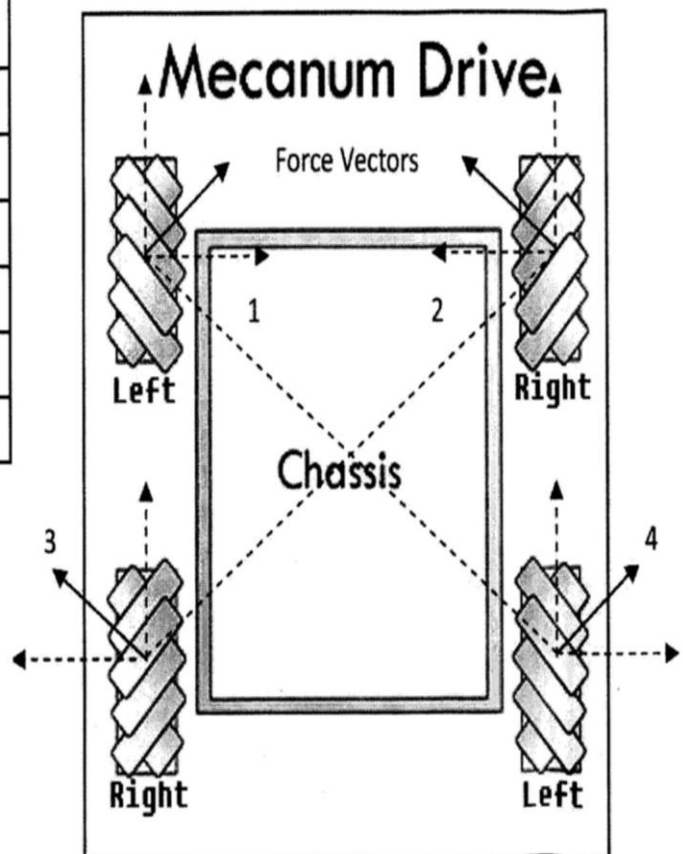


Figure 13: Nexus robot direction of motion (reproduced from (Vectoring robots with Omni or Mecanum wheels))

The following sections will consider the individual components in the robot kit.

3.1.2 Nexus duino microcontroller:

Nexus duino is an Arduino microcontroller that uses “ATmega328” 8-bit microcontroller chip. The microcontroller contains several features such as USB port, 5V voltage regulator, clock speed of 16MHz and reset button. The functionality of the microcontroller is comprised of those main parts. The microcontroller is represented in Figure 14 and Table 2 indicate the board’s specifications.

Table 2: Nexus duino board specifications (Nexus Automation Limited, no date)

Microcontroller Chip:	ATmega328P with motor driver
Operational voltage:	5V
Recommend input voltage:	7-12V
Limited input voltage:	6-20V
Digital I/O pins:	14
PWM I/O pins:	6
Flash memory:	32 KB
SRAM:	2 KB (ATmega328)
EEPROM:	1 KB (ATmega328)
Clock speed:	16 MHz

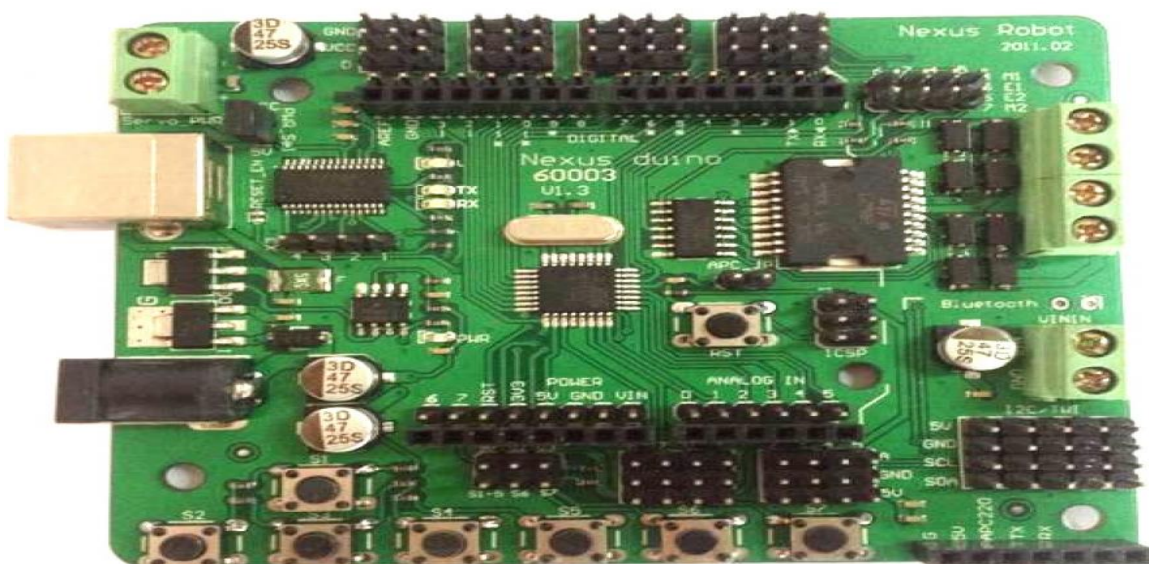


Figure 14: Nexus duino board (reproduced from (Nexus Automation Limited, no date))

3.1.3 Nexus duino I/O expansion shield:

Nexus duino I/O shield is an expansion shield where it allows extra connections to be made to the Nexus duino board. The shield contains some extra features such as 5 external power supply input pins, wireless data-transmission interface, 1 servo external power supply terminal and reset button. Figure 15 illustrates the expansion board and Table 3 represent the characteristics of the board.

Table 3: Nexus duino expansion board specifications(Nexus Automation Limited, no date)

Microcontroller:	Nexus duino I/O Expansion V1.2
Module power supply:	+5V
Servo power supply:	+5V
I/O PWM pins:	6
I/O analog pins:	8



Figure 15: I/O Expansion Shield V1.2 (reproduced from (Nexus Automation Limited, no date))

3.1.4 Raspberry Pi 3:

Raspberry Pi 3 is a single computer board that uses Quad Core 1.2GHz Broadcom 64-bit CPU. Moreover, the board contains the following:

- HDMI port
- Wireless LAN and onboard low energy Bluetooth modules
- MicroSD card port
- CSI camera port
- DSI display port

Figure 16 shows the microcontroller and Table 4 for its specifications.

Table 4: Raspberry Pi 3 specification(RASPBerry PI 3 MODEL B, no date)

Operational current (limit):	2.5A
General Purpose Input/Output (GPIO) Pins:	40
USB ports:	4
RAM:	1GB



Figure 16: Raspberry Pi 3 model (reproduced from (RASPBerry PI 3 MODEL B, no date))

3.1.5 DC motor:

DC motor is a motor that includes the following parts brushes, rotor, magnet, commutator, axle, and stator. As the stator rotates it causes an intersection with the magnetic fields of the magnet and the direction of the produced magnetic field is opposite to the produced current in the rotor. Therefore, the resultant force will cause the rotor to rotate. The motor has a built-in encoder where it can find the position of the shaft based on the digital signal which is generated by the rotation of the shaft. Figure 17 shows the motor used in the robot and Table 5 include the parameters of the coreless motor.

Table 5: Faulhaber DC Coreless Motor(12V 8100 RPM DC Motor, no date)

Operational Voltage:	12V
Gearbox Ration:	64:1
Load current:	1400mA
Power	17W
Encoder type:	AB
Encoder resolution:	12 CPR



Figure 17: 12V DC motor (reproduced from (12V 8100 RPM DC Motor, no date))

3.1.6 Hokuyo Laser:

Hokuyo laser is a laser range finder sensor. The laser works on the same principle to Laser range finder sensor in section 2.2.1. This part will extend the knowledge of the sensor where it uses the principle of phase delay to calculate the distance between the transmitter and the target and the equation used is found below and it is based on Figure 18.

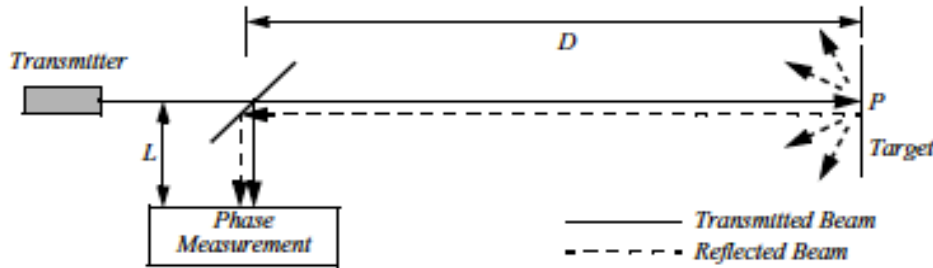


Figure 4.9
Schematic of laser ranging by phase-shift measurement.

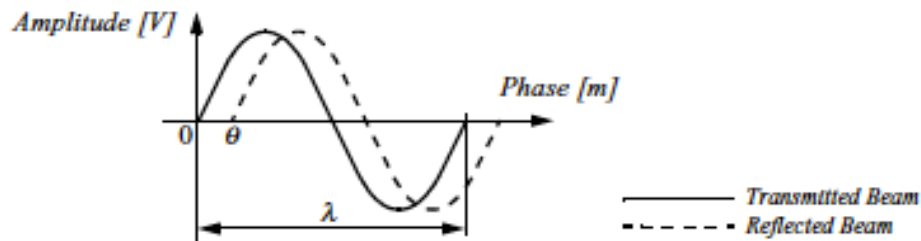


Figure 4.10
Range estimation by measuring the phase shift between transmitted and received signals.

Figure 18: Modulated laser situation (reproduced from (Christina Lee; pbworks, 2014))

$$c = \frac{D}{T}$$

where ‘c’ is the speed of light ($3 \times 10^8 \text{ ms}^{-1}$), ‘D’ is the distance that wave has travelled and ‘T’ is the time taken for the phase shift to occur and this is determined by using the following equation,

$$T = \frac{\theta}{2\pi * F}$$

Where ‘θ’ is the phase shift and ‘F’ is the modulation frequency.

Therefore, combining both equations together, distance travelled can be determined.

$$D = \frac{c * \theta}{2\pi * F}$$

Figure 19 shows the module, Figure 20 shows the laser scanner range and operation, and Table 6 shows the data specifications for the laser.

Table 6: Module Specification(Hokuyo UBG-04LX-F01 (Rapid URG) Scanning Laser Rangefinder, no date)

Module:	UBG-04LX-F01
Range:	20mm to 5600mm
Scanning time:	28 msec
Operating voltage:	12V
Area scanning range:	240°
Angular resolution:	0.36°



Figure 19: Hokuyo Laser range finder (reproduced from (Hokuyo UBG-04LX-F01 (Rapid URG) Scanning Laser Rangefinder, no date))

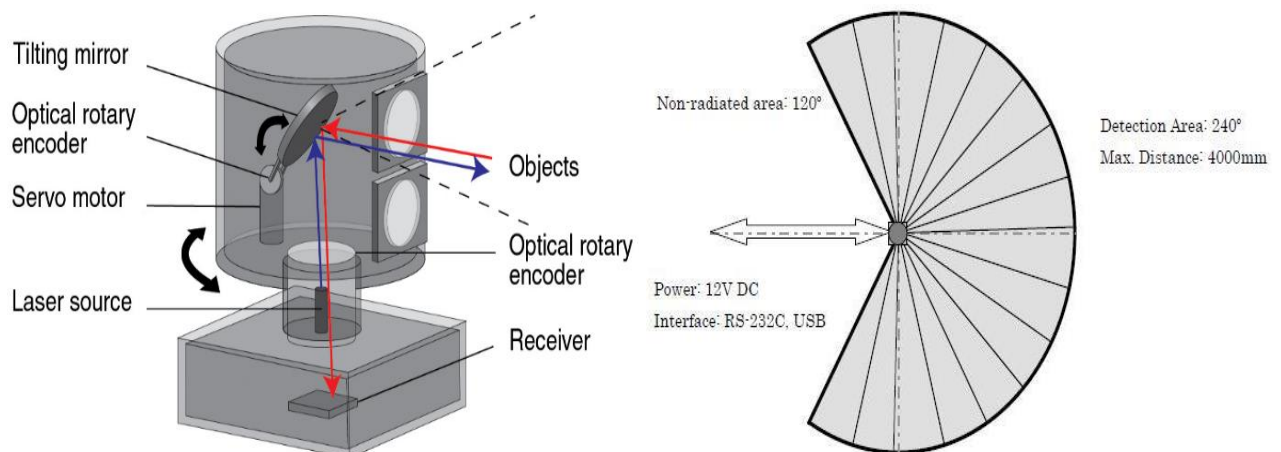


Figure 20: Range detection system and operation of the laser (reproduced from (Localize with a Hokuyo laser range finder, no date; Renishaw, 2016))

3.2 Software description:

This part of the section consists of a couple of main software programs involved in the project.

3.2.1 Arduino Integrated Development Environment (IDE):

Arduino IDE is an open source program where programmers can openly share their code with the community and can obtain or share libraries through Arduino's website. The program uses C programming language where it contains a set of instructions that can be transferred to an Arduino microcontroller through USB cable to perform a specific function. In addition, the IDE has a serial monitor that can be used as a form of testing and debugging the program. The testing involves looking into the input and output pins in order to observe the behaviour of the program. Figure 21 shows an example of Arduino IDE and serial monitor.

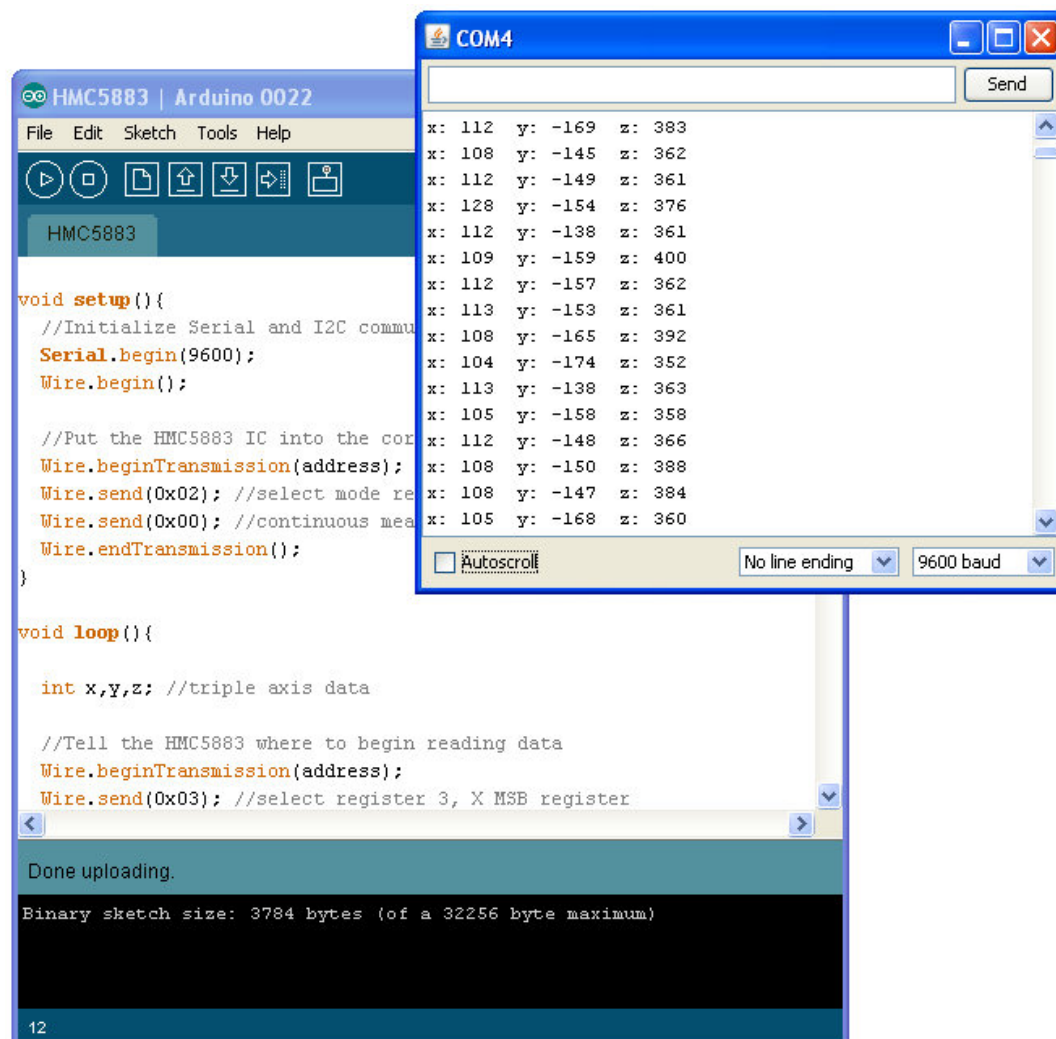


Figure 21: Arduino IDE and Serial monitor (reproduced from (Seidle, 2011))

3.2.2 Robot Operating System (ROS):

Robot Operating System is an open source system that can be used in a robot frame. ROS is used as a platform for peers to share their code in order to help improve the research and development in the field of robotics. Moreover, operating system implements a process called nodes where it consists of lines of code which represent a certain function and can be executed individually. The nodes can be combined together in which it will form packages and stacks. Furthermore, there are number of advantages of using ROS:

- I. Codes used in ROS are designed to be simple and thin where it can be implemented into different robots.
- II. It uses several programming languages such as C++ programming, Python, and Lisp. Also, all programming languages are mostly open source where custom made libraries can be used and shared online.
- III. ROS can be utilised at large-scale systems
- IV. ROS contains some useful nodes that can be used to help in the process of debugging, cleaning the system and much more.

Figure 22 represents an example structure of how ROS looks like.

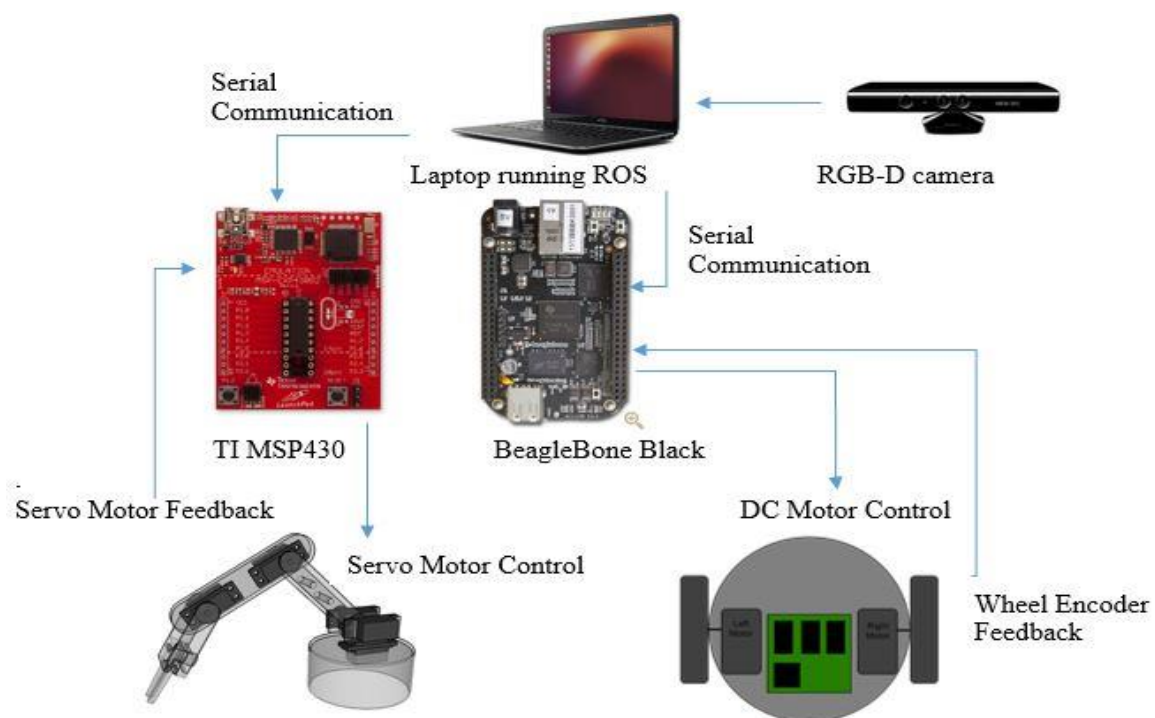


Figure 22: ROS structure between microcontrollers and hardware components(reproduced from(Srikanth, Parhar and Patravali, 2015))

Chapter 4 – Design methodology:

This chapter will discuss the methodologies used during the project. It will involve considering the software methodology that includes ROS and Gazebo. In terms of hardware methodology, this part will contain Nexus robot and the connections made to the robot. Finally, combining all the designs into the robot as to achieve the objectives of navigating the terrain without colliding with obstacles, map the terrain and find its location on the map.

4.1 Software methodology:

This methodology will consider ROS where it contains packages that will help the Nexus robot and robot model to navigate, map the environment while avoiding obstacles and use Gazebo to simulate the robot actions in some environments.

4.1.1 Operating System:

ROS is the core operating system for an unmanned ground vehicle and this system is installed in both Raspberry Pi and Personal Computer (PC). For the operating system to work, it requires an ROS master where it can communicate between the registered devices to send data and receive it from parts called nodes.

For this project, the desktop computer will act as the ROS master and it will be connected to the Raspberry Pi through a Wi-Fi connection. The Raspberry Pi will act as ROS slave where it can receive velocity commands from the master and it is connected to Nexus duino where it receives the command from Raspberry Pi and start to turn on the motors. However, for the project, it will be using the scan data and odometry from a simulation program called Gazebo. Gazebo is part of ROS package that involves in simulating the behaviour of the robot model in a 3D rigid body simulator. Gazebo uses ROS topics, messages and services in which it integrates with ROS with the intention of controlling the functions of the robot in the simulator. Figure 23 will represent the connections between the devices.

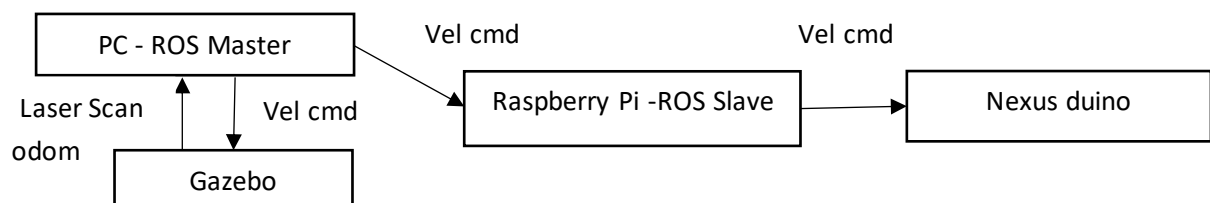


Figure 23: Relationship between hardware and simulation where *vel cmd* is velocity command and *odom* is odometry data

4.1.2 System design:

A node is an ROS client library that performs a certain function and/or controls a hardware. ROS needs a master to link the nodes together and establish a form of communication. The nodes use topics as a form of communication where it can be either publishing (sending) or subscribing (receiving) the message to a topic. Looking at Figure 24, it shows the node's network and topics.

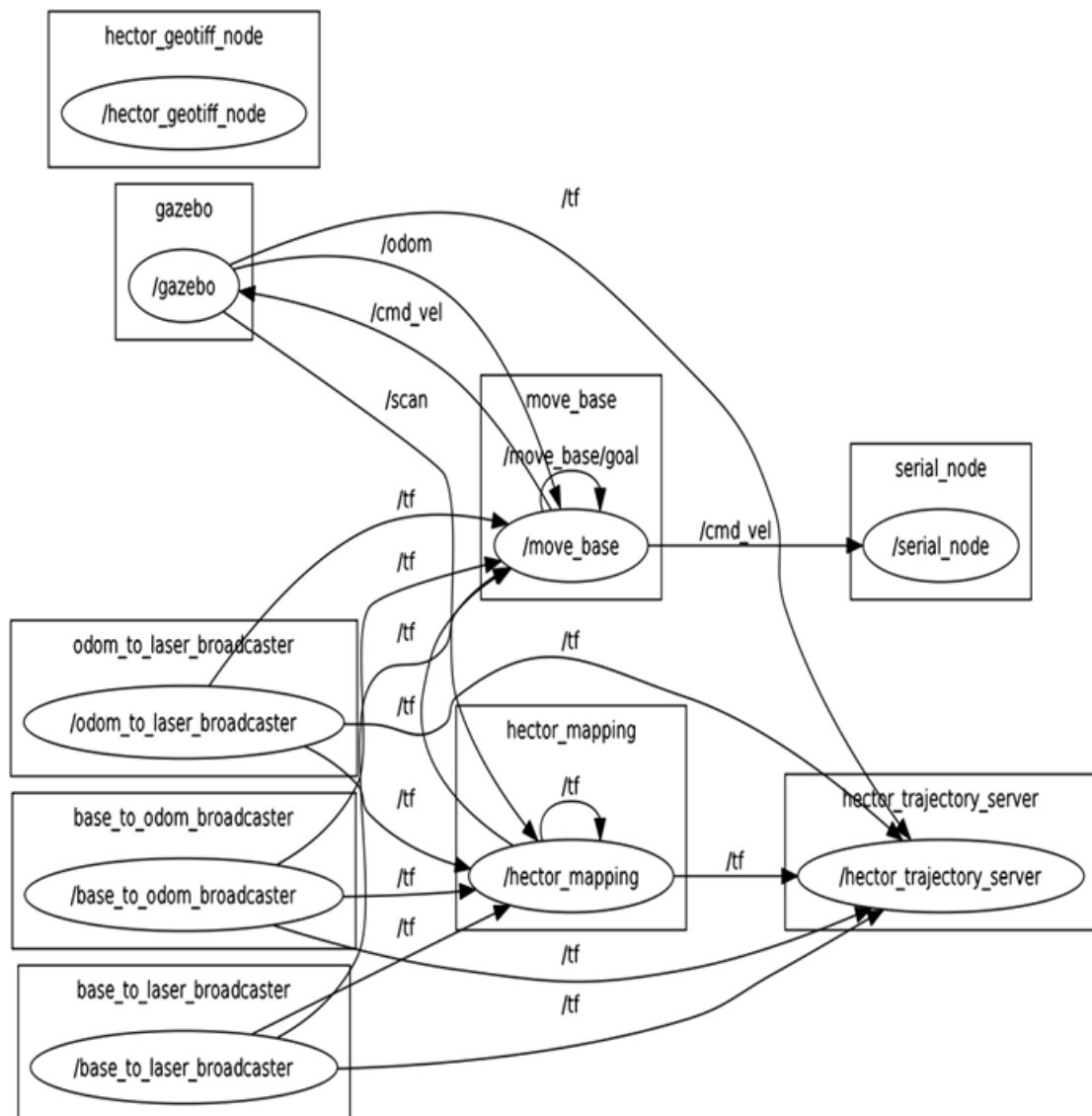


Figure 24: System nodes(Oval) and topics(Arrow)

The following points define each node used in the system:

- *Base_to_laser_broadcaster* utilises a static transform broadcaster where it broadcasts the transformation coordinate frame of the base frame to the laser coordinate frame.

- *Odom_to_laser_broadcaster* employs a static transform broadcaster that broadcasts the transformation coordinate frame of the odometry to the laser coordinate frame.
- *Base_to_odom_broadcaster* applies static transform broadcaster as it broadcasts the transformation coordinate frame of the base frame to the odometry coordinate frame.
- *Gazebo* represents the gazebo simulation where it publishes /odom, /scan, /tf and subscribe /cmd_vel message.
- *Move_base* is a package which performs the required movement based on the incoming /scan data and assigned /goal topic. This node oversees the navigation of the robot model.
- *Hector_mapping* is responsible for mapping the terrain.
- *Hector_trajectory_sever* is used to save and watch the transformation coordinate frame of the system.

The ROS master contains all the information about each topic and message type, for instance, a message could be a geometry twist type where it sends numerical values to another node. Also, when the node communicates with the master, it can gain access to other nodes being involved and it is possible to form a connection between two nodes. Furthermore, it is possible to generate new nodes by simply using call-backs of other nodes and create a new topic where its message has a different type. The advantage in topics is that there could be multiple subscribers and publishers for one topic and a single node can act as subscriber and/or publisher. The topics used in Figure 24 are /odom, /cmd_vel, /scan and /tf. The points below give a brief description of each topic:

- */odom* acts as a publisher where it transfers the position of the robot in the simulation and sending it to the move_base node.
- */cmd_vel* publishes the velocity commands from move_base to gazebo with the purpose of moving the robot model and the Nexus robot.
- */scan* contains the laser scanner data from gazebo and it publishes the information to hector_mapping which is responsible for mapping the terrain.
- */tf* consists of transformation coordinate frame of the robot and it publishes these coordinates to move_base, hector_mapping and hector_trajectory_sever nodes and some subscribing to itself.

The /tf topic is a library where it keeps track of the coordinate frame of the robot with the ability to give information to nodes about the state of the robot. The core structure of /tf is a

transform tree as it gets the transform between coordinate frames and updating rate of the frame. Figure 25 displays the transform tree.

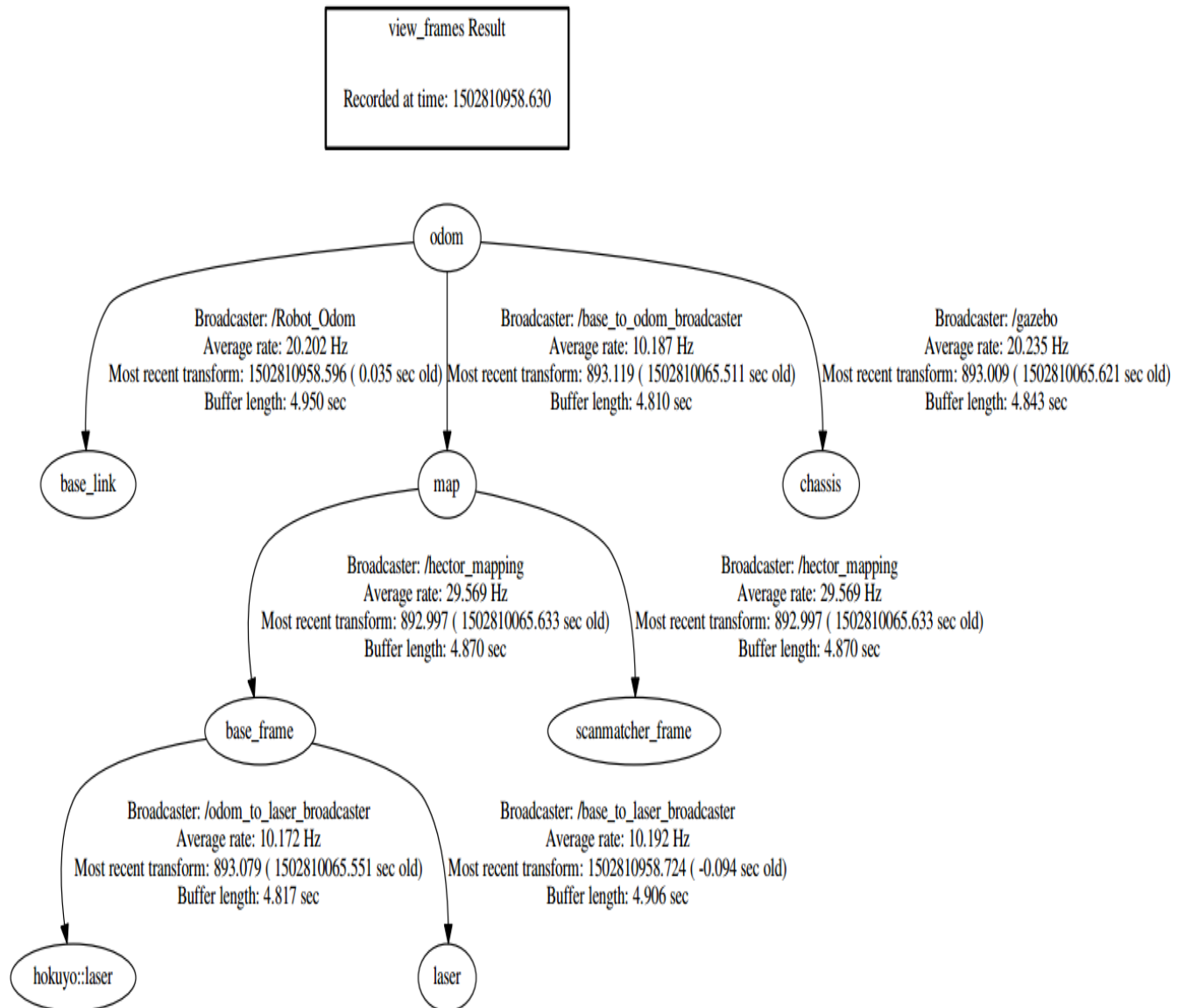


Figure 25: Transform tree of the robot

Hence, when the laser scanner scans the terrain and the user assign a goal in the map the laser sends the data to the base frame which is the robot ID frame and then send the data to the map ID frame where it is used to map the environment and sends the odometry information of the robot to odom ID frame.

4.1.3 SLAM mapping:

For the robot to be able to navigate the terrain successfully, it would need a map of the surrounding environment and to be able to know where it is located on the map. Therefore, the package used to implement the mapping part is HectorSLAM. Figure 26 displays the overview of HectorSLAM package.

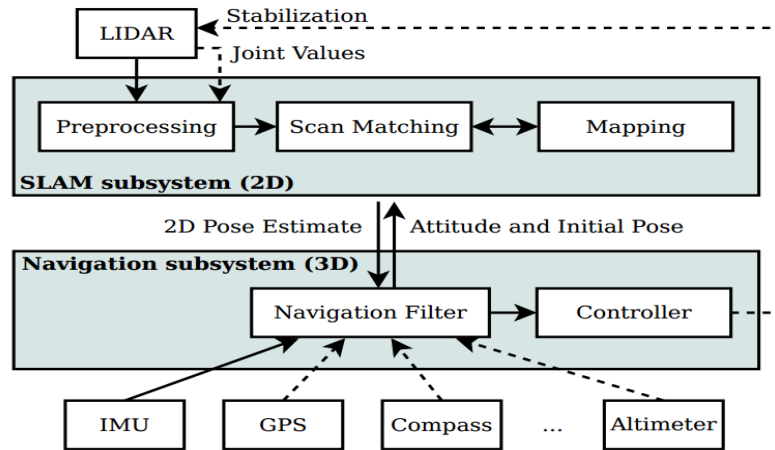


Figure 26: HectorSLAM package (reproduced from (Kohlbrecher et al., 2011))

HectorSLAM uses Hokuyo range finder sensor data to scan the territory so that the Nexus robot will be able to move around the map. The package uses an algorithm called FastSLAM where the land is characterised as 2D map. As the update rate of the laser is around 30 Hz it can produce an approximate map of the landscape. The FastSLAM algorithm also involves in converting the scanned map into point cloud endpoint scan, the algorithm only uses filtered endpoints on the z coordinate and bilinear filtering makes use of these endpoints in order to improve the accuracy of the map through estimating the occupancy spatial derivatives and probabilities (Kohlbrecher et al., 2011). Figure 27 shows the process of Hector mapping.

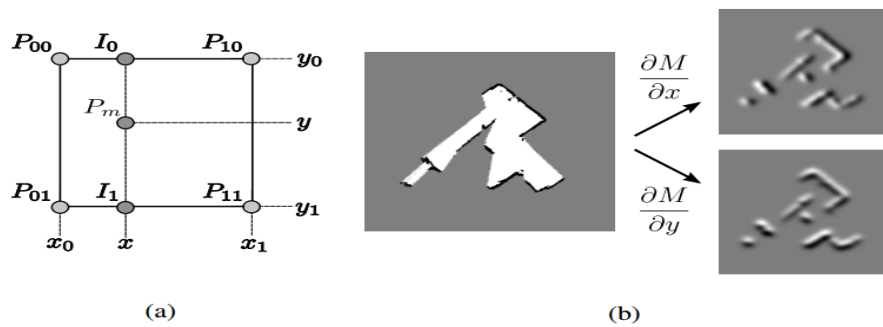


Figure 27: Hector SLAM mapping method; (a) bi-linear filtering method, (b) occupancy grid map and spatial derivatives (reproduced from (Kohlbrecher et al., 2011))

The scan matching algorithm is also another part of HectorSLAM package that utilises the z coordinate endpoints and aligns the laser scans with the endpoints. Scan matching algorithm contains an optimisation approach called Gauss-Newton approach where it tries to align the scanned data and the endpoints with the existing map and tries to eliminate all laser scanned data that are not associated with the map (Kohlbrecher *et al.*, 2011).

4.1.4 Navigation:

After acquiring the map of the environment, the unmanned vehicle must be able to path plan its way around the terrain and this is done through the Navigation_Stack package from ROS. The Navigation_Stack helps the autonomous robot to acquire global plan from the robot's starting point to the end goal. The planner divides into two parts; *Global Planner* and *Local Planner*. Figure 28 shows the concept of Navigation_Stack.

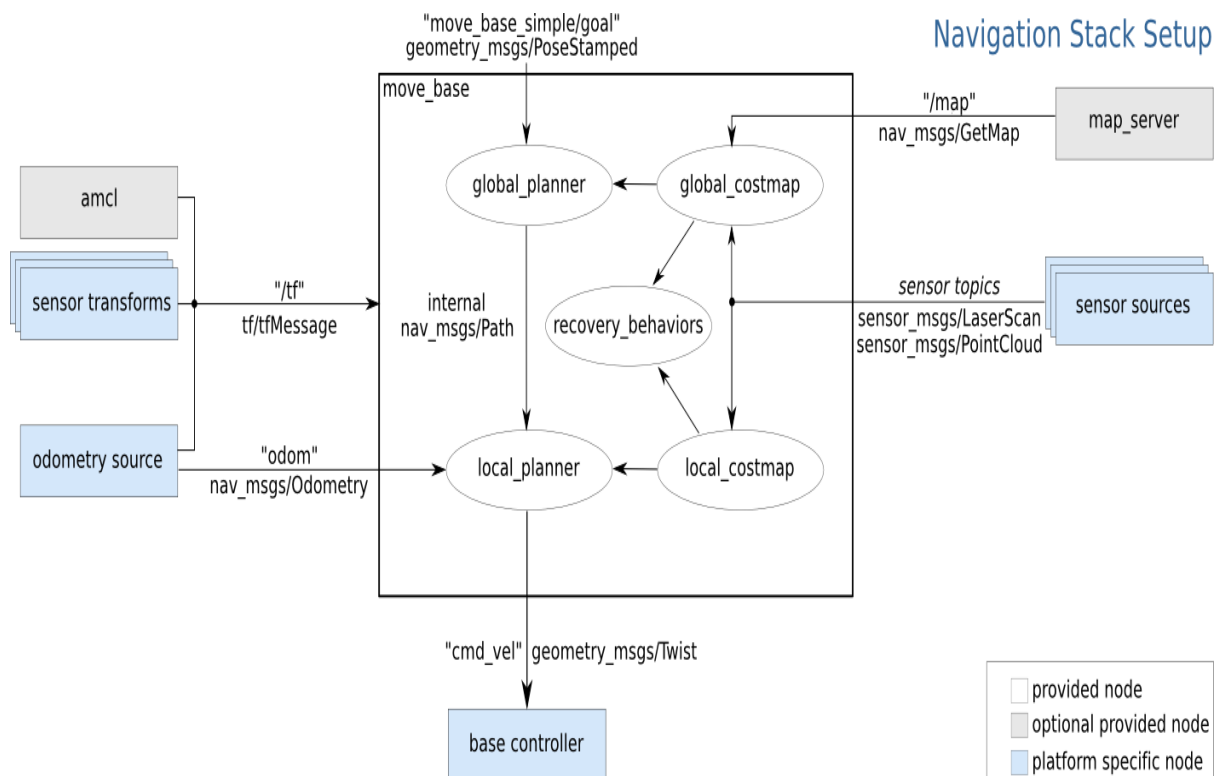


Figure 28: Overview Structure of Navigation_Stack (reproduced from (ROS, no date))

Global Planner is responsible for calculating the global route towards the goal and these paths are calculated through path planning algorithms. The planner contains several path planning algorithms, but for this project, it will concentrate on two path planning algorithms. First one is called A star algorithm and second is called Dijkstra algorithm. These algorithms can be activated through costmap parameters in the Navigation_Stack.

I. Dijkstra algorithm:

Dijkstra algorithm is a path planning algorithm that calculates the shortest path between two nodes in a map. The points below explain the process of Dijkstra algorithm and Figure 29 represents the pseudo code of the algorithm:

- The algorithm takes the whole map and places the starting node in the initial position of the robot and place the distance to other nodes to infinity.
- The algorithm contains two statements:
 - 1) At the initialization stage, All the visiting nodes are considered empty.
 - 2) All the surrounding nodes expect a starting node.
- On the assumption that the unvisited nodes are not empty, then mark the node that has a minimum distance as visited and observe the surrounding nodes if it is possible to find the shortest path. The cost of the plan is the sum of the edge costs from the initial node to the goal node.
- If there is a path, then it will be marked and follow the previous step.

```
dist[s] ← 0                                (distance to source vertex is zero)
for all v ∈ V - {s}
do dist[v] ← ∞                             (set all other distances to infinity)
S ← ∅                                       (S, the set of visited vertices is initially empty)
Q ← V                                       (Q, the queue initially contains all vertices)
while Q ≠ ∅                                (while the queue is not empty)
do u ← mindistance(Q, dist)                (select the element of Q with the min. distance)
   S ← S ∪ {u}                             (add u to list of visited vertices)
   for all v ∈ neighbors[u]
   do if dist[v] > dist[u] + w(u, v)         (if new shortest path found)
      then d[v] ← d[u] + w(u, v)           (set new value of shortest path)
                                           (if desired, add traceback code)

return dist
```

Figure 29: Dijkstra algorithm Pseudocode (reproduced from (Yan, 2016))

The algorithm can only work with a 2D map with no negative transitions. The 2D map is basically from SLAM and the algorithm considers the initial robot position pixel on the map as the starting node and the surrounding pixels will be an unvisited node except the obstacles. When the goal is assigned in the map, the pixel will consider it as the goal node and will undergo the process to find the minimum distance towards the goal and hence it will develop a global path towards the endpoint.

II. A star algorithm:

A star algorithm is another path planning algorithm that calculates the cost of the node using cost function and it is based on Dijkstra algorithm. The following equation is the cost function:

$$F(n) = G(n) + H(n)$$

Where $F(n)$ is the total cost of the path, $G(n)$ is the cost of starting node to the robot's present position node, $H(n)$ is the cost that uses either Manhattan Distance or Chebyshev Distance to calculate the estimated distance cost from the node (n) to the goal node (Gross, 2017). Figure 30 shows the A star algorithm using Chebyshev Distance.

			Goal g=5	
g=3, h=3, f=6			g=4, h=1, f=5	g=4, h=1, f=5
g=2, h=3, f=5				g=3, h=2, f=5
g=1, h=3, f=4			g=2, h=3, f=5	g=3, h=3, f=6
g=1, h=4, f=5	g=0, h=4, f=4	g=1, h=4, f=5	g=2, h=4, f=6	g=3, h=4, f=7

Figure 30: A algorithm with Chebyshev Distance where the red boxes are obstacles and the orange boxes are path taken towards the goal(reproduced from (Gross, 2017))*

Since the A star algorithm is based on Dijkstra algorithm (Sniedovich, 2006), then it will look at the minimum cost towards the goal. However, the important fact that $H(n)$ should not be overestimated where if $H(n)$ was overvalued it would cause errors in the calculations and it would lead to a path that would take a longer route towards the goal.

As the Global Planner assigns the path from an initial position towards the goal, the **Local Planner** uses the Global path for the robot to follow and the Local Planner can detect the surrounding environment for obstacles. So, if the Local Planner detects an obstacle and it is blocking its path towards the goal, it will recalculate the path so that it can avoid the obstacle. Hence, this will create a new path towards the goal.

4.2 Hardware methodology:

This methodology will consider Nexus robot kit connections where it contains sensors and motor controller that will receive velocity commands from Raspberry Pi and transmit the Laser scan data to SLAM in the computer to map the landscape.

4.2.1 Robot function block:

Figure 31 shows the functional block diagram of the robot. A 12V Lithium-Polymer (Li-Po) battery is used to power the Nexus duino board since the voltage regulator regulates the 12 volts into 5 volts in which it lies within the safe operating voltage of the Nexus duino board. The battery is also used to power the expansion board, the motor controller chip L298P and the motors. The Raspberry Pi board is powered by Power-Bank of 2.4V and the board is connected to the Nexus duino board. The Raspberry Pi is used to send velocity commands to Nexus duino and Nexus duino collects these commands and send it to the motor controller to turn on the motors. Nexus duino board collects the odometry information from the encoders and send it to Raspberry Pi where it can calculate the distance travelled by the robot. Moreover, the laser scanner will scan the environment, send it to Raspberry Pi so that it can be sent to a desktop computer to be used in HectorSLAM. Figure 32 shows the hardware connections.

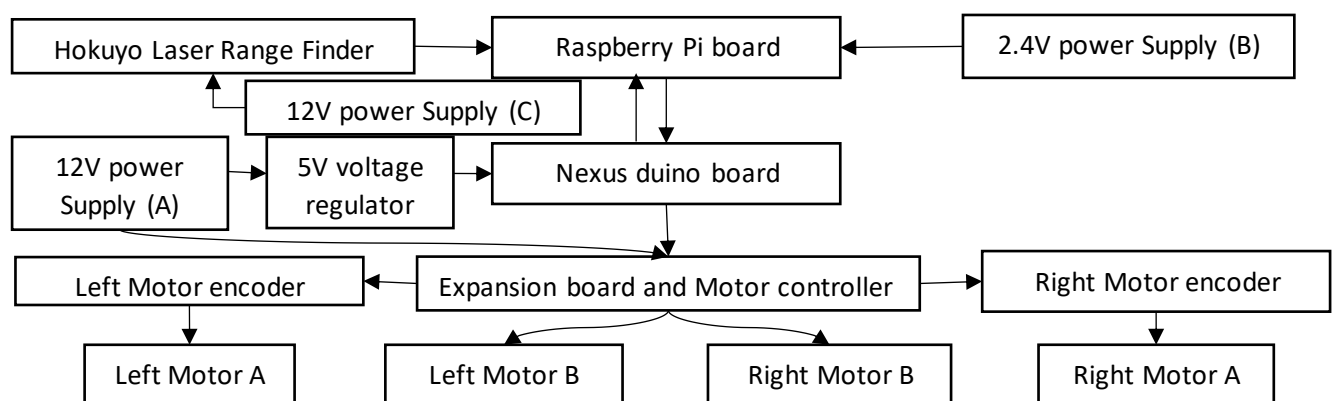


Figure 31: Nexus robot block diagram



(a)



(b)



(c)

Figure 32: Final design of Nexus robot: (a) Nexus robot with laser connection, (b) connection between Nexus duino and Raspberry Pi, (c) sideways image of robot

4.2.2 Laser power supply:

Hokuyo laser requires around 12V for safe operating voltage and 375mA current. Therefore, 4 series Ni-MH battery was used to power the laser. Single cell releases around 3300mAh and 1.2V. However, as the batteries are connected in series, then the voltage will be added where the final output voltage is 4.8V, so, the circuit board B6286 is used where it can boost and regulate low input voltage to high output voltage. The board can give out an adjustable voltage from 2V to 28V by turning the screw on Trimming Potentiometer as shown in Figure 33 and the installation of the circuit board is shown in Figure 34 and 35. Hence, by using this board it is possible to supply 12V and 375mA to Hokuyo laser.

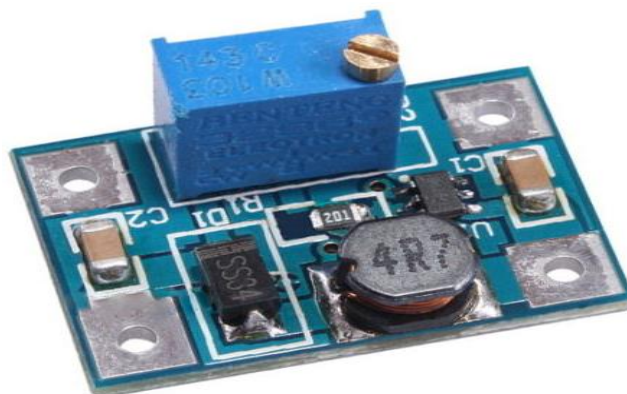


Figure 33: B6286 Step-Up Adjustable Boost converter board

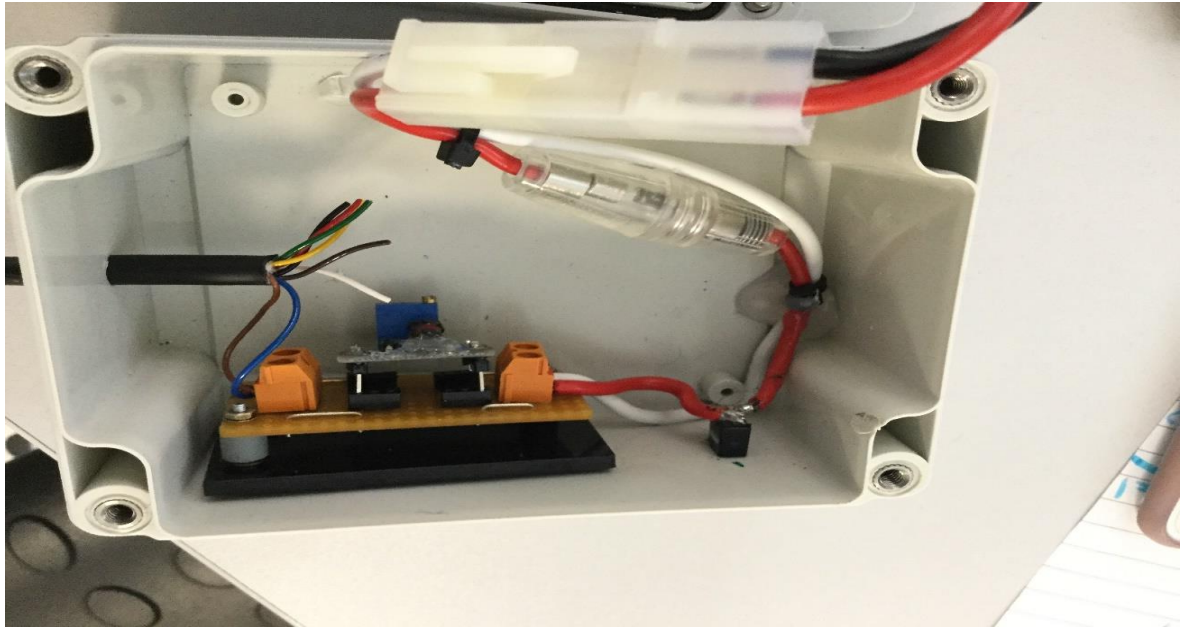


Figure 34: B6286 with 2A fuse

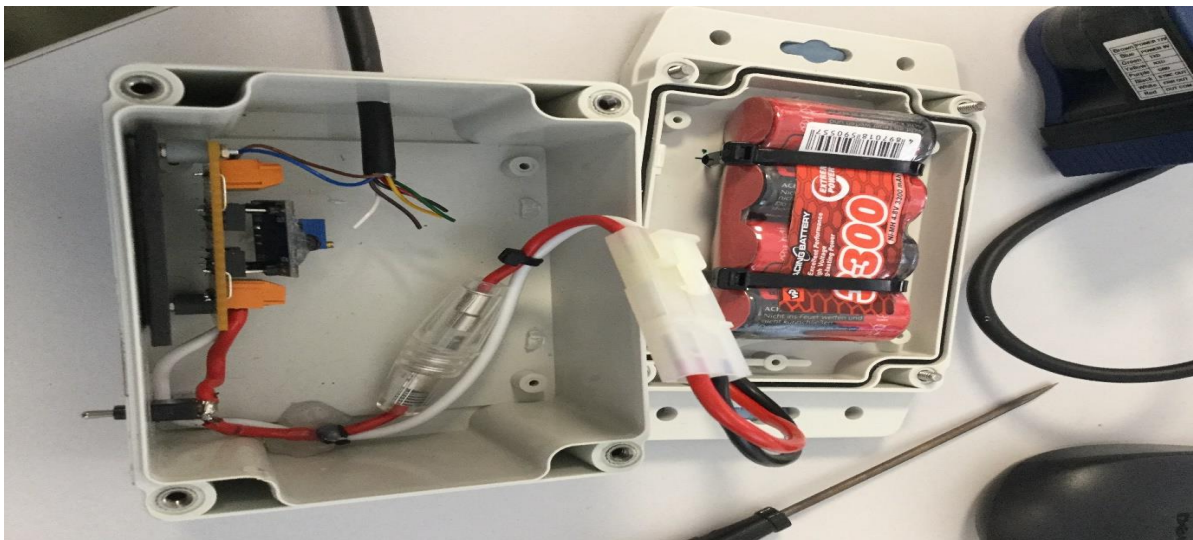


Figure 35: Final design for Laser power supply

As shown in Figure 34 and 35, the wires are also fitted with 2A fuse as a safety precaution where it prevents the load current to exceed the safe operating current. Since the Laser draw continuously around 375mA so it is possible to find the battery runtime by using battery capacity equation:

$$\begin{aligned} \text{battery runtime} &= \frac{\text{Battery capacity}}{\text{current discharge}} \\ &= \frac{3300\text{mAh capacity}}{375\text{mA drawn}} \approx 8.8 \text{ hours} \end{aligned}$$

Chapter 5 – Experimental procedure and results:

5.1 Experimental procedure:

Before combining all the components together, some preliminary tests were conducted to form an understanding about individual component's functionality, limitations and produce some codes before installing them into the Nexus robot. Figure 36 looks into testing Hokuyo Laser and Figure 37 represents a test for odometry of the robot.

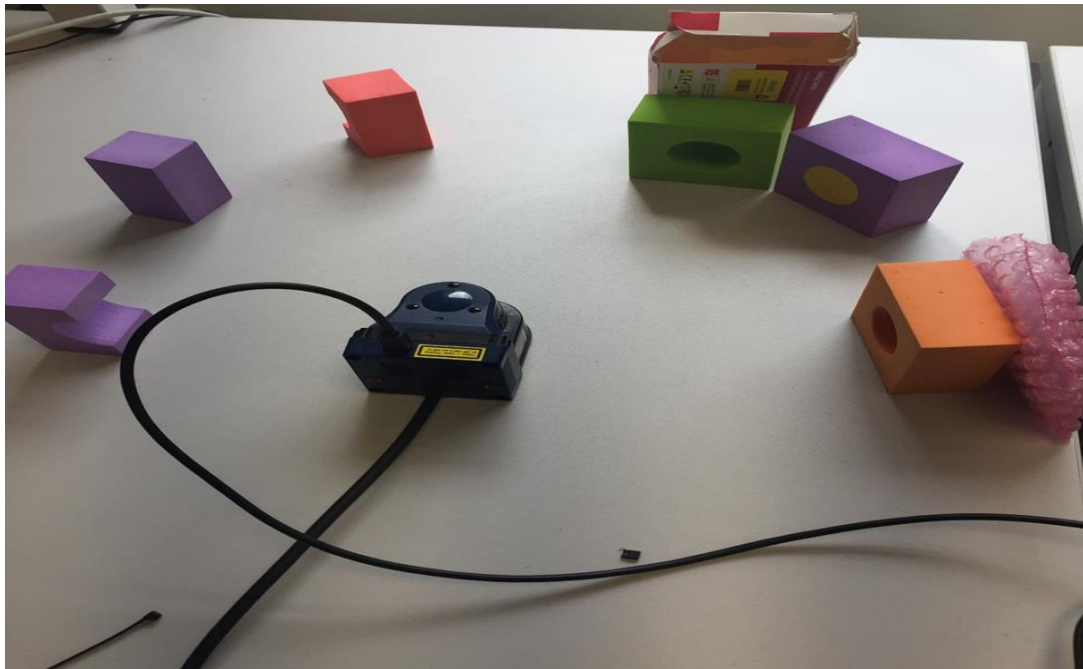


Figure 36: Hokuyo laser test for range detection

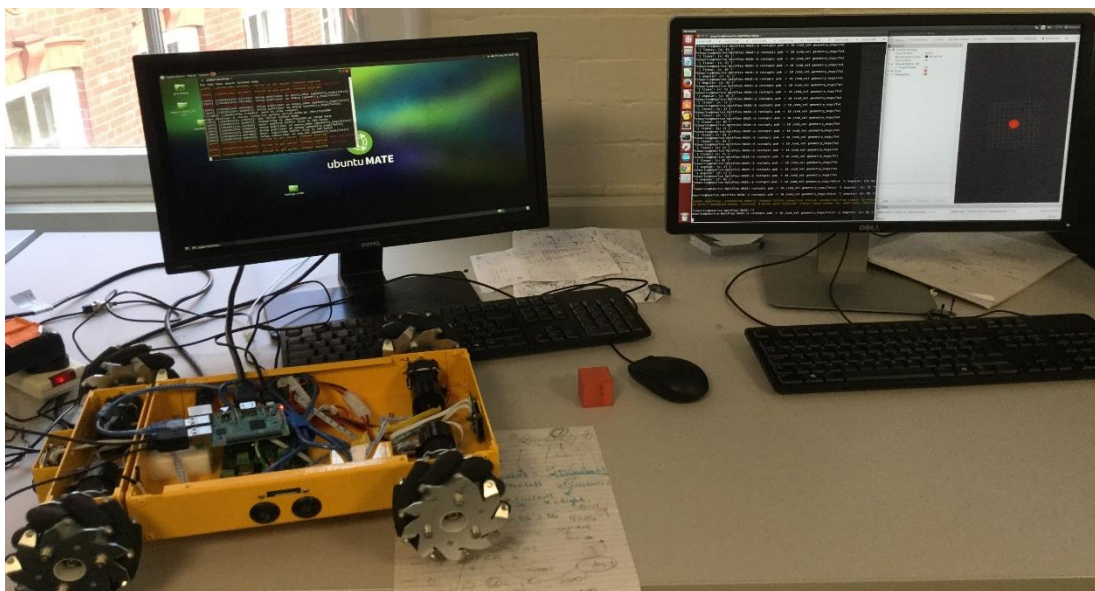


Figure 37: Encoder test for localisation

However, during the testing phase and acquiring the distance between the laser and the objects, the laser was damaged due to problems in internal components, on the other hand, Gazebo has a model of the Hokuyo laser as shown in Figure 38.

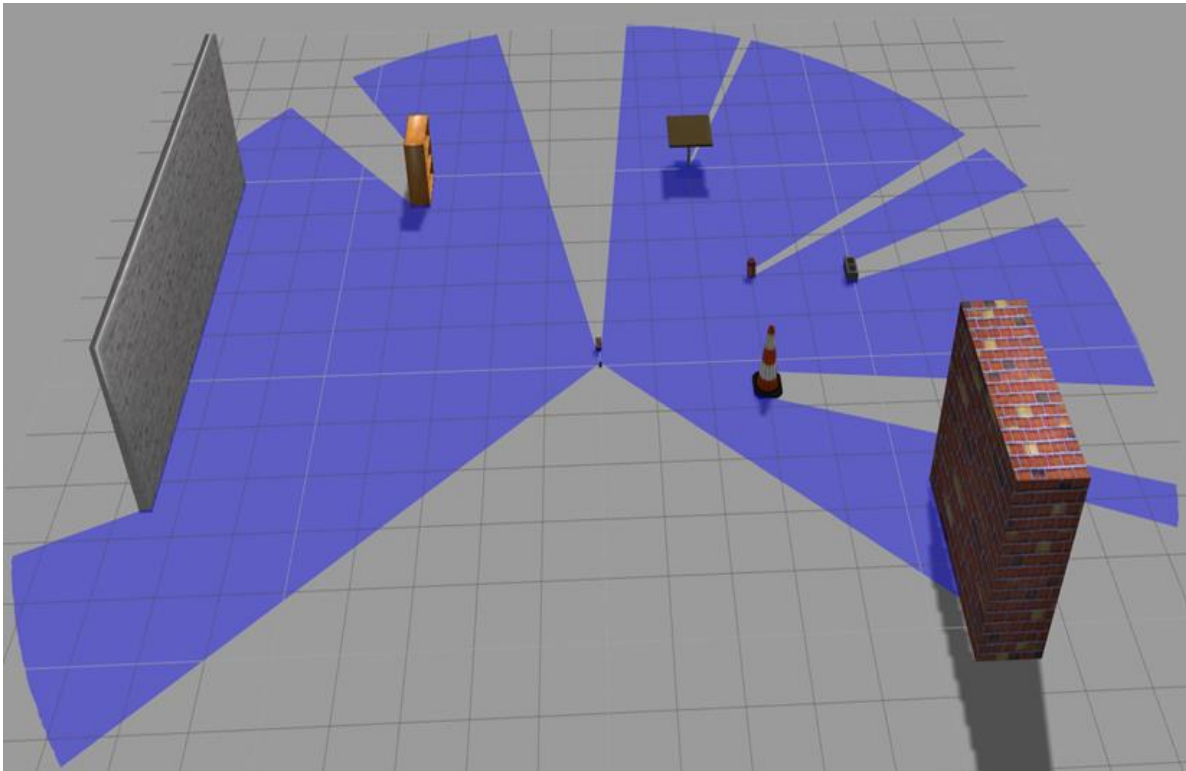


Figure 38: Hokuyo laser test for range detection in Gazebo where the blue lines represents the emitted laser from scanner

As soon as the individual testing was done, a simple environment was constructed in both real and in simulated worlds as shown in Figure 39 and 40. Initially, the robot was placed in a corner of the simple environment. Through the visualization program called RViz, it is possible to use the scanned data from the laser to produce the map of the terrain. Furthermore, a goal can be assigned in RViz which will cause the robot to start navigating through the terrain in order to reach the goal and it also undergoes obstacle avoidance if it detects an obstruction on the way to the goal. To make the experiment more challenging, a complex map was developed in simulation in order to observe the navigational behaviour of the robot through more complex terrain and this map is illustrated in Figure 41. The results of the experiment are shown in the following section.

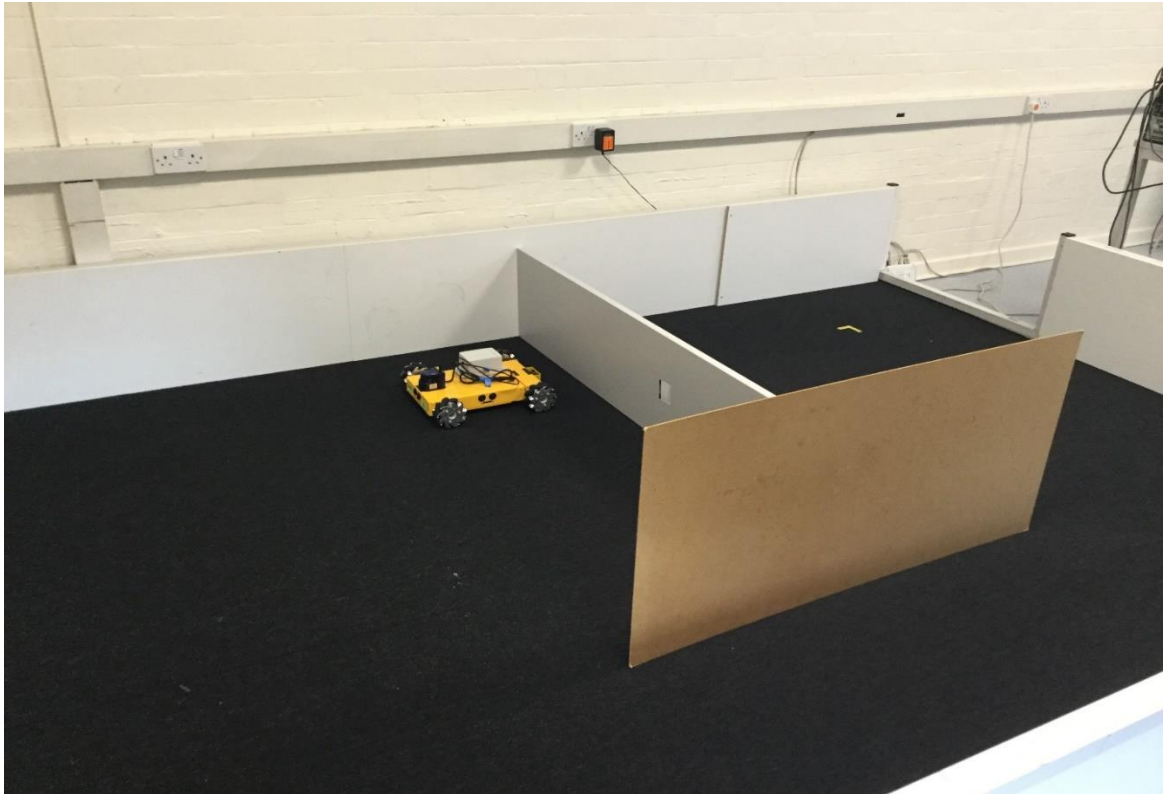


Figure 39: Robot inside simple environment

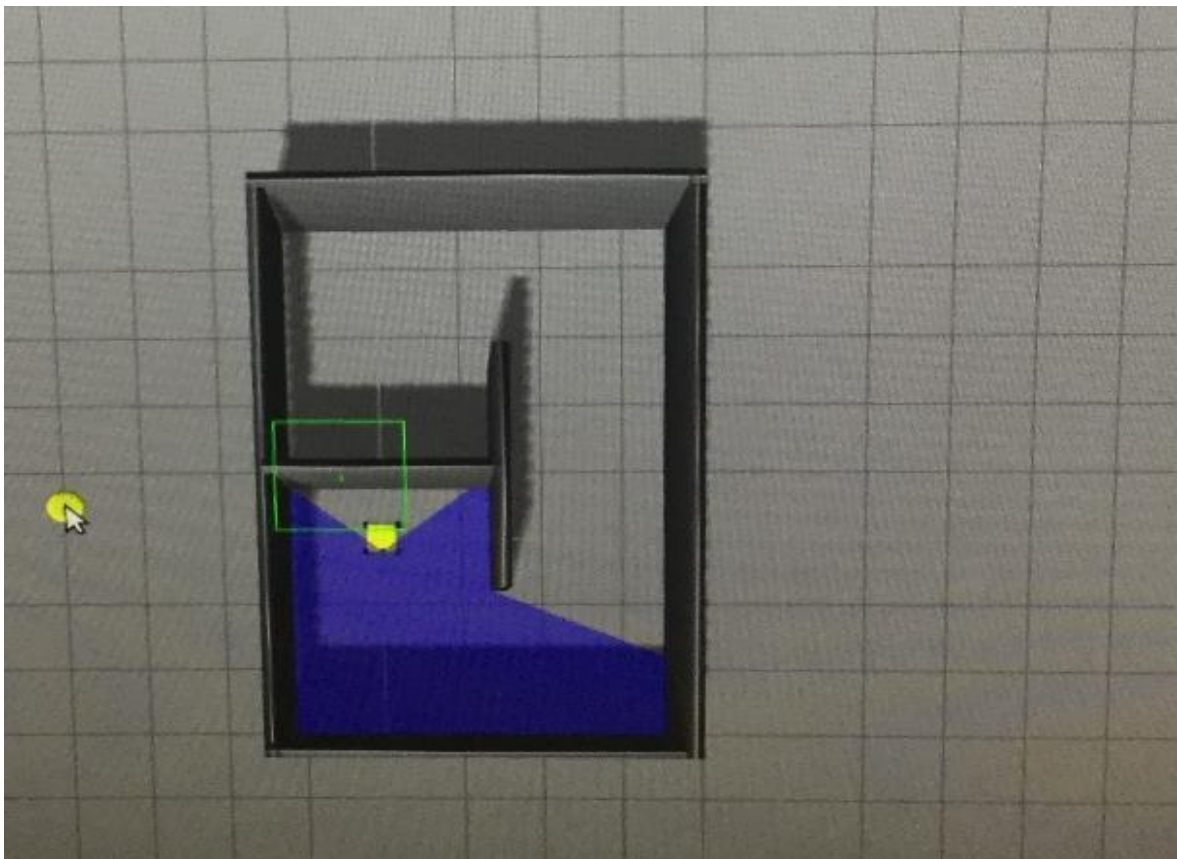


Figure 40: Gazebo version of the simple environment with Nexus robot.

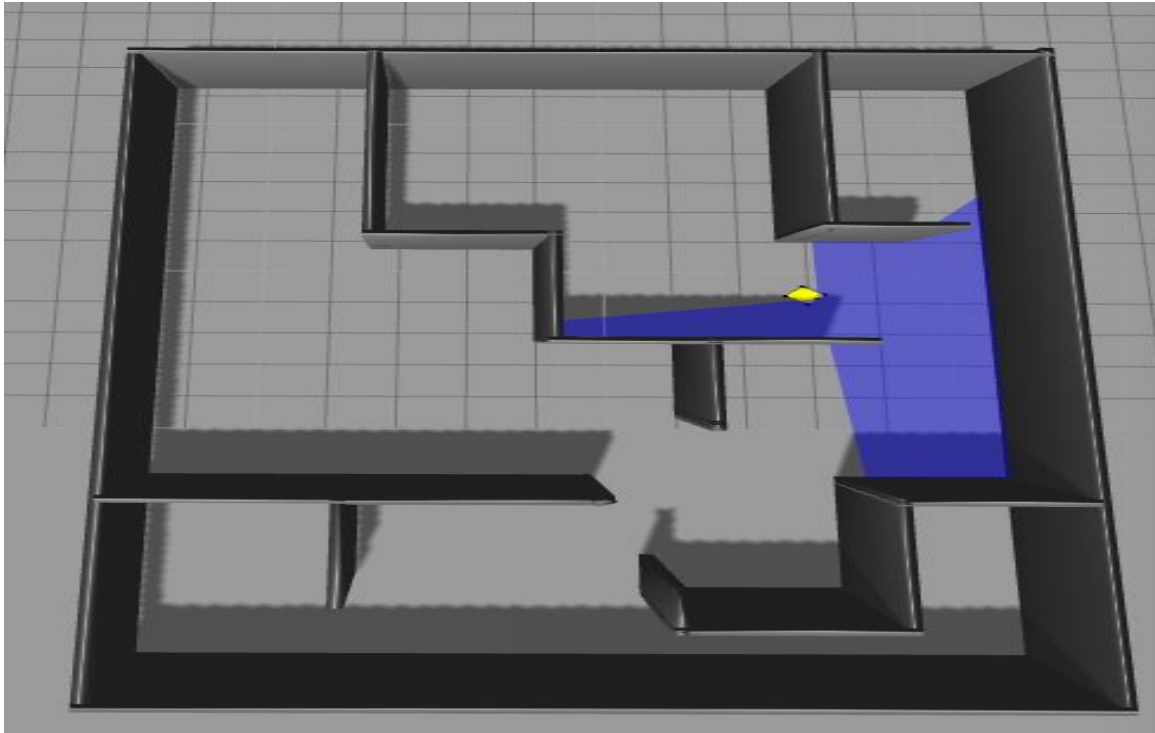


Figure 41: Gazebo version of the complex environment with Nexus robot.

5.2 Results:

When conducting the preliminary tests, the laser managed to scan the environment and could acquire the distance of each object in the environment. Figure 42 shows the scanned environment in the real world before the laser was damaged and Figure 43 represents the scanned environment in Gazebo.

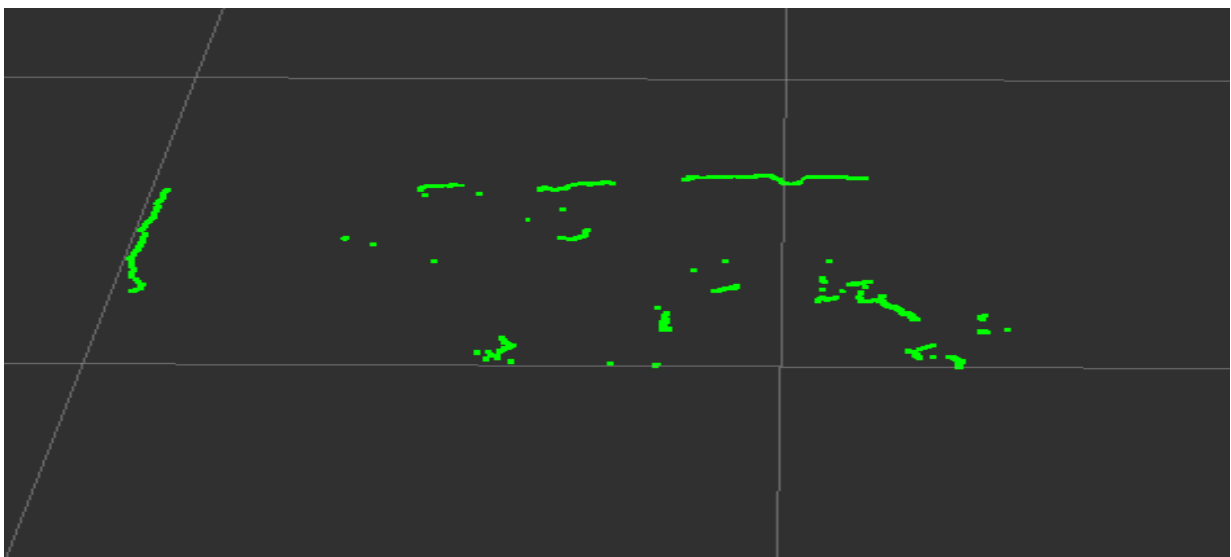


Figure 42: RViz visualisation of the laser scanner where the green lines represent the objects being detected by the laser

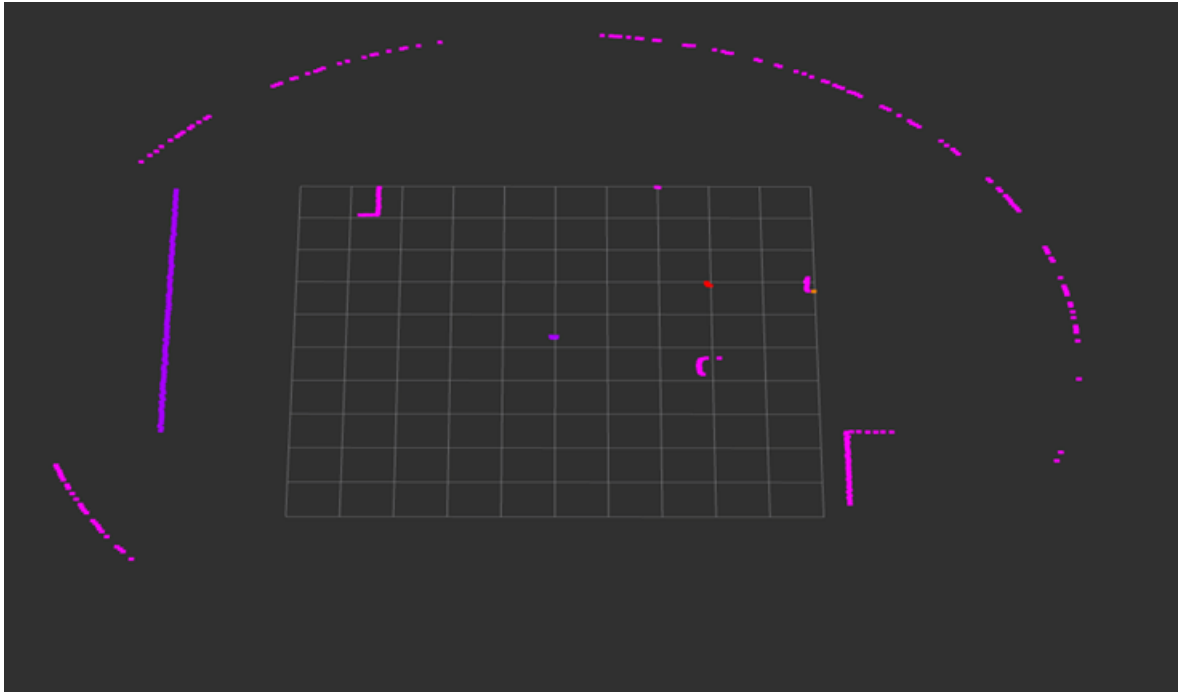


Figure 43: RViz visualisation of the laser scanner where the pink and purple lines represent the objects being detected by the laser in Gazebo

Moreover, the odometry of the robot was recorded and visualised in RViz with the intention of finding the location of the robot. Figure 44 and 45 illustrate the forward movement and Figure 46 and 47 displays the rotational movement of the robot.

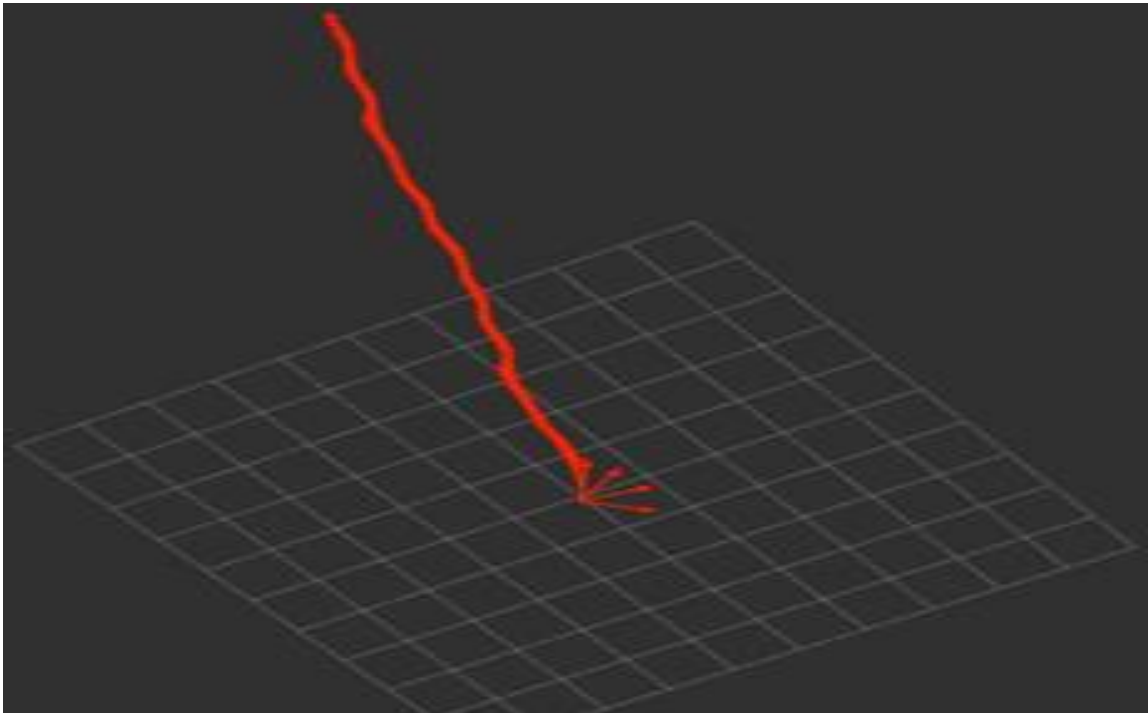
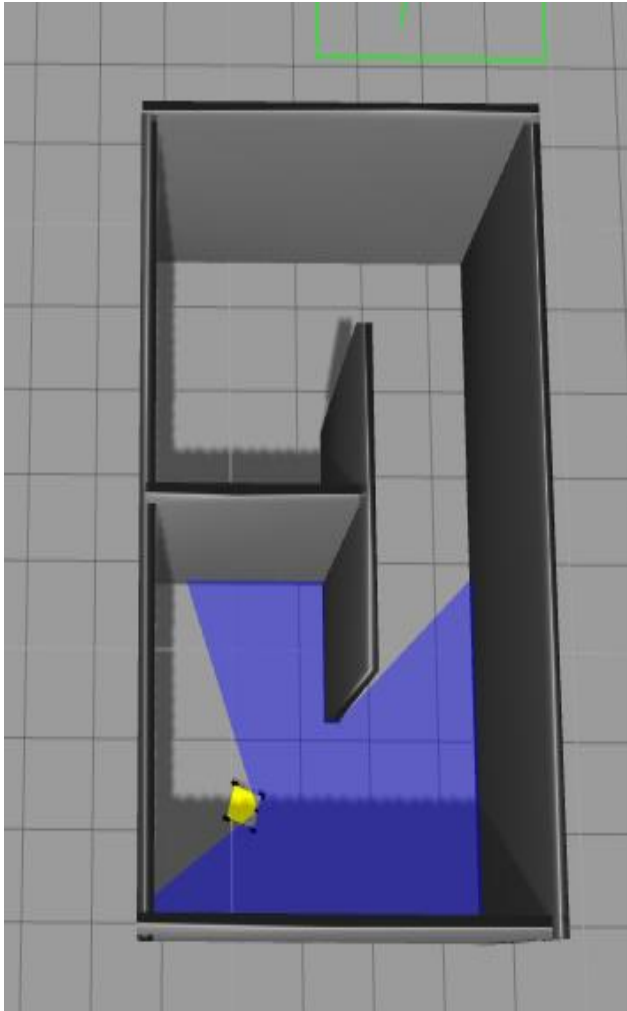
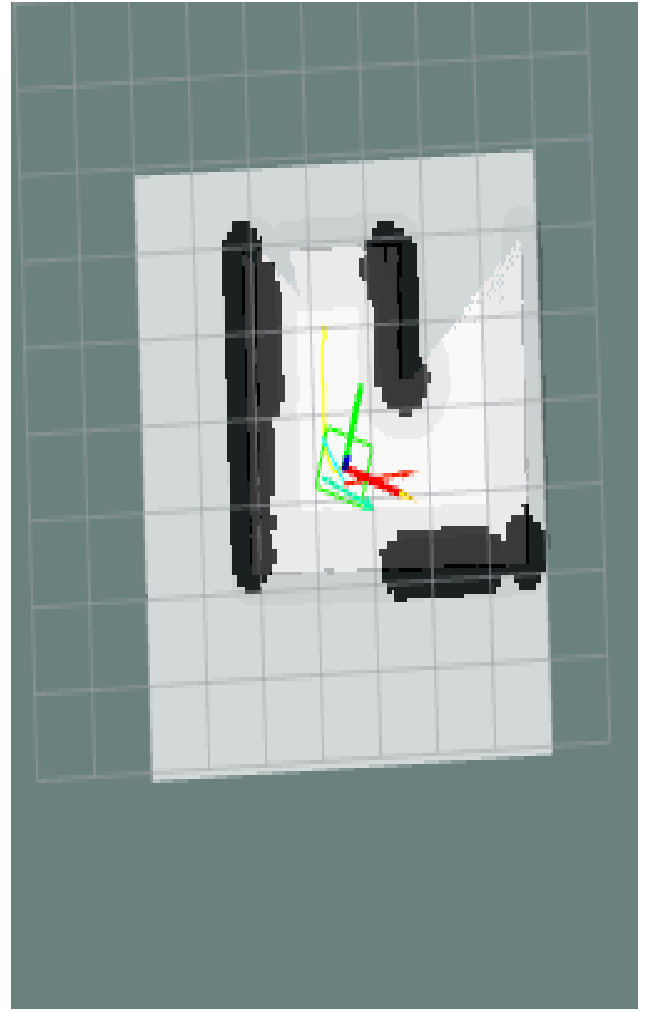


Figure 44: Robot forward movement visualisation where red arrow represents Nexus robot

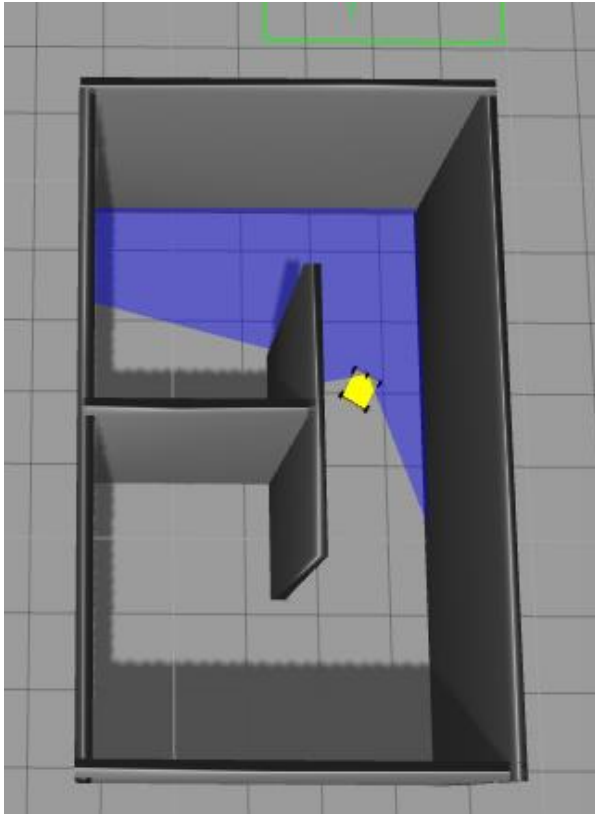


(a)

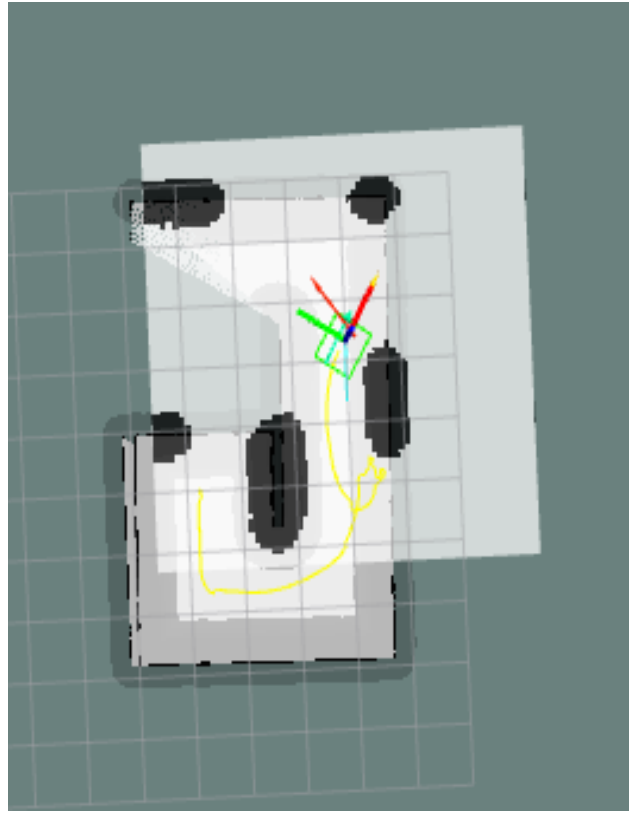


(b)

Figure 48: Robot's movement in the simple environment: (a) Gazebo result, (b) RViz result

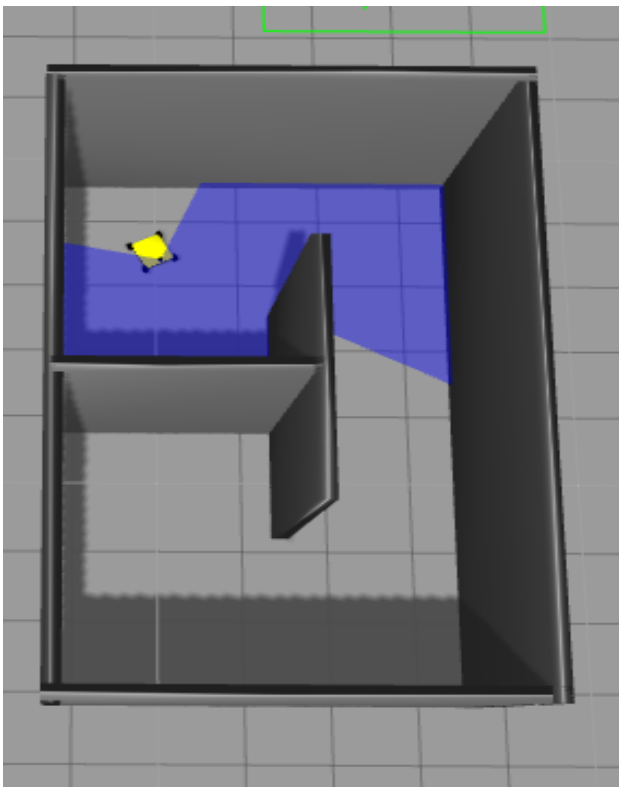


(a)

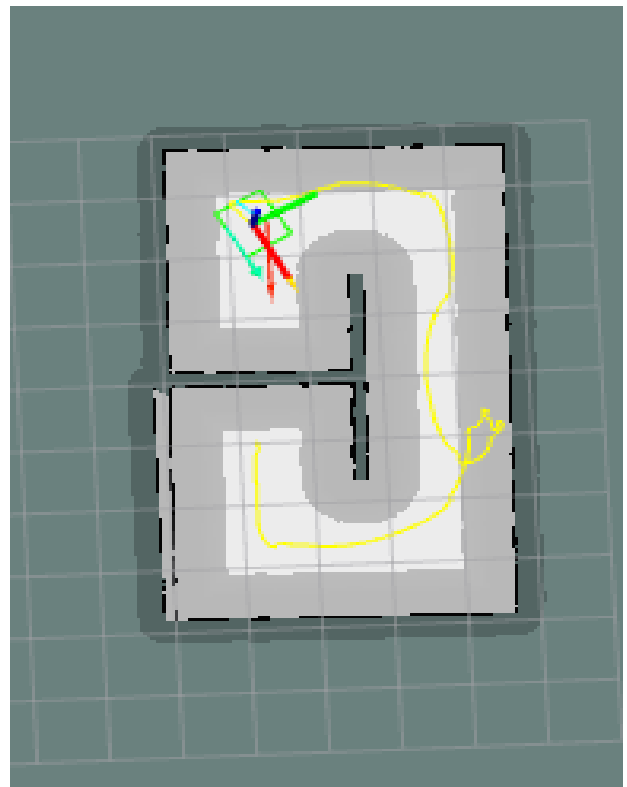


(b)

Figure 49: Robot's movement in the simple environment: (a) Gazebo result, (b) RViz result

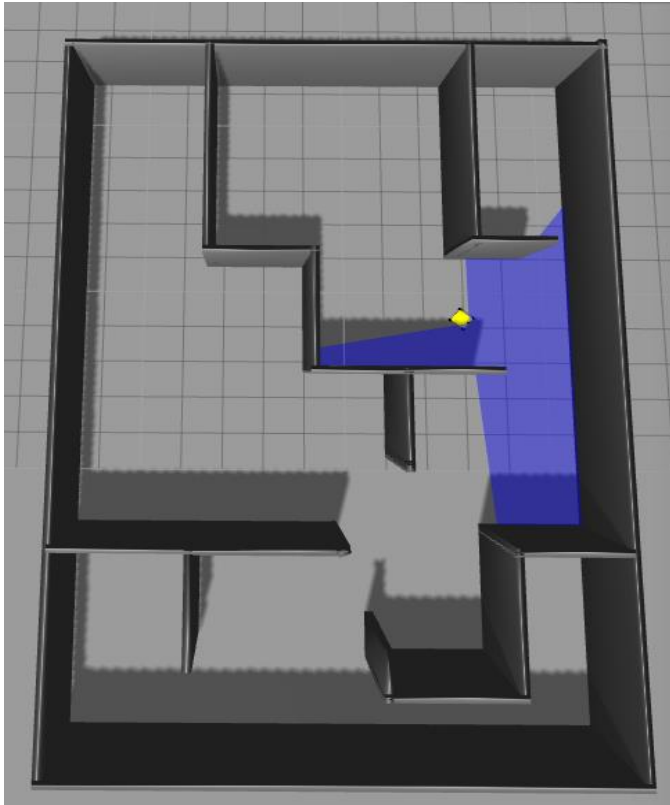


(a)

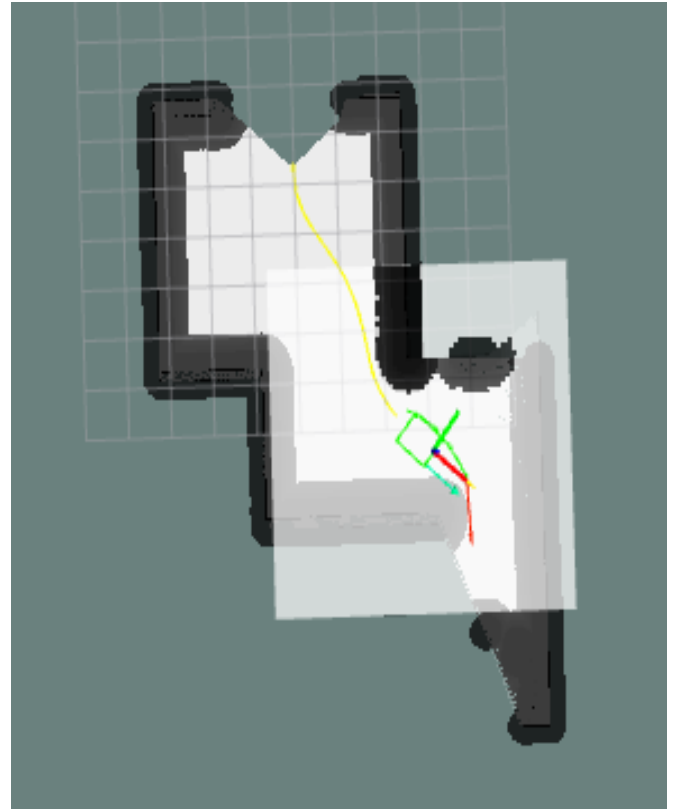


(b)

Figure 50: Final map of the simple environment

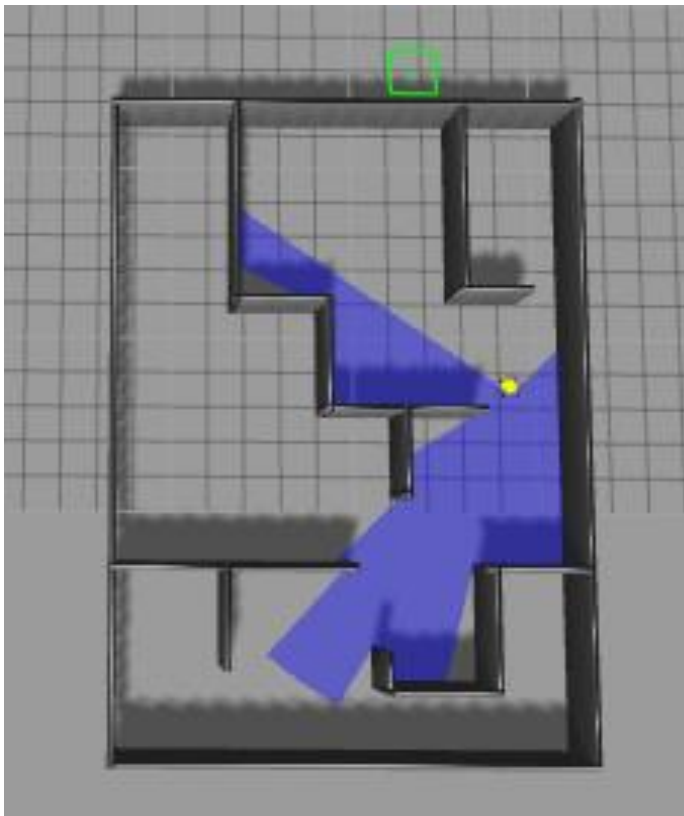


(a)

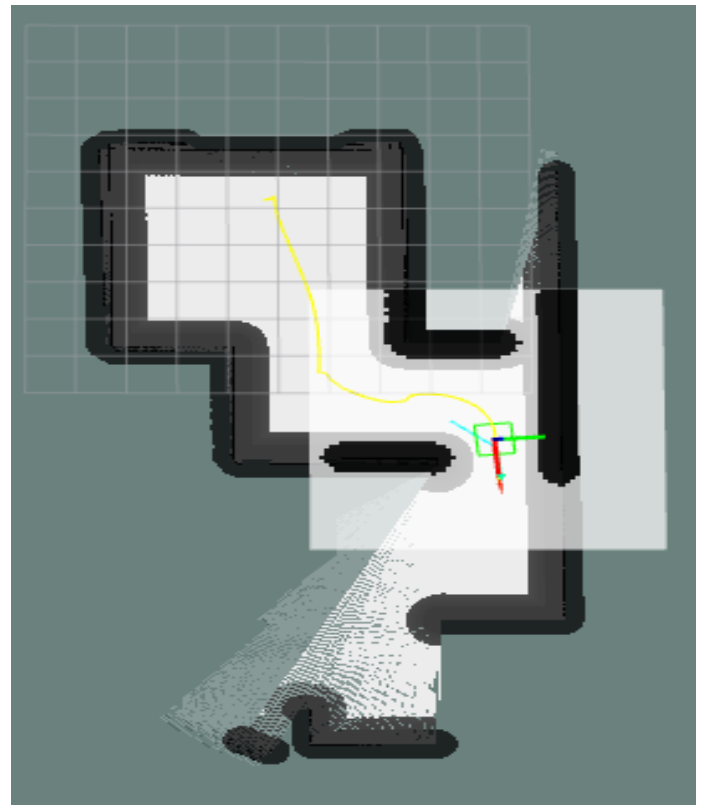


(b)

Figure 51: Robot's movement in the complex environment: (a) Gazebo result, (b) RViz result

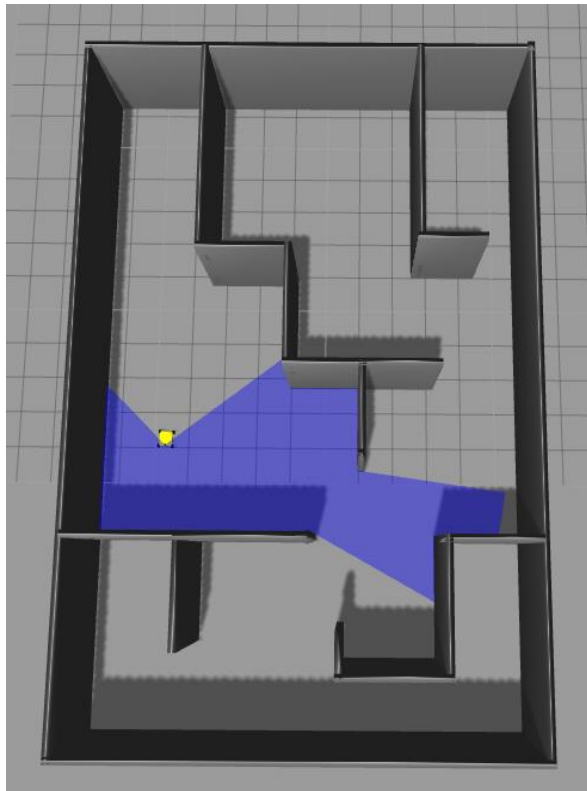


(a)

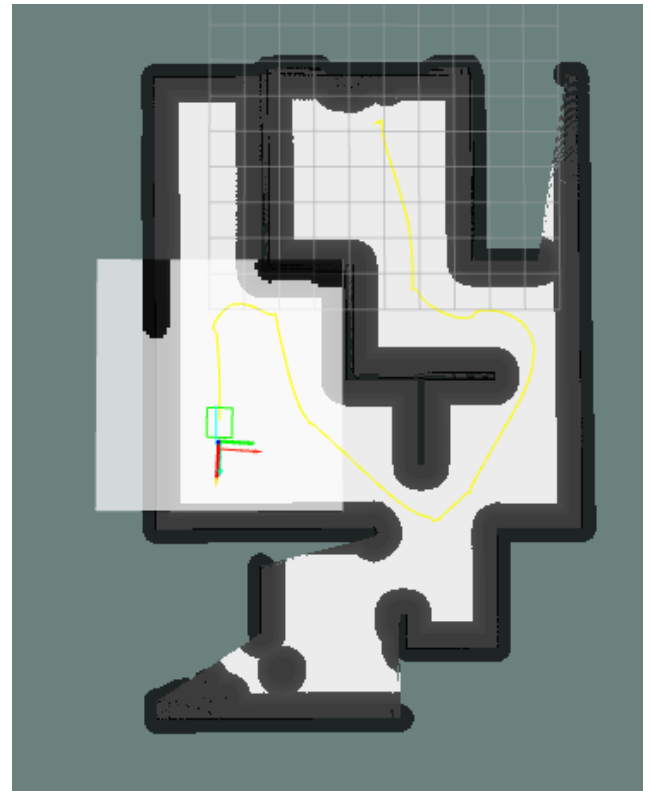


(b)

Figure 52: Robot's movement in the complex environment: (a) Gazebo result, (b) RViz result

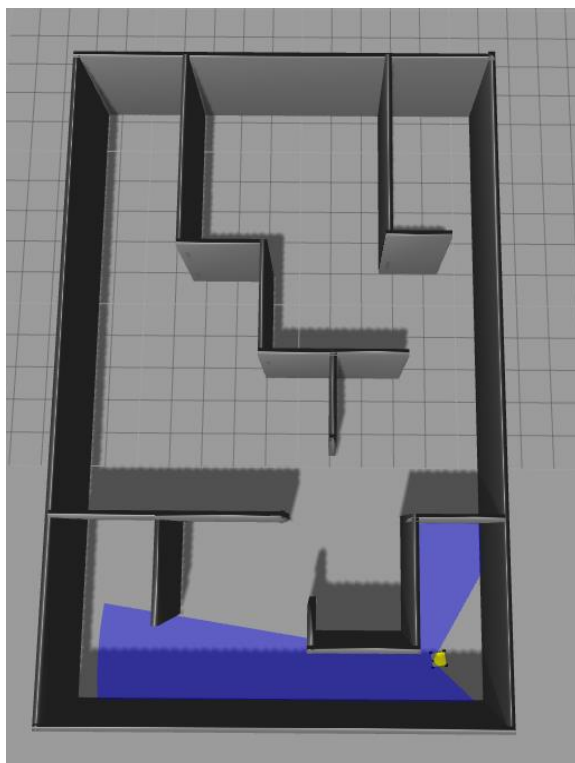


(a)

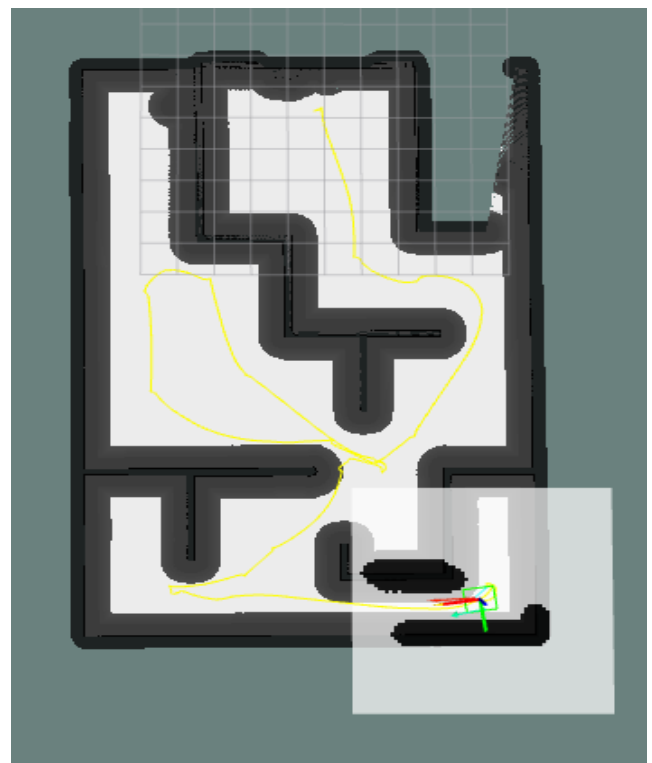


(b)

Figure 53: Robot's movement in the complex environment: (a) Gazebo result, (b) RViz result

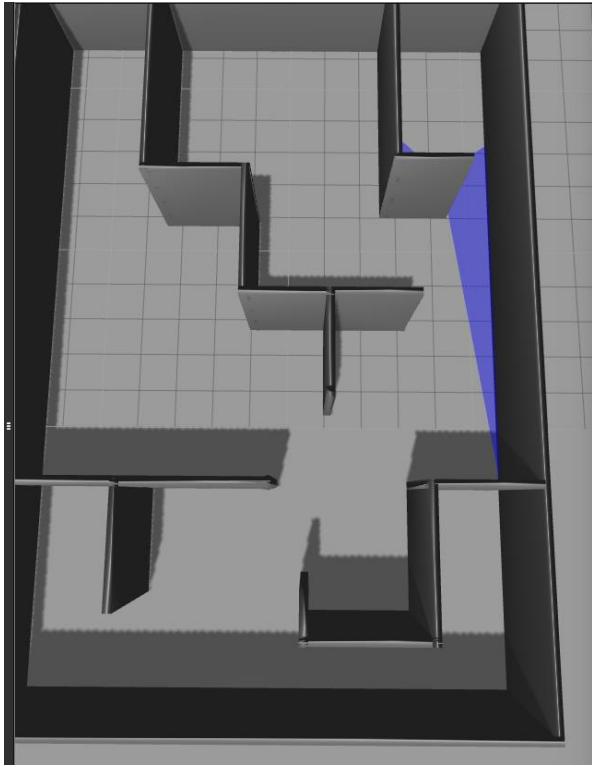


(a)

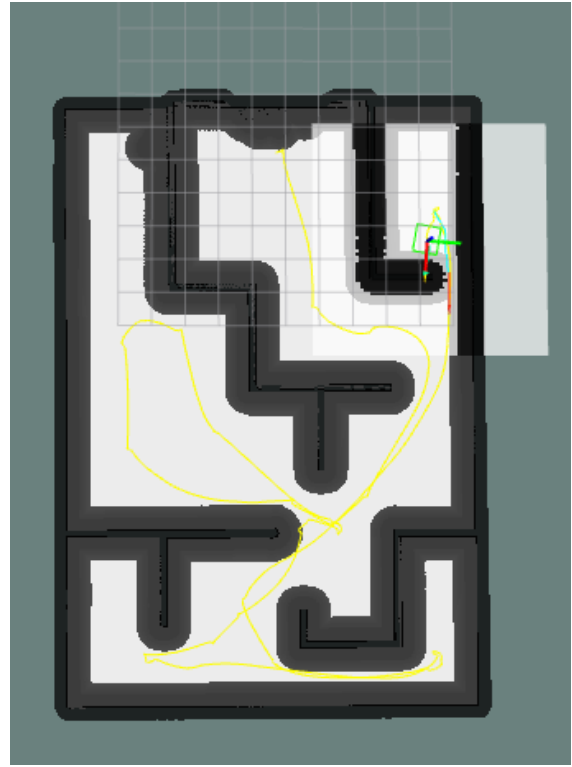


(b)

Figure 54: Robot's movement in the complex environment: (a) Gazebo result, (b) RViz result

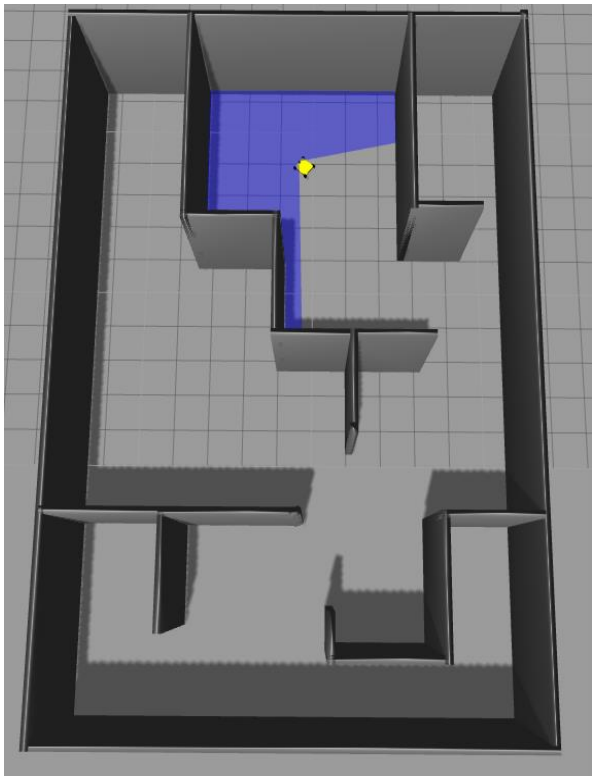


(a)

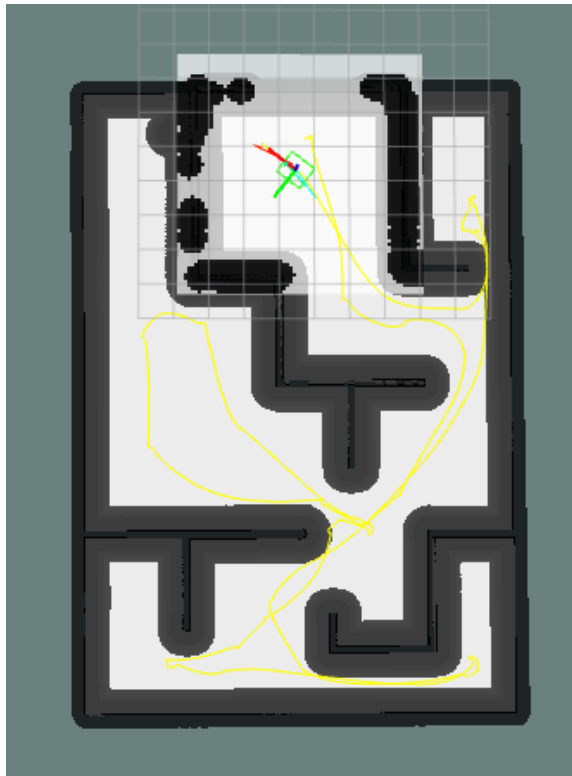


(b)

Figure 55: Robot's movement in the complex environment: (a) Gazebo result, (b) RViz result

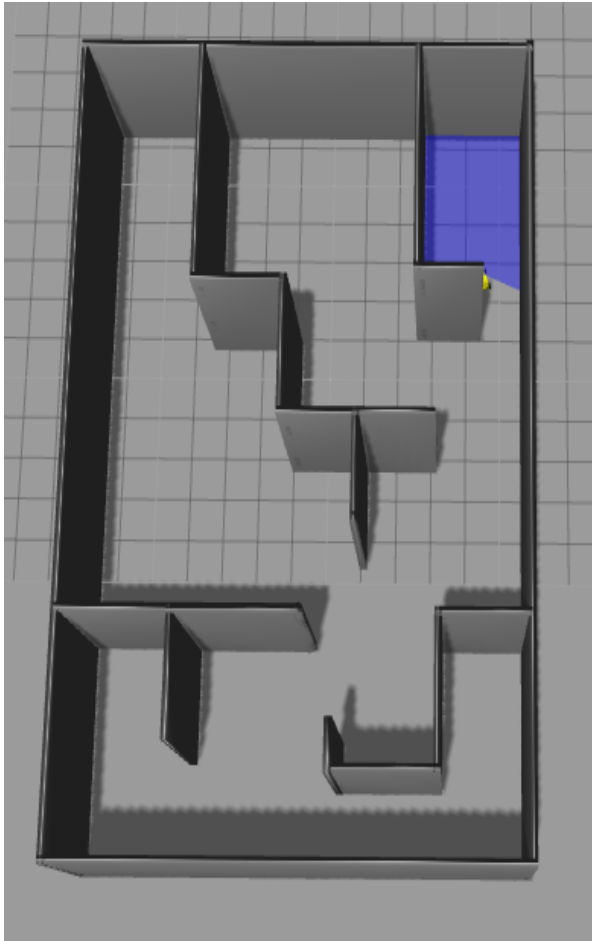


(a)

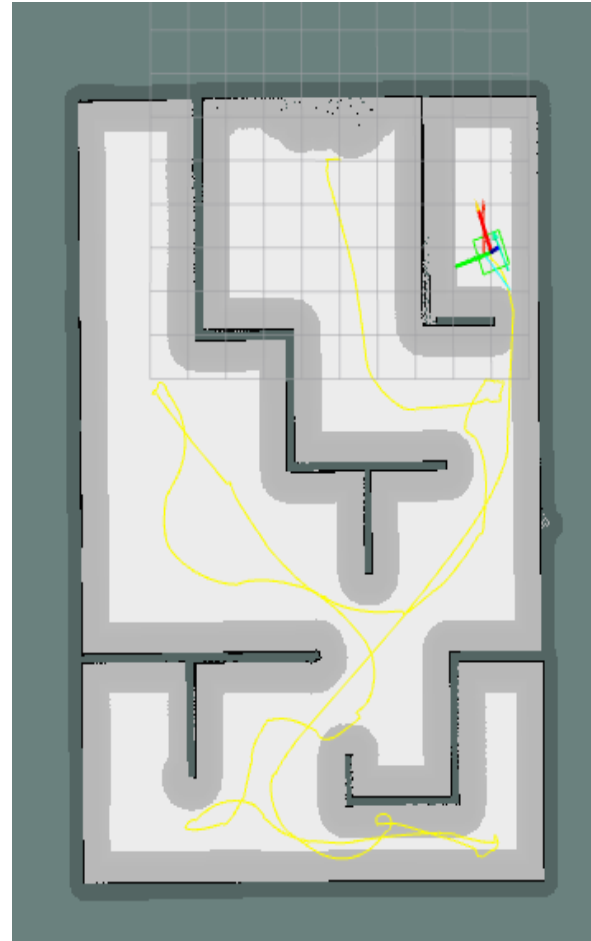


(b)

Figure 56: Robot's movement in the complex environment: (a) Gazebo result, (b) RViz result



(a)



(b)

Figure 57: Final map of the complex environment

Chapter 6 – Discussion:

During the testing phase, the Hokuyo Laser was tested to observe its ability to detect objects and giving out the distance between the sensor and the obstacles. Figure 36 shows the laser scanner with the environment and Figure 42 shows the visualisation of the scanner. While testing the laser, some points were not added to RViz due to the loss of some laser rays returning to the laser scanner. This loss is because of the reflective property of the object, thus, if the object's reflective coefficient lies within 70% to 100%, then, the object has a bright and smooth surface where it can reflect the laser ray back to the laser scanner. If the reflective coefficient lies below 70%, it means that the surface becomes darker and rougher where the ray could be deflected and not returning to the scanner (Colombo and Marana, 2010). However, since the laser was damaged, therefore, simulation of the laser was used to acquire the distance of the objects as shown in Figure 38 and Figure 43 displays the scanned area around the laser in Gazebo. Using the phase delay equation from hardware and software description, it prints out the ranges and it can be viewed from Figures 58.

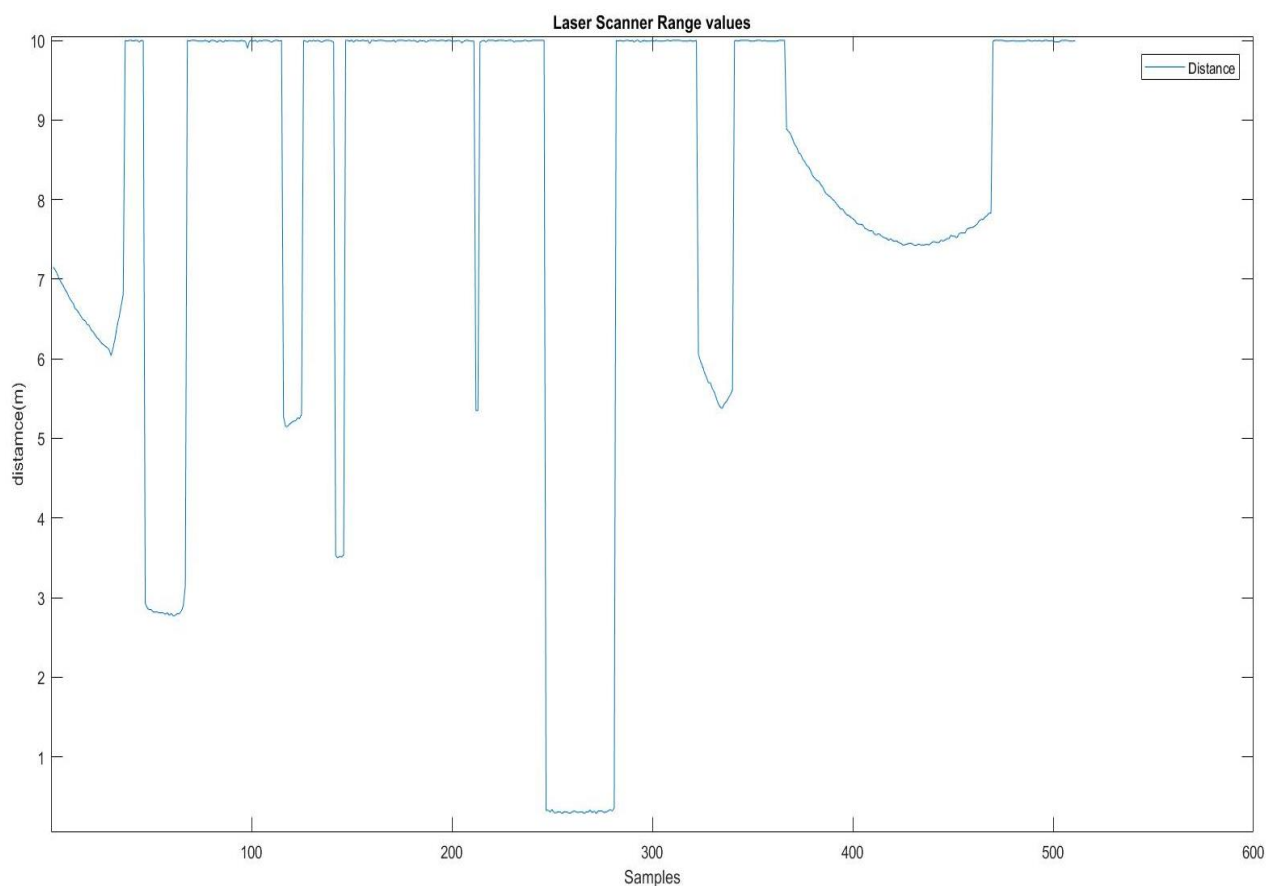


Figure 58: Range values acquired by the laser in Gazebo

From these samples, it is possible to determine the object from its distance values. The identity is shown in Table 7.

Table 7: Object Identities from the range values

Samples:	Object:	Ranges:
0 to 36	Brick Wall	Around 6 metres
47 to 67	Construction cone	Around 2 metres
116 to 125	Hollow brick	Around 5 metres
142 to 146	Coke can	Around 3 metres
212 to 213	Café Table	Around 5 metres
247 to 281	Beer can	Around 0.32 metres
324 to 341	Bookshelf	Around 5 metres
391 to 469	Grey wall	Around 6 metres

Furthermore, the encoders of the motors were also tested to see the odometry of the robot. The odometry was obtained by calculating the speed of the right and the left wheels using the encoders and these values were used in equations below:

$$\begin{aligned}
 dt &= \text{current time} - \text{last time} \\
 \Delta\text{Tick} &= \text{Tick}_{\text{New}} - \text{Tick}_{\text{Old}} \\
 T\text{Distance} &= \frac{d}{dt} (\Delta\text{Tick}) \\
 \text{Vel} &= \frac{T\text{Distance}}{\text{Time}_{\text{Diff}}}
 \end{aligned}$$

The first equation calculates the time duration between the recording of the recent and previous velocity command. The next equation function is to calculate the difference in the number of ticks the encoder recorded when the wheel moved, hence, it looks into the old total tick counts and it is subtracted with new total tick counts. The tick counts of the right or left wheels could be different where it shows the robot is turning or the same where the robot is moving forward. Furthermore, the distance can be found by differentiating the difference in the tick counts. Then, after acquiring the distance and tick counts the velocity values of right and left wheels can be attained. These values are used as messages to be sent to Raspberry Pi where it can make use of the values to obtain the odometry of the robot and the process is explained below:

$$\begin{aligned}
 \text{Left}_{\text{Distance}} &= dt * \text{Vel}_x \\
 \text{Right}_{\text{Distance}} &= dt * \text{Vel}_y
 \end{aligned}$$

Where Vel_x is the left wheel velocity and Vel_y is the right wheel velocity. It uses the same time equation to calculate the distance travelled by each wheel. After collecting the distance travelled, these values have been utilised to get the mean distance in which it will give the midpoint of the robot. The equations below are used to obtain the robot's midpoint:

$$mean_{xy} = \frac{Left_{Distance} + Right_{Distance}}{2}$$

$$mean_{th} = \left(\frac{Right_{Distance} - Left_{Distance}}{WheelSeparation} \right) * OdomTurnMultiplier$$

Where 'WheelSeparation' is the distance between the wheels and 'OdomTurnMultiplier' is a multiplier that adjusts the odometry calculations of the angular motion and it can be within the range of 0.95 to 1.05. Finally, using the mean values it is possible to obtain the odometry translation using the following equations:

$$x = mean_{xy} * \cos(\theta)$$

$$y = mean_{xy} * \sin(\theta)$$

$$\theta = mean_{Th}$$

Looking at Figure 44, it showed that the robot was shifting left and right while moving forward, this is due to the drifting of the wheels where the wheels continue to rotate while it should have stopped. Looking at Figure 46, it illustrated the expected behaviour of the robot turning around. Therefore, combining the encoder with the robot it showed that it is possible to know the location of the robot when it scans the terrain. Instead, the laser was malfunctioning; hence, it will be hard to test the robot in the lab because the Navigation_Stack would require the map of the terrain from the laser scanner, therefore, a simulation was used.

According to the Figures 48 and 57 in the results section, the robot managed to detect the walls and places a circular radius called inflation radius where it assigns a safe distance between the robot and the obstacle. The robot used the inflation radius to avoid the walls and managed to reach the assigned goal. Most interesting part is the complex environment where it showed on the map that there is a tight area for the robot to move. Figure 59 shows the area.



Figure 59: section of the complex environment

When the robot entered this section, it found difficult to navigate through due to the inflation radius tightening the path. However, due to fast response in path calculation, it managed to update the path in order to reach the goal inside the corridor and escape to move towards a new goal.

An investigation was conducted into the odometry of the robot in the simulation because of the deviations between the pose of the robot that is calculated by HectorSLAM and odometry of the robot in the simulation which is calculated by the differential drive control of the robot using the HectorSLAM gazebo plugin package. The investigation showed that when the simulation and HectorSLAM were initiated, there is a difference in the pose of the robot where the SLAM records the pose at -1 and the odometry at 0. Figure 60 shows the pose of the robot at the initial position.

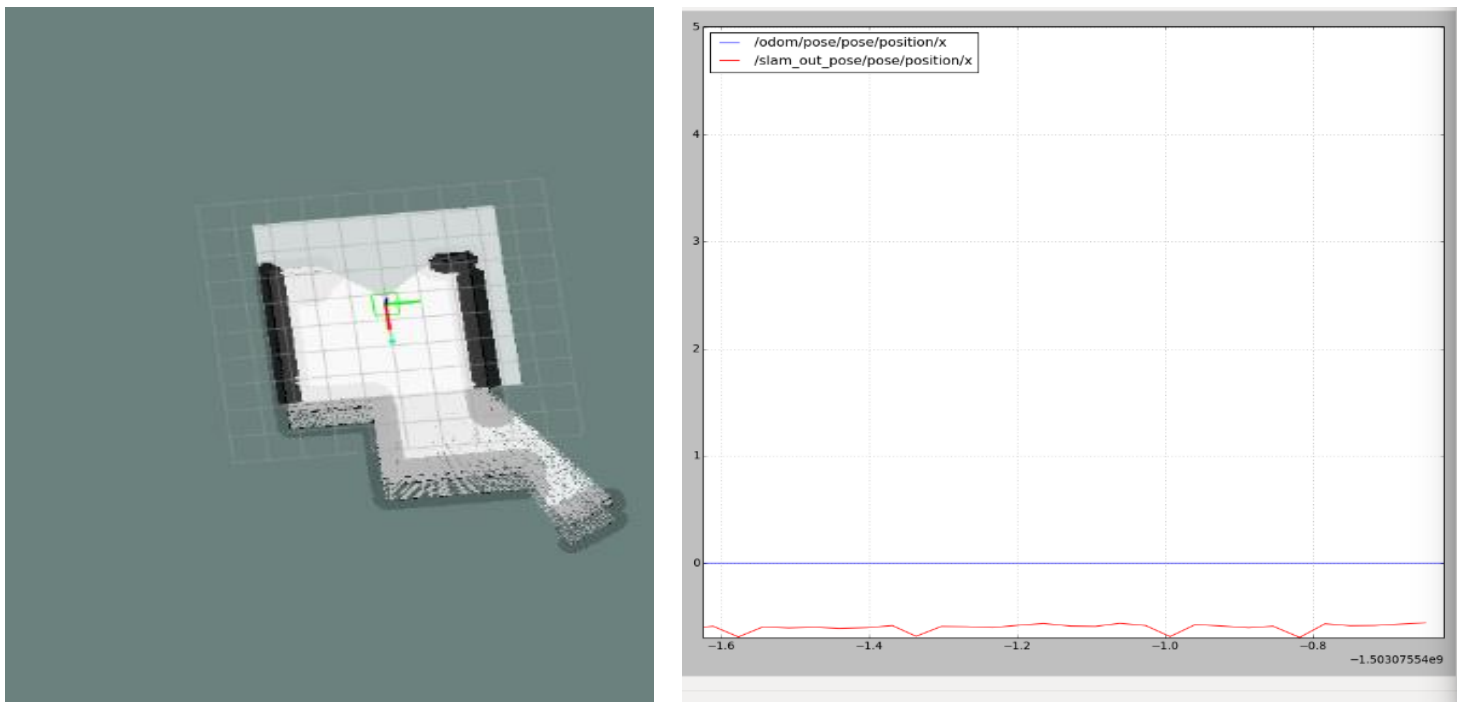


Figure 60: Odom vs. SLAM robot Pose when the robot is at initial position

In addition, when the robot starts moving forward the pose from SLAM and odometry deviate and the difference is noticeable in RViz and in the plot which can be shown in Figure 61.

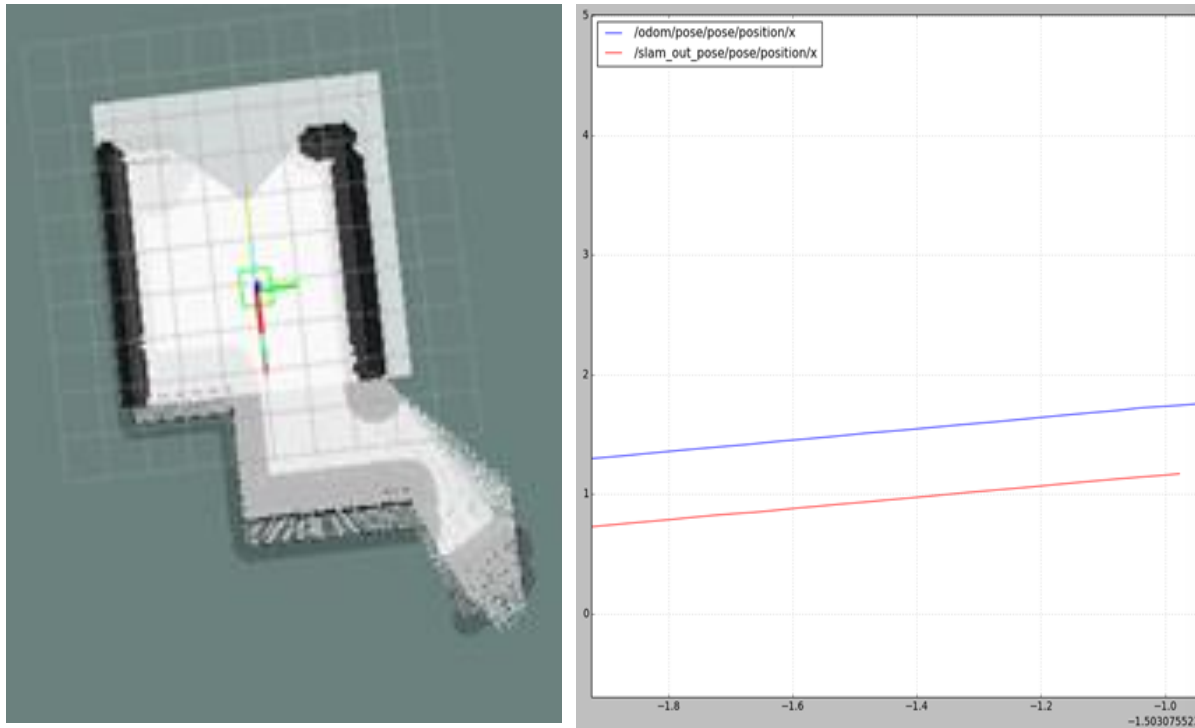


Figure 61: SLAM vs. Odometry pose when the robot is moving forward

Also, the deviation becomes larger when the robot turns around and Figure 62 shows the deviation when the robot is turning left.

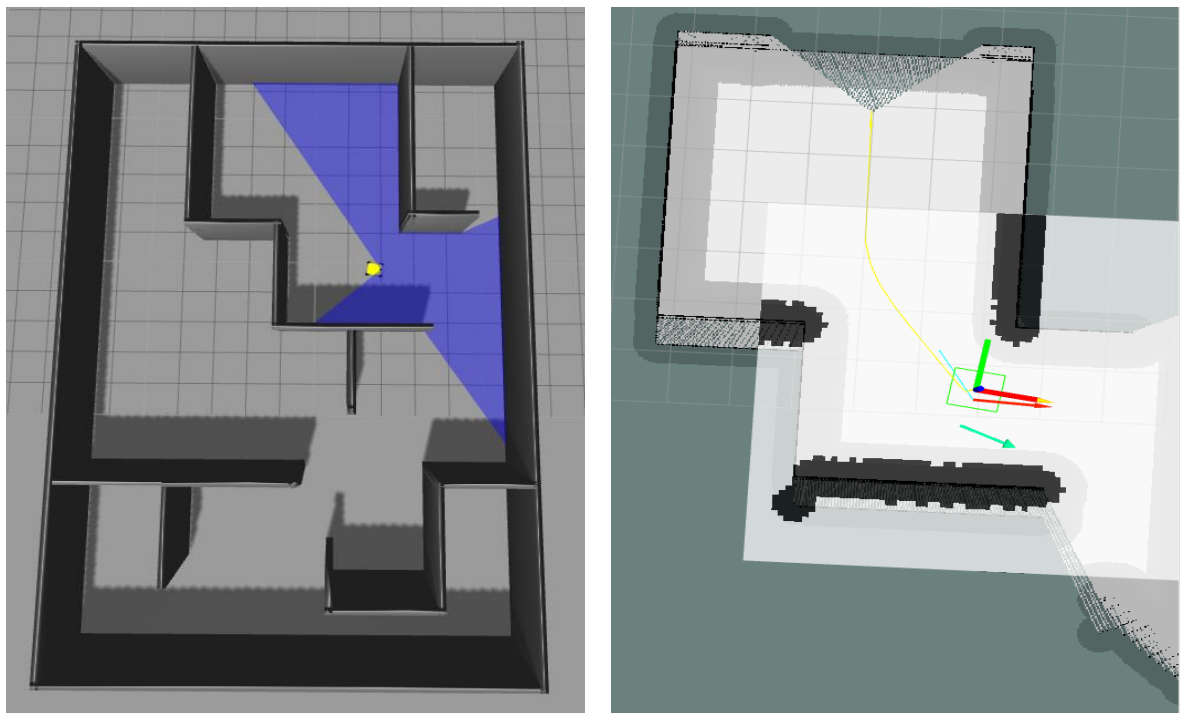


Figure 62: SLAM vs. Odometry when robot is turning left

This problem arises due to data association where the robot might scan an area, but the robot might think that the scanned area has shifted, for example, forward, but in real life it didn't, therefore, in the map it shifts the pose of the robot with the intention of adjusting to the shift and this cause the two layers of previous and recent scanned map. This effect can be shown in Figure 63.

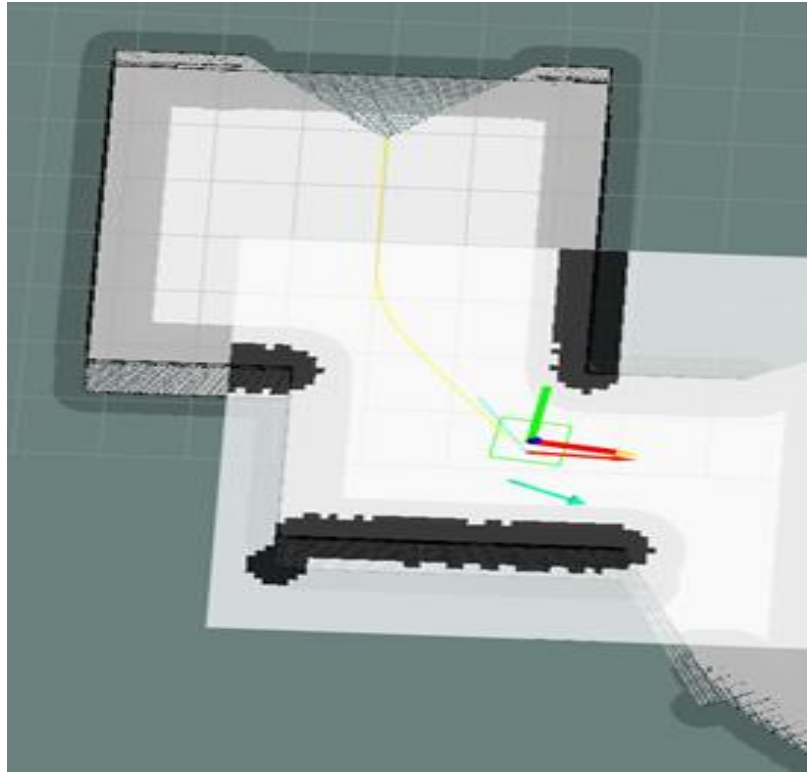


Figure 63: Shifted map positions in HECTORSLAM

On the other hand, this problem can be solved if the encoder was used as the source of odometry data so that in RViz it will show that the robot is at the initial position where it will always stay at zero axes or stays constant at certain axis if the robot stops at a certain position. This process can be done using information fusion in HECTORSLAM between the laser scan and odometry data from the encoders. Judging from these results, it indicated that all the basic objectives were achieved.

In terms of advanced objectives, it looks into power management for motors and positional error of the robot. To achieve these objectives, it needs a control system to control the amount of voltage going towards the motors and reduce the positional error between the path taken by the robot and global path plan. The system used is PID control where it can control Pulse Width Modulation (PWM) going from Nexus duino to motors and it turns out, by doing this

it will also assist in reducing the positional error of the robot. Figure 64 shows the general PID structure.

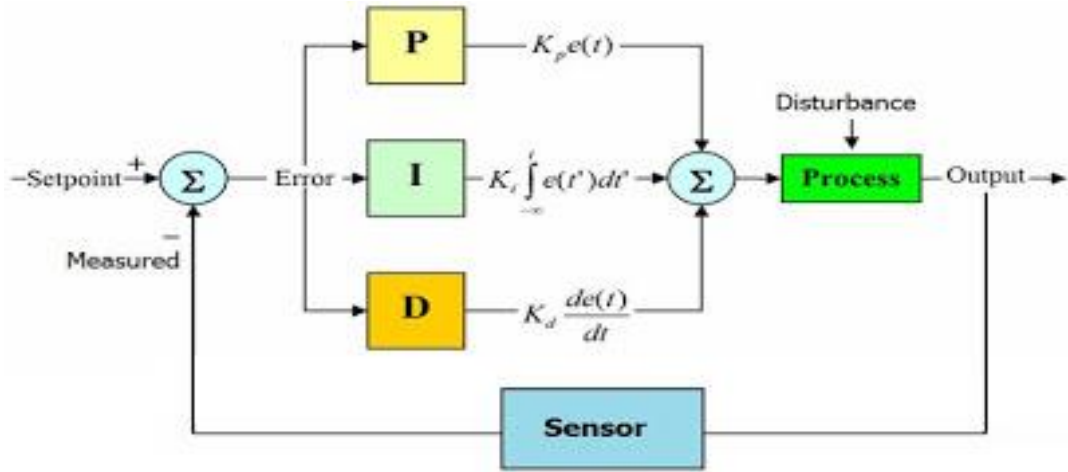


Figure 64: general PID controller (reproduced from (TILZOR, 2014))

Looking at Figure 64, the sensor used is the encoder so the error difference is calculated by subtracting the evaluated speed and the commanded speed to the motors. The equation below shows how the error is calculated:

$$Error = Eval_{speed} - Comm_{speed}$$

After obtaining the error value, it is used to acquire the integration of the error by summing the error every time. The following equation accumulates the error to be used in the integral part of the controller:

$$Accu_{Error} += Error$$

In addition, the derivative uses the previous error and the current error in order to obtain the derivative error. The equation below uses the delta-time to obtain the differential error:

$$Der_{Error} = \frac{Error - Prev_{Error}}{dt}$$

Where 'dt' is delta-time where it calculates the difference between the current time with previous time. Figures 65 and 66 looks into the debug message that provides the outcome of the motors when using the PID control by applying speed of 1ms^{-1} to the robot.

```

linear:
x: 1.0
y: 1.03623390198
z: 15.0
angular: und_plane
x: 1.0
y: 0.972791016102
z: 11.0
--- Camera Measure
linear:
x: 1.0
y: 1.03623390198
z: 15.0
angular:
x: 1.0
y: 0.972791016102
z: 11.0
--- Lights
linear:
x: 1.0
y: 1.00481116772
z: 14.0
angular:
x: 1.0
y: 1.00481116772
z: 12.0
--- Value
linear:
x: 1.0
y: 1.01854896545
z: 14.0
angular:
x: 1.0
y: 1.03623390198
z: 12.0
---
linear:
x: 1.0
y: 1.02918469906
z: 14.0
angular:
x: 1.0
y: 0.935513794422
z: 11.0
---
linear:
x: 1.0
y: 1.02918469906
z: 14.0
angular:
x: 1.0
y: 0.930495738983
z: 11.0
---
linear:
x: 1.0
y: 1.0080370903
z: 14.0
angular:
x: 1.0
y: 0.946820259094
z: 12.0
---
linear:
x: 1.0
y: 0.970162510872
z: 14.0
angular:
x: 1.0
y: 0.914724647999
z: 12.0
---
linear:
x: 1.0
y: 1.03623390198
z: 14.0
angular:
x: 1.0
y: 0.993938624859
z: 12.0

```

Figure 65: Velocity speeds from the motors (linear z=Left motor speed; angular z=Right motor speed)

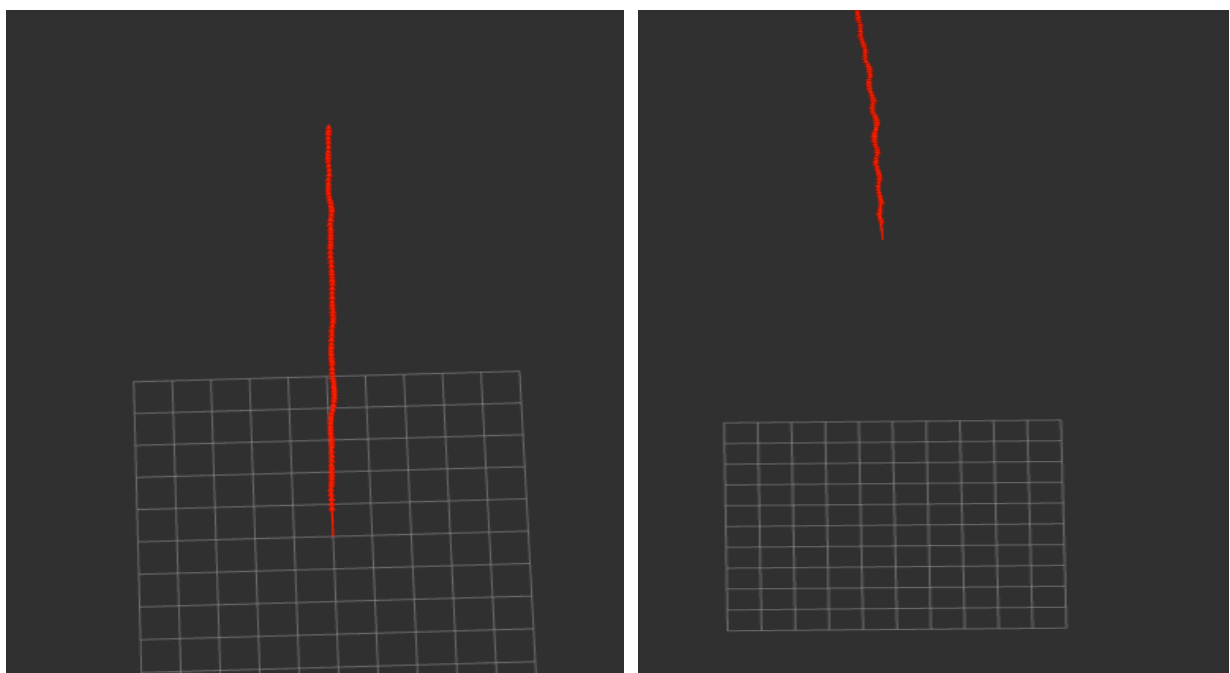


Figure 66: Robot movement with PID control

Observing Figures 65, The robot motor speed almost got stable, but it sometimes fluctuates in left motor between 15 to 14 and right motor between 12 to 11. Figure 66 showed the movement of the robot is almost in a straight line, but slightly deviated and this is due to the change in the speed between right and left motor speed. Judging from the performance of the robot with PID control, the results show potential that it can control the amount of voltage going to the motors and manage the positional error between the calculated path from path planning algorithms and path taken by the robot.

Chapter 7 – Conclusion and Further work:

7.1 Conclusion:

All the basic and advanced objectives were achieved since the robot was able to map the terrain, detect walls of the environment and avoid obstacles in Gazebo. In addition, the robot almost managed to control the power going towards the motors by controlling the amplitude of the PWM and it almost regulates the positional error of the robot. The advanced objectives were worked out using PID control, in particular, PD controller because it improves the response by decreasing the offset error, reduce the settling and rise time of the system and diminish the high-frequency noise going towards the system. However, these performances were done in simulation and while the robot is not in contact with the ground because the laser sensor was not functioning. Therefore, utilising a functioning laser scanner would be possible to scan the terrain and transmit the ranges and map to a desktop computer. Additionally, acquiring the map would help the Navigation_Stack to send goal coordinates with help of move_base which will set velocity commands to the Nexus robot to be able to reach the goal by using the path planning algorithm. In addition, further gain calibration for PID control is needed since there is a small difference in the speeds in right and left motors and by finding an appropriate gain this will solve the power difference and positional error of the robot.

7.2 Further work:

These tests were performed on gazebo simulation because the laser was damaged during the testing phase which makes it hard to map and navigate the terrain; hence, a simulation model with laser scanner was used to observe the behaviour of the robot. However, the model did not include some extra features such as the PID control and mecanum wheels to robot model. Therefore, the movement would be different because the wheels contain rollers, which are tilted at 45-degree angle. This will cause the wheel to move based on the direction of the rollers and the speed of the motors. Moreover, in the Navigation_Stack the parameters governing the speed limit assigned in base_local_planner was only fitted to the robot model in the simulation where in reality the robot struggled to move using these parameter values because the acceleration multiplier is low where the multiplier is responsible for increasing the speed values so that it can fit for the Nexus robot.

Therefore, to overcome these challenges, adjust the acceleration multiplier in Nexus robot code by increasing the multiplier value and further gain calibrations are needed in PID control to obtain an acceptable performance. These calibrations can be done by increasing the proportional and the derivative gain to obtain equal velocities in both motors and straight-line motion.

References:

- [1] 12V 8100 RPM DC Motor (no date) *RobotShop*. Available at: <http://www.robotshop.com/uk/12v-120-rpm-dc-motor-641.html> (Accessed: 17 August 2017).
- [2] 4WD 100mm Mecanum Wheel Robot Kit (no date) *Active Robots*. Available at: <http://www.active-robots.com/4wd-mecanum-wheel-robot-kit> (Accessed: 17 August 2017).
- [3] Adarsh, S. *et al.* (2016) ‘Performance comparison of Infrared and Ultrasonic sensors for obstacles of different materials in vehicle/ robot navigation applications’, *IOP Conference Series: Materials Science and Engineering*, 149, p. 12141. doi: 10.1088/1757-899X/149/1/012141.
- [4] Aqel, M. O. A. *et al.* (2016) ‘Review of visual odometry: types, approaches, challenges, and applications’, *SpringerPlus*. Springer International Publishing, 5(1), p. 1897. doi: 10.1186/s40064-016-3573-7.
- [5] Aulinas, J. *et al.* (2008) ‘The SLAM problem: A survey’, *Frontiers in Artificial Intelligence and Applications*, 184(1), pp. 363–371. doi: 10.3233/978-1-58603-925-7-363.
- [6] Bailey, T., Nieto, J. and Nebot, E. (2006) ‘Consistency of the FastSLAM algorithm’, in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 424–429. doi: 10.1109/ROBOT.2006.1641748.
- [7] Berardinis, L. (2004) *Review of quadrature-encoder signals.pdf*, *MachineDesign*. Available at: <http://www.machinedesign.com/technologies/review-quadrature-encoder-signals> (Accessed: 23 August 2017).
- [8] Borenstein, J. and Feng, L. (1996) ‘Measurement and Correction of Systematic Odometry Errors in Mobile Robots’, *IEEE Transactions on Robotics and Automation*, 12(6), pp. 845–857. doi: 10.1109/70.544768.
- [9] Bruno Steux, O. E. H. (2010) ‘CoreSLAM: a SLAM Algorithm in less than 200 lines of C code’, *International Conference on Control, Automation, Robotics and Vision (ICARCV)*, 1(July), pp. 1975–1979. doi: 10.1109/ICARCV.2010.5707402.
- [10] Carlone, L. and Aragues, R. (2011) ‘A linear approximation for graph-based simultaneous localization and mapping’, *Robotics: Science and ...*, p. 8. Available at: http://books.google.com/books?hl=en&lr=&id=Ziy81kH3KfUC&oi=fnd&pg=PA41&dq=A+Linear+Approximation+for+Graph-based+Simultaneous+Localization+and+Mapping&ots=ZZ9rsV0o5_&sig=GSbzACjVmG4tYIz0MIezcRW4c3g%5Cnhttp://books.google.com/books?hl=en&lr=&id=Ziy81kH3KfU.
- [11] Carlson, C., Gerdes, J. and Powell, J. (2004) ‘Error sources when land vehicle dead reckoning with differential wheelspeeds’, ... *-Los Angeles and ...*, (February), pp. 1–28. Available at: <http://www.crcarlson.com/Academia/Research/ION2004.pdf>.
- [12] Chen, Z., Samarabandu, J. and Rodrigo, R. (2007) ‘Recent advances in simultaneous localization and map-building using computer vision’, *Adv. Robot.*, 21(3–4), pp. 233–265. doi: 10.1163/156855307780132081.
- [13] Chong, T. J. *et al.* (2015) ‘Sensor Technologies and Simultaneous Localization and Mapping (SLAM)’, in *Procedia Computer Science*, pp. 174–179. doi: 10.1016/j.procs.2015.12.336.
- [14] Christina Lee; pbworks (2014) *Laser Rangefinder, Mechatronics*. Available at: [http://mymechatronics.pbworks.com/w/page/74907566/Laser Rangefinder](http://mymechatronics.pbworks.com/w/page/74907566/Laser%20Rangefinder) (Accessed: 17 August 2017).
- [15] Colombo, L. and Marana, B. (2010) ‘HOW IT WORKS AND WHAT IT

- DOES Terrestrial Laser Scanning’, *GIM International*, pp. 1–3. Available at: <http://www.gim-international.com/content/article/terrestrial-laser-scanning-2>.
- [16] Discant, A. *et al.* (2007) ‘Sensors for obstacle detection - A survey’, *ISSE 2007 - 30th International Spring Seminar on Electronics Technology 2007; Conference Proceedings: Emerging Technologies for Electronics Packaging*, pp. 100–105. doi: 10.1109/ISSE.2007.4432828.
 - [17] Dudek, G. and Jenkin, M. (2008) ‘Inertial Sensors, GPS, and Odometry’, *Robotics*, 11(10), pp. 477–490. doi: 10.1049/ep.1965.0279.
 - [18] Grisetti, G., Stachniss, C. and Burgard, W. (2007) ‘Improved techniques for grid mapping with Rao-Blackwellized particle filters’, *IEEE Transactions on Robotics*, 23(1), pp. 34–46. doi: 10.1109/TRO.2006.889486.
 - [19] Gross, R. (2017) ‘Planning (Week 9)’, (Week 9).
 - [20] Hokuyo UBG-04LX-F01 (Rapid URG) Scanning Laser Rangefinder (no date) *RobotShop*. Available at: <http://www.robotshop.com/uk/hokuyo-ubg-04lx-f01-scanning-laser-rangefinder-eu.html> (Accessed: 17 August 2017).
 - [21] Kohlbrecher, S. *et al.* (2011) ‘A Flexible and Scalable SLAM System with Full 3D Motion Estimation’, *9th IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2011*, pp. 155–160. doi: 10.1109/SSRR.2011.6106777.
 - [22] Konolige, K. *et al.* (2010) ‘Efficient sparse pose adjustment for 2D mapping’, in *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pp. 22–29. doi: 10.1109/IROS.2010.5649043.
 - [23] Leonard, J. J. and Durrant-Whyte, H. F. (1991) ‘Simultaneous map building and localization for an autonomous mobile robot’, *Intelligent Robots and Systems ’91. Intelligence for Mechanical Systems, Proceedings IROS ’91. IEEE/RSJ International Workshop on*, (91), pp. 1442–1447 vol.3. doi: 10.1109/IROS.1991.174711.
 - [24] *Localize with a Hokuyo laser range finder* (no date) *Generation robots*. Available at: <https://www.generationrobots.com/en/content/52-localize-with-a-hokuyo-laser-range-finder> (Accessed: 17 August 2017).
 - [25] Madhavan, R., Fregene, K. and Parker, L. E. (2004) ‘Distributed cooperative outdoor multirobot localization and mapping’, *Autonomous Robots*, 17(1), pp. 23–39. doi: 10.1023/B:AURO.0000032936.24187.41.
 - [26] Mohamed Kassim, A. *et al.* (2016) ‘Conceptual design and implementation of electronic spectacle based obstacle detection for visually impaired persons’, *Journal of Advanced Mechanical Design, Systems and Manufacturing*, 10(7). doi: 10.1299/jamdsm.2016jamdsm0094.
 - [27] National Research Council (2005) *Autonomous Vehicles in Support of Naval Operations*. Washington, D.C.: National Academies Press. doi: 10.17226/11379.
 - [28] Nexus Automation Limited (no date) *NEXUS ROBOT: Robot Kits Users Manual, Nexus Robot*. Available at: www.nexusrobot.com (Accessed: 17 August 2017).
 - [29] On-Road Automated Driving (ORAD) committee (2014) ‘AUTOMATED DRIVING’, *SAE international*, p. 2. Available at: https://www.sae.org/misc/pdfs/automated_driving.pdf.
 - [30] Osian Haines (2016) *An Introduction to Simultaneous Localisation and Mapping, Kudan Computer Vision*. Available at: <https://www.kudan.eu/kudan-news/an-introduction-to-slam/> (Accessed: 13 August 2017).
 - [31] Panich, S. (2010) ‘Comparison of distance measurement between stereo vision and ultrasonic sensor’, *Journal of Computer Science*, 6(10), pp. 1108–1110. doi: 10.3844/jcssp.2010.1108.1110.
 - [32] Park, J., Lee, J.-H. and Son, S. H. (2016) ‘A Survey of Obstacle Detection

- Using Vision Sensor for Autonomous Vehicles’, *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 264–264. doi: 10.1109/RTCSA.2016.54.
- [33] RASPBERRY PI 3 MODEL B (no date) RASPBERRY PI FOUNDATION. Available at: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/> (Accessed: 17 August 2017).
- [34] Reese, H. (2016) *Autonomous driving levels 0 to 5: Understanding the differences*, TechRepublic. Available at: <http://www.techrepublic.com/article/autonomous-driving-levels-0-to-5-understanding-the-differences/> (Accessed: 24 August 2017).
- [35] Renishaw (2016) ‘Optical encoders and LiDAR scanning’, (September). Available at: <http://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244>.
- [36] Riisgaard, S. and Blas, M. R. (2004) ‘SLAM for Dummies’, *A Tutorial Approach to Simultaneous Localization and Mapping*, 22(June), pp. 1–127. doi: 10.1017/S0025315400002526.
- [37] ROS (no date) *Setup and Configuration of the Navigation Stack on a Robot, ROS*. Available at: <http://wiki.ros.org/navigation/Tutorials/RobotSetup> (Accessed: 25 August 2017).
- [38] Santos, J. M., Portugal, D. and Rocha, R. P. (2013) ‘An evaluation of 2D SLAM techniques available in Robot Operating System’, *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*. doi: 10.1109/SSRR.2013.6719348.
- [39] Se, S., Lowe, D. G. and Little, J. J. (2005) ‘Vision-based global localization and mapping for mobile robots’, *IEEE Transactions on Robotics*, 21(3), pp. 364–375. doi: 10.1109/TRO.2004.839228.
- [40] Se, S., Lowe, D. and Little, J. (2001) ‘Local and global localization for mobile robots using visual landmarks’, *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*, 1, pp. 414–420. doi: 10.1109/IROS.2001.973392.
- [41] Se, S., Lowe, D. and Little, J. (2002) ‘Mobile Robot Localization and Mapping with Uncertainty using Scale-Invariant Visual Landmarks’, *The International Journal of Robotics Research*, 21(8), pp. 735–758. doi: 10.1177/027836402761412467.
- [42] Seidle, N. (2011) *Triple Axis Magnetometer - HMC5883L Breakout Quickstart Guide, SparkFun*. Available at: <https://www.sparkfun.com/tutorials/301> (Accessed: 25 August 2017).
- [43] Shrivastava, a K., Verma, a and Singh, S. P. (2010) ‘Distance Measurement of an Object or Obstacle by Ultrasound Sensors using P89C51RD2’, *International Journal of Computer Theory and Engineering*, 2(1), pp. 2–6.
- [44] Singer, P. W. (2009) ‘Military Robots and the Laws of War’, (23), pp. 25–45. Available at: http://www.thenewatlantis.com/docLib/20090203_TNA23Singer.pdf.
- [45] Skog, I. and Handel, P. (2007) ‘State-of-the art and future in-car navigation - a survey’, *{IEEE} Transactions on Intelligent Transportation Systems*, (March).
- [46] Smith, R. C. and Cheeseman, P. (1986) ‘On the Representation and Estimation of Spatial Uncertainty’, *The International Journal of Robotics Research*, 5(4), pp. 56–68. doi: 10.1177/027836498600500404.
- [47] Sniedovich, M. (2006) ‘Dijkstra’s algorithm revisited: The dynamic programming connexion’, in *Control and Cybernetics*, pp. 599–620.

- [48] Srikanth, M., Parhar, T. and Patravali, J. (2015) *Inventory Management Robot*. Vellore. Available at: <http://jaypatravali.github.io/pages/invento.html>.
- [49] TILZOR (2014) *Project 03- STM32F4xx PID controller - STM32F4 Discovery.pdf*, *STM32F4 Discovery*. Available at: <https://stm32f4-discovery.net/2014/11/project-03-stm32f4xx-pid-controller/> (Accessed: 25 August 2017).
- [50] *Vectoring robots with Omni or Mecanum wheels* (no date) *SuperDroid Robots*. Available at: https://www.superdroidrobots.com/images/customPages/mecanum_drive_wheels_vectoring_robot.jpg (Accessed: 17 August 2017).
- [51] Yan, M. (2016) 'Dijkstra' S Algorithm'. Available at: <http://math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf>.
- [52] Yuan, X. *et al.* (2017) 'AEKF-SLAM: A New Algorithm for Robotic Underwater Navigation', *Sensors*, 17(6), p. 1174. doi: 10.3390/s17051174.
- [53] Zamora, E. and Yu, W. (2013) 'Recent advances on simultaneous localization and mapping for mobile robots', *IETE Technical Review*, 30(6), pp. 490–496. doi: 10.4103/0256-4602.125671.

Appendices:

Arduino:

Using Arduino IDE, it is possible to program the Nexus robot where it can use ROS library to form a connection with Raspberry Pi. Raspberry Pi contains ROS system where it can communicate with the ROS master which is a desktop computer.

A. Nexus robot code:

This code is an Arduino code which is installed in the Nexus robot.

```
// Libraries:
```

```
#include <ros.h>
```

```
#include <geometry_msgs/Twist.h>
```

```
#include <Encoder.h>
```

```
// Pins:
```

```
//// PWM motor pins:
```

```
#define LMotPWMPin 9
```

```
#define RMotPWMPin 6
```

```
#define LMot2PWMPin 10
```

```
#define RMot2PWMPin 5
```

```
//// Dir motor pins:
```

```
#define LMotDIRPin 8
```

```
#define RMotDIRPin 7
```

```
#define LMot2DIRPin 11
```

```
#define RMot2DIRPin 4
```

```

//// Motor variables:

int LMotor = 1;
int RMotor = 0;

//// Encoder pins:

#define InterEncoderL 0
#define LEncoderA 2
#define LEncoderB 12

#define InterEncoderR 1
#define REncoderA 3
#define REncoderB 13

// Motor Parameters:

float WheelSeparation = 0.3;
float WheelDiameter = 0.1;
int TPR = 768;    // Encoder ticks per rotation
int AccParam = 3; // Acceleration multiplier
double Prev_T;
double Prev_Error;
double Prev_Input;
double Accu_Error;
double pidterm;
int Kp = 0.8;    // PID proportional control gain.
int Kd = 1.3;    // PID Derivative control gain

// sudo vars:
int OdomWait = 3;

```

```

int OdomCount = 0;

double WCS[2] = {0,0};

// ROS variables:
ros::NodeHandle nh;
//// ROS publisher
geometry_msgs::Twist odom_msg;
ros::Publisher Pub ("nexus_odom", &odom_msg);
geometry_msgs::Twist debug_msg;
ros::Publisher Debug ("debug", &debug_msg);
//ROS subscriber

void messageCb( const geometry_msgs::Twist& CVel){
    //geometry_msgs::Twist twist = twist_msg;

    double vel_x = CVel.linear.x;
    double vel_th = CVel.angular.z;

    double right_vel = 0.0;
    double left_vel = 0.0;

    // Movement:
    // turning
    if(vel_x == 0){
        right_vel = vel_th * WheelSeparation / 2.0;
        left_vel = (-1) * right_vel;
    }
    // forward / backward:
    else if(vel_th == 0){
        left_vel = right_vel = vel_x;
    }
    // moving doing arcs:
    else{

```

```

    left_vel = vel_x - vel_th * WheelSeparation / 2.0;
    right_vel = vel_x + vel_th * WheelSeparation / 2.0;
}

//write new command speeds to global vars
WCS[0] = left_vel;
WCS[1] = right_vel;
}

ros::Subscriber<geometry_msgs::Twist> Sub("cmd_vel", &messageCb );

// Motor variables:
#define CW 1
#define CCW 2

int inApin[4] = {LMotDIRPin, RMotDIRPin,LMot2DIRPin, RMot2DIRPin}; // INDIR: Clockwise input
int pwmpin[4] = {LMotPWMPin, RMotPWMPin,LMot2PWMPin, RMot2PWMPin}; // PWM input
int MotorNum[2] = {LMotor, RMotor};

//encoder variables:

Encoder LEncoder(LEncoderA, LEncoderB);
Encoder REncoder(REncoderA, REncoderB);

long EncoderVal[2] = {0,0};
double DDis[2] = {0,0};
long Time[2] = {0,0};

//debug variables:

double Vels[2] = {0,0};
int CVEL[2]={0,0};
int Mspeeds[2] = {0,0};

```

```

// Main Program:

//// Setup Code:
void setup()
{
    nh.getHardware()->setBaud(19200);
    nh.initNode();
    nh.advertise(Pub);
    nh.advertise(Debug);
    nh.subscribe(Sub);

    nh.getParam("/serial_node/WheelSeparation", &WheelSeparation,1);
    nh.getParam("/serial_node/WheelDiameter", &WheelDiameter,1);
    nh.getParam("/serial_node/AccParam", &AccParam,1);

    // Pin mode for motors
    pinMode(LMotDIRPin,OUTPUT);
    pinMode(RMotDIRPin,OUTPUT);
    pinMode(LMot2DIRPin,OUTPUT);
    pinMode(RMot2DIRPin,OUTPUT);

}

//// Main code:
void loop(){

    nh.spinOnce();

    //first couple of times dont publish odom
    if(OdomCount > OdomWait){

```

```

        odom_msg.linear.x = Vels[0];
        odom_msg.linear.y = Vels[1];
        Pub.publish(&odom_msg);
    }
    else{OdomCount++;}

//// Motor variables:
//int LMotor = 1;
//int RMotor = 0;

// Assigning values to debugger message:
debug_msg.linear.x = WCS[0];
debug_msg.linear.y = Vels[0];
debug_msg.linear.z = Mspeeds[0];
debug_msg.angular.x = WCS[1];
debug_msg.angular.y = Vels[1];
debug_msg.angular.z= Mspeeds[1];

// Publishing the debug message:
Debug.publish(&debug_msg);

MotorWrite();        //Takes WCS and corrects speed of motors with encoders

delay(3);
}

//// Encoder function code:
// Motor write speed variables(in motor units):
double MWS[2]= {0,0};

```



```

double CorrectedSpeed(int M, double CVel)
{
    //if fist time in program return 0 and init time vars
    if(Time[0]==0 && Time[1] == 0){
        Time[0] = millis();
        Time[1] = millis();
        return 0;
    }

    //read encoder ticks
    if(M == LMotor){
        EncoderVal[0] = LEncoder.read();
        LEncoder.write(0);
    }
    if(M == RMotor){
        EncoderVal[1] = REncoder.read();
        REncoder.write(0);
    }

    //diferencial of time in seconds
    long T = millis();
    int DTime = T-Time[M];
    Time[M] = T;

    //diferential of distance in meters
    DDis[M] = TicksToMeters(EncoderVal[M]);

    //calculate short term measured velocity
    double EVel = (DDis[M]/DTime)*1000;

```

```

//save to publish to /ard_odom
Vels[M] = EVel;

EVel = abs(EVel);
CVel = abs(CVel);

// Differential in calculation Time:
unsigned long T_Now = millis();
double Delta_T = double (T_Now - Prev_T);
Prev_T = T_Now;

// Propotional controller:
double dif = EVel - CVel;

// Differential controller:
double Der_Error = (dif - Prev_Error) / Delta_T;

// PID Term:
double pidterm = (Kp * dif) + (Kd * Der_Error); // PD controller

// PWM output to motors
if(MWS[M]<60 && MWS[M]>=0)
{
MWS[M]=MWS[M]-pidterm;
}
if(MWS[M]>60)
{
MWS[M]=59;
}
if(MWS[M]<0)
{

```

```

    MWS[M]=0;
}

    if(CVel == 0)
{
    MWS[M] = 0;
}

    //DEBUG
    CVEL[M] = MWS[M];

    return MWS[M];

}

double TicksToMeters(int Ticks){
    return (Ticks*3.14*WheelDiameter)/TPR;
}

//// Motor function code:
void MotorWrite()
{
    // For the right motor
    if (WCS[0] > 0){
        digitalWrite(inApin[0], HIGH);
        digitalWrite(inApin[2], HIGH);
    }else{
        digitalWrite(inApin[0], LOW);
        digitalWrite(inApin[2], LOW);
    }

    double MSpeed0 = CorrectedSpeed(0, WCS[0]);

```

```

Mspeeds[0]=MSpeed0;
analogWrite(pwmpin[0], int(MSpeed0));
analogWrite(pwmpin[2], int(MSpeed0));

// For the left motor
if (WCS[1] > 0){
    digitalWrite(inApin[1], HIGH);
    digitalWrite(inApin[3], HIGH);
}else{
    digitalWrite(inApin[1], LOW);
    digitalWrite(inApin[3], LOW);
}

double MSPEED1 = CorrectedSpeed(1, WCS[1]);
Mspeeds[1]=MSPEED1;
analogWrite(pwmpin[1], int(MSPEED1));
analogWrite(pwmpin[3], int(MSPEED1));
}

```

ROS:

In ROS, it is possible to install HectorSLAM by obtaining the Github file from *Stefan Kohlbrecher* and the link is below:

https://github.com/tu-darmstadt-ros-pkg/hector_slam.git

The following codes are made and implemented in ROS master by following several tutorials in ROS and ROS forum.

A. Robot odometry:

Robot odometry is a C++ code where it is made in ROS as it receives the velocities of each wheel and does some calculations to obtain the odometry information of the Nexus robot.

```

#include <ros/ros.h>
#include <std_msgs/String.h>
#include <sensor_msgs/JointState.h>
#include <tf/transform_broadcaster.h>
#include <nav_msgs/Odometry.h>

```

```

#include <geometry_msgs/PoseStamped.h>
#include <geometry_msgs/Twist.h>

double vx, vy, vth;
double OdomTurnMultiplier = 0.95;
double dist_l;
double dist_r;
double WheelSeparation = 0.3;

void OdomCallback(const geometry_msgs::Twist::ConstPtr& msg)
{
    vx = msg->linear.x;
    vy = msg->linear.y;
}

int main(int argc, char** argv){
    ros::init(argc, argv, "Robot_Odom");

    ros::NodeHandle n;
    ros::Publisher odom_pub = n.advertise<nav_msgs::Odometry>("odom", 1);
    ros::Subscriber sub = n.subscribe("nexus_odom", 1, OdomCallback);
    tf::TransformBroadcaster odom_broadcaster;

    double x = 0.0;
    double y = 0.0;
    double th = 0.0;

    ros::Time current_time, last_time;
    current_time = ros::Time::now();
    last_time = ros::Time::now();

    ros::Rate r(20.0);
    while(n.ok())
    {
        ros::spinOnce();          // check for incoming messages

        current_time = ros::Time::now();

        //compute odometry in a typical way given the velocities of the robot
        double dt = (current_time - last_time).toSec();

        //Initialise distance:
        dist_l = dt*vx;
        dist_r = dt*vy;

        // Mean distance:
        double mean_xy = (dist_l + dist_r) / 2;
        double mean_th = (((dist_r - dist_l) / WheelSeparation)) * OdomTurnMultiplier;
    }
}

```

```

double delta_x = mean_xy * cosf(th);
double delta_y = mean_xy * sinf(th);
double delta_th = mean_th;

x += delta_x;
y += delta_y;
th += delta_th;

//Current velocity:
double spd = mean_xy / dt;
double ang_spd = mean_th / dt;

//since all odometry is 6DOF we'll need a quaternion created from yaw
geometry_msgs::Quaternion odom_quat = tf::createQuaternionMsgFromYaw(th);

//first, we'll publish the transform over tf
geometry_msgs::TransformStamped odom_trans;
odom_trans.header.stamp = current_time;
odom_trans.header.frame_id = "odom";
odom_trans.child_frame_id = "base_link";

odom_trans.transform.translation.x = x;
odom_trans.transform.translation.y = y;
odom_trans.transform.translation.z = 0.0;
odom_trans.transform.rotation = odom_quat;

//send the transform
odom_broadcaster.sendTransform(odom_trans);

//next, we'll publish the odometry message over ROS
nav_msgs::Odometry odom;
odom.header.stamp = current_time;
odom.header.frame_id = "odom";

//set the position
odom.pose.pose.position.x = x;
odom.pose.pose.position.y = y;
odom.pose.pose.position.z = 0.0;
odom.pose.pose.orientation = odom_quat;

//set the velocity
odom.child_frame_id = "base_link";
odom.twist.twist.linear.x = spd;
odom.twist.twist.linear.y = 0.0;
odom.twist.twist.linear.z = 0.0;
odom.twist.twist.angular.x = 0.0;
odom.twist.twist.angular.y = 0.0;
odom.twist.twist.angular.z = ang_spd;

//publish the message

```

```

odom_pub.publish(odom);

last_time = current_time;
r.sleep();
}
ros::spin();
}

```

B. Launch files:

The following launch files are made in ROS as it launches the Gazebo simulation with the environments.

a) Configuration_Gazebo.launch:

```

<launch>

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

<!-- start up wg world -->

<include file="$(find nexus_robot_2D_nav)/launch/gazebo_world.launch">

</include>

<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher" >

</node>

<!-- start robot state publisher -->

<node pkg="robot_state_publisher" type="robot_state_publisher" name="state_publisher"
output="screen" >

<param name="publish_frequency" type="double" value="50.0" />

<node name="state_publisher" pkg="nexus_robot_model" type="state_publisher" />

</node>

<node name="spawn_robot" pkg="gazebo" type="spawn_model" args="-urdf -param
robot_description -z 0.1 -model robot_model" respawn="false" output="screen" />

<!-- <node name="rviz" pkg="rviz" type="rviz"
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/navigation.vcg" /> -->

```



```
<!-- <node pkg="tf" type="static_transform_publisher" name="base_to_odom_broadcaster" args="0
0 0 0 0 /odom_frame /odom 100" /> -->
```

```
<node pkg="tf" type="static_transform_publisher" name="odom_to_laser_broadcaster" args="0 0
0 0 0 /base_frame /hokuyo::laser 100" />
```

```
<!-- <node pkg="tf" type="static_transform_publisher" name="frame_to_chassis_broadcaster"
args="0 0 0 0 0 /base_frame /chassis 100" /> -->
```

```
<!-- <node pkg="tf" type="static_transform_publisher" name="map_to_frame_broadcaster"
args="0 0 0 0 0 /map /base_frame 100" /> -->
```

```
</launch>
```

b) Gazebo_World.launch:

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

```
<launch>
```

```
<arg name="args" default=""/>
```

```
<include file="$(find nexus_robot_2D_nav)/launch/start_gazebo.launch">
```

```
<arg name="world"
value="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/Worlds/nexus_robot_lab_arena_final_
6.world"/>
```

```
<arg name="args" default="$(arg args)"/>
```

```
</include>
```

```
</launch>
```

c) Move_base.launch:

```
<launch>
```

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

```
<!-- Running Move Base -->
```

```

<node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/costmap_common_params.yaml"
command="load" ns="global_costmap" />

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/costmap_common_params.yaml"
command="load" ns="local_costmap" />

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/local_costmap_params.yaml"
command="load" />

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/global_costmap_params.yaml"
command="load" />

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/costmap_params.yaml"
command="load" />

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/base_local_planner_params.yaml"
" command="load" />

  <param name="base_global_planner" value="navfn/NavfnROS"/>

  <rosparam
file="/home/maurice/catkin_ws/src/nexus_robot_2D_nav/launch/costmap_exploration.yaml"
command="load" />

</node>
</launch>

```

C. Cost-map parameters:

The Navigation_Stack uses the .yaml files where it prints out the inflation radius in RViz, contains speed limit parameters for velocity commands and activates the path planning algorithms.

```

a) Base_Local_Planner.yaml:
recovery_behavior_enabled: false

TrajectoryPlannerROS:
  max_vel_x: 0.5
  min_vel_x: 0.1
  max_vel_theta: 0.8

```

```

min_in_place_vel_theta: 0.4
escape_vel: -0.4

acc_lim_theta: 0.8
acc_lim_x: 0.8
acc_lim_y: 0.8

yaw_goal_tolerance: 0.1
xy_goal_tolerance: 0.2
latch_xy_goal_tolerance: false # do not rotate on finding goal

pdist_scale: 0.5
gdist_scale: 0.5

occdist_scale: 2.0

publish_cost_grid_pc: true

holonomic_robot: false
meter_scoring: true

# Global Planner Parameters
BaseGlobalPlanner:
  allow_unknown: false
  use_dijkstra: false #Use A* instead, true to use Dijkstra
  use_quadratic: true
  use_grid_path: false
  old_navfn_behavior: false

  b) Costmap_common_params.yaml:

  obstacle_range: 2.5
  raytrace_range: 3.0

  footprint: [[-0.4,-0.35],[-0.4,0.35], [0.4, 0.35], [0.4,-0.35]]
  #robot_radius: ir_of_robot

  robot_radius: 1.0 # distance a circular robot should be clear of the
  obstacle
  inflation_radius: 0.55

  observation_sources: laser_scan_sensor
  # laser_scan_sensor: {sensor_frame: laser, data_type: LaserScan,
  topic: /scan, marking: true, clearing: true}
  # marking - add obstacle information to cost map
  # clearing - clear obstacle information to cost map

  c) Costmap_params.yaml:

  global_costmap:
    global_frame: /map
    robot_base_frame: /base_frame

  plugins:
    - {name: static, type: "costmap_2d::StaticLayer"}
    - {name: inflation, type: "costmap_2d::InflationLayer"}

```

```

# - {name: explore_boundary, type:
"frontier_exploration::BoundedExploreLayer"}

static:
  unknown_cost_value: -1
  map_topic: /map

inflation:
  inflation_radius: 0.7
  cost_scaling_factor: 0.2

local_costmap:
  global_frame: /map
  robot_base_frame: /base_frame
  robot_radius: 2.0

plugins:
- {name: static, type: "costmap_2d::StaticLayer"}
- {name: inflation, type: "costmap_2d::InflationLayer"}
- {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
# - {name: explore_boundary, type:
"frontier_exploration::BoundedExploreLayer"}

static:
  unknown_cost_value: -1
  map_topic: /map

inflation:
  inflation_radius: 0.4
  cost_scaling_factor: 0.2

```

d) Global_costmap_params.yaml:

```

global_costmap:
  global_frame: /map
  robot_base_frame: /base_frame
  update_frequency: 0.4
  publish_frequency: 0.4
  static_map: false
  width: 100.0
  height: 100.0

transform_tolerance: 20
origin_x: -10.0
origin_y: -10.0

```

e) Local_costmap_params.yaml:

```

local_costmap:
  global_frame: /map
  robot_base_frame: /base_frame
  update_frequency: 0.4
  publish_frequency: 0.4
  static_map: false
  rolling_window: true

```

```
width: 7.0
height: 7.0
resolution: 0.1
transform_tolerance: 10

plugins:
- {name: static_layer,    type: "costmap_2d::StaticLayer"}
- {name: obstacle_layer, type: "costmap_2d::ObstacleLayer"}
```