Design, Construction, Energy Modeling, and Navigation of a Six-Wheeled Differential Drive Robot to Deliver Medical Supplies inside Hospitals

Makhluk Hossain Prio

DESIGN, CONSTRUCTION, ENERGY MODELING, AND NAVIGATION OF A SIX-WHEELED DIFFERENTIAL DRIVE ROBOT TO DELIVER MEDICAL SUPPLIES INSIDE HOSPITALS

by

MAKHLUK HOSSAIN PRIO

(Under the Direction of Fernando Rios-Gutierrez)

ABSTRACT

Differential drive mobile robots have been the most ubiquitous kind of robots for the last few decades. As each of the wheels of a differential drive mobile robot can be controlled, it provides additional flexibility to the end-users in creating new applications. These applications include personal assistance, security, warehouse and distribution applications, ocean and space exploration, etc. In a clinic or hospital, the delivery of medicines and patients' records are frequently needed activities. Medical personnel often find these activities repetitive and time-consuming. Our research was to design, construct, produce an energy model, and develop a navigation control method for a six-wheeled differential drive robot designed to deliver medical supplies inside the hospital. Such a robot is expected to lessen the workload of medical staff. Therefore, the design and implementation of a six-wheeled differential drive robot with a password-protected medicine carrier were presented. This password-protected medicine carrier ensures that only the authorized medical personnel can receive medical supplies. The low-cost robot base and the medicine carrier were built in real life. Besides the actual robot design and fabrication, a kinematic model for the robot was developed, and a navigation control algorithm to avoid obstacles was implemented using MATLAB/Simulink. The kinematic modeling is helpful for the robot to achieve better energy optimization. To develop the object avoidance algorithm, we investigated the use of the Robot Operating System (ROS) and the Simultaneous Localization and Mapping (SLAM) algorithm for the implementation of the mapping and navigation of a robotic platform named TurtleBot 2. Finally, using the Webot robot simulator, the navigation of the six-wheeled mobile robot was demonstrated in a hospital-like simulation environment.

INDEX WORDS: Differential drive, Kinematic model, Simultaneous localization and mapping, Potential field, Virtual reality, Webot simulator, TurtleBot, Wheeled robot

DESIGN, CONSTRUCTION, ENERGY MODELING, AND NAVIGATION OF A SIX-WHEELED DIFFERENTIAL DRIVE ROBOT TO DELIVER MEDICAL SUPPLIES INSIDE HOSPITALS

by

MAKHLUK HOSSAIN PRIO

B.S., Rajshahi University of Engineering and Technology, Bangladesh, 2014

M.S., Georgia Southern University, 2020

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

DESIGN, CONSTRUCTION, ENERGY MODELING, AND NAVIGATION OF A SIX-WHEELED DIFFERENTIAL DRIVE ROBOT TO DELIVER MEDICAL SUPPLIES INSIDE HOSPITALS

by

MAKHLUK HOSSAIN PRIO

DEDICATION

I would be glad to dedicate this thesis to my supportive parents, my sweet sister, and my lovely wife.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Rios, who was instrumental and supportive throughout my M.S. research. I am also grateful to my thesis committee members whom I always found cooperative and cordial to me.

TABLE OF CONTENTS

Page

LIST OF FIGURES

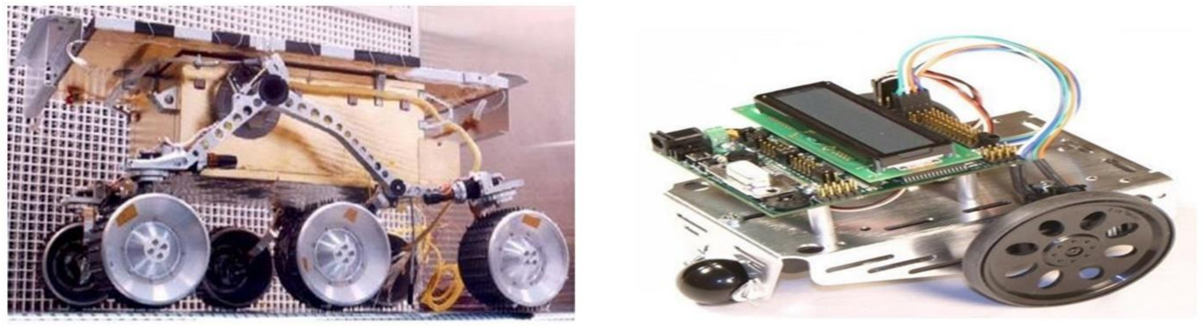
CHAPTER 1

INTRODUCTION

Mobile robotics is an ever-growing and well-known field of technological research interest. It has seen revolutionary advancements over the last few decades. As the name suggests, mobile robots are capable of moving from one point to another in real-time indoors or outdoors, on the ground, and on aerial, or aquatic environments, employing locomotive elements such as wheels, artificial legs, artificial wings, and propellers (Wahab, Rios-Gutierrez, & El Shahat, 2015). The evolution of mobile robots has been governed by human necessities (Garcia, Jimenez, De Santos, Armada, & Magazine, 2007). Nowadays, mobile robots are being deployed to autonomously perform complicated, arduous, and risky tasks like exploration in hazardous areas, delivering medical supplies inside hospitals, rescuing injured people in disasters and calamities, carrying heavyweight industrial goods, assisting humans in household works, etc. (Prio and Rios 2019). Among all kinds of mobile robots, the differential drive robot is the most popular and widespread, which is the focal point of this research.

1.1 Differential Drive Mobile Robot

The differential drive mobile robot architecture is one of the most commonly used robot architectures due to its simple construction and dynamics (Konduri, Torres, Pagilla, & Control, 2017). It has the configuration used by most electric wheelchairs (Balkcom & Mason, 2000). The number of wheels placed on either side of the robot body can be one, two, or three depending upon the intended application. The robot is moved by controlling the rotation and speed of wheels on either side of the robot body separately. It often needs an extra castor wheel in front of the body to maintain a better balance and support. Examples of differential drive robots can be the Sojourner and the IntelliBrain-Bot, as shown in Figure 1.1 (Mester, 2006).



(a) The Sojourner (b) The IntelliBrain-Bot

Figure 1.1: Examples of differential drive robots

1.2 Motivation of the Thesis

Nowadays, smart delivery differential drive autonomous robots have been promoted into the market. They are seen to deliver foods, packages, or industrial goods in indoor environments: restaurants, hospitals, or industries. An example of such a robot would be MO1 Smart Delivery Robot made by New Era AI Robotics Inc, as shown in Figure 1.2.

In hospitals and medical environments, it is often necessary to carry or move medical supplies from a central location to other places where they are needed. This operation is typically performed by nurses who take the medicines to the places where they will be used. However, this operation is repetitive, time-consuming, and inefficient. An autonomous differential drive robot can be used to perform the delivery, freeing the nurse staff to do other more important activities. In medical environments, there are doors, corridors, and in general small available spaces to maneuver. An autonomous delivery vehicle needs to have the intelligence to move in these spaces.

Figure 1.2: M01 smart delivery robot (Source: New Era AI Robotics)

Due to the above reasons, the primary purpose of this thesis is focused on the development and navigation of a six-wheeled differential drive, smart delivery intelligent robot for applications in a hospital or home environment where limited space is available.

One of the main constraints of any kind of robot is its limited energy storage capacity. It depends mostly on the energy provided by the principal power source, which typically are DC batteries that have a limited energy capacity. This inadequate amount of energy leads to a short time of functioning for the robot. Therefore, the energy optimization of the robot is needed to overcome this shortcoming to a certain extent. Hence, in this thesis, we investigated and developed a model for the kinematics of the proposed six-wheeled differential drive robot, its links to the dynamics, and some considerations regarding more accurate modeling for energy optimization.

1.3 Objectives of the Thesis

- To design a six-wheeled mobile differential drive robot using SolidWorks software to deliver medical supplies inside hospitals.
- To build the robot in real life.

- To develop a kinematic model of the robot.
- To learn and use the Robotic Operating System (ROS) to develop a navigation technique that uses the Simultaneous Localization and Mapping (SLAM) algorithm for controlling the navigation of the smart delivery robot. In particular, a TurtleBot 2 was used to test the navigation control algorithm, before finally implementing these techniques into the six-wheeled robot.
- To construct a standalone, password-protected, medical supply carrier on the robot's body so that only authorized personnel can take the medical supplies from the robot.
- To develop and demonstrate the establishment of an indoor navigation system for the robot to avoid obstacles using Webot simulator software.

1.4 Outline of the Thesis

The rest of the thesis is organized as follows.

- Chapter 2 presents a brief description of the findings of previous research conducted on the design, kinematic modeling, and navigation of differential drive robots.
- Our proposed design and construction of the six-wheeled differential drive robot are presented in Chapter 3.
- Chapter 4 discusses the kinematic modeling developed for our robot to avoid obstacles and optimize energy loss.
- Chapter 5 introduces the concepts and application of the Robot Operating System (ROS) and Simultaneously Localization and Mapping (SLAM) algorithm with the help of TurtleBot 2.
- Chapter 6 demonstrates the navigation of the robot in a simulation environment.

- Finally, conclusions are drawn in Chapter 7.

CHAPTER 2

LITERATURE REVIEW

In the past decade, many researchers worked and proposed different designs and navigation control of six-wheeled differential drive robots. This chapter is organized with an aim to accumulate and review the works that have already been done by the researchers on the design, kinematic modeling, and navigation of six-wheeled mobile robots and use it as the background for the development and implementation of the proposed project. Implementation history of the Robotic Operating System (ROS) and Simultaneous Localization and Mapping (SLAM) algorithm are also presented in this chapter.

2.1 Review of Six-Wheeled Mobile Robot Design

Gu et al., (Gu, Wang, & Zhao, 2007) proposed a lunar-bogie rover with six independently driven wheels mounted on an articulated passive suspension system in which four wheels were steerable. With this design, while traversing rugged terrain, each wheel tended to make contact with the ground. The scheme of this rover is illustrated in Figure 2.1.

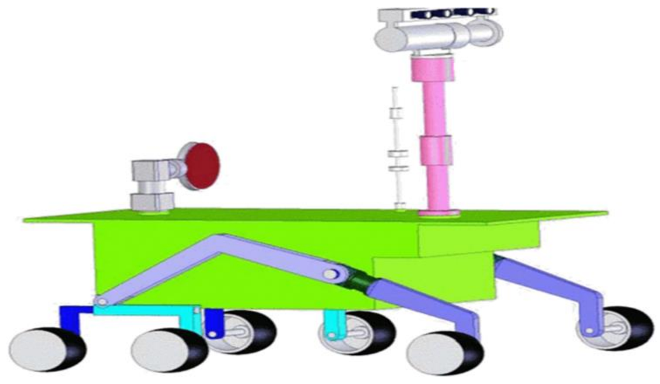


Figure 2.1: Six- wheeled lunar-bogie rover (Gu et al., 2007)

A six-leg-wheel hybrid mobile robot, consisting of six wheels and six swing legs, to move under uneven terrains, was designed by Yanjie Li (Li, 2010). The robot had the characteristics of multi-joints, multi-configuration, and adapting to multi-kinds of environments. The robot also possessed lateral symmetry with individually driven legs and wheels. The wheels were connected with the shaft by swing legs. The structure principle is shown in Figure 2.2. Adjusting the configuration of the robot using swing legs, the robot could improve its maneuverability and its terrain traversability.

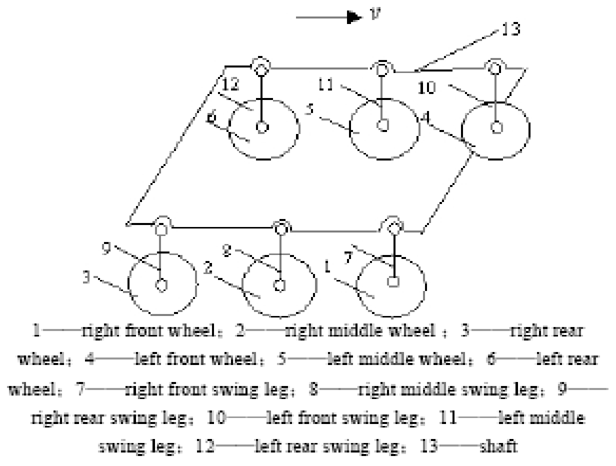

v
13
12
11
10
6
5
4
9
8
7
3
2
1
1——right front wheel; 2——right middle wheel ; 3——right rear wheel; 4——left front wheel; 5——left middle wheel; 6——left rear wheel; 7——right front swing leg; 8——right middle swing leg; 9——right rear swing leg; 10——left front swing leg; 11——left middle swing leg; 12——left rear swing leg; 13——shaft

Figure 2.2: Structure of the six-leg-wheel hybrid robot (Li, 2010)

Gupta and Gupta developed a multi-terrain six-wheeled robot that had the capability to run in rocky and sandy areas, to move on an inclined plane, and to climb on stairs (Gupta & Gupta, 2013). The mechanical structure of the proposed robot consisted of six high frictional rubber wheels, two semicircular rocker structure, one connecting rod, and one rear chassis. The CAD model of the proposed mechanism is shown in Figure 2.3. The robot mechanism had lateral symmetry. The semi-circle rocker wheels were driven simultaneously by one actuator, and the rear wheels were driven individually for getting more power. The total degree of freedom of the six-wheeled multi-terrain robot was ten, which provide more flexibility in rough terrain. The

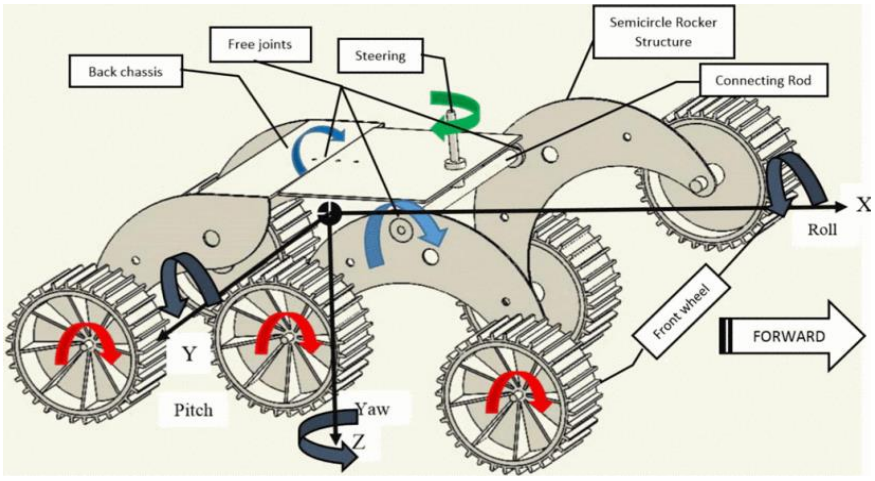


Figure 2.3: CAD model for a six-wheeled multi-terrain robot (Gupta & Gupta, 2013)

semicircular rocker joint moved independently to climb stairs and to adjust the height difference on uneven terrain.

Xixia et al. designed a six-wheeled bow-swing arm robot to cross obstacles (Xixia & Wenwen, 2017). The first half of the robot had two wheels, while the second half of the robot had four wheels. These four wheels could rotate 360 degrees around the hinge. The hinge and hydraulic cylinder were connected with the body of the robot. The design of the bow-arm robot was instrumental in improving the obstacle crossing performance of the robot. Figure 2.4 shows the CAD design of the bow-swing arm robot.

An omnidirectional six-wheel-legged mobile robot was presented by Xu et al. (Xu, Hu, & Li, 2018). It used three walking mechanisms: double crank climbing mechanism, parallelogram rotating mechanism, and balancing mechanism. The six motorized wheels were mounted with steering motors to improve the mobility of the robot. The robot was also equipped with a robot arm for the rescue tasks. The design of the overall size and weight of the robot was in large scales because it would go through rescue missions in complex unstructured environments with necessary payloads. Figure 2.5 shows the overall structure of the robot.

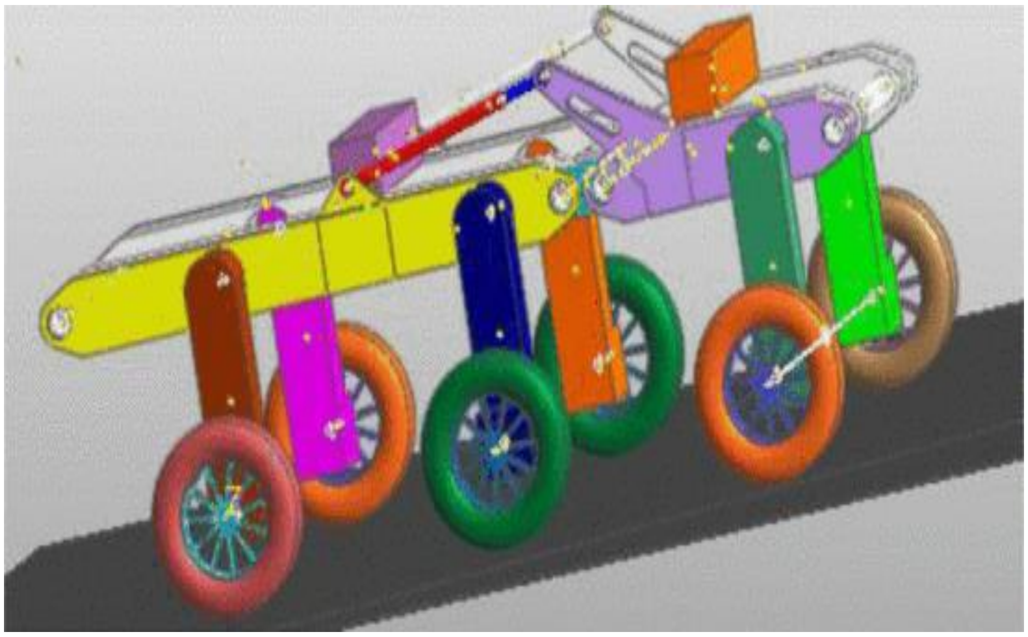


Figure 2.4: CAD design of the six-wheeled bow-swing arm robot (Xixia & Wenwen, 2017)

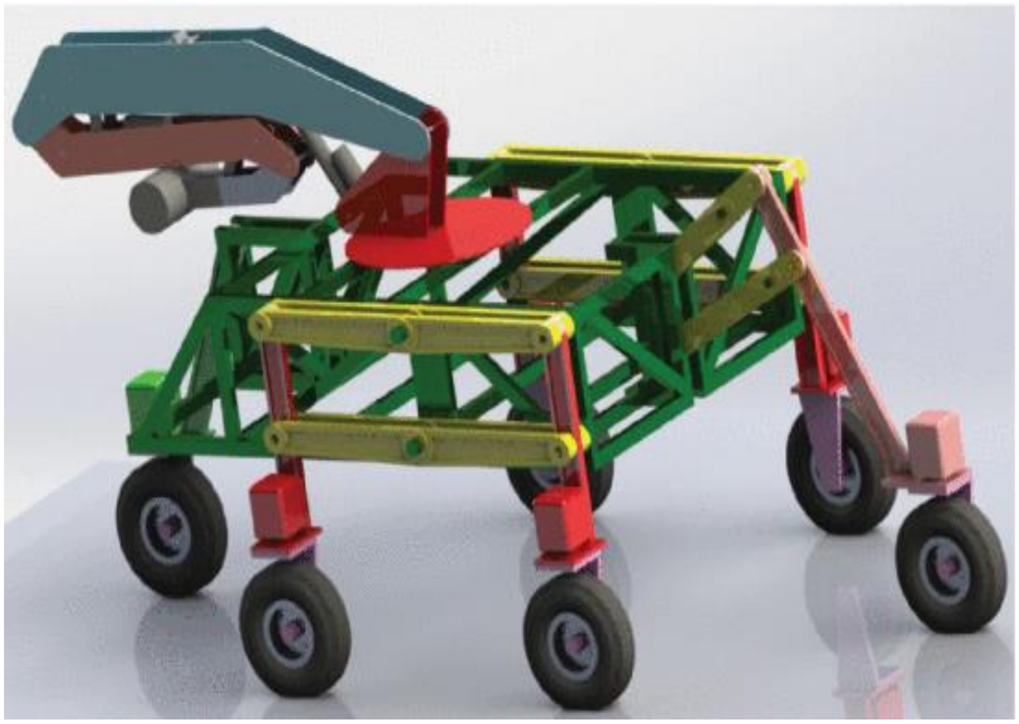
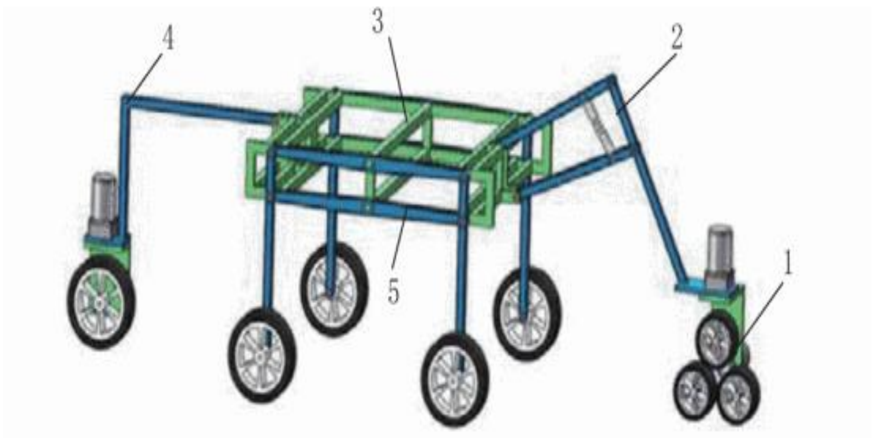


Figure 2.5: Structure of omnidirectional six-leg-wheeled rescue robot (Xu et al., 2018)

Wang et al. (X. Wang, Xu, Feng, & Wu, 2018) demonstrated the design of a fast walking six-wheel-legged robot to overcome obstacles. The structure was divided into five parts, with a total of six independent drive trains. The front-wheel adopted a planetary gear train mechanism, which was connected with the double crank mechanism to form the front body of the robot. The double crank mechanism was installed on the front end of the mainframe, and the parallelogram mechanism was symmetrically mounted on both sides of the mainframe. The balance mechanism was fixed at the rear end of the mainframe, which is called the rear body. Both the balance mechanism and the double crank mechanism were equipped with a steering motor to control the steering of the robot. The 3D model of the robot is shown in Figure 2.6.



1-Planetary train drive mechanism; 2-double crank mechanism; 3-main frame; 4-balance mechanism; 5-parallelogram mechanism

Figure 2.6: 3D model of fast walking six-wheeled-leg robot (X. Wang et al., 2018)

2.2 Review of Kinematic Modeling

Heikkinen et al. proposed three self-tuning fuzzy PID controllers for a three-wheeled differential drive mobile robot (Heikkinen, Minav, & Stotckaia, 2017). The fuzzy PID controllers were used to match the rotation and speed of two non-identical motors, which in turn helped the robot to follow a straight trajectory without deviation. In this process, a robot model was developed incorporating DC motor subsystem, kinematic model subsystem, and controller subsystem, as shown in Figure 2.7. The kinematic model subsystem used speed and position values to calculate the current position of the robot's center point.

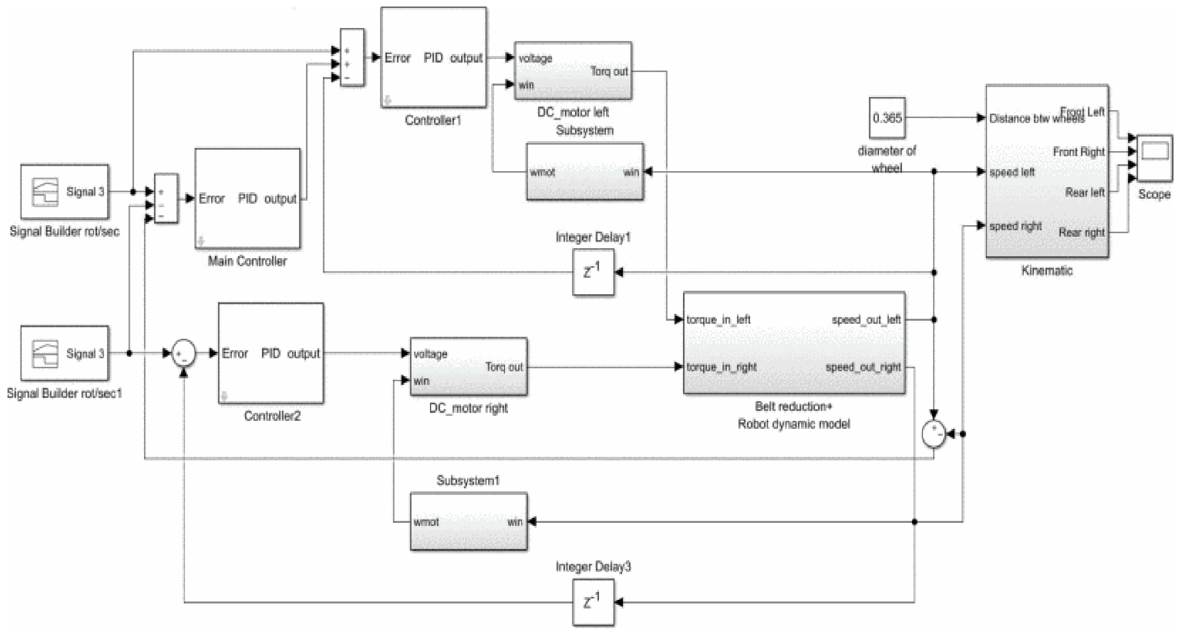


Figure 2.7: MATLAB/Simulink robot model using fuzzy PID controllers

To follow the circular path of the desired radius with constant velocity, a PID controller - used to control the desired angular velocity for both wheels of a two-wheeled differential robot – was presented by Singh et al., (Singh & Chouhan, 2017). They developed a mathematical model of the robot outlining the kinematic and permanent magnet DC motor models. The overall system model in MATLAB/Simulink included a kinematic subsystem, as illustrated in Figure 2.8.

2.3 Review of ROS and SLAM Algorithm on Differential Drive Robots

Ibanez et al. implemented the Simultaneous Localization and Mapping (SLAM) algorithm using the Robot Operating System (ROS) framework and Arduino technology (Ibáñez, Qiu, & Li, 2017). A simple, cost-effective, and SLAM capable three-wheeled differential robot was designed, leaving enough space to incorporate a Kinect 360 camera, a ROS running laptop, electronic components, and batteries. ROS Hydro with packages like *gmapping*, *RViz* installed was used in Linux Ubuntu 12.04 LTS distribution.

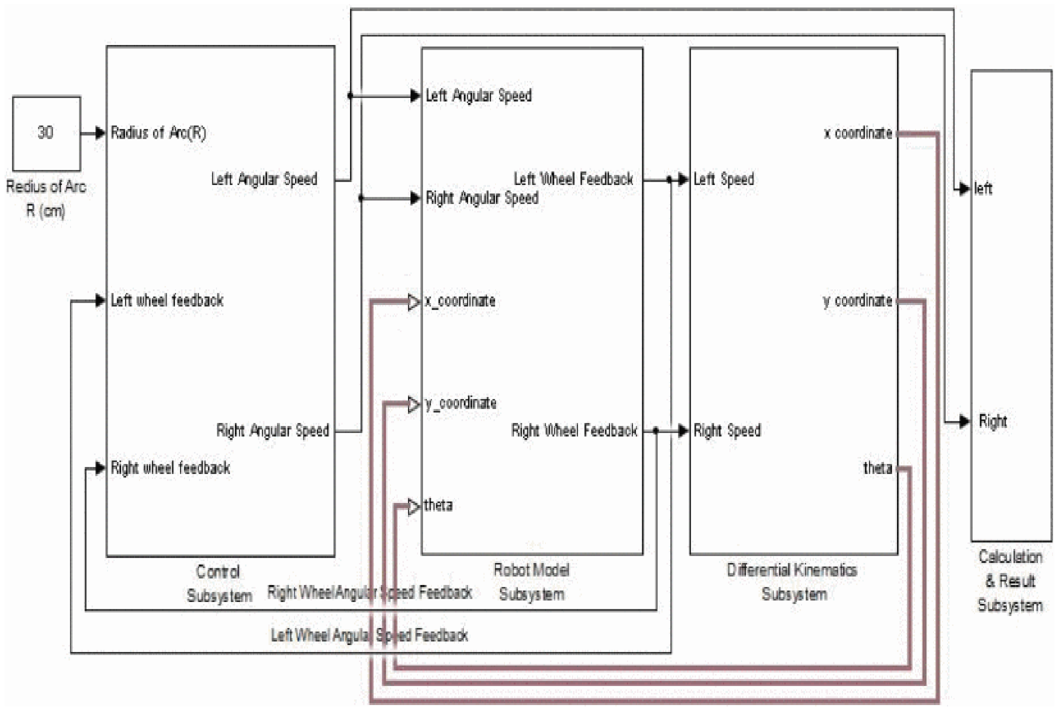


Figure 2.8: Simulation model of a two-wheeled robot in Simulink (Singh & Chouhan, 2017)

A basic Arduino Mega 2560 microcontroller was interfaced with ROS using *ros_lib* library. The robot was used to generate an experimental map of the domestic environment in the *RViz* simulator while *gmapping* package aided the SLAM algorithm to create the map. Executable codes for the project were written both in Arduino microcontroller and ROS hydro.

Two cooperative self-driving autonomous robots, aimed to perform a certain task together in an unknown arena, were built by Park et al., (Park & Lee, 2017). SLAM algorithm was applied to recognize the robot’s positions and environments mapping in an unknown arena. ROS based control system hardware was promoted to maintain communication between two robots.

Researchers intended to compare ROS-based SLAM systems in the past also. In 2018, Filipenko et al., developed a prototype of a four-wheeled unmanned ground vehicle housing a 2D lidar, a monocular camera, and ZED stereo camera for experimental purposes and presented a comparative analysis of mobile robot trajectories computer by various ROS-based SLAM systems (Filipenko & Afanasyev, 2018).

Nowadays, TurtleBot has become a very popular differential drive robotic platform because it is compatible with both ROS and SLAM. Oajsalee et al. evaluated the efficiency of the TurtleBot 3 and ROS in order to survey and map inaccessible areas such as old buildings, a disaster area, and tunnels (Oajsalee, Tantrairatn, & Khaengkarn, 2019). In this evaluation, the TurtleBot 3 was housed with Lidar sensor, OpenCR board, and Raspberry Pi3 board, as shown in Figure 2.9. A lidar sensor was used to scan the environment while *gmapping* utilized the SLAM algorithm to create the map from the scanned data.

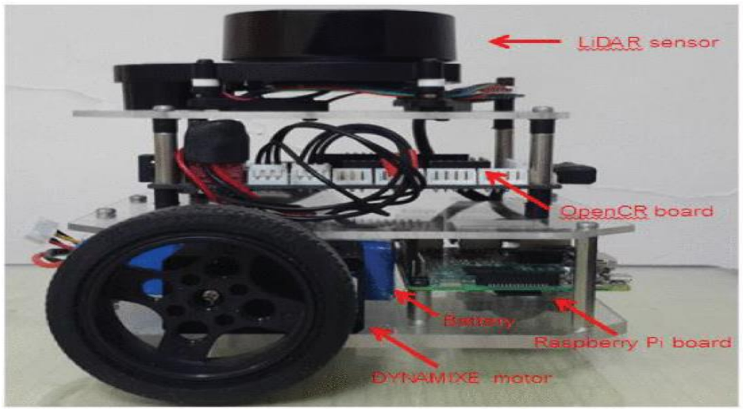


Figure 2.9: Arrangement for survey robot (TurtleBot 3) (Oajsalee et al., 2019)

Liao et al. investigated a SLAM algorithm that uses the Adaptive Monte Carlo particle filter for the localization of the pose of a mobile robot (Liao, Wang, & Yang, 2019). They showed how effectively Adaptive Monte Carlo particle filter-based SLAM algorithm localizes a mobile robot in different indoor environments -static and dynamic. During the simulations, a differential drive robot was localized and autonomously navigated in *Gazebo* and *Rviz* environments.

2.4 Review of Simulation of Differential Drive Robots Indoor Navigation Using the Webot Robot Simulator

There are various software simulators available to test the navigation of mobile robots such as Webot, Gazebo, V-REP, Microwave Robotics Developer Studio, Roboguide. Among them, Webot is the most popular simulation platform for education and research purposes.

In the very early version of Webot simulator software, obstacle avoidance of Khepera – a miniature differential drive robot- was presented by Wang et al. (L. Wang, Tan, & Prahlad, 2000). The basic configuration of the Khepera robot and its development methodologies were introduced at first. Later, they showed how to utilize Webot environments to allow Khepera to move and avoid static obstacles.

Later in 2003, another essential step in the development of mobile robot navigation in both indoor and outdoor environments was taken when Magyar et al., developed the robot control algorithm under Webot professional simulation environment (Magyar, Forhecz, & Korondi, 2003). Two general approaches for robot navigation were also presented: the model-based, which uses a complete model of the environment to navigate efficiently, and the behavior-based approach, which uses no or only sparse modeling of the environment for the navigation. In this preparation, a two-wheeled differential drive robot, Khepera II, was chosen for the testing of robot navigation.

E-puck, a small two-wheeled differential drive mobile robot, basically used for educational purposes, was chosen by many researchers to test navigation in Webot simulation environments. A probabilistic, cell-based, multi-agent map-making and merging scheme was proposed using three E-puck robots (Scott & Yu, 2009). The scheme was shown in simulation to be effective in representing dynamic environments. A static obstacle avoidance controller using fuzzy logic and artificial neural networks was developed (Jeffril & Sariff, 2013). In doing so, both Webot PRO and MATLAB software were used for the simulation. Alajlan et al. investigated the trajectory planning and collision avoidance algorithm for E-puck robots. They developed a line follower E-puck robot that could detect and avoid any obstacles that emerged in its path (Almasri, Alajlan, & Elleithy, 2016). Low-cost infrared sensors - distance sensors and ground sensors - were used to measure and obtain the distance and orientation of the robot. Stan et al. presented a case study of using the Particle Swarm Optimization (PSO) algorithm to create a coordination method for a multi-robot system composed of 10 E-puck robots (Stan & Oprea, 2019).

2.5 Conclusions

The design techniques of the previously built six-wheeled robot have been reviewed alongside the review of kinematic modeling, implementation of the SLAM algorithm, and navigation techniques in the simulation environment. These reviews are instrumental in developing the concepts for our research.

CHAPTER 3

DESIGN AND FABRICATION OF THE SIX-WHEELED DIFFERENTIAL DRIVE ROBOT

3.1 Introduction

In hospitals and medical environments, it is often necessary to carry or move medicines and medical supplies from a central location to other places where they are needed. This operation is typically performed by nurses that take the medicines to the places where they will be used. However, this operation is repetitive, time-consuming, and inefficient. A robot can be used to perform the delivery, freeing the nurse staff to do other more important activities. Also, medical environments and buildings have doors, corridors, and in general small available spaces to maneuver. An autonomous vehicle needs to have the intelligence to move in these spaces.

Due to the above, the primary purpose of this thesis focused on the design, fabrication and development of a navigation control for a six-wheeled differential drive robot for applications on medical or home environments where limited space is available. This chapter presents the first step of the investigation – the design of the robot.

3.2 Robot Characteristics

The proposed six-wheeled robot possesses the following features.

a) A robotic platform with characteristics that enable it to navigate in corridors and rooms on different types of surfaces autonomously and to deliver the medicines and medical materials safely with minimal human intervention.

b) Sensor arrays and corresponding interfaces that allows the detection of objects and autonomous navigation of the robotic platforms in an indoor or hospital environment.
c) A well-planned intelligent control and navigation system.

3.3 Design of the Robot in Solidworks

The entire design of the robot was carried out using Solidworks CAD software. The robot base or chassis was built using an aluminum frame to make it light and sturdy at the same time. The number of wheels chosen for the robot was six instead of four or two. The reason behind designing a six-wheeler robot is that it provides the robot with better traction and balance compared to a four or two-wheeler. Better traction is necessary as the robot will navigate on different types of surfaces and maneuver in small areas inside a hospital or home. The cabinet on top of the robot base was made of thin aluminum sheet and housed compartments of different sizes that are used to store medicines and documents securely.

The dimension of the robot base or chassis was chosen as length=17.25 inches, width= 12.5 inches, and height= 2.25 inches. Under the robot chassis, the six DC motors and a Sabertooth motor controller were mounted. On top of the robot base, the cabinet was located, which houses five compartments. The lower compartment had a height of 8 inches and housed electronic components such as two 12 V batteries to power up the motors, a microcontroller, and a 9V battery to power up the microcontroller. The electrical circuit for the primary and Arduino batteries is shown in Fig. 3.1.

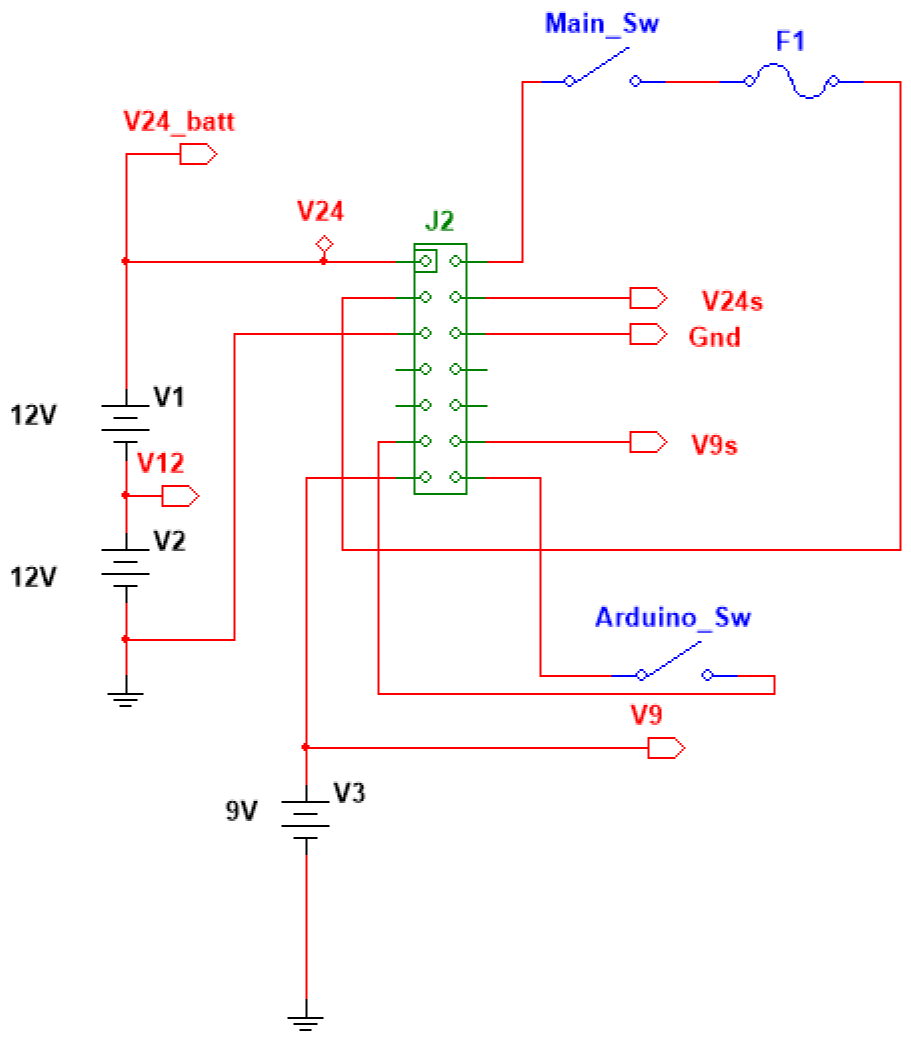


Figure 3.1: Power distribution circuit connection design for primary battery and some parts of the robot

The three compartments in the middle were three drawers inside which medicines and medical supplies can be stored. Each of the middle compartments was 7 inches in height. The 3-inch-high slanted compartment at the top was designed to keep medical documents. A matrix keypad and LCD display screen were also there in the design to make the medicine carrier compartments password protected. Figure 3.2 shows the Solidworks design for some of the parts of the robot.

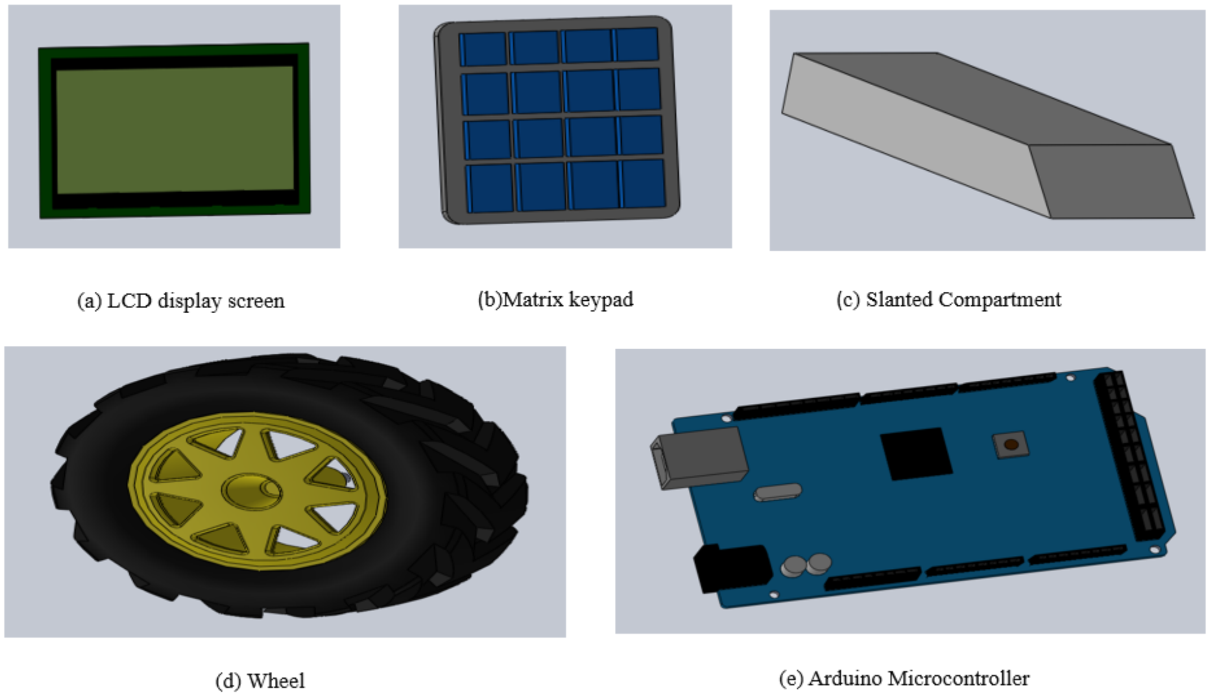
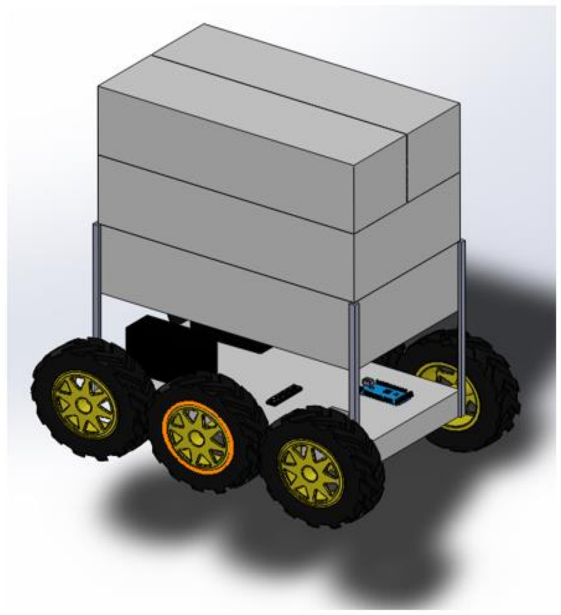


Figure 3.2: Solidworks design for some parts of the robot

The side and front view of the robot showing the electronic components are shown in Figure 3.3. In this Figure, the cover that surrounds the electronic components in the lowermost compartment is made open. Figure 3.4 shows the top view and side view of the final design of the robot.

3.4 Electrical and electronic components used in building the robot base

Electronic components, such as Arduino Mega 2560 microcontroller, Sabertooth 25×2 motor controller, 24V DC motor, battery, sensor, etc. were used on or underneath the robot base in order to prepare it for proper navigation.


(a) Side view of the robot showing electronic components

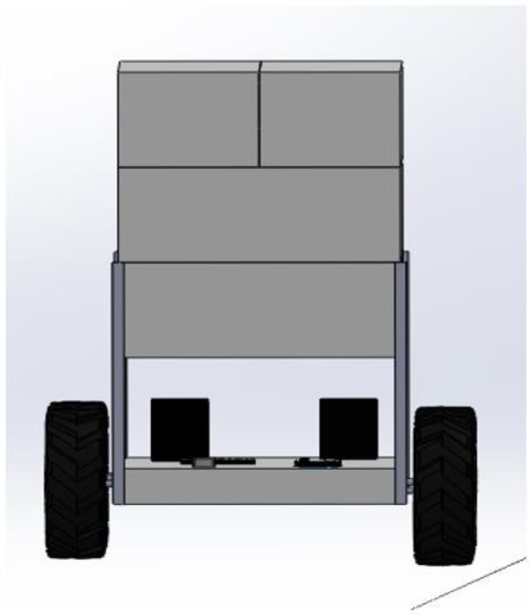
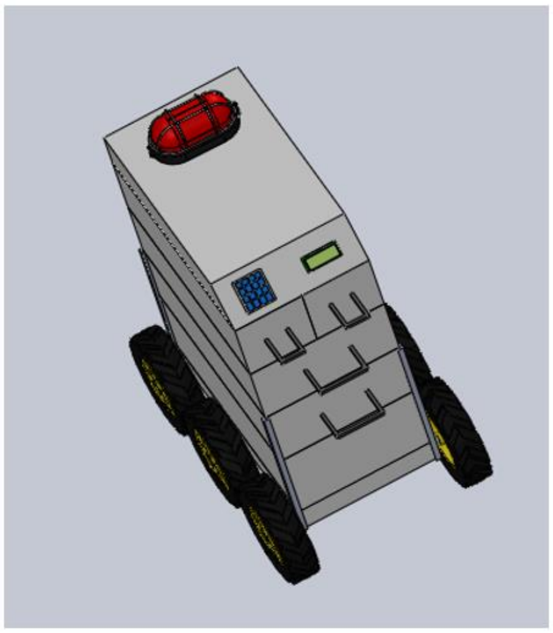

(b) Front view of the robot showing electronic components

Figure 3.3: Design of the robot showing electronic components


(a) Top view of the robot

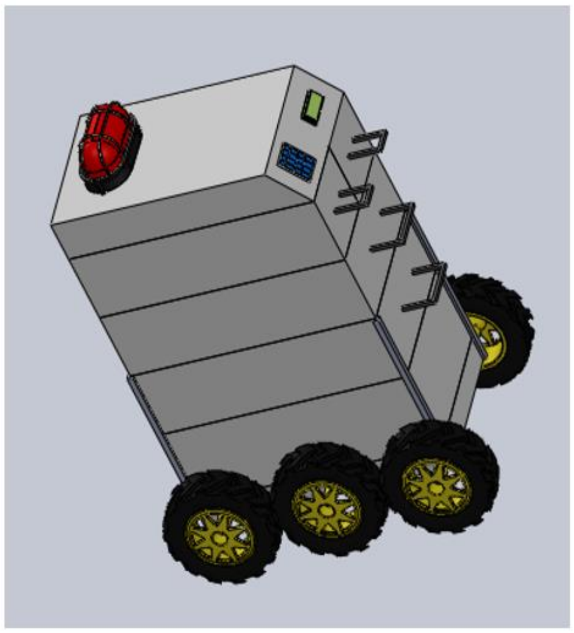

(b) Side view of the robot

Figure 3.4: Final design of the robot

3.4.1 Arduino Mega 2560 Microcontroller

We used the Arduino Mega 2560 microcontroller board because it has a higher number of PWM, analog, and digital input or output pins compared to other Arduino microcontrollers. It has 54 digital input/output pins, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. The microcontroller is shown in Figure 3.4. A 9V DC battery was used as a power source for the microcontroller. The Arduino Mega 2560 also offers a flexible programming environment where the complex task could be efficiently designed. The mainboard for the Arduino Mega is shown in Figure 3.5.

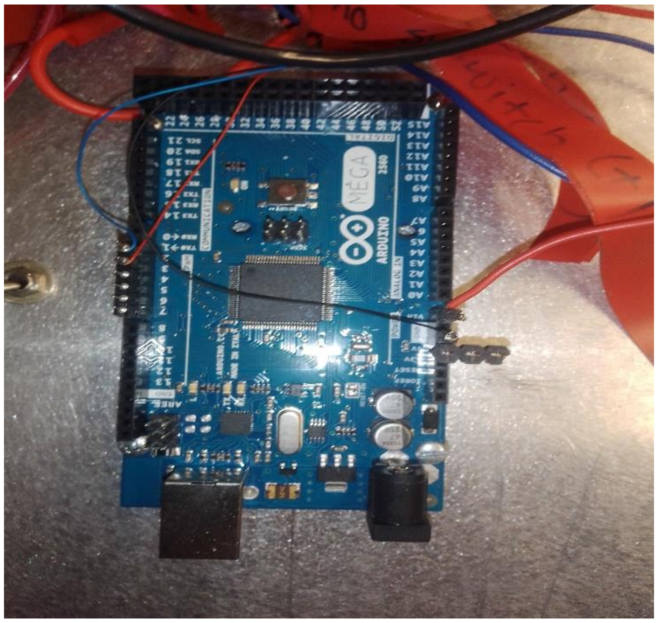


Figure 3.5: Arduino Mega 2560 microcontroller

3.4.2 Sabertooth Motor Controller

A Sabertooth 2×25 motor controller with an input nominal voltage of 6-30V was used to control the rotation of the wheels. Two 12V Lithium-ion batteries connected in series were used to provide the required voltage to the motor controller. It can supply up to 25A continuous output current per channel. This motor controller has four terminals – M1A, M1B, M2A, M2B. Three 24V DC motors were connected to M1A and M1B terminals. The other three 24V DC motors are connected to M2A and M2B terminals. The input signals that control the Sabertooth motor controller are connected to two terminals - S1 and S2. Any operating mode among analog input, R/C input, simplified serial, and packetized serial can be selected to initiate the operation of the motor controller. The Sabertooth controller is shown in Fig. 3.6. The circuit diagram for the Motor controller connections to the 24V DC motors is shown in Figure 3.7.

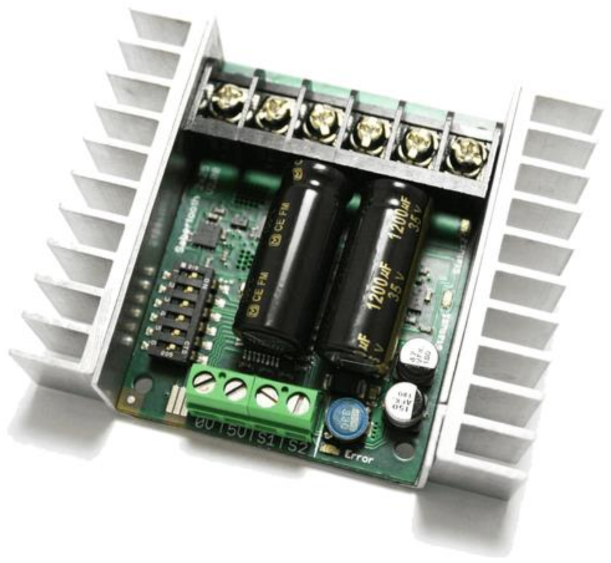


Figure 3.6: Sabertooth motor controller

Figure 3.8 shows the connection of the motor controller with batteries, four of the motors, and Arduino. In this Figure, right-sided motors connected in parallel are attached to the M1A and M1B terminals, whereas left-sided motors connected in parallel are attached to the M2A and M2B terminals. The battery terminals are connected to B+ and B- terminals of the motor controller. Signal pins are also there in the motor controller to maintain communication with the Arduino.

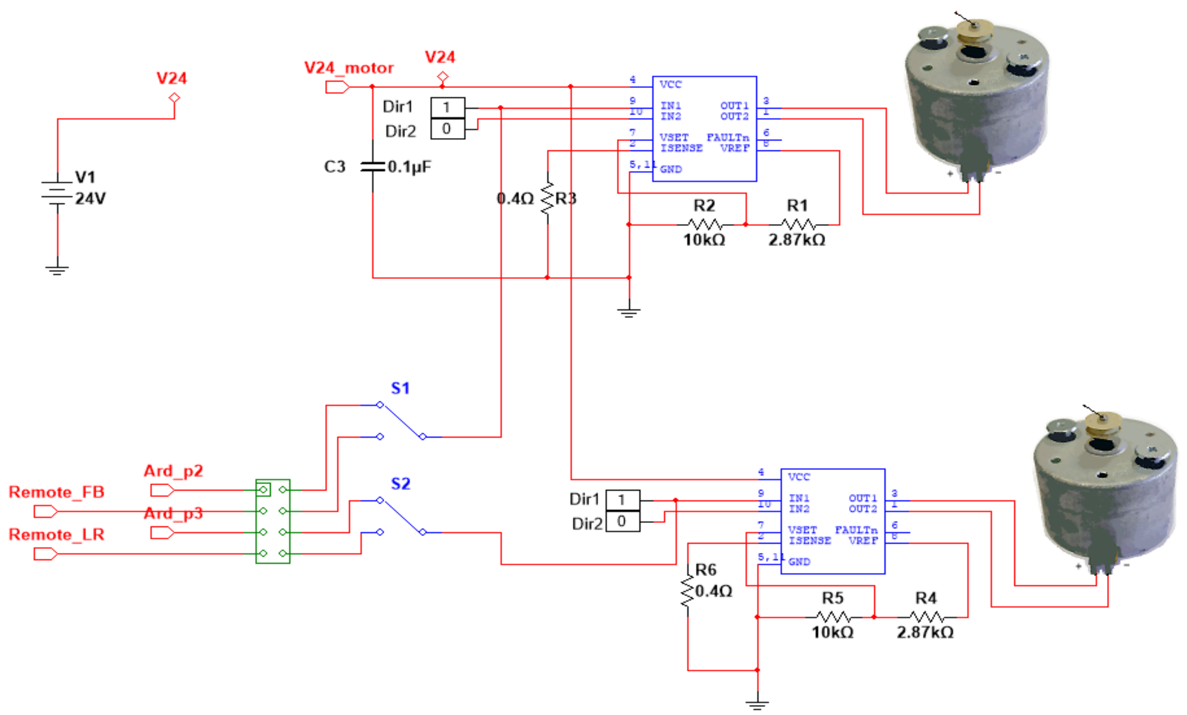


Figure 3.7: Motor controller circuit diagram

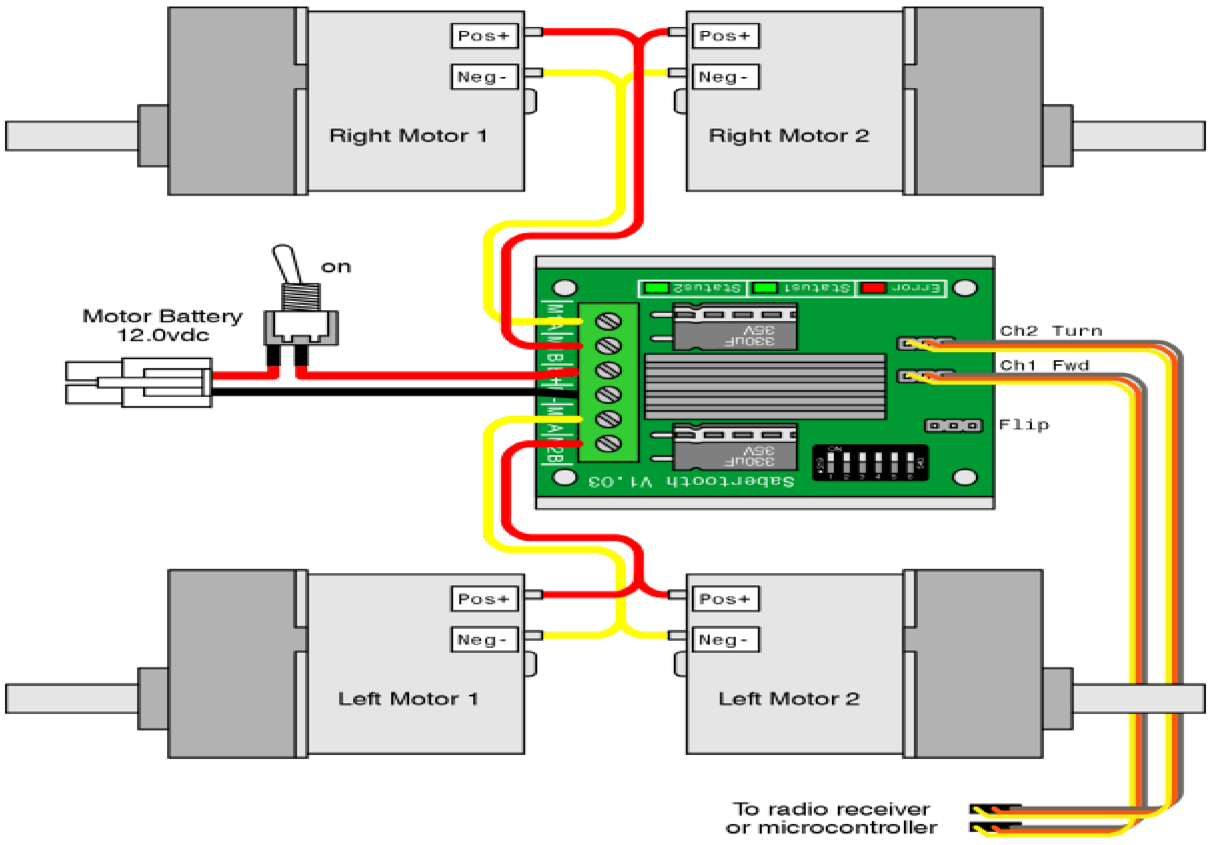


Figure 3.8: Motor controller connection diagram

3.4.3 Sensors

The primary sensors needed for an autonomous robot to function are those that can detect objects in the vicinity of the robot. For this purpose, sensors need to have the capability to measure the distance to the nearest object. Several kinds of distance sensors with object detection and range measurement capability are available in the market. These sensors operate by broadcasting a signal, picking up the reflection of the signal, and calculating the distance based on the signal latency. The transmitted signal can be light, sound, or radiofrequency. In this project, several distance sensors: ultrasonic, infrared, and lidar were used. Ultrasonic sensors rely on high-frequency sound waves, whereas infrared sensors measure and detect infrared radiation in its surrounding environment. The infrared sensor, ultrasonic sensor, and lidar sensor used in our project are shown in Figure 3.9, Figure 3.10, and Figure 3.11, respectively.

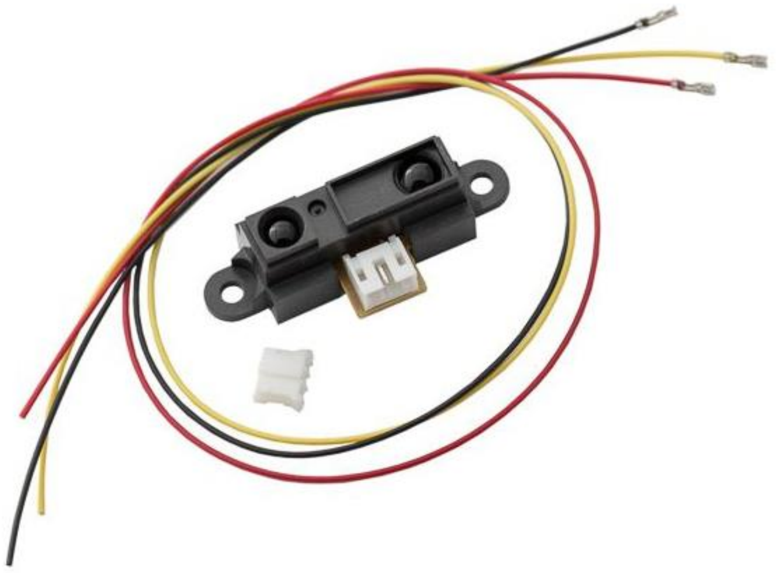


Figure 3.9: Infrared sensor

In the infrared sensor, a beam of modulated infrared light from the sensor illuminates an object. The beam reflects off the object and bounces back into the sensor. The reflected beam is focused on a Position-Sensitive Device or PSD. The PSD has a surface whose resistance changes

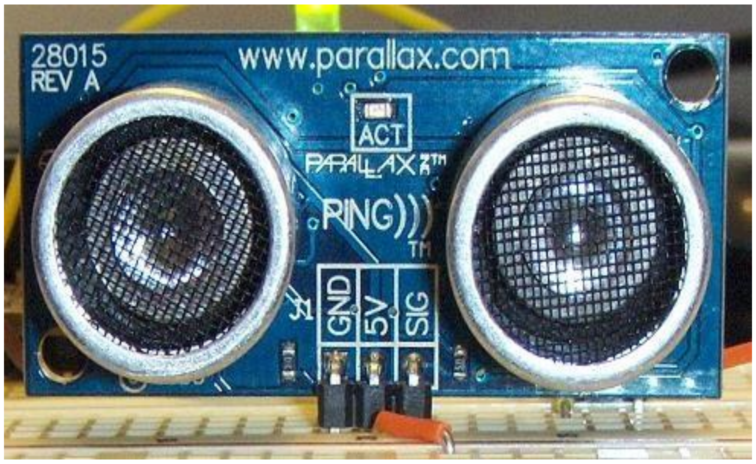


Figure 3.10: Ultrasonic sensor

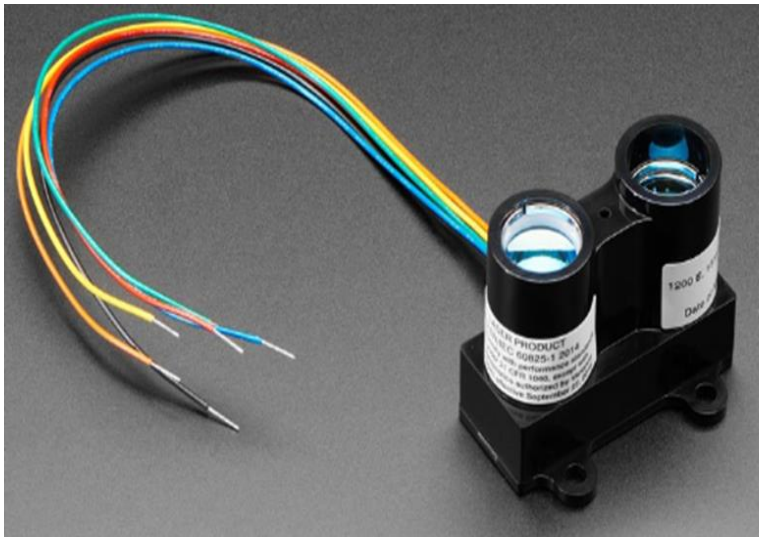


Figure 3.11: Lidar sensor

depend on where light strikes it. As the distance between sensor and object changes, so does the linear position of the light falling on the PSD, as illustrated in Figure 3.12. Circuitry in the sensor monitors the resistance of the PSD element and calculates the distance based on this resistance. Besides, as the distance of the object from the infrared sensor decreases, both the resistance and analog voltage increase. Thus, the voltage change depends on how close an object is to the sensor. But when the object stays in very close proximity to the sensor, the voltage always decreases. Figure 3.13 shows the typical output voltage response of an infrared sensor.

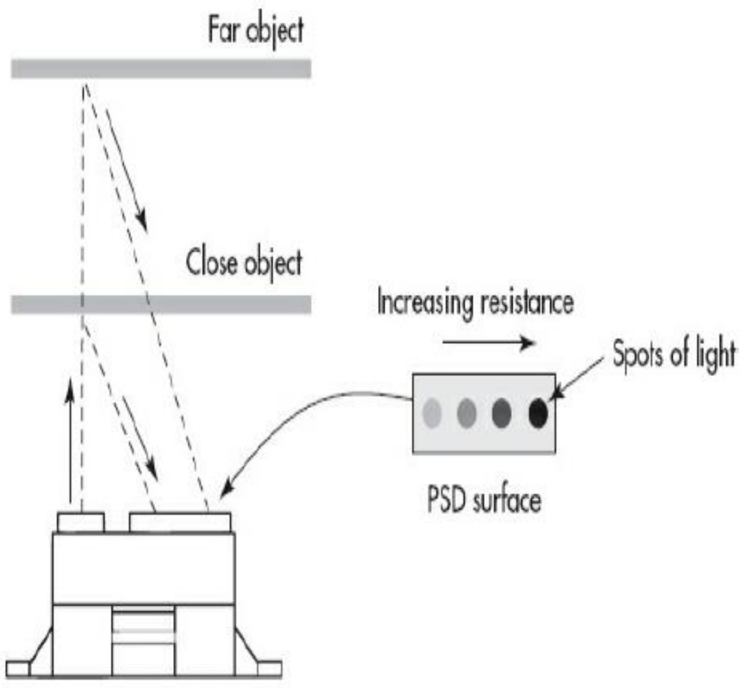


Figure 3.12: Infrared sensor working principle

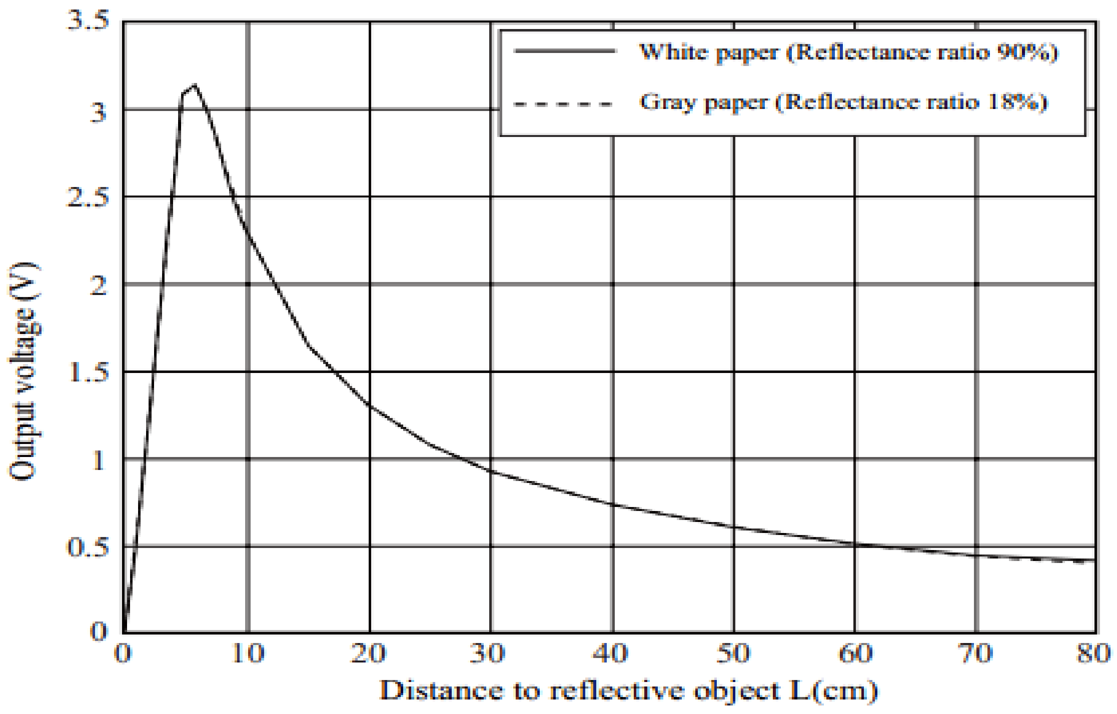


Figure 3.13: Typical analog voltage output response of an infrared sensor

Ultrasonic sensors use high-frequency sound outside the range of human hearing; the typical frequency is around 40 kHz. The sound is emitted as a quick burst from a transducer, a kind of speaker. The sound bounces off an object, and the echo is received by the same or another transducer. A circuit then computes the time it took between the transmit pulse and the echo and comes up with a distance. Ultrasonic sensors can be operated using either mode 1 or mode 2. Ultrasonic sensor with Mode 1 operation has a separate trigger and echo pins, whereas, in ultrasonic sensor having Mode 2 operation, the same pin works to trigger the sound and receive the echo. In our project, we used an ultrasonic sensor having a Mode 2 operation. The signal control diagram of a typical Mode 2 ultrasonic sensor is shown in Figure 3.14.

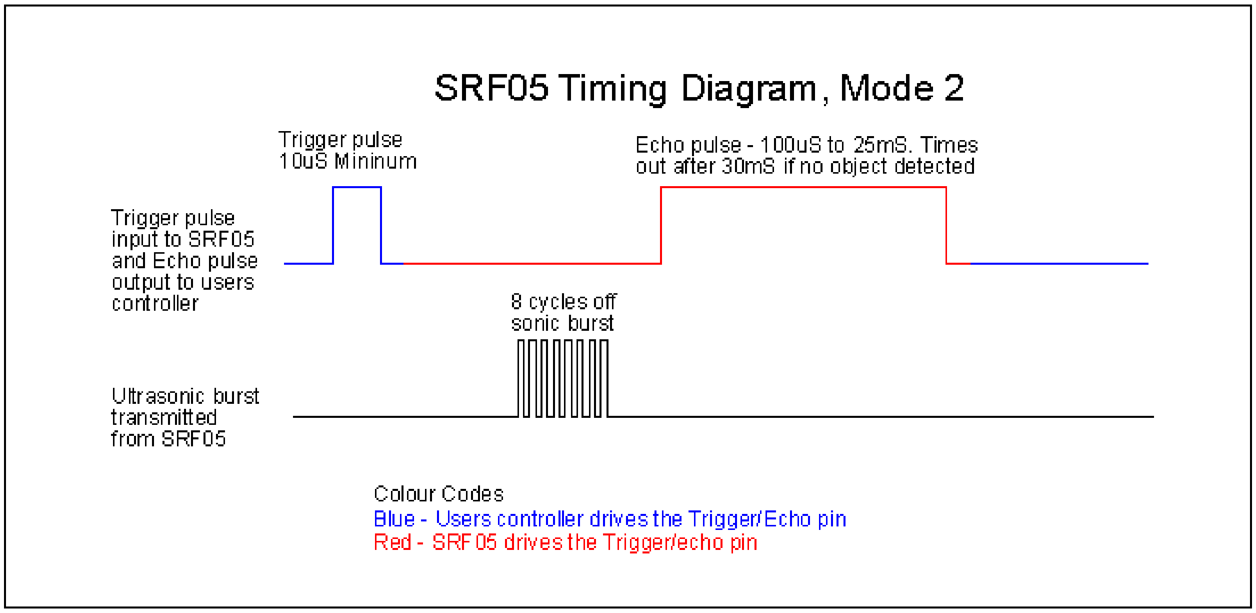

SRF05 Timing Diagram, Mode 2
Trigger pulse 10uS Mininum
Echo pulse - 100uS to 25mS. Times out after 30mS if no object detected
Trigger pulse input to SRF05 and Echo pulse output to users controller
8 cycles off sonic burst
Ultrasonic burst transmitted from SRF05
Colour Codes
Blue - Users controller drives the Trigger/Echo pin
Red - SRF 05 drives the Trigger/echo pin

Figure 3.14: Ultrasonic sensor signal control diagram

From the signal control diagram in Figure 3.14, it is understandable that we only need to supply a short 10uS pulse to the trigger input of the ultrasonic to start the ranging. The ultrasonic sensor will then send out eight cycles burst of ultrasound at 40kHz and raise its trigger line high. It then listens for an echo, and as soon as it detects one, it lowers the trigger line again. The echo pulse is, therefore, a pulse whose width is proportional to the distance to the object. By timing the pulse, it is possible to calculate the range in inches or centimeters or anything else. If nothing is detected, then the ultrasonic will lower its trigger line anyway after about 30mS.

A typical lidar sensor emits pulsed light waves into the surrounding environment. These pulses bounce off surrounding objects and return to the sensor. The sensor uses the time it took for each pulse to return to the sensor to calculate the distance it traveled. Repeating this process, millions of times per second creates a precise, real-time 3D map of the environment. An onboard computer can utilize this map for safe navigation. Figure 3.15 shows a typical signal control diagram of a lidar sensor. The figure depicts that a minimum of 10µS pulse is needed to trigger input of the lidar to start the ranging. After a pause, the echo pulse is received, whose width is proportional to the distance to the object.

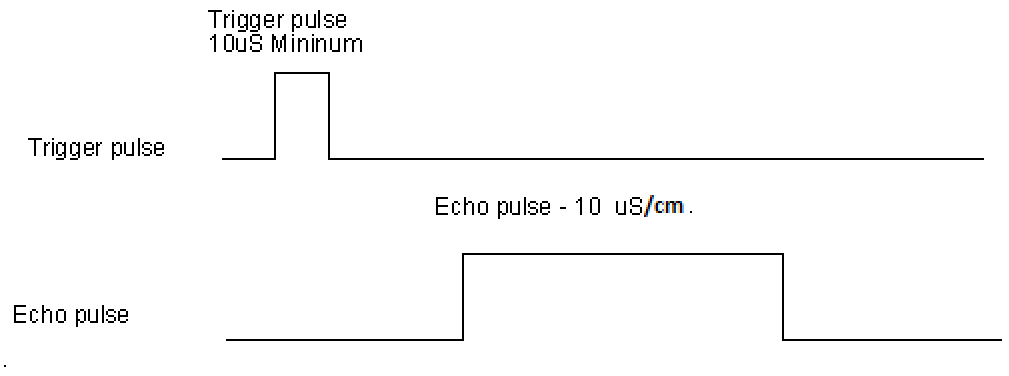
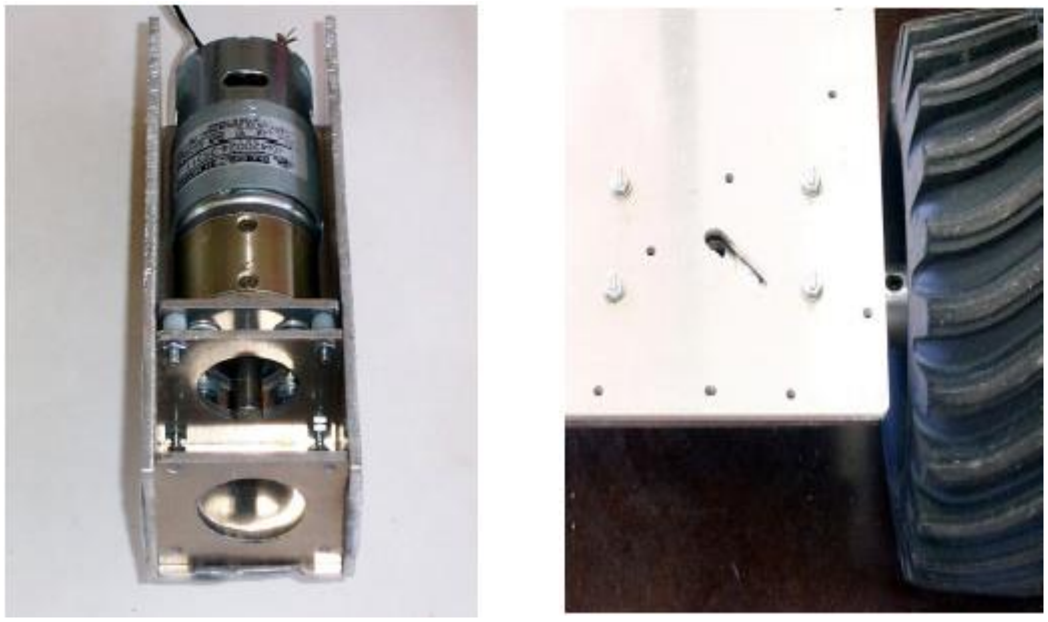


Figure 3.15: Lidar sensor signal control diagram

3.5 Fabrication of the Robot Base

Six 24V DC motors coupled with six wheels and the Sabertooth 2×25 motor controller were mounted underneath the robot base, while batteries and Arduino Mega 2560 microcontroller are located on the robot base. Power sources: two 12V battery and a 9V battery were placed on the top of the robot base. Ultrasonic and infrared sensors used to detect the obstacles were mounted in front of the robot base. Figure 3.16 (a) shows the 24V DC motor mount, and Figure 3.16 (b) shows the wheel attachment to the motor. Figure 3.17 and Figure 3.18 show the top view and the bottom view of the completed robot base, respectively.



(a)24V DC motor (b)Wheel attachment

Figure 3.16: 24V DC motor and Wheel attachment

3.6 Design and Fabrication of Password-Protected Medical Supply Cabinet

A standalone smart carrier box was built using a thin aluminum sheet inside which we can put medicines, hospital documents, or small-sized necessary things. The box was made password protected with the help of a 3×4 matrix keypad, a 16×2 LCD display, an Arduino Uno microcontroller, and a servo motor. The Arduino Uno was powered from a DC battery. Figure 3.18 illustrates the Solidworks design of the smart medicine carrier box.Only the authorized person is allowed to open the box provided that the password he enters on the keypad matches with the actual password.This standalone smart carrier box can be placed on the top of the robot base. Thus, the robot will be able to navigate indoors along with the smart carrier box and deliver medical

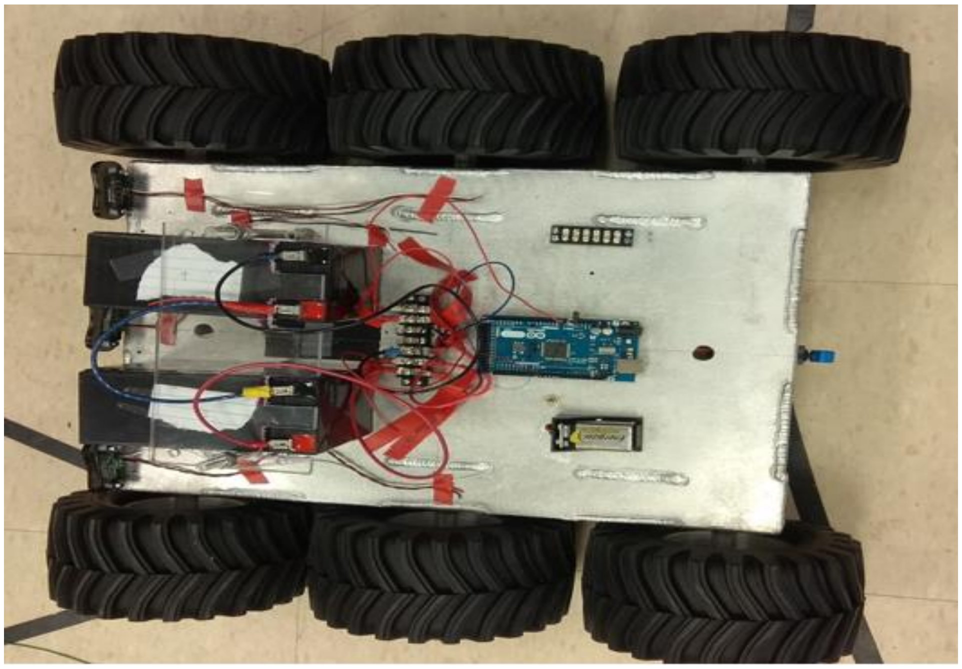


Figure 3.17: Top view of the fabricated robot base

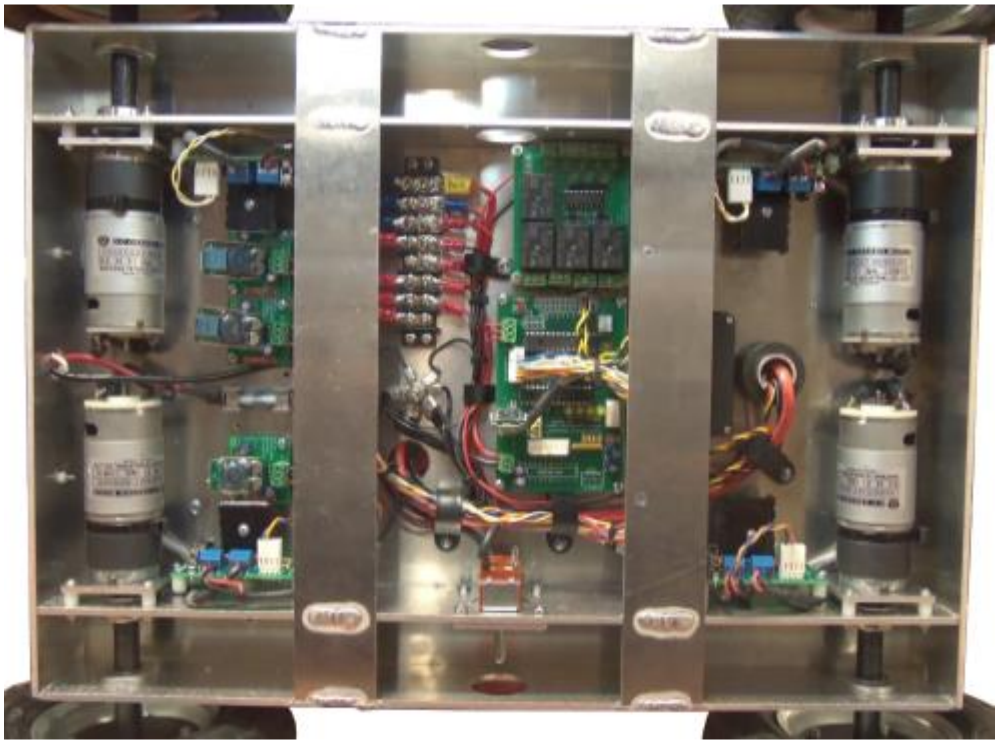


Figure 3.18: Bottom view of the fabricated robot base

supplies and small-sized necessary things to the authorized persons. The design and fabrication of the smart medical supply carrier are shown in Figure 3.19 and Figure 3.20, respectively.

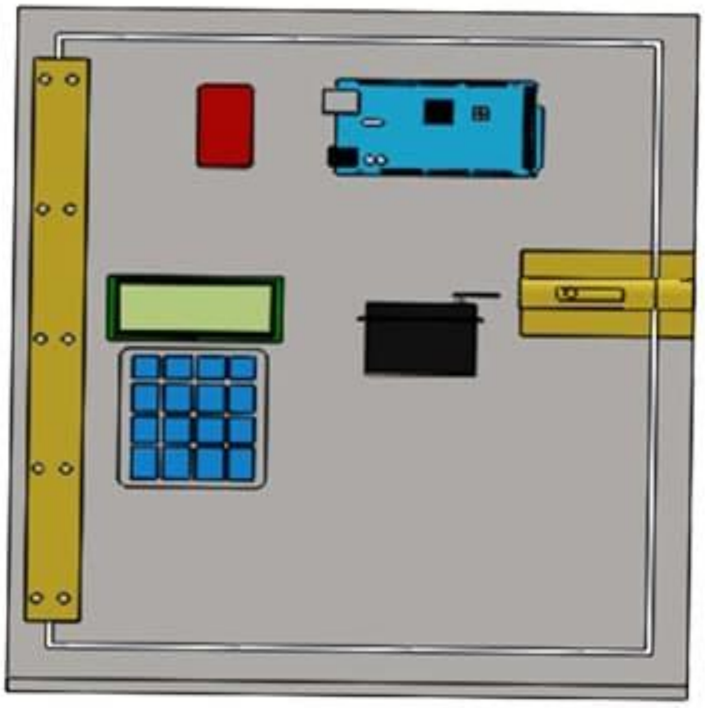


Figure 3.19: Design of smart medical supply carrier box in Solidworks

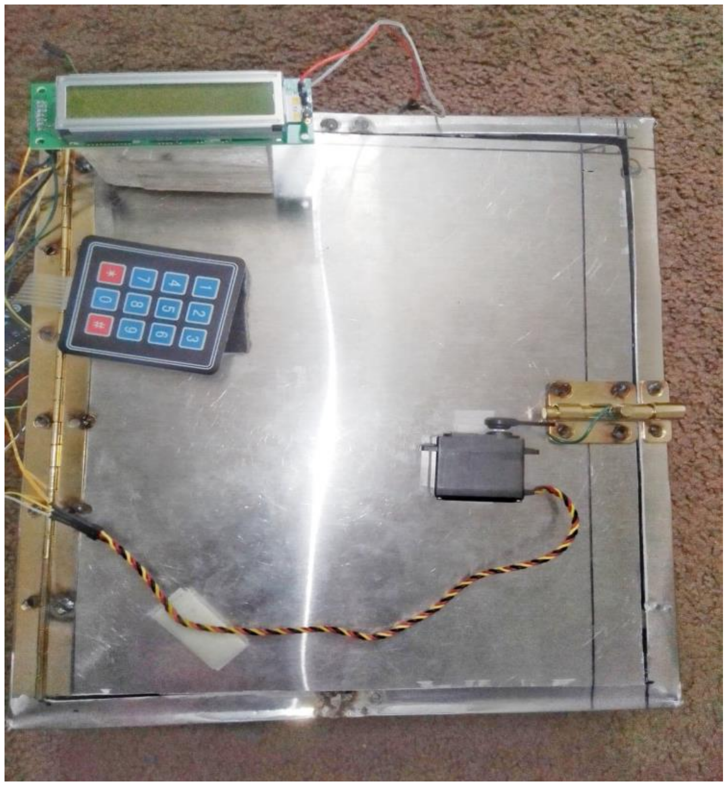


Figure 3.20: Prototype and Testing of the password-protected smart carrier box

3.7 Conclusions

This chapter presents the complete design and fabrication for our six-wheeled robot. In addition, the fabrication process of the robot base and password-protected medical supply carrier have been discussed. In doing so, a detailed connection for the electrical and electronic components used to build the robot has been presented. The robot maneuver was tested using a basic navigation program implemented in the Arduino microcontroller.

CHAPTER 4
KINEMATIC MODELING OF THE ROBOT

4.1 Introduction

Developing mobile robots is a very complex task. Most robotics projects require time and cost to develop, which is why it is very important to have a generic mobile robotic simulation platform for the purpose of research and development. This chapter presents the research to develop a MATLAB/Simulink kinematic model of the proposed six-wheeled differential drive autonomous navigation robot with obstacle avoidance capability. Kinematics is the study of the mathematics that describes the geometric relationships that govern the system as well as deals with the control parameters and behavior of a system in state space (Prio & Rios, 2019). Understanding the kinematics of mobile robots is extremely important because they provide a platform for designing navigation controllers. They also provide constraints on the motion of the robot, which should be taken into account before doing any kind of navigation planning.

There are two primary types of navigation techniques: reactive control and planning control. Reactive control is a low-level control in which the robot navigates in the environment while avoiding collision with real-time obstacles using its sensory system. In planning control, the robot's motion profile is generated before the robot has even moved. This technique doesn't utilize real-time information from the robot's workspace (Prio & Rios, 2019).

In this research, we will be mainly focused on reactive control because it deals with the issues of real-time obstacle avoidance (Fu-guang, Peng, Xin-qian, & Hong-Jian, 2005). There are many reactive control techniques available in the literature, but the most attractive of them is the

Potential Field method. This method is attractive because of its simplicity and time efficiency (Fu-guang et al., 2005). It has also been widely used by many researchers.

4.2 Reasons behind Investigating Kinematic Modeling

Designing an intelligent controller is a must to build an autonomous mobile robot capable of performing complicated tasks (Prio & Rios, 2019). An intelligent controller aids the robot to move from one place to another while avoiding a collision. One of the notable steps to design a controller is to understand the mechanical behavior of mobile robots - kinematics and dynamics of the system (Siegwart, Nourbakhsh, & Scaramuzza, 2011). Our research explores the development of a model for the kinematics of the six-wheeled differential drive mobile robot, its links to the dynamics, and some considerations regarding more accurate modeling for energy optimization. The kinematic model will also be used to help in developing optimal use of energy while the robot is navigating and developing a hybrid control system for navigation of the robot.

4.3 Background

Basic concepts of kinematic modeling and potential field method is discussed in this section.

4.3.1 Kinematic Modeling

The kinematic modeling of mobile robots is a bottom-up process. Every single wheel contributes to the robot's overall motion and also imposes constraints. Therefore, the forces or the driving vectors on each wheel must be clearly defined. The control or driving vectors on a differential drive robot are the rotation speed of the DC motors, which are attached to the wheels.

The difference in speed of the DC motors allows the robot to perform various rotations and translations (Siegwart et al., 2011).

In the literature, a three-wheeled differential drive robot with two fixed standard rear wheels and a caster in the front was proposed, as illustrated in Figure 4.1 (Kim, Kim, & Systems, 2007).

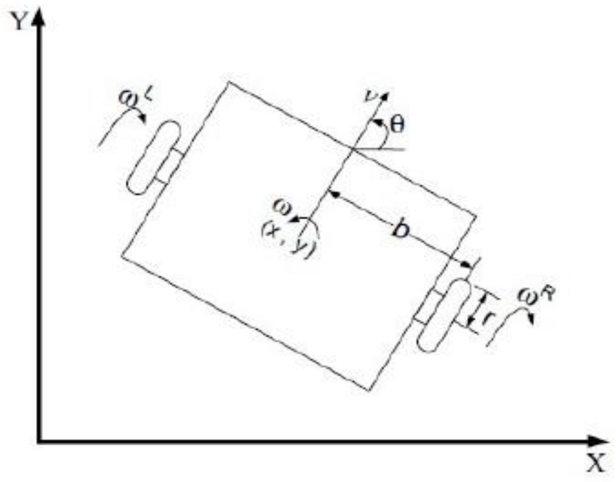


Figure 4.1: Basic wheeled mobile robot kinematic diagram

The drive wheels have a radius of length r. The wheels are at a distance 2b from each other. The axes X and Y define the arbitrary inertial base on the plane as a global frame of reference. The angular velocities of the right and left wheels are ω_R and $\omega_{L,}$ respectively. Initially, we assume that the speed of the DC motors and the wheels are exactly the same. It is also assumed that there are no obstacles in the surrounding.

The robot's position in the global frame of reference is represented by a vector of three elements called the pose of the robot. It is given by (Prio & Rios, 2019),

$$\xi = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{4.1}$$

Then the differential drive robot's kinematics are defined by (Prio & Rios, 2019),

$$x = v\cos\theta \tag{4.2}$$

$$y = v\sin\theta \tag{4.3}$$

$$\theta = \omega \tag{4.4}$$

Here, θ is the heading of the robot, v is the linear velocity, and the ω is the rotational velocity of the robot. The v and ω relationship to the motor's speed can be described by (Prio & Rios, 2019),

$$v = \frac{r(\omega_R + \omega_L)}{2} \tag{4.5}$$

$$\omega = \frac{r(\omega_R - \omega_L)}{2b} \tag{4.6}$$

4.3.2 Potential Field Method

The potential field method is a low-level navigation technique in which the robot is represented by a point in a configuration space moving under the influence of a potential field that is created as a result of attraction by the goal point and repulsion by the obstacles (Yin, Yin, & Lin, 2009). The resultant of these forces determines the direction of motion for the robot. This force is used to calculate the driving vectors or the rotational velocities of the wheels of the robot. This resultant force is given by,

$$F_{att} = F_{att} + F_{rep} \tag{4.7}$$

Here, F_{att} is the force applied by the goal on the robot, and the F_{rep} is the force applied by the obstacle. Many functions have been developed in the literature to model both the forces.

4.4 Assumptions for Our Model

Both attractive and repelling potential field was assumed to be included in the potential field method for our work. The assumptions made for our potential field model are presented next.

4.4.1 Assumptions Made for the Attractive Potential Field

We assumed that the goal point is a circle with a very small radius r_g. The goal attracts the robot by a force that is directly proportional to d_1, which is the Euclidean distance between the goal and the robot. If the position of the robot is (x,y), then the distance is given by (Prio & Rios, 2019),

$$d_1 = \sqrt{(x_g - x)^2 + (y_g - y)^2} \tag{4.8}$$

The forces here will be represented by how fast the robot changes its coordinates towards the goal. Considering the field has a spread of s, the following three assumptions were applied in order to generate the force field.

$$\text{If, } d_1 < r_g,\ \Delta x = \Delta y = 0 \tag{4.9}$$

$$\text{If, } r_g \le d_1 \le s + r\ ,\ \Delta x = \alpha(d_1 - r_g)\cos\theta\ ,\ \Delta y = \alpha\,(d_1 - r_g)\sin\theta \tag{4.10}$$

$$\text{If, } d_1 > s + r_g,\ \Delta x = \alpha\, s\cos\theta\ ,\ \Delta y = \alpha\, s\sin\theta \tag{4.11}$$

The first assumption implied that the robot is not moving when it is within the radius of the goal, and hence no force is applied. The second assumption implied that when the robot is outside the circle of goal but inside the spread of the field, the force by which the goal attracts the robot is proportional to the distance between the goal and the robot. The third assumption suggested that when the robot is outside the spread of the field, the force by which the goal attracts the robot is maximum and proportional to the spread s of the field. The parameter α is greater than zero, and it is used to define the strength of the attractive force field. The higher the value of α is, the faster the robot goes towards the goal. The angle by which the robot moved towards the goal is given by (Prio & Rios, 2019),

$$\theta_g = tan^{-1}\left(\frac{y_g - y}{x_g - x}\right) \tag{4.12}$$

4.4.2 Assumptions Made for the Repelling Potential Field

The second assumption was that the obstacle, regardless of its shape, is enclosed in an imaginary circle of radius r_0. This imaginary circle is a barrier that the robot cannot cross. Anytime the robot comes closer to the obstacle, it will be repelled. Considering the coordinates of the center of the obstacle as (x_0,y_0) and the position of the robot is (x,y), then the distance, d_2 between the obstacle and robot is given by (Prio & Rios, 2019),

$$d_2 = \sqrt{(x_0 - x)^2 + (y_0 - y)^2} \tag{4.13}$$

Again, considering the field has a spread of s, we had the following two assumptions (Prio & Rios, 2019).

$$\text{If, } d_2 < r_0,\ \Delta x = -\beta\ d_2 \cos\theta \text{ and } \Delta y = -\beta\ d_2 \sin\theta \tag{4.15}$$

$$\text{If, } d_2 > r_0,\ \Delta x = \Delta y = 0 \tag{4.16}$$

The first assumption implied that inside the circle of an obstacle, the repelling force by the obstacle on the robot sustains and points out from the center of the obstacle. The repulsion force here means how fast the robot moved away from the obstacle. β is always greater than zero and is used to scale the strength of the repulsive field. The second assumption meant that no repelling force is applied to the robot by the obstacle if the robot is away from the vicinity of the robot. The angle by which the robot moved away from the obstacle due to repelling force is given by (Prio & Rios, 2019),

$$\theta_0 = tan^{-1}(\frac{y_0 - y}{x_0 - x}) \qquad (4.17)$$

4.4.3 Assumptions Made for the Resultant Potential Field

Once both the attractive field and repulse field were calculated in terms of the rate of change of the coordinates, they were added together to get the resultant force field. This resultant field gave us the linear velocity v and the direction of the motion θ of the robot.

$$v = \sqrt{(x_0 + x_g)^2 + (y_0 + y_g)^2} \qquad (4.18)$$

$$\theta = tan^{-1}(\frac{y_0 + y_g}{x_0 + x_g}) \qquad (4.19)$$

4.5 MATLAB/Simulink Kinematic Model of the Robot

In order to develop a realistic model for the six-wheeled robotic platform, we followed the steps described next.

Step 1: The kinematic modeling process started with measuring parameters of one of the DC motors. These measured parameters are responsible for energy loss in the motor. Such parameters for the motor are armature resistance, armature inductance, armature capacitance, the moment of inertia, and angular velocity. At first, the DC motor was modeled using the following two equations (Prio & Rios, 2019),

$$V = R_a I_a + K_b \omega_{dc} \qquad (4.20)$$

$$J \omega_{dc} + F v \omega_{dc} = K_t I \qquad (4.21)$$

Here V is the terminal voltage, R_a is the armature resistance, I_a is the armature current, K_b is the back emf constant, and ω_{dc} is the angular velocity of the motor, J is the moment of inertia the DC motor, and F_v is the viscous damping coefficient. To measure the armature resistance, a locked rotor test was conducted. In this test, the DC motor was kept locked while applying directly to the motor terminals. We measured the current withdrawn by the DC motor while the rotor was still in the locked state. Because of the locking, there was no back emf. Hence, the value of the back emf constant, K_b became zero. Thus, equation (4.20) was reduced to the following equation.

$$R_a = \frac{V}{I_a} \tag{4.22}$$

Therefore, the armature resistance of the motor was measured using equation (4.22). Afterward, we took the help of an LCR meter to measure armature inductance and armature capacitance. The armature resistance, armature inductance, armature capacitance we recorded for the DC motor were 3.3994 Ω, 10.19 mH, and 19.80 nF, respectively.

Step 2: The next step in the process was to model equations (4.2), (4.3), (4.4), (4.5), (4.6) to obtain the outputs of the plant: x, y, and θ. This step completed the modeling of the plant.

Step 3: The final step was to implement the navigation controller to avoid the obstacles. For that purpose, we used MATLAB Function block to implement the logic presented by the assumptions for the potential field method. The outputs of this block were the linear velocity, v, and the direction of motion, θ of the robot. These two parameters were used by equations (4.5) and (4.6) to generate the control commands. Figure 4.2 shows the complete kinematic model for the proposed six-wheeled differential drive robot in MATLAB/Simulink.

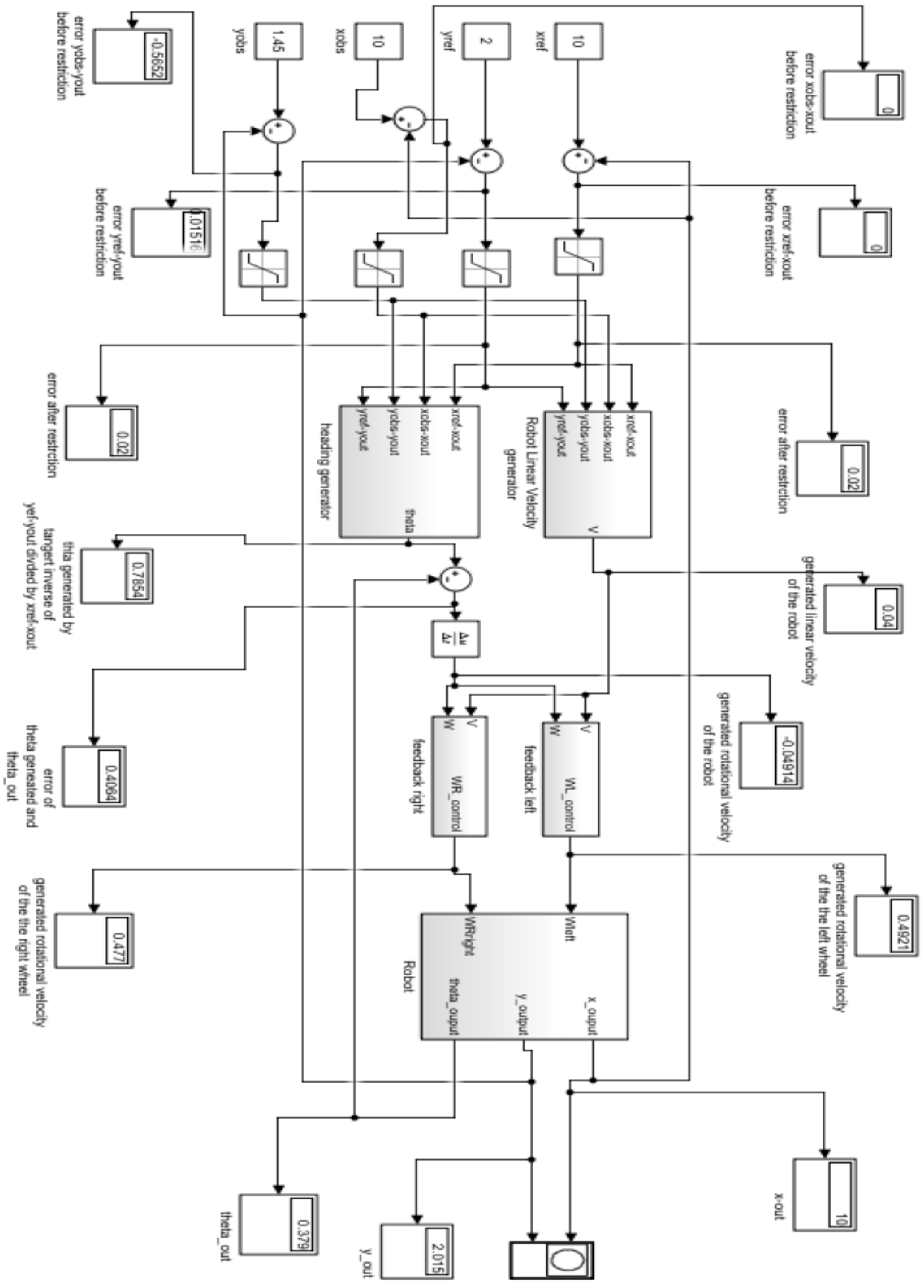

xref
10
yref
2
xobs
10
yobs
1.45
error xobs-xout before restriction
0
error xref-xout before restriction
0
error yobs-yout before restriction
-0.5652
error yref-yout before restriction
0.01516
error after restriction
0.02
error after restriction
0.02
xref-xout
xobs-xout
yobs-yout
yref-yout
v
Robot Linear Velocity generator
xref-xout
xobs-xout
yobs-yout
yref-yout
theta
heading generator
generated linear velocity of the robot
0.04
thta generated by tangent inverse of yef-yout divided by xref-xout
0.7854
error of theta generated and theta_out
0.4064
generated rotational velocity of the robot
-0.04914
V
W
WL_control
feedback left
V
W
WR_control
feedback right
generated rotational velocity of the the left wheel
0.4921
generated rotational velocity of the the right wheel
0.477
Wleft
WRight
x_ouput
y_output
theta_ouput
Robot
x-out
10
y_out
2.015
theta_out
0.379

Figure 4.2: Kinematic model of the six-wheeled robot

4.6 Simulation Results and Discussion

The input to the model was the position of the goal. It showed the response on the robot's position when a rectangular-shaped obstacle is inserted into the kinematic model as input. The result of the obstacle avoidance is shown in Figure 4.3. Therefore, a smooth trajectory was achieved for the robot from the kinematic modeling to avoid obstacles, and the obstacle avoidance capability of the robot was observed in the presence of a rectangular shaped obstacle.

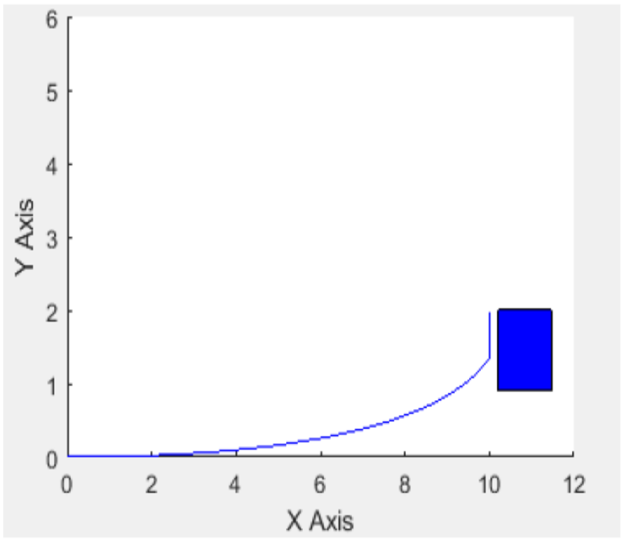


Figure 4.3: Simulation results of obstacle avoidance by the robot

4.7 Potential Utilization of the Kinematic Modeling in Energy Optimization of the Robot

Energy Modeling in mobile robots is the process of developing an equation that combines all the energy loss components in a robot system. Energy losses in a robot occur due to DC motors, gearheads of motors, kinetic energy, the friction of the wheels, and electronic components (Wahab et al., 2015). After measuring all the individual energy loss components of a robot system, those can be summed together to get the complete energy model. The energy equation to develop energy modeling is given by the following equation (Wahab et al., 2015).

$$E_{tot} = E_{armature} + E_{gearhead} + E_{KE} + E_{friction} + E_{elect} \tag{4.23}$$

Here, E_{tot} is the overall energy consumption in the robot system, $E_{armature}$ is the energy loss due to DC motor, $E_{gearhead}$ is the energy loss due to gearheads of the motors, E_{KE} is the kinetic energy loss, $E_{friction}$ is energy loss due to friction of the wheel, and E_{elect} is the energy loss due to onboard electronics.

In the process of developing kinematic modeling of the six-wheeled robot, we calculated the energy losses due to DC motors in terms of loss calculation of armature resistance, armature inductance, and armature capacitance. The kinetic energy losses were also calculated as we measured the linear and angular velocity of the wheels of the robot. The energy model or equation we developed for the energy modeling of the six-wheeled robot is the sum of DC motor energy loss and kinetic energy loss, while ignoring the energy loss due to the gearhead of the motor, energy loss due to friction, and energy loss due to robot onboard electronics. The energy model was used in the development of the kinematic model for the six-wheeled robot. The energy equation we developed for the six-wheeled robot can be utilized as a cost function to optimize the consumption in the robot. The robot will follow the optimal velocity profile brought forth by the kinematic model of the six-wheeled robot while moving from an initial point to a final location. This will result in the improved energy efficiency of the robot.

The kinetic energy losses are presented in Fig 4.4. It can be seen that we considered the kinetic energy losses for the whole operation time of the robot. The reason for that is the robot is functioning in an open-loop, and we don't have a feedback control in place to compensate for the sudden variation in speed, which could be caused by the unevenness of the surface and slipping. The open-loop was chosen to capture the natural response of the robot system.

From Figure 4.4, it is evident that when the robot was accelerating between 0 to 4.2 seconds, the kinetic energy of the robot system was increasing. When the robot reached a somewhat steady velocity, there was no further increase in the system's energy. The dips shown in uniform speed phase between 4.2 and 8.2 sec in the figure are due to the sudden speed changes caused by the minute deviations of the robot from the straight path.

The amount of kinetic energy gain is around 1.2J. This number makes a lot of sense because the test robot is going with a fairly slow linear speed of $0.58ms^{-1}$. We considered only a straight path for the robot because it is the dominant travel path. Even when there is a turn involved, the time for which the robot will be rotating to make the turn is much smaller than the time it will spend in a straight line. So, the translational kinetic energy is preeminent compared to the rotational component, and it is very safe to just select a straight path in order to make the measurement easy.

4.8 Conclusions

This chapter of the thesis presents the kinematic modeling of the six-wheeled differential drive mobile robot on the basis of the potential field method to avoid obstacles. The position of the robot in the simulation was tested from an initial point to a final point in the presence of a rectangular shaped obstacle, and obstacle avoidance by the robot was observed. The entire kinematic modeling and testing were conducted using MATLAB Simulation.

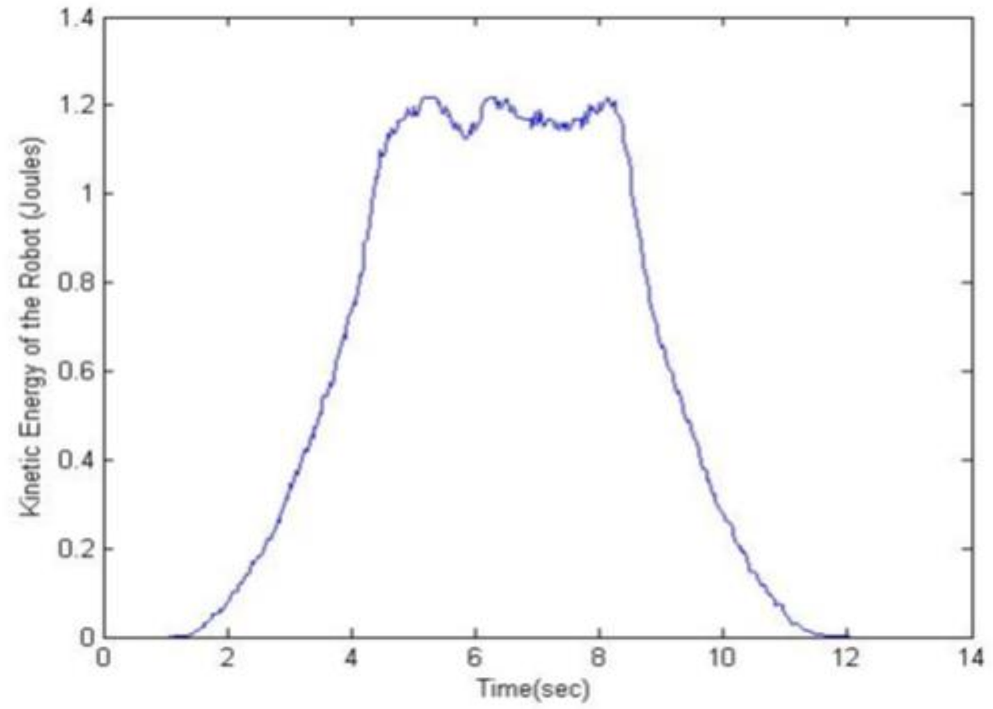


Figure 4.4: Kinetic energy losses of the Robot (Wahab et al., 2015)

CHAPTER 5

IMPLEMENTATION OF ROS AND SLAM ALGORITHM ON TURTLEBOT 2

5.1 Introduction

Human-robot interaction has become one of the 'buzzwords' in the ever-widening field of robotics (Mishra & Javed, 2018). There are many indoor areas like hospitals, homes, warehouses, hotels, etc. where humans can be substituted by a minimal error robotic platform to do certain tasks (Mishra & Javed, 2018). The use of the autonomous wheeled mobile robotic platform for this purpose is also becoming ubiquitous. But, the robotic platforms, which are used to replace human activities, must have a proper navigation system and path planning algorithm. Proper navigation in an unknown indoor environment by an autonomous mobile robot is only possible when the robot is capable of localizing itself in the environment and creating a map of the environment simultaneously. Hence, the Simultaneous Localization and Mapping (SLAM) algorithm has come into the picture with a solution. By using SLAM techniques, a robotic platform localizes itself in the environment while creating a map of the same environment. Therefore, the combination of a SLAM algorithm and a low-cost Microsoft XBOX 360 Kinect depth sensor offers an effective solution for proper navigation (Mishra & Javed, 2018). Hence, we opt to implement the SLAM algorithm on our six-wheeled robot. But before doing that, we tested the SLAM algorithm on a popular robotic platform called the TurtleBot 2, because this is a smaller and lighter platform that is easier to move and test indoors. This chapter discusses our implementation of the SLAM algorithm on the TurtleBot 2 robotic platform. Since the SLAM algorithm is built on the Robot Operating System (ROS), the concept of ROS is also described in the chapter.

5.2 Building Blocks for the TurtleBot 2 Navigation

There are five major building blocks used to prepare the TurtleBot 2 functioning for indoor navigation and mapping. Subsection 5.2.1 describes the Robot Operating System (ROS) briefly while subsections 5.2.2 and 5.2.3 give some information about TurtleBot 2 and client PC, respectively. Finally, subsections 5.2.4 and 5.2.5 briefly sketch the workstation PC and networking technique, respectively.

5.2.1 Robot Operating System (ROS)

ROS is a Linux distribution based open-source implementation for a public operated robotics system (Hamzeh & Elnagar, 2015). ROS operates by creating an ecosystem among a ROS node, a ROS master, and a ROS topic. ROS nodes are executable programs which are registered to a ROS master (Hamzeh & Elnagar, 2015). Nodes become capable of knowing each other's existence with the help of a ROS master (Mishra & Javed, 2018). Nodes need to communicate among themselves during runtime. They don't communicate directly; instead, they maintain the communication indirectly via publishing or subscribing messages to ROS topics (Mishra & Javed, 2018). Messages like odometry information, control commands, video streams or images, or any kind of information are passed among two or more nodes (Hamzeh & Elnagar, 2015). The communication process among nodes through ROS topics is kind of a decoupled system in which a node in need of data has to subscribe to the relevant topic and a node that produces data publish messages to the topic (Mishra & Javed, 2018). This decoupled system aids the robot in performing different functions without hampering each other (Mishra & Javed, 2018). Therefore, in case, one of the functions of the robot crashes, the entire robot doesn't stop functioning (Mishra & Javed, 2018).

There are many advantages to using ROS. Since ROS is an open-source implementation, the plugin source codes and libraries are easily accessible and available. Furthermore, ROS codes are reusable, which allows the researcher to proceed more from the already existing codes (Mishra & Javed, 2018). ROS provides hardware abstractions and low-level device controls to control the connected devices and also leverages algorithms implemented in projects to provide a navigation system (Hamzeh & Elnagar, 2015). Instead of typing command lines to execute each node, ROS makes the whole process easier by introducing 'packaged' functionality (Hamzeh & Elnagar, 2015). The ROS packaged functionality reads an XML description of a graph and represents the graph on the cluster, optionally on specific hosts (Hamzeh & Elnagar, 2015).

The most important feature of ROS used in the mapping of a robot is *RViz*. *RViz* is a visualization tool which instantiates a graphical user interface to view images, point clouds, geometric primitives, sensor data, robot poses and trajectories, robot models, environmental maps, etc. (Hamzeh & Elnagar, 2015; Mishra & Javed, 2018).

The first ROS distribution version named 'ROS Box Turtle' was released in March 2010 (Hamzeh & Elnagar, 2015). Since then, many popular versions of ROS distributions like ROS Indigo, ROS Kinetic, ROS Melodic, etc. have been released. Since a TurtleBot 2 is used for our project and most of the libraries of ROS Indigo support and sync with the TurtleBot 2, using ROS Indigo for our project was an obvious choice.

5.2.2 TurtleBot 2

TurtleBot was first released in the market by Willow Garage back in 2010 (Aagela, Al-Nesf, & Holmes, 2017). It is an improved version of TurtleBot (Bergeon, Křivánek, & Motsch, 2019; Hamzeh & Elnagar, 2015; Hsu & Huang, 2016). The wheels, battery, integrated gyroscope, wheel drop sensor, cliff sensor, and wheel encoders are housed in the Kobuki base (Hamzeh & Elnagar, 2015; Lee, Salam, Ibrahim, Rahni, & Mohamed, 2015). The Microsoft XBOX 360 Kinect sensor is mounted on the top of the Kobuki base, which provides the range of data and depth information (Hamzeh & Elnagar, 2015; Kondo, Ohba, & Kashiwase, 2016; Lee et al., 2015). The normal operating range for the Kinect sensor and the maximum speed of TurtleBot 2 are 0.5 to 3.5 meter and 50 cm/s, respectively (Hamzeh & Elnagar, 2015; Lee et al., 2015). Both the Kobuki base and Kinect camera are powered up by a lithium-ion battery housed in the Kobuki base (Hamzeh & Elnagar, 2015). TurtleBot 2 is shown in Figure 5.1.

5.2.3 Client PC

Client PC is connected to the robot and Kinect sensor. Linux distribution Ubuntu and ROS are installed on the client PC. The client PC receives depth information, video, and image stream from the Kinect sensor. It also collects odometry data from the wheel encoders of the TurtleBot (Hamzeh & Elnagar, 2015).

The depth_to_laserscan, map server, and the *gmapping* application are run on the client PC. The depth_to_laserscan application converts the depth information into laser scan data, which the G-Mapping application processes to build a grid map. Later, the map server application collects the built grid map from the G-Mapping application and dispatches a copy of the map to the workstation PC (Hamzeh & Elnagar, 2015).

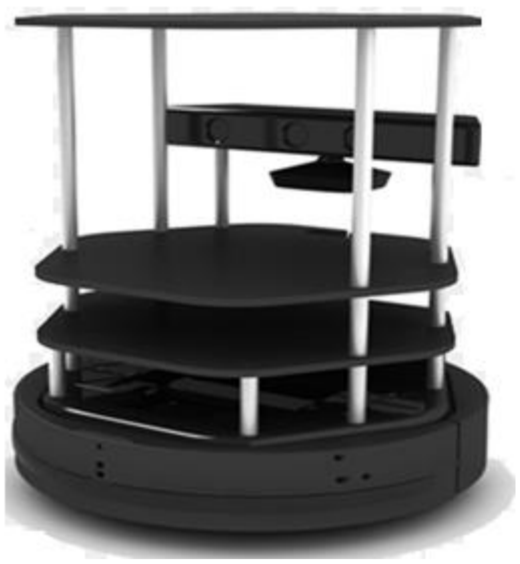
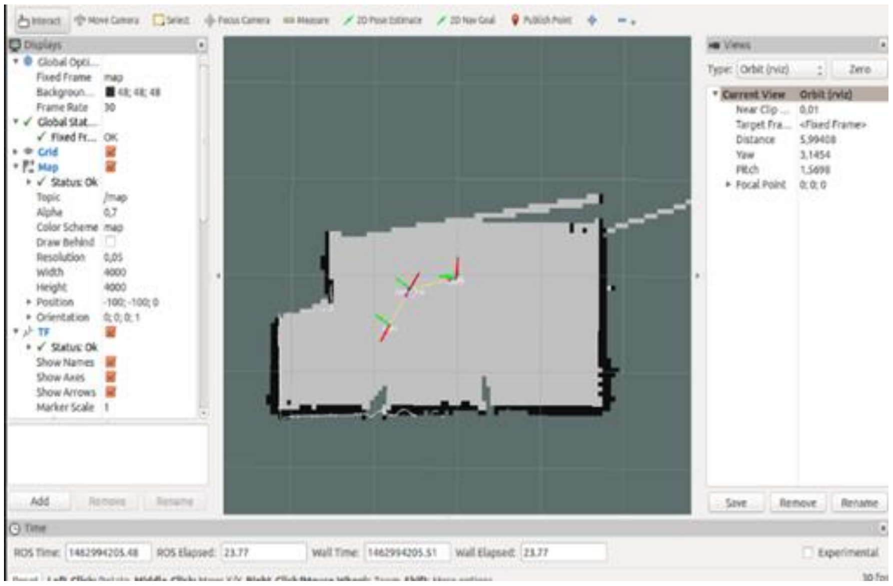


Figure 5.1: TurtleBot 2 (Source: TurtleBot)

5.2.4 Workstation PC

Workstation PC is a remote PC used to control the Client PC, to view the map and video stream, and to save the grid map. Ubuntu 14.04 and ROS Indigo is installed on the workstation PC (Hamzeh & Elnagar, 2015). Applications like ROS Core, teleoperation, *RViz*, and map saver are run on the workstation PC.

The ROS Master node operates using the ROS Core application while the robot's movement is controlled from the workstation PC. The user can visualize the current configuration of the robot on a virtual model utilizing the *RViz* application (Hamzeh & Elnagar, 2015). The sensor values are displayed on the *RViz* interface (Hamzeh & Elnagar, 2015). Fig. 5.2 illustrates an *RViz* interface. Finally, the map saver application saves the latest grid map on the workstation PC (Hamzeh & Elnagar, 2015).



```
Global Opti...
  Fixed Frame      map
  Backgroun...     48; 48; 48
  Frame Rate       30
✓ Global Stat...
  ✓ Fixed Fr...    OK
Grid               ✓
Map                ✓
  ✓ Status: Ok
  Topic            /map
  Alpha            0,7
  Color Scheme     map
  Draw Behind      ☐
  Resolution       0,05
  Width            4000
  Height           4000
  Position         -100; -100; 0
  Orientation      0; 0; 0; 1
TF                 ✓
  ✓ Status: Ok
  Show Names       ✓
  Show Axes        ✓
  Show Arrows      ✓
  Marker Scale     1
```

```
Type: Orbit (rviz)
Current View       Orbit (rviz)
  Near Clip ...    0,01
  Target Fra...    <Fixed Frame>
  Distance         5,99408
  Yaw              3,1454
  Pitch            1,5698
  Focal Point      0; 0; 0
```

```
ROS Time: 1462994205.48   ROS Elapsed: 23.77   Wall Time: 1462994205.51   Wall Elapsed: 23.77   ☐ Experimental
```

Figure 5.2: Sample of *RViz* interface

5.2.5 Network Technique

Both the client PC and the workstation PC are connected under the same WiFi. Any TCP/IP address works for ROS till the nodes have access to the ROS master (Hamzeh & Elnagar, 2015). Poor bandwidth may cause the robot to be disconnected or operating without having any video stream (Hamzeh & Elnagar, 2015). Poor latency causes the failure of teleoperation, which in turn, features an uncontrolled robot hitting the obstacle (Hamzeh & Elnagar, 2015).

5.3 Simultaneous Localization and Mapping (SLAM)

In order to navigate properly in an unknown environment, an autonomous robot must localize itself within the environment and create a map of the environment simultaneously (Mishra & Javed, 2018). Simultaneous Localization and Mapping (SLAM) algorithm paves the way to do both simultaneously (Hamzeh & Elnagar, 2015; Hsu & Huang, 2016; Syaqur et al., 2018).

Map generation is the first and most important step before the robot can navigate itself in a precise indoor environment (Syaqur et al., 2018). Many kinds of SLAM algorithms like G-

Mapping, Hector-SLAM, Core-SLAM, etc. are deployed to create maps (Syaqur et al., 2018). Depending upon the type of sensor used and the characteristics of the environment itself, each of these algorithms shows some advantages and drawbacks (Syaqur et al., 2018). G-Mapping has been selected in our project as it provides the best time and size-independent solution for TurtleBot 2 (Mishra & Javed, 2018).

A map is required for localization, while a pose estimation is necessary for mapping (Wu, Abrahantes, & Edgington, 2016). An approximation is used to process the SLAM algorithm (Hamzeh & Elnagar, 2015). ROS comes with several packages for localization and automatic path optimization for G-Mapping of the SLAM algorithm (Hamzeh & Elnagar, 2015).

In the G-Mapping process, the Kinect sensor collects depth information and translates it in real-time to laser format data (Hamzeh & Elnagar, 2015). This initial data starts generating an initial local grid map and initial pose of the robot in each time frame for a limited region (Mishra & Javed, 2018). The latest laser scan data is used to detect obstacles in the path of the robot. The G-Mapping process generates a grid map in the Field of View (FOV) of the Kinect (Hamzeh & Elnagar, 2015).

Along with the odometry information received from the Kobuki base, the generated grid map is saved (Hamzeh & Elnagar, 2015). The estimation of the pose of the robot is based upon the above-received data (Hamzeh & Elnagar, 2015). The pose of the robot is evaluated as a particle within the previously built local grid map. Afterward, the local grid map itself is added to the global map with the help of a scan matching algorithm (Hamzeh & Elnagar, 2015).

5.4 Experimental Setup

The laptop, placed on the top of TurtleBot 2, was used as both client and workstation PC for the mapping in our project as shown in Figure 5.3. The laptop with Ubuntu 14.04 and ROS Indigo installed was connected to both the Microsoft XBOX 360 Kinect sensor and the TurtleBot 2. The power circuit for both the Kobuki base and the Kinect sensor was housed in the Kobuki base itself. The purpose of the experimental setup was to move the TurtleBot 2 from an initial position to a final position inside the robotics lab, get the map of the indoor environment, and make the TurtleBot 2 capable of proper navigation.

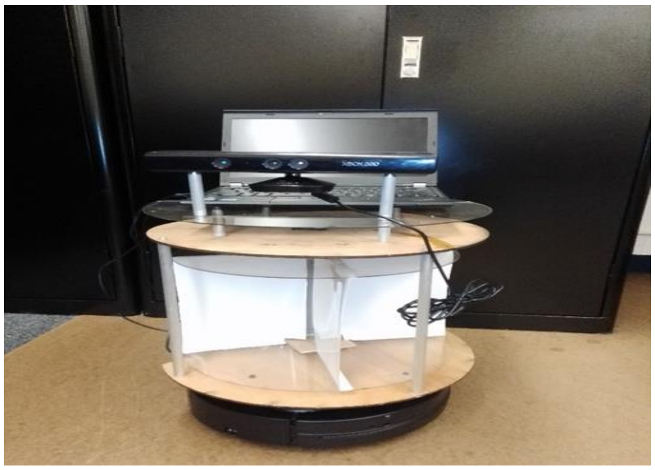


Figure 5.3: Experimental setup for TurtleBot 2

5.5 Methodology

We utilized the *gmapping* technique of the SLAM algorithm to bring forth the map of the indoor environment. The indoor movement of the TurtleBot 2 navigation, as well as the mapping of the environment, were set to be displayed on the laptop through the *RViz* interface. TurtleBot 2 was controlled and operated through the keyboard manually by initiating a teleoperation application.

Figure 5.4 shows the flowchart for the mapping algorithm we implemented on the TurtleBot 2. At first, we gave the command on the laptop to move the Kobuki base. Then the movement of the Kobuki base was controlled by launching the teleoperating application on the laptop. Using the teleoperation application, we could maneuver the TurtleBot 2 by using the keyboard. Finally, we tried to launch the gmapping technique of the SLAM algorithm. According to our algorithm, the successful launching of *gmapping* would generate a map of the indoor environment, and the map could be saved in the map server application.

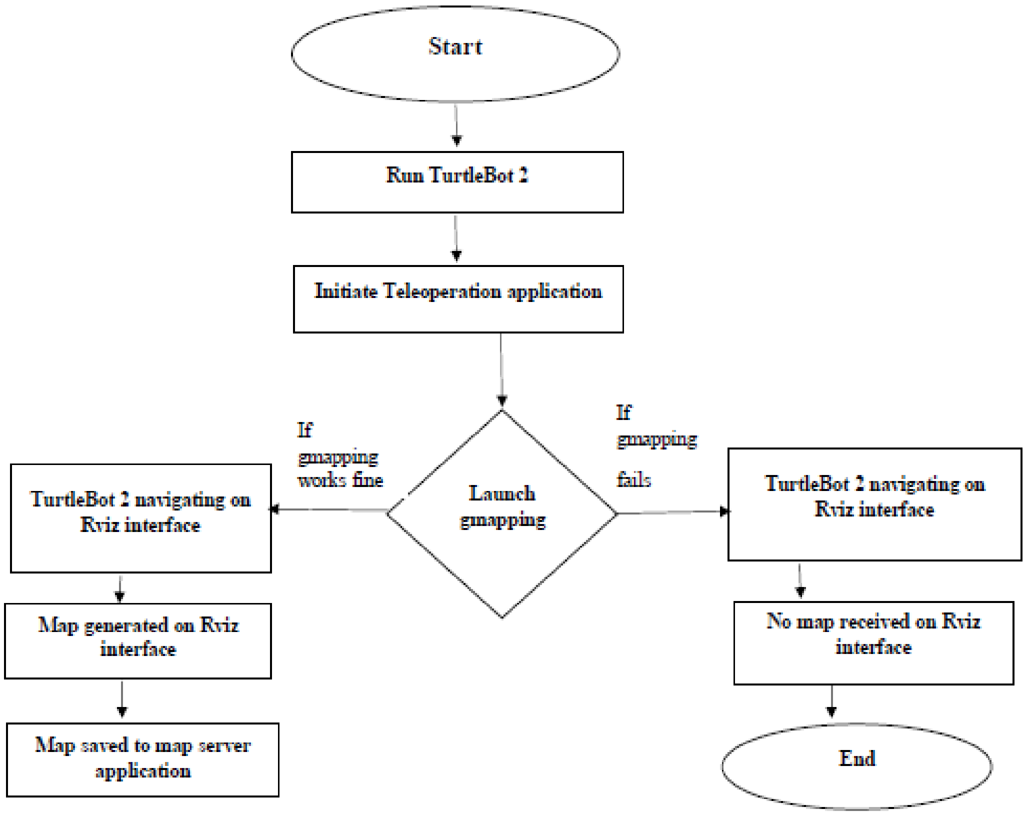

Start
Run TurtleBot 2
Initiate Teleoperation application
Launch gmapping
If gmapping works fine
If gmapping fails
TurtleBot 2 navigating on Rviz interface
Map generated on Rviz interface
Map saved to map server application
TurtleBot 2 navigating on Rviz interface
No map received on Rviz interface
End

Figure 5.4: Flowchart for our mapping algorithm

A successful SLAM localization which follows our mapping algorithm will collect the odometry data and laser data from the Internal Measurement Unit (IMU) and laser sensor of TurtleBot 2, respectively, and localize itself in the environment. Each SLAM self-localization record is updated in the map server application. The flowchart for the SLAM self-localization is shown in Figure 5.5.

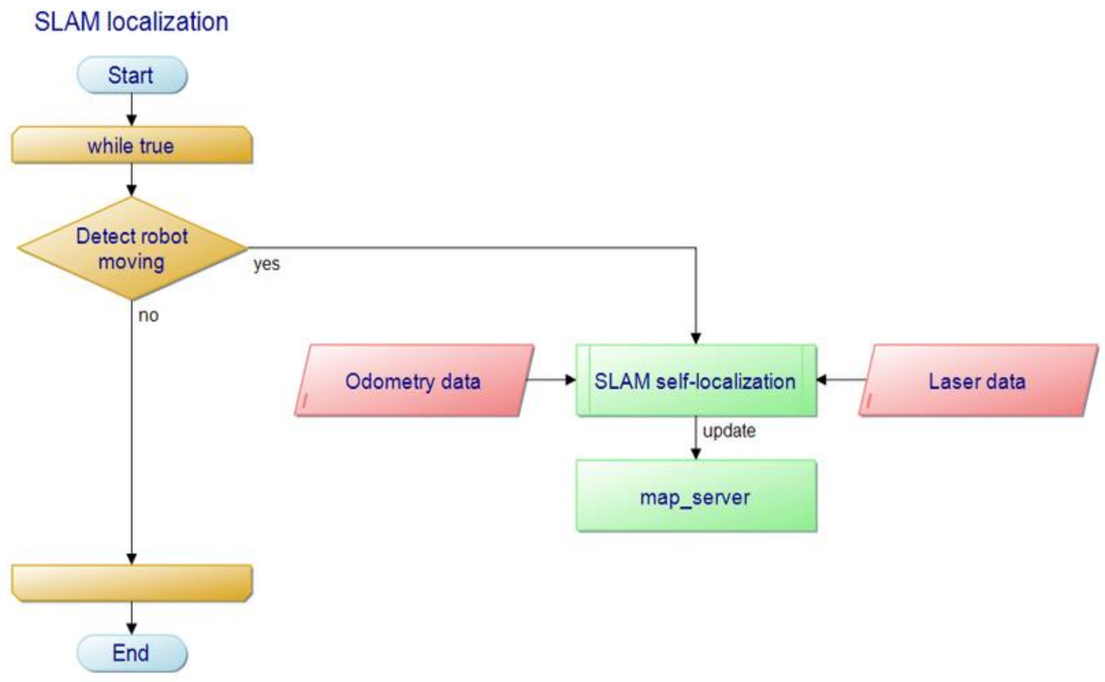
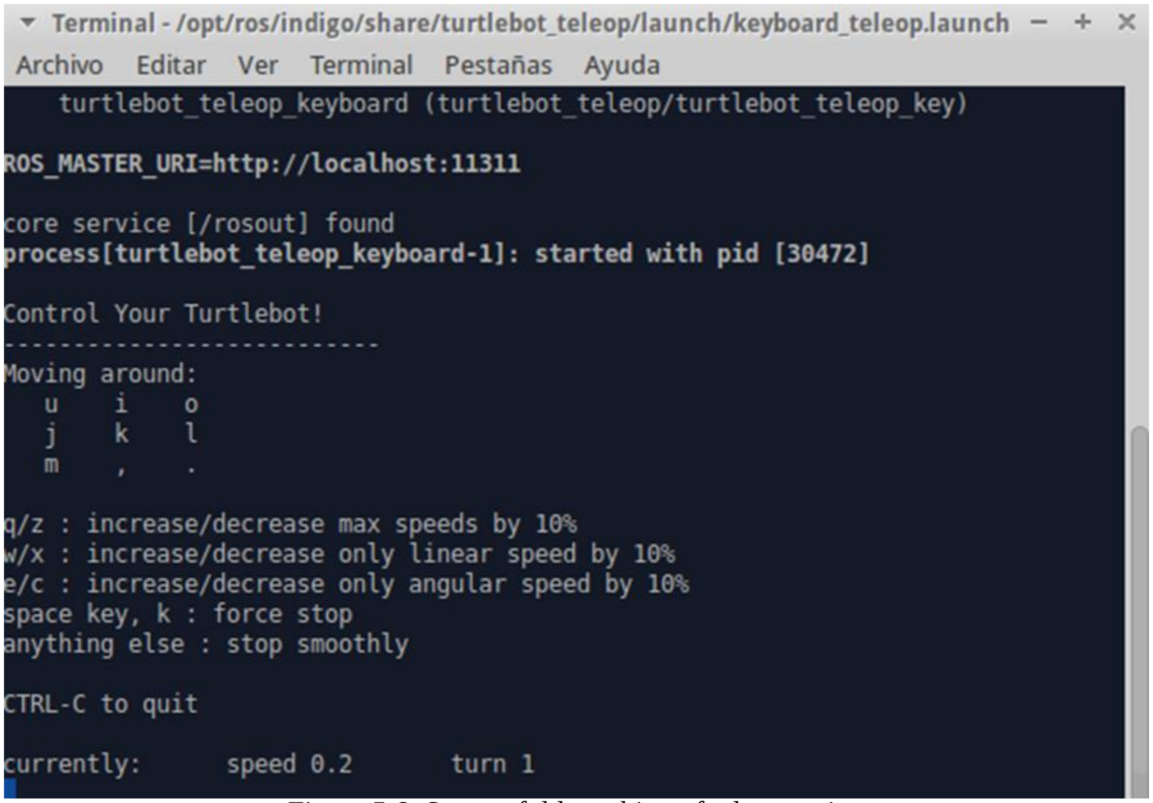


Figure 5.5: Flowchart for SLAM self-localization

5.6 Experimental Results

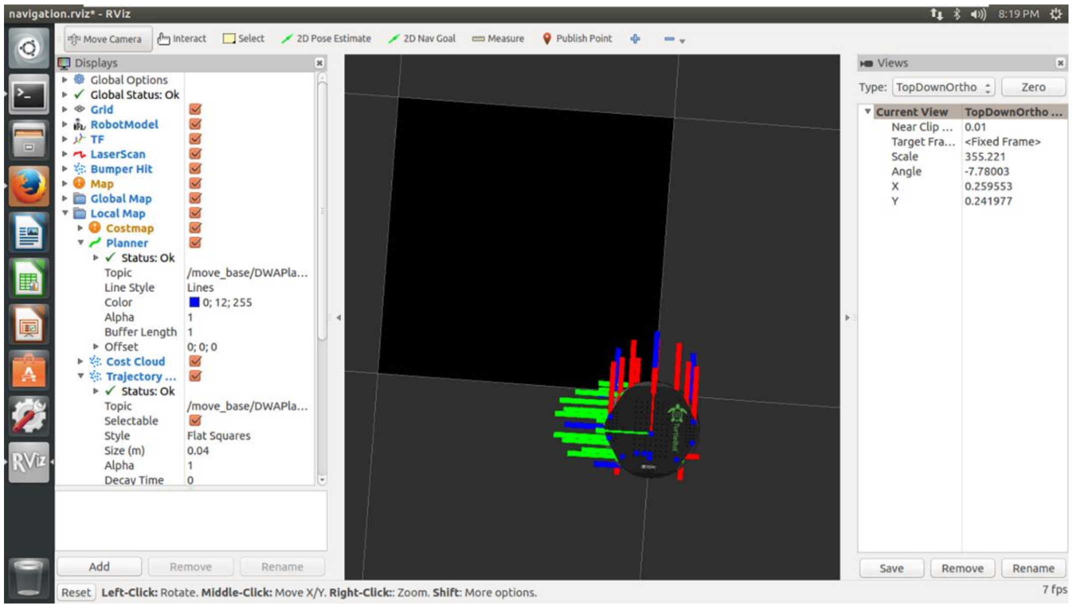
As we maneuvered the TurtleBot 2 in real life by initiating the commands to launch the Kobuki base and teleoperation application, it also moved in the *RViz* environment simultaneously. Figure 5.6 illustrates the window for our successfully launched of teleoperation application. Meanwhile, we began the *gmapping* technique and *Rviz* applications in the laptop by prompting their corresponding command lines. Figure 5.7 shows the robot in the RViz environment. Figure

5.8 depicts the robot's movement in real life while the maneuver of the robot in RViz environment is illustrated in Figure 5.9.


Terminal - /opt/ros/indigo/share/turtlebot_teleop/launch/keyboard_teleop.launch
Archivo
Editar
Ver
Terminal
Pestañas
Ayuda

```
    turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)

ROS_MASTER_URI=http://localhost:11311

core service [/rosout] found
process[turtlebot_teleop_keyboard-1]: started with pid [30472]

Control Your Turtlebot!
---------------------------
Moving around:
   u    i    o
   j    k    l
   m    ,    .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:      speed 0.2       turn 1
```

Figure 5.6: Successful launching of teleoperation



```
Global Options
✓ Global Status: Ok
Grid                 ✓
RobotModel           ✓
TF                   ✓
LaserScan            ✓
Bumper Hit           ✓
Map                  ✓
Global Map           ✓
Local Map            ✓
  Costmap            ✓
  Planner            ✓
    ✓ Status: Ok
    Topic            /move_base/DWAPla...
    Line Style       Lines
    Color            0; 12; 255
    Alpha            1
    Buffer Length    1
    Offset           0; 0; 0
  Cost Cloud         ✓
  Trajectory ...     ✓
    ✓ Status: Ok
    Topic            /move_base/DWAPla...
    Selectable       ✓
    Style            Flat Squares
    Size (m)         0.04
    Alpha            1
    Decay Time       0
```

```
Type: TopDownOrtho    Zero
Current View     TopDownOrtho ...
  Near Clip ...  0.01
  Target Fra...  <Fixed Frame>
  Scale          355.221
  Angle          -7.78003
  X              0.259553
  Y              0.241977
```

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.

Figure 5.7: TurtleBot 2 in the *RViz* navigation environment

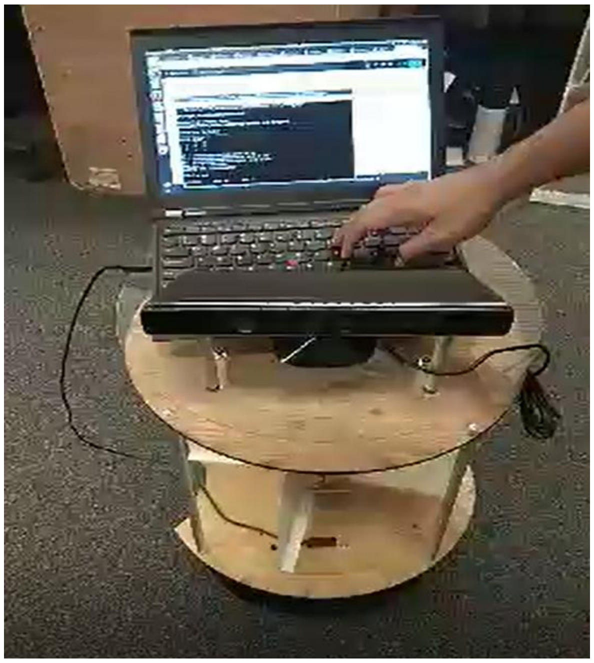
Although the maneuver of the TurtleBot was made possible inside the robotics lab, the map of the environment hadn't been received due to the map error and camera error as shown in Figure 5.10 and Figure 5.11, respectively. The map error and camera error were retrieved from the RViz window and *gmapping* window, respectively.

5.7 Discussions

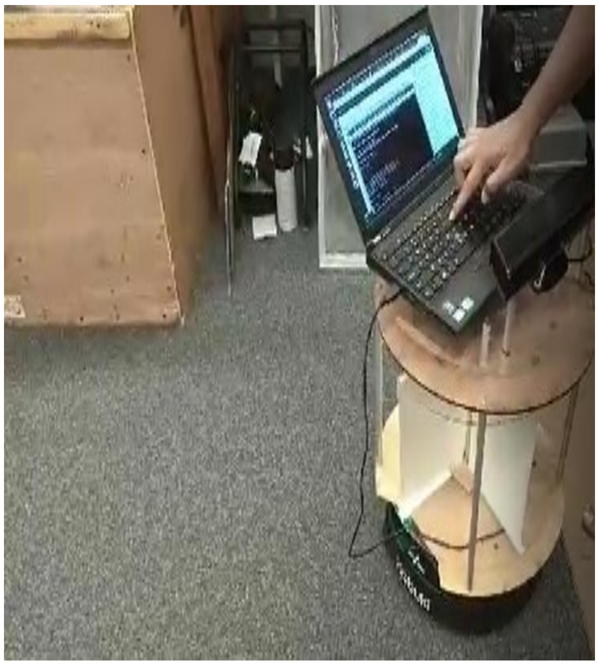
The errors suggest that the *gmapping* process and Kinect camera were trying to get the map, but those failed as the device timed out. We followed several different strategies to resolve the errors. We installed the Kinect driver again and checked the USB connection of the Kinect sensor. But none of these processes could solve the issues. Later, we scrutinized the Kinect image model, nodes and topics of the depth image, and debugging trees. But our investigation failed to generate the map of the indoor environment. Conclusively, we could initiate ROS and SLAM algorithms and successfully implement these to maneuver the TurtleBot 2 through teleoperation but failed to generate the map of the indoor environment of the robotics lab.

5.8 Conclusions

We could successfully implement ROS and SLAM algorithms on TurtleBot 2 to maneuver it but were unable to get a map of the indoor environment, which is essential for proper navigation. Therefore, instead of implementing the ROS and SLAM algorithms on our six-wheeled robot, we simulated the six-wheeled robot in a simulation environment. The description of the simulation is given in the next chapter.

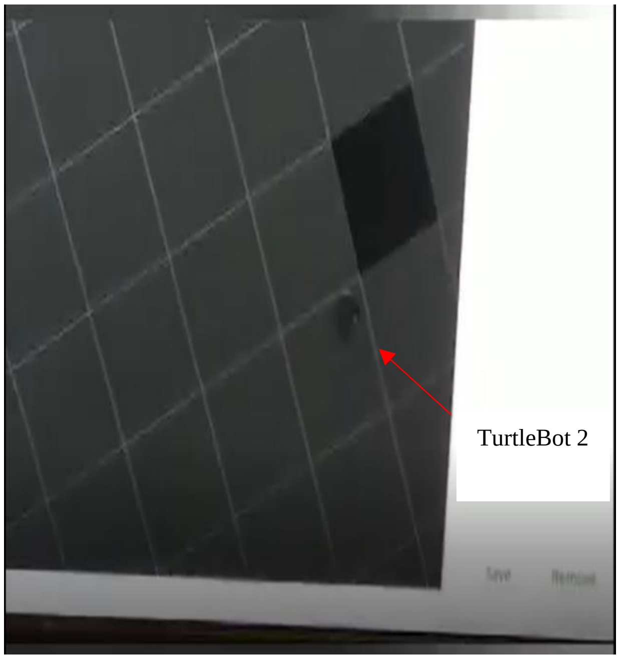


(a)Initial position of the TurtleBot 2 in real-time environment

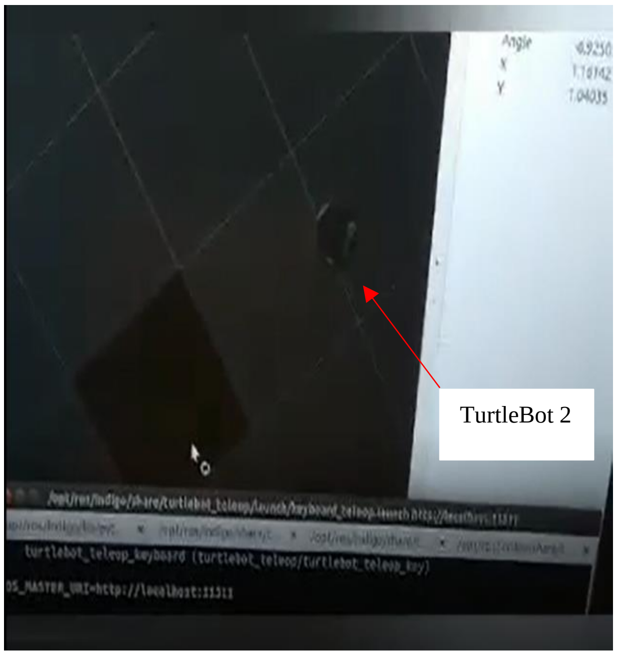


(b)Final Position of the TurtleBot 2 in real-time environment

Figure 5.8: TurtleBot 2 maneuver in real-time environment



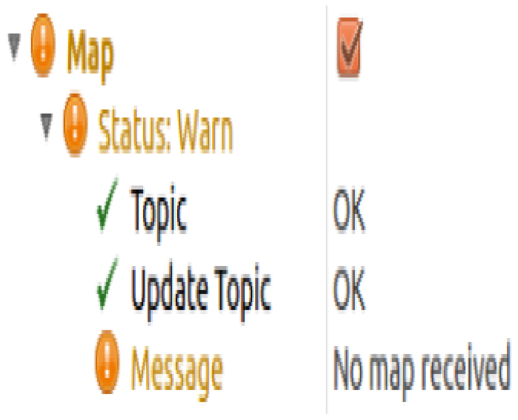
(a)Initial position of the TurtleBot 2 in RViz environment



```
/opt/ros/indigo/share/turtlebot_teleop/launch/keyboard_teleop.launch http://localhost:11311
turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)
OS_MASTER_URI=http://localhost:11311
```

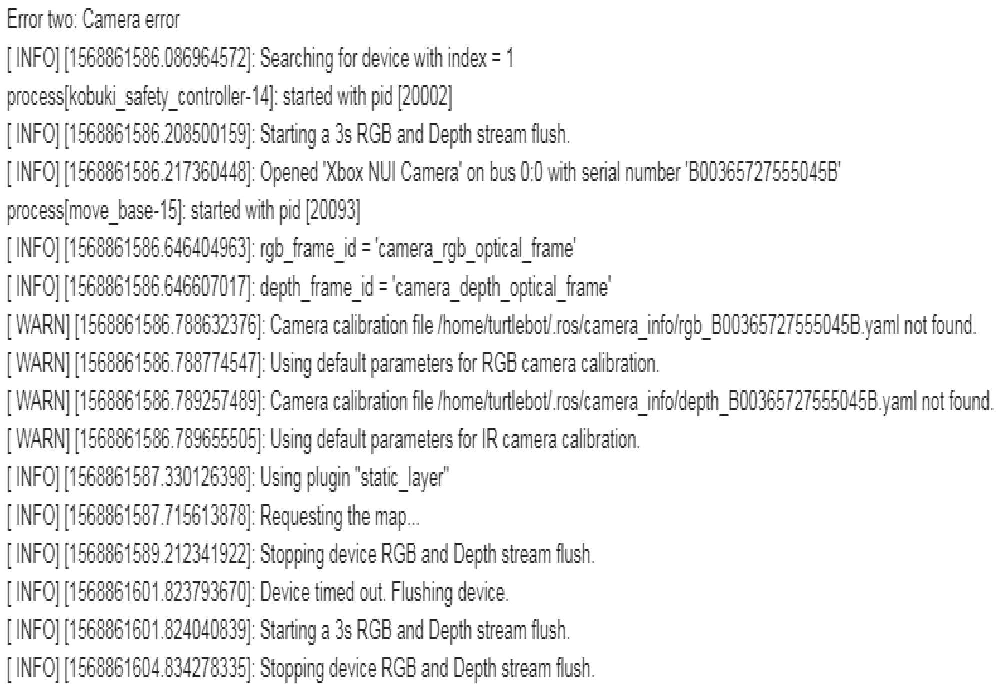
(b)Final Position of the TurtleBot 2 in RViz environment

Figure 5.9: TurtleBot 2 maneuvers in *RViz* environment



```
Map                  ✓
  Status: Warn
    Topic            OK
    Update Topic     OK
    Message          No map received
```

Figure 5.10: Map error



```
Error two: Camera error
[ INFO] [1568861586.086964572]: Searching for device with index = 1
process[kobuki_safety_controller-14]: started with pid [20002]
[ INFO] [1568861586.208500159]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1568861586.217360448]: Opened 'Xbox NUI Camera' on bus 0:0 with serial number 'B00365727555045B'
process[move_base-15]: started with pid [20093]
[ INFO] [1568861586.646404963]: rgb_frame_id = 'camera_rgb_optical_frame'
[ INFO] [1568861586.646607017]: depth_frame_id = 'camera_depth_optical_frame'
[ WARN] [1568861586.788632376]: Camera calibration file /home/turtlebot/.ros/camera_info/rgb_B00365727555045B.yaml not found.
[ WARN] [1568861586.788774547]: Using default parameters for RGB camera calibration.
[ WARN] [1568861586.789257489]: Camera calibration file /home/turtlebot/.ros/camera_info/depth_B00365727555045B.yaml not found.
[ WARN] [1568861586.789655505]: Using default parameters for IR camera calibration.
[ INFO] [1568861587.330126398]: Using plugin "static_layer"
[ INFO] [1568861587.715613878]: Requesting the map...
[ INFO] [1568861589.212341922]: Stopping device RGB and Depth stream flush.
[ INFO] [1568861601.823793670]: Device timed out. Flushing device.
[ INFO] [1568861601.824040839]: Starting a 3s RGB and Depth stream flush.
[ INFO] [1568861604.834278335]: Stopping device RGB and Depth stream flush.
```

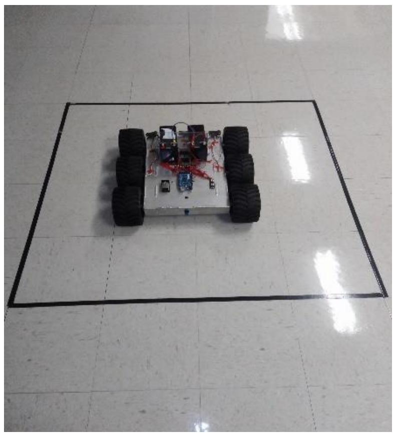
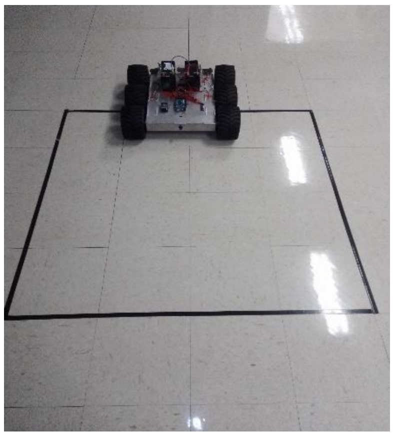
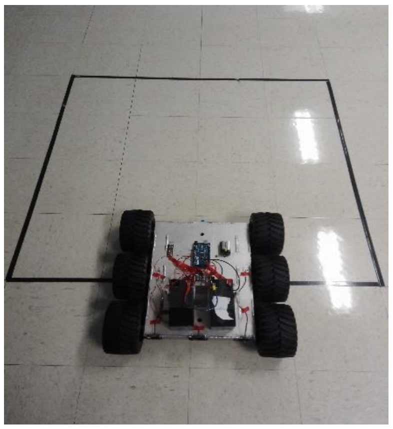
Figure 5.11: Camera error

CHAPTER 6

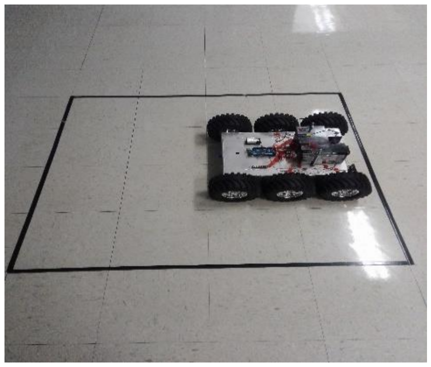
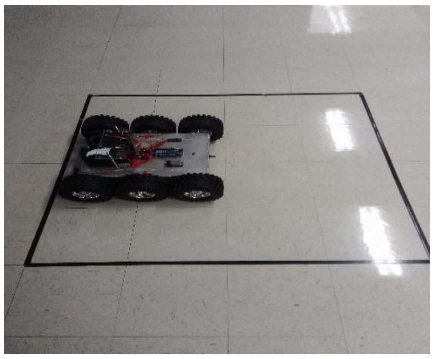
NAVIGATION OF THE SIX-WHEELED ROBOT IN SIMULATION ENVIRONMENT

6.1 Introduction

The mapping, path planning, and exploration in an unknown space are necessary for an autonomous robot to navigate properly (Scott & Yu, 2009). Our plan was to map the indoor environment of the hospital by implementing ROS and SLAM algorithms for our six-wheeled robot. ROS and SLAM algorithms were tested for mapping the indoor environment before implementing these techniques on the six-wheeled robot. A robotic platform named TurtleBot 2 was picked in this testing purpose. Although we maneuvered TurtleBot 2 inside the indoor environment, we couldn't retrieve the map of the surroundings due to some errors which we failed to solve. Hence, implementing ROS and SLAM algorithms on the six-wheeled robot to get a map of the indoor environment became ineffectual. Therefore, we opted to create a simulation model of the six-wheeled robot, develop an obstacle avoidance controller for the, and navigate the modeled six-wheeled in the simulation environment. The proposed simulation environment resembled the indoor environment of the hospital. Webot simulator software was used for the entire simulation work. We also maneuvered the six-wheeled differential drive robot in different directions in real life alongside developing a proper navigation scheme in the simulation environment. Figure 6.1 shows the navigation of the six-wheeled robot in a real-time environment.

(a)Robot initial position (b) Robot forward movement (c) Robot backward movement

(d) Robot right turn (e) Robot left turn

Figure 6.1: Navigation of the six-wheeled robot in a real-time environment

6.2 Project Development Process in Webot Simulator

Webot simulator, a professional mobile robotics simulation package distributed by Cyberbotics, is very popular for the purpose of research and education in universities (Scott & Yu, 2009). It is a framework used to simulate, model, and program robots (Nandi, 2013). It provides 3D visualization of simulation environments and utilizes the Open Dynamics Engine (ODE) for rigid body dynamics and collision detection (Scott & Yu, 2009). The development process of a robotic project that uses the software Webot normally runs in three stages (Obdržálek, 2016).

The first stage is the phase of modeling, which consists of the design of the physical robot's body, including their sensors and actuators and the robot's physical environment. The model of mobile robots can be constructed from pre-built building blocks. Webot provides flexibility to change the properties like color, shape, sensors, actuators, etc. of the pre-built building blocks. The environment of the robots is created in the same way, by filling the space objects, such as walls, obstacles, etc. Values of parameters such as the distribution of mass, friction, reflection parameters, etc. can be modified in the process of creating an object. As soon as virtual robots and virtual environments are created, it is possible to go to the second stage (Obdržálek, 2016).

The second stage is the phase of programming. We can program the behavior of each robot with the help of programming languages such as C, C++, Java, Python, or MATLAB (Obdržálek, 2016).

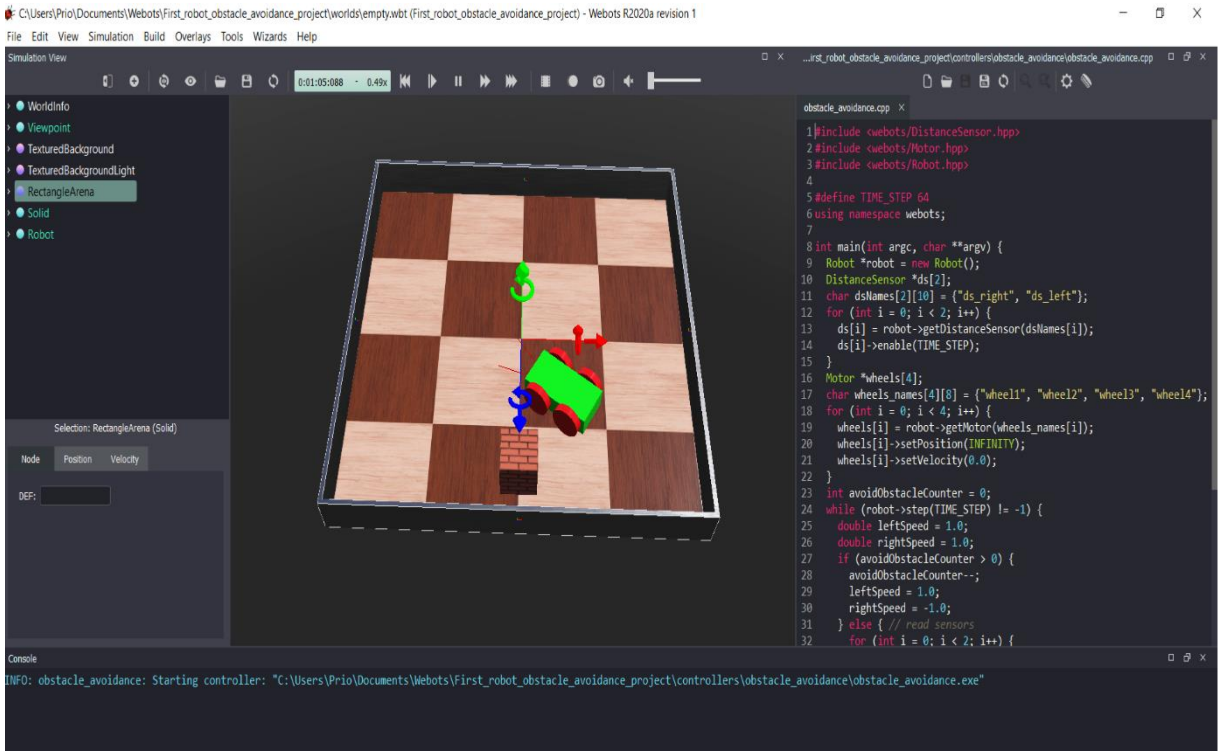
The third stage is the simulation phase. It illustrates whether the created program behaves according to our ideas in the simulation environment (Obdržálek, 2016).

6.3 Simulation Environment in Webot

Figure 6.2 shows an example of the Webot simulation environment. It is composed of several parts. The virtual 3D scene allows us to track the robot into the virtual world and allows us to interact with them. 3D description of the properties of the robots and their environment is used to create this virtual world. Virtual Reality Modeling Language (VRML) is used as this description. Though the appearance of the world and the robots is described by the VRML tree, it does not contain code that ensures the behavior of robots. Behavior is implemented through a controller). The controller is a computer program compiled by C, C++, Java, Python or MATLAB programming language. It is used to load the sensory data and to issue instructions to the robot actuators. The last two components constituting the Webot Graphical User Interface (GUI) are Text Editor and Console. The Text Editor is designed to edit the source code of the controller. The console is used to display the compilation results and the output of the robot's controller (Obdržálek, 2016).

6.4 Simulation Setup for our Six-Wheeled Robot

We created a simulation setup for the six-wheeled robot in which the robot navigates and


C:\Users\Prio\Documents\Webots\First_robot_obstacle_avoidance_project\worlds\empty.wbt (First_robot_obstacle_avoidance_project) - Webots R2020a revision 1
File Edit View Simulation Build Overlays Tools Wizards Help
Simulation View
0:01:05:088 - 0.49x
WorldInfo
Viewpoint
TexturedBackground
TexturedBackgroundLight
RectangleArena
Solid
Robot
Selection: RectangleArena (Solid)
Node Position Velocity
DEF:
...irst_robot_obstacle_avoidance_project\controllers\obstacle_avoidance\obstacle_avoidance.cpp
obstacle_avoidance.cpp
Console

```
#include <webots/DistanceSensor.hpp>
#include <webots/Motor.hpp>
#include <webots/Robot.hpp>

#define TIME_STEP 64
using namespace webots;

int main(int argc, char **argv) {
  Robot *robot = new Robot();
  DistanceSensor *ds[2];
  char dsNames[2][10] = {"ds_right", "ds_left"};
  for (int i = 0; i < 2; i++) {
    ds[i] = robot->getDistanceSensor(dsNames[i]);
    ds[i]->enable(TIME_STEP);
  }
  Motor *wheels[4];
  char wheels_names[4][8] = {"wheel1", "wheel2", "wheel3", "wheel4"};
  for (int i = 0; i < 4; i++) {
    wheels[i] = robot->getMotor(wheels_names[i]);
    wheels[i]->setPosition(INFINITY);
    wheels[i]->setVelocity(0.0);
  }
  int avoidObstacleCounter = 0;
  while (robot->step(TIME_STEP) != -1) {
    double leftSpeed = 1.0;
    double rightSpeed = 1.0;
    if (avoidObstacleCounter > 0) {
      avoidObstacleCounter--;
      leftSpeed = 1.0;
      rightSpeed = -1.0;
    } else { // read sensors
      for (int i = 0; i < 2; i++) {
```

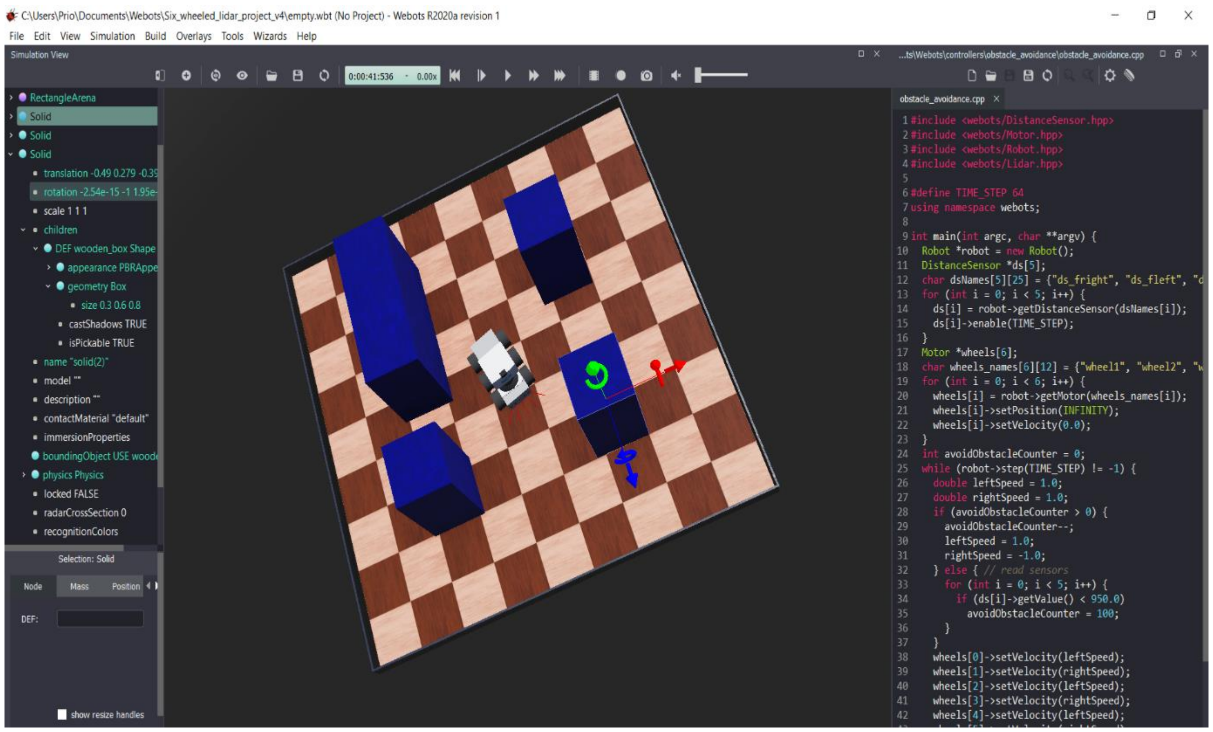
```
INFO: obstacle_avoidance: Starting controller: "C:\Users\Prio\Documents\Webots\First_robot_obstacle_avoidance_project\controllers\obstacle_avoidance\obstacle_avoidance.exe"
```

Figure 6.2: An example of Webot simulation environment

avoids obstacles. We needed to model both the environment and the robot in the setup.

The environmental setup was started by creating a new world. A basic checkered floor appeared with basic world settings. The checkered floor was surrounded by the walls. We created the walls by adjusting the 'Geometry' and 'Appearance' nodes. We created four obstacles inside the environment. Each obstacle resembled the small rooms or pillars of the hospital. To create each obstacle, a 'Solid' node was added, and we adjusted the 'children' node inside it. ‘Bounding Object’ node and ‘Physics’ node were used to bind each object (Jeffril & Sariff, 2013).

A six-wheeled mobile robot model with five distance sensors mounted in front of it was created. A 180-degree Lidar sensor was also put on the top of the robot. These sensors are used for data collection and to detect obstacles within the environment. The robot controller is programmed using C++. Figure 6.3 illustrates the complete simulation setup for the navigation of the robot.


C:\Users\Prio\Documents\Webots\Six_wheeled_lidar_project_v4\empty.wbt (No Project) - Webots R2020a revision 1
File Edit View Simulation Build Overlays Tools Wizards Help
Simulation View
0:00:41:536 0.00x
...ts\Webots\controllers\obstacle_avoidance\obstacle_avoidance.cpp
obstacle_avoidance.cpp
Selection: Solid
Node Mass Position
DEF:
show resize handles

```
RectangleArena
Solid
Solid
Solid
  translation -0.49 0.279 -0.39
  rotation -2.54e-15 -1 1.95e-
  scale 1 1 1
  children
    DEF wooden_box Shape
      appearance PBRAppe
      geometry Box
        size 0.3 0.6 0.8
      castShadows TRUE
      isPickable TRUE
  name "solid(2)"
  model ""
  description ""
  contactMaterial "default"
  immersionProperties
  boundingObject USE woode
  physics Physics
  locked FALSE
  radarCrossSection 0
  recognitionColors
```

```cpp
#include <webots/DistanceSensor.hpp>
#include <webots/Motor.hpp>
#include <webots/Robot.hpp>
#include <webots/Lidar.hpp>

#define TIME_STEP 64
using namespace webots;

int main(int argc, char **argv) {
  Robot *robot = new Robot();
  DistanceSensor *ds[5];
  char dsNames[5][25] = {"ds_fright", "ds_fleft", "d
  for (int i = 0; i < 5; i++) {
    ds[i] = robot->getDistanceSensor(dsNames[i]);
    ds[i]->enable(TIME_STEP);
  }
  Motor *wheels[6];
  char wheels_names[6][12] = {"wheel1", "wheel2", "w
  for (int i = 0; i < 6; i++) {
    wheels[i] = robot->getMotor(wheels_names[i]);
    wheels[i]->setPosition(INFINITY);
    wheels[i]->setVelocity(0.0);
  }
  int avoidObstacleCounter = 0;
  while (robot->step(TIME_STEP) != -1) {
    double leftSpeed = 1.0;
    double rightSpeed = 1.0;
    if (avoidObstacleCounter > 0) {
      avoidObstacleCounter--;
      leftSpeed = 1.0;
      rightSpeed = -1.0;
    } else { // read sensors
      for (int i = 0; i < 5; i++) {
        if (ds[i]->getValue() < 950.0)
          avoidObstacleCounter = 100;
      }
    }
    wheels[0]->setVelocity(leftSpeed);
    wheels[1]->setVelocity(rightSpeed);
    wheels[2]->setVelocity(leftSpeed);
    wheels[3]->setVelocity(rightSpeed);
    wheels[4]->setVelocity(leftSpeed);
```

Figure 6.3: Complete simulation environment for the six-wheeled robot

6.5 Simulation Results and Discussions

In the hospital environment, the robot will sometimes need to move through a narrow passage. Figure 6.4 shows the six-wheeled robot moving through the narrow space between two obstacles. This validates the model is capable of navigating properly through the limited space.

The robot is also capable of avoiding static obstacles and walls inside the hospital environment. Figure 6.5 and Figure 6.6 exhibit the robot avoiding the wall and the obstacle, respectively. Hence, the simulated robot model was validated to move inside the hospital indoor environment while avoiding obstacles.

6.6 Conclusions

A hospital-like environment and a six-wheeled robot model were created using Webot robot simulator. The navigation controller programmed for the robot model was able to avoid static obstacles and walls. The robot model could also move through the narrow passage, which resembles the narrow passages of hospitals.

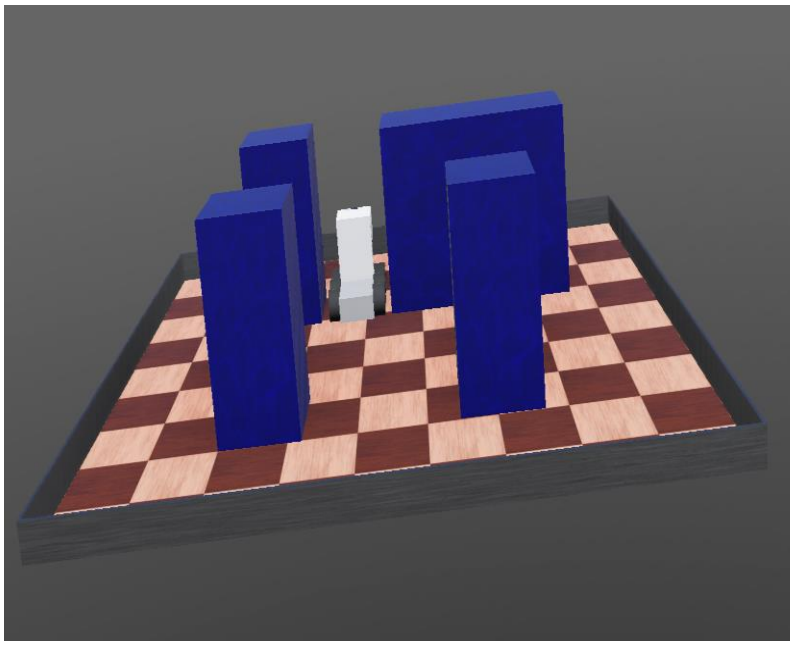
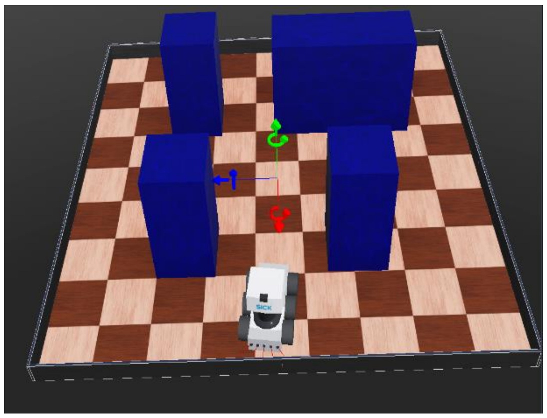
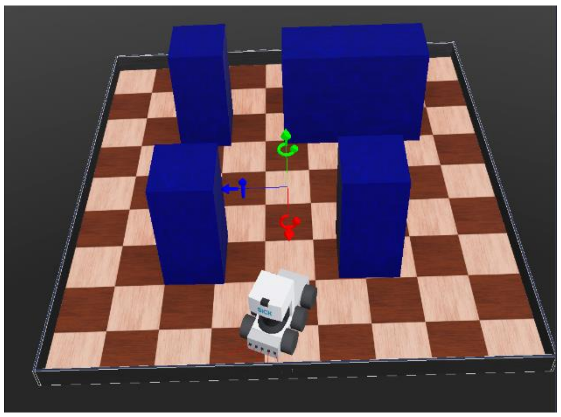


Figure 6.4: Six-wheeled robot moving through narrow passage


(a) Position 1


(b) Position 2

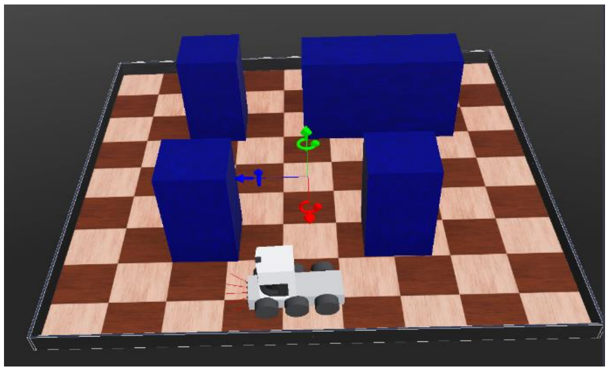
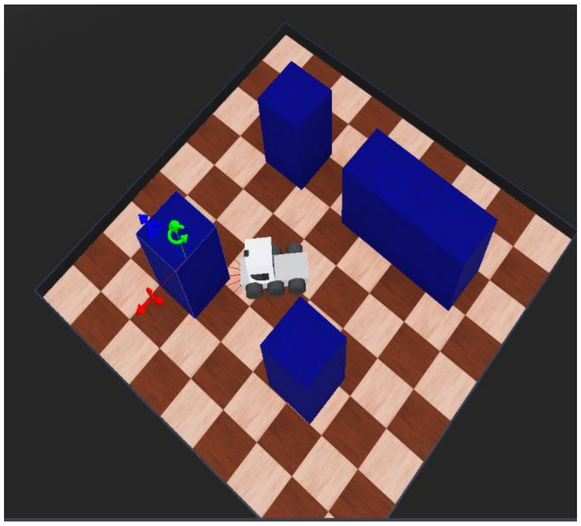

(c)Position 3

Figure 6.5: Six-wheeled robot avoiding walls


(a) Position 1

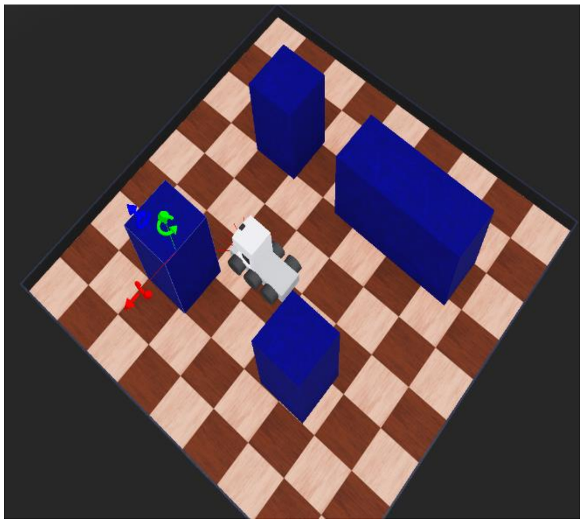

(b) Position 2

Figure 6.6: Six-wheeled robot avoiding static obstacles

CHAPTER 7
CONCLUSIONS AND SUGGESTIONS FOR FUTURE WORK

The goal of this research project was to develop the design and navigation technique of a six-wheeled differential drive robot from scratch. The robot was intended to carry medical supplies and hand over the supplies to the patients and authorized medical personnel.

In the first step, we designed a six-wheeled differential drive robot that has compartments to keep and deliver medical supplies and patient records to authorized personnel while navigating in an indoor environment. The robot base and medical supply carrying compartment were built in real life. The maneuver of the robot base was tested in real life.

Again, energy consumption plays an important role in the operation of a robot. Sometimes, even an advanced intelligent robot can’t operate for a longer period of time. Hence, investigating the energy or kinematic modeling comes into importance of having a better energy optimization of a robot. Therefore, we developed a kinematic model for our six-wheeled robot using Simulink and showed that it is avoiding obstacles using the potential field method.

Mapping is one of the most important required capabilities that a mobile robot needs to have for indoor navigation. We investigated ROS and SLAM algorithms to establish a mapping scheme. These ROS and SLAM algorithms were implemented on a TurtleBot 2 before applying these to the six-wheeled robot. We faced difficulty with generating a map for the indoor environment of the robotics lab using ROS and SLAM algorithms. Therefore, a hospital-like indoor environment and a six-wheeled robot model were created in Webot simulator. The robot model was successful in avoiding obstacles and walls and maneuvered through limited space in the simulation. Hence, the robot model attained proper navigation techniques to move inside the hospital environment.

In our thesis, the kinematic modeling of the robot was simulated utilizing an energy cost function developed on the basis of energy losses due to DC motors and kinetic energy. But the developed energy cost function is incomplete because it ignores energy consumption elements such as the gear mechanism attached to the DC motors, onboard electronics, friction between ground and wheels of the robot, and losses due accelerations and deceleration. Future researchers may dig deep to develop an energy cost function adding the above-mentioned energy consumption elements. This cost function or energy model can lead the way to establish a more accurate kinematic modeling for the six-wheeled robot.

We may also use this improved energy cost function to have better optimization of the energy consumption in the six-wheeled robot. In the process, optimization techniques such as genetic algorithm, Lagrange Multiplier method, etc. can be utilized to develop a more accurate optimal velocity profile from the improved kinematic model. The robot will follow that profile while moving from a starting point to a goal point, and as a result, the energy efficiency of the robot will be improved.

In the future, ROS and SLAM algorithms can be investigated more deeply to get a map of the environment. This map will help the six-wheeled differential drive robot to move from one position to another position in real life. In the process, the robot will be able to attain an effective navigation technique in real life alongside the simulation environment. Besides, MATLAB and lidar sensors can also be utilized to have a proper navigation algorithm for the robot in real life. In this process, a probabilistic grid map can be generated by maneuvering the robot first in the indoor environment of the hospital using the SLAM algorithm. The data of the map are stored as a 3D point cloud. MATLAB robotics system toolbox can be used to extract that 3D lidar scanned map data. Finally, proper path planning and path following algorithms can be implemented to make the robot move from one point to another along a specific path inside the indoor map.

REFERENCES

Aagela, H., Al-Nesf, M., & Holmes, V. (2017). *An Asus_xtion_probased indoor MAPPING using a Raspberry Pi with Turtlebot robot.* Paper presented at the 2017 23rd International Conference on Automation and Computing (ICAC).

Almasri, M. M., Alajlan, A. M., & Elleithy, K. M. J. I. S. j. (2016). Trajectory planning and collision avoidance algorithm for mobile robotics system. *16*(12), 5021-5028.

Balkcom, D. J., & Mason, M. T. (2000). *Time optimal trajectories for bounded velocity differential drive robots.* Paper presented at the Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065).

Bergeon, Y., Křivánek, V., & Motsch, J. (2019). *Raspberry Pi as an Interface for a Hardware Abstraction Layer: Structure of Software and Extension of the Turtlebot 2–Kobuki Protocol.* Paper presented at the 2019 International Conference on Military Technologies (ICMT).

Filipenko, M., & Afanasyev, I. (2018). *Comparison of various slam systems for mobile robot in an indoor environment.* Paper presented at the 2018 International Conference on Intelligent Systems (IS).

Fu-guang, D., Peng, J., Xin-qian, B., & Hong-Jian, W. (2005). *AUV local path planning based on virtual potential field.* Paper presented at the IEEE International Conference Mechatronics and Automation, 2005.

Garcia, E., Jimenez, M. A., De Santos, P. G., Armada, M. J. I. R., & Magazine, A. (2007). The evolution of robotics research. *14*(1), 90-103.

Gu, K., Wang, H., & Zhao, M. (2007). *The analyse of the influence of external disturbance on the motion of a six-wheeled lunar rover.* Paper presented at the 2007 International Conference on Mechatronics and Automation.

Gupta, A. K., & Gupta, V. K. (2013). *Design and development of six-wheeled Multi-Terrain Robot.* Paper presented at the 2013 International Conference on Control, Automation, Robotics and Embedded Systems (CARE).

Hamzeh, O., & Elnagar, A. (2015). *A Kinect-based indoor mobile robot localization.* Paper presented at the 2015 10th International Symposium on Mechatronics and its Applications (ISMA).

Heikkinen, J., Minav, T., & Stotckaia, A. D. (2017). *Self-tuning parameter fuzzy PID controller for autonomous differential drive mobile robot.* Paper presented at the 2017 XX IEEE International Conference on Soft Computing and Measurements (SCM).

Hsu, G.-S. J., & Huang, J.-W. (2016). *A photographer robot with multiview face detector.* Paper presented at the 2016 IEEE International Conference on Industrial Technology (ICIT).

Ibáñez, A. L., Qiu, R., & Li, D. (2017). *An implementation of SLAM using ROS and Arduino.* Paper presented at the 2017 IEEE International Conference on Manipulation, Manufacturing and Measurement on the Nanoscale (3M-NANO).

Jeffril, M. A., & Sariff, N. (2013). *The integration of fuzzy logic and artificial neural network methods for mobile robot obstacle avoidance in a static environment.* Paper presented at the 2013 IEEE 3rd International Conference on System Engineering and Technology.

Kim, C. H., Kim, B. K. J. J. o. I., & Systems, R. (2007). Minimum-energy translational trajectory generation for differential-driven wheeled mobile robots. *49*(4), 367-383.

Kondo, K., Ohba, M., & Kashiwase, S. (2016). *Efficiency evaluation of a super-directive masking system that tracks the masker beam towards the target human head.* Paper presented at the 2016 IEEE 5th Global Conference on Consumer Electronics.

Konduri, S., Torres, E. O. C., Pagilla, P. R. J. J. o. D. S., Measurement,, & Control. (2017). Dynamics and control of a differential drive robot with wheel slip: application to coordination of multiple robots. *139*(1).

Lee, W. C., Salam, A. S. A., Ibrahim, M. F., Rahni, A. A. A., & Mohamed, A. Z. (2015). *Autonomous industrial tank floor inspection robot.* Paper presented at the 2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA).

Li, Y. (2010). *Dynamic Simulation Analyses of a Six-leg-wheel Hybrid Mobile Robot under uneven terrains.* Paper presented at the 2010 Third International Conference on Intelligent Networks and Intelligent Systems.

Liao, M., Wang, D., & Yang, H. (2019). *Deploy Indoor 2D Laser SLAM on a Raspberry Pi-Based Mobile Robot.* Paper presented at the 2019 11th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC).

Magyar, B., Forhecz, Z., & Korondi, P. (2003). *Developing an efficient mobile robot control algorithm in the Webots simulation environment.* Paper presented at the IEEE International Conference on Industrial Technology, 2003.

Mester, G. (2006). *Distance Learning in Robotics.* Paper presented at the Proceedings of The Third International Conference on Informatics, Educational Technology and New Media in Education.

Mishra, R., & Javed, A. (2018). *ROS based service robot platform.* Paper presented at the 2018 4th International Conference on Control, Automation and Robotics (ICCAR).

Nandi, G. (2013). *Human-robot communication through visual game and gesture learning.* Paper presented at the 2013 3rd IEEE International Advance Computing Conference (IACC).

Oajsalee, S., Tantrairatn, S., & Khaengkarn, S. (2019). *Study of ROS Based Localization and Mapping for Closed Area Survey*. Paper presented at the 2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR).

Obdržálek, Z. (2016). *Software environment for simulation of UAV multi-agent system.* Paper presented at the 2016 21st International Conference on Methods and Models in Automation and Robotics (MMAR).

Park, S., & Lee, G. (2017). *Mapping and localization of cooperative robots by ROS and SLAM in unknown working area.* Paper presented at the 2017 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE).

Prio, M. H., & Rios, F. (2019). *Kinematic Modeling of a Six Wheeled Differential Drive Intelligent Robot and Potential Field Method to Attain Obstacle Avoidance Capability.* Paper presented at the 2019 SoutheastCon.

Scott, A. F., & Yu, C. (2009). *Cooperative multi-agent mapping and exploration in Webots®.* Paper presented at the 2009 4th International Conference on Autonomous Robots and Agents.

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*: MIT press.

Singh, J., & Chouhan, P. S. (2017). *A new approach for line following robot using radius of path curvature and differential drive kinematics.* Paper presented at the 2017 6th International Conference on Computer Applications In Electrical Engineering-Recent Advances (CERA).

Stan, A.-C., & Oprea, M. (2019). *A Case Study of Multi-Robot Systems Coordination using PSO simulated in Webots.* Paper presented at the 2019 11th International Conference on Electronics, Computers and Artificial Intelligence (ECAI).

Syaqur, W. A., Yeon, A. S. A., Abdullah, A. H., Kamarudin, K., Visvanathan, R., Ismail, A. H., . . . Zakaria, A. (2018). *Mobile Robot Based Simultaneous Localization and Mapping in UniMAP's Unknown Environment.* Paper presented at the 2018 International Conference on Computational Approach in Smart Systems Design and Applications (ICASSDA).

Wahab, M., Rios-Gutierrez, F., & El Shahat, A. (2015). *Energy modeling of differential drive robots*. Paper presented at the 2015 SoutheastCon.

Wang, L., Tan, K., & Prahlad, V. (2000). *Developing Khepera robot applications in a Webots environment.* Paper presented at the MHS2000. Proceedings of 2000 International Symposium on Micromechatronics and Human Science (Cat. No. 00TH8530).

Wang, X., Xu, X., Feng, Z., & Wu, H. (2018). *Structural Optimization Design and Performance Simulation of A Novel Six-wheel-legged Mobile Robot.* Paper presented at the 2018 IEEE International Conference on Robotics and Biomimetics (ROBIO).

Wu, X., Abrahantes, M., & Edgington, M. (2016). *MUSSE: A designed multi-ultrasonic-sensor system for echolocation on multiple robots.* Paper presented at the 2016 Asia-Pacific Conference on Intelligent Robot Systems (ACIRS).

Xixia, L., & Wenwen, W. (2017). *Analysis of the six wheels of bow-swing arm robot obstacle crossing.* Paper presented at the 2017 2nd International Conference on Robotics and Automation Engineering (ICRAE).

Xu, X., Hu, J., & Li, K. (2018). *Downhill Stability Analysis and Dynamics Simulation of the Six-wheel-legged Mobile Robot.* Paper presented at the 2018 3rd International Conference on Advanced Robotics and Mechatronics (ICARM).

Yin, L., Yin, Y., & Lin, C. J. J. A. J. o. C. (2009). A new potential field method for mobile robot path planning in the dynamic environments. *11*(2), 214-225.