Line Follower Obstacle Avoidance Robot – Final Report

By

Preshen Govender
21355491

Submitted in the Faculty of Engineering: Department of Electronic Engineering

in Partial Fulfilment of the Requirements for the

Bachelor of Technology in Electronic Engineering

at the

Durban University of Technology

August 2020

______________________________ ______________________________

Signature of Student Signature of Supervisor

____

Date

PLAGIARISM DECLARATION

1. I know and understand that plagiarism is using another person’s work and pretending it is one’s own, which is wrong.

2. This report is my own work.

3. I have appropriately referenced the work of other people I have used.

4. I have not allowed, **and will not allow**, anyone to copy my work with the intention of passing it off as his/her own work.

______________________________ ________________ ______________________________

Surname and Initials Student Number Signature

____

Date

TABLE OF CONTENTS

List of Figures

List of Tables

Constants and Abbreviations

GPS - Global Positioning System

IoT- Internet of Things

GPIO - General Purpose Input/Output

PWM – Pulse Width Modulation

HMI – Human Machine Interface

iOS – iPhone Operating System

LED – Light Emitting Diode

Wi-Fi – Wireless Fidelity

IDE – Integrated Development Environment

R-Pi – Raspberry Pi

Hz – Hertz

CHAPTER 1

1.1 Introduction

The concept of autonomous cars is not new. The car manufacturer Tesla has perfected it and is now available on all their production cars. The use of self-driving robots has also been implemented in Ford factories.

General Motors was the first company to produce an autonomous car by using electromagnetic fields generated with magnetized metal spikes embedded in the roadway. The front of the car was fitted with pick up coils that could detect the current flowing through a wire in the road. The current could be manipulated to tell the vehicle to move the steering wheel left or right.[1]

At present, many vehicles on the road are semi-autonomous due to safety features like assisted parking and braking systems, and a few have the capability to drive, steer, brake, and park themselves. Autonomous vehicle technology relies on GPS capabilities as well as advanced sensing systems that can detect lane boundaries, signs and signals, and unexpected obstacles. Autonomous vehicles are expected to bring with them a few different benefits, but the most important one is likely to be improved safety on the roads. The number of accidents caused by impaired driving is likely to drop significantly. [1]

Using this idea, autonomous robots can be produced for industry use. It can be used to move material from one process line to the next without relying on human driving errors. It can be used in restaurants and hospitals to deliver food and medication respectively at a time when social distancing is essential.

1.2 Project Specifications

The aim of the project is to is to produce a robot that can drive autonomously in the correct environment. The robot will not rely on human inputs to carry out its driving, thus eliminating accidents in the industry. The robot will require a ‘brain’ to process the correct movements and complete the relevant checks before it can allow any movements.

The robot will have sensors at the front in order to drive autonomously. These sensors will have to detect a light-coloured surface from a dark path as well as know how far away from the path it is. The further away the robot from the line, tighter and quicker turns are needed to put the robot back on its path. Infrared sensors can be used for this application. The transmitter sends an infrared signal, which can be received by the receiver. However, signals are easily reflected by light-coloured surfaces whereas on dark-coloured the signal is absorbed therefore not received by the receiver. This is how the sensors can tell the difference between light and dark.

The robot will also have to aware of its surroundings to avoid any crashes as you would expect from an autonomous car. The is necessary in the real world as when we consider road safety, pedestrians are the number one priority. The robot will have to know how far it is from an obstacle before it moves in any direction. The robot will also have to send all the data wirelessly to a mobile app.

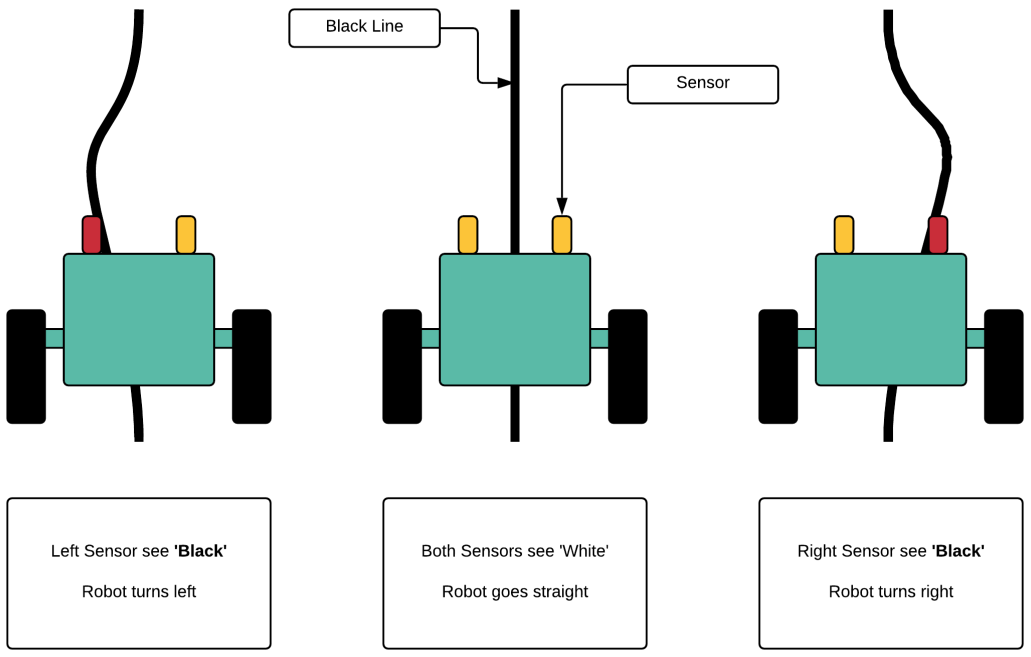


Figure 1.1: Basic Operation of a Line Follower Vehicle

1.3 Project Management

Figure 1.2, a time log bar graph, shows the amount of time spent on the relevant tasks in order to complete this project.

From ***Figure 1.2***, we notice that the most time spent of the project was the software development stage. The reason for this is that a new programming language had to be learnt. It took many hours of research to learn the different methods of programming in Python.

The structural build time is also long but constant changes were made to the materials and length of the wheel-base to ensure the project functioned correctly. Materials were changed to improve weight and the length of the wheelbase was changed to ensure tighter turns.

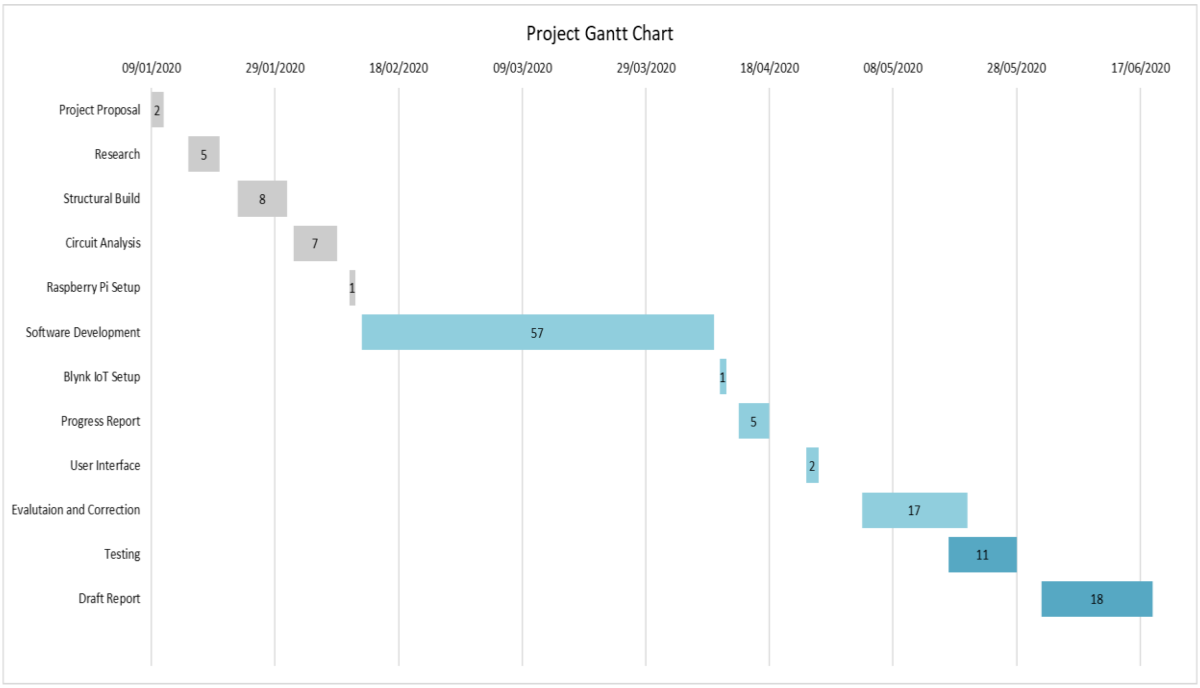


Figure 1.2 – Time Log of project

1.4 Lessons Learned

Continuous planning and working under pressure were needed throughout the project build. Managing the project regarding how certain inputs can delay other processes or improve their capabilities. It was found that when running the motors at full power caused the line follower to miss some turns as it could not stop in time to turn along the line therefore steering off the course.

The most significant lesson learnt was the new programming language, Python. It is a complex programming language that needed to be learnt in order to program the Raspberry Pi. Also using the Raspbian OS and how to load it onto the R-Pi was valuable.

Use of the Blynk IoT interface is simple when using dummy GPIO pins. However, not many realize the ease of this IoT system. It being recommended by a fellow student has greatly improved my project.

1.5 Compliance

The following needed to be considered:

- Accuracy of mechanics – control of the motors and input signals from sensors.
- IoT system – using Blynk IoT.
- Showing the input and output activity of the system on the user interface.
- Monitoring the Raspberry Pi data
- Structural integrity.

CHAPTER 2

2.1 Design Specification

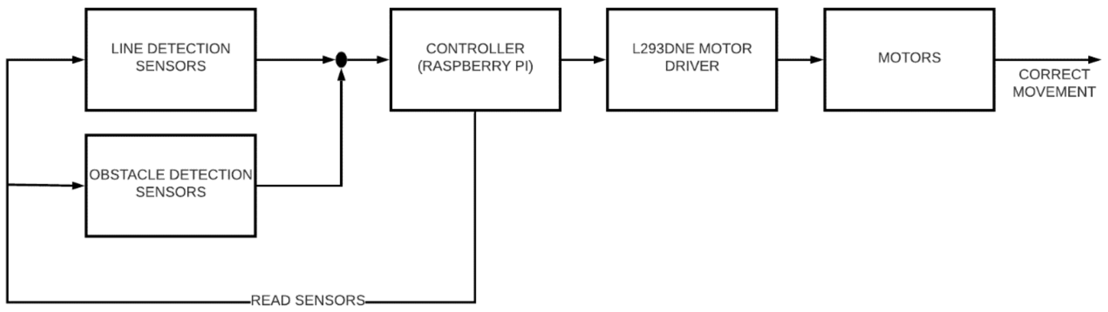


Figure 2.1 – Block Diagram

Figure 2.1 shows the block diagram of the system. The two 3-way channel IR sensors will be used to detect the black line, switching between a digital 'HIGH' and 'LOW' depending on whether the infrared signal is received or not. The two FC-51 sensors mounted on the front will detect if there are obstacles in the robot's path, switching between a digital 'HIGH' and 'LOW' depending on whether the infrared signal is received or not. These signals are sent to the Raspberry Pi as inputs. The Raspberry Pi will then use these inputs to supply the L293DNE motor driver with the appropriate PWM signal and the L293DNE then supplies the motor with a the appropriate amplified PWM signal. This PWM signal is supplied by the R-Pi. The duty cycle of this signal determines the average output.

The following equations were used to calculate the average output:

$$Average\ Output = PWM\ Voltage\ \times Duty\ Cycle$$

[2-1]

The duty cycle of a PWM Signal determines the voltage output. If we apply a 50% duty cycle to a 5V PWM, the average output voltage will be 2,5V. Therefore, by using this concept, we can manipulate a signals average output voltage. ***Figure 2.2*** shows the average output signals of different duty cycles applied to a 5V PWM Signal.

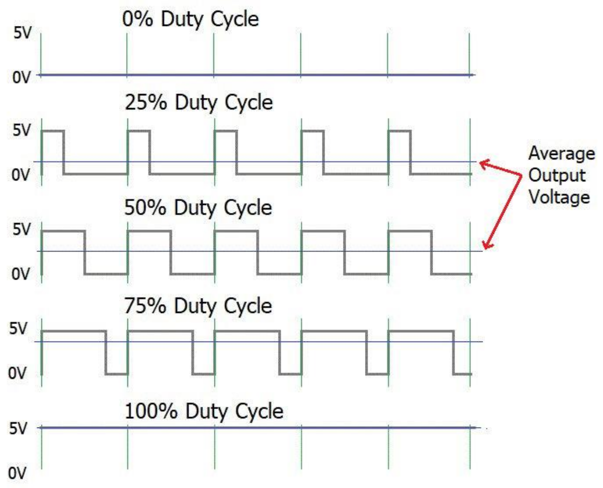


Figure 2.2 – PWM Operation [2]

The two FC-51 Obstacle sensors mounted at the front will transmit signals to the Raspberry Pi. This will be the first check the robot will need to make before any movement is made. When the sensor detects an obstacle, it transmits a digital '0'. When the sensor does not detect an obstacle, it transmits a digital '1'. The robot will stop if either of the sensors are detecting obstacles. The Raspberry Pi will then determine the appropriate movement needed by checking the two 3-way channel IR sensors. ***Table 2.1*** describes the obstacle sensing aspect of the robot:

Sensor 1 - Output	Sensor 2 - Output	Output
0	0	Stop Movement
0	1	Stop Movement
1	0	Stop Movement
1	1	Allow Movement

Table 2.1 – Obstacle Sensors

The 4 line tracking sensors closest to the center will work as changing PWM signals. The 2 outside line tracking sensors will work as boosting PWM signals. When the outside signals detect the line the PWM applied to the motors are boosted as a higher degree of turning is needed. For forward motion a custom PWM signal is applied to allow the robot to move in a straight line. The reason for this is no 2 products are the same. The motors used in the project had slight differences therefore a custom signal is needed to allow the motor to move straight. The duty cycle values were determined by trial and error. Running the robot across various degrees of turns with different duty cycles and choosing the best performing result. Setting up a series of turns starting at 30 degrees and ending at 100 degrees with a interval of 10 degrees. ***Figure 2.3*** shows the layout of sensors.

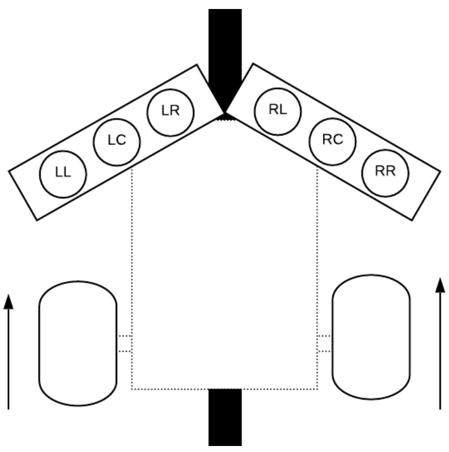


Figure 2.3 – Layout of Sensors

The truth table for these conditions are as follows:

LC	**LR**	**RL**	**RC**	**Movement**	**Duty Cycle Supplied**
0	0	0	0	Stop	0%
0	0	0	1	Left	90%
0	0	1	0	Left	70%
0	0	1	1	Left	80%
0	1	0	0	Right	70%
0	1	0	1	Don’t care	-
0	1	1	0	Don’t Care	-
0	1	1	1	Left	80%
1	0	0	0	Right	90%
1	0	0	1	Stop	0%
1	0	1	0	Don’t care	-
1	0	1	1	Left	70%
1	1	0	0	Right	80%
1	1	0	1	Right	70%
1	1	1	0	Right	80%
1	1	1	1	Forward	Custom

Table 2.2 – Truth Table

Before the Raspberry Pi can execute the above PWM Signals, it will first check if sensor ‘LL’ and ‘RR’ are detecting the black line. If the sensors are detecting the black line the PWM signal from the previous table is boosted. The following table illustrates this:

Sensor	**Is the line detected?**	**Duty Cycle**
LL	Yes	Duty Cycle = Duty Cycle + Boost
LL	No	Duty Cycle = Duty Cycle
RR	Yes	Duty Cycle = Duty Cycle + Boost
RR	No	Duty Cycle = Duty Cycle

Table 2.3 – Boosting Signals

Next we look at how the PWM signals are applied to dictate the direction of motion. As this is a 3 wheeled robot with the front wheel a free-spinning wheel, the 2 back wheels will have to turn the robot in the required direction. The L293DNE motor driver connects 2 of its pins to the motor therefore depending on which side of the motor the signal is applied; the motor can turn in both clockwise and anti-clockwise direction. In order to move left, the right motor should move in a clockwise direction and the left motor should move in an anti-clockwise direction. Similarly, in order to move right, the right motor should move in an anti-clockwise direction and the left motor should move in a clockwise direction. For a forward movement both wheels should turn in a clockwise direction. Careful wiring consideration is needed for the wheel to turn in the same way. However, no matter which way the motor is wired, the desired movement can be manipulated in the code. The table shows how the PWM signals are applied for movement across each of the motor terminals:

Movement	Left Motor +	Left Motor -	Right Motor +	Right Motor -
Forward	65% Duty Cycle	0	45% Duty Cycle	0
Left	0	Pre-Determined Duty Cycle	Pre-Determined Duty Cycle	0
Right	Pre-Determined Duty Cycle	0	0	Pre-Determined Duty Cycle
Stop	0	0	0	0

Table 2.4 – PWM applied for Movement

The Blynk IoT app will be the HMI of the robot. It will provide PWM details for each motor terminal and show the sharpness of a left of right turn using a scale. The obstacle sensor outputs will also be displayed, and the HMI will provide information about movements and whether obstacles are in its path.

2.2 Components

The following components were to execute the project design specification

2.2.1 Raspberry Pi Model 3 B+

The R-Pi 3 is the third-generation microcontroller of the R-Pi Series. The project will use the GPIO Pins on the R-Pi. The R-Pi has a total of 40 GPIO pins with 28 usable GPIO Pins to connect inputs and outputs. The GPIO Pins have a maximum voltage rating of 3.3V. Therefore, extra consideration is needed when trying to interface with these pins as some signals received from sensors are higher. The GPIO pins also have the capability of outputting pulse width modulation signals. ***Figure 2.4*** shows the layout of all 40 GPIO pins.

The R-Pi will run the Raspbian Operating System. This is a Linux based operating system and the programs written for this project is coded in Python.

Python is an interpreted, high-level, general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects[3]. The Thonny Python IDE (pre-loaded on Raspbian) is used to debug and write code to the R-Pi.

The Raspberry Pi has built-in Wi-Fi so that interfacing with the project is simpler. This needed to communicate wirelessly to the interface of the project. Other technical specifications include [4]:

- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- Power-over-Ethernet (PoE) support (requires separate PoE HAT) [4]

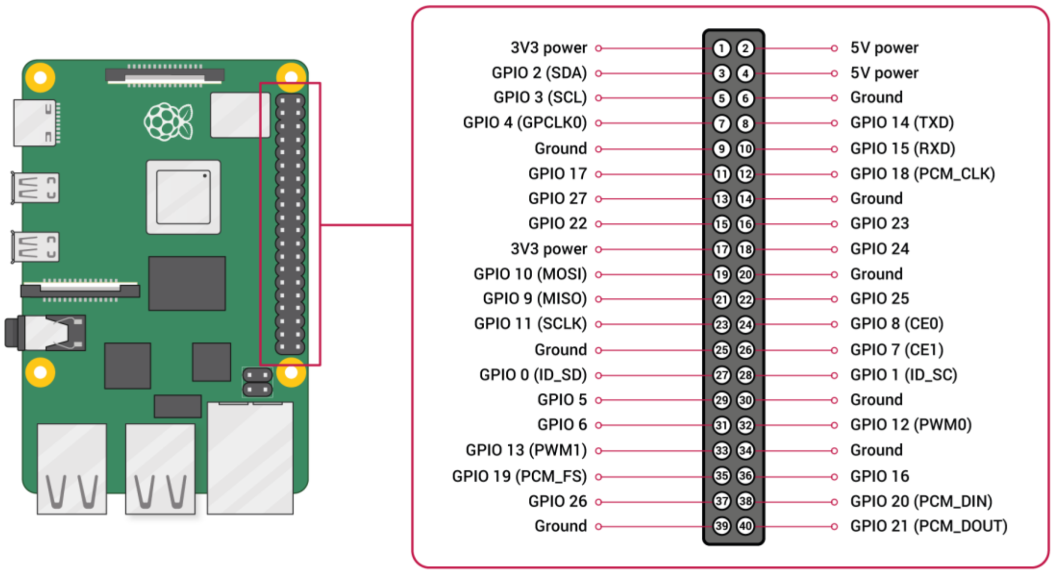


Figure 2.4 – GPIO Layout [5]

2.2.2 Geared 3V DC Motors with Wheels

The motors are used to drive the wheel of the robot. Running the motors in opposite ways enables the robot to turn faster and sharper. The motors have the following specifications:

- Operating Voltage – 3V to 12V DC
- Maximum Torque (3V) – 800 (gf.cm)
- Load Current (3V) – 70mA.

2.2.3 Funduino 3-Channel IR Line-tracking Sensor

The Funduino 3-Channel IR Sensor is an infrared sensor used to detect different surfaces. The sensor sends out infrared signals and waits for the signal to be returned. It can detect the difference in reflective and non-reflective surfaces using this principle.

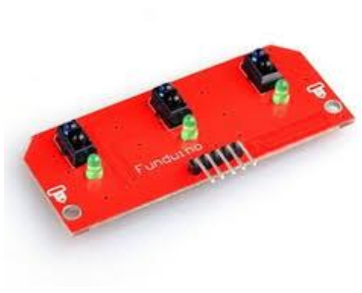


Figure 2.5 – Funduino 3-Channel IR Sensor [6]

The sensor houses 3 individual CTRT5000 sensors. For this application, the sensors will continuously send out infrared signals through the transmitter. On a reflective surface the infrared signal is reflected and received by the receiver which then outputs a digital '1'. When the sensors send out a signal on the black line, the infrared signal is absorbed and not received by the receiver, therefore the sensor outputs a digital '0'. ***Figure 2.6*** describes the operation of a single CTRT5000 sensor. For the project, 2 Funduino 3-Channel IR Sensors will be used which makes the total amount of CTRT5000 sensors 6. These will be mounted to the front of the robot. The sensor is powered by 5V and has three outputs that will be connected to different GPIO pins.

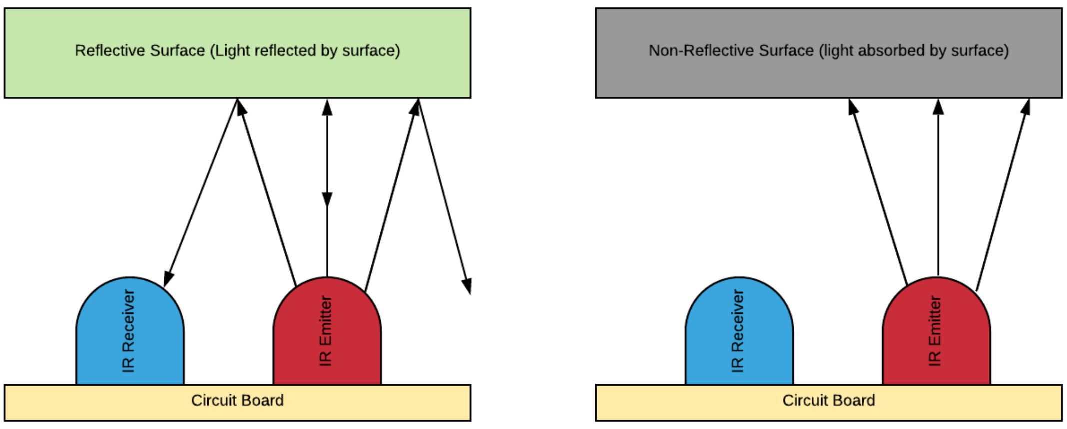

Reflective Surface (Light reflected by surface)
IR Receiver
IR Emitter
Circuit Board
Non-Reflective Surface (light absorbed by surface)
IR Receiver
IR Emitter
Circuit Board

Figure 2.6 – Infrared Sensor Operation

2.2.4 *FC-51 Obstacle Avoidance Sensor*

The FC-51 Sensor is an infrared sensor with an emitter and receiver. The emitter is an infrared LED and the receiver is a photodiode. When the IR LED fires a signal, depending on the surface the signal is reflected to the photodiode. The FC-51 produces a low digital output when an obstacle is in range and a high when there is no obstacle.

When the IR LED signal hits a surface within range, the signal is reflected to the photodiode, producing a digital low, meaning there is an obstacle that is within range. However, when the IR signal is transmitted and not received it will produce a digital high, meaning there is no obstacle in front of the sensor.

Two FC-51 sensors will be mounted to the front of the robot to detect obstacles in its path. The sensors are powered by 5V and each sensor output will occupy a GPIO pin. The sensors range can be adjusted manually.

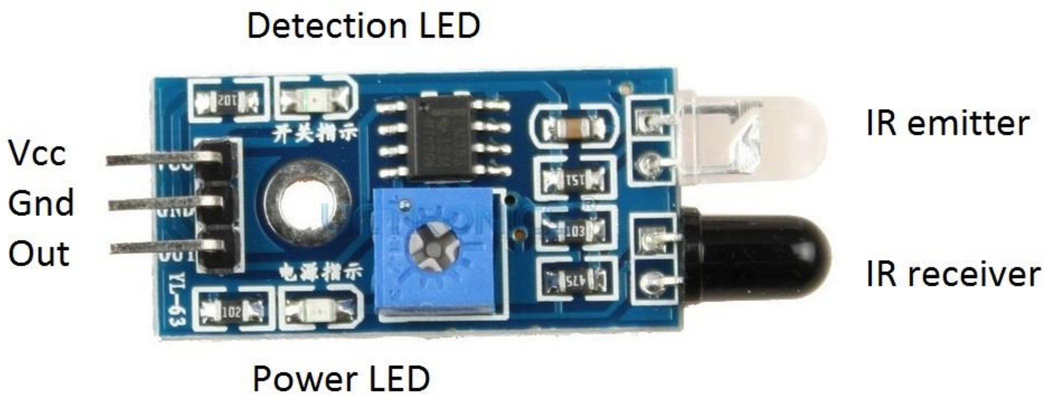


Figure 2.7 – FC – 51 Obstacle Sensor [7]

2.2.5 L293DNE Motor Driver

The L293D is a 16-pin Motor Driver IC which can control a set of two DC motors simultaneously in any direction. The L293D is designed to provide bidirectional drive currents of up to 600 mA (per channel) at voltages from 4.5 V to 36 V. For the Project, the L293DNE will drive both DC motors.

The following shows the circuit diagram of the L293DNE for one channel:

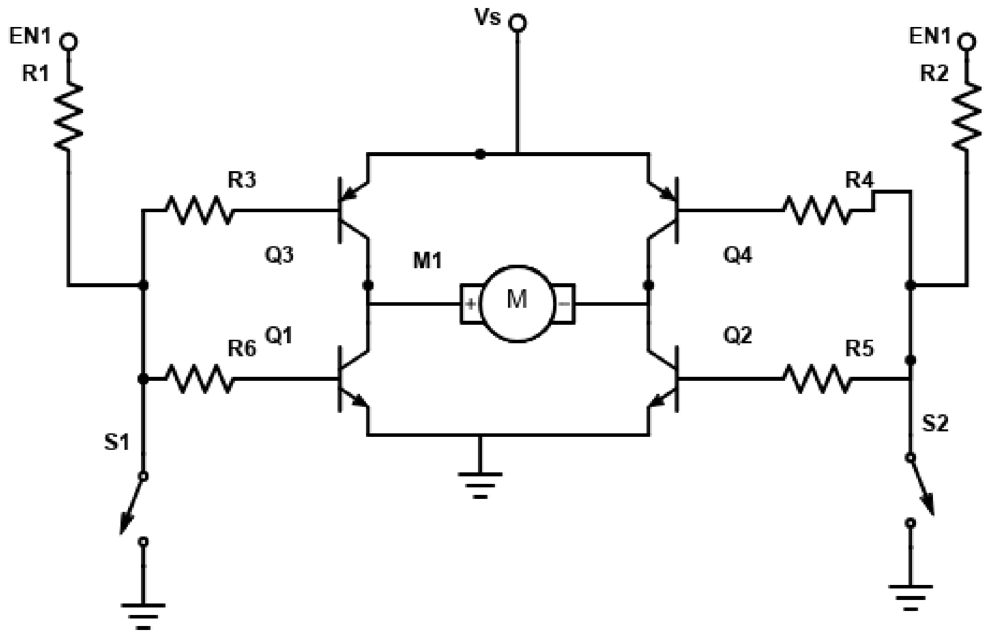


Figure 2.8 - Circuit diagram of Channel 1

When S1 is closed and S2 is opened transistor Q3 and Q2 are opened/operational, allowing current to flow from them. The motor now starts rotating. The motor direction of rotation is clockwise. One PNP and one NPN transistor is activated when one switch is closed and other is opened. If both the input 1 and input 2 are opened or closed no current path will be established and motor will not rotate. If any input is closed and enable is not at high potential, the current will not flow through the motor. Enable is a master pin and controls the whole circuit.

2.3 Structure

The structure of the project will be made from two rectangular pieces of Perspex which is 3mm thick. The rectangular pieces will be held together by bolts (4mm x 55mm). Using 4mm nuts 2 platforms are created. **Annexure A** shows the layout of the panels.

On level 1 (bottom platform), the following is mounted:

1. Power bank
2. 2 x Motors
3. 2 x Wheels
4. 1 x trolley wheel
5. 2 x FC-51 Sensors
6. 2 x 3-way channel IR sensors
7. Power Circuit for sensors

On level 2 (top platform), the following is mounted:

1. Raspberry Pi 3 Model B+
2. Driver circuit

On the bottom platform, the two FC-51 sensors and the two 3-way channel IR sensors will be mounted to the front of the robot along with the trolley wheel. The two 3-way channel IR sensors are mounted at a 30-degree angle so that it will allow the robot to turn smoothly while reading the inputs. The angle allows the increase of the duty cycle progressively. The trolley wheel will as the balancing wheel for the structure and provides smooth driving. The IR sensors wiring will travel up to the top platform and connect to the Raspberry Pi 3 Model B+ for the data to be interpreted. The power for all the sensors will come the power circuit. The driver circuit wiring will travel from the top platform to the bottom platform and connect to the motor. The driver circuit will also connect to the Raspberry Pi.

2.4 Wiring

The following describes all wiring needed for the robot. **Annexure B** contains the full circuit diagram. The connections for the sensors to the Raspberry Pi are as follows:

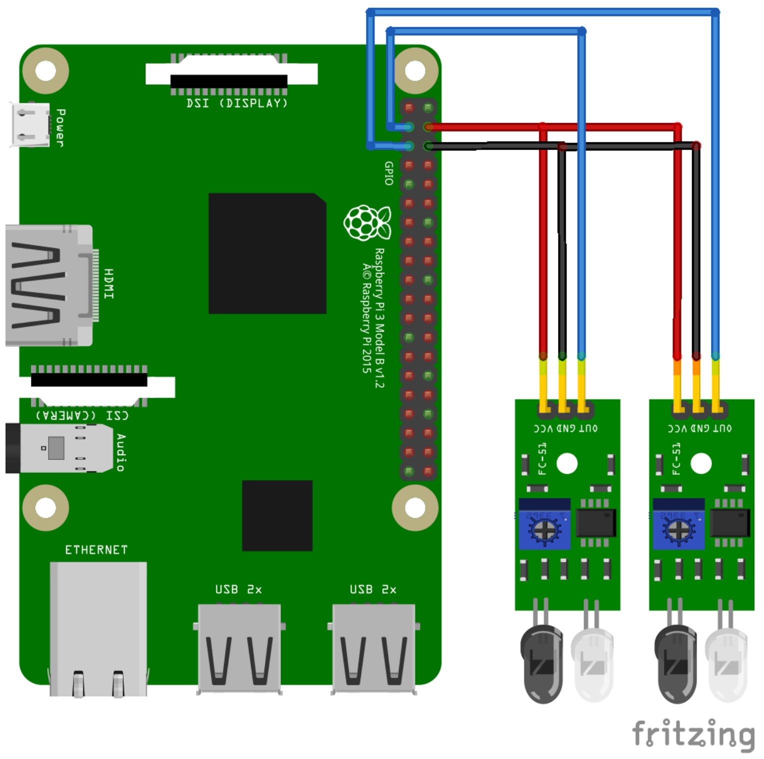


Figure 2.9 – Obstacle Sensors Wiring

Figure shows the obstacle sensors connected to the Raspberry Pi. The two FC-51 sensors used for obstacle sensing will be connected to the Raspberry Pi 5V pin and the Ground pin respectively. Each sensor output will be connected to GPIO pin. Sensor 1 is connected to GPIO 2 and sensor 2 is connected to GPIO 3.

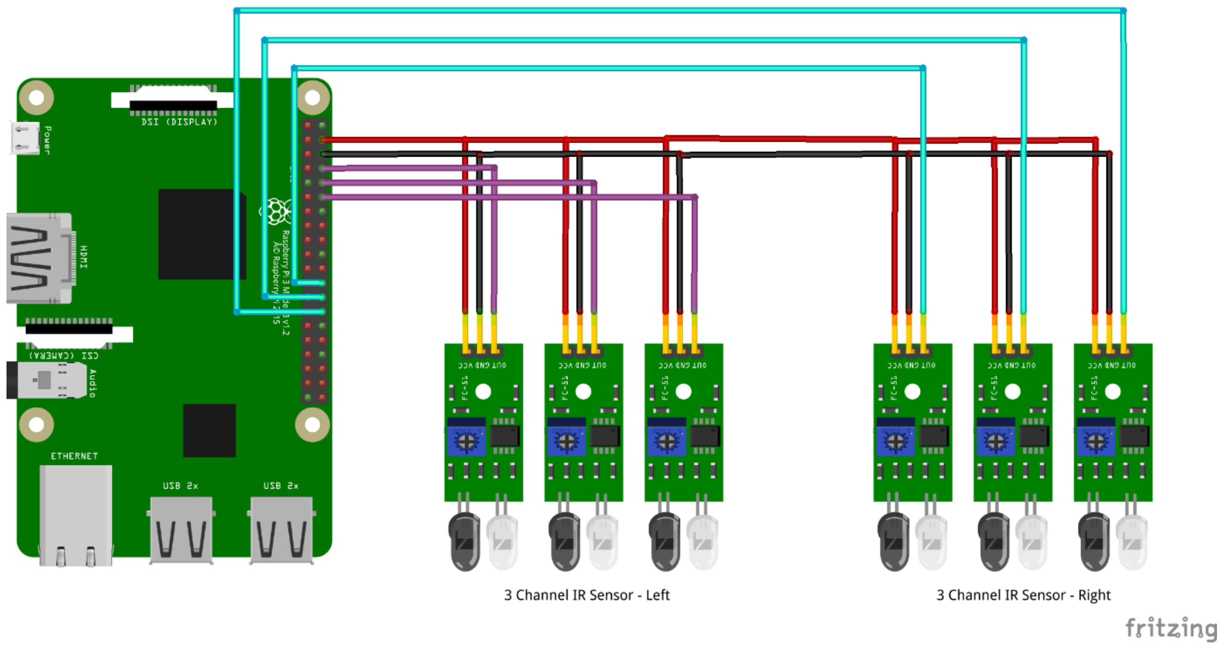


Figure 2.10 – 3 Channel IR Sensors Wiring

Figure shows the two 3-channel IR sensors connected to Raspberry Pi used for line tracking. Both sensors are connected to the 5V and Ground of the Raspberry Pi. Each channel (sensor output) is connected to a GPIO pin on the Raspberry Pi. The sensors are connected to the following GPIO pins:

Sensor Name	Position	Raspberry Pi Pin
LL	Left Sensor – Left Channel	GPIO 14
LC	Left Sensor – Center Channel	GPIO 15
LR	Left Sensor – Right Channel	GPIO 18
RL	Right Sensor – Left Channel	GPIO 8
RC	Right Sensor – Center Channel	GPIO 7
RR	Right Sensor – Right Channel	GPIO 1

Table 2.5 – 3 Channel IR Sensors Connections

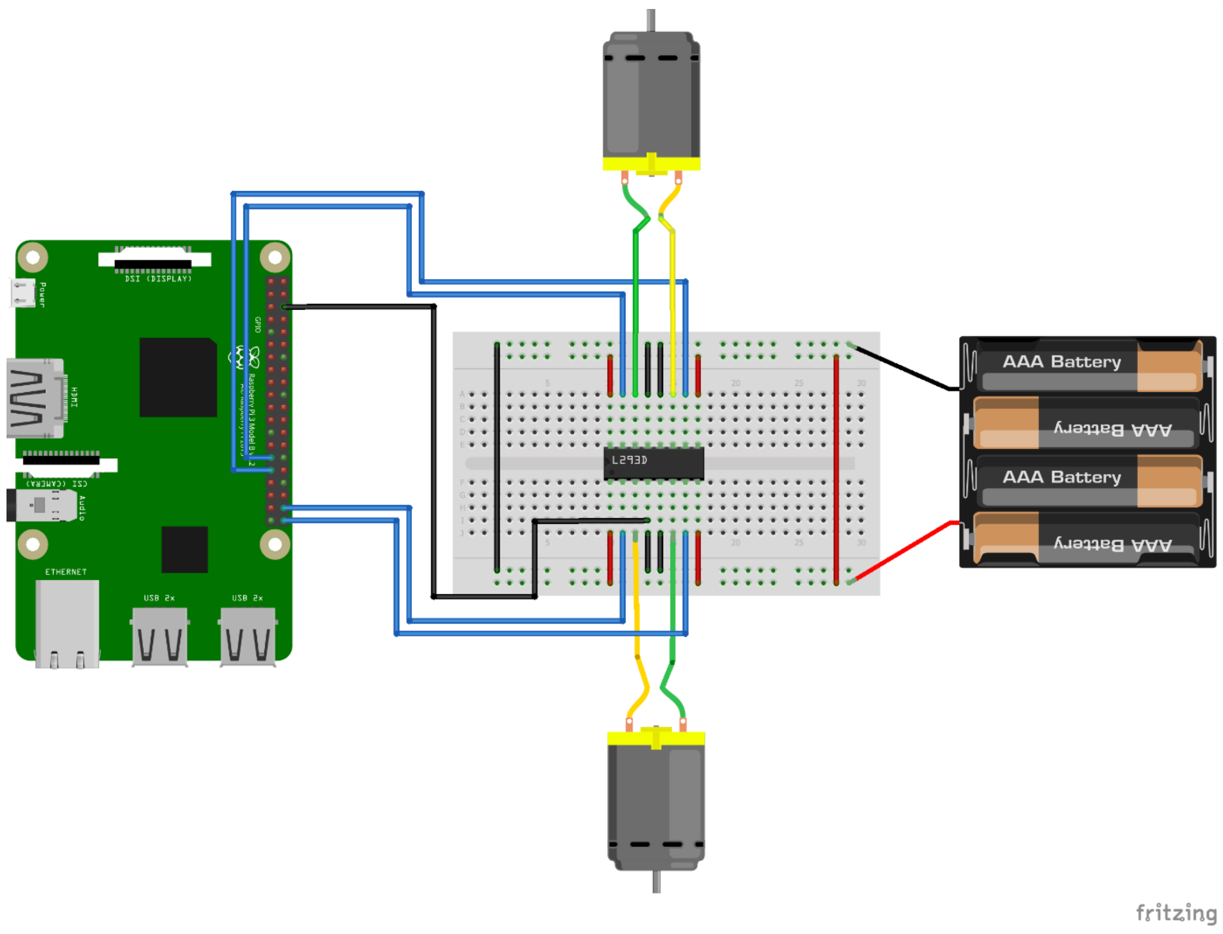


Figure 2.11 – L293DNE Wiring

Figure shows the connections for the L293DNE motor driver and motors. The motor driver is powered by a 5V/1A supply. The Raspberry will be providing 3.3V PWM signals which is converted to a 5V PWM signal by the driver circuit. The following connections are made between the driver circuit and the Raspberry Pi:

L293DNE	Raspberry Pi
Ground (Pin 4 or 5 or 12 or 13)	Ground
Pin 2 (Input 1)	GPIO 20
Pin 7 (Input 2)	GPIO 21
Pin 10 (Input 3)	GPIO 6
Pin 15 (Input 4)	GPIO 5

Table 2.6 – L93DNE Connections

The motors are connected across each terminal. The left motor is connected across Pin 11 and 14 and the right motor is connected across Pin 3 and 6.

CHAPTER 3

In this chapter, we look at the Raspberry Pi and Blynk IoT setup. For the Raspberry Pi, we will be using NOOBS and how-to setup Blynk IoT for communication. We will also setup the complete user interface for the project using Blynk.

3.1 Raspberry Pi Setup

The use of NOOBS is the recommended method for beginners using the Raspberry Pi Model 3 B+. In order to setup the Raspbian OS using NOOBS the following is needed:

- Raspberry Pi Model 3 B+
- Keyboard
- Mouse
- Screen
- HDMI cable
- MicroSD card
- MicroSD card adapter
- Computer/Laptop
- Micro USB cable

To install the Raspbian OS the following steps should be followed:

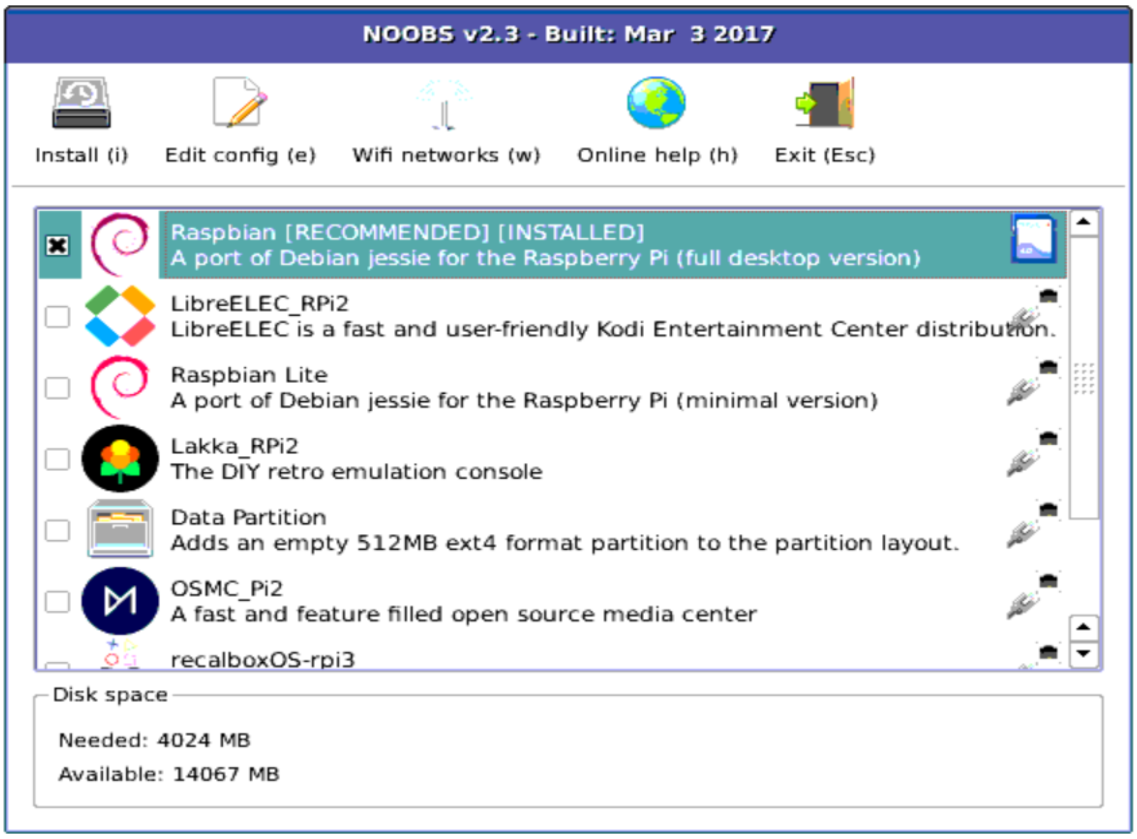
1. Insert the MicroSD card into the MicroSD car adapter and insert into the computer. The SD card will need to be formatted to file system FAT.
2. Download the NOOBS zip file from https://www.raspberrypi.org/downloads/noobs/. Once downloaded, extract the files to the formatted SD card.
3. Once extracted, insert the SD card into to the Raspberry Pi SD slot.
4. Plug in the screen using the HDMI cable and the keyboard and mouse into the USB ports on the Raspberry Pi.
5. Next, power the Raspberry Pi using the micro USB cable.

   Once plugged in:

   - The red LED should be on. This is the power indicator.

- The green LED should be flashing. This shows activity on the Raspberry Pi, meaning the SD card is working properly.

6. The Raspberry Pi will boot, and a window will appear with a list of different operating systems that you can install. Tick the box next to Raspbian and click on Install. For this project the Raspbian OS is used. ***Figure 3.1*** shows the selection screen of the different operating systems.



[✓ Raspbian [RECOMMENDED] [INSTALLED]
A port of Debian jessie for the Raspberry Pi (full desktop version)]

LibreELEC_RPi2
LibreELEC is a fast and user-friendly Kodi Entertainment Center distribution.

Raspbian Lite
A port of Debian jessie for the Raspberry Pi (minimal version)

Lakka_RPi2
The DIY retro emulation console

Data Partition
Adds an empty 512MB ext4 format partition to the partition layout.

OSMC_Pi2
A fast and feature filled open source media center

recalboxOS-rpi3

Disk space

Needed: 4024 MB
Available: 14067 MB

Figure 3.1 – OS Selection Screen [8]

7. Once selected, allow the system to install. If prompted for credentials, the following are defaults:

- Username: pi
- Password: raspberry

3.2 Blynk IoT Setup

Blynk is an online IoT dashboard that takes works well with Raspberry Pi. Blynk is a drag-and-drop programming system for the IoT that really does make it much easier. It not only makes it possible to build programs using drag-and-drop, it standardizes the connection of devices such as sensors and motors and makes sure that drivers are in place. In this sense it makes the programming and the hardware much easier.

Blynk was designed for the Internet of Things. It can control hardware remotely, it can display sensor data, it can store data and visualise it. There are three major components in the platform:

- Blynk App - allows to you create amazing interfaces for your projects using various widgets we provide. [9]
- Blynk Server - responsible for all the communications between the smartphone and hardware. You can use our Blynk Cloud or run your private Blynk server locally. It's open source, could easily handle thousands of devices and can even be launched on a Raspberry Pi. [9]
- Blynk Libraries - for all the popular hardware platforms - enable communication with the server and process all the incoming and outcoming commands. [9]

Blynk also supports several connection methods, including but not limited to:

- Wi-Fi
- Bluetooth
- GSM
- USB (Serial)
- Ethernet

Once registered for the Blynk app, an authorization token is generated by Blynk. This authorization token is a unique identifier which is needed to connect your hardware to your smartphone through the Blynk server. Every new project created will have its own authorization token.

The following steps should be followed to properly install Blynk on Android/iOS and the Raspberry Pi:

1. The Blynk application will need to be downloaded from the App Store (iOS) or Play Store (Android).
2. Next, an account needs to be created using an email and a password of your choice.

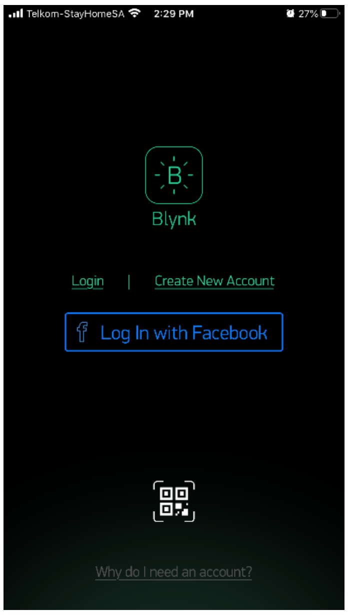


Figure 3.2 –Blynk Home Screen

3. Once completed, a new project will need to be created.

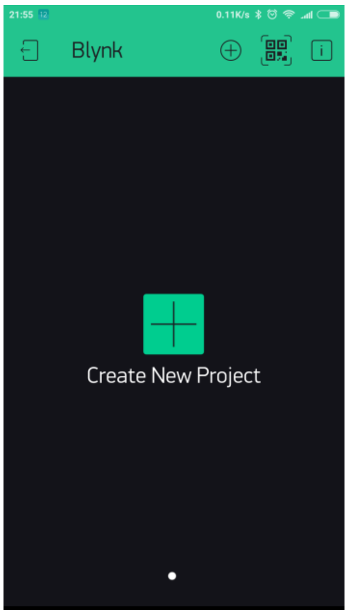
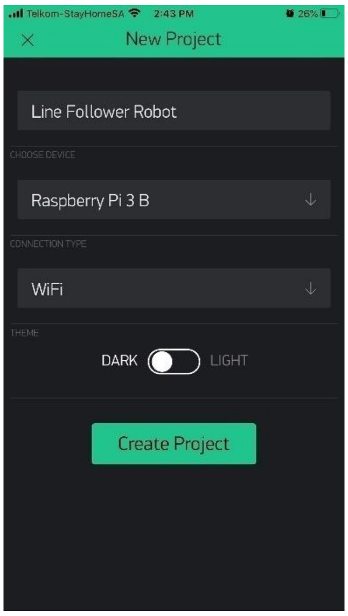


Figure 3.3 – Create New Project Screen [10]

4. Selecting the appropriate settings for this project and selecting create Project. ***Figure 3.4*** shows the settings used for this project.



New Project

Line Follower Robot

CHOOSE DEVICE

Raspberry Pi 3 B

CONNECTION TYPE

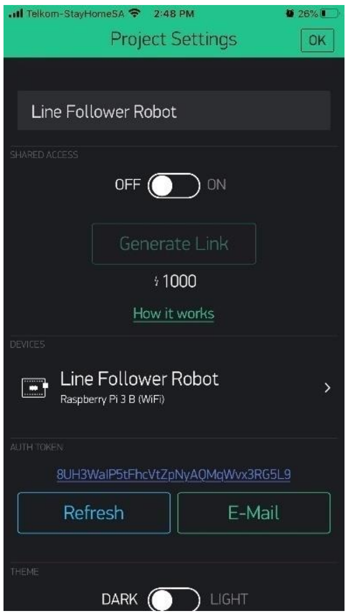
WiFi

THEME

DARK LIGHT

Figure 3.4 – Project Settings Page

5. Once created, the Blynk will provide an authorization token (AUTH TOKEN). This token will need to be programmed to the Raspberry Pi.



Project Settings

Line Follower Robot

SHARED ACCESS

OFF ON

ϟ1000

DEVICES

Line Follower Robot
Raspberry Pi 3 B (WiFi)

AUTH TOKEN

8UH3WaIP5tFhcVtZpNyAQMqWvx3RG5L9

THEME

DARK LIGHT

Figure 3.5 – Authorization Token Screen

6. Next, open the terminal on the Raspberry Pi and run the following lines of code [11]:

   - To install git core:

     ```
     sudo apt-get install git-core
     ```

   - Next, clone the Blynk project to the Raspberry Pi using:

     ```
     git clone https://github.com/blynkkk/blynk-
     ```

   - Next, navigate to the Blynk folder using:

     ```
     cd blynk-library/linux
     ```

   - Next, use the make clean command

     ```
     make clean all target=raspberry
     ```

   - Next, build the project on the Raspberry Pi using:

     ```
     ./build.sh raspberry
     ```

7. Once completed, test the connection with the code below with the authorization token[11]:

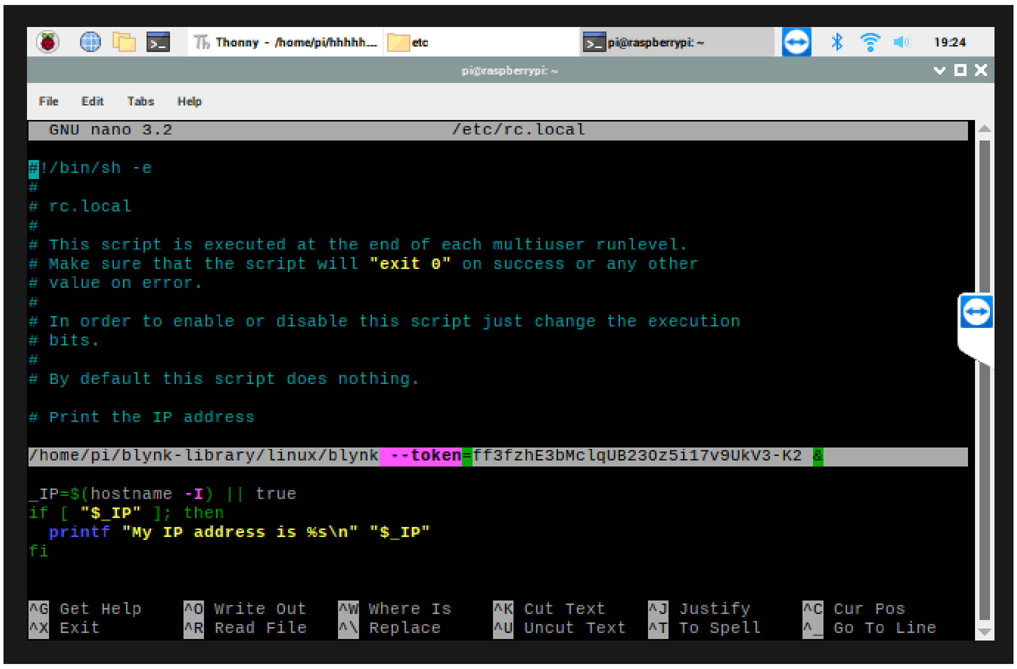
   ```
   sudo ./blynk --token=2432a5113a448fcb45e664a1934215
   ```

8. Next, the connection between the R-Pi and the iOS device will need to be automatic when the R-Pi is turned on. To do this, navigate to the boot file of the Raspberry Pi using:

   ```
   sudo nano /etc/rc.local
   ```

9. Once the file opens, the following code will need to be added as shown in ***Figure 3.7***:

   ```
   /home/pi/blynk-library/linux/blynk--token=ff3fzhE3bMclqUB23Oz5i17v9UkV3-
   K2 &
   ```



```
  GNU nano 3.2                                       /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address

/home/pi/blynk-library/linux/blynk --token=ff3fzhE3bMclqUB23Oz5i17v9UkV3-K2 &

_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi


^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line
```

Figure 3.6 – Add code to rc.local

10. Followed by the following commands:

- *Ctrl+X* – To save
- *Y* – To confirm
- *Enter* – To exit

CHAPTER 4

In this chapter, we look at the programming aspect of the project. The program needs an understanding of the Python programming language, pulse width modulation and Blynk IoT. For Python, we understand the use of functions and statements to program the Raspberry Pi. The program will be loaded to the Raspberry Pi via the Thonny Python IDE.

4.1 Python

The python programming language has many different uses including web development, software development, mathematics and system scripting. Python can also connect to database systems which is normally used for reading and modifying file or data as well as database reporting[12].

Python was designed for readability and has some similarities to the English language with influence from mathematics. Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses. Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly brackets for this purpose[12].

The following Python functions and modules were used in to build the software for the project. The functions, instructions, modules and code format had to be learnt from a beginner level. The following aspects of the Python programming language will have to be considered:

4.1.1 Import

Import in python is similar to '#include' header file in C/. Python modules can get access to code from another module by importing the file/function using import. When 'import' is used it searches for the module locally.

4.1.2 GPIO.setwarnings()

This instruction relates to the GPIO pins on the Raspberry Pi. For this project, we will disable any warnings to the GPIO to avoid the code producing an error and stopping the process.

4.1.3 GPIO.setmode()

The ‘setmode’ function helps the program to understand which pins are being used. Using ‘board mode’ will tell the program that you are using physical positions. However, for this project, ‘BCM’ is used which tells the program to point the name of the pin and not its position on board.

4.1.4 GPIO.setup()

The setup function sets the GPIO pin as an input or an output.

4.1.5 GPIO.PWM()

The PWM function sets the GPIO pin as PWM signal. This function allows the frequency to be chosen.

4.1.6 while 1:

The ‘while 1’ or ‘while true’ statement takes an expression and executes the loop body while the expression evaluates to ‘true’. True always evaluates to Boolean ‘true’ and thus executes the loop body indefinitely. Therefore, all initialization of pins should happen before the while statement as it is an infinite loop.

4.1.7 if statement

The ‘if’ statement is used to perform a logical test in python. In this project, we used the ‘if’ statement to perform a logical test between inputs.

4.1.8 elif statement

The ‘elif’ statement, understood as else if, is used to perform tests for multiple conditions. It is Python’s way or saying if the first condition did not work, try this condition.

4.1.9 else statement

The ‘else’ statement is the ‘false’ part of the ‘if’ statement. When the logical test is performed, if the condition is false, it will perform the ‘else’ conditions.

4.1.10 Using Functions

The use of functions is useful as code does not need to be repeated causing unnecessary length of code. Once written as a function it can be called at any time.

4.2 The Program

The program designed for this project has two different control aspects. The line tracking aspect of the code will make the sensors continuously send signals to the Raspberry Pi where it then uses the conditions in the code to determine the correct PWM signal to apply to the L293DNE motor driver. The obstacle sensing aspect of the code will involve the obstacle sensors continuously checking for obstacles and sending the signals to Raspberry Pi. If an obstacle is detected the motors must be stopped and wait for the obstacle to be removed.

Figure 4.1 shows the flowchart of the program. The figure shows how the program should be designed. After importing all libraries and setting GPIO pins as inputs and outputs we move to the infinite loop of the program.

Once in the loop, the priority check is the obstacle which the code will perform first before making any movement. When the obstacle sensor does not detect the line, the code then checks the type of movement needed and then executes it. Even if the type of movement does not change, the obstacle sensor is still read. This makes sure that the robot does not have any collisions.

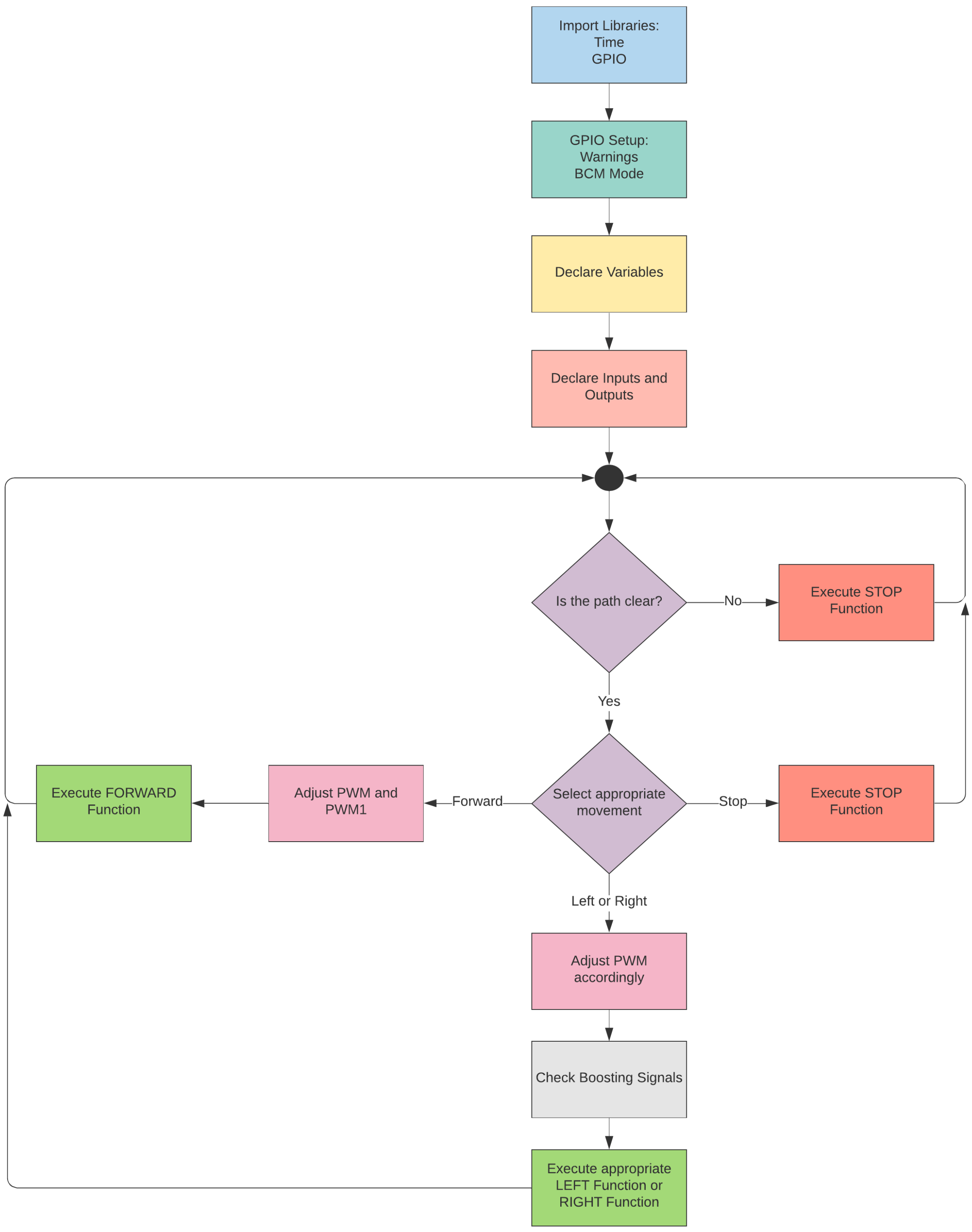


Figure 4.1 – Operational Flowchart

4.2.1 Importing necessary libraries and modes

The necessary libraries used in this project include time and GPIO. The GPIO pins need to be set to BOARDCOM mode as we use the names of the pins rather than its position. All GPIO warning also need to be disabled. The following code extract is used to do this:

```
import RPi.GPIO as GPIO   #Importing Raspberry Pi GPIO Pins
import time               #Importing time module
GPIO.setwarnings(False)   #Disabling GPIO warnings
GPIO.setmode(GPIO.BCM)    #Using GPIO name throughout code
```

4.2.2 Declaring Variables

Declaring variables is the most efficient way for ease of programming. Variables included in this code include all inputs, outputs and delays. Variables such as PWM and delays will be used throughout the loop and functions. Each sensor connected to a GPIO is stored as a variable for ease of programming. The following code extract was used to do this:

```
PWM = 0             #Variable throughout code
PWM1=0              #Variable for forward motion
delay=0.01          #Delay
delay1=0.35         #Delay for Sharp Turns

LL = 14             #Left Sensor Left Side
LC = 15             #Left Sensor Center Side
LR = 18             #Left Sensor Right Side
RL = 8              #Right Sensor Left Side
RC = 7              #Right Sensor Center Side
RR = 1              #Right Sensor Right Side
OB1 = 17            #Obstacle Sensor 1
OB2 = 22            #Obstacle Sensor 2
```

4.2.3 *Declaring Inputs*

All inputs for the code are declared in the initialization stage. The 6 line tracking sensors and the 2 obstacle sensors are declared and set as inputs. The following code extract is used to do this:

```
GPIO.setup(OB1, GPIO.IN)  #OB1 as Input
GPIO.setup(OB2, GPIO.IN)  #OB2 as Input
GPIO.setup(LL, GPIO.IN)   #LL as Input
GPIO.setup(LC, GPIO.IN)   #LC as Input
GPIO.setup(LR, GPIO.IN)   #LR as Input
GPIO.setup(RL, GPIO.IN)   #RL as Input
GPIO.setup(RC, GPIO.IN)   #RC as Input
GPIO.setup(RR, GPIO.IN)   #RR as Input
```

4.2.4 *Declaring PWM Outputs*

Four GPIO pins are used as PWM outputs which are wired to the driver circuit. All PWM outputs have a frequency of 50Hz and start at 0% duty cycle. At 50Hz the average signal supplied is smooth as opposed to a lower frequency where the on and off switching of the PWM can be witnessed as the motors stop and start repeatedly. The duty cycle will vary depending on the movement of the robot. The following code extract is used to do this:

```
GPIO.setup(5,GPIO.OUT)     # GPIO 5 as output (Left Motor -)
my_pwm1=GPIO.PWM(5,50)     # Setting GPIO 5 as PWM Signal 50Hz
my_pwm1.start(0)           # GPIO 5 PWM start value = 0

GPIO.setup(6,GPIO.OUT)     # GPIO 6 as output (Left Motor +)
my_pwm2=GPIO.PWM(6,50)     # Setting GPIO 6 as PWM Signal 50Hz
my_pwm2.start(0)           # GPIO 6 PWM start value = 0

GPIO.setup(20,GPIO.OUT)    # GPIO 20 as output (Right Motor -)
my_pwm3=GPIO.PWM(20,50)    # Setting GPIO 20 as PWM Signal 50Hz
my_pwm3.start(0)           # GPIO 20 PWM start value = 0

GPIO.setup(21,GPIO.OUT)    # GPIO 21 as output (Right Motor +)
my_pwm4=GPIO.PWM(21,50)    # Setting GPIO 21 as PWM Signal 50Hz
my_pwm4.start(0)           # GPIO 21 PWM start value = 0
```

4.2.5 Defining Functions

For the robot to perform its duties fully it needed 4 types of movement. These movements are stop, forward, left and right. These movements are used throughout the infinite loop by checking the type of movement needed, then adjusting the PWM signals accordingly and then executing the function. The ‘PWM’ and ‘PWM1’ variables are used in the functions. The following code extract defines these four functions:

```
def stop():                     # Defining the STOP Function
     my_pwm1.ChangeDutyCycle(0)  # Supply 0% duty cycle to GPIO5
     my_pwm2.ChangeDutyCycle(0)  # Supply 0% duty cycle to GPIO6
     my_pwm3.ChangeDutyCycle(0)  # Supply 0% duty cycle to GPIO20
     my_pwm4.ChangeDutyCycle(0)  # Supply 0% duty cycle to GPIO21

def right():                    # Defining the RIGHT Function
    my_pwm1.ChangeDutyCycle(0)   # Supply 0% duty cycle to GPIO5
    my_pwm2.ChangeDutyCycle(PWM) # Supply ‘PWM’ duty cycle to GPIO6
    my_pwm3.ChangeDutyCycle(PWM) # Supply ‘PWM’ duty cycle to GPIO20
    my_pwm4.ChangeDutyCycle(0)   # Supply 100% duty cycle to GPIO21

def left():                     # Defining the LEFT Function
    my_pwm1.ChangeDutyCycle(PWM) # Supply ‘PWM’ duty cycle to GPIO5
    my_pwm2.ChangeDutyCycle(0)   # Supply 0% duty cycle to GPIO6
    my_pwm3.ChangeDutyCycle(0)   # Supply 0% duty cycle to GPIO20
    my_pwm4.ChangeDutyCycle(PWM) # Supply ‘PWM’ duty cycle to GPIO21

def forward():                  # Defining the FORWARD Function
    my_pwm1.ChangeDutyCycle(0)   # Supply 0% duty cycle to GPIO5
    my_pwm2.ChangeDutyCycle(PWM1) #Supply ‘PWM1’ duty cycle to GPIO6
    my_pwm3.ChangeDutyCycle(0)   # Supply 0% duty cycle to GPIO20
    my_pwm4.ChangeDutyCycle(PWM) # Supply ‘PWM’ duty cycle to GPIO21
```

4.2.6 Obstacle Sensing

Reading the obstacle sensors is the first step when entering the infinite loop. The obstacles sensors will have to be read constantly, and only if there is no obstacle in its path then it shall allow the movement of the robot. Using the 'if', 'or' and 'else' the check is conducted. If any of the obstacle sensors produce a digital 'False' the robot stops, otherwise check for movement needed. The following code extract show how the obstacle sensors are read before allowing any movement.

```
while 1:

    if (GPIO.input(OB1)==False or GPIO.input(OB2)==False):
         stop() #Using STOP Function

    else:
         #Check for Movement
```

4.2.7 Left and Right Movement

The line tracking sensors are read only when the obstacle sensors permit its movement. Using ***Table 2.1*** and ***Table 2.2***, the conditions for each movement are checked as per ***Table 2.1*** and the correct PWM signals are chosen. However, the booster signals are also checked once the movement is selected. If the boosted signal is needed, the PWM variable is adjusted and then the movement is executed. A sharp turn is when the signal applied is greater than 80% duty cycle. The following code extract shows how this done for one type of movement:

```
if(GPIO.input(LC)==False and GPIO.input(LR)==False and GPIO.input(RL)==False
and GPIO.input(RC)==True):
            PWM= 90                     #Selecting 90% Duty Cycle
                                        #Inputs indicate a left turn
            if(GPIO.input(LL)==False): #Check booster signal
                PWM = PWM + 10          #Booster Signal needed – add 10%
            else:
                PWM=PWM #Booster signal not needed – PWM=PWM
            left()
            time.sleep(delay1)
```

This function is repeated for the 5 types of left movement and the 5 types of right movement. The varying PWM signal will allow the robot to move smoother and faster. We use the delay for sharp turns for the robot which allows it to turn fully before accepting a condition. If there is no delay for sharp turns, it was found that the robot oscillates around the black line.

4.2.8 Forward Movement

Checking whether a forward movement is needed is slightly different compared to left and right movements because of the booster signals. Figure 2.1 describes how an initial forward movement is selected but is changed to a left or right movement because of the booster signals. Although the four center sensors are dictating a forward movement we see in the figure that this can be incorrect for the conditions shown in ***Figure 4.2***.

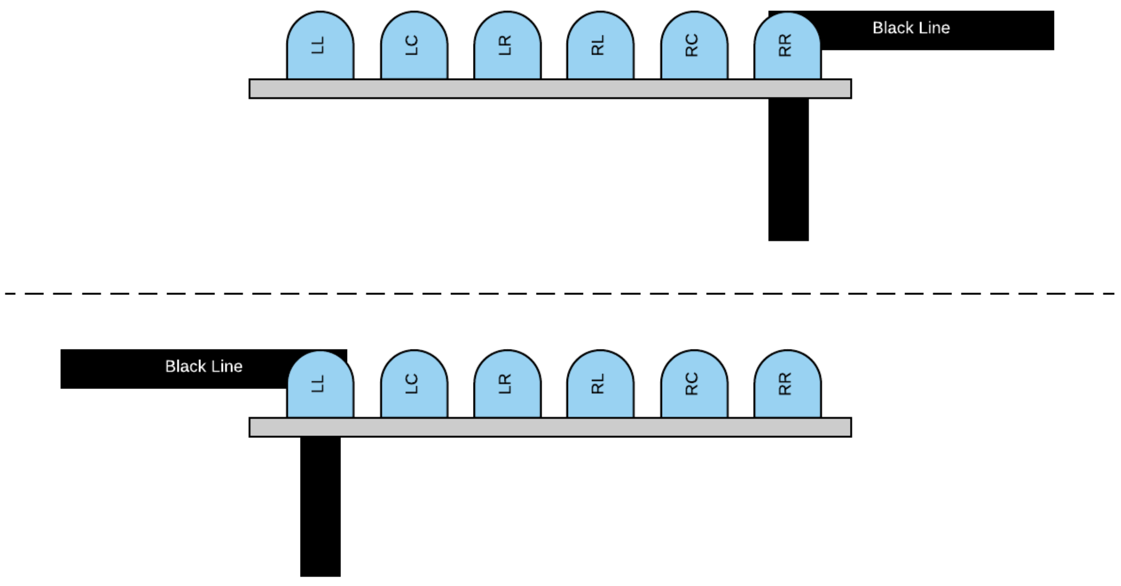


Figure 4.2 – Special Cases

As seen in the figure the initial condition selected by the code will be forward for both conditions shown above. Therefore, in order to select the correct movement 2 additional checks will need to be done. Checking whether 'LL' sees the line and whether 'RR' does not see the line will result in a sharp left movement. Checking whether 'RR' sees the line and whether 'LL' does not see the line will result in a sharp right movement. If both these conditions are not true, then the robot continues to move with the initial condition which is forward. The forward movement has custom PWM signals in order to make the robot move forward in a straight line. The following code extract is used to do this:

```
elif(GPIO.input(LC)==True      and      GPIO.input(LR)==True      and
GPIO.input(RL)==True and GPIO.input(RC)==True):
 PWM = 75
 if(GPIO.input(LL)==False and GPIO.input(RR)==True):
     PWM = 100
     left()
     time.sleep(delay1)
 elif(GPIO.input(RR)==False and GPIO.input(LL)==True):
     PWM = 100
     right()
     time.sleep(delay1)
 else:
     PWM=45
     PWM1=65
     forward()
```

4.2.9 Stop Movement

The stop movement is needed to stop the robot, and this done by drawing a perpendicular black line across the track's path. When all four center sensors (LC, LR, RL, RC) detect the black line the robot is instructed to stop using the stop function. The following code extract is used to do this:

```
elif(GPIO.input(LC)==False and GPIO.input(LR)==False and
GPIO.input(RL)==False and GPIO.input(RC)==False):

 PWM = 0

 stop()
```

4.3 Blynk Python Code

The Blynk IoT application is used as the HMI of the project. The Blynk IoT application will have to communicate with the Raspberry Pi via Wi-Fi and visually represent how the robot is performing. The Blynk IoT application uses a token that communicates with a Blynk server which enables the communication between an iOS device and the Raspberry Pi. A limitation with Blynk is that it needs a short delay to execute the communication. ***Figure 4.3*** shows the flowchart used to design the Blynk application.

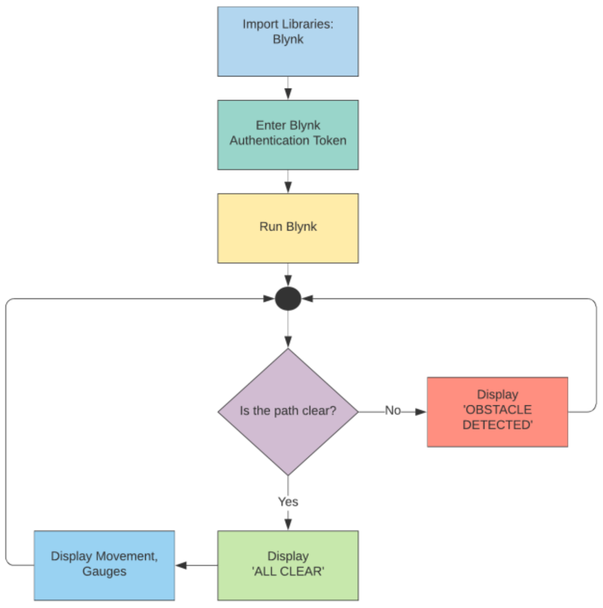


Figure 4.3 – Blynk IoT Flowchart

For this project, virtual pins were used to communicate. Virtual pins allow communication without needing physical connections. ***Table 4.1*** shows the tools that were used to create the HMI:

Tool Used:	**Used to display:**	**Pin**
Labelled Value	Display types of movement	Virtual Pin 3
Labelled Value	Display path clearance	Virtual Pin 4
LED	Obstacle Sensing	Virtual Pin 9
Horizontal Level	To indicate sharpness of turns	Virtual Pin 1 (Left) and Virtual Pin 2 (Right)
Gauge	Shows PWM Signals	Virtual Pins 5,6,7,8
Superchart	Plot graphs of virtual pins	N/A

Table 4.1 *– Blynk Tools Used*

4.3.1 Initializing Blynk

For Blynk to connect to the iOS device, a matching token is needed. The Blynk libraries also need to be imported in the initialization stage of the program. The following code extract shows how Blynk is initialized in the program:

```
import BlynkLib            #Importing the Blynk Library

blynk = BlynkLib.Blynk('ja7MnU7v8fQGAlvXCMUs8FMfRp3CsnOH')
                           #Authentication Token

blynk.run()                #Start blynk
                           #Delay to allow Blynk to connect
time.sleep(4)              #Wait for connection
```

4.3.2 Type of Movement

To indicate the type of movement of the robot we use a ‘labelled value’. The tool is connected to virtual pin blank. It will display a worded movement e.g. ‘STOP’. This tool will need to be added to all movement functions in the code in order to display the movement correctly. The following code extract is added to the relevant functions:

```
      def forward():
            blynk.virtual_write(3,'FORWARD')

      def left():
            blynk.virtual_write(3,'LEFT')

      def right():
            blynk.virtual_write(3,’RIGHT’)

      def stop():
            blynk.virtual_write(3,’STOP’)
```

4.3.3 Indicating Obstacles

To indicate to indicate whether an obstacle is in a robot's path, LED's and a labelled value was used. The labelled value displays 'ALL CLEAR' or 'OBSTACLE DETECTED' depending on the state of the sensors. The LEDs will remain off unless an obstacle is detected. In that case the LEDs will turn on. The data is also sent to a superchart where it can be exported into a spreadsheet. The following code extract is used to do this:

```
if (GPIO.input(OB1)==False or GPIO.input(OB2)==False):
         stop()
         blynk.virtual_write(4,'Obstacle Detected')
         blynk.virtual_write(9,255)

    else:
         blynk.virtual_write(4,'All Clear')
         blynk.virtual_write(9,0)
```

4.3.4 Indicating Sharpness of Turns

We use a horizontal level tool to indicate the sharpness of a turn. The higher the PWM signal when turning, the sharper the turn. The horizontal level will fill up to a maximum of 100% when displaying the turn. For forward and stop movements the level indicators will display 0%. The following code extract is added to the relevant functions:

```
        def forward():
              blynk.virtual_write(1,0)
              blynk.virtual_write(2,0)

        def left():
              blynk.virtual_write(1,PWM)
              blynk.virtual_write(2,0)

        def right():
              blynk.virtual_write(1,0)
              blynk.virtual_write(2,PWM)

        def stop():
              blynk.virtual_write(1,0)
              blynk.virtual_write(2,0)
```

4.3.5 Displaying Terminal Duty Cycles

There are a total of 4 terminals where duty cycles are applied, these are the two terminals on each motor. The gauge tool is used to show the total duty cycle applied to each terminal. Each terminal has its own gauge. The following code extract is added to the relevant function in order to this:

```
def forward():
      blynk.virtual_write(5,PWM1)
      blynk.virtual_write(6,PWM)
      blynk.virtual_write(7,0)
      blynk.virtual_write(8,0)

def left():
      blynk.virtual_write(5,PWM)
      blynk.virtual_write(6,0)
      blynk.virtual_write(7,0)
      blynk.virtual_write(8,PWM)

def right():
      blynk.virtual_write(5,0)
      blynk.virtual_write(6,PWM)
      blynk.virtual_write(7,PWM)
      blynk.virtual_write(8,0)

def stop():
      blynk.virtual_write(5,0)
      blynk.virtual_write(6,0)
      blynk.virtual_write(7,0)
      blynk.virtual_write(8,0)
```

4.4 HMI Setup

The Blynk application allows the user to design a custom HMI for their on application using a drag and drop method. Once the selected tool is dropped onto the HMI its needs to setup with the appropriate headings and corresponding pin numbers. Aspects such as colour and text size are also adjusted via the application. ***Figure 4.4*** shows the complete HMI. The following describes how each tool is setup.

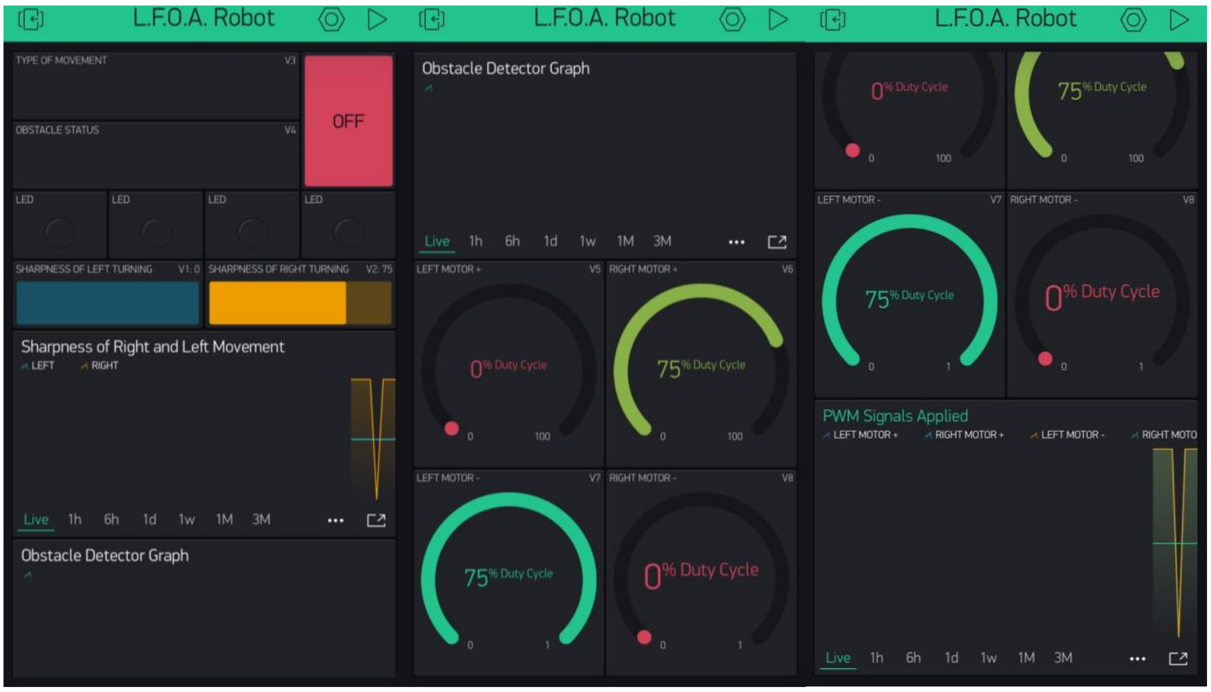


Figure 4.4 – Robot HMI

4.4.1 Type of Movement Display

For the 'Type of Movement' display, the labelled value tool was used. The title of the display set to 'Type of Movement' and connected to virtual pin 3. The code will write text to this tool. The 'Refresh Interval' is set to 'Push' which means it will update the tool whenever a new value is available.

4.4.2 Obstacle Status Display

The 'Obstacle Status' display is setup similarly to the 'Type of Movement' display with the only change being that the 'Obstacle Status' Display is connected to virtual pin 4.

4.4.3 Obstacle LED Display

A total of 4 LEDs was used all connected to virtual pin 9 so that all will turn on by adjusting only one value. The colour of the LEDs was set to red to indicate a warning. These LEDs are correlated to the 'Obstacle Status' display.

4.4.4 Sharpness of Turns Display

For the 'Sharpness of Turns' display, 2 horizontal level tools were used. The left level tool is connected to virtual pin 1 and the right level tool is connected to virtual pin 2. The values for the level indicators are set from 0-100. The sharper the turn of the robot, the level tends further to the maximum of 100.

4.4.5 Gauges Display

The 'Gauges' display is used to show the duty cycle applied to each motor terminal. The gauges are coloured with a gradient from green to red. The higher the duty cycle applied the redder the gauge turns. This is an efficient way of displaying the duty cycle as the colours are also an indicator of how hard the motors are working. The refresh interval for the gauges are set to 'Push'. The program will only transmit numbers for the gauges. Therefore, a suffix added is '% Duty Cycle'. The gauges also range from 0 – 100.

- Left Motor Positive Terminal – Virtual Pin 5
- Left Motor Negative Terminal – Virtual Pin 7
- Right Motor Positive Terminal – Virtual Pin 6
- Right Motor Negative Terminal – Virtual Pin 8

4.4.6 Supercharts

Supercharts are used to graph the data and export it for later use. Supercharts can track live data as well store data for up to one year. The LEDs are set to a superchart to show when there was an obstacle in the path of the robot. The Sharpness of Turns displays are superimposed on a separate superchart to show what type of turns are made by the robot. All 4 gauges are superimposed on another superchart to show the duty cycles and how hard each terminal is working.

CHAPTER 5

In this chapter, we test the project for its ability to follow the line, avoid obstacles as well as test consistency. There will be 4 tests conducted using the same track. For test 1 the robot will follow the laid-out track and for test 2 it will follow the same track in a reverse direction. For test 3 the track from test 1 is used but 3 obstacles are placed on the track to check the obstacle avoidance of the robot.

To summarize the performance of the robot we use time and speed. Using the fastest time and slowest time as a range, we will calculate average times and speeds. Standard deviation will also be calculated to show how far spread out each measurement is from the average.

To calculate the average time, we use:

$$Average\ Time = \frac{Sum\ of\ the\ time\ measurements}{Number\ of\ measurements}$$

[5-1]

To calculate the average speed, we use:

$$Average\ Speed = \frac{Distance}{Total\ Time}$$

[5-2]

To calculate the standard deviation, we will subtract the average from each measurement and square the result. Then find the average of the squared differences and square root the result. This is given by the equation:

$$SD = \sqrt{\frac{\sum(x-\overline{x})^2}{n}}$$

[5-3]

5.1 Test 1 Analysis

Figure 5.1 shows the track used for test 1. The test will be run 10 times in order to check operation and consistency. The times for each run will be recorded and analysed. A measurement is taken only once the robot stops on the start/finish line. The track has a total distance of 5,33m (533cm).

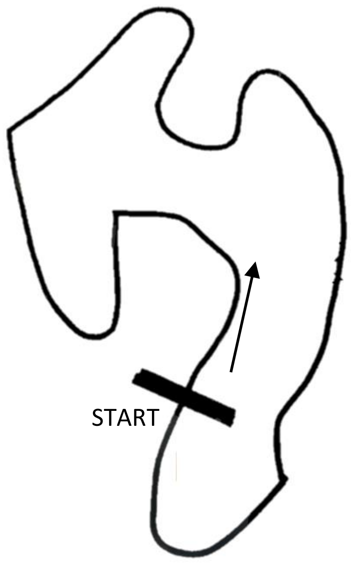


Figure 5.1 – Test Track 1

Table 5.1 shows the times for each run on Test track 1 with the fastest time highlighted in green and the slowest time highlighted in red:

Run Number	**Time Taken (seconds)**	**Calculated Average Speed (m/s)**
1	37,27	0.143
2	36,66	0.145
3	37,09	0.144
4	36,60	0.146
5	36,23	0.147
6	36,57	0.146
7	35,42	0.150
8	35,59	0.150
9	36,52	0.146
10	35,02	0.152

Table 5.1 – Test 1 Results

From ***Table 5.1*** we can calculate the following:

- Average Time = 36,30 seconds
- Standard Deviation = 0,73 seconds
- Average Speed = 0,147 m/s

Figure 5.2 shows the time results on a graph. ***Figure 5.3*** shows the speed results on a graph

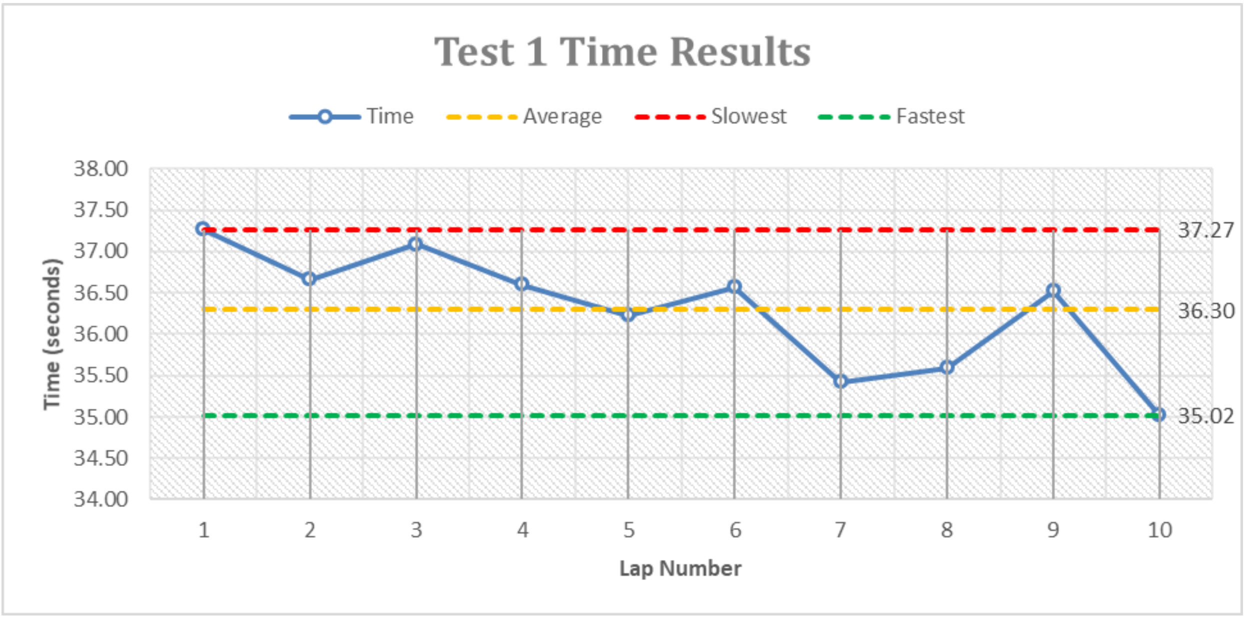


Figure 5.2 – Graph of Test 1 Time Results

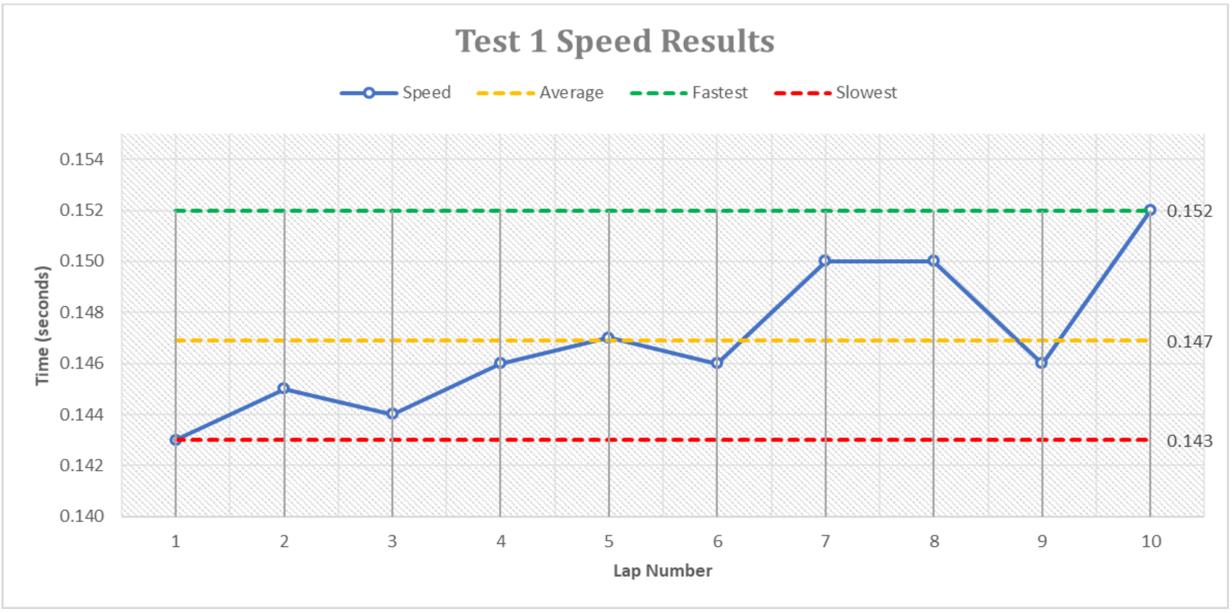


Figure 5.3 – Graph of Test 1 Speed Results

5.2 Test 2 Analysis

Figure 5.4 shows the track used for test 2. The test 2 track is the same as the test1 track but in a reverse direction. The test will be run 10 times in order to check operation and consistency. The times for each run will be recorded and analysed. A measurement is taken only once the robot stops on the start/finish line. The track has a total distance of 5,33m (533cm).

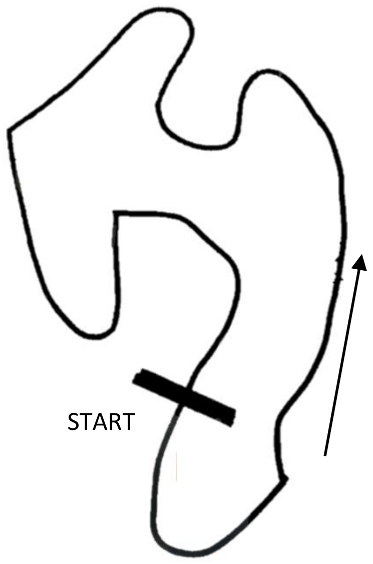


Figure 5.4 – Test Track 2

Table 5.2 shows the times for each run on Test track 2:

Run Number	**Time Taken (seconds)**	**Calculated Average Speed (m/s)**
1	35,52	0.150
2	34,90	0.153
3	35,39	0.151
4	37,19	0.143
5	37,42	0.142
6	37,50	0.142
7	37,80	0.141
8	37,07	0.144
9	36,62	0.146
10	38,20	0.140

Table 5.2 – Test 2 Results

From ***Table 5.2*** we can calculate the following:

- Average Time = 36,76 seconds
- Standard Deviation = 1,12 seconds
- Average Speed = 0,145 m/s

Figure 5.5 shows the time results on a graph. ***Figure 5.6*** shows the speed results on a graph.

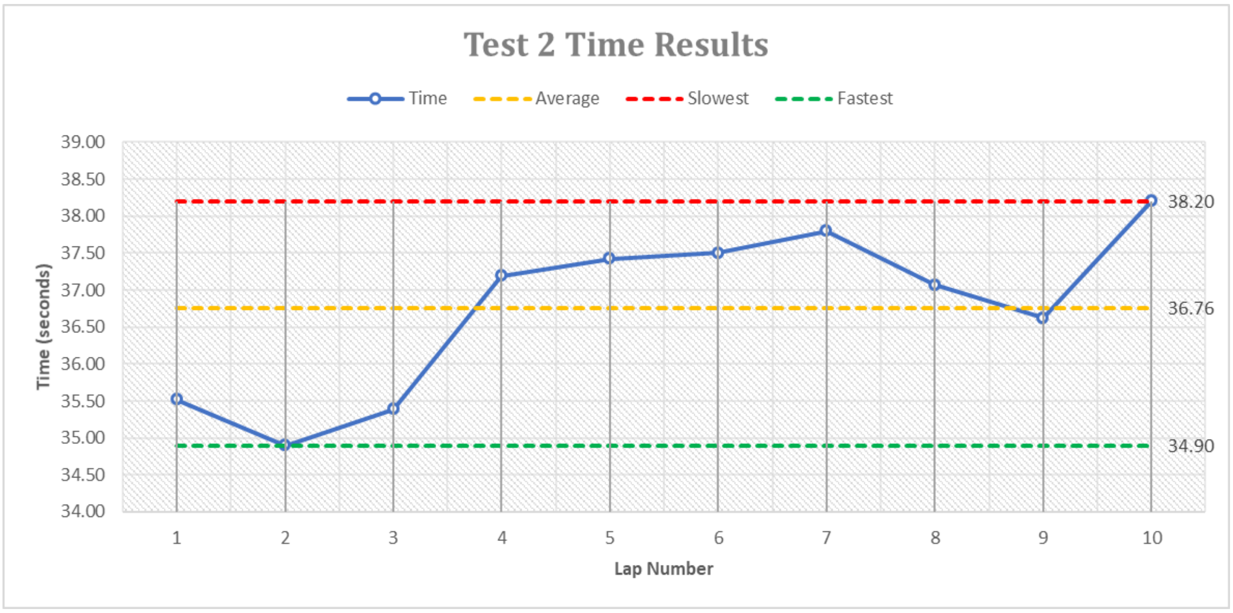


Figure 5.5 – Graph of Test 2 Time Results

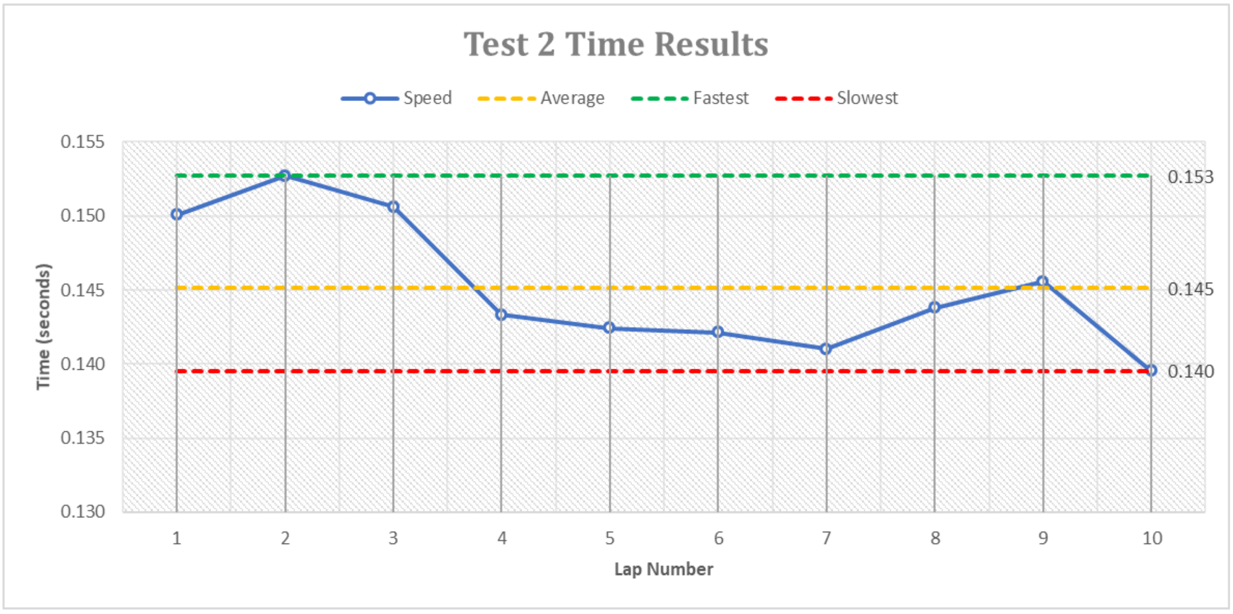


Figure 5.6 – Graph of Test 2 Speed Results

5.3 Test 3 Analysis

For test is designed to test the obstacle avoidance of the robot. For test 3, the test track from test 1 is used with 3 obstacles placed on the track. The robot needs to stop before contacting the obstacles. The test is run for a total of 5 laps and the results are recorded. ***Figure 5.7*** shows the test track with the position of the obstacles.

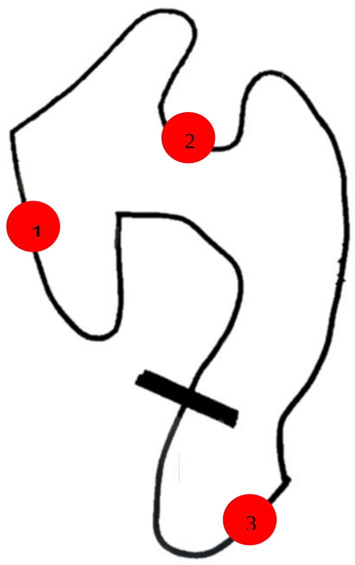


Figure 5.7 – Test track 1 with obstacles

Table 5.3 shows the results of test 3

Lap Number	***Obstacle 1***	***Obstacle 2***	***Obstacle 3***
1	Avoided	Avoided	Avoided
2	Avoided	Avoided	Avoided
3	Avoided	Collision	Avoided
4	Avoided	Avoided	Avoided
5	Avoided	Avoided	Avoided

Table 5.3 – Test 3 Results

On Lap 3, the robot collided with the obstacle 2 before stopping. The robot was making a sharp turn when the line tracking sensors collided with the obstacle before the obstacle sensors could change its state. Repositioning and re calibrating of the sensors corrected this.

CHAPTER 6

6.1 Conclusion

The main aim of this control system was to make the robot drive along a black line and to avoid any obstacles along its path. Using many line tracking sensors, the PWM signals applied to the motors could be manipulated so that the robot may make a variety of turns. The interface must be linked to the system to allow all important information to be supplied to the user of the system. This system was intended to work with no human contact. Correct planning and hardware wiring were needed to make the robot small and compact. Writing the code in Python correctly enabled the robot to fulfil its purpose as well as communicate to the mobile application. The decision to implement a mobile application using IoT proved to be efficient in making the system more user friendly and attractive to anyone operating this small application.

To conclude the key findings are as follows:

- The Funduino 3-way channel IR sensors is easily compatible with the Raspberry Pi GPIO Pins. The use of many sensors (6) allowed the manipulation of the PWM Signals for better control and a variety of turns.

- The FC-51 IR Sensor is reliable and effective when it comes to detecting a obstacle and is easily compatible with the Raspberry Pi. However, the distance is set manually and does not work in very bright environments.

- The motors, mounted to the rear, can be manipulated to control the direction of the robot using different PWM signals without any physical turning. Turning the wheels in an opposing direction can result in a left or right movement.

- The mobile application captures live data of the sensors and motors. This data can be back tracked for up to 1 year.

- Although the biggest challenge of the project was to learn a new programming language, the Raspberry Pi makes it simple to use when running the Raspbian OS.

REFERENCES:

1. History of the autonomous car, available at https://www.titlemax.com/resources/history-of-the-autonomous-car/ [Accessed 14 May 2020]

2. Figure 2.2 – PWM Operation
   https://circuitdigest.com/tutorial/what-is-pwm-pulse-width-modulation [Accessed July 2020]

3. Python (programming language) , available at:
   https://en.wikipedia.org/wiki/Python_(programming_language) [Accessed March 2020]

4. Raspberry Pi Model 3B+ Specifications:
   https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

5. Raspberry GPIO Configuration, available at:
   https://www.raspberrypi.org/documentation/usage/gpio/ [Accessed March 2020]

6. Funduino 3-Channel IR Sensor
   http://www.alselectro.com/3-way-ir-track-tcrt5000.html [Accessed July 2020]

7. FC – 51 Obstacle Sensor
   https://surtrtech.com/2018/01/27/how-to-use-the-fc-51-infrared-proximity-obstacle-avoidance-sensor-with-arduino/ [Accessed July 2020]

8. How to install Raspbian on Raspberry Pi
   https://thepi.io/how-to-install-noobs-on-the-raspberry-pi/ [Accessed March 2020]

9. Blynk – How it works
   http://docs.blynk.cc/#intro-how-blynk-works [Accessed April 2020]

10. Blynk – Create new project
    http://docs.blynk.cc/#getting-started-getting-started-with-the-blynk-app-2-create-a-new-project [Accessed April 2020]

11. Install Blynk on Raspberry Pi:

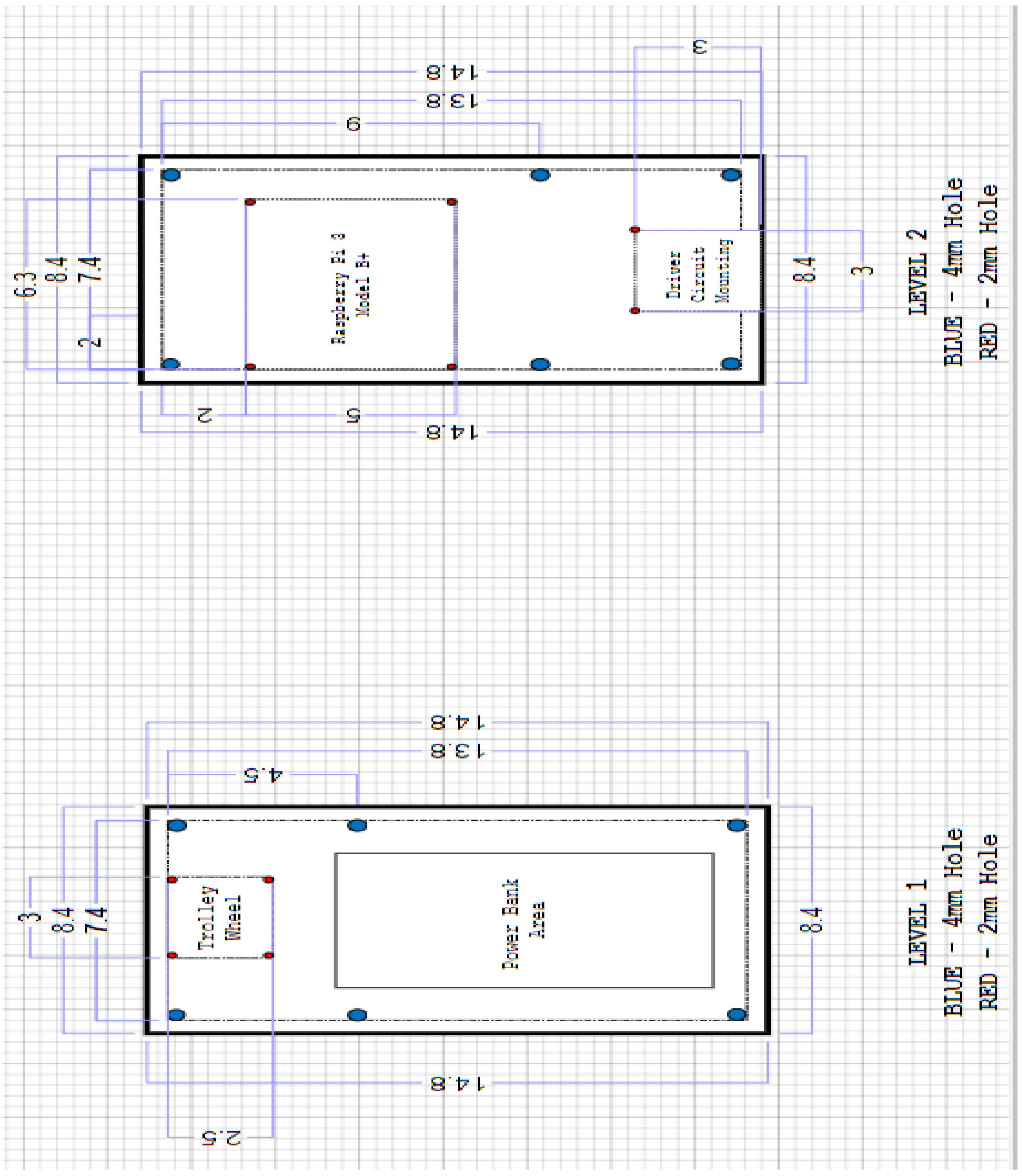
    https://www.bc-robotics.com/tutorials/getting-started-blynk-raspberry-pi-3/

    [Accessed April 2020]

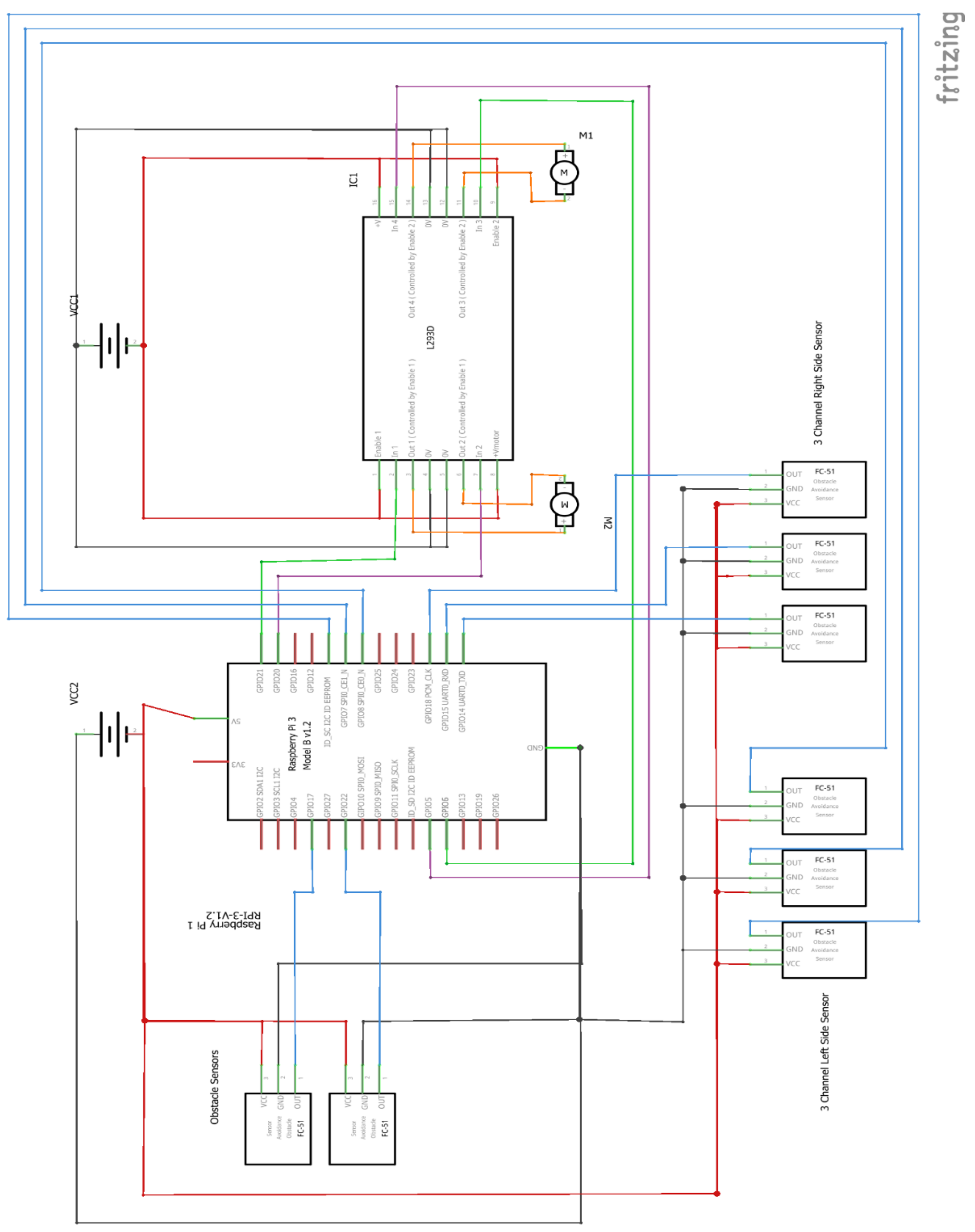
12. Python Introduction

    https://www.w3schools.com/python/python_intro.asp [Accessed April 2020]

Annexure A: Structrual Layout



Annexure B: Full Circuit Diagram



Annexure C: Bill of Materials

BILL OF MATERIALS

PROJECT:	LFOA Robot
STUDENT NUMBER:	21355491
NAME:	Preshen Govender
MODULE:	Industrial Project 4
COURSE:	BTech Electronic Engineering (L/C) INSTR
UNIVERSITY:	Durban University of Technology

COMPONENT	QTY	SOURCE	COST PER UNIT		TAX	TOTAL	
Wheel with 3VDC Motor	2	Mantech	ZAR	35.00	15%	ZAR	80.50
L293DNE	1	DIY Electronics	ZAR	56.00	15%	ZAR	64.40
IC Socket	1	Mantech	ZAR	6.60	15%	ZAR	7.59
Raspberry Pi 3 Model B+	1	DIY Electronics	ZAR	521.70	15%	ZAR	599.96
FC-51 Sensor	2	DIY Electronics	ZAR	19.09	15%	ZAR	43.91
Funduino 3-way channel IR Sensors	2	Mantech	ZAR	50.41	15%	ZAR	115.94
Jumper Wire x 30	1	Mantech	ZAR	28.57	15%	ZAR	32.86
Veroboard	1	Mantech	ZAR	27.57	15%	ZAR	31.71
Perspex	2	Builders Warehouse	ZAR	80.00	0%	ZAR	160.00
4mm x 50mm Bolt	4	Desai's Hardware	ZAR	2.50	0%	ZAR	10.00
4mm Nut	12	Desai's Hardware	ZAR	2.00	0%	ZAR	24.00
2mm x 10mm Bolt	2	Desai's Hardware	ZAR	2.50	0%	ZAR	5.00
2mm x 20mm Bolt	4	Desai's Hardware	ZAR	2.00	0%	ZAR	8.00
2mm Nut	6	Desai's Hardware	ZAR	2.00	0%	ZAR	12.00
Trolley Wheel with fittings	1	DIY Electronics	ZAR	34.00	0%	ZAR	34.00
Power Bank	1	Takealot	ZAR	120.00	0%	ZAR	120.00
Mico SD card with Adapter	1	DIY Electronics	ZAR	68.70	15%	ZAR	79.01
Headers 24 Pack	1	DIY Electronics	ZAR	3.26	15%	ZAR	3.75
TOTAL						**ZAR**	**1,432.61**

Annexure D: Full Python Code

```
import BlynkLib
import RPi.GPIO as GPIO # Importing Raspberry Pi GPIO Pins
import time # Importing time module
GPIO.setwarnings(False) # Disabling GPIO warnings
GPIO.setmode(GPIO.BCM) # Using GPIO Name

blynk = BlynkLib.Blynk('ja7MnU7v8fQGAlvXCMUs8FMfRp3CsnOH')
blynk.run()
time.sleep(4)

PWM = 0
PWM1=0
delay=0.01
delay1=0.3

#set GPIO Pins

LL = 14
LC = 15
LR = 18
RL = 25
RC = 7
RR = 1
OB1=17
OB2=22

GPIO.setup(OB1, GPIO.IN)
GPIO.setup(OB2, GPIO.IN)
GPIO.setup(LL, GPIO.IN)
GPIO.setup(LC, GPIO.IN)
GPIO.setup(LR, GPIO.IN)
GPIO.setup(RL, GPIO.IN)
GPIO.setup(RC, GPIO.IN)
GPIO.setup(RR, GPIO.IN)
GPIO.setup(5,GPIO.OUT) # Setting GPIO 5 as output (Input signal for
L293DNE - Left Motor +)
my_pwm1=GPIO.PWM(5,50) # Setting GPIO 5 as PWM Signal 50Hz
my_pwm1.start(0) # GPIO 5 PWM start value = 0

GPIO.setup(6,GPIO.OUT) # Setting GPIO 6 as output (Input signal for
L293DNE - Left Motor +)
my_pwm2=GPIO.PWM(6,50) # Setting GPIO 6 as PWM Signal 50Hz
my_pwm2.start(0) # GPIO 6 PWM start value = 0

GPIO.setup(20,GPIO.OUT) # Setting GPIO 20 as output (Input signal
for L293DNE - Right Motor -)
my_pwm3=GPIO.PWM(20,50) # Setting GPIO 20 as PWM Signal 50Hz
my_pwm3.start(0) # GPIO 20 PWM start value = 0

GPIO.setup(21,GPIO.OUT) # Setting GPIO 21 as output (Input signal
for L293DNE - Right Motor +)
my_pwm4=GPIO.PWM(21,50) # Setting GPIO 21 as PWM Signal 50Hz
my_pwm4.start(0) ## GPIO 21 PWM start value = 0



def forward():
    my_pwm1.ChangeDutyCycle(0) # Supply duty cycle to GPIO 5
    my_pwm2.ChangeDutyCycle(PWM1) # Supply duty cycle to GPIO 6
    my_pwm3.ChangeDutyCycle(0) # Supply duty cycle to GPIO 20
    my_pwm4.ChangeDutyCycle(PWM) # Supply duty cycle to GPIO 21

    blynk.virtual_write(5,PWM1)
    blynk.virtual_write(6,PWM)
    blynk.virtual_write(7,0)
    blynk.virtual_write(8,0)
    blynk.virtual_write(3,'Forward')
    blynk.virtual_write(1,0)
    blynk.virtual_write(2,0)
    time.sleep(0.05)
```

```
def left():
    my_pwm1.ChangeDutyCycle(PWM) # Supply duty cycle to GPIO 5
    my_pwm2.ChangeDutyCycle(0) # Supply duty cycle to GPIO 6
    my_pwm3.ChangeDutyCycle(0) # Supply duty cycle to GPIO 20
    my_pwm4.ChangeDutyCycle(PWM) # Supply duty cycle to GPIO 21

    blynk.virtual_write(5,PWM)
    blynk.virtual_write(6,0)
    blynk.virtual_write(7,0)
    blynk.virtual_write(8,PWM)
    blynk.virtual_write(3,'Left')
    blynk.virtual_write(1,PWM)
    blynk.virtual_write(2,0)
    time.sleep(0.05)

def right():
    my_pwm1.ChangeDutyCycle(0) # Supply duty cycle to GPIO 5
    my_pwm2.ChangeDutyCycle(PWM) # Supply duty cycle to GPIO 6
    my_pwm3.ChangeDutyCycle(PWM) # Supply duty cycle to GPIO 20
    my_pwm4.ChangeDutyCycle(0) # Supply  duty cycle to GPIO 21

    blynk.virtual_write(5,0)
    blynk.virtual_write(6,PWM)
    blynk.virtual_write(7,PWM)
    blynk.virtual_write(8,0)
    blynk.virtual_write(3,'Right')
    blynk.virtual_write(1,0)
    blynk.virtual_write(2,PWM)
    time.sleep(0.05)

def stop():
    my_pwm1.ChangeDutyCycle(0) # Supply duty cycle to GPIO 5
    my_pwm2.ChangeDutyCycle(0) # Supply duty cycle to GPIO 6
    my_pwm3.ChangeDutyCycle(0) # Supply duty cycle to GPIO 20
    my_pwm4.ChangeDutyCycle(0) # Supply duty cycle to GPIO 21

    blynk.virtual_write(5,0)
    blynk.virtual_write(6,0)
    blynk.virtual_write(7,0)
    blynk.virtual_write(8,0)
    blynk.virtual_write(3,'Stop')
    blynk.virtual_write(1,0)
    blynk.virtual_write(2,0)
    time.sleep(0.05)


while 1:

    if (GPIO.input(OB1)==False or GPIO.input(OB2)==False): #Stop
         stop()
         blynk.virtual_write(4,'Obstacle Detected')
         blynk.virtual_write(9,255)
         time.sleep(0.03)

    else: #Check for Movement
         blynk.virtual_write(4,'All Clear')
         blynk.virtual_write(9,0)
         time.sleep(0.03)

         if(GPIO.input(LC)==False and GPIO.input(LR)==False and
GPIO.input(RL)==False and GPIO.input(RC)==True):
            PWM= 90

            if(GPIO.input(LL)==False):
                PWM = PWM + 10
            else:
                PWM=PWM

            left()
            time.sleep(delay1)

         elif(GPIO.input(LC)==False and GPIO.input(LR)==False and
GPIO.input(RL)==True and GPIO.input(RC)==True):

            PWM = 80
            if(GPIO.input(LL)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            left()
            time.sleep(delay1)


         elif(GPIO.input(LC)==False and GPIO.input(LR)==True and
GPIO.input(RL)==True and GPIO.input(RC)==True):

            PWM = 80
            if(GPIO.input(LL)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            left()
            time.sleep(delay1)


         elif(GPIO.input(LC)==True and GPIO.input(LR)==False and
GPIO.input(RL)==False and GPIO.input(RC)==False):


            PWM = 90
            if(GPIO.input(RR)==False):
                PWM = PWM + 10

            else:
                PWM=PWM

            right()
            time.sleep(delay1)

         elif(GPIO.input(LC)==True and GPIO.input(LR)==False and
GPIO.input(RL)==True and GPIO.input(RC)==True):
```

```
            PWM = 70
            if(GPIO.input(LL)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            left()
            time.sleep(delay)

         elif(GPIO.input(LC)==True and GPIO.input(LR)==True and
GPIO.input(RL)==False and GPIO.input(RC)==False):

            PWM = 80
            if(GPIO.input(RR)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            right()
            time.sleep(delay1)

         elif(GPIO.input(LC)==True and GPIO.input(LR)==True and
GPIO.input(RL)==False and GPIO.input(RC)==True):

            PWM = 70
            if(GPIO.input(RR)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            right()
            time.sleep(delay)

         elif(GPIO.input(LC)==True and GPIO.input(LR)==True and
GPIO.input(RL)==True and GPIO.input(RC)==False):

            PWM = 80
            if(GPIO.input(RR)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            right()
            time.sleep(delay)

         elif(GPIO.input(LC)==False and GPIO.input(LR)==False and
GPIO.input(RL)==True and GPIO.input(RC)==False):

            PWM = 70
            if(GPIO.input(RR)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            left()
            time.sleep(delay)

         elif(GPIO.input(LC)==False and GPIO.input(LR)==True and
GPIO.input(RL)==False and GPIO.input(RC)==False):

            PWM = 70
            if(GPIO.input(RR)==False):
                PWM = PWM + 20
            else:
                PWM=PWM

            left()
            time.sleep(delay)



         elif(GPIO.input(LC)==True and GPIO.input(LR)==True and
GPIO.input(RL)==True and GPIO.input(RC)==True):

            PWM = 75
            if(GPIO.input(LL)==False and GPIO.input(RR)==True):
                PWM = 100
                left()
                time.sleep(delay1)
            elif(GPIO.input(RR)==False and GPIO.input(LL)==True):
                PWM = 100

                right()
                time.sleep(delay1)
            else:
                PWM=50
                PWM1=65

                forward()

         elif(GPIO.input(LC)==False and GPIO.input(LR)==False and
GPIO.input(RL)==False and GPIO.input(RC)==False):

            PWM = 0

            stop()

         elif(GPIO.input(LC)==True and GPIO.input(LR)==False and
GPIO.input(RL)==False and GPIO.input(RC)==True):

            PWM = 40
            PWM1=60

            forward()
            time.sleep(0.1)
            stop()



         else:

            stop()
```