Sudan University of Science and Technology

College of Engineering

Electrical Engineering

Proportional-Integral-Derivative Line Tracker Using Raspberry Pi

متتبع الخط التناسبي-التفاضلي-التكاملي بإستخدام الراسبيري باي

A Project Submitted in Partial Fulfillment for the Requirement of the Honor Degree of B.Sc in Control Electrical Engineering

Prepared By:

1. **Ahmed Al-Gasim Mohammed Ibrahim**
2. **Ahmed Mohammed Osman Elmubarak**
3. **Abdulaziz Abdulkhalig Awad Mohammed-Noor**
4. **Abdulla Bushra Abdulla Ali**

Supervised By:

Dr. AwadAllah Taifour Ali

October 2018

قال تعالى:

﴿يَرْفَعِ اللَّه الذِينَ آمَنُوا مِنكُمْ وَالذِينَ أُوتُوا الْعِلْمَ دَرَجَات﴾

(المجادلة:11)

صدق الله العظيم

Dedication

For my beloved mother, my dear father, my dear brothers and sisters, my friends who always supported me and stood by my side, my teachers and mentors, my respectful partners.

Acknowledgement

First and always, we thank Allah for his guidance and blessings, the reason behind our success.

We are grateful for our families, friends, colleges and teachers for their never ending support.

We thank Dr. AwadAllah Taifour Ali for his guidance and supervision.

In the memory of our beloved teacher and mentor:

Dr. Aamir Ahmed Dawood

God rest his soul

ABSTRACT

Controlling devices and systems are essential in modern times, especially when robots do important and dangerous jobs accurately. The main goal of this project is to make an economic control system, with high measures of safety, can be used by a normal person without any complications. Line Follower Robots are self-instructing robots, their main purpose is to follow a certain line and never go out of track. Usually, this line is a black colored line. The principle of those robots depends on the reflection of light for the two different surfaces. The reflection is measured through an infrared sensor. Then a controller processes this data to guide the robot on the right track. This robot solves a lot of problems like unnecessary-time wasting labor work in assembly lines, factories and mass storage facilities. The problem with this robot is it takes a long time to do it's required job, due to its uneven movement and tracking. And it goes out of bounds a lot, also it can't turn on tough corners. Solution to this problem is to enhance its input data, response, processing and structure. This concept was tested in various conditions and proved to be successful with satisfying results.

المستخلص

إن التحكم بالاجهزة و الأتظمة أصبح مجال إهتمام مؤخراَ, خصوصاَ التحكم في الروبوتات حيث انها تقوم بالاعمال الخطرة والمهمة والدقيقة. إن الهدف الاساسي لهذا المشروع هو عمل نظام تحكم باقل تكلفة ويضيف قدر عالي من الأمان ويمكن المستخدم العادي من استعماله بدون تعقيدات. روبوتات تتبع الخط هي روبوتات ذاتية الحركة، مهمتها الرئيسية تتبع الخط و عدم الخروج من المسار. عادةً ما يكون لون الخط أسود. مبدأ عمل هذه الروبوتات يعتمد على انعكاس الضوء بين سطحين مختلفي اللون. و يتم استشعار الإنعكاس في الضوء عن طريق أجهزة الإستشعار (متحسس ضوئي). و من ثم يتم معالجة قوة الإشارة و المقارنة بين القيم لتوجيه الروبوت و بقائه في المسار الصحيح دون الخروج عن الخط. يقوم هذا الروبوت بحل مشاكل الأيدي العاملة و زمن الإنتاج في خطوط المصانع والمخازن و خطوط التجميع. مشكلة هذا الربوت أنه يتأخر في القيام بالوظيفة المطلوبة، يستخدم طاقة اكبر نتيجة حركته الغير إنسيابية، و في بعض الاحيان يقوم بالخروج عن الخط المحدد، و لا يستطيع الإلتفاف بزوايا كبيرة. حل هذه المشكلة يكون بتحسين استجابة الروبوت عن طريق تحسين معطيات الدخل و طريقة المعالجة و بنية الروبوت. تم اختبار هذا النظام في حالات مختلفة وكانت النتائج مرضية محققةً الأهداف المرجوة من النظام.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

PID	Proportional Integral Derivative
P	Proportional
PI	Proportional Integral
PD	Proportional Derivative
AWD	All Wheel Drive
PWM	Pulse Width Modulation
I/O	Input Output
MCU	Micro-Controller Unit
RAM	Random Access Memory
EPROM	Erasable Programmable Read-Only Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
IoT	Internet of Things
ATM	Automated Teller Machine
UK	United Kingdom
BBC	British Broadcasting Corporation
SBC	Single Board Computer
HDMI	High Definition Multimedia Interface
OS	Operating System
CPU	Central Processing Unit
IDE	Integrated Development Environment
SoC	System-on-a-Chip
Wi-Fi	Wireless Fidelity
HAT	Hardware Attached on Top
RBG	Red Blue Green
USB	Universal Serial Bus
LED	Light Emitting Diode
IR	Infra-Red

mm	Millimeter
nm	Nanometer
K	Constant
PIXEL	Pi Improved Xwindows Environment Lightweight
LXDE	Pi Improved Xwindows Desktop Lightweight
SD	Storage Device
DC	Direct Current

CHAPTER ONE

INTRODUCTION

1.1 Overview

A robot is a machine that is capable of carrying out a complex series of actions and automatically executing tasks, especially one programmable by a computer. Robots can be utilized for a wide range of applications, and they offer many advantages like: task repetition, fast execution, precision, time and energy saving, immunity to hazards and intense labor.

A mobile robot is a robot capable of locomotion. It can move around in its environment or terrain and navigate its surroundings, it is not fixed to one physical location. Mobile robots are generally controlled by software, they use sensors and other gear to identify their surroundings and safely maneuver their obstacles. Mobile robots are a subfield of robotics that combine the progress in artificial intelligence with physical robotics.

A line tracker (or a line follower) is an autonomous mobile robot which have the capability to navigate an uncontrolled environment. It is designed to follows a certain line with a distinctive color different from the background.

It is composed by two sensors identifying the line and a microcontroller loaded with a code or a program to run the motors according to the input of the sensors which will result in the tracing movement.

The term ‘line follower’ might sound plain and undeveloped, but line tracking is one of the most useful and popular behavior of mobile robots that has been acquiring increasing importance in the recent times. Line tracking has become the most convenient and reliable navigation technique by which autonomous mobile robots navigate in a controlled usually indoor environment. Line followers are needed in a wide selection of applications like: industries,

factories, corps management, assembly lines, surveillance, massive storage facilities and public transport.
Moreover, the same principles and theories are used in modern vehicles' cruise control and auto-parking systems.

1.2 Problem Statement

As much as normal line followers are useful and simple, they come with great disadvantages: The robot must not exceed a certain speed or it will steer out of track due to its limited response and feedback. The thickness of the line drawn has to be in a certain diameter. The robot will run in a sluggish, jerky, irregular pattern of movement which will result in instability and loss in overall traction. The robot will certainly run out of tracks on hard corners and 90 degrees or more turns. The track has to be made in a complicated and more costing way to compensate the lack of cornering power. The robot will mathematically and physically cross a longer distance due to its uneven tracking and linear sideways movement. More distance to cross consumes more energy which is an undesirable drawback. All the previous disadvantages will result in an overall less practical line follower due to the lack of calibration.

1.3 Objectives

- Designing a controller that enhances the robot performance and reliability by reducing and nullifying irregular and uneven pattern of movements.
- To design a system increasing the speed limit of the robot and decreasing the power consumption, saving more energy.
- Modeling the system by acquiring smoother steering around the corners.
- To design a model increasing the stability and reducing stress on the robot's body and cargo. And improving the vehicle's traction making it safer for itself and its environment.
- Implementing a system that reduces the time required for the robot to run its course and do its task.

1.4 Methodology and Proposed Solution

Increasing the number of sensors, and using Raspberry PI as a microcontroller. And using a PID controller embedded inside the system's software (script), and using the manual tuning method to find the constants' values till we get an optimal system that will make the robot able to predict corners, measure their angle and level of difficulty. All-Wheel-Drive system will resolve the instability and increase traction. Using valued logic to describe the motors' speed, name the sensors, describe the corners' angle and difficulty. While valued logic will pave the way for the possibility of coding the program's script by accurately describing the PWM value. And the PWM will make the motors run in various speed values depending on the required turn angle.

1.5 Project Layout

This project contains five chapters: Chapter One is about the introduction; it presents an overview of the project, the problem statement, objectives, methodology and proposed solution. Chapter Two presents the literature review, theoretical background and previous models. Chapter Three describes the chosen software, hardware and their specifications. Chapter Four demonstrates implementation, assembly, execution, testing and results. Chapter Five includes the conclusion and further recommendations.

CHAPTER TWO

THEORETICAL BACKGROUND AND LITERATURE REVIEW

2.1 Introduction

A microcontroller is a compact integrated circuit designed to govern a specific operation in an embedded system. A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip. Sometimes referred to as an embedded controller or microcontroller unit (MCU), microcontrollers are found in vehicles, robots, office machines, medical devices, mobile radio transceivers, vending machines and home appliances among other devices.

2.2 Microcontrollers

A microcontroller's processor will vary by application. Options range from the simple 4-bit, 8-bit or 16-bit processors to more complex 32-bit or 64-bit processors. In terms of memory, microcontrollers can use random access memory (RAM), flash memory, EPROM or EEPROM. Generally, microcontrollers are designed to be readily usable without additional computing components because they are designed with sufficient onboard memory as well as offering pins for general I/O operations, so they can directly interface with sensors and other components. Microcontroller architecture can be based on the Harvard architecture or von Neumann architecture, both offering different methods of exchanging data between the processor and memory. With a Harvard architecture, the data bus and instruction are separate, allowing for simultaneous transfers. With a Von Neumann architecture, one bus is used for both data and instructions.

When they first became available, microcontrollers solely used assembly language. Today, programming languages are a popular option.

Microcontrollers are used in multiple industries and applications, including in home and enterprise, building automation, manufacturing, robotics, automotive, lighting, smart energy, industrial automation, communications and internet of things (IoT) deployments. The simplest microcontrollers facilitate the operation of electromechanical systems found in everyday convenience items, such as ovens, refrigerators, toasters, mobile devices, key fobs, video games, televisions and lawn-watering systems. They are also common in office machines such as photocopiers, scanners, fax machines and printers, as well as smart meters, ATMs and security systems. More sophisticated microcontrollers perform critical functions in aircraft, spacecraft, ocean-going vessels, vehicles, medical and life-support systems, and robots. In medical scenarios, microcontrollers can regulate the operations of an artificial heart, kidney or other organ. They can also be instrumental in the functioning of prosthetic devices.

2.2.1 Raspberry Pi

Used as a microcontroller, the Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science. It’s a credit card-sized computer originally designed for education, inspired by the 1981 BBC Micro. Creator Eben Upton's goal was to create a low-cost device that would improve programming skills and hardware. But thanks to its small size and accessible price, it was quickly adopted by tinkerers, makers, and electronics enthusiasts for projects that require more than a basic microcontroller (such as Arduino devices).

The Raspberry Pi 3 is a Single Board Computer (SBC). This means that the board is a fully functional computer with its own dedicated processor, memory, and can run an operating system (runs on Linux). The Raspberry Pi 3 includes its own USB ports, audio output, and has a graphic driver for HDMI output, showing how it can run multiple programs. You can even install other operating systems that include Android, Windows 10, or Firefox OS. The Raspberry Pi is

slower than a modern laptop or desktop but is still a complete Linux computer and can provide all the expected abilities that implies, at a low-power consumption level. The Raspberry Pi Foundation is a registered educational charity (registration number 1129409) based in the UK [4].

The Raspberry Pi is an overclock-able piece of hardware; it's CPU can be overclocked in the range of 20% extra performance, however no over-clocking was done in this project. Also, heatsinks can be used but no heatsink was needed, as the chip used in the Raspberry Pi is equivalent to one used in a mobile phone, and should not become hot enough to require any special cooling. However, depending on the case and the overclocking settings, a heatsink might be found to be advantageous.

The Raspberry Pi Foundation provides Raspbian as the default operating system, which is a Debian-based Linux distribution for download. Figure 2.1 shows Raspberry Pi 3 model B.

Figure 2.1**:** Raspberry Pi 3 model B

2.2.2 Arduino

Arduino is also a microcontroller, it's an open-source platform used for building electronics projects. Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (Integrated Development Environment) that runs on your computer, used to write and upload computer code to the physical board. Arduino does not need a separate piece of hardware (called a programmer) in order to load new code onto the board – you can simply use a USB cable. Additionally, the Arduino IDE uses a simplified version of C++ (programing language), making it easier to learn to program. Arduino, was born in Italy. It was named after the bar where inventor Massimo Banzi and his cofounders first forged the idea. Banzi, a teacher at the Interaction Design Institute Ivrea, wanted a simple hardware prototyping tool for his design students.

2.2.3 Raspberry Pi vs Arduino

Arduino and Raspberry Pi are quite different. For starters, Raspberry Pi is a fully functional computer, while Arduino is a microcontroller, which is just a single component of a computer. Raspberry Pi 3 is a mini-computer, it can multitask several programs with its Broadcom BCM2837 SoC, meaning that building a complex project that needs multiple actions at a time are easily handled. The Raspberry Pi 3 can connect to Bluetooth devices and the Internet right out of the box using Ethernet or by connecting to Wi-Fi. The Arduino Uno cannot do that without a Shield that adds Internet or Bluetooth connectivity. HATS and Shields help with this. HATs (Hardware Attached on Top) and Shields have essentially the same goal, adding an extra, or simplifying functionality. The HATs can be used on the Raspberry Pi 3, where some HATs include the Pi to control an RBG Matrix, add a touchscreen, or even create an arcade system. The Shields that can be used on the Arduino Uno include a Relay Shield, a Touchscreen Shield, or a Bluetooth Shield. There are hundreds of Shields and HATs that provide the functionality that you regularly use. The

Raspberry Pi 3 also has an HDMI port, audio port, 4 USB ports, camera port, and LCD port, making it ideal for media applications. The Arduino Uno does not have any of these ports in the board (though some of can be added through Shields.

The price and size of the two devices are comparable; we already knew Raspberry Pi and Arduino were tiny and cheap. It's the stuff inside that sets them apart. The Raspberry Pi is 40 times faster than an Arduino when it comes to clock speed. Even more seemingly damning for Arduino, Pi has 128,000 times more RAM. The Raspberry Pi is an independent computer that can run an actual operating system in Linux. It can multitask, support two USB ports, and connect wirelessly to the Internet.

In short, it's powerful enough to function as a personal computer. Table 2.1 shows a comparison between the two microprocessors:

Table 2.1**:** Arduino and Raspberry Pi comparison:

	Arduino Uno	**Raspberry Pi Model B**
Price	$30	$35
Size	7.6 x 1.9 x 6.4 cm	8.6cm x 5.4cm x 1.7cm
Memory	0.002MB	512MB
Clock Speed	16 MHz	700 MHz
On Board Network	None	10/100 wired Ethernet RJ45
Multitasking	No	Yes
Input voltage	7 to 12 V	5 V
Flash	32KB	SD Card (2 to 16G)
USB	One, input only	Two, peripherals OK
Operating System	None	Linux distributions
Integrated Development Environment	Arduino	Scratch, IDLE, anything with Linux support

2.3 Motor Driver

Motor drives are circuits used to run a motor. In other words, they are commonly used for motor interfacing. These drive circuits can be easily interfaced with the motor and their selection depends upon the type of motor being used and their ratings (current, voltage).

2.4 IR Sensors

An Infrared light emitting diode (IR LED) is a special purpose LED emitting infrared rays ranging 700 nm to 1 mm wavelength. Different IR LEDs may produce infrared light of differing wavelengths, just like different LEDs produce light of different colors. IR LEDs are usually made of gallium arsenide or aluminum gallium arsenide. In complement with IR receivers, these are commonly used as sensors in figure 2.2:

Figure 2.2**:** IR sensor

An IR LED is a type of diode or simple semiconductor. Electric current is allowed to flow in only one direction in diodes. As the current flows, electrons fall from one part of the diode into holes on another part. In order to fall into

these holes, the electrons must shed energy in the form of photons, which produce light.

The appearance of IR LED is same as a common LED. Since the human eye cannot see the infrared radiations, it is not possible for a person to identify if an IR LED is working. A camera on a cell phone camera solves this problem. The IR rays from the IR LED in the circuit are shown in the camera. An IR sensor consists of an IR Receiver and an IR Emitter. IR emitter is an IR LED that continuously emits infrared radiations while power is supplied to it. IR receiver can be thought of as a transistor with its base current determined by the intensity of IR light received. Lower intensity of IR light causes higher resistance between collector-emitter terminals of transistors and limits current from collector to emitter. This change of resistance will further change the voltage at the output of voltage divider. In others words, the greater the intensity of IR light hitting IR receiver, the lower the resistance of IR receiver. Hence the output voltage of voltage divider will decrease as shown in Figure 2.3 below.

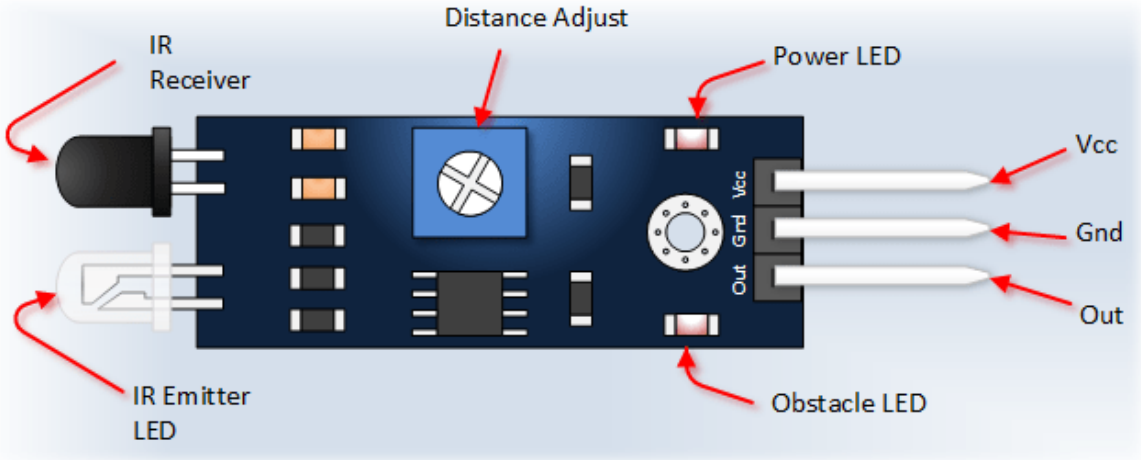

Figure 2.3: IR sensor pins

An IR sensor consists of two parts, the emitter circuit and the receiver circuit. This is collectively known as a photo-coupler or an optocoupler.

The emitter is an IR LED and the detector is an IR photodiode. The IR phototdiode is sensitive to the IR light emitted by an IR LED. The photo-diode's resistance and output voltage change in proportion to the IR light received. This is the underlying working principle of the IR sensor.

The type of incidence can be direct incidence or indirect incidence. In direct incidence, the IR LED is placed in front of a photodiode with no obstacle in between. In indirect incidence, both the diodes are placed side by side with an opaque object in front of the sensor. The light from the IR LED hits the opaque surface and reflects back to the photodiode as in Figure 2.4:

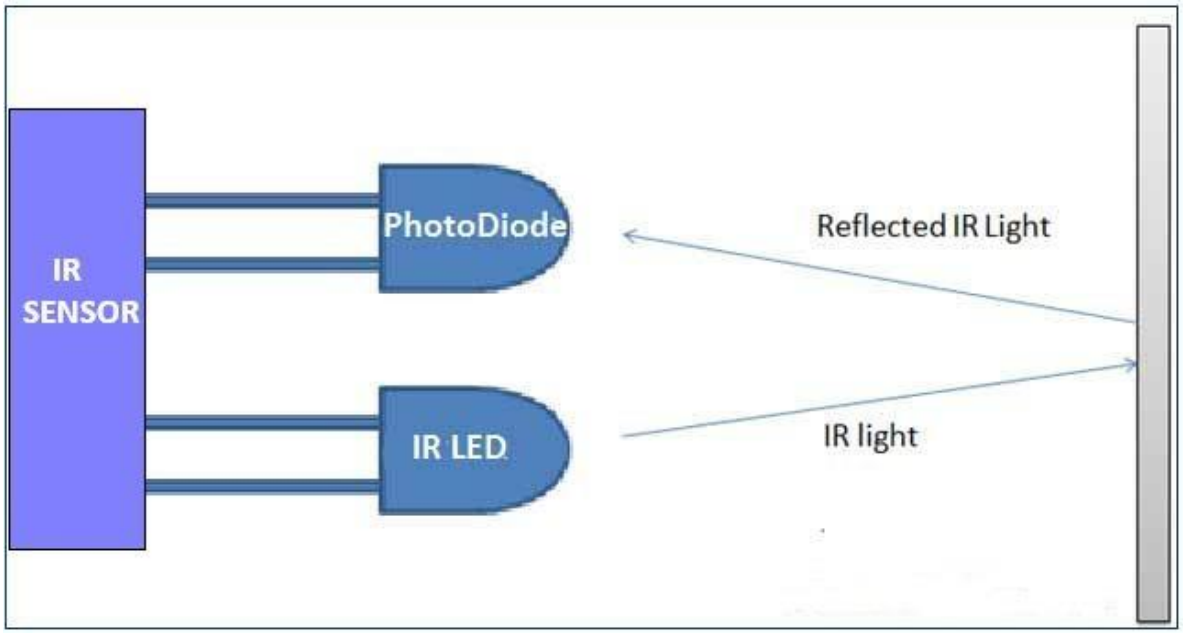

Figure 2.4: IR reflection principle

In line following robots, IR sensors detect the color of the surface underneath it and send a signal to the microcontroller or the main circuit which then takes decisions according to the algorithm set by the creator of the bot. Line followers employ reflective or non-reflective indirect incidence. The IR is reflected back to the module from the white surface around the black line. But IR radiation is absorbed completely by black color. There is no reflection of the IR radiation going back to the sensor module in black color.

2.5 Concept of the Normal Line Follower

Line Follower Robot is able to track a line with the help of an IR sensor. This sensor has a IR Transmitter and IR receiver. The IR transmitter (IR LED) transmits the light and the Receiver (Photodiode) waits for the transmitted light to return back. An IR light will return back only if it is reflected by a surface. Whereas, all surfaces do not reflect an IR light, only white the color surface can completely reflect them and black color surface will completely observe them as shown in Figures 2.5, 2.6 [2].

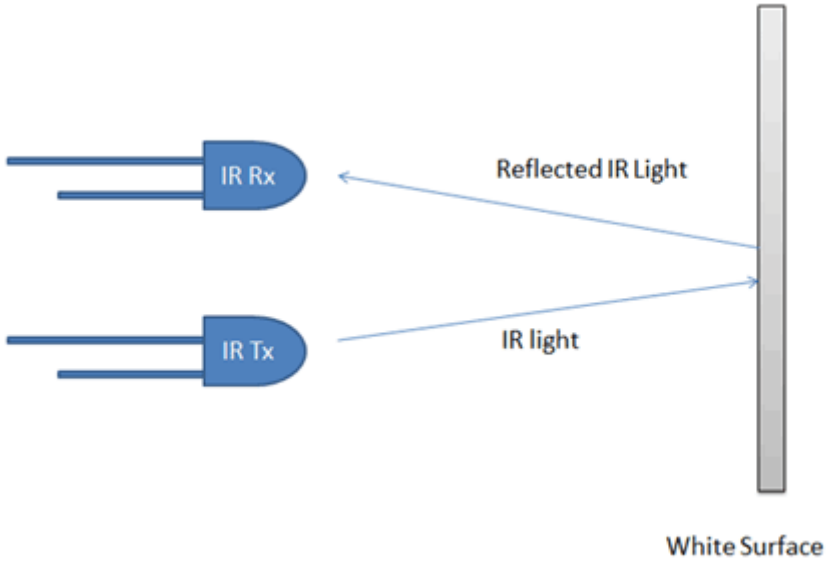

Figure 2.5: IR light reflection white

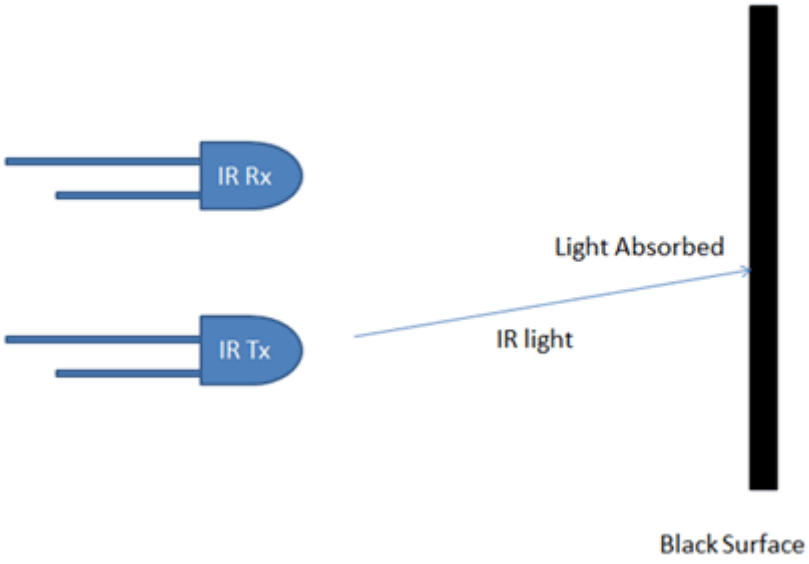

Figure 2.6: IR light reflection Black

The two IR sensors to check if the robot is in track with the line and two motors to correct the robot if its moves out of the track. These motors require high current and should be bi-directional; hence a motor driver is required. a computational device like Raspberry Pi is needed to instruct the motors based on the values from the IR sensor. A simplified block diagram of the same is shown in Figure 2.7

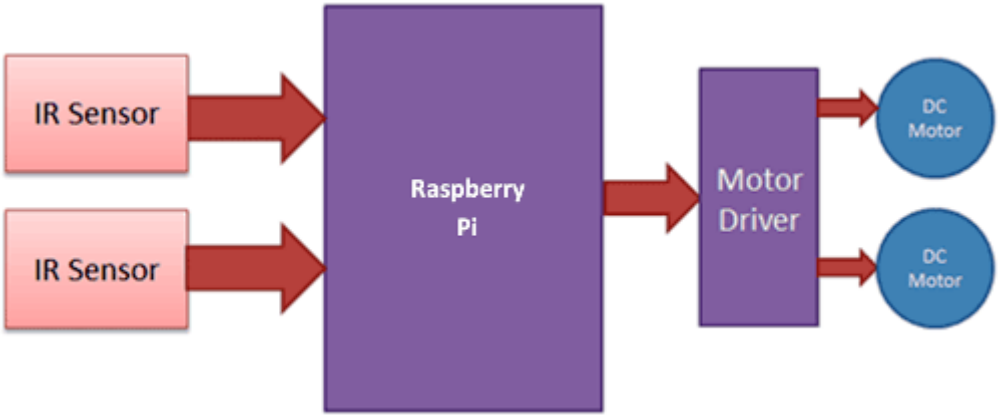

Figure 2.7 Normal line follower block diagram

These two IR sensors will be placed one on either side of the line. If none of the sensors are detecting a black line them they PI instructs the motors to move forward as shown in figure 2.8:

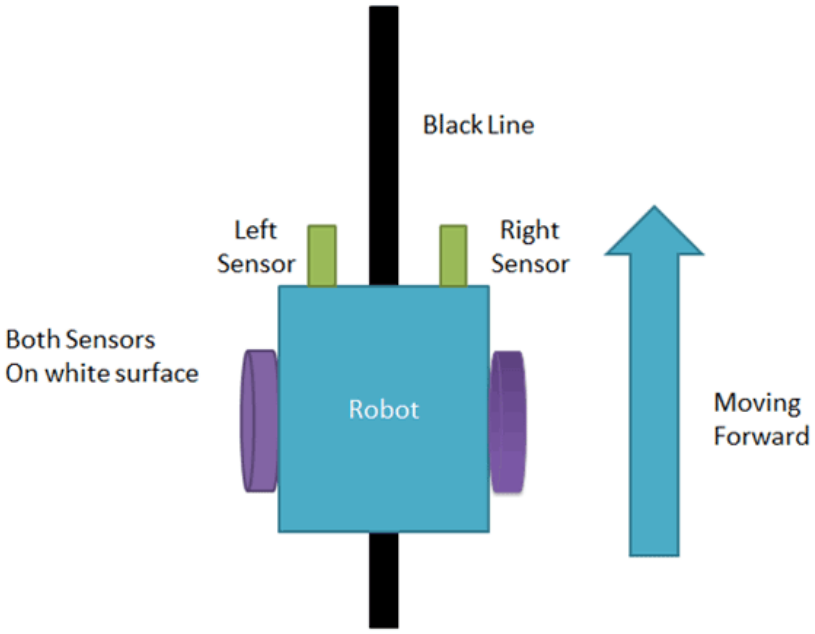

Figure 2.8: Line follower moving forward

If left sensor comes on the black line, then the PI instructs the robot to turn left by rotating the right wheel alone as shown in figure 2.9:

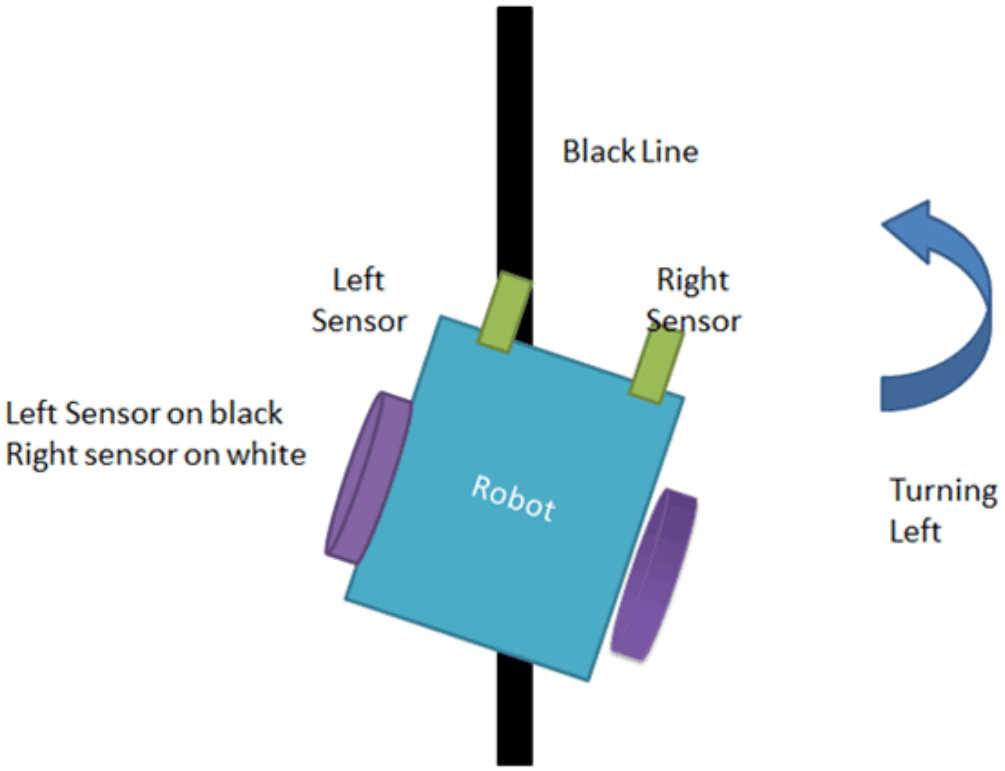

Figure 2.9: Line follower turning left

If right sensor comes on black line, then the PI instructs the robot to turn right by rotating the left wheel alone as shown in figure 2.10:

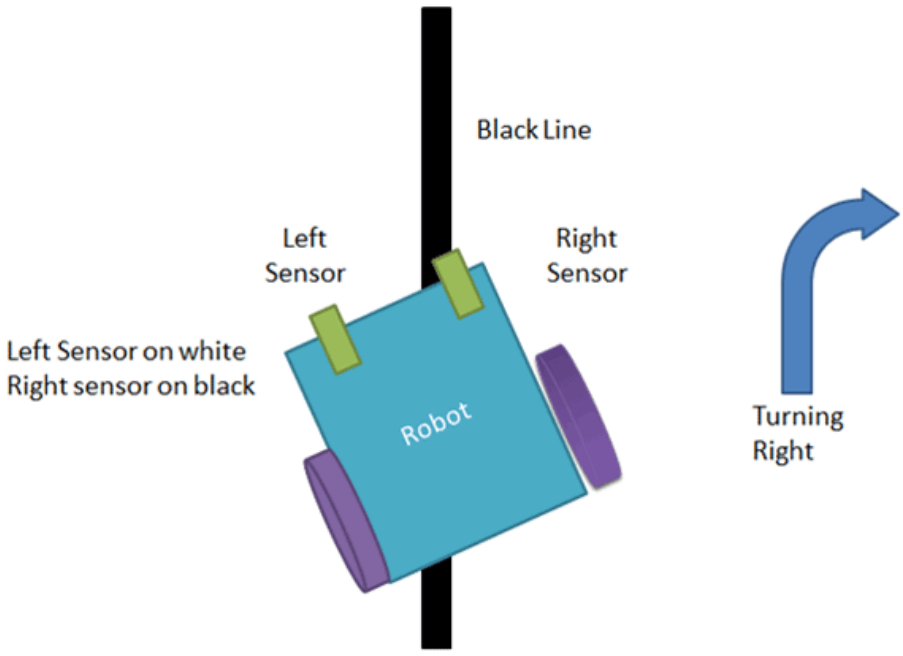

Figure 2.10: Line follower turning right

If both sensors come on black line, the robot stops as shown in figure 2.11:

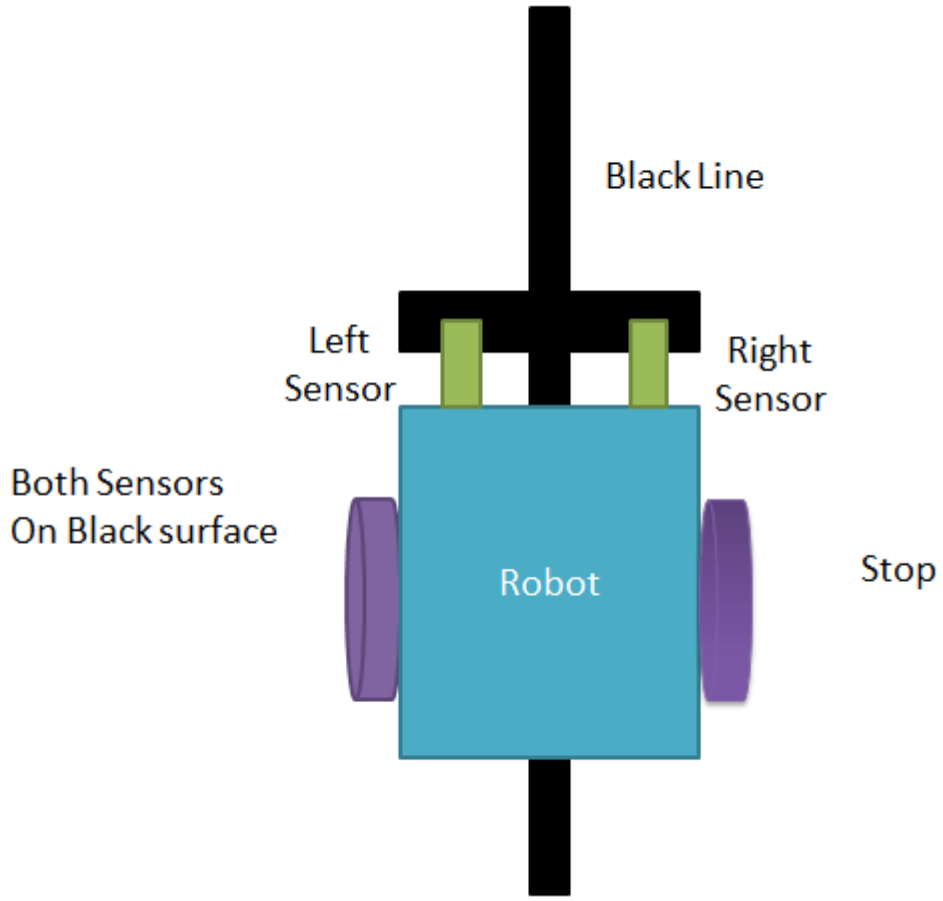

Figure 2.11: Line follower stopping

This way the Robot will be able to follow the line without getting outside the track

2.6 PID Controller

A proportional–integral–derivative controller (PID) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. Approximately 95% of the closed loop operations of industrial automation sector use PID controllers. A PID controller continuously calculates an error value e(t) as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively), hence the name as shown in Figure 2.12. In practical terms it automatically applies accurate and responsive correction to a control function. An everyday example is the cruise control on a car; where external influences such as hills (gradients) would decrease speed. The PID algorithm restores from current speed to the desired speed in an optimal way, without delay or overshoot, by controlling the power output of the vehicle's engine. The first theoretical analysis and practical application was in the field of automatic steering systems for ships, developed from the early 1920s onwards. It was then used for automatic process control in manufacturing industry, where it was widely implemented in pneumatic, and then electronic, controllers. Today there is universal use of the PID concept in applications requiring accurate and optimized automatic control [3,5] as shown in figure 2.12:

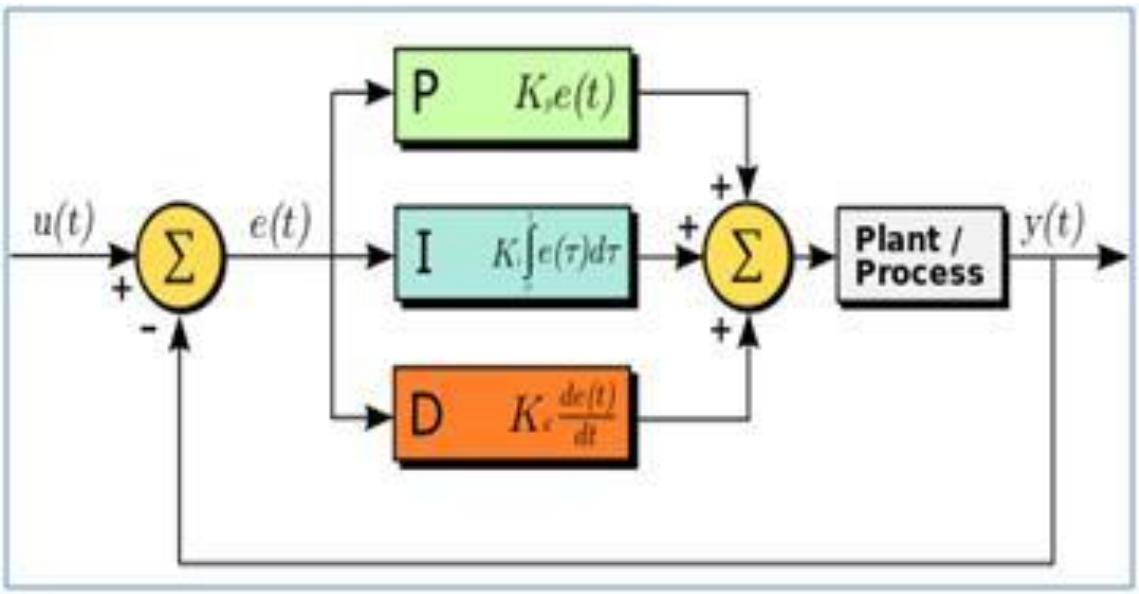

Figure 2.12: PID controller

$$G_{pid} = K_p\left(1 + \frac{1}{T_i s} + T_d s\right) \tag{2.1}$$

where Kp, Ti and Td, are all non-negative, denote the coefficients for the proportional, integral, and derivative terms respectively (sometimes denoted P, I, and D).

As a feedback controller, it delivers the control output at desired levels. Before microprocessors were invented, PID control was implemented by the analog electronic components. But today all PID controllers are processed by the microprocessors. Programmable logic controllers also have the inbuilt PID controller instructions. Due to the flexibility and reliability of the PID controllers, these are traditionally used in process control applications [3,5].

2.6.1 P- Controller:

Proportional or P controller gives output which is proportional to current error e(t). It compares desired or set point with actual value or feedback process

value. The resulting error is multiplied with proportional constant to get the output. If the error value is zero, then this controller output is zero as shown in figure 2.13:

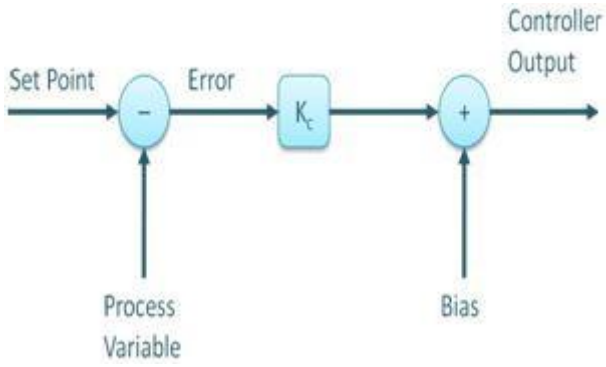

Figure 2.13: P controller

P-Controller Response:

This controller requires biasing or manual reset when used alone. This is because it never reaches the steady state condition. It provides stable operation but always maintains the steady state error. Speed of the response is increased when the proportional constant Kc increases as shown in figure 2.14:

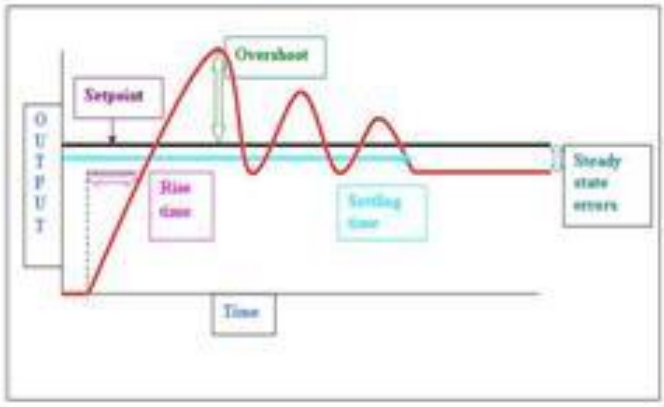

Figure 2.14: P controller response

2.6.2 I-Controller:

Due to limitation of p-controller where there always exists an offset between the process variable and set point, I-controller is needed, which

provides necessary action to eliminate the steady state error. It integrates the error over a period of time until error value reaches to zero. It holds the value to final control device at which error becomes zero as shown in figure 2.15:

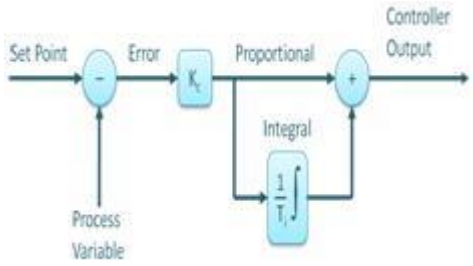

Figure 2.15: PI controller

Integral control decreases its output when negative error takes place. It limits the speed of response and affects stability of the system. Speed of the response is increased by decreasing integral gain Ki as shown in figure 2.16:

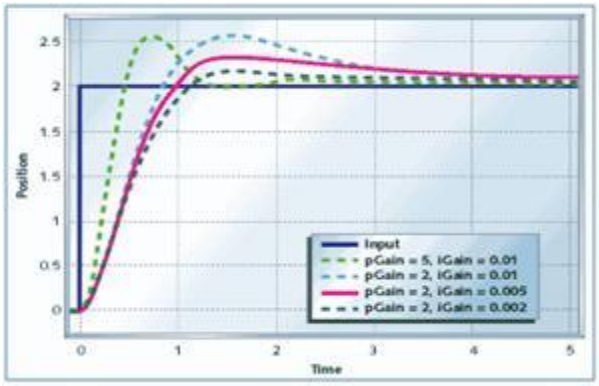

Figure 2.16: PI Controller Response

In above figure, as the gain of the I-controller decreases, steady state error also goes on decreasing. For most of the cases, PI controller is used particularly where high speed response is not required.

While using the PI controller, I-controller output is limited to somewhat range to overcome the integral wind up conditions where integral output goes on increasing even at zero error state, due to nonlinearities in the plant.

2.6.3 D-Controller:

I-controller doesn't have the capability to predict the future behavior of error. So it reacts normally once the set point is changed. D-controller overcomes this problem by anticipating future behavior of the error. Its output depends on rate of change of error with respect to time, multiplied by derivative constant. It gives the kick start for the output thereby increasing system response as shown in figure 2.17, 2.18:

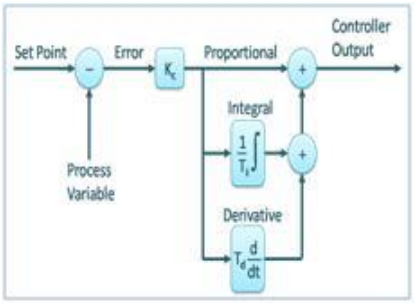

Figure 2.17: PID controller

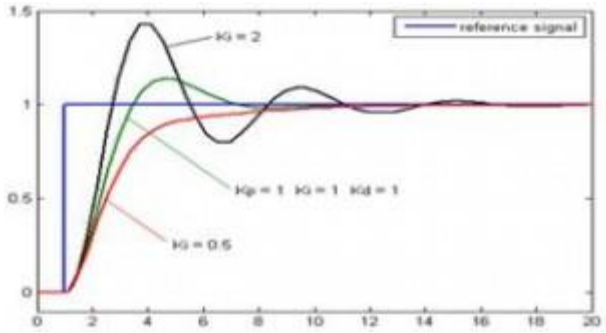

Figure 2.18: PID Controller Response

In the above figure response of D controller is more, compared to PI controller and also settling time of output is decreased. It improves the stability of system by compensating phase lag caused by I-controller. Increasing the derivative gain increases speed of response.

So finally it’s observed that by combining these three controllers, the desired response for the system is approachable. Different manufactures design different PID algorithms [3,5].

2.7 Tuning Methods of PID Controller

Before the working of PID controller takes place, it must be tuned to suit with dynamics of the process to be controlled. Designers give the default values for P, I and D terms and these values couldn’t give the desired performance and sometimes leads to instability and slow control performances. Different types of tuning methods are developed to tune the PID controllers and require much attention from the operator to select best values of proportional, integral and derivative gains.

2.7.1 Manual Tuning:

If the system must remain online, one tuning method is to first set Ki and Kd values to zero. Increase the Kp until the output of the loop oscillates, then the Kp should be set to approximately half of that value for a "quarter amplitude decay" type response. Then increase Ki until any offset is corrected in sufficient time for the process. However, too Ki will cause instability. Finally, increase Kd, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, too much Kd will cause excessive response and overshoot. A fast PID loop tuning usually overshoots slightly to reach the setpoint more quickly; however, some systems cannot accept overshoot, in which case an overdamped closed-loop system is required, which will require a Kp setting significantly less than half that of the Kp setting that was causing oscillation.

Table 2.2: Manual Tuning

Parameter	Rise time	Overshoot	Settling time	Steady-state error	Stability
Kp	Decrease	Increase	Small change	Decrease	Degrade
Ki	Decrease	Increase	Increase	Eliminate	Degrade
Kd	Minor change	Decrease	Decrease	No effect in theory	Improve if Kd small

2.7.2 Trial and Error Method:

It is a simple method of PID controller tuning. While system or controller is working, we can tune the controller. In this method, first we have to set Ki and Kd values to zero and increase proportional term (Kp) until system reaches to oscillating behavior. Once it is oscillating, adjust Ki (Integral term) so that oscillations stops and finally adjust D to get fast response.

2.7.3 Process Reaction Curve Technique:

It is an open loop tuning technique. It produces response when a step input is applied to the system. Initially, we have to apply some control output to the system manually and have to record response curve.After that we need to calculate slope, dead time, rise time of the curve and finally substitute these values in P, I and D equations to get the gain values of PID terms.

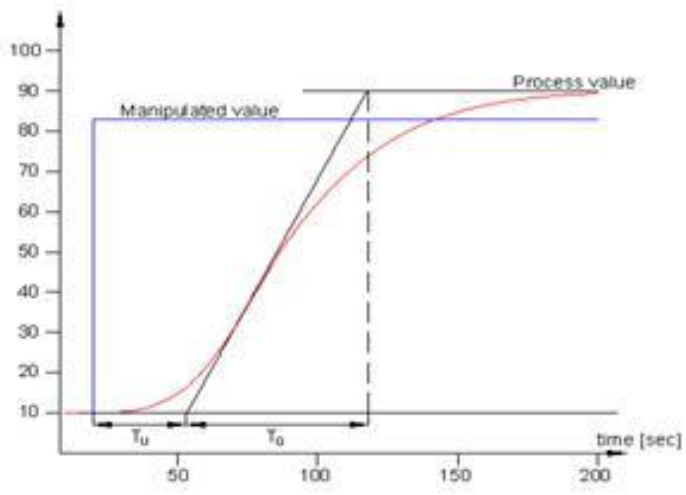

Figure 2.19: Process reaction curve

2.7.4 Zeigler-Nichols Method:

Zeigler-Nichols proposed closed loop methods for tuning the PID controller. Those are continuous cycling method and damped oscillation method. Procedures for both methods are same but oscillation behavior is different. In this, first we have to set the p-controller constant, Kp to a particular value while Ki and Kd values are zero. Proportional gain is increased till system oscillates at constant amplitude.Gain at which system produces constant oscillations is called ultimate gain (Ku) and period of oscillations is called ultimate period (Pc). Once it is reached, we can enter the values of P, I and D in PID controller by Zeigler-Nichols table depends on the controller used like P, PI or PID, as shown below:

Table 2.3: Zeigler-Nichols table

	Kc	Ti	Td
P	Ko/2		
PI	Ko/2.2	Po/1.2	
PID	Ko/1.7	Po/2	Po/S

2.8 Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM), is a way of describing a digital (binary/discrete) signal that was created through a modulation technique, which involves encoding a message into a pulsing signal. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially to inertial loads such as motors. The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load. The PWM switching frequency has to be much higher than what would affect the load (the device that uses the power), which is to say that the resultant waveform perceived by the load must be as smooth as possible. The rate (or frequency) at which the power supply must switch can vary greatly depending on load and application, for example Switching has to be done several times a minute in an electric stove; between a few kilohertz (kHz) and tens of kHz for a motor drive; and well into the tens or hundreds of kHz in audio amplifiers and computer power supplies.

The term duty cycle describes the proportion of 'on' time to the regular interval or 'period' of time; a low duty cycle corresponds to low power, because the power is off for most of the time. Duty cycle is expressed in percent, 100% being fully on. When a digital signal is on half of the time and off the other half of the time, the digital signal has a duty cycle of 50% and resembles a "square" wave. When a digital signal spends more time in the on state than the off state, it has a duty cycle of >50%. When a digital signal spends more time in the off state than the on state, it has a duty cycle of <50%. figure 2.20 shows a pictorial that illustrates these three scenarios:

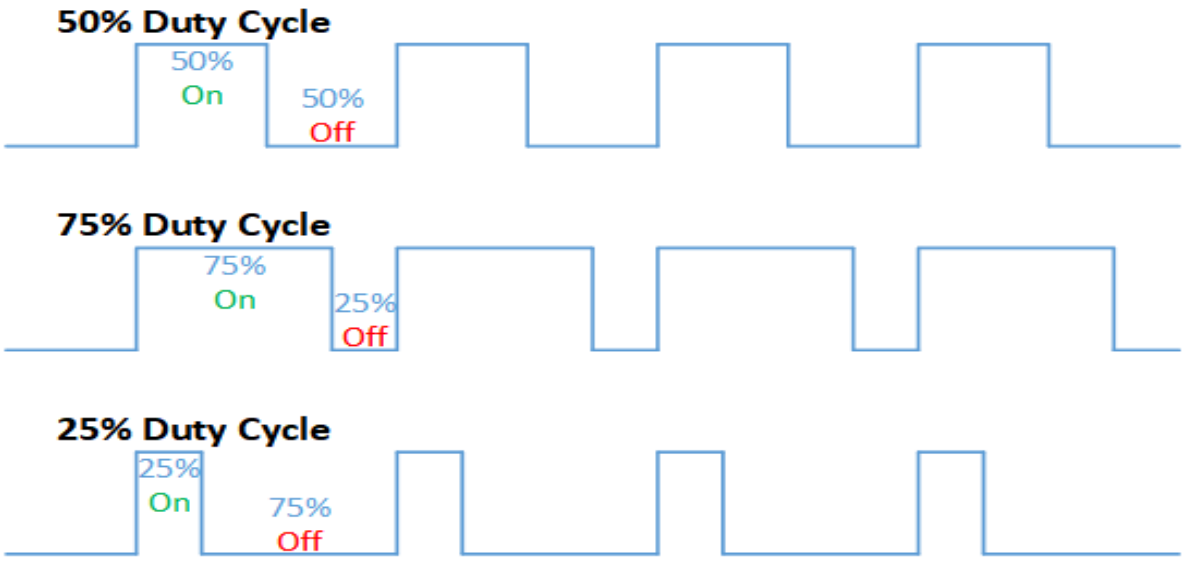

Figure 2.20: Pulse width modulation

2.9 All-Wheel Drive (AWD)

An All-Wheel Drive vehicle (AWD vehicle) is one with a powertrain capable of providing power to all its wheels, whether full-time or on-demand. All-wheel drive is a much more recent innovation to simplify, it's actually very similar to the concept of part-time four-wheel drive. Think of all-wheel drive as similar to the part-time four-wheel drive system, where the main distinction with four-wheel drive is that the system tries to send as much power to all four wheels as equally as possible for utmost traction, all-wheel drive is all about varying the amount of power to each wheel, either by physically with differentials or transfer cases or electronically by brake vectoring (where the brakes are used to slow-down a specific wheel due to traction loss). adjust power delivery according to which wheel loses or maintains traction. All-Wheel Drive does have some clear advantages. These days, computers are involved in most AWD systems. Sensors on each wheel monitor traction, wheel speed, and several other data points hundreds of times per second. A control unit analyzes traction conditions and decides which wheel receives power. This type of system, usually called torque vectoring, appears on everything from the modern cars to space mobile robots. Each axle gets its own electric motor so the four wheels are always powered but there's no mechanical connection between the front and the back of the vehicle. This improves traction and performance.

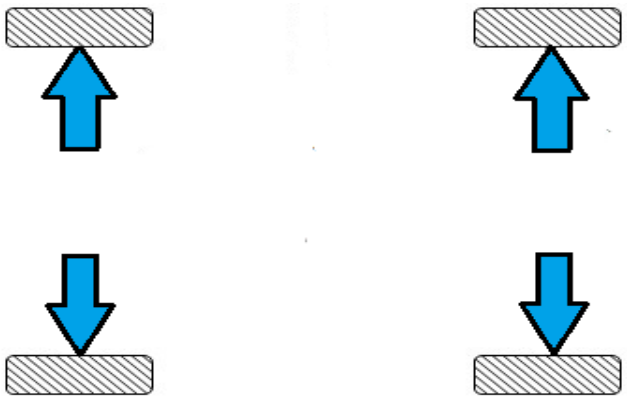

Figure 2.21 Power delivery in AWD system

2.10 Line Follower Without Using Microcontrollers

It consists of an IR-LED and Photodiode arrangement for each motor which is controlled by the switching on and off of the transistor.

The IR LED on getting proper biasing emits Infra-red light. This IR light is reflected in case of a white surface and the reflected IR light is incident on the photodiode. The resistance of the photodiode decreases, which leads to an increase in current through it and thus the voltage drop across it. The photodiode is connected to the base of the transistor and as a result of increased voltage across the photodiode, the transistor starts conducting and thus the motor connected to the collector of the transistor gets enough supply to start rotating. In case of a black color on the path encountered by one of the sensor arrangement, the IR light is not reflected and the photodiode offers more resistance, causing the transistor to stop conduction and eventually the motor stops rotating.

Thus the whole system can be controlled using a simple LED-Photodiode-Transistor arrangement [1,2]. Figure 2.22 shows the block diagram of the Line Follower without using microcontrollers.

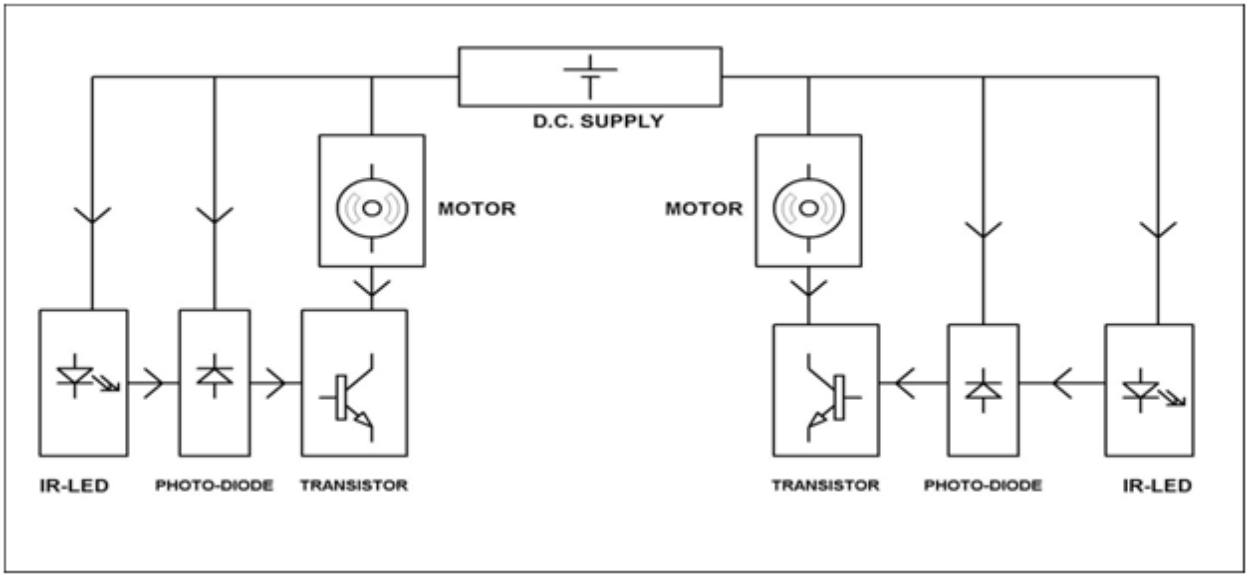

Figure 2.22: Block diagram of the Line Follower without using microcontrollers.

2.11 Line Follower Using Microcontrollers

The line is indicated by white line on a block surface or block line on a white surface. This system must be sense by the line. This application depends upon the sensors. Here we are using two sensors for path detection purpose. That is proximity sensor and IR sensor. The proximity sensor used for path detection and IR sensor used for obstacle detection. These sensors mounted at front end of the robot. The microcontroller is an intelligent device the whole circuit is controlled by the microcontroller [1,2]. Figure 2.23 shows a Line Follower using Microcontrollers circuit diagram.

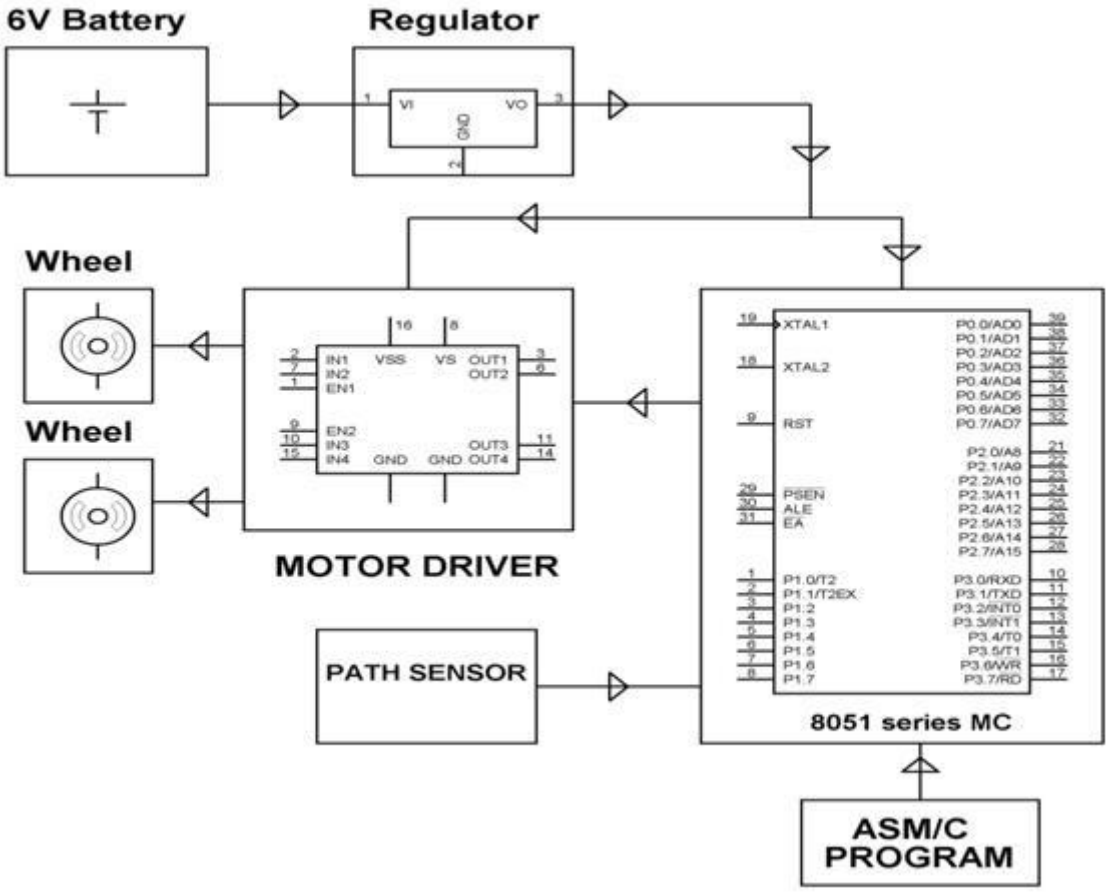

Figure 2.23: Line Follower using Microcontrollers circuit diagram.

2.12 Two Wheels PID line Follower Using Arduino

The test robot is a two-wheeled mobile robot with differential drive. A ball castor provides third point contact to ensure stability. The robot is driven by two high speed miniature permanent magnet gear motors coupled to wheels. The robot is battery powered, with the motors supplied with regulated power supply to ensure consistent motor power at all operating scenarios shown in Figure 2.24. To move the robot forward, both motors are rotated in the forward direction. To make the robot turn to the left or to the right, the speed of one motor is reduced. The amount of turn increases as the speed difference increases. Maximum amount of turn is achieved when one motor is turned in a backward direction at maximum speed. This results in maximum speed difference and the robot just spins in place. The brain of the robot is an R8C25 16-bit microcontroller from Renesas. Bidirectional motor speed control is achieved by using an H-Bridge motor driver circuit. The power applied to the motor is varied by using PWM generated by the microcontroller.

The mentioned model works suitably well for straight lines, curves and moderate turns. However, for sharp acute angled turns, the performance is not acceptable and the robot misses the turn or has excessive overshoot issues. This is due to the fact that, when the path has an abrupt turn, the robot goes entirely off the line causing the sensor error parameter to saturate to values of either 4500 or -4500 depending on the direction of the turn. In this condition, the control effort produced by the modified PID controller will be unable to turn the robot fast enough.

In conclusion, the robot requires an implementation of a lot of systems and a very long code to be viable [1,2].

Figure 2.24: Two wheeled PID line follower using Arduino.

2.13 The Necessity of Further Development

The Arduino chosen as a microprocessor in the previous mentioned models, is popular affordable viable microcontroller. However, it has a limited processing speed; it's designed for single tasking, which will result for models to be solely designed to follow lines and nothing else, no additional tasks can be added. Also it limits the number of input/output methods, thus the control methods and applications.

Arduino forces the user to code using C++, a language developed by Bjarne Stroustrup in 1979, much complicated and limited language compared to recent ones, which will result in a code with hundreds of lines to be written, and that is a lot of coding and an impossible task for a normal designer since the code won't compile if it contained a single error.

Two wheeled models proved speed in tracking, that might be useful for "Line Follower Robot Collage Competitions", but physically speaking ... no high speed object with high momentum can make a sharp turn in a sudden, which makes them not practical for production or manufacturing applications.

2.14 PID Modified Line Tracker

For the PID line tracker, two additional sensors were added to the line tracker, the additional sensors are peripheral. They are added on the sides of the front bumper of the robot for the purpose of determining the value of Kd, which is the "Derivative value", this value will be one responsible for anticipating line turn angle and responding to it.

Now, the robot will no longer follow the line using only the "proportional value" of the two main sensors. Instead, the robot will calculate the error value using the two additional sensors as the displacement away from the line, that's why the name has been changed to "Line Tracker". Because Practically the robot isn't following the line. But instead, the robot is anticipating the upcoming turns, calculating the error and then adjusting the motors speed according to the turns and error values.

It's noticed that no one developed a script or a code for a PID modified line tracker in python for the Raspberry Pi before. Thus, it's needed to start the coding manually from the beginning.

And for the Ki "Integral value", it will be last calculated error converted to a boost of speed when exiting a turn or corner into a straight line. Figure 2.25 shows PID Line Tracker block diagram.

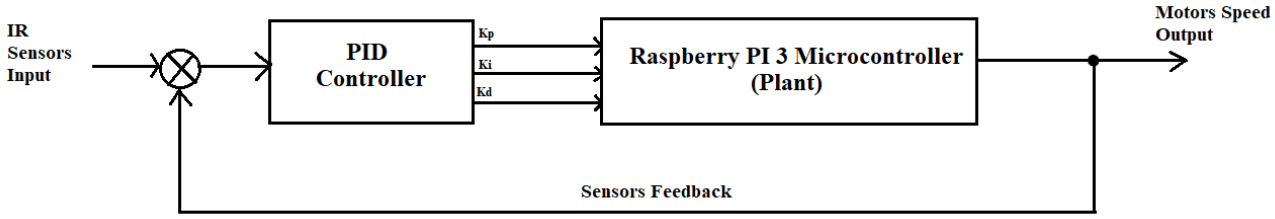

Figure 2.25: PID Line Tracker block diagram.

CHAPTER THREE
SOFTWARE AND HARDWARE COMPONENTS AND SPECIFICATIONS

3.1 Linux

Linux is the best-known and most-used open source operating system. As an operating system, Linux is software that sits underneath all of the other software on a computer, receiving requests from those programs and relaying these requests to the computer's hardware. The term "Linux" to refer to the Linux kernel, but also the set of programs, tools, and services that are typically bundled together with the Linux kernel to provide all of the necessary components of a fully functional operating system. Linux was created in 1991 by Linus Torvalds, a then-student at the University of Helsinki. Torvalds built Linux as a free and open source alternative to Minix, another Unix clone that was predominantly used in academic settings.

In many ways, Linux is similar to other operating such as Windows, OS X, or iOS. Like other operating systems, Linux has a graphical interface, and types of software you are accustomed to using on other operating systems, such as word processing applications, have Linux equivalents. In many cases, the software's creator may have made a Linux version of the same program you use on other systems. If an individual can use a computer or other electronic device, Linux can be used. But Linux also is different from other operating systems in many important ways. First, and perhaps most importantly, Linux is open source software. The code used to create Linux is free and available to the public to view, edit, and—for users with the appropriate skills—to contribute to. Linux is also different in that, Linux is incredibly customizable, because not just applications, such as word processors and web browsers, can be swapped out.

Linux users also can choose core components, such as which system displays graphics, and other user-interface components.

The Linux kernel is a widely ported operating system kernel, available for devices ranging from mobile phones to supercomputers; it runs on a highly diverse range of computer architectures, including the hand-held ARM-based iPAQ and the IBM mainframes System z9 or System z10. Specialized distributions and kernel forks exist for less mainstream architectures; for example, the ELKS kernel fork can run on Intel 8086 or Intel 80286 16-bit microprocessors, while the µClinux kernel fork may run on systems without a memory management unit. The kernel also runs on architectures that were only ever intended to use a manufacturer-created operating system, such as Macintosh computers (with both PowerPC and Intel processors), PDAs, video game consoles, portable music players, and mobile phones.

3.2 Raspbian

Raspbian is a free operating system based on Debian optimized for the Raspberry Pi hardware. An operating system is the set of basic programs and utilities that make your Raspberry Pi run. However, Raspbian provides more than a pure OS; it comes with over 35,000 packages, pre-compiled software bundled in a nice format for easy installation on your Raspberry Pi. Raspbian packages were optimized for best performance on the Raspberry Pi and were completed in June of 2012. However, Raspbian is still under active development with an emphasis on improving the stability and performance of as many Debian packages as possible. Raspbian is a fantastic Raspberry Pi Linux OS. It can picked from several versions of Raspbian including Raspbian Stretch with Desktop and Raspbian Stretch Lite, a minimal Debian Stretch-based Raspbian image.

Since 2015 it has been officially provided by the Raspberry Pi Foundation as the primary operating system for the family of Raspberry Pi single-board computers.

Raspbian was created by Mike Thompson and Peter Green as an independent project. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs. The Raspberry Pi Foundation also maintains its own recommended version of Raspbian which you can install using the Foundation's NOOBS installer. Raspbian uses PIXEL, Pi Improved Xwindows Environment, Lightweight as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft Pi as well as a lightweight version of Chromium as of the latest version. [4]

3.3 Python 3

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers prefer Python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of

local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test-debug cycle makes this simple approach very effective [4].

A list of 8 World-Class Software Companies That Use Python:

1. Industrial Light and Magic
2. Google
3. Facebook
4. Instagram
5. Spotify
6. Quora
7. Netflix
8. Dropbox
9. Reddit

"Python has been an important part of Google since the beginning, and remains so as the system grows and evolved. Today dozens of Google engineers use Python, and we're looking for more people with skills in this language" -- Peter Norvig, Director of Search Quality at Google.

Python 3.0 (also called "Python 3000" or "Py3K") was released on December 3rd, 2008. It was designed to rectify fundamental design flaws in the language. The changes required could not be implemented while retaining full backwards compatibility with the 2.x series, which necessitated a new major version number. The guiding principle of Python 3 was: "reduce feature duplication by removing old ways of doing things".

Python 3.0 was developed with the same philosophy as in prior versions. However, as Python had accumulated new and redundant ways to program the

same task, Python 3.0 had an emphasis on removing duplicative constructs and modules, in keeping with "There should be one— and preferably only one — obvious way to do it".

Nonetheless, Python 3.0 remained a multi-paradigm language. Coders still had options among object-orientation, structured programming, functional programming and other paradigms, but within such broad choices, the details were intended to be more obvious in Python 3.0 than they were in Python 2.x.

3.4 Raspberry PI Generations and Models

Table 3.1: Raspberry Pi different generations and models [4].

Product	SoC	Speed	RAM	USB Ports	Ethernet	Wireless/Bluetooth
Raspberry Pi Model A+	BCM2835	700MHz	512MB	1	No	No
Raspberry Pi Model B+	BCM2835	700MHz	512MB	4	Yes	No
Raspberry Pi 2 Model B	BCM2836/7	900MHz	1GB	4	Yes	No
Raspberry Pi 3 Model B	BCM2837	1200MHz	1GB	4	Yes	Yes
Raspberry Pi 3 Model B+	BCM2837	1400MHz	1GB	4	Yes	Yes
Raspberry Pi Zero	BCM2835	1000MHz	512MB	1	No	No
Raspberry Pi Zero W	BCM2835	1000MHz	512MB	1	No	Yes
Raspberry Pi Zero WH	BCM2835	1000MHz	512MB	1	No	Yes

Raspberry Pi 3 Model B:

As the second latest Pi to be released (Previous to model B+), Raspberry Pi 3 Model B was launched in February 2016, the model is recommended for use in schools, due to its flexibility for the learner. The Raspberry Pi 3 Model B

contains a wide range of improvements and features that will benefit the designers, developers, and even engineers who are looking to integrate Pi systems into their products [4].

3.5 Raspberry PI 3 Model B Specifications, Features and Pins layout

Table 3.2: Raspberry PI 3 Model B specifications and features.

Chip	• Broadcom BCM2837 • 64bit • ARMv8 • Quad Core Cortex A53 • 1.2 GHz
Storage	microSD Card
Memory	1 GB
Graphics	• 400 MHz Dual Core VideoCore IV GPU • OpenGL ES 2.0 • Hardware-Accelerated OpenVG • 1080p30 H.264 high-profile decode
Weight	42g
Audio	• HDMI port supports multichannel audio output • Audio line out/3.5-mm headphone jack (analog)
Connections and Expansions	USB 2.0 ports10/100BASE-T Ethernet portsComposite Video and Audio jackCSI Camera portHDMI portMicroUSB Power InputDSI Display port40-pin GPIO • Four USB 2.0 ports (up to 480 megabits per second) • HDMI port • 3.5mm 4-pole Composite Video and Audio jack • MicroUSB Power Input

	• DSI Display Port • CSI Camera Port • MicroSD card Sold • 40-pin GPIO (Male headers)
Communications	**Wi-Fi** 802.11n WiFi wireless Networking;IEEE 802.11a/g/b/n compatible **Bluetooth** Bluetooth 4.1 wireless technology **Ethernet** 10/100BASE-T Ethernet (RJ-45 connector)
Electrical and Operating Requirements	**Input voltage:** 5V DC **Current Requirement:** 2.5 Amps
Operating System	**Raspberry Pi Foundation's Offical supported Operating Systems** • NOOBS • Raspbian **Third Party Operating Systems** • Libreelec • Open Elec • OSMC • Pinet • RISC OS • Snappy Ubuntu Core • Ubuntu Mate • Weather Station • Windows IOT Core • XBian

The Raspberry Pi Model B versions measure 85.60mm x 56mm x 21mm (or roughly 3.37″ x 2.21″ x 0.83″), with a little overlap for the SD card and connectors which project over the edges. They weigh 45g. The Pi Zero and Pi Zero W measure 65mm x 30mm x 5.4mm (or roughly 2.56″ x 1.18″ x 0.20″) and weigh 9g [4].

Figure 3.1: Raspberry Pi 3 model B

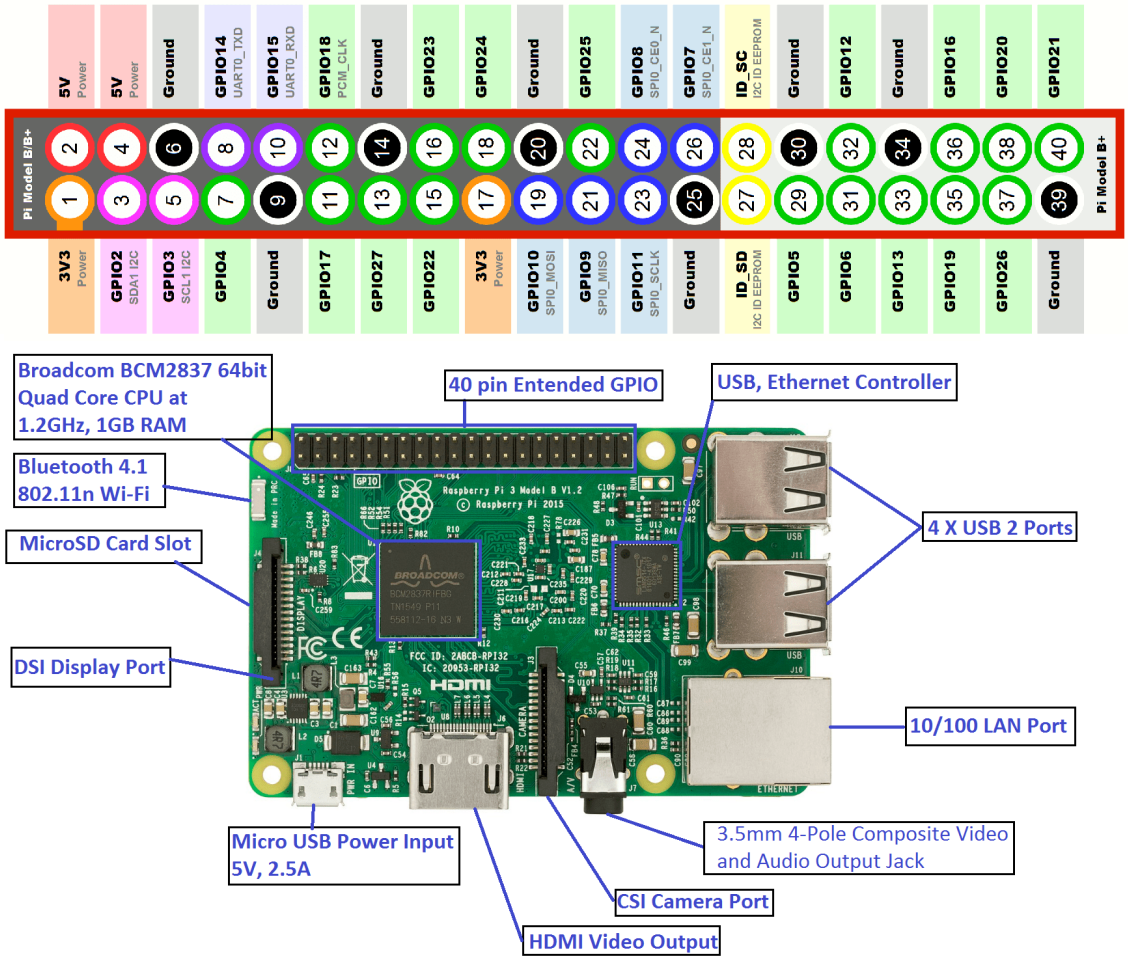

Figure 3.2: Raspberry Pi 3 Model B Components and Pinout

Additional Requirements (Not Included):

- Micro SD card with NOOBS.
- Micro USB power supply (2.1 A).

And to use it as a desktop computer:

- TV or monitor and HDMI cable.
- Keyboard and mouse.

3.6 L298N Motor Driver

The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A. The L298N is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

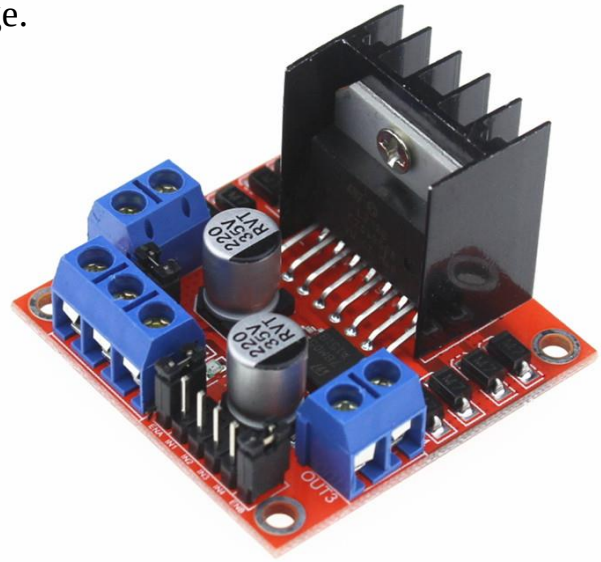

Figure 3.3: L298N Motor Driver

3.7 Vehicle Model

The chosen model is a 4-wheel, 4-motor vehicle model. It's light weighted, transparent, adjustable with rubber coating removable wheels, allowing maximum grip and minimum slide across the track.

The used motors are a set of four DC motors. A DC motor is a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor as will be shown in figures (3.7, 3.8, 3.9). DC motors are speed variant; their speed is adjustable by current control through PWM.

Figure 3.7: Vehicle model (top view)

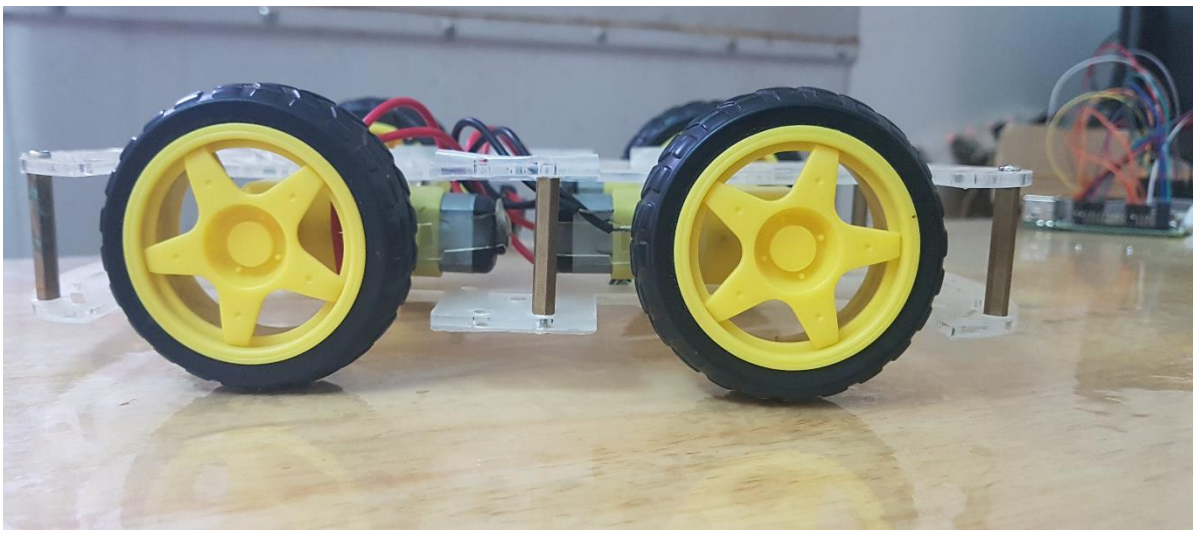

Figure3.8: Vehicle model (side view)

Figure 3.9: DC motors

CHAPTER FOUR

IMPLEMENTATION AND TESTING

4.1 Installing the Raspbian Using NOOBS on the Raspberry PI 3 Model B

To get started with Raspberry Pi, an operating system is needed. NOOBS (New Out of Box Software) is an easy operating system install manager for the Raspberry Pi. NOOBs is not an OS. It is an OS installer. Raspbian installed by the NOOBs method is the same as Raspbian installed by the direct install method using Etcher. SD cards with NOOBS preinstalled are available from many of distributors and independent retailers, such as Pimoroni, Adafruit and The Pi Hut. For Pi 3. A micro SD card is needed. The SD card minimum capacity is 8GB. Using a computer with an SD card reader, NOOBS was downloaded from the official webpage as a zip file. SD card was formatted before copying the NOOBS files onto it. SD card was inserted into the laptop's SD card reader. In SD Formatter, the drive letter for your SD card was selected and formatted. Once the SD card has been formatted, all the extracted NOOBS files were written on the SD card drive. When this process was finished, the SD card was safely removed and inserted into the Raspberry Pi.

First Boot:

The keyboard, mouse, and monitor cables were plugged in. Then the USB power cable was plugged into the Pi. The Raspberry Pi booted, and a window appeared with a list of different operating systems that can be installed. Raspbian will then run through its installation process. Note that this took a while. When the install process was completed, the Raspberry Pi configuration menu (raspi-

config) was loaded. The default login for Raspbian is username pi with the password raspberry. Note that no writing will appear when the password is typed. This is a security feature in Linux. To load the graphical user interface, **startx** has to be typed and then Enter.

4.2 Implementing the Normal Line Follower

The complete circuit diagram for this Raspberry Pi Line Follower Robot is shown below in figure 4.1:

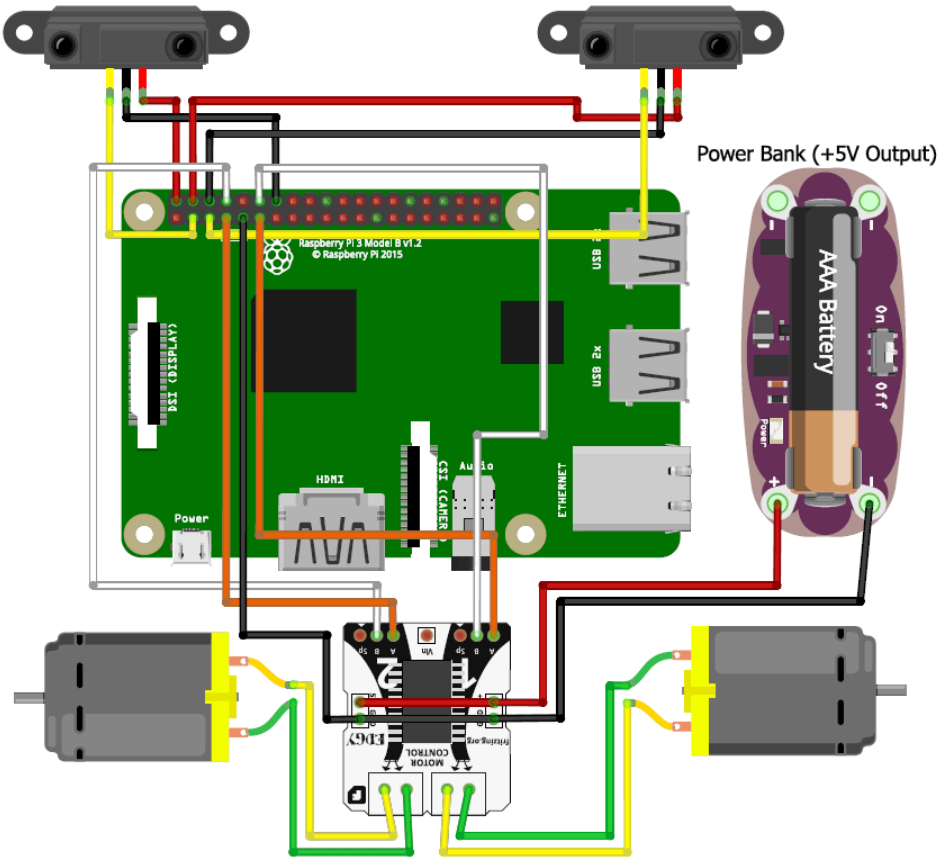

Figure 4.1: Normal line follower circuit diagram

As can be observed the circuit involves two IR sensor and a pair of motors connected to the Raspberry pi. The complete circuit is powered by a Mobile Power bank (represented by AAA battery in the circuit above).

Since the pins details are not mentioned on the Raspberry Pi, it is needed to verify the pins using the below figure:

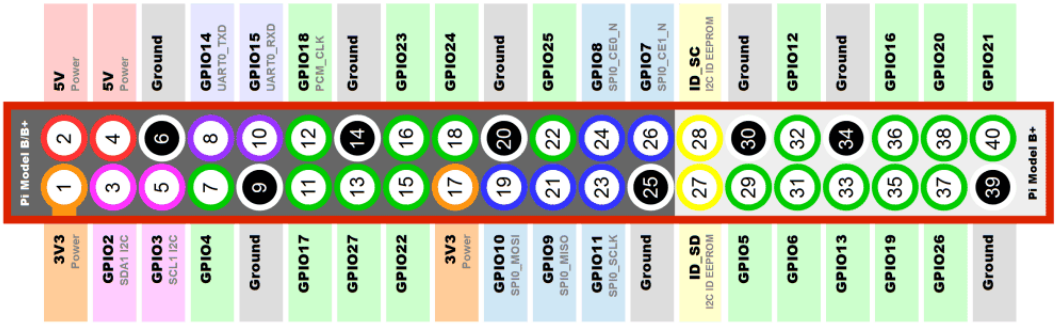

Figure 4.2: Raspberry PI pin layout

As shown in figure 4.2, pin of the PI is the +5V pin, the +5V pin to power the IR sensors as shown in the circuit diagram (red wired). Then the ground pins are connected to the ground of the IR sensor and motor driver module using black wire. The yellow wire is used to connect the output pin of the sensor 1 and 2 to the GPIO pins and 3 respectively.

To drive the Motors, four pins (A,B,A,B) are needed. Those four pins are connected from GPIO14,4,17 and 18 respectively. The orange and white wire together forms the connection for one motor. So it's two such pairs for two motors.

The motors are connected to the L298N Motor Driver module it's powered by a power bank. The ground of the power bank has to be connected to the ground of the Raspberry Pi, only then your connection will work.

4.3 Implementing PID Line Tracker and Placing the Sensors

Since an array of four IR sensors were used, they were identified as follows:

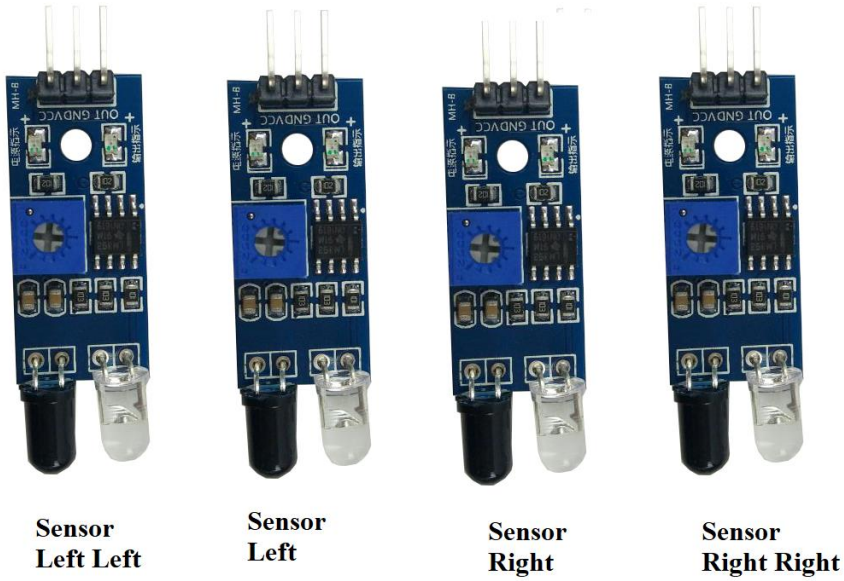

Figure 4.3: Sensors recognition

And the input of two sensors together is also considered as a distinct value, increasing the number of rules.

Both Python 3 and L298N Motor Driver can handle valued logic through PWM.

4.4 Setting the L298N Motor Driver: Connecting the Motors and Setting the PWM Pulse Input

The output of the Raspberry Pi IO pins for the PWM should be connected in the following L298N pins in figure 4.4:

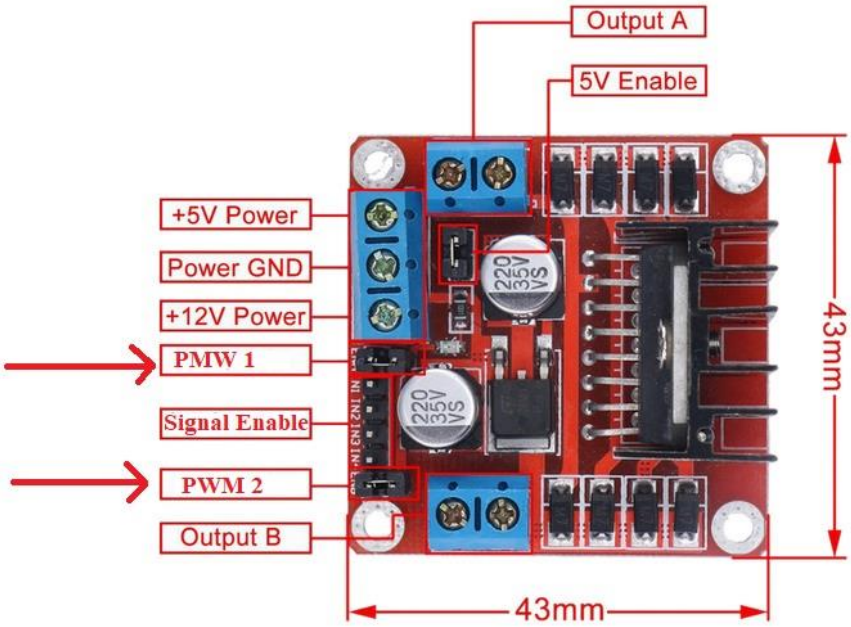

Figure 4.4: L298N PWM pins

Noting that while using the PWM mode in the L298N Motor Driver, the Enable Signal IO in the motor driver must get a startup pulse input or the whole program loop won't execute due to lack of initiation to the motor driver.

The module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output. As shown in figure 4.5:

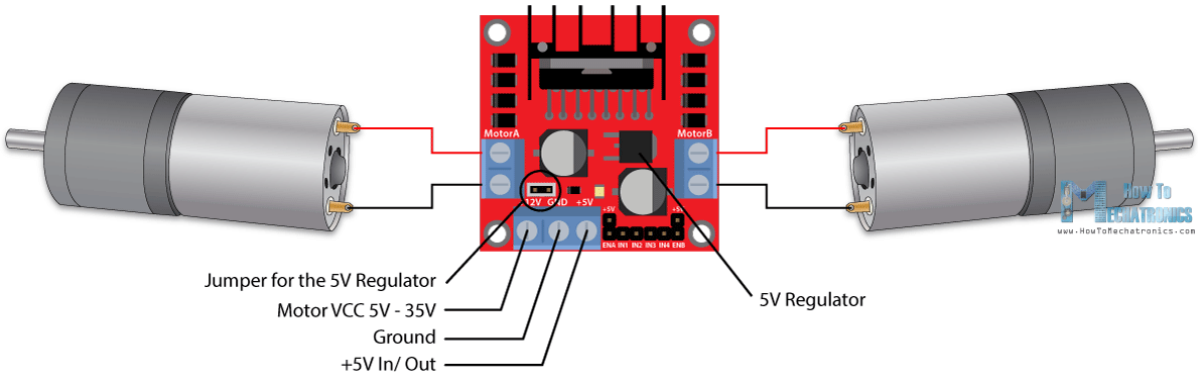

Figure 4.5: L298N double motor connection

This depends on the voltage used at the motors VCC. The module has an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need connect it to a 5V power supply in order the IC to work properly.

It's noted here that this IC makes a voltage drop of about 2V. So for example, if we use a 12V power supply, the voltage at motors terminals will be about 10V, which means that we won't be able to get the maximum speed out of our 12V DC motor as in figure 4.6.

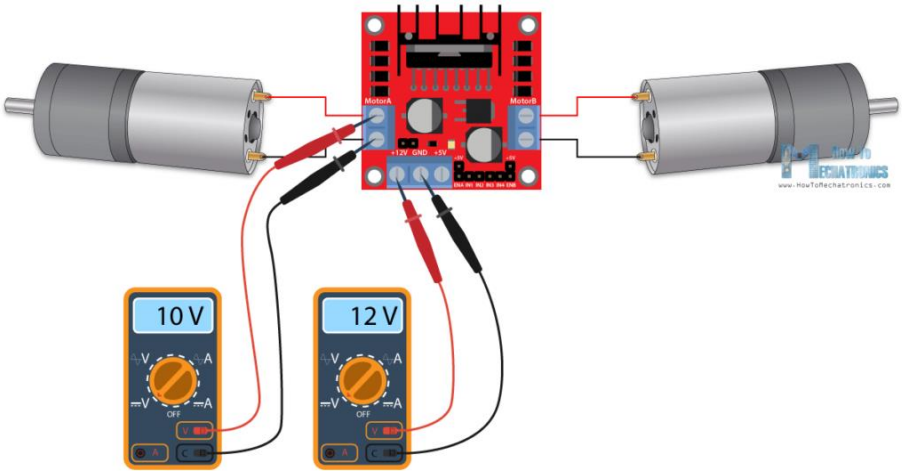

Figure 4.6: L298N voltage settings

Next are the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled as in figure 4.7 below.

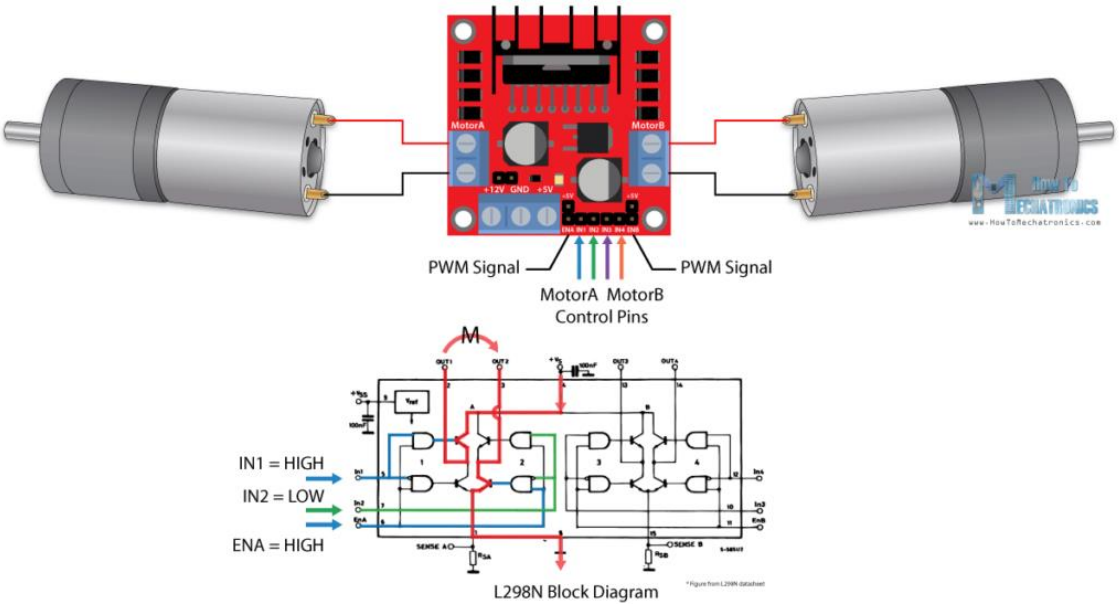

Figure 4.7: L298N logic circuit

Next, the Input 1 and Input 2 pins are used for controlling the rotation direction of the motor A, and the inputs 3 and 4 for the motor B. Using these pins we actually control the switches of the H-Bridge inside the L298N IC. If input 1 is LOW and input 2 is HIGH the motor will move forward, and vice versa, if input 1 is HIGH and input 2 is LOW the motor will move backward. In case both inputs are same, either LOW or HIGH the motor will stop. The same applies for the inputs 3 and 4 and the motor B.

4.5 Calculations

The mathematical calculation for the error value in the program will be as follows:

Error= 100*Kp*(1 + (1/Ki) + Kd)
Last error = error
Motor speed= (100 - Error) (4.1)

And the script will contain the following conditions to calculate the constants as follows:

For the Kp value

If sensor $_{(L)}$ = 0;
Kp =0
If sensor $_{(L)}$ = 1;
Kp =0.3

If sensor $_{(R)}$ = 0;
Kp= 0
If sensor$_{(R)}$ = 1;
Kp=0.3

For the Kd value

If sensor $_{(LL)}$ = 0;

Kd =0

If sensor $_{(LL)}$ = 1;

Kd =1.1

If sensor $_{(RR)}$ = 0;

Kd= 0

If sensor$_{(RR)}$ = 1;

Kd=1.1

For the Ki value

Ki=Last error*0.1

4.6 Python 3 PMW Command

The command for PWM control in Python 3 is as follows:

```
p = GPIO.PWM(channel, frequency)
```

To start PWM:

```
p.start(dc)
```

To change the frequency:

```
p.ChangeFrequency(freq)
```

To change the duty cycle:

```
p.ChangeDutyCycle(dc)
```

4.7 Sensors' Truth Table

For that algorithm, it's needed to make a truth table containing all the possibilities for the four sensors (The Logic $^{\text{Number of sensors}}$) which is $2^4 = 16$.

Table 4.1: Sensors' truth table

Sensor / NO	LL	L	R	RR
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

4.8 Implementing the All-Wheel-Drive

The All-Wheel-Drive system structure is very simple, instead of two motors spinning the two front wheels, four motors were used to spin all of the model's four wheels. With each side's motors spinning simultaneously. And Despite its easiness and simplicity, it was the most effective enhancement among all of the previous enhancements. Massively increasing the traction, and having an enormous effect on the overall stability. Without the AWD, the robot would be struggling to start moving. Before the AWD, the robot wheels used to spin in place due to lack of grip and uneven floor.

However, this is not a mechanical system, divining power upon two wheels won't divide torque, but instead in DC motors, the speed is half the original speed due to the current dividing on two motors, which means this made the model slower. For further speed results, a power source with a higher current output is required. For this project 2.1A current was fed to the motors through the motor driver.

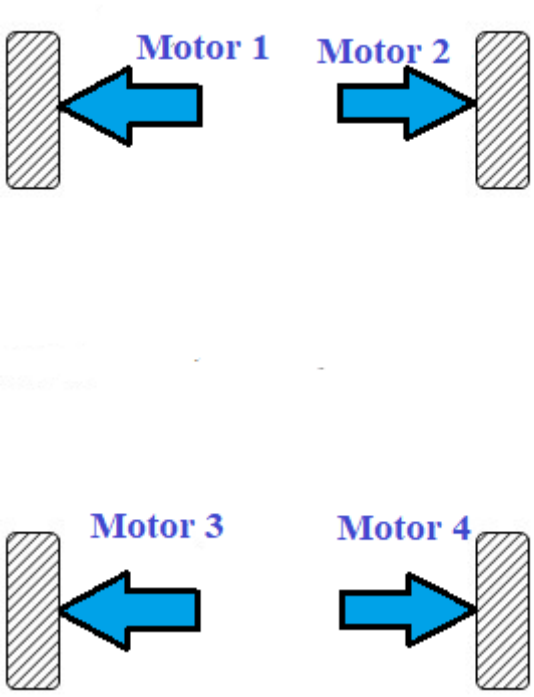

Figure 4.8**:** AWD motors placement

4.9 Model Testing

The robot is tested in the primary testing track to apply the theory, check abnormalities or shenanigans and measure the robot speed as shown in figures (4.9, 4.10, 4.11), and also to double check if the robot will go out of track.

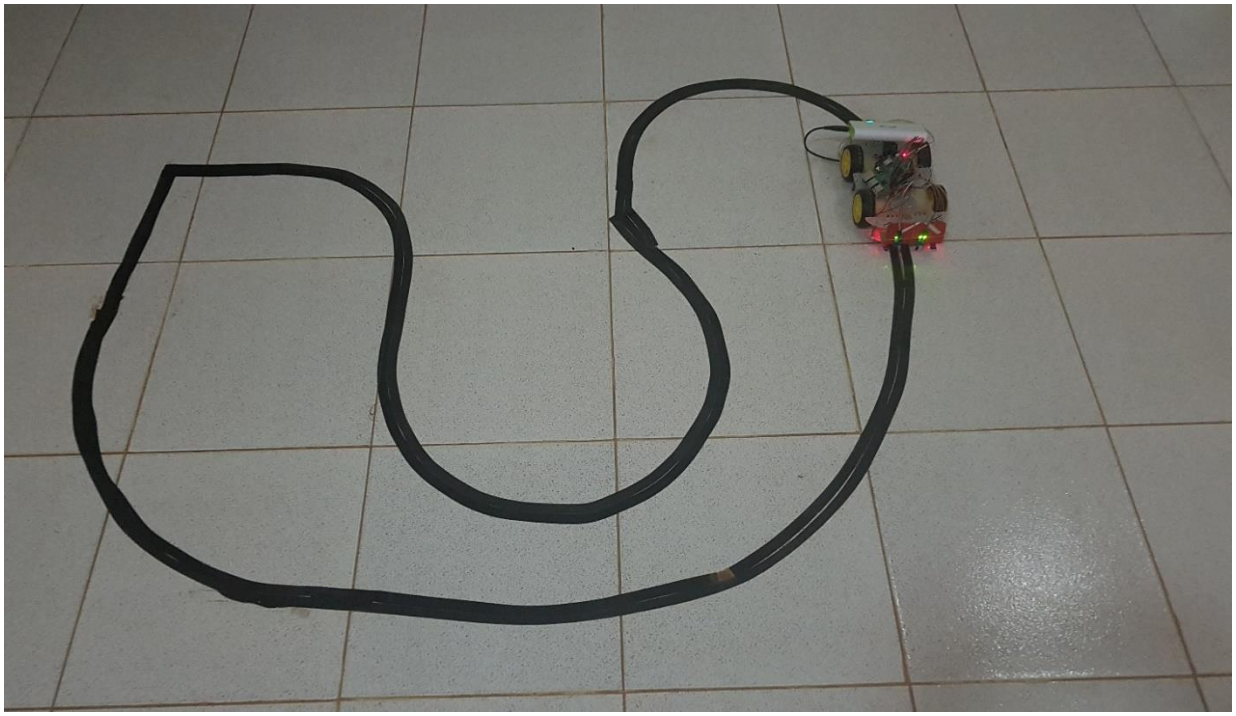

Figure 4.9: Track testing

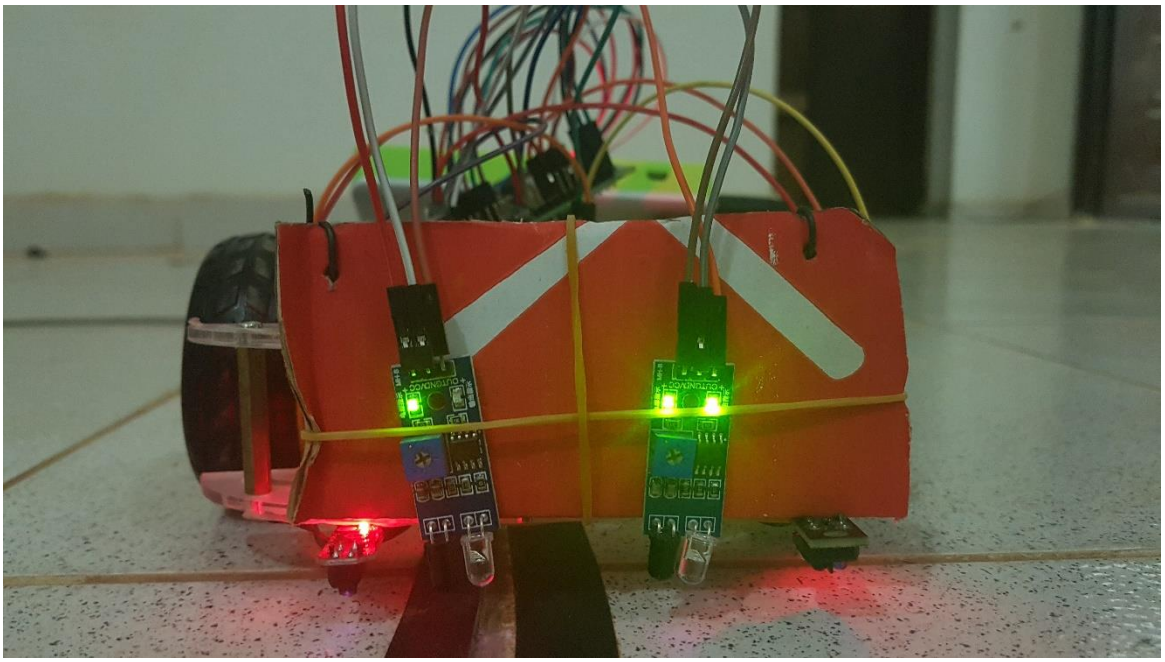

Figure 4.10: Sensors placement

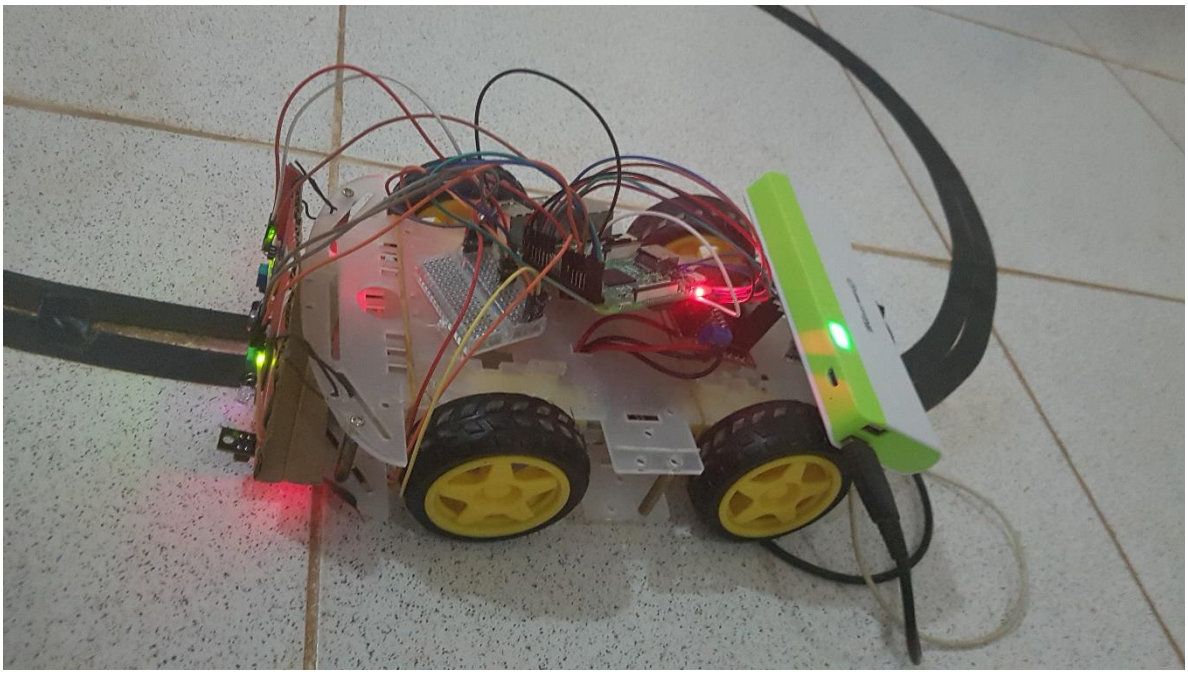

Figure 4.11: Components' side view

4.10 Results

Table 4.2 will show the sensors conditions and the following motors speed. Figure 4.12 shows the testing track after being divided into sectors and table 4.3 shows each sector's robot timing.

Table 4.2: Motors output

Sensor LL Input	**Sensor L Input**	**Sensor R Input**	**Sensor RR Input**	**Right Motor Speed**	**Left Motor Speed**
False	False	False	False	100	100
False	False	False	True	0	100
False	False	True	False	35	100
False	False	True	True	0	100
False	True	False	False	100	35
False	True	False	True	65	35
False	True	True	False	35	65
False	True	True	True	65	100
True	False	False	False	100	65
True	False	False	True	65	35
True	False	True	False	65	65
True	False	True	True	35	35
True	True	False	False	100	0
True	True	False	True	35	35
True	True	True	False	100	0
True	True	True	True	0	0

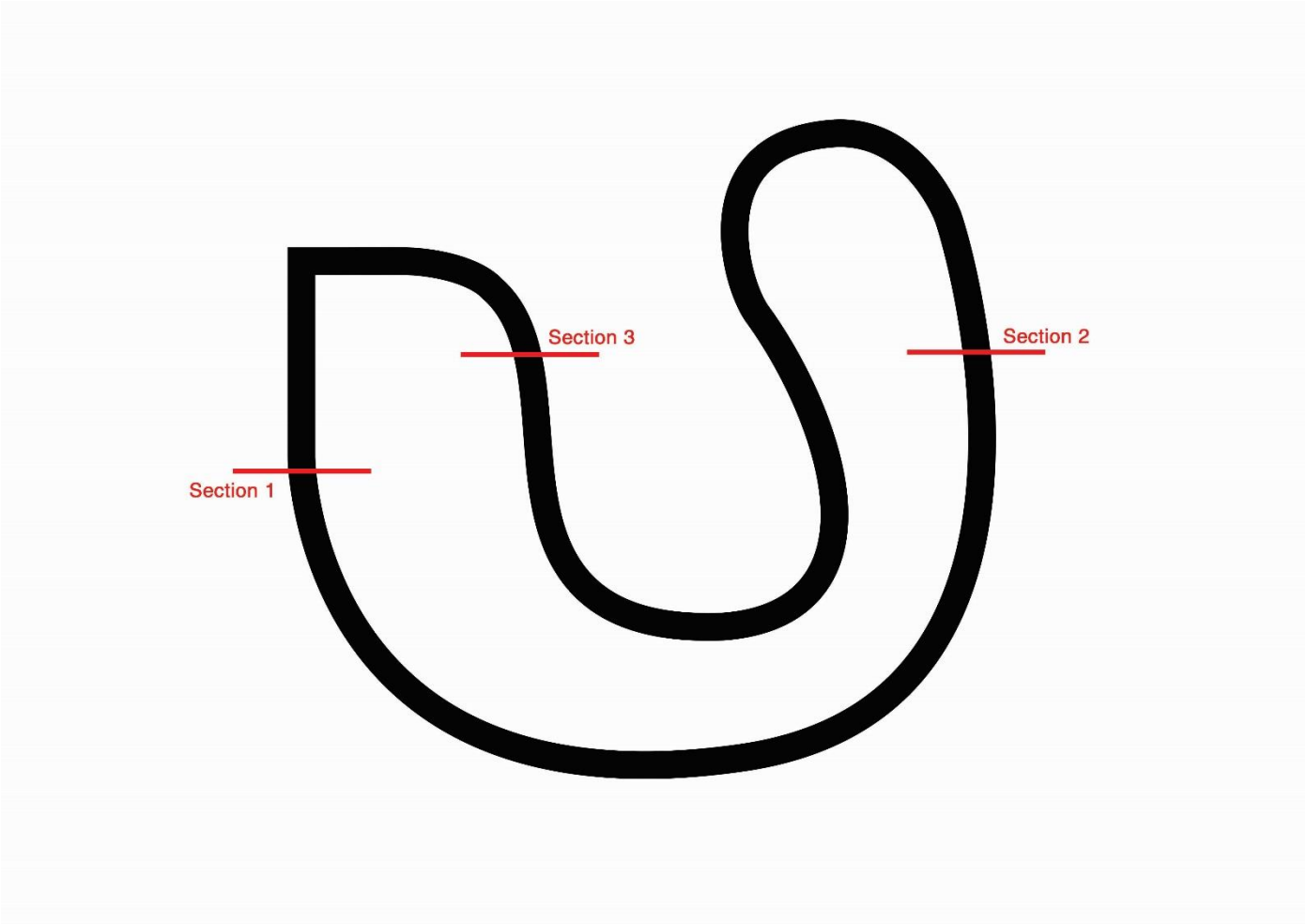

Figure 4.12: Testing track

Table 4.3: Result timings

Sections / Systems	Section 1 (seconds)	Section 2 (seconds)	Section3 (seconds)
Normal Line Follower	15.2	18.6	NA
PID Line Tracker	11.5	13.9	6.4
PID Line Tracker (with Adjusted Kp,Ki,Kd values)	8.9	12.3	5.6

CHAPTER FIVE

CONCLUSION AND RECOMNDATIONS

5.1 Conclusion

In this study, the optimized PID control scheme, which was implemented in the robot proved successful and allowed the robot to negotiate a non-trivial track with the highest speed possible. The robot moves with a maximum speed of about 20cm per second. The control algorithm gives optimal performance and its suitable implementation in a scaled up robotic system for various real world applications.

5.2 Recommendations

- The Raspberry PI gets newer versions released each period of time, it's recommended to keep up with the latest versions (while we were working on this model, a newer version "the model **B+**" got released).
- And Implementing additional peripheral hardware and deploying the model in other more complicated applications due to high processing speed of the Raspberry PI.
- Increasing the number of sensors by using an array of sensors for better variety of speed, and using a better power supply for higher current values resulting in higher speeds.

REFRENCES

[1] Omer Gumusa, Murat Topaloglub and Dogan Ozcelikbcomputer, “The Use of Computer Controlled Line Follower Robots in Public Transport”, 12th International Conference on Application of Fuzzy Systems and Soft Computing, ICAFS 2016, 29-30 August 2016, Vienna, Austria.

[2] Tarun Agarwal,” Line Follower Robots – Controlling, Working Principle and Applications”, El-Pro-Cus: Electronics-Projects-Focus, The Budding Electronic Engineers’ Knowledge Space.

[3] Vikram Balajia, M.Balajib, M.Chandrasekaranc, M.K.A.Ahamed khand, Irraivan Elamvazuthie, “Optimization of PID Control for High Speed Line Tracking Robots”, 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS 2015).

[4] Raspberry PI official webpage, "Raspberry Pi 3 specs", Retrieved on 12 October 2018.

[5] Araki, M, Hills, Richard L, "PID Control: Power From the Wind”, Cambridge University, (1996).

APPENDIX A

Sensors Testing Script

```
#import GPIO library
import RPi.GPIO as GPIO

#set GPIO numbering mode and define input pin
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12,GPIO.IN)

try:
    while True:
        if GPIO.input(12):
            print "White"
        else:
            print "Black"

finally:
    #cleanup the GPIO pins before ending
    GPIO.cleanup()
```

APPENDIX B

Normal Line Follower Script

```
import RPi.GPIO as IO
import time
IO.setwarnings(False)
IO.setmode(IO.BCM)
IO.setup(2,IO.IN) #GPIO 2 -> Left IR out
IO.setup(3,IO.IN) #GPIO 3 -> Right IR out
IO.setup(4,IO.OUT) #GPIO 4 -> Motor 1 terminal A
IO.setup(14,IO.OUT) #GPIO 14 -> Motor 1 terminal B
IO.setup(17,IO.OUT) #GPIO 17 -> Motor Left terminal A
IO.setup(18,IO.OUT) #GPIO 18 -> Motor Left terminal B

while 1:

    if(IO.input(2)==True and IO.input(3)==True): #both
while move forward
        IO.output(4,True) #1A+
        IO.output(14,False) #1B-
        IO.output(17,True) #2A+
        IO.output(18,False) #2B-
    elif(IO.input(2)==False and IO.input(3)==True): #turn
right
        IO.output(4,True) #1A+
        IO.output(14,True) #1B-
```

```
        IO.output(17,True) #2A+
        IO.output(18,False) #2B-
    elif(IO.input(2)==True and IO.input(3)==False): #turn
left
        IO.output(4,True) #1A+
        IO.output(14,False) #1B-
        IO.output(17,True) #2A+
        IO.output(18,True) #2B-

else:  #stay still
        IO.output(4,True) #1A+
        IO.output(14,True) #1B-
        IO.output(17,True) #2A+
        IO.output(18,True) #2B-
```

APPENDIX C

PID Line Tracker Script (Stable)

```
import RPi.GPIO as IO
import time
from time import sleep
IO.setwarnings(False)
IO.setmode(IO.BCM)

IO.setup(17,IO.IN) #GPIO 17 (pin11) -> Left IR in
IO.setup(27,IO.IN) #GPIO 27 (pin13) -> Right IR in
IO.setup(23,IO.IN) #GPIO 23 (pin16) -> Left Left IR in
IO.setup(24,IO.IN) #GPIO 24 (pin18) -> Right Right IR in

IO.setup(5,IO.OUT) #GPIO 05 (pin 29)-> Motor 1 terminal A
IO.setup(6,IO.OUT) #GPIO 06 (pin 31)-> Motor 1 terminal B
IO.setup(13,IO.OUT) #GPIO 13 (pin 33)->Motor 2 terminal A
IO.setup(19,IO.OUT) #GPIO 19 (pin 35)->Motor 2 terminal B

IO.setup(20,IO.OUT) #GPIO 20 (pin 38)-> Motor PWM R
IO.setup(21,IO.OUT) #GPIO 21 (pin 40)-> Motor PWM L

IO.output(5,True) #1A+
IO.output(6,False) #1B-
IO.output(13,True) #2A+
IO.output(19,False) #2B-
```

```
speedR=IO.PWM(20, 100)
speedR.start(100)
speedL=IO.PWM(21, 100)
speedL.start(100)


while 1:


     if(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==False and IO.input(24)==False): #straight
forward

        speedR.ChangeDutyCycle(100)
        speedL.ChangeDutyCycle(100)


    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==False and IO.input(24)==True): #low right

        speedR.ChangeDutyCycle(0)
        speedL.ChangeDutyCycle(100)


    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==True and IO.input(24)==False): #Hard right

        speedR.ChangeDutyCycle(35)
        speedL.ChangeDutyCycle(100)
```

```
    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==True and IO.input(24)==True): #low Left
        speedR.ChangeDutyCycle(0)
        speedL.ChangeDutyCycle(100)

    elif(IO.input(23)==True and IO.input(17)==True and
IO.input(27)==False and IO.input(24)==False): #Hard Left
        speedR.ChangeDutyCycle(100)
        speedL.ChangeDutyCycle(35)

    elif(IO.input(23)==True and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==False): #low right
        speedR.ChangeDutyCycle(35)
        speedL.ChangeDutyCycle(35)

    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==True and IO.input(24)==True): #Hard right
        speedR.ChangeDutyCycle(0)
        speedL.ChangeDutyCycle(100)

    elif(IO.input(23)==True and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==True): #medium Left
        speedR.ChangeDutyCycle(0)
        speedL.ChangeDutyCycle(100)
```

```
    elif(IO.input(23)==False and IO.input(17)==False and
IO.input(27)==False and IO.input(24)==False): #medium
right

        speedR.ChangeDutyCycle(100)
        speedL.ChangeDutyCycle(0)


    elif(IO.input(23)==False and IO.input(17)==True and
IO.input(27)==False and IO.input(24)==False): #low Left

        speedR.ChangeDutyCycle(100)
        speedL.ChangeDutyCycle(0)




    elif(IO.input(23)==False and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==False): #Hard Left

        speedR.ChangeDutyCycle(100)
        speedL.ChangeDutyCycle(0)


    elif(IO.input(23)==False and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==True):
#abnormalities

        speedR.ChangeDutyCycle(0)
        speedL.ChangeDutyCycle(0)


    else:#shenanigans
```

```
            speedR.ChangeDutyCycle(35)
            speedL.ChangeDutyCycle(35)


    IO.cleanup()
```

APPENDIX D

PID Line Tracker Script (Embedded)

```
import RPi.GPIO as IO
import time
from time import sleep
IO.setwarnings(False)
IO.setmode(IO.BCM)

IO.setup(17,IO.IN) #GPIO 17 (pin11) -> Left IR in
IO.setup(27,IO.IN) #GPIO 27 (pin13) -> Right IR in
IO.setup(23,IO.IN) #GPIO 23 (pin16) -> Left Left IR in
IO.setup(24,IO.IN) #GPIO 24 (pin18) -> Right Right IR in

IO.setup(5,IO.OUT) #GPIO 05 (pin 29)-> Motor 1 terminal A
IO.setup(6,IO.OUT) #GPIO 06 (pin 31)-> Motor 1 terminal B
IO.setup(13,IO.OUT) #GPIO 13 (pin 33)->Motor 2 terminal A
IO.setup(19,IO.OUT) #GPIO 19 (pin 35)->Motor 2 terminal B

IO.setup(20,IO.OUT) #GPIO 20 (pin 38)-> Motor PWM R
IO.setup(21,IO.OUT) #GPIO 21 (pin 40)-> Motor PWM L

IO.output(5,True) #1A+
IO.output(6,False) #1B-
IO.output(13,True) #2A+
IO.output(19,False) #2B-
```

```
speedR=IO.PWM(20, 100)
speedR.start(100)
speedL=IO.PWM(21, 100)
speedL.start(100)


while 1:


     if(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==False and IO.input(24)==False): #straight
forward
```

Kp=0; Kd =0; Ki=last_error*0.1

error=100*Kp*(1+(1/Ki)+Kd)

last_error = error

motorspeed=(100-error)

```
speedR.ChangeDutyCycle(motorspeed)
speedL.ChangeDutyCycle(motorspeed)
```

```
    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==False and IO.input(24)==True): #low right
```

Kp=0; Kd =0; Ki=last_error*0.1

error=100*Kp*(1+(1/Ki)+Kd)

last_error = error

motorspeed=(100-error)

```
speedR.ChangeDutyCycle(motorspeed)
speedL.ChangeDutyCycle(motorspeed)
```

```
    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==True and IO.input(24)==False): #Hard right
```

Kp=0; Kd =1.1; Ki=last_error*0.1

error=100*Kp*(1+(1/Ki)+Kd)

last_error = error

motorspeed=(100-error)

```
speedR.ChangeDutyCycle(motorspeed)
speedL.ChangeDutyCycle(motorspeed)
```

```
    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==True and IO.input(24)==True): #low Left
```

Kp=0.3; Kd =0; Ki=last_error*0.1

error=100*Kp*(1+(1/Ki)+Kd)

last_error = error

motorspeed=(100-error)

```
speedR.ChangeDutyCycle(motorspeed)
speedL.ChangeDutyCycle(motorspeed)
```

```
    elif(IO.input(23)==True and IO.input(17)==True and
IO.input(27)==False and IO.input(24)==False): #Hard Left
```

Kp=0.3; Kd =0; Ki=last_error*0.1

error=100*Kp*(1+(1/Ki)+Kd)

last_error = error

motorspeed=(100-error)

```
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)

    elif(IO.input(23)==True and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==False): #low right

        Kp=0.3; Kd =0; Ki=last_error*0.1
        error=100*Kp*(1+(1/Ki)+Kd)
        last_error = error
        motorspeed=(100-error)
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)

    elif(IO.input(23)==True and IO.input(17)==False and
IO.input(27)==True and IO.input(24)==True): #Hard right

        Kp=0.3; Kd =1.1; Ki=last_error*0.1
        error=100*Kp*(1+(1/Ki)+Kd)
        last_error = error
        motorspeed=(100-error)
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)

    elif(IO.input(23)==True and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==True): #medium Left
```

```
        Kp=0.3; Kd =1.1; Ki=last_error*0.1
        error=100*Kp*(1+(1/Ki)+Kd)
        last_error = error
        motorspeed=(100-error)
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)

    elif(IO.input(23)==False and IO.input(17)==False and
IO.input(27)==False and IO.input(24)==False): #medium
right

        Kp=0.3; Kd =1.1; Ki=last_error*0.1
        error=100*Kp*(1+(1/Ki)+Kd)
        last_error = error
        motorspeed=(100-error)
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)

    elif(IO.input(23)==False and IO.input(17)==True and
IO.input(27)==False and IO.input(24)==False): #low Left

        Kp=0; Kd =1.1; Ki=last_error*0.1
        error=100*Kp*(1+(1/Ki)+Kd)
        last_error = error
        motorspeed=(100-error)
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)
```

```
    elif(IO.input(23)==False and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==False): #Hard Left
        Kp=0; Kd =1.1; Ki=last_error*0.1
        error=100*Kp*(1+(1/Ki)+Kd)
        last_error = error
        motorspeed=(100-error)
        speedR.ChangeDutyCycle(motorspeed)
        speedL.ChangeDutyCycle(motorspeed)

    elif(IO.input(23)==False and IO.input(17)==True and
IO.input(27)==True and IO.input(24)==True):
#abnormalities
        speedR.ChangeDutyCycle(0)
        speedL.ChangeDutyCycle(0)

    else:#shenanigans
        speedR.ChangeDutyCycle(35)
        speedL.ChangeDutyCycle(35)

IO.cleanup()
```