
# ASHESI UNIVERSITY

**AUTONOMOUS DELIVERY ROBOT**

**CAPSTONE**

Mechanical Engineering

**Robert Boateng-Duah**

**2020**

**ASHESI UNIVERSITY**

# AUTONOMOUS DELIVERY ROBOT

**CAPSTONE**

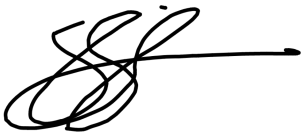
Capstone Project submitted to the Department of Engineering, Ashesi University in partial fulfilment of the requirements for the award of Bachelor of Science degree in Mechanical Engineering.

**Robert Boateng-Duah**

**2020**

# DECLARATION

I hereby declare that this capstone is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature: 

……

Candidate's Name:

Robert Boateng-Duah……

Date:

29/05/2020……

I hereby declare that preparation and presentation of this capstone were supervised in accordance with the guidelines on supervision of capstone laid down by Ashesi University.

Supervisor's Signature:

……

Supervisor's Name:

……

Date:

……

## Acknowledgements

I would like to thank God for the gift of life, which has enabled me to embark on this project. I am also especially thankful to my supervisor who has been very patient with me, very inspiring and supportive. I would not conclude without thanking my lovely family for their support and to Ashesi University.

# Abstract

Autonomous robot navigation is a problem that has been tackled for many years now, since the invention of electronic computers. Since then, many advancements have been made to the extent of the development of self-driving cars on the streets in some countries across the world. This project implements some algorithms for Autonomous Mobile Robot control and navigation.

# Table of Contents

# Table of Figures

# Chapter 1: Introduction

## 1.1 Background Information

Human society, due to advancement in technology, is in constant evolution. One of the most dominant technological revolutions in our time is automation. Processes, systems, and procedures are gradually gravitating towards autonomy, and this is indeed same for transportation systems. One major development in the transportation space is the Autonomous Mobile Robot.

The Autonomous Mobile Robot (AMR) is a type of robot that has the ability to move from one point to another on their own, while having the ability to handle environmental changes and obstacles that may impede or inhibit their movement towards a certain goal location. This robot technology incorporates knowledge from Physics (Mechanics), Control Systems, Mechatronics, and Algorithms.

AMRs have many uses in the world, ranging from research, search-and-rescue, sports, and day-to-day activities. In the evolving field of robotics, AMRs are a central theme and have large areas of opportunity for research and implementation.

## 1.2 Problem Definition

For many industrial/logistical companies, the use of manpower to handle delivery and logistics tends to be quite tedious and takes up a lot of time and resources. The use of manpower makes logistics susceptible to human error and human inefficiencies. The development of the AMR will propel the necessity for industry to shift its interest towards the use of machines to increase efficiency and boost their production outputs. Also, in a time like this where the coronavirus has taken its toll on the world, non-contact measures have become very essential in providing services in a way that will mitigate the spread of the virus.

### 1.3 Project Aim and Objective:

The aim of this project is to design an autonomous mobile robot platform (hardware and software) that can do delivery. The objectives are:

- To design and model an AMR platform
- To implement and test suitable algorithms for control and autonomous navigation.

### 1.4 Scope

The scope of this project is only limited to the control system of the mobile robot and its integration with the mapping and localisation system. This will be implemented and tested by the use of computer simulations.

### 1.5 Expected Outcome of the Project

It is expected that this project will lead to the development of control software that can serve as the base for some AMRs.

### 1.6 Motivation for the Project Topic

I have always been fascinated by robots since I was young. This passion followed me through high school where I was in the robotics club and together, we as a team excelled in several competitions. Coming to university, I gained a larger context of robotics in its ability to solve real life problems and I felt the capstone was a great opportunity to explore that, while also being able to meet my undergraduate course requirements.

### 1.7 Research Methodology

The main source of information for this project would be from primary and secondary research. The primary research data will come from conducted computer simulations while the secondary data will come from journals, articles, books, videos, etc.

## 1.8 Thesis Chapter Outline

Chapter One is composed of the main introduction to the project, which has the background information, project scope, project significance, expected outcome of the project, the motivation behind the project, and the problem the project seeks to solve.

Chapter Two reviews the literature of which the project is based around, and explores works conducted by researchers relating to the project topic.

Chapter Three constitutes the methodology, design selection procedure, and the tools and techniques used in the project.

Chapter Four looks at how that methodology is implemented in software and computer simulations

Chapter Five constitutes the results of the work

Chapter Six describes a summary of the project and concludes on the work done.

# Chapter 2: Literature Review

## 2.1 Overview

This section explores what AMRs are, their types, and how they function. It also looks into tools, techniques and systems that come together to make the AMRs function.

## 2.2 Literature Review

### 2.2.1 Autonomous Mobile Robots (AMRs)

AMRs are a class of mobile robots that are pre-programmed to move from one place to another without physical guidance. This is in contrast to tele operated mobile robots, which require a user to input commands in real-time to the robot. [1]. AMRs usually come in several categories. Some of these are:

- Unmanned Aerial Vehicles (UAVs)
- Autonomous Ground Vehicles (AGVs)
- Autonomous Underwater Vehicles (AUVs)

There are also a class of autonomous vehicles that are able to operate both on air and land or underwater and land. Modern technology has allowed the possibility for practically all vehicles to be made autonomous. As per this project, the major focus is on AGVs with a specific look into Wheeled Mobile Robots (WMRs) which are suitable for mobility on mostly smooth ground surfaces. The major types of WMRs include [1]:

- Differential Drive WMRs
- Car-like (Ackerman steering) WMRs
- Omnidirectional WMRs

There are several areas that come together to form the complete design of a mobile robot. These include the kinematics and dynamics of the robot, odometry, and navigation.

### 2.2.2 Kinematics & Dynamics

Robot kinematics deals with the configuration of robots in their workspace, the relations between their geometric parameters, and the constraints imposed in their trajectories [1] The kinematic model of a robot is therefore a mathematical model that predicts the future state of the robot per a given set of kinematic inputs. Some of the parameters that make up the state of the robot include position, acceleration, heading, angular speed, etc.

The dynamics of the robot on the other hand refer to the relationship between different types of forces that interact with and affect the robot. The main physical elements considered during dynamic modelling include inertia, elasticity, force or torque, and friction [1]. Together, the dynamic and the kinematic model of the robot give a complete and accurate approximation of the behaviour of the robot behaviour. Control algorithms are then applied to these models in order to influence the robot behaviour to achieve desired results.

### 2.2.3 Odometry

Odometry is the process of determining the current state parameters (position, speed, heading, etc.) using motion sensors [2]. Robot odometry is very essential in autonomous robot navigation as knowing the robot's state at every point in time informs the decisions the robot has to make to get to its goal. Some sensors used for robot odometry include wheel encoders, Inertial Measurement Units (IMUs), Global Positioning System (GPS), and range sensors like ultrasonic sensors, infrared proximity sensors, and laser sensors.

Sensors used for odometry are prone to noise, which affects the accuracy of the sensor readings. The IMU (Inertial Measurement Unit), which gives information on the attitude of the robot, consists of a gyrometer, an accelerometer, and a compass (magnetometer). All these sensors

are prone to error due to electrical signal noise and drift. The gyrometer and accelerometer are prone to drift, which causes the data readings from the sensors to offset at equilibrium position, while the magnetometer is prone to magnetic field interference from high-powered radio signals or magnets in its environment. These errors have debilitating effects on the autonomous performance of the robot. With this, algorithmic techniques are used to fuse the sensors to estimate a state close to the actual state of the robot. [3]

### 2.2.4 Navigation

An essential part of the AMR is its ability to find its way from its current position to its goal position. There are various navigation techniques used by mobile robots, of which some involve mapping the entire environment before navigating it. One example of this is SLAM (Simultaneous Localization and Mapping). [4]

Other techniques are related to Waypoint Navigation, where the robot is given local or global coordinates to go to. These techniques do not require a map of the surroundings. Some of these techniques are bug algorithms and potential field methods [obst and path planning].

# Chapter 3: Design and Modelling

## 3.1 Requirements

1. The system should be capable of avoiding obstacles
2. The system should be able to recognise where it is at every point in time and be able to move from one recognisable location to another

## 3.2 Design Decisions

Before the project could be initiated, there were a series of decisions that had to be made based on the requirements, cost, resources, and time constraint on the project. A decision matrix was employed choosing the best approach in each of the given scenarios. The first decision to be made was whether the project would be done completely physically, completely as a simulation, or would be both simulated and physically built.

Table 3.1

| **Parameters (weight)** | **Physical Build** | **Simulation** | **Both** |
|---|---|---|---|
| **Availability of resources (35%)** | 30 | 90 | 50 |
| **Cost Effectiveness (35%)** | 30 | 90 | 20 |
| **Complexity (30%)** | 50 | 50 | 20 |
| **TOTAL** | **36** | **78** | **30.5** |

The decision to go with simulations was the most reasonable one given that it is more expensive to build, especially that failures during physical build can be very costly to development time. Simulations on the other hand, are virtually free, especially when the software is open-source.

Upon arriving at simulations, the next decision was to figure out which simulation environment would be suitable:

Table 3.2

| **Parameters (weight)** | **MATLAB-Simulink** | **ROS (Robot Operating System)** | **CARMEN** |
|---|---|---|---|
| **Cost Effectiveness (50%)** | 30 | 90 | 20 |
| **Ease of Use (20%)** | 50 | 30 | 70 |
| **Robotics Packages Available (20%)** | 50 | 70 | 80 |
| **Community (10%) Support** | 50 | 70 | 40 |
| **TOTAL** | **40** | **72** | **40** |

The decision to go with ROS is because ROS has a massive community and the software is completely open-source. Many manufactures have their sensors and robot platforms available to use on ROS, thus it makes it very practical for use for robotic simulations. On top of that, it has the most seamless interoperability between simulations and physical build. It is also the most widely used software for robotics in industry, such that MATLAB even has packages to write software for ROS.

Table 3.3

| **Parameters (weight)** | **Differential drive** | **Ackermann (car-like)** | **Omnidirectional** |
|---|---|---|---|

| | | | |
|---|---|---|---|
| **Manoeuvrability (20%)** | 70 | 50 | 90 |
| **Complexity (20%)** | 50 | 40 | 30 |
| **Efficiency (30%)** | 65 | 80 | 40 |
| **Speed (30%)** | 50 | 80 | 50 |
| **TOTAL** | **58.5** | **66** | **51** |

The decision to go with the four-wheel robot was mostly based on speed and efficiency. A four-wheel platform is also more capable of handling unstable terrain compared to other mobile robot platforms. Considering that the autonomous robot is aimed that deliveries, speed and efficiency are the most important benchmarks for making the best decision as to which platform to use.

### 3.3 System Functionality

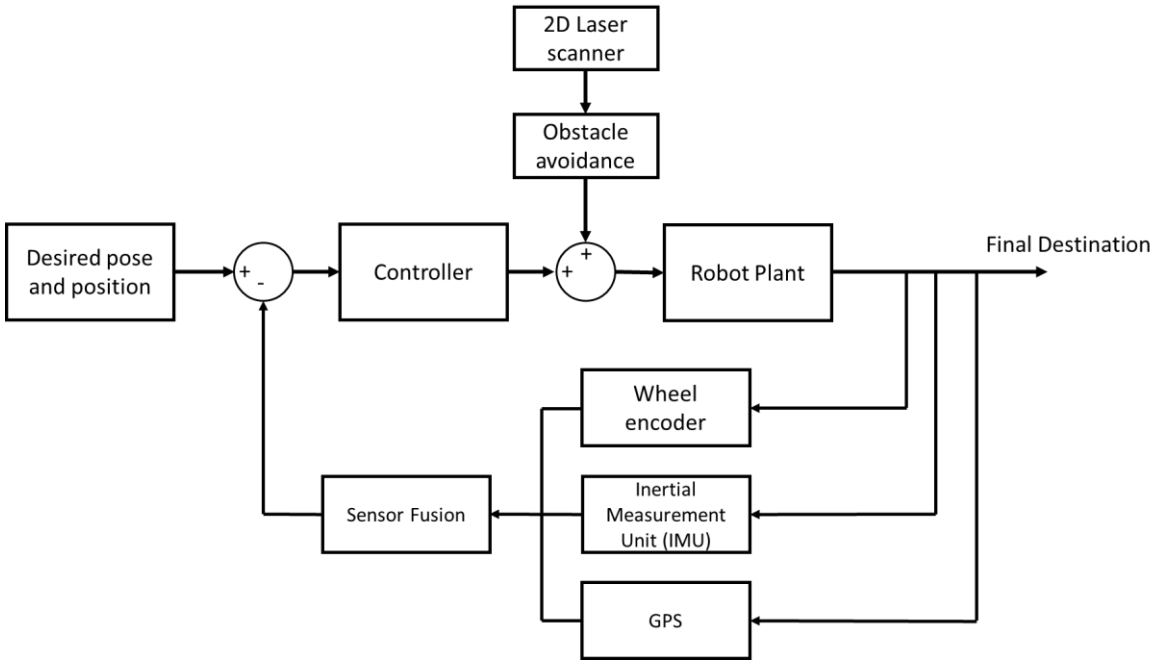



*Figure 3.1: Block diagram showing how systems will interact*

The way the system works is that it takes input parameters such as the desired pose and position of the robot. This information feeds into the controller, which ensures that the robot is

on the right trajectory. An obstacle avoidance algorithm interferes causing a shift in the robot's planned path when a disturbance (an obstacle is detected). The present state of the robot is then measured by a wheel encoder, IMU, and GPS and fused using a filter or optimal estimation algorithm. The resulting state is then compared against the desired robot state to see if it has achieved its goal. When the goal is achieved, the program ends.

### 3.4 Autonomous Mobile Robot Design

#### 3.4.1 Ackermann steering

The Ackerman steering is a method of steering where the inner tyres are bent at a higher angle than the outer tyre, with respect to the radius the inner tyre has to turn as compared to the outer tyre. The Ackerman steering geometry ensures that the robot tyres do not slip while negotiating a curve. This is ideal for maintaining the no-slip assumption for the mobile robot and for calculating the centre of turn of the robot, which will be used by the bicycle kinematic model to predict the robot future states and calculate its trajectory [5].

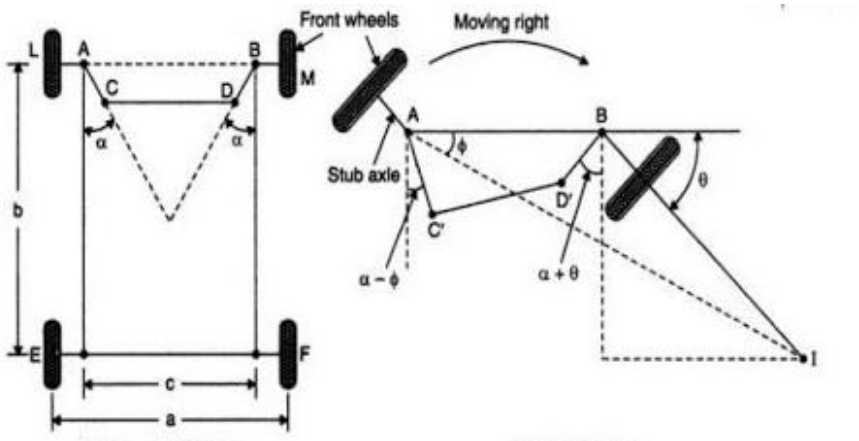



*Figure 3.2: Ackerman steering free body diagram*

### 3.4.2 Kinematic Model

The mobile robot uses a four-wheel, car-like platform. This platform adopts the bicycle kinematic model with non-holonomic constraints. A free-body diagram of the system is illustrated below [1]:

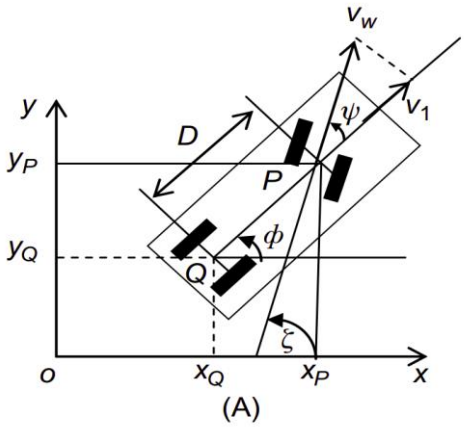



*Figure 3.3: Bicycle model free body diagram*

The state equation of this system is also presented below:

$$\dot{P} = \begin{bmatrix} \dot{\phi} \\ \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \tan\frac{\psi}{D} & 0 \\ \cos\phi & 0 \\ \sin\phi & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \omega \end{bmatrix}$$

Where:

- Phi is the heading of the robot
- X is the displacement of the robot in the x-axis
- Y is the displacement of the robot in the y-axis
- Psi is the steering angle of the robot

A MATLAB Simulink model with the implementation of the bicycle kinematic model is shown below:

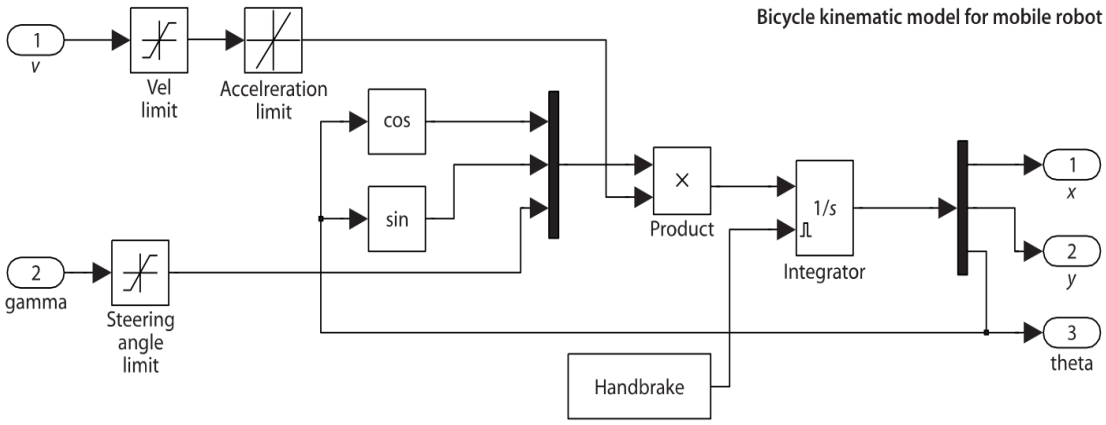



*Figure 3.4: Simulink representation of bicycle kinematic model*

### 3.4.3 Control Algorithms

The robot acts like a state machine, switching between different algorithms depending on the situation the robot finds itself in. These algorithms are [6]:

1. Move to a point
2. Move to a pose

#### 3.4.3.1 Move to a point

This algorithm is to move the robot from one point to another. It calculates the error from the goal by calculating its translational and angular deviations from the goal. The angular deviation between the robot and the goal is defined by:

$$\theta_g = \theta_{goal} = \arctan\left(\frac{y_g - y}{x_g - x}\right)$$

And the linear or translational deviation from the goal is defined by:

$$\sqrt{(x_g + x)^2 + (y_g - y)^2}$$

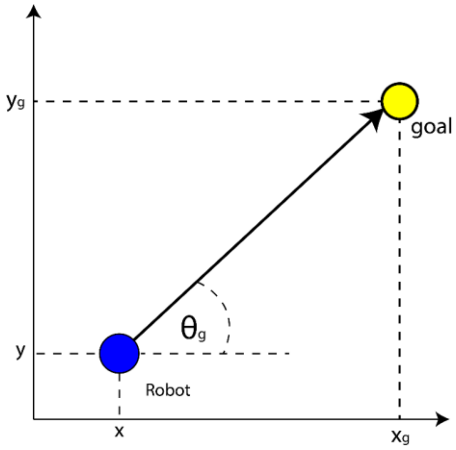



*Figure 3.5: Current position and goal*

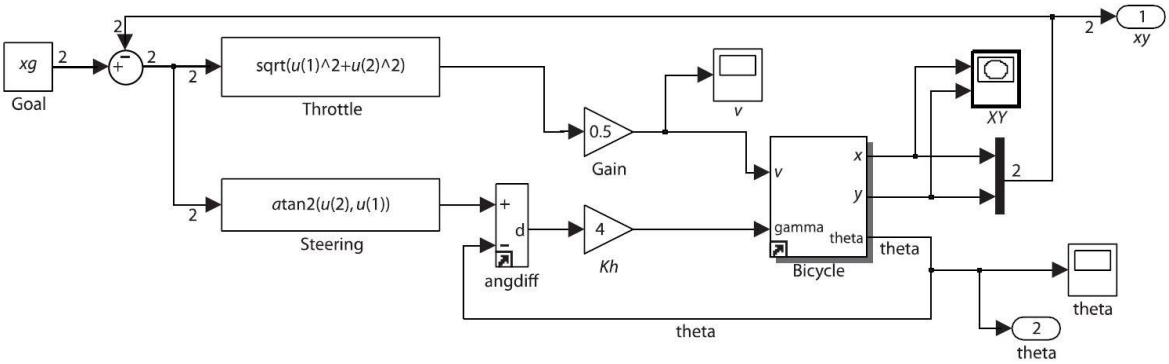



*Figure 3.6: Simulink model for move-to-a-point*

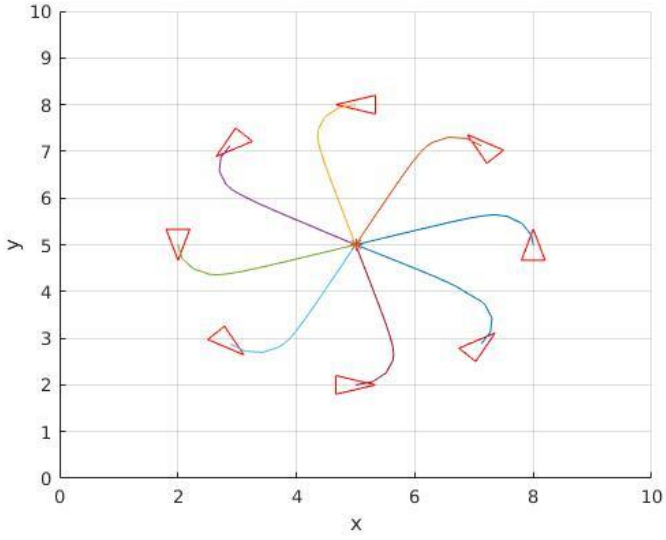



*Figure 3.7: Simulation behavior of move-to-a-point for 8 different trajectories*

### 3.4.3.2 Move to a Pose

Moving to a point may not be enough in many scenarios. Some require reaching a goal with a particular heading or pose. To do this, a different approach in the algorithm design is required. The bicycle kinematic equations are turned into polar equations as seen below [6]:

$$\begin{pmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{pmatrix}=\begin{pmatrix}-\cos a & 0\\ \frac{\sin a}{\rho} & -1\\ -\frac{\sin a}{\rho} & 0\end{pmatrix}\begin{pmatrix}v\\ \omega\end{pmatrix},\ \text{if}\alpha\in\left(-\frac{\pi}{2},\frac{\pi}{2}\right]$$

A linear control law with the equations

$$v=K_\rho\rho \quad \omega=K_\alpha\alpha+K_\beta\beta$$

These drive the robot to a unique equilibrium while driving beta to zero to achieve the desired pose [6]. The system is stable as long as all the constants in the linear control law are greater than zero.

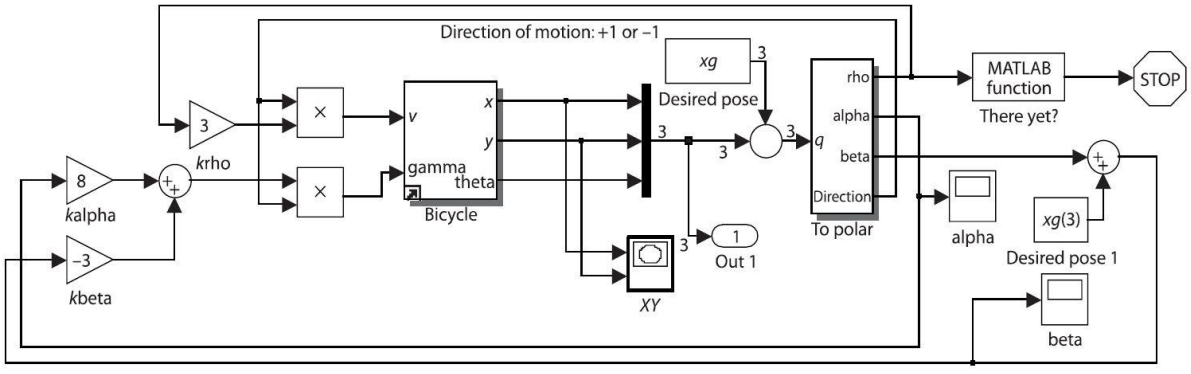



*Figure 3.8: Simulink model for move-to-a-pose*

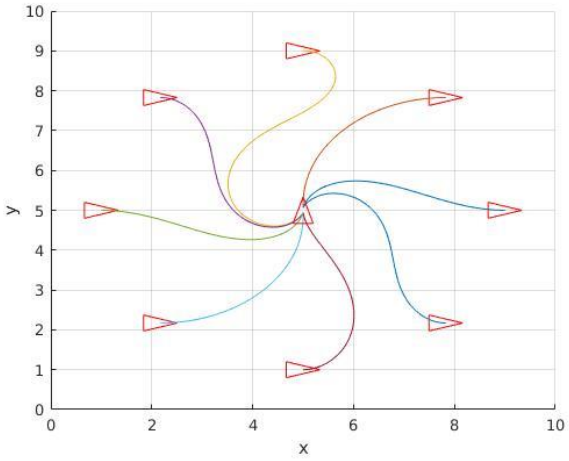



*Figure 3.9: Simulation behavior for move-to-a-pose for 8 different trajectories*

### 3.5.4 Enhanced Vector Field Histogram (VFH+) Algorithm

This is a method of obstacle avoidance and path planning for real-time fast-moving mobile robots [7]. The VFH+ algorithm comes with a safety distance, which is the minimum distance from the robot to the obstacle from which obstacle avoidance should begin. It also comes with a turn radius parameter, which takes in the maximum turn radius of the robot, used by robots with non-holonomic constraints. The algorithm also enhances the "size" of the obstacle to allow the robot safely navigate around it without having to worry about the possibility of hitting it. Data collected from the range sensors go through five stages of processing before the results are acquired.

#### 3.5.4.1 The First Stage

In the first stage, a 2-dimensional map grid is converted to a primary polar histogram grid. In this histogram grid, there is the active region and the inactive region. The active region represents the region of the map grid within the robot sensor's FOV (Field of View). This active

region has active cells $C_{i,j}$, which is treated as an obstacle vector with $\beta_{i,j}$ representing the **obstacle vector direction**. $\beta_{i,j}$ is given by:

$$\beta_{i,j} = \tan^{-1}\left(\frac{y_o - y_j}{x_o - x_i}\right),$$

The magnitude of the active cell, $C_{i,j}$, is given by:

$$m_{i,j} = c_{i,j}^2\left(a - bd_{i,j}^2\right)$$

Where:

- $C_{i,j}$ is the certainty value of the active cell
- $d_{i,j}$ distance from the RCP (Robot Centre Point) to the active cell
- a and b represent arbitrary constants.

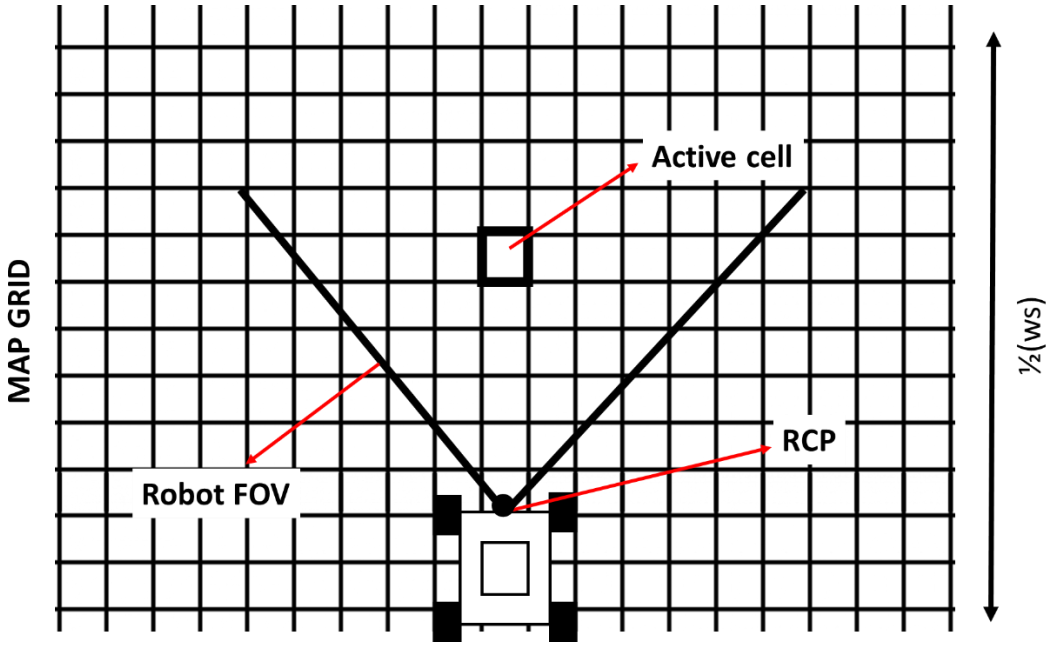



*Figure 3.10: Visualization of active and inactive cells around the robot*

$D_{i,j}$ is squared so that as the robot moves, cells with obstacles produce larger vector magnitudes than when far away. This is also done for the active cell certainty value, allowing for greater confidence in the obstacle vector magnitude values. As said earlier, obstacles are also enlarged by the robot 'radius' (with the assumption that the robot can be approximated as a disk).

Next, an arbitrary angular resolution is determined by the equation:

$$n = \frac{360°}{\alpha}$$

Where n represents the number of angular sectors and *k* represents the index of each angular sector. Therefore, a discrete angle, $\rho = k\alpha$.

The distance around an occupied active cell, $r_{r+s}$ is enlarged by:

$$r_{r+s} = r_r + d_s$$

Where $r_r$ is the robot radius and $d_s$ is the minimum specified distance at which the robot should start avoiding the obstacle. From this, the enlargement angle $\gamma_{i,j}$ can be determined by:

$$\gamma_{i,j} = \sin^{-1}\frac{r_{r+s}}{d_{i,j}}$$

For each sector *k*, the polar obstacle density can be calculated by:

$$H_k^p = \sum_{i,j \in C_{i,j}} m_{i,j} * h'_{i,j}$$

Where $h'_{i,j}$= 1 when $\rho \epsilon [\beta_{i,j} - \gamma_{i,j}, \beta_{i,j} + \gamma_{i,j}]$ and $h'_{i,j} = 0$ otherwise.

$\beta_{i,j} - \gamma_{i,j}$ represents the minimum angle between the active cell and the enlarged obstacle radius while $\beta_{i,j} + \gamma_{i,j}$ represents the maximum angle. This makes sure that the polar histogram incorporates the width of the robot as well as the obstacle enlargement circumference. It also smoothens out the polar histogram.

### 3.5.4.2 The Second Stage

Two hysteresis thresholds $\tau_{low}$ and $\tau_{high}$ are introduced. It is based on these two thresholds that the binary polar histogram. This groups all occupied cells into either free (0) or blocked (1), similar to a non-inverting Schmitt trigger. It is mainly to indicate the directions that are free for a robot to move in, and is the foundation for developing the robot trajectory [8].

$$
\begin{aligned}
H^b_{k,i} &= 1 && \text{if } H^p_{k,i} > \tau_{high} \\
H^b_{k,i} &= 0 && \text{if } H^p_{k,i} < \tau_{low} \\
H^b_{k,i} &= H^b_{k,i-1} && \text{otherwise}
\end{aligned}
$$

### 3.5.4.3 The Third Stage

A masked polar histogram is developed from the binary polar histogram. This masked polar histogram incorporates the robot's kinematic and dynamic model to make more accurate approximations of the path the robot should take. It considers that the fact that a space is free does not mean that the robot can move through that space.

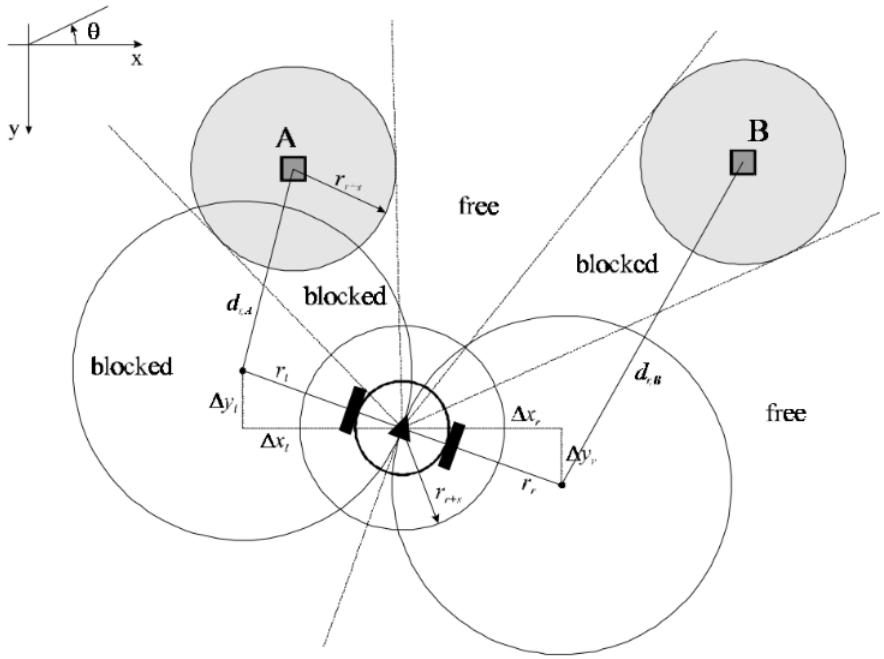



*Figure 3.11: Visualization of the working of the algorithm.*

### 3.5.4.4 The Fourth Stage

In this stage, a range of possible paths the robot can take are developed per the information on free and blocked areas given by the masked polar histogram. A cost function is used to determine the best path among the candidate paths to be taken. This cost function is given by [8]:

$$g(c) = \mu_1 \cdot \Delta(c, k_t) + \mu_2 \cdot \Delta\left(c, \frac{\theta_i}{\alpha}\right) + \mu_3 \cdot \Delta(c, k_{n,i-1})$$

$\mu_1$ ,$\mu_2$, and $\mu_3$ are weights of the cost function responsible for the difference between the candidate direction and target direction, difference between target heading and current heading, and the difference between previous direction and current direction respectively.

# Chapter 4: Simulation

## 4.1 Simulation Environment

The simulation is done in ROS, Gazebo and RViz. ROS (Robot Operating System) is a middleware for robotics-based applications. It is modular and open-source, allowing engineers to reuse boilerplate code and other system algorithms without having to completely start from scratch. In its computational graph, ROS consists of a parameter server, nodes, messages, topics, services, and bags. It has a core feature called ROS master, which is the core program that tracks and manages all the software processes. The node is the smallest block of code needed to run a system in ROS. Nodes communicate with the ROS master and with each other through a topic which they can subscribe to or publish. They communicate on a topic by sending 'messages' which, depending on the topic, have special structure to the information they carry. ROS has a special feature called ROSbag which is primarily used to store data and also develop graphs. It also has a parameter server which is a multivariate dictionary that is accessible via network APIs [9].

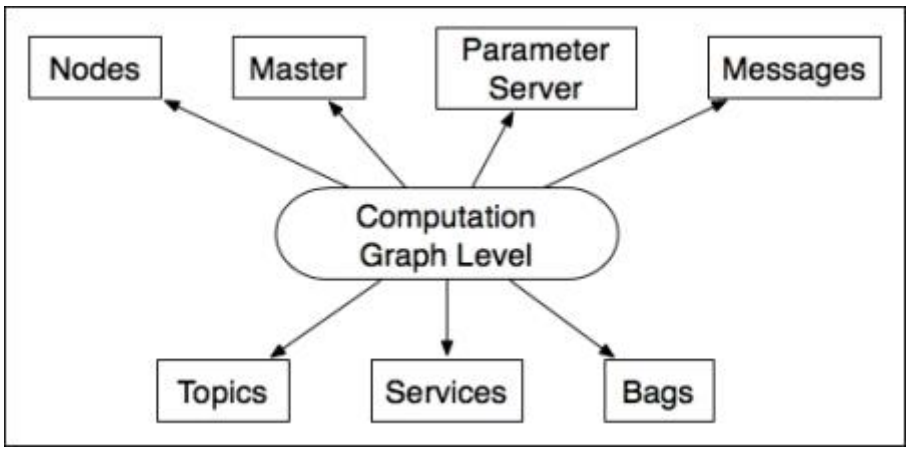



*Figure 4.1: ROS Computational Graph Diagram*

Gazebo on the other hand is a graphical simulation environment that allows robot models to be simulated in a game-like manner. It takes data from ROS as parameters and lets users visualize and analyse the performance of robotic systems under various conditions.

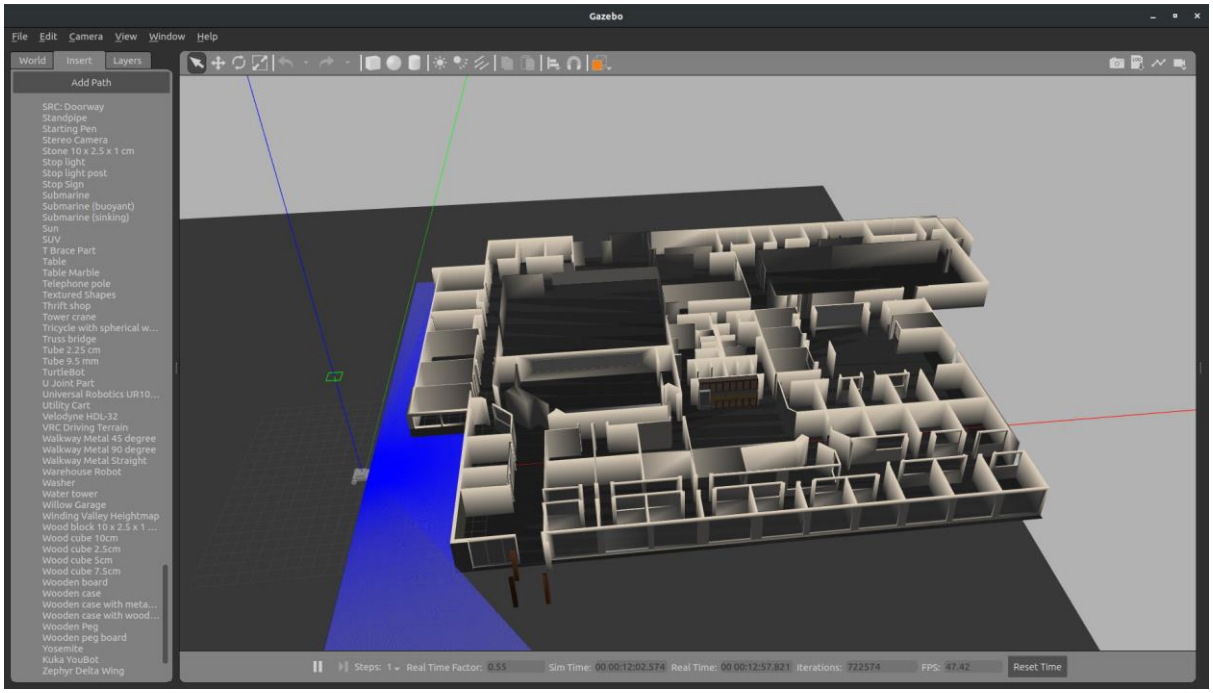



*Figure 4.2: The robot and a floor plan of a building shown in Gazebo*

RViz, which stands for Robot Visualization is in some ways similar to Gazebo, except it is mainly used to visualize the robot as described in the URDF and make some slight tweaks.

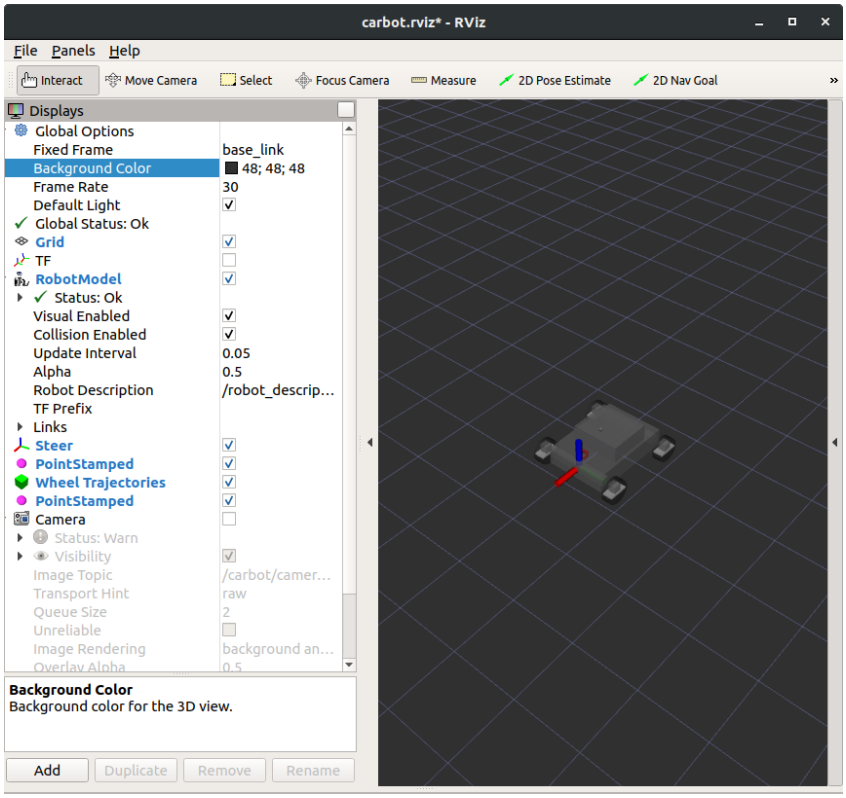



*Figure 4.3: Robot visualized in RViz*

### 4.2 Robot Model

The high-level goal of the robot is to be able to move from one global way-point to another while avoiding obstacles. To achieve this in simulation, the simplest model of a four-wheel mobile robot with non-holonomic constraints was used. This mobile robot used primitive shapes only. Primitive 3D shapes such as cuboids and cylinders, where the cuboids were used for the body and cylinders used for the wheels. This allows for easier modelling of the moment of inertia of the vehicle as well as its collision matrices.

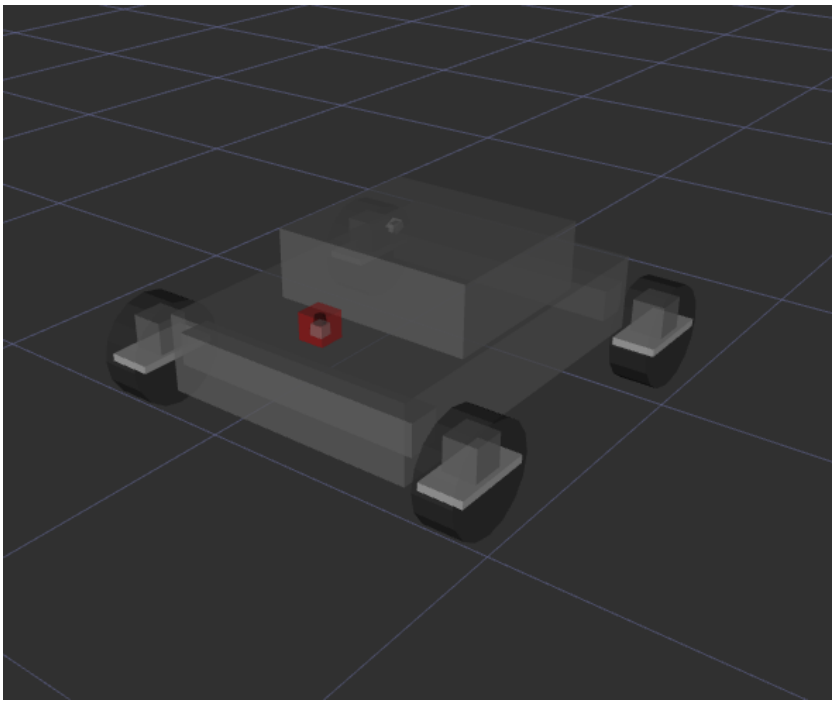


*Figure 4.4: 3D view of the robot*

The robot model is described using a special XML-like format called URDF (Unified Robot Descriptor Format) [10]. This format contains descriptions of geometric properties of the robot such as the dimensions/size, shapes and connecting parts. It also describes the dynamic properties of the robot such as the collision matrix, the moment of inertia, mass, friction coefficient, and the force and torque in the crucial moving or rotating parts. The URDF contains description of the kinematic properties of the robot like the degrees of freedom/rotation and

limits like the maximum speed and acceleration of critical moving parts. In addition to that, PID controllers can be added with gain parameters.

The robot description is formatted in a graph tree-like structure with nodes being called **links** and vertices being called **joints**. The URDF graph tree of the robot is shown below:

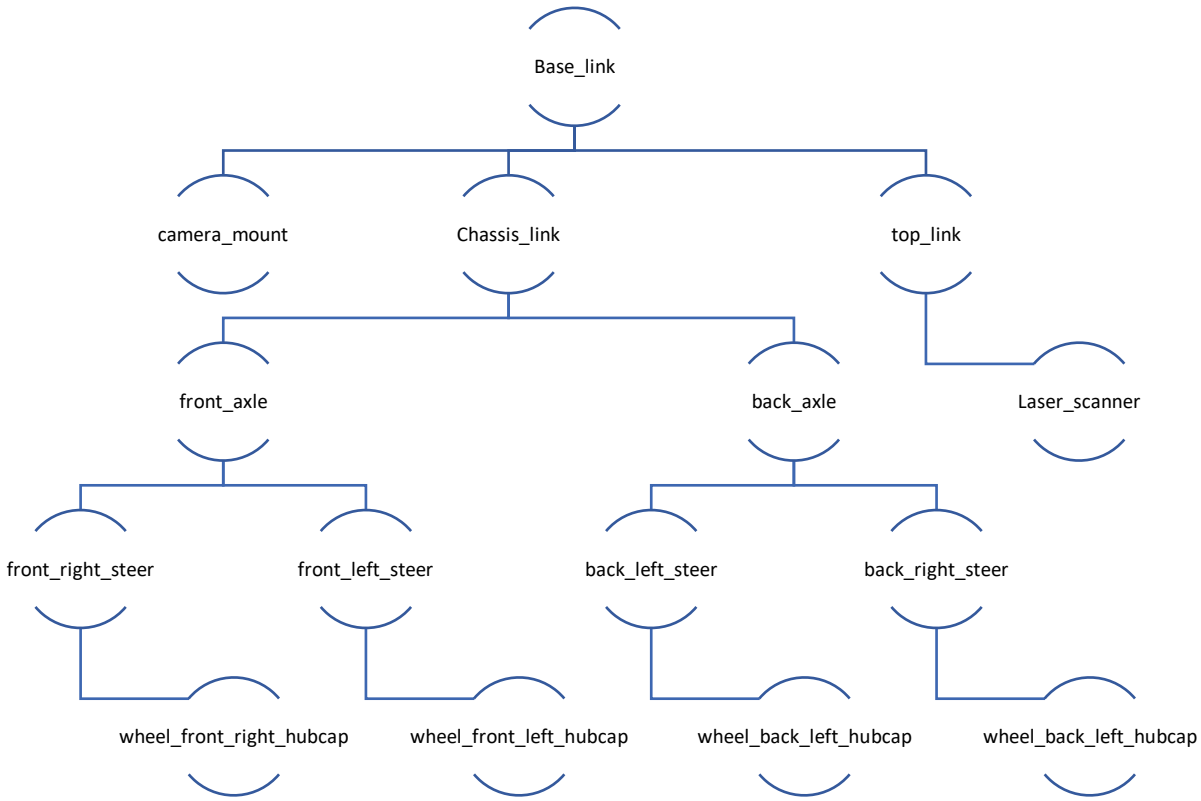



*Figure 4.5: URDF Graph of the robot*

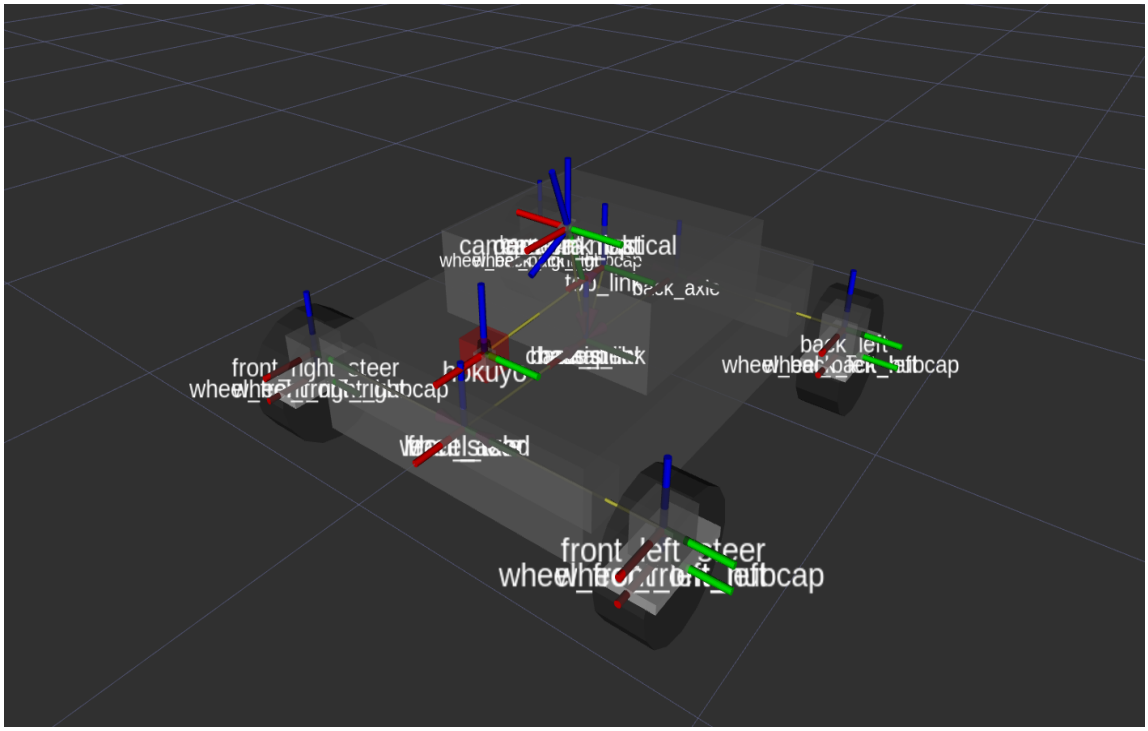


*Figure 4.6: Visualization of geometric positions of robot joint links*

The robot description also has in it a *transmission* tag. The transmission represents the actuators needed to provide torque for the vehicle. The transmission also comes with a *joint_state_controller* which sets up a controller for all joints that are going to be in motion due to the torque provided by the actuator. Position controllers using the joint state controller are created for the *front_left_steer, front_right_steer, back_left and back_right.* The position controllers ensure that, as in a real system, the desired velocity and position are met when torque is applied, by correcting any errors. The transmission is applied on the front axle to control the steering and to the back axle for forward and reverse movements.

### 4.3 Robot Control

The robot is controlled by the interaction of special nodes, mainly the **tf** (meaning transform) and the joint_state_publisher. The joint_state_publisher is a node that takes the states of all the joints from the URDF file and publishes them. The published states go through

the kinematic equations written as nodes and goes to the **tf** node which updates the joint states with the results given by the equations.

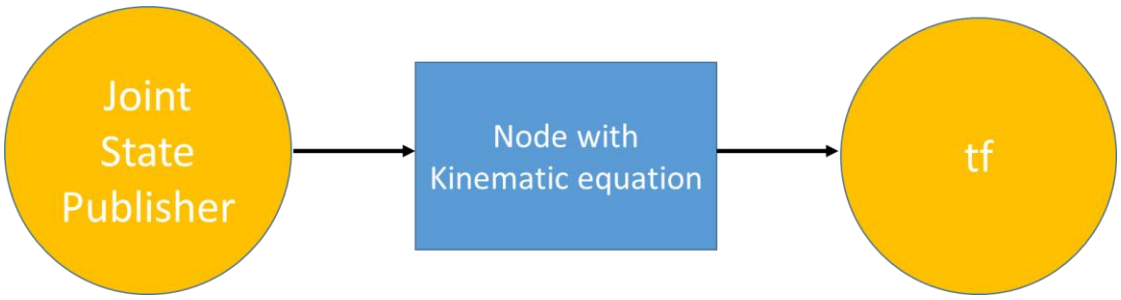



*Figure 4.7: Robot control process diagram*

After going through the tf node, the new joint states can also be published on Gazebo and RViz, where it is now possible to visualize what is going on.

### 4.3.1 The Ackerman Steering Node

A python API for ROS called *rospy* is used to create a node that computes the future states of the robot given the current robot state using the kinematic model and the ackermann steering equations. The node imports all controlled joint states from the joint state publisher. From there, it initializes a module to send commands to the Gazebo environment. It then initiates a program to read the angle and current position of each wheel. Using the Ackermann equation, the spin center is computed and a special marker is generated to visualize the trajectory of the robot in RViz.

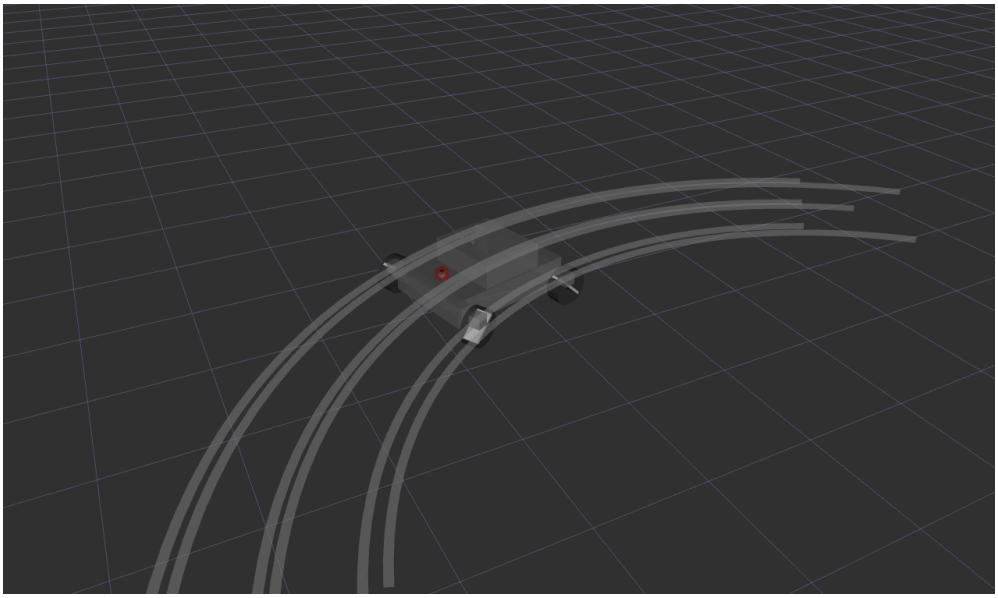


*Figure 4.8: Trajectory as projected by marker*

### 4.3.2 Navigation Node

The VFH+ algorithm is implemented using the laser scanner as the range detection sensor. The scanner FOV is divided into bins to be processed into the histogram grid. The navigation node subscribes to range messages published by the topic */carbot/laser/scan* which it uses to compute the path the robot must take to avoid the obstacles and reach the goal point. The path is then translated into the necessary linear and angular velocities required and then published to the */cmd_vel* topic which is responsible for issuing movement commands to the robot. The move-to-a-point and move-to-a-pose control algorithms embedded in the navigation node are used to ensure that the robot follows the VHF+ algorithm's recommended path.

## 4.4 Addition of Sensors

Adding sensors was done by including a new link and joint tag in the robot URDF file of the sensor. The sensor plugin code was also added to the Gazebo file, which contains a link to the sensor's simulation files. From there, the sensor can be initialized and visualized on the Gazebo simulation environment. The first sensor to be added was the 2-D laser scanner with

had a 180-degree field of view and a range of 30 meters, with a resolution of 20 lasers per degree manufactured by Hokuyo.

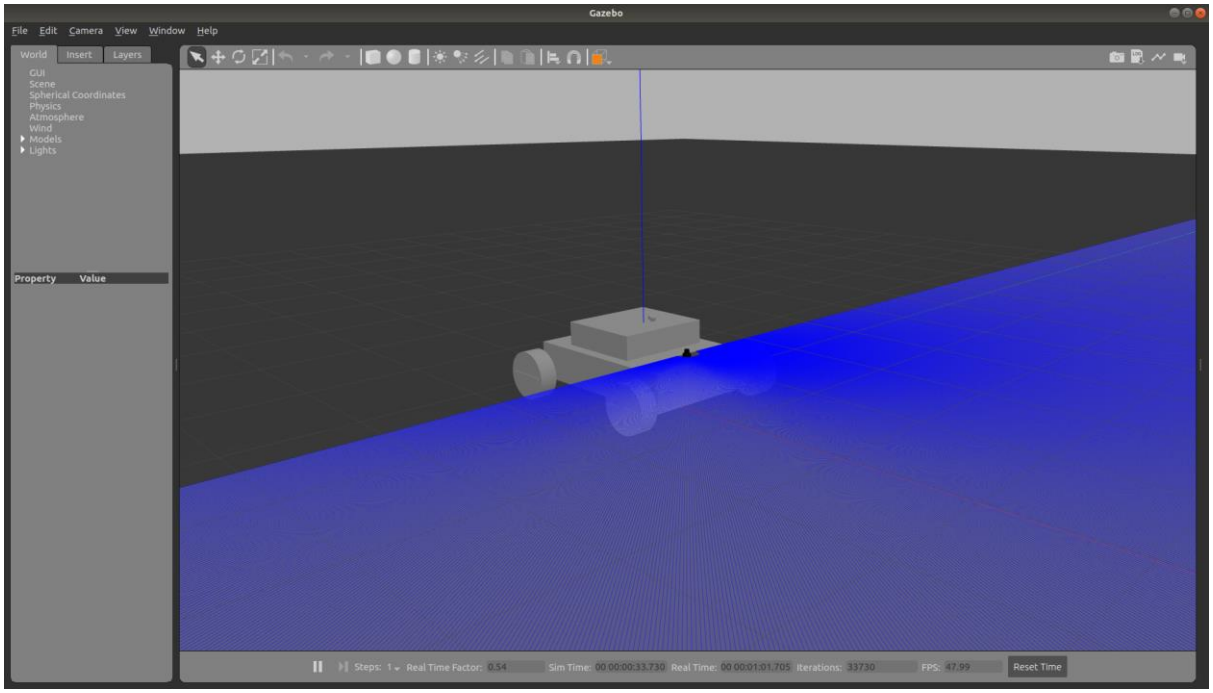



*Figure 4.9:Robot simulation with laser scanner. Blue color indicates the visualization of the laser scanner*

This same procedure was followed to add the GPS and the IMU. For the wheel encoder, it was modelled as a simple programme to track the angular position of the wheels at every point in time in the Ackermann steering node.

# Chapter 5: Results & Discussion

## 5.1 Simulation Outcome

The simulation was run with a world designed in Gazebo to mimic some obstacles the robot may encounter. The simulation was done with the robot's rear wheels moving at a constant translational velocity of 0.1m/s

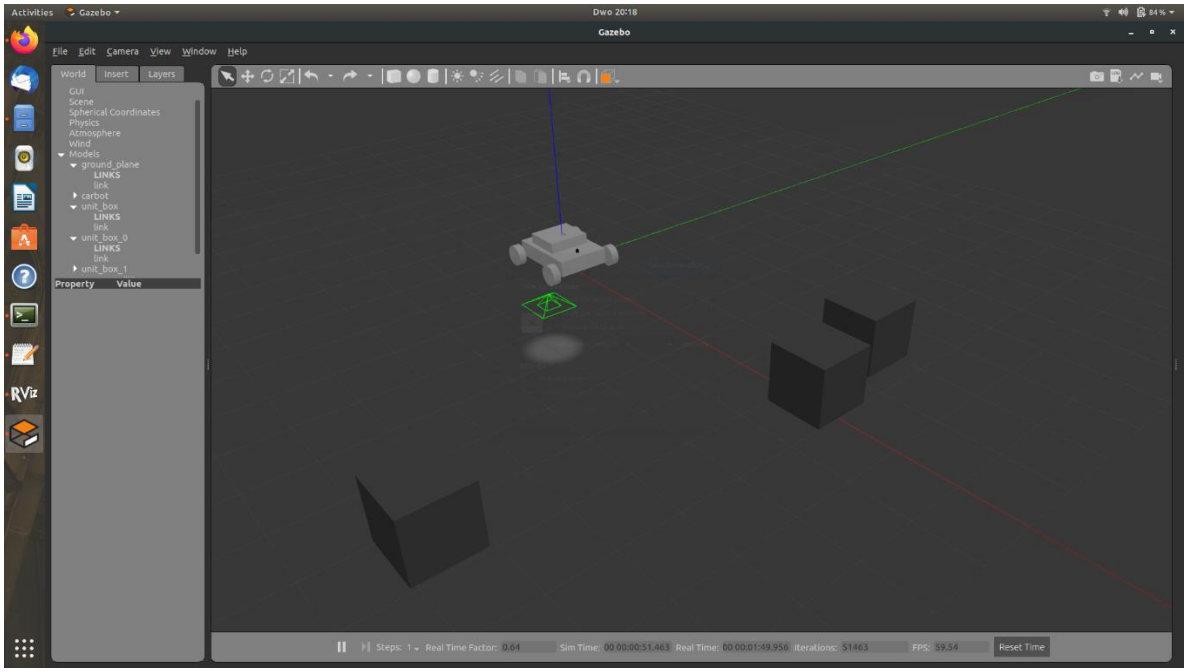



*Figure 12: Robot starts moving from rest*

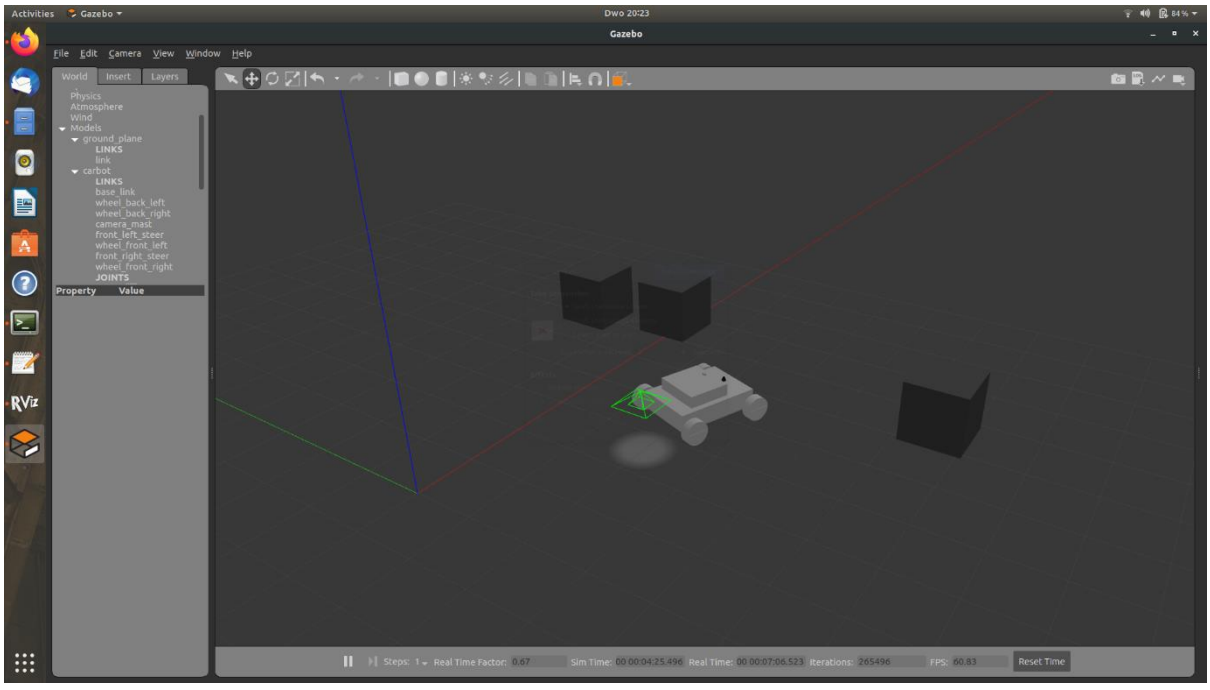



*Figure 13: Robot succeeds in avoiding obstacle*

This is for a very simple scenario, however for some more complex scenarios, the robot was not able to successfully maneuver its way.

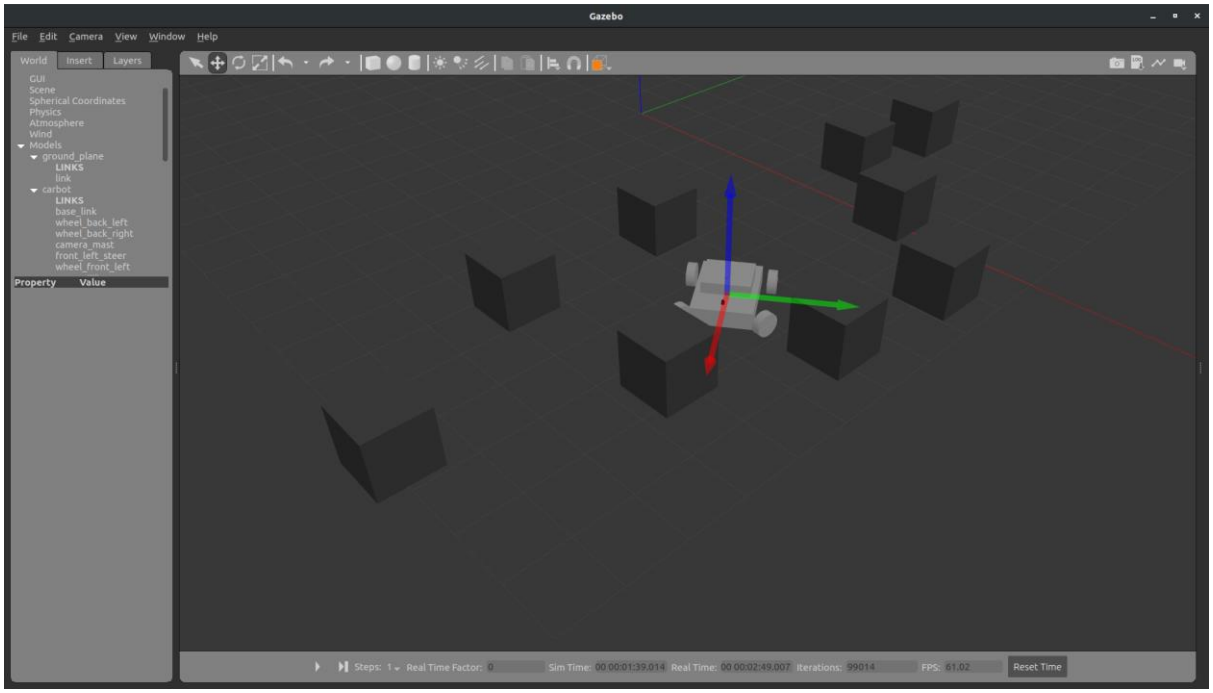



*Figure 5.3: Robot unable to work its way around the obstacle*

## 5.2 Discussion

The reason for the failure in the second scenario was that the robot saw similar distances between the boxes, and given the constraints due to its kinematics, computing the most efficient path became more complicated. This could be curbed by modifying the cost function parameters; however, doing that would also affect its performance in less complex scenarios.

# Chapter 6: Conclusion, Limitations and Future Works

## 6.1 Conclusion

Following the success of the robot design to achieve all of its objectives, the fact remains that more can be done at improving the design to make it more comprehensive and more modular. The simulations do not perfectly mirror the exact performance of the robot in real life, however it is a great way to visualize how the robot is going to perform if built. It is also a great way of observing which parameters in the robot need to be prioritized or paid attention to.

## 6.2 Limitation

Below are a few issues that were limitations to the project:

- Short project duration
- COVID-19 pandemic delayed the arrival of some components which would have enabled the simulation to be implemented on a physical model
- Limited computational power did not allow for more advanced simulations

## 6.3 Future Works

- Test the simulations in a physical model
- Use more complex shapes other than primitive shapes for modelling the robot
- Make the software more modular and adaptable to different robot platforms

# References

[1] Tzafestas, Spyros G. *Introduction to mobile robot control*. Elsevier, 2013.

[2] Howard, Andrew. "Real-time stereo visual odometry for autonomous ground vehicles." 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2008.

[3] Jing, Xiaofei, et al. "Attitude estimation for UAV using extended Kalman filter." 2017 29th Chinese Control And Decision Conference (CCDC). IEEE, 2017.

[4] Durrant-Whyte, H.; Bailey, T. (2006). "Simultaneous localization and mapping: part I". IEEE Robotics & Automation Magazine. 13 (2): 99–110. CiteSeerX 10.1.1.135.9810. doi:10.1109/mra.2006.1638022. ISSN 1070-9932.

[5] Mitchell, Wm C., Allan Staniforth, and Ian Scott. Analysis of ackermann steering geometry. No. 2006-01-3638. SAE Technical Paper, 2006.

[6] Corke, P., 2011. Robotics, Vision and Control: Fundamental Algorithms in MATLAB. 1st Edn., Springer, Berlin Heidelberg Springer, ISBN-10: 3642201431, pp: 570.

[7] Borenstein, J. and Koren, Y., "The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots", IEEE Journal of Robotics and Automation, June 1991, Vol. 7, No. 3, pp. 278-288.

[8] Ulrich, Iwan, and Johann Borenstein. "VFH+: Reliable obstacle avoidance for fast mobile robots." Proceedings. 1998 IEEE international conference on robotics and automation (Cat. No. 98CH36146). Vol. 2. IEEE, 1998.

[9] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[10] Garage, Willow. "Xml robot description format (urdf)." URL {http://www. ros. org/wiki/urdf/XML}{accessed May 4 2020.} (2012).