TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
Department's title

BOE-BOT SIMULATION IN GAZEBO

BOE-BOTI SIMULEERIMINE GAZEBOS

MASTER THESIS

Student: Harris Antony

Student code: 194383MAHM

Supervisor: Priit Ruberg
Researcher, Department of Computer Systems

Course-Supervisor: Saleh Alsaleh
Engineer, Department of Electrical Power Engineering and Mechatronics

Tallinn 2021

(*On the reverse side of title page*)

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"18". May . 2021

Author: Harris Antony
/signature /

Thesis is in accordance with terms and requirements

"18". May . 2021

Supervisor: Priit Ruberg
/signature/

Accepted for defence

"....."....20… .

Chairman of theses defence commission: .....
/name and signature/

Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I, Harris Antony
hereby

1. grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis Boe-Bot Simulation in Gazebo,

supervised by
Priit Ruberg,

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

_____ (date)

[1] *The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.*

Department of Electrical Power Engineering and Mechatronics

THESIS TASK

Student: Harris Antony, 194383MAHM

Study programme, MAHM02/19 - Mechatronics

Main speciality: Mechatronics

Supervisor(s): Priit Ruberg, Researcher, Department of Computer Systems

Course-Supervisor: Saleh Alsaleh, Engineer, Department of Electrical Power Engineering and Mechatronics

Thesis topic:

(in English) Boe-Bot Simulation in Gazebo

(in Estonian) Boe-Boti simuleerimine Gazebos

Thesis main objectives:

1. Perform background research of simulators including ROS.
2. Modelling Boe-Bot and Perform simulation in Gazebo.
3. Build and Interface between MATLAB and ROS.

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Literature Review	December
2.	Reaserch about Gazebo and ROS	January
3.	Modelling and simulation of different environments	February-March
4.	Study of MATLAB and creating interface	April
5.	Thesis writing and Review	May

Language: …… **Deadline for submission of thesis:** "....."..... 20….a

Student: …….. ..... "....."…….....20….a
/signature/

Supervisor: …… …….. "....."..... 20….a
/signature/

Consultant: …… …..... "....."..... 20….a
/signature/

Head of study programme: …… ..... "....."..... 20…..a
/signature/

Terms of thesis closed defence and/or restricted access conditions to be formulated on the reverse side

Table of Contents

PREFACE

The idea of this thesis is to develop a Boe-Bot simulation environment in Gazebo for the students to test their python codes on the Boe-Bot. The design and implementation of the Boe-Bot simulation system are based on ROS and Gazebo. It aims to test different algorithms in the same Boe-Bot. After building the Boe-Bot model in Fusion 360, the URDF file generated has been exported to the Gazebo for further simulations. Through the Gazebo ROS Package, ROS is tightly integrated with Gazebo. This package contains a Gazebo plugin module that allows Gazebo and ROS to communicate in both directions.

ROS provides libraries and tools that analyze the Boe-Bot movement and different functionalities. Two simulation environments have been created one is line following Boe-Bot environment and the other is Maze following Boe-Bot. ROS nodes are used here for the communication between python scripts and Boe-Bot movement. The line follower Boe-Bot works with the help of a camera and laser sensor which are mounted on the robot body. OpenCV libraries make the real-time vision recognition faster and make the Boe-Bot movement easy and fast. The Maze follower simulation works under the right-hand algorithm and ultrasonic sensor.

The second objective was very interesting to build an interface between MATLAB and ROS. The study which has been done about different simulators was very interesting. The interface has created with the help of the ROS toolbox. Ant ROS-enabled robot or simulator can use this toolbox to import data from sensors into MATLAB for visualization, exploration, and calibration. In this task different data of Boe-Bot while simulating has been imported to MATLAB and analyzed.

The author would like to thank the supervisor of this thesis researcher Priit Ruberg and co-supervisor Saleh Ragheb Saleh Alsaleh for all the support and for providing all the facilities to complete this thesis.

This thesis is written in English and is 68 pages long, including 5 chapters.

List of abbreviations and symbols

ROS	– The Robot Operating System.
SDF	– Simulation Description Format.
ODE	– Open Dynamics Engine.
RVIZ	– Robot Vizualization Software
URDF	– Universal Robotic Description Format
PRM	– Probabilistic Road Map
ICR	– Instantaneous center of Rotation
RADOE	– Robot Application Development and Operating Environment
TF	– Transform
XACRO	– XML macro language
RQT	– ROS Tool

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

''Day by day, robotics is a constantly expanding field. Robots are used in human life as self-driving cars, packing robots, vacuum cleaners, etc. Nowadays, the availability of virtual tools allows developers to create virtual working environments, and to do a targeted stimulation of robots' dynamics. Some of them are laggy and the metrics are so complex and the simulations difficult. These problems may lead to the possibility of testing the algorithms of robot control, as there will be significant discrepancies between life and simulated environments. It is useful to study the use of alternative simulation environments. More significantly, how simulators can cope with more complex and multi-robots and how important the elements of the simulation are exposed to each other. In the current generation, the complexity has increased and the systems are fully automated so numerous sensors and transducers are introduced to make it harder to choose proper simulation software'' [1].

Simulators differ from each other based on the features they are providing, the quality of the output, and the time consumption for the whole process. The gazebo is an open-source 3D robotics simulator. At the same time, Gazebo has different types of absences in the environment, which makes simulating incomplete. The designing of robot models can be done in a standard way by importing from the library as a simulation description format. The specific design of robots from the user's view is quite difficult in this simulation software. Gazebo's environment is remotely sufficient to simulate a simple robot. The world and or the complexity of the robot falls in the real-time factor (RTF) of the simulation, as the Gazebo's environment can't perform the necessary calculations quickly enough in uneven soil. These problems mediate the testing capabilities of robot control algorithms. This thesis is a study based on different simulators and how the robot performs on different environments of simulators.

The primary objective of this thesis is to create an environment for the students to test their python codes and analyze the behavior of Boe-Bot according to their commands.

- The first stage is to model the Boe-Bot. Different sensors, cameras, and encoders were also added with the design for simulating the robot with different features and exporting it into Universal Robotic Description Format(URDF).
- Setup the environment for the simulation software Gazebo and Robot operating system(ROS) to initialize the work.
- Conducting two different simulations on Boe-Bot, one is line following Boe-Bot and the other is maze following Boe-Bot.

- The second objective is to build an interface between MATLAB and Robot operating system using a ROS toolbox.
- Analyze the type of messages, sensor data, and different quantities through MATLAB.

Chapter 1 provides an introduction to the thesis followed by the motivation and objectives of the study. The objectives are further detailed in the body of the report.

Chapter 2 contains an analysis of previous studies conducted about the objectives of the thesis referring to the published data. The section further discusses the justification of the scope of the thesis.

Chapter 3 consists of the methodology of the thesis where it describes the selection of the hardware and software that will be used in the project. Furthermore, it provides details on different tools and libraries used for the development of the simulations.

Chapter 4 comprises a detailed explanation of mechanical aspects of Boe-Bot modeling and design flow is obtained.

Chapter 5 provides the process flow chart, the programming methods used in PC and robot. The interface between MATLAB and the ROS was also explained.

Chapter 6 comprises the summary of the thesis.

2 LITERATURE REVIEW AND BACKGROUND

This chapter discusses existing solutions and studies on robot modeling as well as the methods that have been used to make simulation easier and more accurate. It will be addressed how important is an advanced technology in robotic industries. This section will dig further into the current literature to clarify previous approaches that are closely related to the thesis goal and will apply the concepts to the development of simulation in various simulation environments.

2.1 Robot and Environment Model

The primary task in simulation is the designing of robot models. According to [2] they have created an unorthodox robot design, for that the first step of designing has been done in CAD as a base and created appropriate controllers for the dynamic joints. The author used two separate files on universal robot description format for the visualization in RVIZ and processed the simulation Gazebo. In addition to that in [3] was created a new simulation software robot application development and operating environment in which they have imported the model which has been created but Gazebo and robot visualization have shown on the screen at the same time. With the help of the robot visualization plugin, they have developed a GUI. The GUI helps to compare the simulation very easily.

Currently, simulation software is widely available in the market but the commonly opted software for application is open-source so particularly Gazebo with ROS is most suitable. In [4] the author created a simulation in Gazebo which works in the ROS platform under Linux operating system. Here they have performed an obstacle distance calculation with the help of ultrasonic sensors which are already fixed while robot design. They have used RVIZ as data gaining software that can collect the data from the sensor and is used for further comparison steps. There are several plugins available in Gazebo[6]. The URDF file is used for the representation of robots and environments in ROS. In [5] the author has added many functions to the robot which are obstacle detection,3D laser finders, camera, IR sensors. Physical properties like friction, inertia have also been added to the modeling for the proper working URDF file in Gazebo.

From [2], [3], [4], [5] all authors have explained the modeling of the robot and how they have imported the model for analysis. The URDF is commonly used for implementing the

elements of the robot. URDF is quite good and accurate and the advantage of the URDF is that analyzing the model and importing it to open source software is easy.

2.2 Mapping/SLAM and Obstacle Detection

Mapping and obstacle detection are the most needful functions while simulating software because in this complex world some people are seeking help from google maps for easy and accurate journeys. But in the case of robots, it is quite complex and still facing problems like broken connections while using mapping. In [4] is stated that the results of simulation both in Gazebo and MATLAB prove the ability of the robot to adapt to different environments. The author, Julius Fusic et al. used sensors for obstacle detection and when the system has prepared physically the control communication was through Bluetooth, which can be difficult to use for long-distance robot control. A laser scanner can be used for the internal localization of the robots, and the laser scanner is well known for its accuracy and range [6].

The author explaining different methods that have been proposed by different people for the improvement of localization. Hone He et al. [7] proposed an algorithm of SLAM based on EKF and particle filter. These two technologies have a specific role in predicting localization. EKF estimates the environment while the particle filter calculates the poses of a mobile robot. The results after the tests were good and the error range decreased, the accuracy increased and one more thing is that it's very easy to understand. Graphical representation of SLAM is much more understandable but still, it has some shortcomings in case the texture is small the accuracy between the frames will be very low [8] [9]. Li Ruihao, the author of [10] developed a new monocular vision odometer system called UnDeepVO. This system is used to estimate the pose of the monocular camera and also the depth of the monocular image. The UnDeepVO has two characteristics the unsupervised deep learning method and the absolute scale recovery.

Simultaneous Localization and Mapping(SLAM) in the robotic field play an important role in the navigation of robots. SLAM algorithm has grown so far over all these years with the advancements of technology and deep learning algorithm.

2.3 ROS and MATLAB

MATLAB is an important tool for simulation, testing, and designing prototypes of robotic systems. In [11] the author performs a combined simulation with MATLAB and ROS. ROS is a developing environment in the robotic industry. The tool which creates the bridge between ROS and MATLAB is the Robotic system toolbox. Barber et al. in [11] also explain different algorithms used for the work. Probabilistic Road Map(PRM) and fast marching square are the two algorithms used in [11] for experimenting with robot path planning. PRM is a motion planning algorithm that works in 2 phases - the learning phase and the query phase. This algorithm is used where there is a high chance of complex computations are required. The other path planning algorithm uses a function that has a behavior like wave propagation in the fluid. ROS integration with MATLAB is a little complex but it is very interesting to see different tools used to redirect the messages from the robot while simulating and how MATLAB and ROS competing themselves.

ROS is a software framework for robotics software development. ROS contains a lot of Functionality. In [12] Corke et al. given an example of laboratory equipment that works with the integration of ROS and MATLAB. The author briefs how the messages are being sent and collected between ROS and MATLAB. One of the main advantages of MATLAB and ROS is its platform independence. The update rate of the messages will depend on the size and complexity of message processing. Rather than programming, MATLAB has its Simulink block diagram modeling environment. it is also easily understandable than complex programming languages. The open-source middleware ROS and Simulink has widely used in autonomous studies and also for the simulation of smart cars nowadays.

Shota et al. [13] used the Autoware toolbox which is a benchmark suite that contains MATLAB codes and Simulink models. Autoware collects information from different modules such as sensing, localization, detection, motion planning. For different modules the way of collecting the information is different. The nonuniform voxel Grid filter node uses a nonuniform box grid filter. The fog rectification node converts foggy images to defogged images. The road information can be obtained by using sensors such as LiDARs and cameras. The data of the lidar scanner is processed by the sensing module. In the case of localization, the module compares the data from the sensing module and 3D module which is obtained in advance.

In [13], [12], [11] the integration between ROS and MATLAB is explained. MATLAB has a Simulink block diagram modeling environment that can use instead of complex programmings. Different algorithms and different tools make the integration of ROS and MATLAB easier.

2.4 Kinematic and Dynamic Modeling

In [14] Gonzalez et al. discussed the kinematic and dynamic modeling issues while a robot model is defined as mathematical expressions. The main issues were obtained while estimating robot location and implementing software simulators. The author discussed two models - one is a car-like robot and the other one is a differential drive robot. Ackerman vehicles are the most commonly used car-like vehicles. The car-like configuration leads to circular paths when the vehicle turns, these circular arcs are centered at a point known as the Instantaneous Centre of Rotation (ICR). Ackerman vehicle robot configuration has been made according to the properties like angular velocity, steering angle, forward speed.

The angular velocity of the robot is calculated by

$$\theta = \frac{v}{R_1} \tag{2.1}$$

where θ - vehicle orientation concerning the x-axis,

v - forward speed of the robot (imposed by the rear wheels),

R_1 - turning radius.

The turning radius is calculated by

$$R_1 = \frac{b1}{\tan \alpha} \tag{2.2}$$

where b1 - length of the vehicle or wheelbase

α - The steering angle.

The steering angle is mechanically constrained and its maximum value influences the robot motion. The important feature can be analyzed with the tool proposed in [14] because the user can change the maximum angle that the steering wheel can turn. The differential drive robot configuration is the easiest mechanical configuration because only two parallel driving wheels mounted on an axis are enough to move the robot.

Apart from this mathematical calculation Singla et al. [15] proposed a concept of D-H notation which is a powerful tool for giving the geometric description to an open kinematic chain. However, ambiguity in it is observed by Seth and Uicker in the case of closed robots. Conventional kinematic studies of serial manipulators involve the proper selection of coordinates frames of reference at proper positions. In the past, it was the D-H

algorithm used for assigning the coordinate frames. When there are two spatial links with two joint axes at the right angle to each other, the forward kinematics derived with the DH algorithm comes geometrically inconsistent. A typical spatial link that involves two consecutive joints axes at the right angle leads to the limitation in the D-H algorithm. However to overcome that dummy frames are proposed to avoid the deficiency.

The traditional D-H method allows the coordinate system to be established on the extension line of the axis, which will lead to problems such as in some case the model's parameters are difficult to obtain, the established model does not match the object which will lead to wrong local joint position analysis and it increases the difficulty of obtaining dynamic parameters. D-H representation with the coordinate system fixed on an object. Guo et al. [16] Compared with the traditional D-H method, this method can model all joint linkages and establish the model that is consistent with the entity. At the same time, the disadvantage is that the complexity of kinematics calculation is slightly increased due to the increase in parameters. In summary, ED-H improves the standard DH convention, which can make the robot model simulation more intuitive and accurate, and has a good guiding significance for engineering practice. It provides a good theoretical guidance basis for the research and design of joint manipulator arm products.

2.5 Gazebo and Turtlebot Simulation

In [17] the authors explained the turtlebot simulation in Gazebo. Turtlebot is a handheld robot with wheels. Because of its low cost and open-source nature, this robot kit is ideal for researchers[17]. The ROS command '*$ roslaunch turtlebot bring up minimal. launch*' can be used in the Ubuntu (Operating System) terminal to start the Turtlebot in the real system. For the simulation in Gazebo, the author created a simple maze (only boundaries) in the Gazebo environment for simulation by adding four jersey barriers and a Turtlebot, as shown in Fig 2.1. The maze shown in the figure is used to evaluate the algorithm which the author explains in [17].

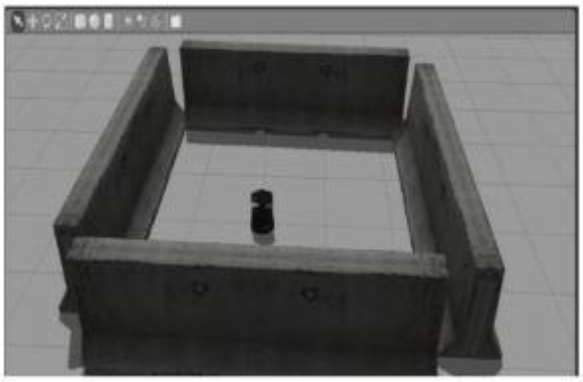


Figure 2.1 Maze environment in Gazebo[17]

2.6 Conclusion for the Literature Review

- The simulators are getting updated with new features day by day. Restrictions are getting lesser and lesser before Gazebo's main requirement was Linux as an operating system but new plugins were released and can be used in windows.
- The use of visual sensors and image processing algorithms improves the mapping technique more reliable.
- The Boe-Bot simulation could be done in Gazebo after designing the model in URDF. CAD Modeling is difficult to import in different simulators.
- The interface performance may be improved by incorporating more functional buttons in the RVIZ plugin.
- Implementing the complex function and design in a robot model makes simulation difficult. The integration with ROS makes the implementation easier and comfortable.
- The tools and types of algorithms are a viable option for the implementation and development of simulations.

2.7 Objective of the Research

The purpose of this thesis is to design a Boe-Bot with different functions and simulate it in the Gazebo environment. The main objectives of this work are:

- Modeling the Boe-Bot robot with the required dimension and create a world for line following and maze following simulations.
- Designing the robot with capabilities of obstacle detection, real-time vision, and mapping with the help of OpenCV, ROS using a laser sensor, ultrasonic sensor, and also the camera feeds.
- Importing the designed robot model into Gazebo environment and simulation of Boe-Bot using ROS in Gazebo.
- The secondary objective is to create an interface between MATLAB to the Gazebo simulator using ROS Toolbox.

3 METHODOLOGY

This chapter will focus on the software that was selected and the different file systems that were used in the project. The different programming languages used for the simulation part are also discussed in this chapter. The final section of the chapter will explain the detailed modeling of the Boe-Bot and the properties of each part. As discussed in Chapter 2 different tools and algorithms are a viable option for the implementation and development of simulations. For the simulation of the Boe-bot, I need to design it first. The mechanical aspects of design and the configuration of the Boe-Bot in the simulation will be provided in Chapter 4.

The proposed solution for exporting the design to a unified robot description format is executed using the fusion to URDF add-ins. The solution for line follower Boe-Bot simulation is described in section 5.2. Arduino camera is used for capturing the real-time image and with the help of OpenCV, the visual transformation is processed. The solution for creating an interface between MATLAB and the robot operating system could be done using ROS toolbox detailed explanation is given in section 5.5.

3.1 Hardware Selection

The main hardware used in the model is sensors and a camera. Laser sensor used in the line follower environment and ultraviolet sensor used in the wall following environment. In this task, I used a Raspi camera that can capture the video streams to follow the line.

3.1.1 Raspi Camera

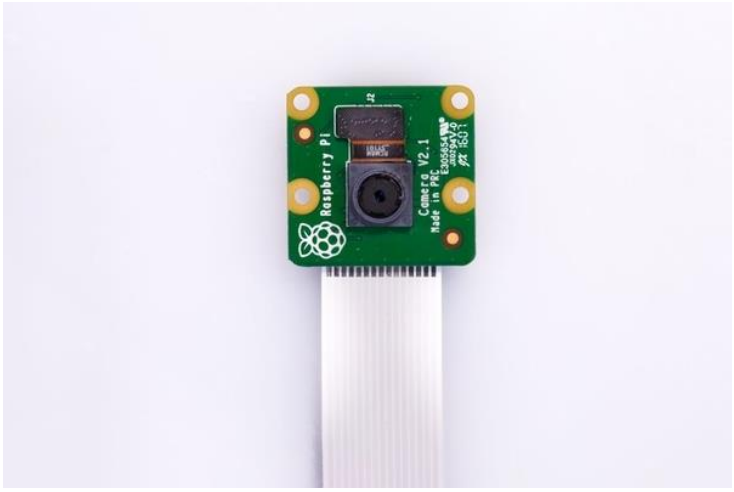


Figure 3.1 Raspi camera [18]

The MIPI CSI camera format is used by RPi Camera modules to save power, increase bandwidth, and fit into a smaller physical space. As compared to USB cams, these modules have higher resolution, a faster framerate, and lower latency. This is because USB 2.0 is too slow for video capture at resolutions higher than 1280 x 720 at 30 frames per second, and USB 3.0 is also very costly. The slow-motion recording is also assisted by RPi cams at 640 x 480 @ 90 frames per second, which is useful for real-time surveillance [19].

3.1.2 The Laser Proximity Sensor

The laser proximity sensor detects objects by sending laser beams on them and detects the reflected light. Inside the detection range, this proximity sensor (obstacle/object detector) can detect very small objects. This is due to the laser light ray's intense focus and near-zero separation over short distances. The digital output signal from the laser proximity sensor module is attached to the Arduino digital IO pin. The digital IO pin value will be read by the software and displayed on the connected MAX7219 seven-segment display (obstacle / free indication), as well as printed through the serial port for further debugging.

Figure 3.2 Laser Proximity Sensor [20]

A laser source (transmitter), a laser detector (receiver), and signal amplification are all part of the sensor (conditioning circuit). The shockwave is produced by an oscillating tube with a frequency of 180 kHz. The shockwave is applied to the laser tube for excitement after being amplified by a transistor. The reflected light can be received by the receiver in the receiving tube, which is matched to the oscillating tube. Since the laser sensor uses modulation processing, the receiving tube can only receive reflected light of the same frequency, effectively blocking visible light. The sensor's effective measurement distance ranges from 0.8 to 1.5 meters. Obstruction detection, pipeline counting, and proximity detection are all possible applications. VCC, GND, and DOUT are the three signals on the laser proximity sensor [21].

3.1.3 Ultrasonic Sensors

For the wall following the Boe-Bot model, I chose an HC-SR04 ultrasonic distance sensor that can report the range of objects up to 3 meters away which is extremely useful knowledge while trying to keep the robot from colliding with a wall. They are low-power for Arduino board using cheap batteries, simple to use, and they're hugely popular among hobbyists. It also has the added benefit of looking amazing, like a pair of Wall-E Robot eyes for the Boe-Bot.

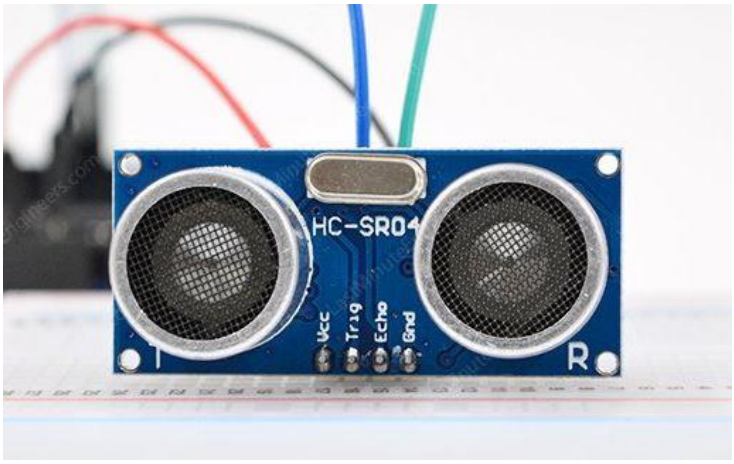


Figure 3.3 Ultrasonic Sensor [20]

The HC-SR04 ultrasonic distance sensor is consists of two ultrasonic transducers at its center. The one transducer serves as a transmitter, converting electrical signals into ultrasonic sound pulses at a frequency of 40 kHz. The receiver receives the pulses that have been transmitted from the object. If the reflected signals are received, an output pulse is generated whose width can be used to calculate the pulse's travel distance.

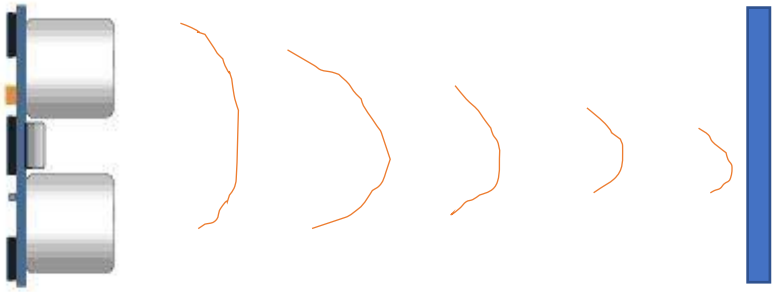


Figure 3.4 Working of Ultrasonic Sensor [22]

3.1.4 Arduino Uno Board

A wide range of microprocessors and controllers are used in Arduino board designs. There are 14 input/output pins, 6 analog inputs, 16MHz crystal oscillators, USB connections, a power jack, and a reset button on this board. The Atmega328 microcontroller is used in the Arduino Uno.

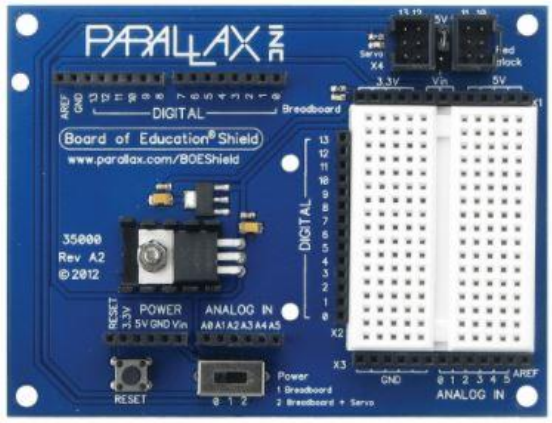


Figure 3.5 Arduino Board [20]

The Arduino Uno can be powered either by a USB or by an external power supply. However, the pin could provide less than 5V, making the board unstable. The voltage regulator can overheat and harm the board if more than 12V is used in Vin pin.

3.2 Simulator Selection

Since the whole thesis is based on different simulators and tools used for the simulations it is necessary to select the compatible software for my use and it will be discussed in this section.

3.2.1 Fusion 360

At the beginning of Boe-Bot modeling, SolidWorks was the priority for designing. The designing of certain parts and exporting to the required file format was very complex in SolidWorks. Because of this, SolidWorks was substituted to Fusion 360.

Fusion 360 is design software that is a product of Autodesk. One of the main reasons for choosing Fusion 360 is that it is an affordable platform for students and teachers. So, it is possible to engineer the designs with a complete set of 3D modeling tools that include parametric, freeform, direct, and surface modeling.

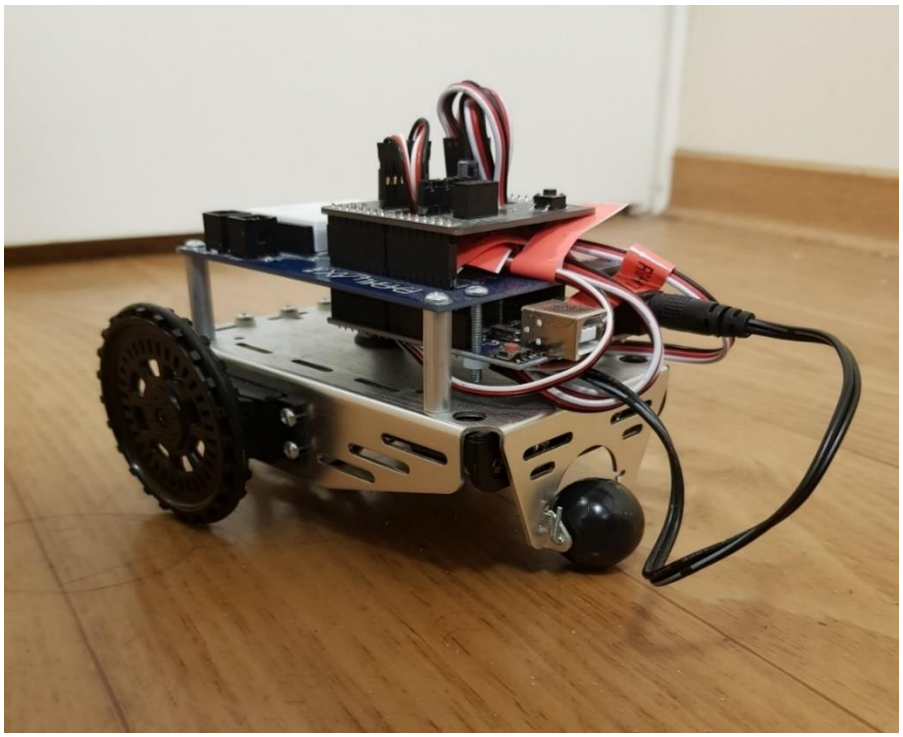


Figure 3.6 Real Boe-Bot

The design that was made using the Fusion 360 can be easily exported to URDF file format or STL format. Therefore, the exported URDF file can be directly used in any simulator for robot simulations.

The materials of each part like Motor, chassis, or tires can be assigned with specific names that are close to the physical world. The physical properties like area, density, volume, and also some of the properties like the center of mass, moment of inertia can be assigned for each part so that the robot will be stable.

3.2.2 Development System Setup

The Ubuntu 20.04.2.0 LTS operating system was used in the simulator development for the Boe-Bot simulation project (Trusty Tahr 64-bit). Since it is compatible with most of the development tools and packages used for this project, this is a long-time support version of Ubuntu. It's also a free and open-source program that can be downloaded from the Ubuntu website. It is recommended to use a 64-bit computer with a high-performance graphics card such as NIVIDA GEFORCE, at least 2 GHz CPU speed, 4GB system memory, and 25GB of free hard drive space to have a good simulator environment for 3D simulation in Gazebo, fast communication in ROS, and computation in MATLAB.

Table 3.1 Configuration of Computer

	System and software	IP address	Subnet mask
Software environment	Ubuntu 20.04.2.0 64bits; ROS Noetic; Gazebo 11.0	127.0.0.1	255.255.254.0

3.2.3 Gazebo

The Gazebo is one of the popular simulators in robotic industries. As we know the Gazebo is a 3D simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. In reality, it is very similar to game engines like unity but produces better simulations and offers a suite of sensors and interfaces for both users and programs. Data visualization using the RVIZ tool, simulation of complex environments, and reverse engineering are the main features that make Gazebo unique from other simulation software. The main components of Gazebo are:

A. World files

The world file is specifically used for simulating background, camera pose, simulation step size, and camera pose.

Table 3.2 Specifications of engine

Property Name	Value
Physics Engine	Open Dynamics Engine(ODE)
Real-Time Factor	0.94
FPS	47.06
Gravity	-9.8m/s^2
Models to spawn	Boe_bot description

- Maze follower world

According to the idea of developing the simulation, the SDF file, images, and scripts are used for the creation of the world. The world has been designed in the Gazebo model editor.

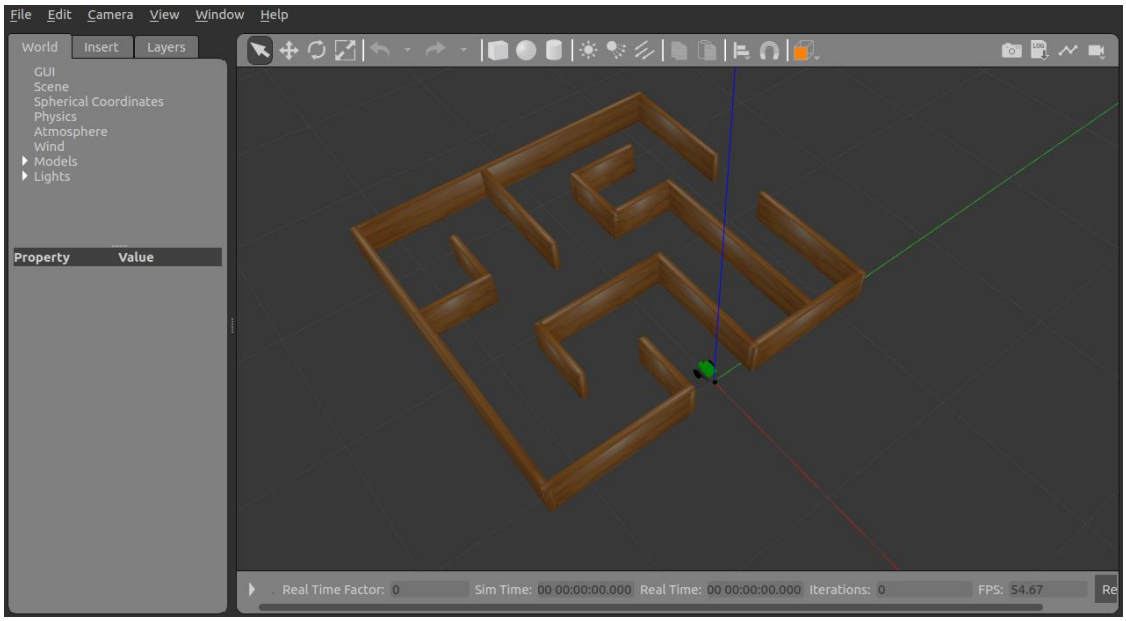


Figure 3.7 Maze follower environment

- Line follower world

For creating a line follower world the importance for the clarity of the black line is very important so that the camera can recognize the line very easily. Firstly, drawing the design in Blender [23] which is a drawing software, and converted it into the png image format file. Again this file has been imported in Gazebo world editor for further modifications and exported as an SDF world file.

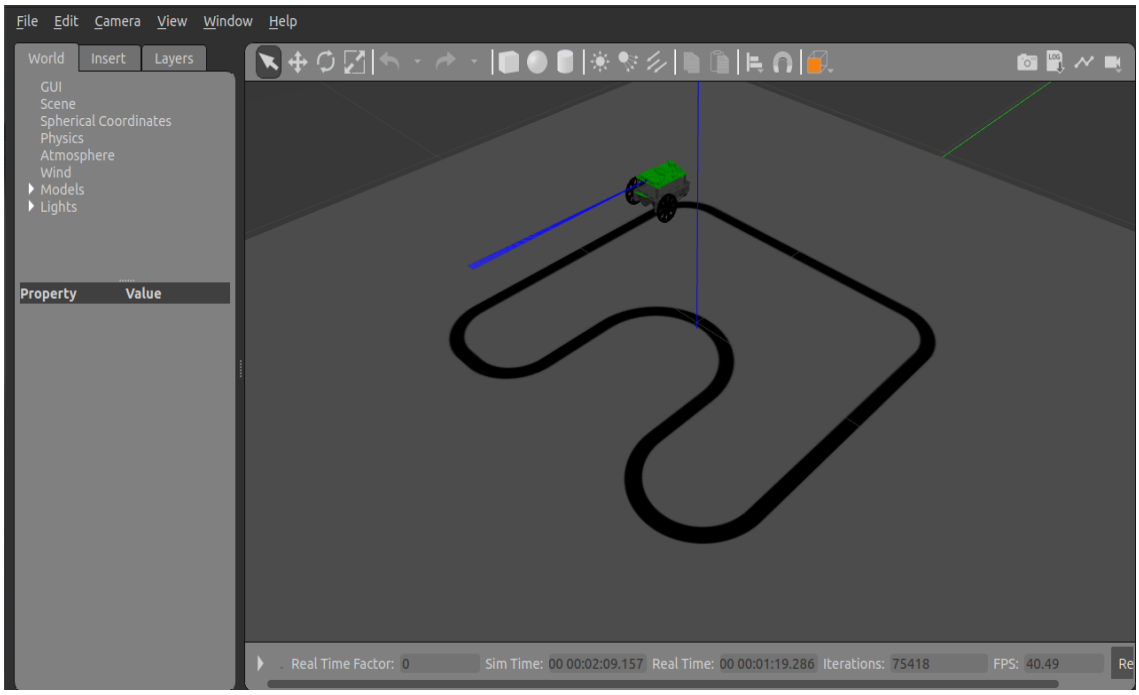


Figure 3.8 Line follower world

B. Models

The gazebo has its library which contains different models. The models are defined well and can be imported to the world for further simulation. An example model is given in fig.

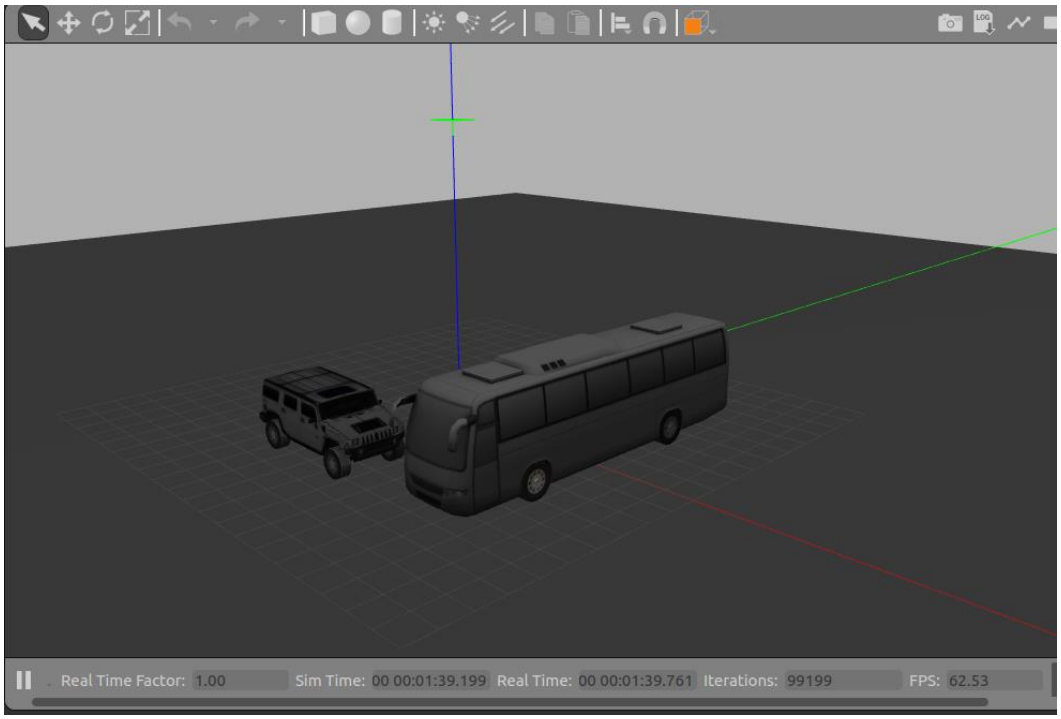


Figure 3.9 Example of Models in Gazebo

C. gzserver

The gzserver reads the world file to generate and populate a world. The gzserver initializes the updated loop of the physics and generation of sensor data. gzserver is the core part of the Gazebo and can be used as a graphical interface.

3.2.4 Overview of Gazebo Plugins

A plugin is a piece of code that is compiled into a shared library and then used in the simulation. Via standard C++ classes, the plugin has direct access to all of Gazebo's features. The key uses of plugins are given below:

- The Developers can control almost all aspects of the Gazebo.
- They are self-contained routines that can easily be shared.
- It is possible to be instantiated and removed from a working system.

- It is possible to programmatically alter the simulation, for example, move models, respond to events, insert models, etc.

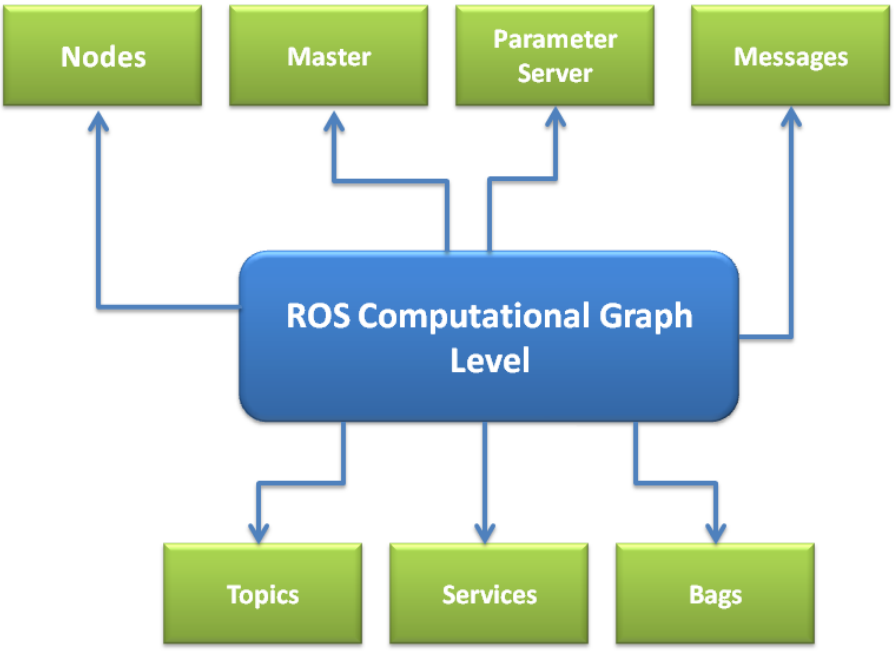


Figure 3.10 The ROS computation graph of the Model plugin [24]

When a robot model is spawned, the model plugin starts automatically. When the robot model 'Boe-Bot' is spawned, the ROS tool RQT graph creates and visualizes a computation graph. The nodes, which publishes (represented by an arrow pointing outward) and subscribes (represented by an arrow pointing inward) to many topics enclosed by small boxes.

The topics are generated by ROS packages, a ROS package that provides wrappers around the standalone Gazebo, allowing Gazebo to fully use ROS messages, services, and dynamic reconfiguration. The titles of all the topics are self-explanatory.

3.2.5 Advantages of Gazebo

Gazebo is a 3D simulator and it has some advantages over other simulators.

- It is possible to define completely our robot in 3D software.
- Sensors such as laser range finders, GPS, Pan-Tilt-Zoom cameras, and other components are simulated.
- Pioneer2DX, PR2, Care-O-Bot Robot, Kobuki Robot, and Nao Humanoid Robot are some of the predefined robot types.

- This is also free software.
- We may add, delete, or alter our environment and models, as well as include multiple robots in our simulation.
- We can also use the physics engine to simulate different materials' behavior.
- It runs on Linux and supports plug-ins written in the C++, C, and Python programming languages.
- Using the Gazebo ROS packages bridge, we can access resources such as models, source code, and plugins for devices that are available within ROS.

3.2.6 ROS

When the sensors and other sensing components are connected to the software the data transfer cannot be done directly because of the difference in a programming language or for some reasons here is the role of middleware comes to play. As the name suggests Robot operating system is not a properly operating system, but it is a set of tool or framework which act as an operating system on a heterogeneous computer cluster.

- Build systems

Ros made some changes to the groovy distribution and found a different way to compile the software. Before, exciting distributions used a build system called rebuild, but the latest versions have already replaced rebuild with a new build system which is called catkin. Catkin also acts as a workspace for ROS thus catkin workspace.

- Packages

ROS contains many packages of its own, each package provides different functions for hardware abstraction, drivers, the communication between different processes over multiple machines. Each package is manifested as a package.xml file. This file gives information about the package including its name, version, and dependencies. The directory where the Package.xml file contains is called the package directory. This directory stores most of the package’s files.

- ROS Master

The basic goal of ROS is to lift the robotics in designing software as a collection of small independent programs called nodes that run at the same time. The ability of nodes to

communicate with each other makes this possible. The part which provides the facility for communication of the nodes is called the ROS master.

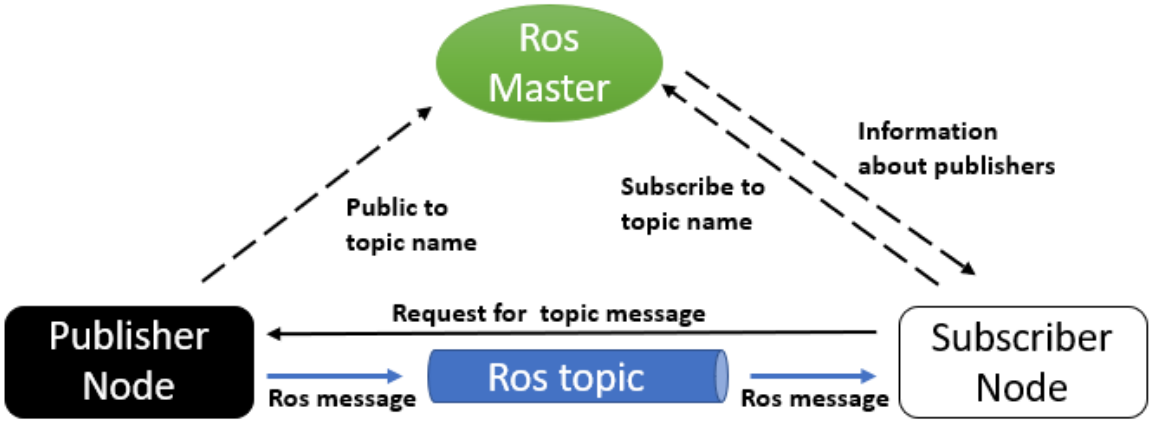


Figure 3.11 ROS node communication [25]

- ROS Nodes

It is a single-purpose, executable program, we can compile them individually and also can manage them individually. So they are the building blocks of programs that we are executing and nodes are organized in packages. ROS nodes communicate with each other through ROS Topics, So one node can publish and subscribe to a topic, and essentially a topic is a stream of messages that can be transferred between the notes.

In the figure given above the Publisher, the node sends a message to the subscriber, then the publisher publishes on a topic name. On these ROS topics, the nodes publish messages, essentially the messages find the data structure of information that flows from one node to the others. It compromised of a simple type such as integers, float, Boolean, strings, etc.

3.2.7 ROS and Python codes

The main task for this thesis is to develop an environment for students so that they can test their python codes and see how their code works on Boe-Bot. The simulation is on three different world environments and the functionality of the Boe-Bot in each environment is different. Python3 is the used version in this simulation work with ROS. ROS node which helps for the data exchange which allows python commands to be sent to Gazebo simulation and receive feedback.

3.3 ROS Tool Box

ROS toolbox allows us to connect MATLAB and Simulink to the ROS. Both ROS and ROS2 can connect with the ROS toolbox for the design and development of robotics and autonomous system. It is possible to connect to the existing ROS network to explore available topics, services, and actions. This allows to send commands to and receive data from any simulators, hardware, or software nodes on the ROS networks.

ROS toolbox provides utilities to work with data types like laser scans, images. A recorded data in the rose bag file can be read by ROS toolbox for post-processing or playback data as simulation input. MATLAB/Simulink and various add-on products provide tools for designing and implementing algorithm components for control, perception, logic, and decision making. ROS toolbox allows connecting ROS-enabled simulators and hardware for testing these components as desktop prototypes.

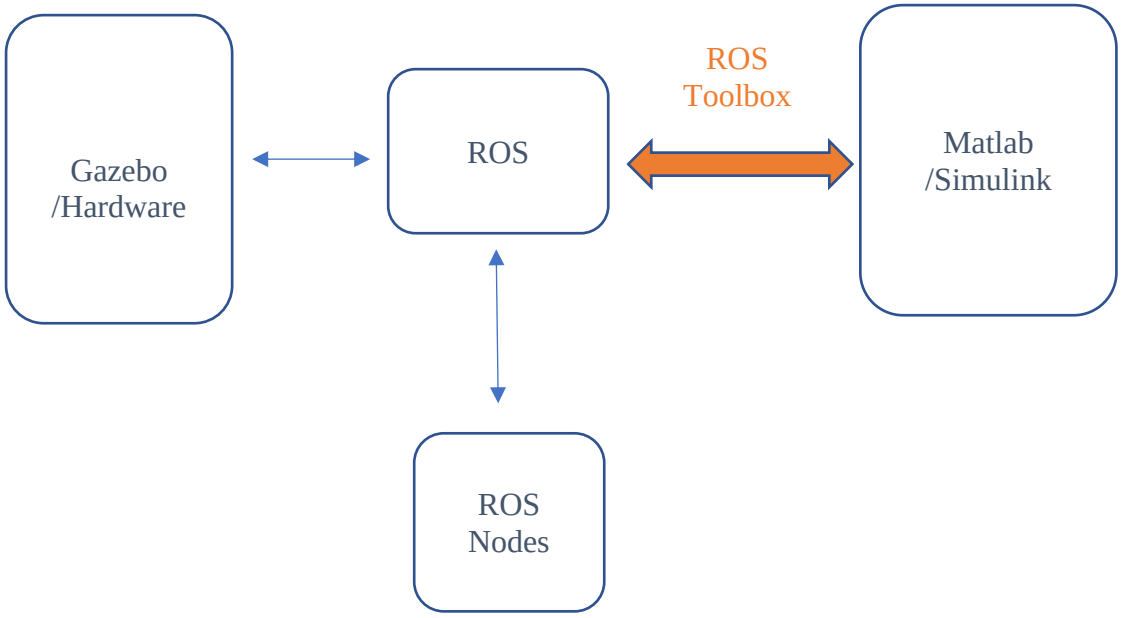


Figure 3.12 Interface using ROS toolbox

After the initial testing, we can automatically generate C++-based executable ROS nodes from the designs. These generated nodes integrate with the ROS and ROS 2 build systems So they could be run on the target system with no dependencies on MATLAB/Simulink. After deploying algorithm components as part of the overall distribution system MATLAB/Simulink can remain connected to the ROS network for interactive design tasks such as Data visualization and parameter tuning.

With ROS it is possible to reuse the software across different simulation and hardware environments, this way once if tested the algorithm in simulation the same MATLAB code and Simulink models can run on the hardware with minor adjustments.

3.4 URDF and XACRO files

URDF (Unified Robot Description Format). URDF is the format ROS uses to store robot models or that URDF is the simulation format of ROS. In actual URDF is the central piece of ROS and is used in many of its packages. It is possible to use ROS without URDF, but it will make many parts of it does not function optimally or at all. Technically, URDF is a domain-specific modeling language based on XML, that allows us to encode the kinematics, some parts of the dynamics, and various other pieces of metadata related to a robot in a format that is readable and writable to a certain extend by both humans and machines.
In other words, we can say that the URDF is the file format ROS uses to describe the layout of the body of a robot with all its links, joints, shapes, and colors. It also allows us to store pieces of extra information, such as the range of motion of all joints in the robot, and how fast it can perform those motions. It is a domain-specific language because it uses names and terminology specific to the robotics and robot modeling domains. This allows us to quickly create robot models using the actual names of the parts of the robot instead of having to resort to generic descriptions.

3.4.1 Implementation

On this level, a URDF file is just a text file containing XML tags: specific keywords that ROS recognizes as being part of URDF. Some of those ML tags refer to other files or 3D models of parts of robots, which allows us to make use of external files to quickly incorporate a detailed shape complete with colors from a 3D mesh file, such as an upper or lower arm. The bulk of a URDF file is made up of elements called links and joints.

These directly correspond to the links and joints of a robot: links give robots their shape, joints connect links and determine how they may move concerning each other. So, a robot in URDF is a set of links connected by joints in a certain order. If that order changes, the motions a robot can make changes as well, so not only the body layout changes by changing the way joints connected links, but also the behavior of the robot.

3.4.2 Joint types

- Fixed joints

  These joints rigidly connect with the parent and child, making motion impossible.
- Revolute joints

  These joints allow rotation of the child concerning the parent, but only in one dimension. They are commonly used to model industrial robots.
- Continuous joints

  These are the same as revolute but without any limits. They can rotate an infinite number of times.
- Prismatic joints

  Only the translation of the child concerning the parent in one dimension is allowed.
- Planar joints

  These are two-dimensional variants of prismatic joints.
- Floating joints

  These joints allow motion and rotation in all directions without any limits whatsoever. Connecting a drone to the rest of a URDF world could be done with this type of joint.

ROS has a system in place called XACRO that solves most of the limitations of URDF files. XACRO supports macroing, which can be explained as programmatic URDF generation. It allows the creation of stand-alone URDF snippets, called macros, that can be added multiple times to a scene by inserting just the name of the macro. The XACRO system will take this name and replace it intelligently with the full URDF snippet. So instead of directly copying and pasting URDF snippets ourselves, Xacro can do this for us and much more efficiently and like using templates.

The best thing about XACRO macros is that they accept parameters, just like c++ or python functions. This is a very powerful system, as XACRO macros can call other macros, as well as using mathematical and python expressions to generate URDF snippets. Xacro macros do not have to be defined in the same file as they are used in. With a special include tag, macros defined in other files can be imported, after which they can be called with any arguments they require.

This makes adding multiple robots to a single scene much easier, as instead of having to copy and pasting everything into a single file, only an import statement and a macro invocation are needed. But the XACRO macro files and statements cannot be directly read

by URDF parsers. So, we must convert the XACRO file to a URDF file with the Xacro tool. The Xacro tool not only converts the file to valid URDF but also checks whether the structure of the XACRO is legal and gives spontaneous feedback. If everything in the conversion process is well, it will generate the corresponding URDF file which can be used as normal with all ROS tools.

3.5 Simulation Tools

Through the graphical and command-line utilities, the ROS tool takes advantage of the introspection capabilities.

3.5.1 Command-line Tools

There are different commands for initializing a group of nodes. One main thing about ROS is that it can be used without a GUI. All core functionality and tools are accessible with the help of command-line tools. Introspecting topics, services, and actions like recording and playing the data, all these services can be launched through the command line. There are other tools available if someone prefers only graphical tools RVIZ and RQT provide the same features.

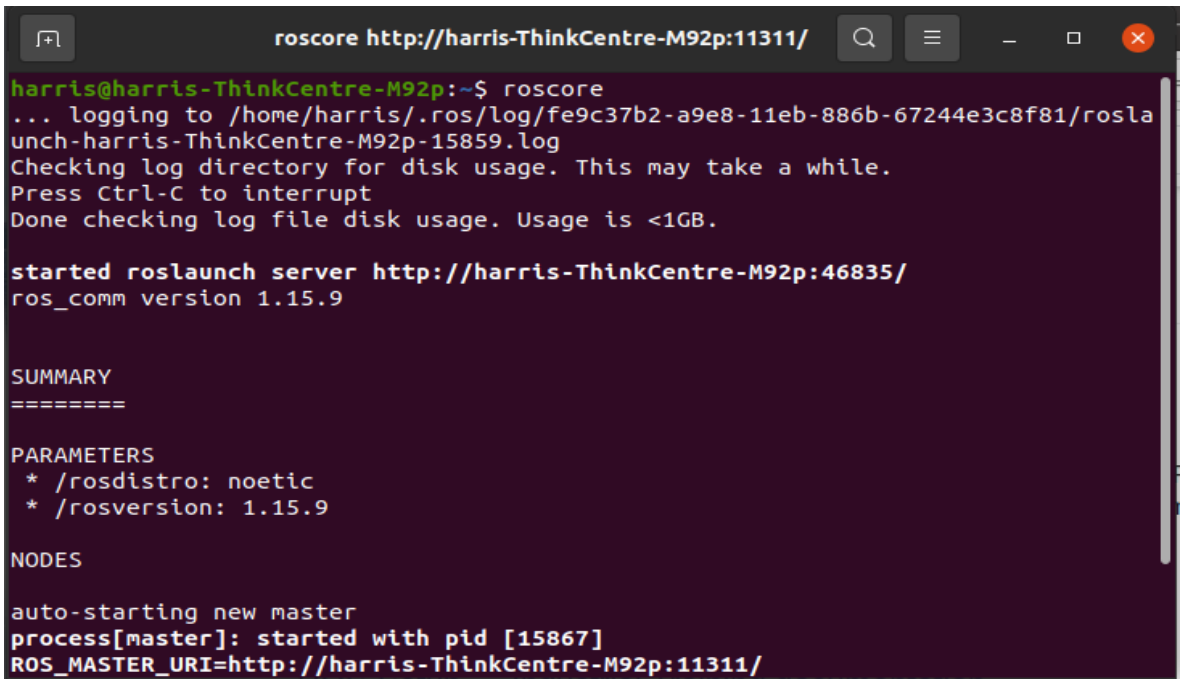


Figure 3.13 Command-line tool

3.5.2 RVIZ

RVIZ is a Robot visualizer program that is used as a data acquiring software that collects data from sensors and stores it for comparison with subsequent operations. The current environment and sensor data configuration model of a robot can be viewed using this 3D visualizer. The RVIZ is also used to measure and quantify real-time data from sensors such as encoders, cameras, displacement sensors, ultrasonic sensors, and other route planning sensors. The RVIZ window can be used to implement 2d and 3d navigation on the model. In RVIZ, odometry will be examined. RVIZ is an excellent tool for determining what went wrong in a vision system. Each item on the left-hand list has a check box. We can immediately reveal or conceal some visual details.

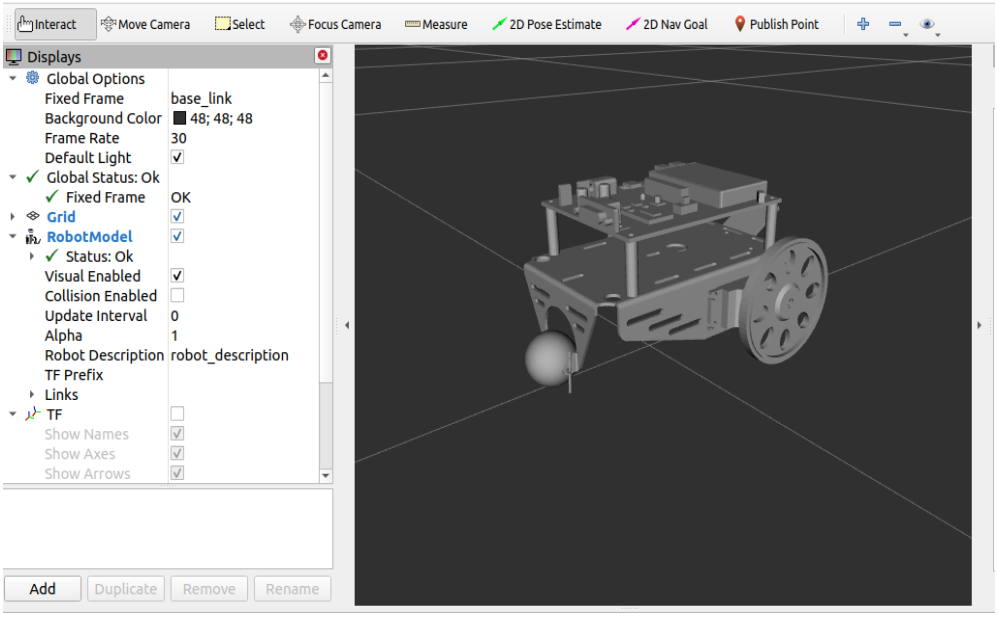


Figure 3.14 Boe-Bot model in RVIZ

3.5.3 RQT

RQT stands for a Qt-based framework. Developing graphical interfaces is the main function of RQT. It is possible to create custom interfaces by configuring the built-in plugins that contain extensive libraries. By writing own RQT plugins it is possible to create new interface components. RQT tool is a debugging tool in ROS. An RQT tree is a tool that allows us to plot the data from the topic data that is being published on a specific topic.

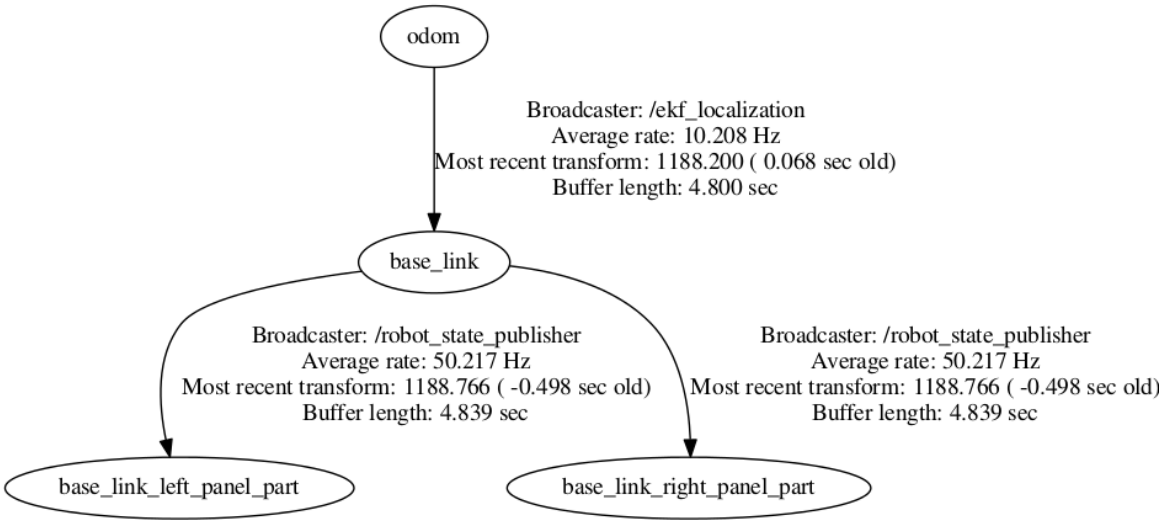


Figure 3.15 Basic RQT model of odometry

RQT tool is commonly used to view the nodes and topics that are actively running. With the help of RQT plot plugins, it's possible to monitor encoders, voltages, or any properties that can be represented as a number that varies over time.

3.5.4 tf on Boe-Bot

tf is a library that runs inside the ROS which keeps track of all the coordinate frames within the robot. It is not a centralized service but it is a standardized protocol for publishing transform data to a distributed system with the help of nodes. There are publishers and listeners in this library. The tf library has many uses while dealing with the static and dynamic properties of Boe-Bot.

While transforming multiple times there will be no data loss because every frame has an ID so it will be easy to keep a track of the data and it is more convenient to find out which frame is associated with the data. Information about the past locations of the data is also stored and it can easily access, but it is not possible before the recording has started.

- Helper Methods

These are methods that will receive the tf data, read the header to get the time stamp and the frame ID and it can also apply the transform to the received data. These are defined on the datatypes, messages, and tf datatypes inside the ROS.

4 MECHANICAL ASPECT OF DESIGN

In this section, the physical features of Boe-Bot namely, the chassis (base link), the drive mechanism, sensors, and electronics are described. The last section of this chapter describes the fusion to URDF conversion.

4.1 Base Link/Chassis

The chassis act as the base and frame of the whole robot. Since it is the base and takes the whole weight of all other components, it must be strong. After examining the physical Boe-Bot the material steel was chosen due to its good strength to weight. The chassis is a rectangular enclosure with four steel pillars. The base link is the parent component where all other child components are mated and joined so that they could be exported as a single body.

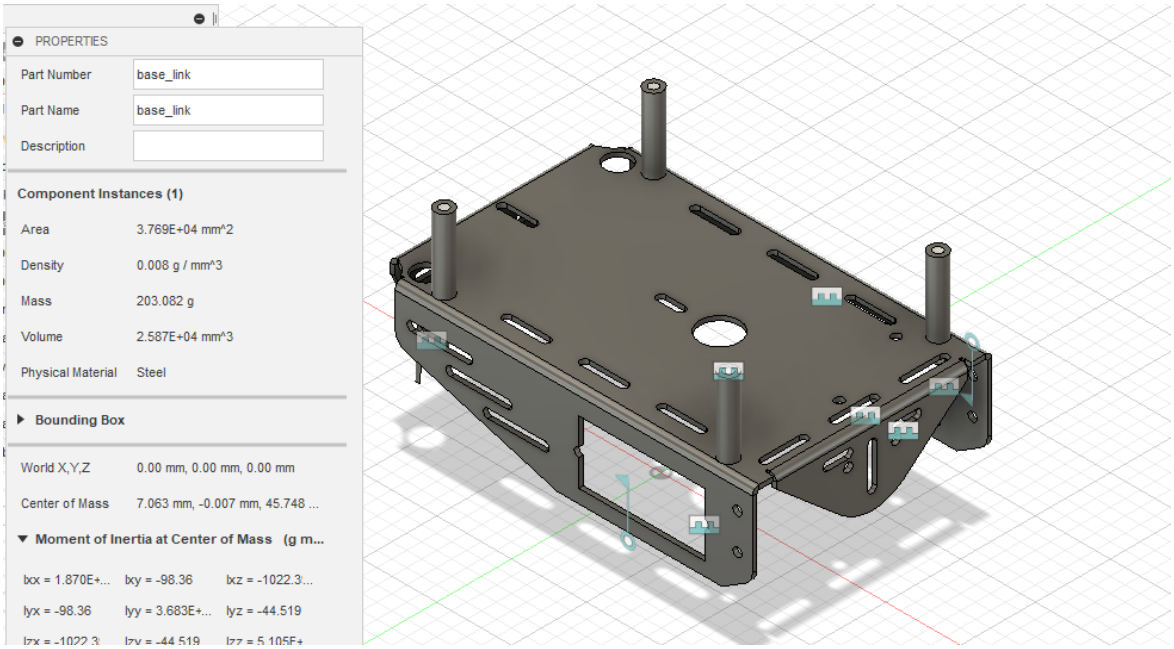


Figure 4.1 Base link

4.2 Differential Drive System

The differential drive is a drive system that has independent actuators for each wheel. It means as the name suggests the motion vector of the Boe-Bot is the total sum of the independent wheel motions. In our design, the wheels are placed on each side of the

robot with the help of two servo motors and A non driven wheel which is called a caster at the backside of the Boe-Bot.

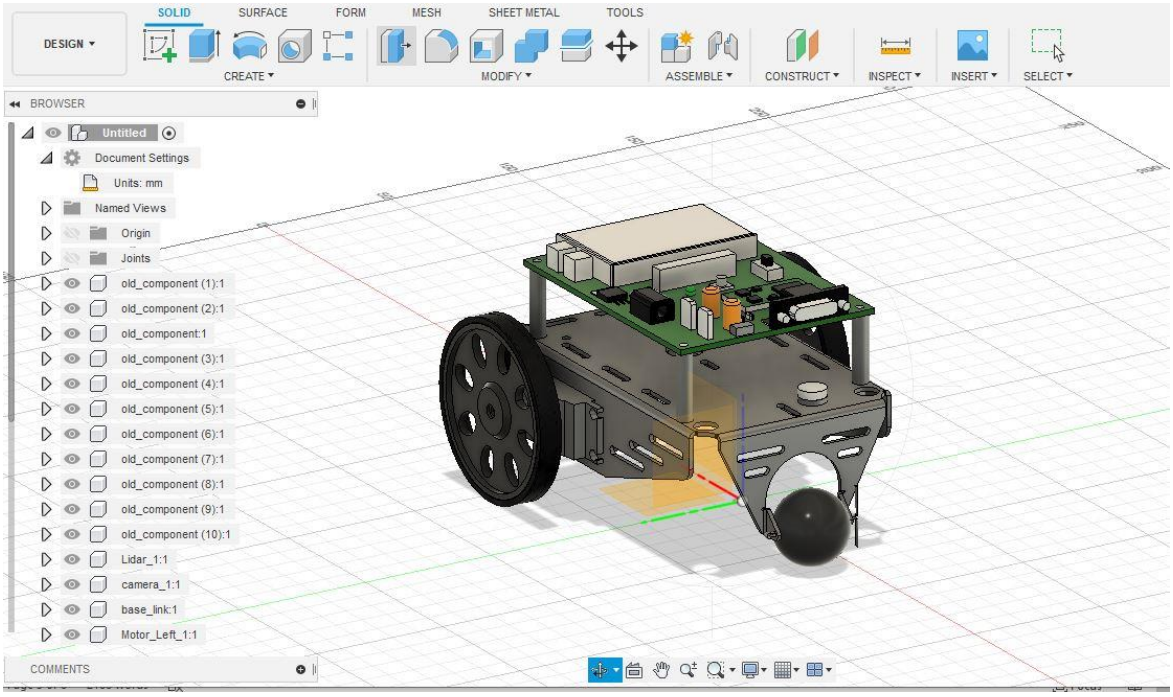


Figure 4.2 The caster and two wheels.

A straight-line motion will only work when the turning drives wheels at the same rate in the same direction. For rotation, the wheels should be driven in opposite directions at the same rate. Arbitrary motion paths can be accomplished by modifying dynamically the angular velocity or the direction of the drive wheels. The drive complexity is reduced by creating an alternative sequence of straight-line translations in motion paths.
The appearance, material used, and properties are given for a stable design.

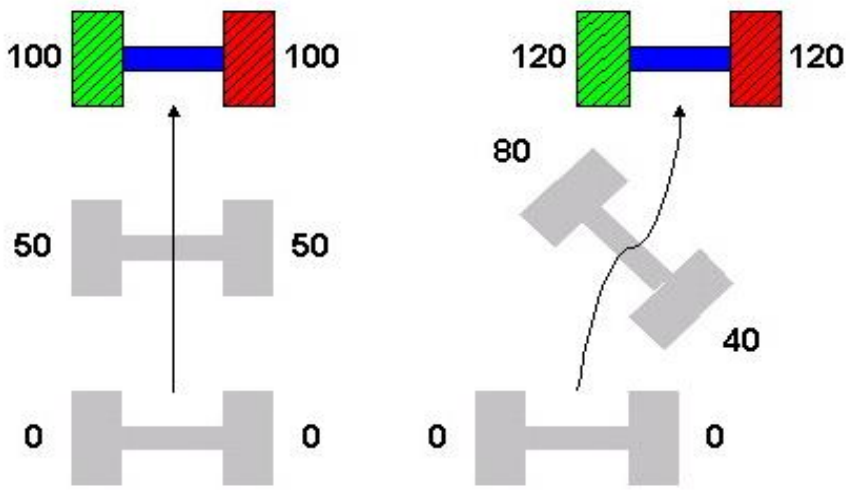


Figure 4.3 Differential drive motion paths.

The fig shows different motion paths for the differential drive of the robot where two separate motors are used for actuating each wheel. In the case where the encoder values are equal, the robot travels a straight line. When the values are different for each wheel that is when the left wheel shows an encoder reading 80 and the right wheel shows 40 meaning that the robot was executing a right turn.Unless the robot. While controlling the software if we consider only the end position, it would think that the robot moved in a straight line. This is one of the reasons why the slow encoder process causes errors. So the odometry reading should process as fast as possible.

4.3 Odometry

To know the location of a robot is very important, especially in the case of the Maze follower environment. People have an excellent vision system and memory to know where they are. But in the case of robots, they have to determine the position in their environment which is called odometry. It is the use of motion sensors to check the robot's change in position relative to some known position.

If the Boe-Bot is traveling a straight line and the diameter of the wheels is already known by it, then by counting the number of wheel revolutions it can be determined how far it has traveled. Odometry is commonly known as a sensor for mobile robots. Since wheel rotation sensing is available on all mobile robots, odometry is convenient. Odometry errors tend to accumulate over time, though, due to slipping or skidding of the wheels and numerical integration error. For this reason, odometry is usually supplemented with estimation techniques using exteroceptive sensors, like cameras, range sensors, and GPS.

In the case of Boe-Bot, it is a three Omni wheel robot, it has a differential drive system that has two individual motors and a caster for free movement. Figure 4.5 shows the RVIZ visualization of odometry of Boe-Bot while moving. We can see the joints and axes.

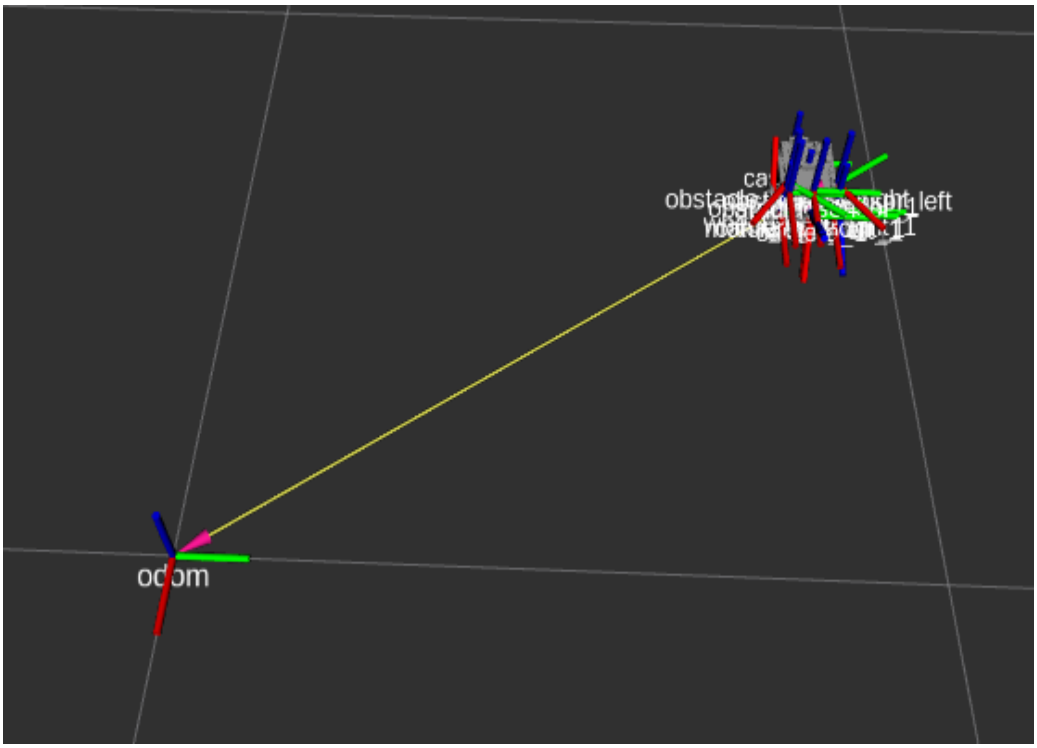


Figure 4.4 Details of Odometry through RVIZ visualization.

4.4 Sensors and Electronics

The major electronic components on this Boe-Bot are the sensors, camera, motors, and Arduino board for the connection.

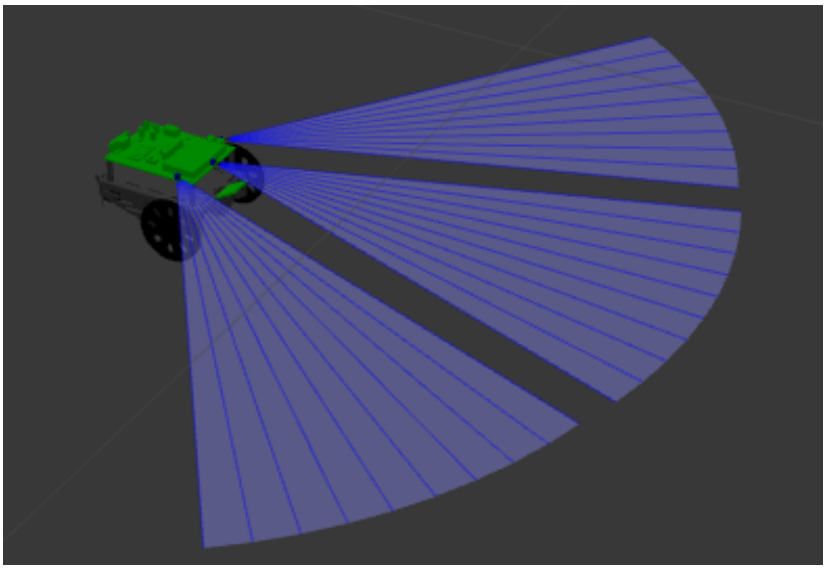


Figure 4.5 Camera and sensor in Gazebo Environment.

There are two kinds of sensors used here Ultrasonic sensor for Maze following environment. Another one is a laser sensor for obstacle detection functionality. Only the sensing unit changed in different environments.

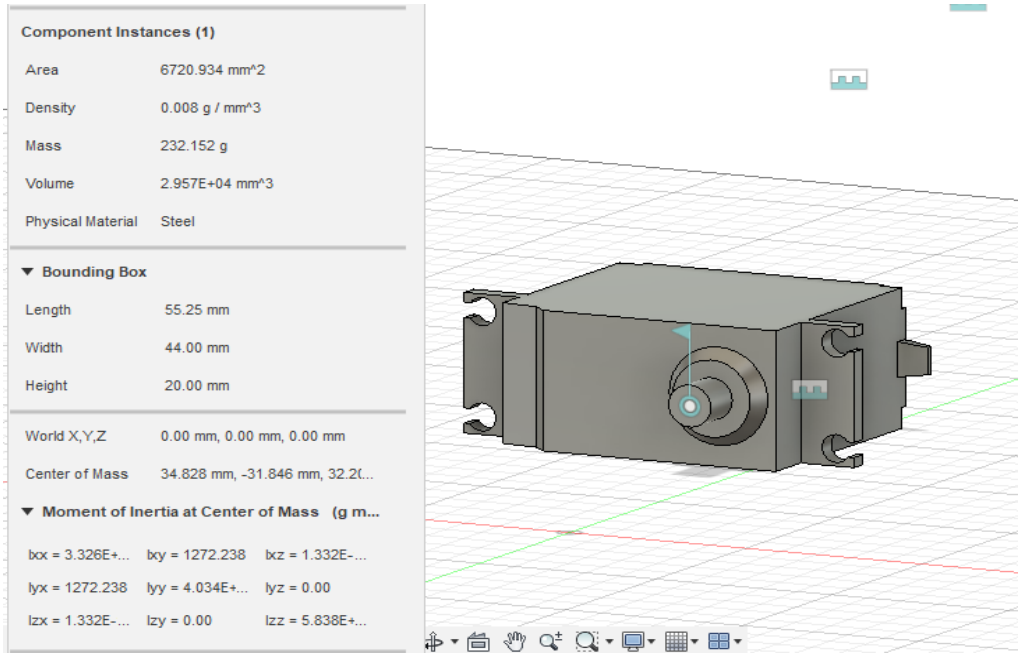


Figure 4.6 Model of servo motor

4.5 From Fusion to Gazebo

For the robot simulation to be accurate, the model of the system is also important. Furthermore, the description of the model is also required with controllers of model-based for the feedforward control command. For Boe-Bot robots, the information of the model, including the mass, inertia, center of mass, is used for both simulations of physics(Gazebo) and the design of controllers. In Gazebo the user-defined SDF and XML files are typically manually crafted, which makes way for an increase in errors.

Designing a Boe-Bot with the help of 3D computer-aided design(CAD) software is advantageous over the manual craft, and the description of the model can be easily obtained from the designed CAD model.

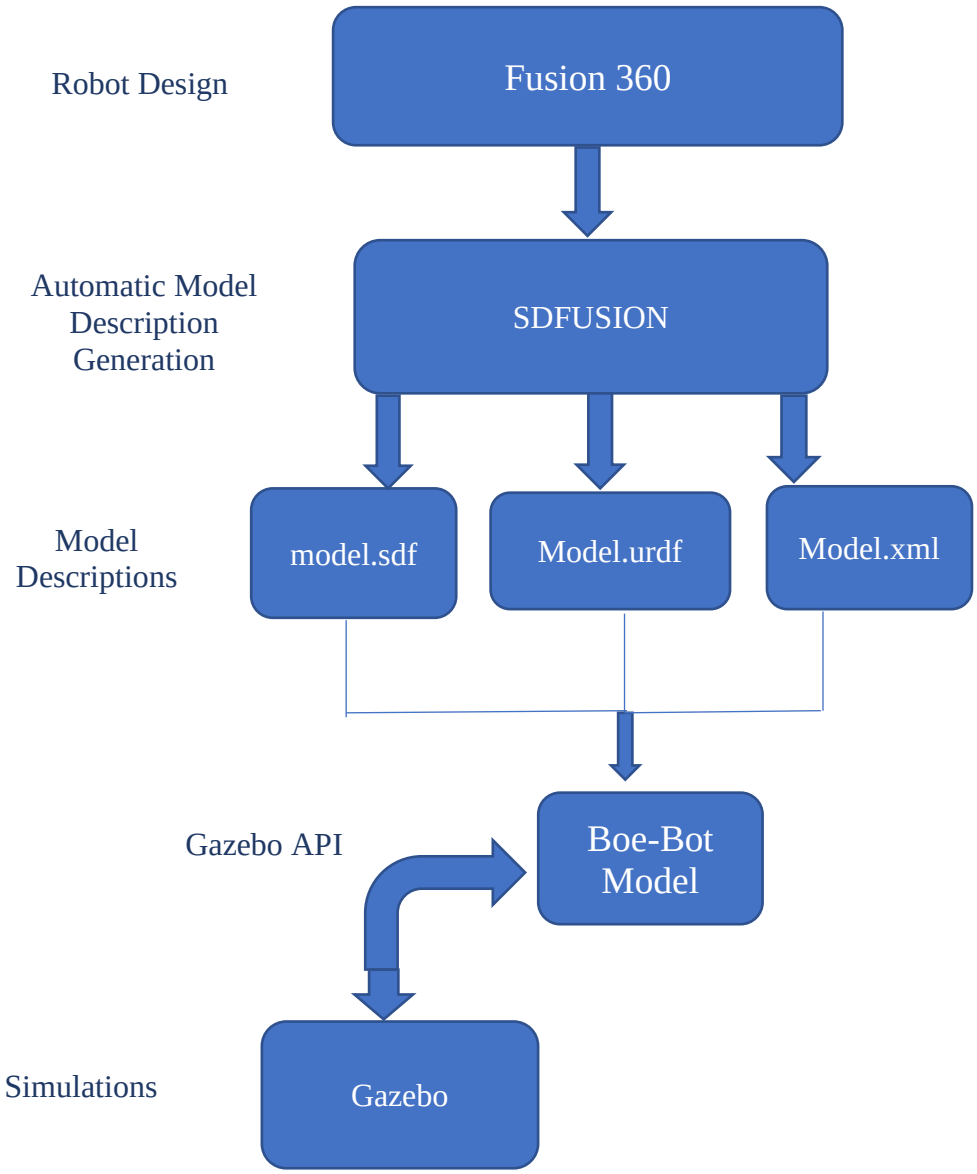


Figure 4.7 Design Flow

Figure 4.7 shows the flow, components, and interaction while the model has been exported to Gazebo as a URDF file. In this, the framework can be considered as end-to-end, wherefrom the fusion design of the robot to a well-accepted simulation environment.

After this, some ROS topics are used for the controlling between the Gazebo and the model. The capability of fusion 360 can be extended using a python module (SDFusion) in devel using the fusions python API. The user can control the robot through the GUI, the defined model can be automatically exported to URDF as an XML model description that is required by Gazebo. Using SDF files and Gazebo C++ API the Boe-Bot module is

developed. The points which are defined in SDFusion will be saved in the SDF file with other physical properties.

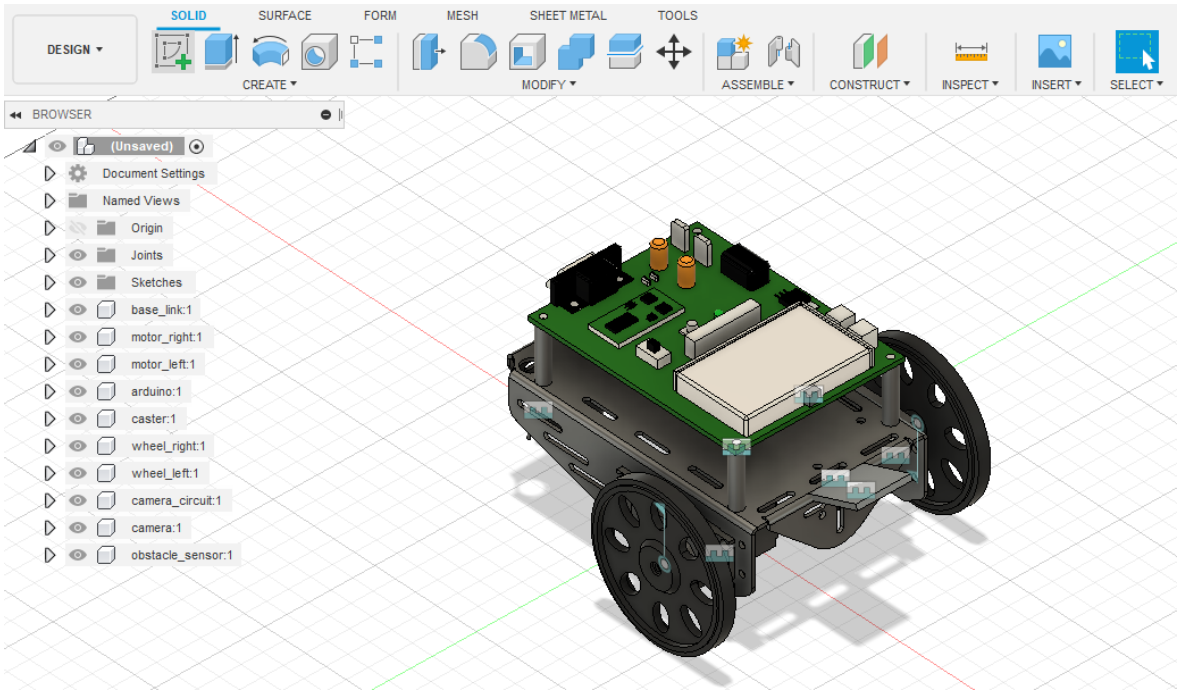


Figure 4.8 The Exported Boe-Bot Model

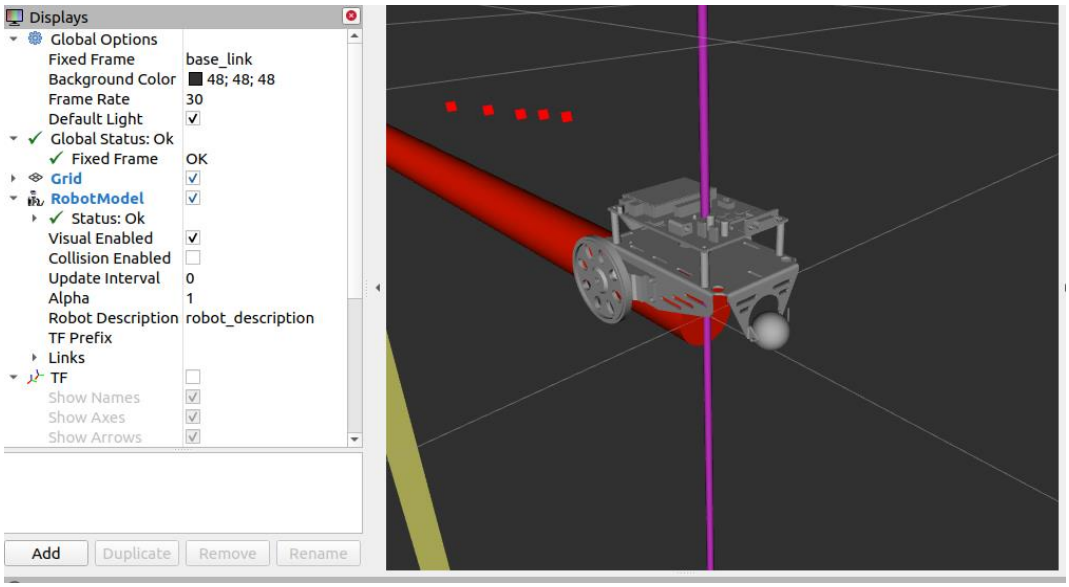


Figure 4.9 Visualizing Boe-Bot through RVIZ

The RVIZ visualization tool helps to show the results of the simulated Boe-Bot model. The other pieces of information about the model like coordinate frames and sensor detection point, each joint angle can also be visualized in RVIZ to monitor the robot's state.

5 SIMULATION IN DIFFERENT ENVIRONMENTS

The simulation of the Boe-Bot has been established in two different worlds or environments in the simulator Gazebo. In each environment different functionality of the robot has been shown. Using different sensors and camera the simulations has executed. For each functionality, we use different python programs. According to these python scripts the robot has been controlled.

In this chapter, we discuss different codes and the way the programs have been executed inside the simulator. The logic behind the python codes has been explained through the flow chart.

5.1 Flowcharts

The flowchart gives a detailed description of the logic behind the script step-by-step process. The operation has two sections. They are Line following with obstacle detection and wall following by Boe-Bot in Gazebo.

5.1.1 Maze Follower

The Wall follower is one of the most simple maze-solving algorithms. The algorithms were created to be used from inside the maze by solvers who had no prior knowledge of the maze. The maze solver is guaranteed to find the exit if the walls are simply connected, that is if the walls inside the maze are all connected or to the outer boundary, and there are no loops. When one hand is in contact with the wall all of the time, the solver can take either constant right or constant left turns, depending on which hand is in contact with the wall. The flowchart representation of the algorithms can be seen in the Figure below. However, this method will only operate if the maze's target is connected to its outer boundary.

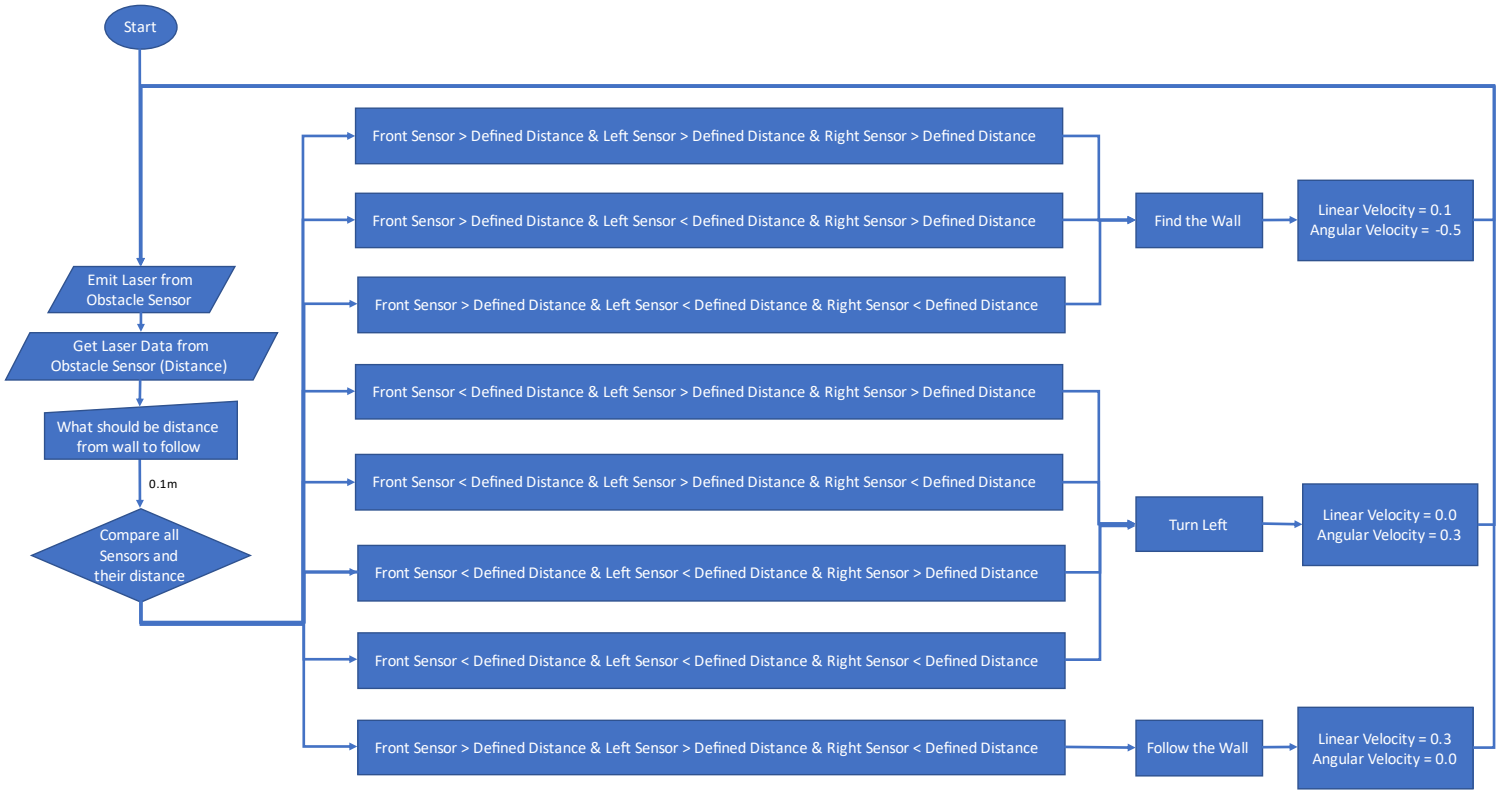


Figure 5.1 Flow chart of Maze follower Boe-Bot

5.1.2 Line Follower

Figure 5.2 depicts a line follower algorithm as a block diagram.

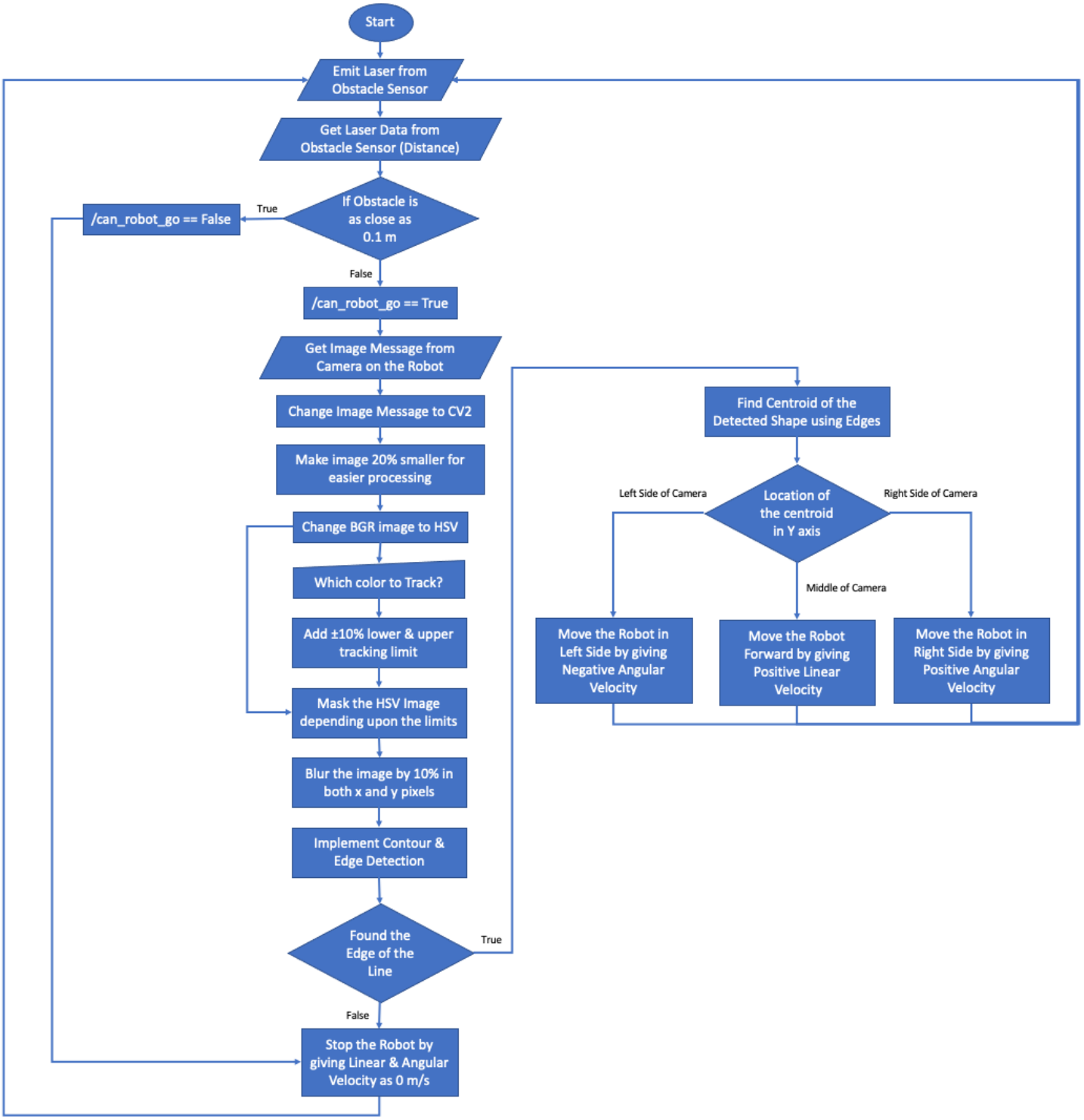


Figure 5.2 Flow chart of Wall follower Boe-Bot

5.2 Line Following Boe-Bot

The Line following simulation is a very interesting topic, different tools and libraries are used for this simulation. In this task the robot should follow the black line, it also needs to detect the obstacles. In my design instead of using reflectance sensors to follow the line, an Arduino camera is used to capture the real-time video and algorithms to follow the line. The laser sensor is used for obstacle detection. The simulation was programmed in python scripts using the OpenCV libraries.

There are many advantages to using a camera. The first one is we could reduce the number of sensors used, it not necessary to have a lot of sensors mounted on the robot or very close to the floor that will interfere with obstacles, and further, it may result in some irregularities. The second benefit is that the vision to see what is in front of the robot without any other complex swinging sensor arm. So we have information on not only the actual position of the robot above the line but also the position of the line which the robot needs to follow, this makes the line following more controllable.

5.2.1 Image Processing using OpenCV

OpenCV is a free and open-source computer vision and machine learning library. It is primarily concerned with image recognition, video recording, and attribute identification such as object and face detection.

- CV_bridge

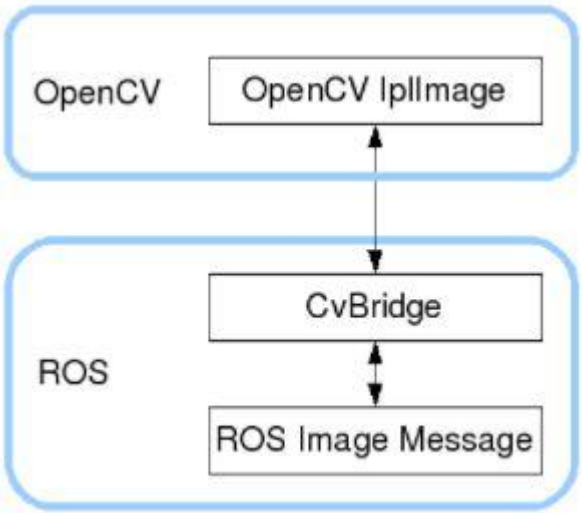


Figure 5.3 Bridging OpenCV with ROS using CvBridge

The OpenCV bridge package links the OpenCV image system to ROS. RGB Systems and OpenCV are incompatible. Most OpenCV functions expect mono8 and bgr8 image encodings, while the ROS Images are in RGB encoding. A feature of the OpenCV bridge is the ability to convert between them. The job is to program the robot to follow a black line. We must first obtain image data from the Boe-Bot and convert it to a format that OpenCV can understand. Then we set up the CVBridge class and subscribe to the subject, which has a callback function that extracts the data and encodes it in the format we want.

- Image Cropping

Cropping the images makes the detecting system faster. Initially, it is necessary to crop out the parts of the picture that aren't useful. This also aids us in working with the image's small scale. This also allows us to work with the image's small scale. And we're working with the image that's needed for this mission, which speeds up the detecting procedure. We can also pick the image range in which the line is clear, which isn't too near or too far and is in the center, which is useful for potential processes.

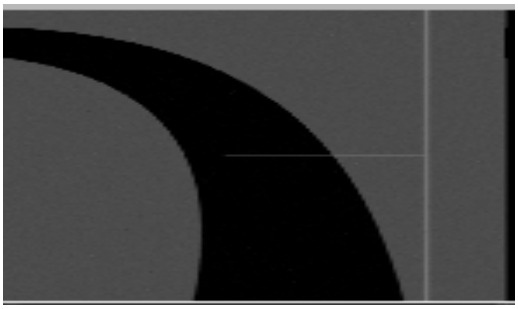


Figure 5.4 Cropped image of Line.

- Converting BGR to HSV image

Since both RGB and BGR images are highly susceptible to color saturation, we convert the BGR image to HSV, which is less susceptible to color saturation under lightning conditions. HSV's basic concept is to remove color saturation. OpenCV has a fantastic function that allows you to convert a BGR image to an HSV image. On the cropped image, we use this function. This is accomplished by choosing the HSV color set, which in our case is black, and making the black color blend in with it.

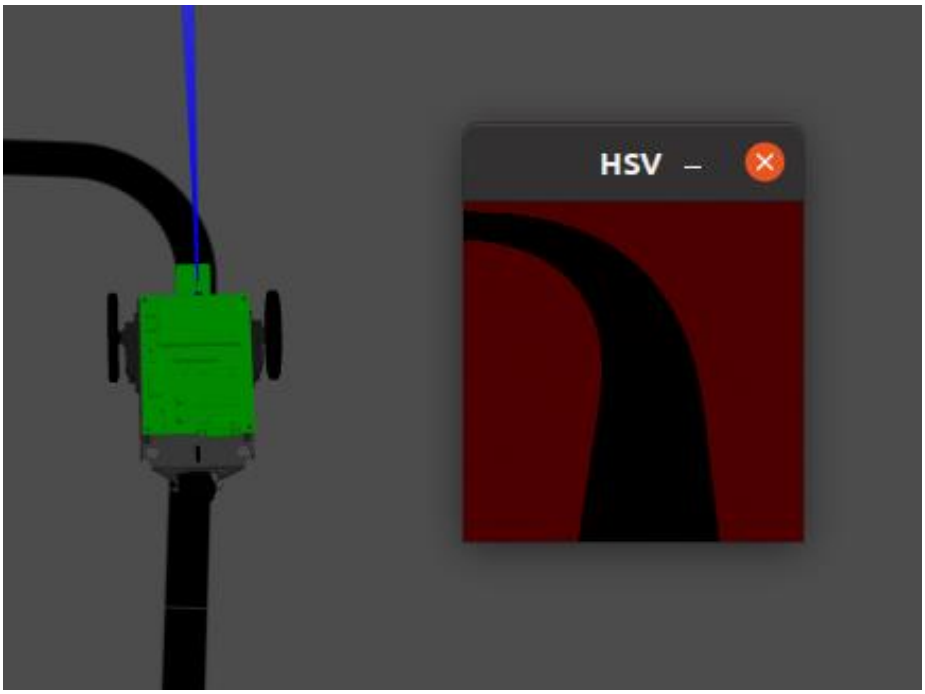


Figure 5.5 HSV image

To choose the acceptable code, it is necessary to take into consideration that, when calling the read function, The stored image should be in BGR format. So for converting the BGR to HSV we should use the COLOR_BGR2HSV code. The converted image will be returned by the cvtColor, which we will save in a variable.

```
hsvImage = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

Following that, we'll show both photos. To display each image, all we have to do is call the imshow function. It takes two inputs: a string with the name to be allocated to the window that will display the image, and the image itself.

```
cv2.imshow('Original image',image)
cv2.imshow('HSV image', hsvImage)
```

To test the code simply run the python script in the simulation.

- Edge Detection

The color-filtered picture is subjected to a Canny edge detector. This approach is used to highlight regions with high spatial derivatives and then monitor them to suppress any pixels that are not at the limit. Hysteresis is used to further minimize the gradient array. To map along the remaining pixels that haven't been suppressed, hysteresis is used. Hysteresis employs two thresholds, with the first being set to zero if the magnitude falls

below it (made a non-edge). It is designated as an edge if the magnitude exceeds the high threshold. If the magnitude is between the two limits, it is set to zero unless a gradient path exists from this pixel to another pixel. The application of the Canny edge detector is explained in the following steps:

- Binary Masking

Then, in a processing function call, use the Binary Mask to inform the function which pixels we need to consider and which pixels we need to remove. The pixels that are not black are discarded here. This function is applied to the HSV file, resulting in a black and white image.

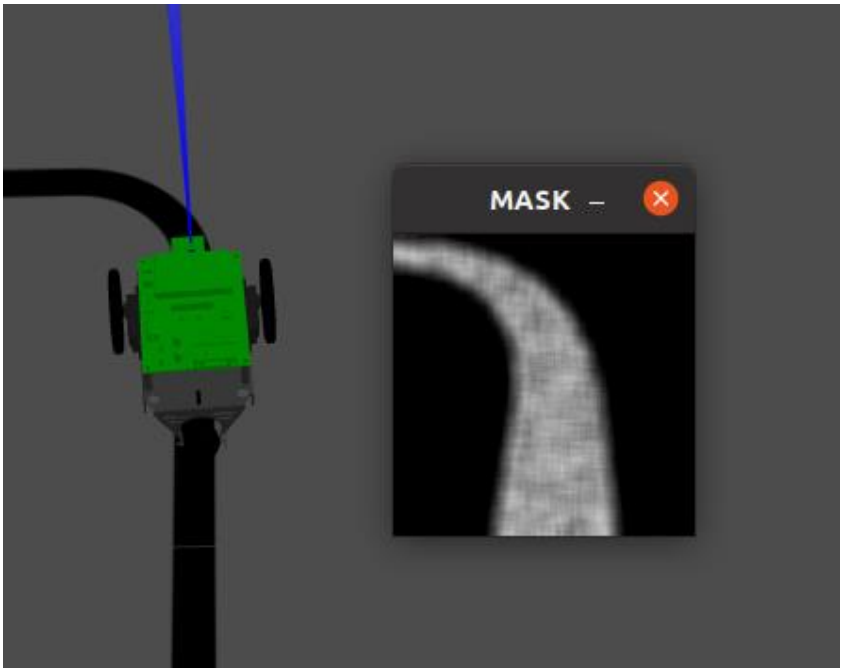


Figure 5.6 Binary Masking

And we just need to remember the white color, which is the black line, and the black in the rest of the picture that we ignore. That is, where the mask pixel is one, the pixel will remain unchanged, and where the mask pixel is zero, the pixel will be set to zero. Then we combine the HSV and binary images to color only the black part of the picture, leaving the rest black.

To measure the fixed point that the robot follows. However, since we have the images, we choose the centroid of the black color to represent the center of mass. To do so, we must locate the contours and measure the center of mass, which we will color red.

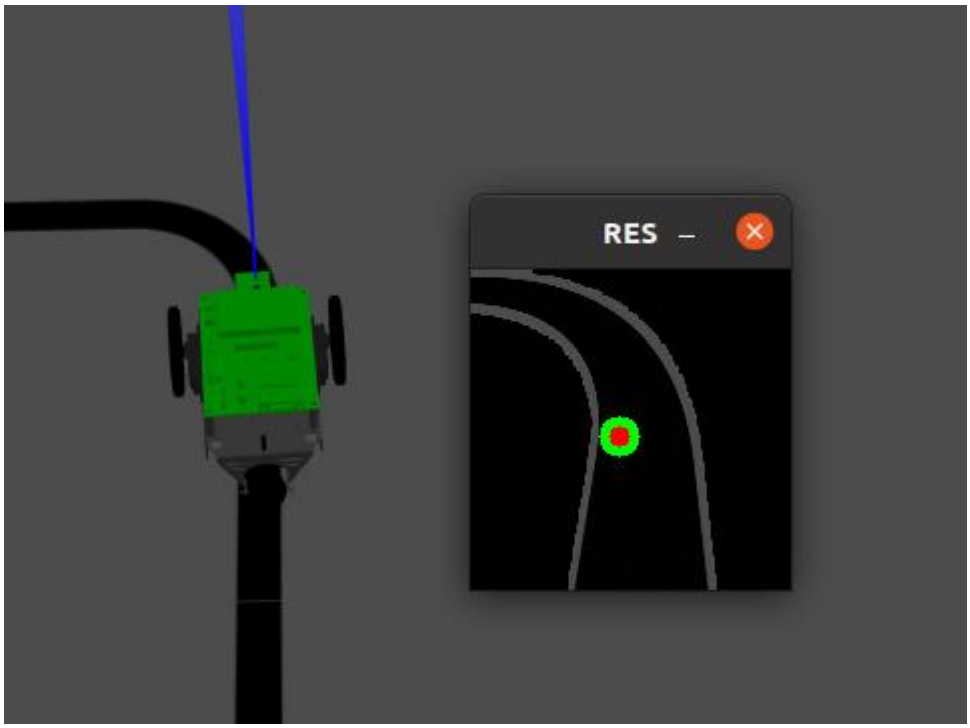


Figure 5.7 Centroid calculation and motion of Boe-Bot

5.3 Obstacle Detection

The Boe-Bot model comprises a laser sensor module including a compact camera module because of its obstacle dodging and laser mapping capabilities, this laser obstacle sensor can detect obstacles in front of it, measure their shape and distance, making it ideal for integration in robotics or autonomous moving vehicles. In this simulation Gazebo plugin for the hokuyo laser. The laser plugin was also declared with the Gazebo tags. But the plugins are referred to as the link obstacle_sensor1.

The laser sensor detects checks whether there are any obstacles in front of the robot while following the line. The logic behind the line following robot is that the laser sensor reads the obstacle and sends a command to another line following the script. The command asks whether the robot can move or not, if it is yes the line follower algorithm initializes then continues the loop.

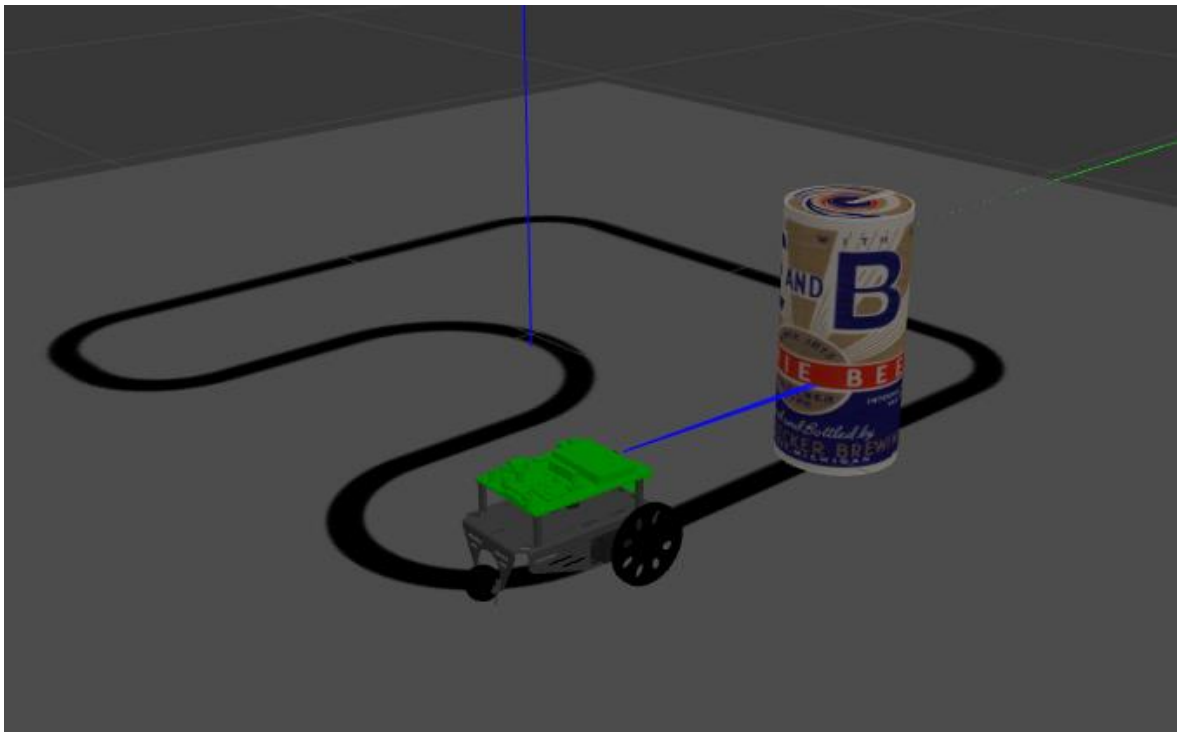


Figure 5.8 Boe-Bot detecting the beer can

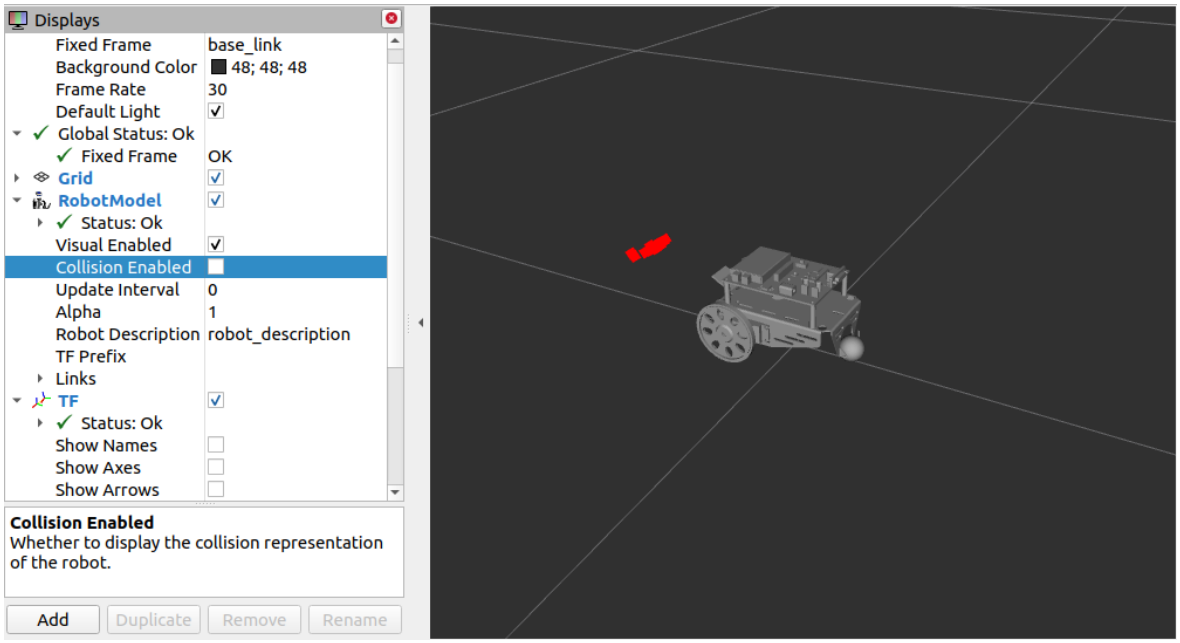


Figure 5.9 RVIZ visualization of laser sensor/Red dots

5.4 Maze Solving Boe-Bot

This wall following task aims to test the Boe-Bot functionality to navigate through a maze. Specifically, the navigation prepared through a maze containing a closed-loop. The Boe-Bot navigate and enter into the maze environment after reaching the goal the robot again moves to the initial position and continues its path. The goa of Boe-Botl is to follow the path of maze and then maneuver around it, either starting from the initial point or ending point.

5.4.1 Modeling Maze in Gazebo

The maze is created using a map. The map which is used as the reference for creating a maze is scaled with height and thickness. Maze height is 0.5m and thickness is 0.5m Figure given below.

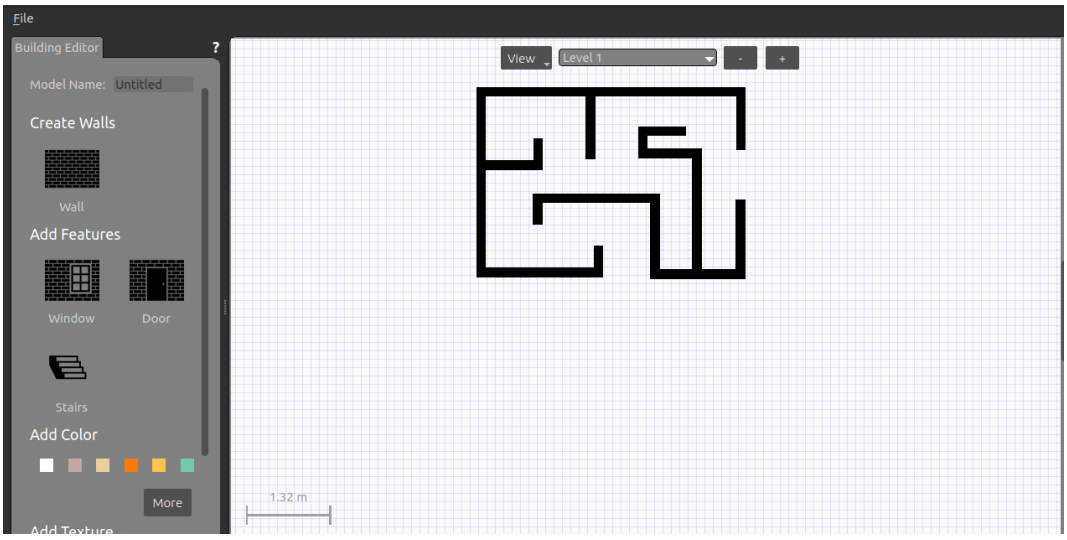


Figure 5.10 Map of maze.

In Gazebo, the keyword model is used to refer to a virtual entity. Any model from Gazebos's current library can be included. Several models can be used in simulations. It is possible to play around with these templates to see if they will suit the project. Cars, motorcycles, trees, buildings, walls, people(both strategic and moving), drones, and types of furniture are also some examples of the models.

The model will be saved in (/home/catkin_ws/src/boebot_description/models/my maze), there will be two file types as .config and .sdf these are file formats for configuration and simulation overview. These two files define any object in Gazebo. In case we need to change an object, these files should be edited manually or by using a GUI and these changes to be saved.

5.4.2 Wall following Algorithm

The Wall following algorithm is the most popular algorithm for maze-solving robots. The robot will follow the left or right wall to determine its path. Left-hand-right-hand rules are another name for this algorithm. When the robot comes to a junction in the road, it will look for open walls and choose a path, giving the selected wall priority. This technique is capable of getting the robot to the maze's finish point by using the walls as a guide.

The table below lists the instructions used in the algorithm for both the left and right walls:

Table 5.1 Left-Right wall following routine

Right wall following Algorithm	**Left wall following Algorithm**
If there is no wall at right,	If there is no wall at right,
Turn right	Keep straight
else	else
If there is no wall at straight	If there is no wall at straight
Keep straight	Keep straight
If there is no wall on left	If there is no wall on left
Keep straight	Turn Left
else	else
Turn Left	Turn Right

In this task i have used the right wall following algorithm for solving the maze. Three ultrasonic sensors are used in the front, right, and left positions. The range of the sensor is 0.25 cms if any sensor detects the wall within that range, the program executes different cases. Suppose the front sensor<d and the other two sensors not sensed the wall means >d, in that case, the Boe-Bot changes its direction of movement to left, it will turn to the left at an angular velocity of 0.3. Thus the movement of Boe-Bot has decided according to the sensor that detects the wall.

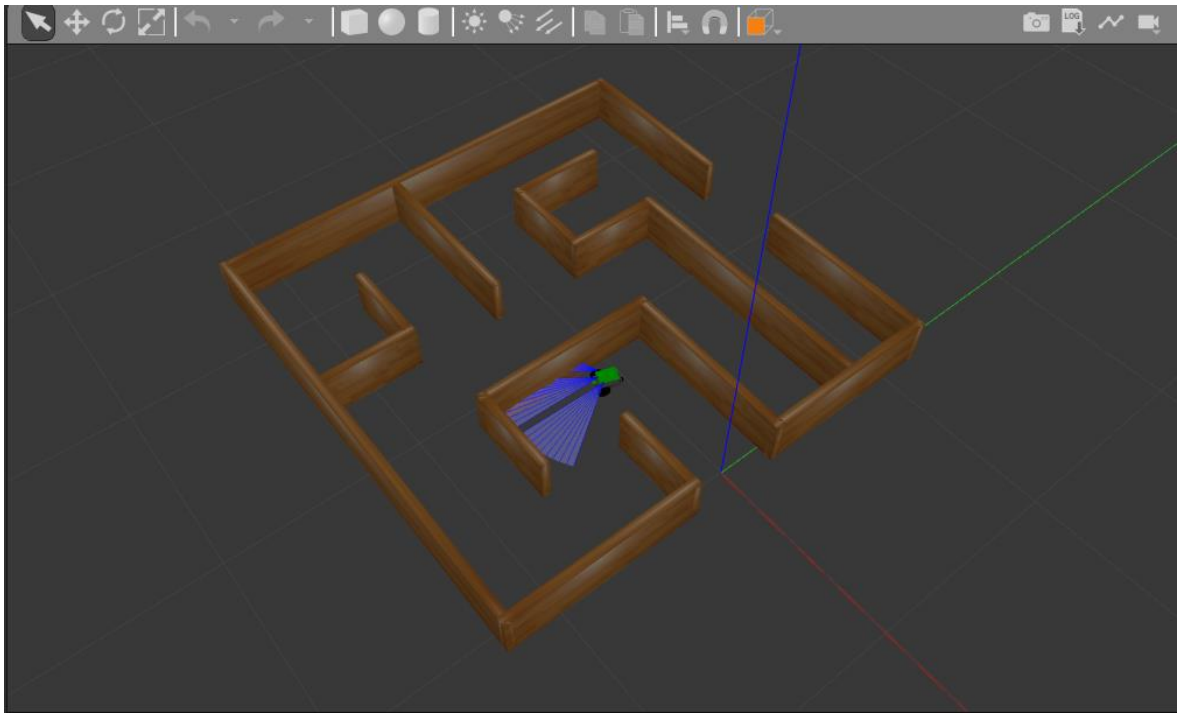


Figure 5.11 Case where frontsensor<d, Right sensor<d, left sensor>d

In figure 5.11, only the left sensor cannot detect the walls or the walls are out of the range of the left sensor.

```
def turn_left():
    msg = Twist()
msg.angular.z = 0.3
    return msg
```

Then it will publish the twist message that left sensor >d. The Boe-Bot turns left with an angular velocity of 0.3. The closest distance to the wall is 0.25cms.

```
harris@harris-ThinkCentre-M92p:~$ rosrun boebot_description wallfollower.py
Wall follower - [2] - follow the wall
Wall follower - [0] - find the wall
Wall follower - [2] - follow the wall
Wall follower - [0] - find the wall
Wall follower - [2] - follow the wall
Wall follower - [1] - turn left
Wall follower - [2] - follow the wall
Wall follower - [1] - turn left
Wall follower - [2] - follow the wall
Wall follower - [1] - turn left
Wall follower - [2] - follow the wall
Wall follower - [1] - turn left
Wall follower - [2] - follow the wall
Wall follower - [1] - turn left
```

Figure 5.12 Execution of the wall following cases using Terminal.

5.5 The interface between MATLAB and ROS

The second objective of the thesis is to build an interface between ROS and MATLAB for that different studies have already done and mentioned in chapter 2. Incorporating MATLAB environment gives the advantage to visualize the messages and sensor data from the ROS. In ROS different types of algorithms are used like control algorithms, perception algorithms, planning, and decision-making algorithms for an autonomous system. If the whole processing is going on offline it is possible to do recordings using rosbags they are considered as log files. It is also possible to communicate between MATLAB and ROS with the help of desktop prototyping. In that, it is possible to extract files, send commands.

Another way for communication is it can automatically generate C++ based executable ROS nodes from the design which was developed for testing. These generated nodes can be integrated with ROS to build a system so they can run on the target system with no dependencies on MATLAB and Simulink. ROS toolbox helps for the communication between the MATLAB/Simulink and ROS.

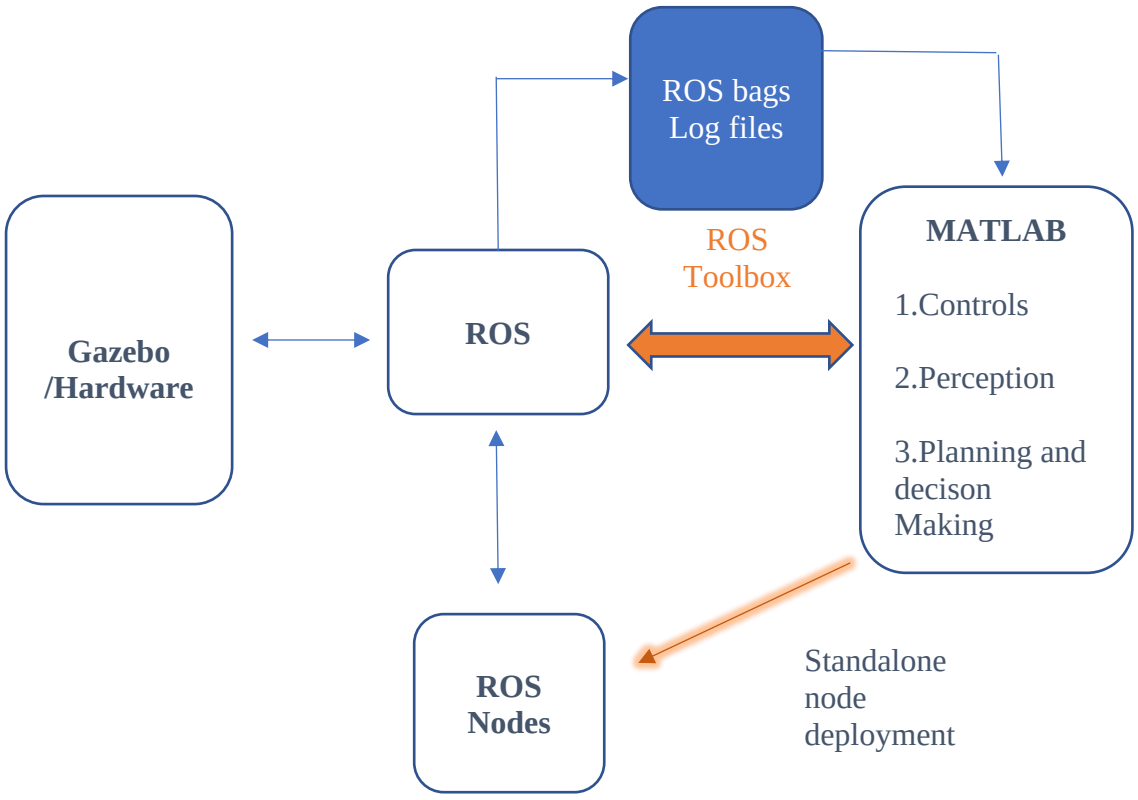


Figure 5.13 Process flow in MATLAB/ROS

It is possible to connect ROS from MATLAB/Simulink for the design and development of robotic autonomous systems with a live connection between MATLAB and Simulink to an existing ROS network. From that, it is very helpful to explore the available topics, services, and actions. This allows to send commands to and receive the data from any simulator or hardware or software node on the ROS network. MATLAB and Simulink provide a utility to work with common sensor data types such as laser scans, images, and 3d point clouds. In case there was any chance of using decoded data from the ROS bag file the ROS toolbox allows to read these files for forced processing or playback data as a simulation input.

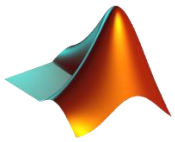
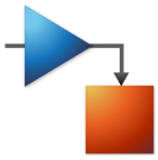
	ROS
MATLAB 	• Topic- Publish/Subscribe • Service – Server/ Client • Action – Client • Parameter – Get/Set • Custom messages (built-in) • ROSbag read
Simulink 	• Topic- Publish/Subscribe • Service – Call • Parameter – Get/Set • ROS Time • ROSbag playback • Code generation
ROS Distro	• ROS Noetic

Figure 5.14 ROS functionality in MATLAB/Simulink

From the figure, it is very evident that there is support for both MATLAB and Simulink workflow although they are not quite the same. A key difference is the code generation, it will only work from Simulink Model up to today. ROS is an enabling technology to develop complex intelligence systems with a given system design requirement ROS

provide a common framework to design simulate and prototype robotic application. With the ROS, there are different possibilities to analyze log data, design algorithm with various hand return codes, and visualize to debug the code result. By connecting MATLAB and Simulink with ROS allows performing system-level simulation with design and requirement traceability. It provides a rapid and innovative design process with many of the tools that Simulink has for the autonomous system. Another advantage is that a certifiable code can also generate for production.

5.5.1 Bridging ROS with MATLAB/Simulink

The traditional ROS enables system shown in fig. There is a host of ROS-enabled simulator and hardware and through the ROS network, they can communicate with a node that would enable autonomy visualization and so on. In the fig, MATLAB Simulink offers tools for autonomous system development. For autonomous systems development system should have a functional component of perception, planning, and control.

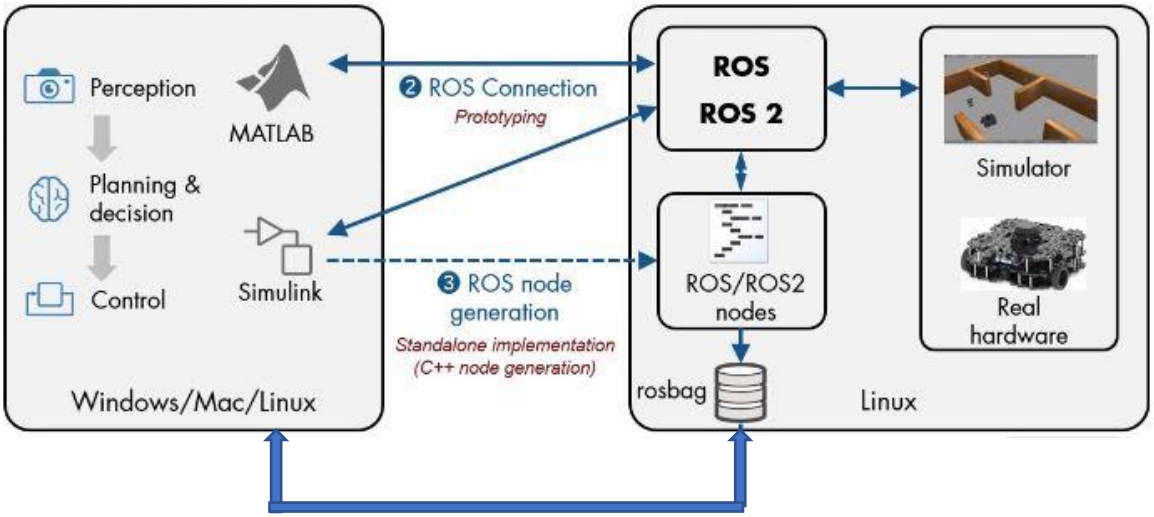


Figure 5.15 Bridging and workflows with MATLAB and ROS

There are mainly three workflows.

1. Data Analysis and Playback

   The data from the simulation and hardware are then established in MATLAB and Simulink environments for offline design tasks

2. Desktop Simulation

   There will be a live connection between ROS and an instance of MATLAB for algorithm development.

3. Deployment

   The same implementation used for the prototyping can be automatically generated as a stand-alone ROS node that will be no longer require MATLAB to learn.

The Boe-Bot runs in the Gazebo platform and it will be bridged with MATLAB using ROS toolbox. The study explains how to publish messages to the Boe-Bot and how to subscribe to topics that the robot publishes. For the development of the study, the maze follower world has been initialized in the Gazebo environment using python scripts.

The communication can be initialized using the IP address.

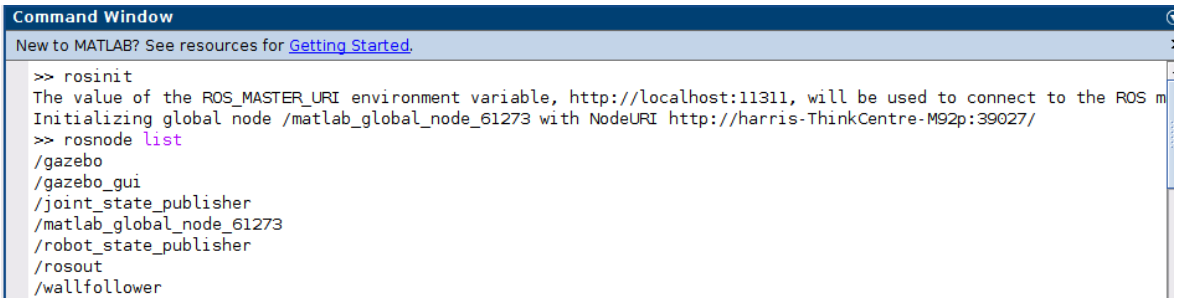


Figure 5.16 Initializing ROS in MATLAB

Some environment variables are set for the computer. ROS IP is different for each machine. The IP address of the pc can be found by using the command ifconfig in the Linux terminal and the IP address I got is inet: 10.10.254.23.The connection is established and it is possible to use the ROS in MATLAB.

The ROS topic list gives the entire list of topics and this is fully dependent on the ROS master machine which is connected. In this part, i have launched the Boe-Bot wall follower simulation which gives the things like odometry and Control topics as well as the ultrasonic sensor topics. The detailed information of each ROS topic can be found out using the command ros topic in MATLAB.

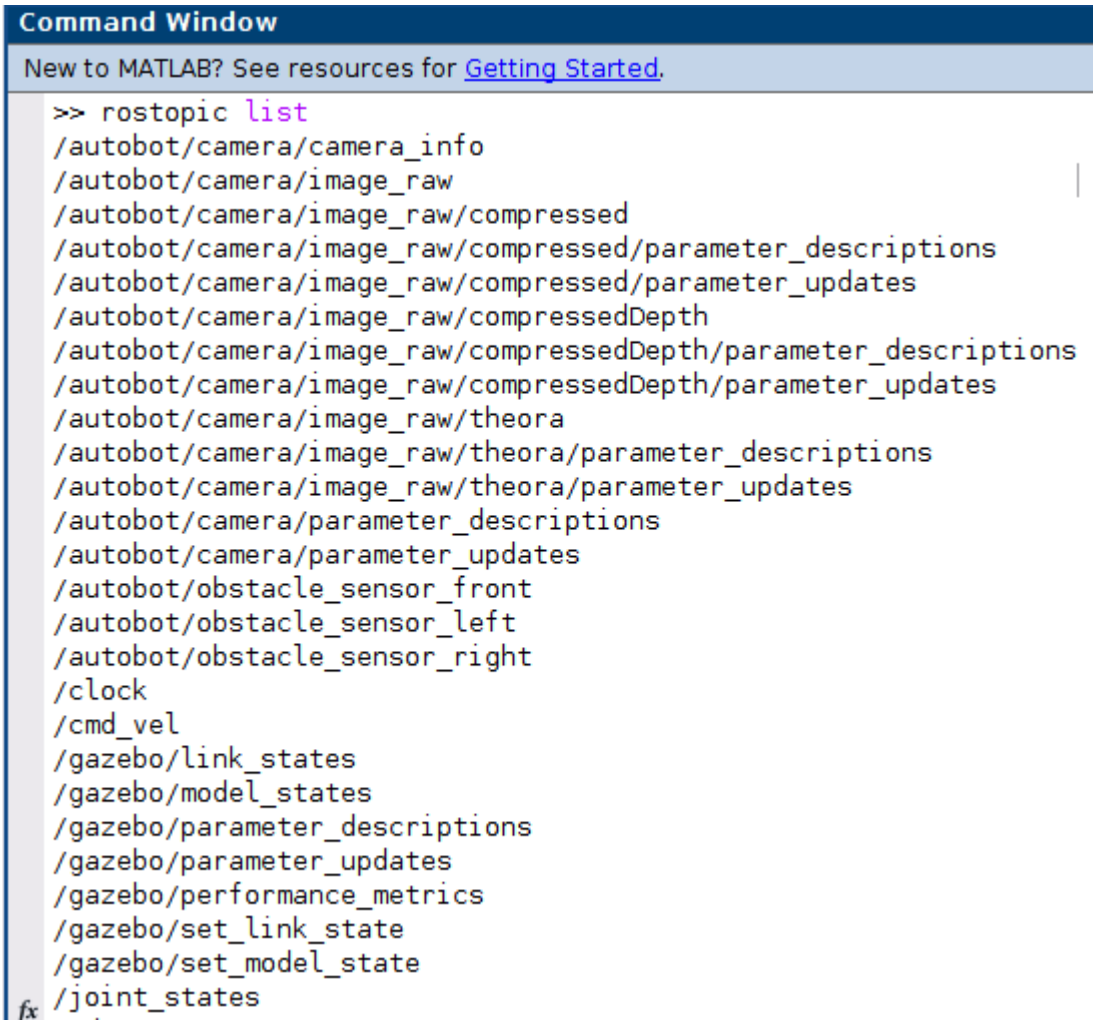


Figure 5.17 ROS topics

In the MATLAB environment, there are a variety of interactive commands that mimic the ROS command-line utilities for finding messages, nodes, topics, parameters, or services. These commands provide all of the necessary programmatic tools for writing MATLAB code that can completely participate in a ROS-based robot control system. This code can run on a Mac, Windows, or Linux system. The real-time update rate will be determined by the size of the messages, the complexity, and reliability of your MATLAB code, and the speed of your machine, however, tens of hertz is possible.

- Twist/geometry msgs

The topic expects messages of the form geometry msgs/Twist, which is the same as the ve_lmsg created earlier. Use the command 'ROStopic data topic name' to see which nodes are publishing and subscribing to a given topic. The velocity topic's publishers and subscribers are mentioned in the command below. One of the publishers mentioned is

MATLAB. In the case of the Boe-Bot wall follower environment, the publisher is the wall follower and the subscriber is the Gazebo. The message can be seen in Fig 5.18.

```
>> rostopic info /cmd_vel
Type: geometry_msgs/Twist

Publishers:
* /wallfollower (http://harris-ThinkCentre-M92p:33661/)

Subscribers:
* /gazebo (http://harris-ThinkCentre-M92p:33233/)
>>
```

Figure 5.18 Messages from cmd_vel

The Boe-Bot publishes its current position and orientation using the /Odom topic (collectively denoted as pose). Since the Boe-Bot lacks a GPS device, the pose will be based on the position of the robot when it was first turned on. Create an odometry message subscriber with the ability to return data, then extract the data and assign it to variables x, y, and z. The values and commands are shown in fig 5.19.

```
>> odomSub = rossubscriber("/odom");
>> odomMsg = receive(odomSub,3);
>> pose = odomMsg.Pose.Pose;
>> x = pose.Position.X;
>> y = pose.Position.Y;
>> z = pose.Position.Z;
>> [x y z]

ans =

   -0.6428    0.0527    0.0001

>>
```

Figure 5.19 Odometry messages

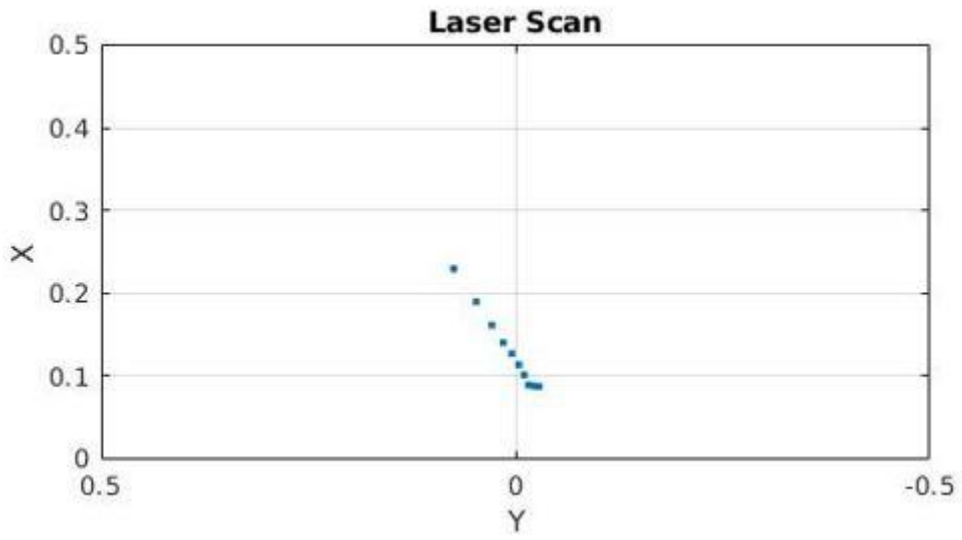


Figure 5.20 Laser scan graph

In ROS, the LaserScan object implements the sensor msgs/LaserScan message form. The item includes meta-data about the message as well as laser scan data. The range property and the readscan angles function can also be used to extract ranges and angles. Use read Cartesian to view, points in Cartesian coordinates. After subscribing to the laser subject, wait for the data to arrive before plotting it.

6 SUMMARY

The Boe-Bot simulation in the Gazebo was a Master's thesis topic that was aimed at developing a test environment for the students. The students can test their python codes on the Boe-Bot model. They can clone the developed environment from the Github repository and test their codes in the simulation. Multiple environments were developed for the simulation. The Boe-Bot was designed well with all the physical properties and fitted with appropriate sensors, cameras, and encoders. The Gazebo software used for creating the simulation environment has purposefully maintained a simple but effective interface to the physics engine and rendering capabilities. Two environments - line follower and maze follower scripts have simulated with the Boe-Bot.

The use of OpenCV for real-time computer vision has supported achieving stability in robot motion while following the line. Different libraries and scripts were taken into account to complete the simulation. This simulation system can be easily adapted to other robot models by modifying messages and services. The ROS incorporated with Gazebo made the work efficient and more attractive. ROS is a versatile and efficient tool for robotics research as well as applications like mobile robots. With help of ROS visualization tools like RVIZ and RQT, different properties like odometry, sensor data, and joints of Boe-Bot have visualized while simulating.

After the successful completion of the simulation in Gazebo with ROS, another task was to build an interface between MATLAB and ROS using the ROS toolbox. The main aim of this task is to analyze the capabilities of using ROS and MATLAB in the implementation of the Boe-Bot. The toolbox is interesting because of the unique environment it provides for generating and visualizing data. The ROS toolbox provides a very powerful setup with the publisher and subscriber which is used largely in ROS.

The research was a combination of Modeling and simulating. The incorporation of different operating systems, simulators, and libraries, bringing them under one roof was the best side of this thesis. The ROS toolbox is a viable choice for the implementation of autonomous programs for mobile robots. it is also accessible to the majority of students and engineers who aren't familiar with Linux or ROS.

KOKKUVÕTE

See magistritöö kirjeldab Boe-Bot robotile simulaatori keskkonna ettevalmistamist Gazebo simulatsioonikeskkonnas. Eesmärk oli luua simulatsioonikeskkond, kus tudengid saaksid Pythonis kirjutatud programme robotisimulaatoril testida. Keskkond on kättesaadav Git repositooriumis ning lihtsalt ülesseatav. Boe-Bot roboti programmi simuleerimiseks loodi kaks keskkonda, joonejälgija rada ning labürindi läbimine. Boe-Bot roboti mudeli loomisel lähtuti tugevalt roboti füüsiliselt platvormist. Mudelile lisati ka sensorid, kaamerad ja enkoodrid simulaatorist info saamiseks. Simulatsioonikeskkond Gazebos on loodud olema lihtsalt hoomatav koos oma füüsikamootori ja visualiseerimise võimekusega. OpenCV kasutamine võimaldab roboti liikumist joonejälgimise ülesandes tänu reaal-aja arvutinägemise võimekusele.

Simulatsiooni täitmiseks võeti kasutusel erinevad teegid ja skriptid. Simulaator on lihtsalt rakendatav ka teistele roboti mudelitele, vaja on vaid muuta sõnumeid ja teenuseid. Robot Operating System (ROS) koos Gazebo simulatsioonikeskkonnaga teeb simulatsiooni efektiivsemaks ja huvitavamaks. ROS on mitmekülgne ja efektiivne tööriist robootika valdkonnas nii teadustöös kui reaalsetes rakendustes. Lisaks sisaldab ROS visualiseerimise tööriistu nagu RVIZ ja RQT ning erinevaid andmehõive võimalusi. Näiteks saab monitoorida simulatsiooni käigus odomeetri, sensorite ja liitekohtade väärtusi. Pärast simulatsiooni keskkonna loomist ning Boe-Bot roboti simuleerimist Gazebos koos ROSi võimalustega oli teiseks ülesandeks luua liides MATLABi ja ROSi ning tema tööriistadega. Peamine eesmärk selle ülesande puhul oli analüüsida ROSi ja MATLABi kooskasutamise võimalusi Boe-Bot roboti simulatsiooni näitel.

MATLABi tööriistad on väga huvitavada, sest nad loovad uusi võimalusi ning keskkonna andmete loomiseks ja visualiseerimiseks. ROSi tööriistad võimaldavad luua võimsaid struktuure andmete saatja ja vastuvõtja vahel. Magistritöö peamiseks probleemiks sai modelleerimise ja simuleerimise sidumine, erinevate operatsioonisüsteemide vahel andmete, simulaatori, teekide ja keskkondade ühildamine ja ülesseadmine. Ühtlasi oli see töö autorile ka väga õpetlik. ROSi tööriistad on autori arvates üks parimaid võimalusi autonoomsete robotite programmide simuleerimiseks. Üheks peamiseks eelmiseks on ka asjaolu, et see keskkond ning tööriistad on kättesaadavad ja tasuta tudengitele ja inseneridele, kes ei ole Linuxi ja ROSiga varem kokku puutunud.

LIST OF REFERENCES

[1] D. Cook, A. Vardy ja R. Lewis, „A Survey of AUV and Robot Simulators for Multi-Vehicle Operations,“ *2014 IEEE/OES Autonomous Underwater Vehicles (AUV),* pp. 1-8, 2014.

[2] P. Vyavahare, S. Jayaprakash ja K. Bharatia, Construction of URDF model based on open source robot dog using Gazebo and ROS, Dubai, United Arab Emirates: IEEE, 2019, p. 5.

[3] L. Zhou, R. Li, K. P. Ng, A. Narayanamoorthy ja Z. Huang, „A Robotics Simulator Platform for RADOE,“ %1 *2nd International Conference on Control, Automation and Robotics (ICCAR),* Hong Kong, China, 2016.

[4] M. I. S. Julius Fusic ja Dr.K.Hariharan, „Path planning for car like mobile robot using robot operating system,“ %1 *National Power Engineering Conference (NPEC),* Madurai, India, 2018.

[5] K. Takaya, T. Asa, V. Kroumov ja F. Smarandache, „Simulation environment for mobile robots testing using ROS and Gazebo,“ %1 *2016 20th International Conference on System Theory, Control and Computing (ICSTCC),* Sinaia, Romania, 2016.

[6] Z. Riaz, A. Pervez, M. Ahmer ja J. Iqbal, „A Fully Autonomous Indoor Mobile Robot using SLAM,“ *2010 International Conference on Information and Emerging Technologies,* pp. 1-6, 2010.

[7] R. Muñoz-Salinas, M. J. Marín-Jimenez ja R. Medina-Carnicer, „Simultaneous localization and mapping with squared planar markers,“ *Science Direct,* kd. 86, 2019.

[8] C. Y. Liz ja G. L, „An 879GOPS 243mW 80fps VGA Fully Visual CNN-SLAM Processor for Wide-Range Autonomous Exploration,“ %1 *2019 IEEE International Solid- State Circuits Conference - (ISSCC),* San Francisco, CA, USA, 2019.

[9] X. Linhul, W. Jinge ja Q. Xiasong, „Semantic monocular visual localization and mapping based on deep learning in dynamic Environment,“ *Robotics and Autonomous Systems,* kd. 117, pp. 1-16, 2019.

[10] L. Ruihao, W. Sen ja L. Zhiqiang, „UnDeepVO: Monocular Visual Odometry Through Unsupervised Deep Learning,“ *2018 IEEE International Conference on Robotics and Automation (ICRA),* pp. 7286-7291, 2018.

[11] M. Galli, R. Barber, S. Garrido ja L. Moreno, „Path planning using Matlab-ROS integration applied to mobile robots,“ *2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC),* pp. 98-103, 2017.

[12] P. Corke, „Integrating ROS and Matlab,“ *IEEE Robotics & Automation Magazine,,* kd. 22, pp. 18-20, June 2015.

[13] S. Tokunaga, K. Miura ja T. Azumi, „MATLAB/Simulink Benchmark Suite for ROS-based Self-driving Software Platform,“ *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC),* pp. 83-84, 2019.

[14] R. Gonzalez, C. Mahulea ja M. Kloetzer, „A Matlab-based Interactive Simulator for Mobile Robotics,“ %1 *2015 IEEE International Conference on Automation Science and Engineering (CASE),* Gothenburg, Sweden.

[15] A. S. A. S. S. Singh, „D-H parameters augmented with dummy frames for serial manipulators containing spatial links,“ *The 23rd IEEE International Symposium on Robot and Human Interactive Communication,* pp. 975-980, 2014.

[16] F. C. H. G. Y. L. T. Guo, „An improved D-H convention for establishing a link coordinate system,“ %1 *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Dali,China, 2019.

[17] Kumar.Neerendra ja Vámossy.Zoltán, „Robot Obstacle Avoidance Using Bumper Event,“ *11th IEEE International Symposium on Applied Computational Intelligence and Informatics*, p. 490, 2016.

[18] „https://www.raspberrypi.org,“ [Võrgumaterjal]. Available: https://www.raspberrypi.org/products/camera-module-v2/. [Kasutatud 3 04 2021].

[19] „https://linuxhint.com,“ [Võrgumaterjal]. Available: https://linuxhint.com/best_raspberry_pi_cameras/. [Kasutatud 15 04 2021].

[20] „http://fec.4art-studio.com,“ FEC, [Võrgumaterjal]. Available: http://fec.4art-studio.com/product/Laser%20proximity%20sensor/2687. [Kasutatud 05 April 2021].

[21] „https://www.sensorpartners.com,“ [Võrgumaterjal]. Available: https://www.sensorpartners.com/en/knowledge-base/the-different-types-of-laser-sensors/. [Kasutatud 12 05 2021].

[22] „https://lastminuteengineers.com,“ [Võrgumaterjal]. Available: https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/. [Kasutatud 11 03 2021].

[23] „https://www.blender.org/,“ [Võrgumaterjal]. Available: https://www.blender.org/. [Kasutatud 12 15 2021].

[24] „https://subscription.packtpub.com,“ [Võrgumaterjal]. Available: https://subscription.packtpub.com/book/hardware_and_creative/9781788478953/1/ch01lvl1sec14/understanding-the-ros-computation-graph-level. [Kasutatud 15 02 2021].

[25] „https://trojrobert.medium.com,“ [Võrgumaterjal]. Available: https://trojrobert.medium.com/hands-on-introdution-to-robot-operating-system-ros-4914386e4a45. [Kasutatud 10 03 2021].

[26] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[27] P. Henry, M. Krainin, E. Herbst, X. Ren ja D. Fox, „RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments,“ *Science Direct*, p. 15, 2014.

APPENDIX

The python scripts, instruction to setup the ROS and Gazebo simulators and two different simulator environment files are uploaded in the given github respository.

https://github.com/harant/Boe-Bot-Simulation-in-Gazebo